Содержание

Предисловие 3
Алгоритм 151б. Порядковый номер сочетания в лексикографически упорядоченном списке сочетаний [M1, G6] 6
Алгоритм 152б. Генератор перестановок нулей и единиц [G6] 7
Алгоритм 153б. Целочисленная задача линейного программирования [H] . . 8
Алгоритм 154б. Генератор лексикографически упорядоченной последовательности сочетаний [G6] 12
Алгоритм 155б. Генератор сочетаний с повторениями [G6] 13
Алгоритм 156б. Сумма знакопеременного ряда произведений из элементов сочетаний [G6] 15
Алгоритм 157б. Аппроксимация рядами Фурье [E2] 15
Свидетельство к алгоритму 158б [C1] 19
Алгоритм 159б. Вычисление определителя (рекурсивная процедура) [F3] . . 19
Алгоритм 160б. Число сочетаний [S03] 20
Алгоритм 161б. Вектор чисел всевозможных сочетаний из m элементов [G6, S03] 21
Алгоритм 162б. Вычерчивание графиков [J6] 21
Алгоритм 163б. Модифицированная функция Ханкеля [S17] 25
Алгоритм 164б. Приближение поверхности ортогональными полиномами по методу наименьших квадратов [E2] 26
Алгоритм 165б. Полные эллиптические интегралы [S21] 32
Алгоритм 166б. Обращение матрицы методом Монте-Карло [F1] 34
Алгоритм 167б. Разделенные разности с повторяющимися точками [E1] . . . 37
Алгоритм 168б. Интерполяция по Ньютону с разделенными разностями в обратном направлении [E1] 39
Алгоритм 169б. Интерполяция по Ньютону с разделенными разностями в прямом направлении [E1] 40
Алгоритм 170б. Определитель с полиномиальными элементами [F3] 44
Свидетельство к алгоритму 171б [Z] 48
Алгоритм 172б. Интерполяция табличной функции нескольких переменных (рекурсивная процедура) [E1] 48
Алгоритм 173б. Присваивание значений массивам разной размерности (рекурсивная процедура) [K2] 54

Свидетельство к алгоритму 174б [C2] 57
Алгоритм 175б. Сортировка последовательностей [M1] 57
Алгоритм 176б. Аппроксимация последовательности точек линейной комбинацией любых заданных функций [E2] 58
Свидетельство к алгоритму 177б [E2, F4] 60
Алгоритм 178б. Минимизация функции нескольких переменных методом прямого поиска (методом конфигураций) [E4] 60
Алгоритм 179б. Отношение неполных бета-функций [S14] 66
Свидетельство к алгоритмам 180б и 181б [S15] 71
Алгоритм 182б. Вычисление интеграла по Симпсону с заданной мерой погрешности [D1] 71
Алгоритм 183б. Преобразование ленточной симметричной матрицы в трехдиагональную [F2] 73
Алгоритм 184б. Табулирование закона распределения Эрланга [S22] 75
Алгоритм 185б. Табулирование функции нормального распределения [S15] . . 78
Алгоритм 186б. Комплексная арифметика [A2] 80
Алгоритм 187б. Разности и производные (рекурсивные процедуры) [E1] . . 82
Алгоритм 188б. Сглаживание по трем точкам [E3] 83
Алгоритм 189б. Сглаживание по пяти точкам [E3] 84
Алгоритм 190б. Комплексная степень комплексного числа [B4] 85
Алгоритм 191б. Гипергеометрическая функция [S22] 86
Алгоритм 192б. Конфлюентная гипергеометрическая функция [S22] 90
Алгоритм 193б. Обращение степенного ряда [C1] 92
Алгоритм 194б. Корни решения системы дифференциальных уравнений [D2]. . 94
Алгоритм 195б. Система линейных уравнений с ленточной матрицей [F4] . . 97
Алгоритм 196б. Метод Мюллера нахождения корней произвольной функции [C5] 99
Алгоритм 197б. Деление матрицы на матрицу [F1] 111
Свидетельство к алгоритму 198б [D1] 113
Алгоритм 199б. Переход от календарной даты к порядковому номеру дня и обратно [Z] 113
Алгоритм 200б. Генератор нормально распределенных случайных чисел [G5] . 115
Приложение 1. Алгоритмы шахматного программирования 118
Приложение 2. Подтверждения и замечания к алгоритмам, опубликованным в предыдущих выпусках 161
Список литературы, которой пользовались составители выпуска 179
Список литературы, на которую ссылаются авторы исходных алгоритмов . . 182

Предисловие

Данный выпуск является продолжением серии, начатой выпуском «Библиотека алгоритмов 1б—50б» [47], и результатом дальнейшего совершенствования алгоритмов выпуска «Алгоритмы (151—200)» [50]. Последний содержал алгоритмы 151а—200а, являющиеся, в свою очередь, результатом переработки соответствующих алгоритмов журнала «Communications of the ACM» [21]. К каждому алгоритму данного выпуска прилагаются соответствующие «Подтверждения» и «Замечания» из вышеуказанного журнала и от советских пользователей алгоритмами, а также «Свидетельства», написанные редактором выпуска. Алгоритмы публикуются здесь на эталонном алгоритмическом языке АЛГОЛ-60 [1], описанном во многих учебниках [22—25, 86—88, 94]. Там, где это возможно без заметного ухудшения алгоритмов, они предварительно переводились редактором выпуска на сокращенный АЛГОЛ-60 [2] * с некоторым его расширением в сторону полного языка АЛГОЛ-60 [допускались: 1) различение идентификаторов по всем содержащимся в них символам, 2) возведение целых чисел в любую целую степень, 3) операция ÷ и 4) условная форма именующего выражения]. В частности, все алгоритмы здесь записаны с использованием только строчных букв латинского алфавита **. Прописные буквы используются только в приложении 1 (в алгоритме 171б) для идентификаторов, являющихся русскими словами.

В свидетельствах указывается оригинал переработанного алгоритма, перечисляются виды работ, произведенных над алгоритмом, внесенные в него изменения и приводятся результаты контрольного решения по данному алгоритму. Работы, которые проводились над всеми алгоритмами, для краткости называются здесь «ординарной переработкой». К ней относятся перевод на русский язык комментариев, подтверждений и замечаний, придание алгоритму наглядной, удобочитаемой формы путем применения однотипной ступенчатой записи по правилам, опубликованным в статье [38], использование идентификаторов интернационального характера, а также перевод алгоритмов на сокращенный АЛГОЛ. Все другие модификации алгоритмов (например, внесение в них исправлений, сокращение их записи, оптимизация и т. д.), а также отличия используемых языковых средств от сокращенного АЛГОЛа [2] и те случаи, когда алгоритмы составлялись заново, оговариваются в свидетельствах особо.

В качестве приложений к выпускам данной серии, имеющим нечетные номера, даются тематические указатели алгоритмов, появившихся в периодической печати к моменту составления выпуска. Алгоритмы

---

* Этот вариант языка АЛГОЛ-60 почти не отличается от получившего в Советском Союзе распространения языка АЛГАМС [58, 92, 93].

** Из технологических соображений латинские буквы, набранные в тексте курсивом, в описаниях процедур набраны прямым шрифтом. Например, в алгоритме 151б *locate* и locate означают один и тот же идентификатор.

в таких указателях группируются в соответствии с классификацией, принятой в журнале «Communications of the ACM». Названия алгоритмов в выпусках сопровождаются индексами, соответствующими этой классификации. Например, в заголовке «Алгоритм 157б. Аппроксимация рядами Фурье [E2]» индекс [E2] указывает на то, что алгоритм 157б относится к классу E2 («Аппроксимирующие кривые и поверхности»). Расшифровка таких индексов имеется в указателях.

В выпусках серии «Библиотека алгоритмов» номера алгоритмов снабжаются буквой «б» для отличия их как от исходных, так и от алгоритмов предыдущей серии [4, 5, 20, 50, 51] (снабжавшихся буквами «а») *. В ссылках на источники аббревиатура «CACM» означает журнал «Communications of the ACM» [21]. В переводах «Подтверждений» и «Замечаний» перечень поправок к алгоритмам обычно опускается, поскольку эти поправки, как правило, уже внесены в переработанный алгоритм. В соответствующих местах ставятся многоточия и делаются сноски. Отладка алгоритмов проводилась с использованием транслятора **ТА-1М [26, 88, 90, 111] на машинах М-220 [59] (30 000 опер./с) и транслятора БЭСМ-АЛГОЛ [62, 63, 88, 90] на машине БЭСМ-6 [60] (1 000 000 опер./с).

В серии «Библиотека алгоритмов» публикуются только отлаженные алгоритмы. Однако, учитывая известный каждому программисту факт, что никакая отладка и даже многолетнее использование не гарантируют абсолютную безошибочность программ, а также то, что пределов совершенствования алгоритмов практически не существует, авторы выпуска обращаются ко всем читателям и пользователям алгоритмами с просьбой присылать свои замечания и подтверждения в адрес издательства (для Агеева М. И.). Публикация таких замечаний будет продолжена, как это делалось в предыдущих выпусках ***.

Работа по подготовке к изданию алгоритмов предыдущей серии [4, 5, 20, 50, 51] была начата в соответствии с объявлением, регулярно (начиная с мая 1964 г.) публиковавшимся в разделе «Алгоритмы» журнала «CACM», где, в частности, говорилось: «Переработанные варианты ранее опубликованных алгоритмов будут считаться новыми публикациями алгоритмов, и эти варианты не должны быть частью «Подтверждений» или «Замечаний»... Репродукция алгоритмов данного раздела разрешается совершенно безвозмездно. Если репродукция делалась с целью публикации, то необходима ссылка на автора алгоритма и на выпуск журнала «CACM», в котором опубликован алгоритм». Первый выпуск «Алгоритмы (1—50)» сразу же после публикации был послан на отзыв президенту ACM проф. Г. Форсайту (бывше-

---

* Если какой-либо алгоритм данного выпуска сопровождается свидетельством только к соответствующему алгоритму с буквой «а», а свидетельство к алгоритму с буквой «б» отсутствует, это означает, что он является стереотипным переизданием алгоритма с буквой «а».

** В дальнейшем здесь наряду со словом «транслятор» будут употребляться слова «транслирующая система» или просто «система».

*** В недавно опубликованном на русском языке «Справочнике алгоритмов на АЛГОЛе» Уилкинсона и Райнша [85, с. 12] эта же мысль выражена следующими словами: «Публикация материала подобного содержания связана с теми особенностями, что выбранные рабочие алгоритмы должны быть полезными в течение длительного времени. Авторы убеждены, что создание работоспособных алгоритмов — процесс непрерывный, и они будут благодарны всем, кто укажет на ошибки и трудности, встретившиеся при практической работе с предложенными алгоритмами».

му в то время редактором раздела «Алгоритмы» журнала «CACM») и получил полное его одобрение.

Появление в печати предыдущей серии [4, 5, 20, 50, 51] было встречено горячим одобрением как подавляющего большинства читателей и пользователей, так и самих авторов и издателей исходных алгоритмов (см., например, «Замечания» Г. Форсайта, Дж. Вараха и Г. Цилке в приложении 1 к выпуску «Библиотека алгоритмов 1б—50б» [47]). Тем не менее, как и в предыдущих выпусках, здесь подчеркивается, что авторские права* коллектива сотрудников, выполнявших работу составителей, переводчиков, переработчиков и отладчиков алгоритмов и собирательно именуемых здесь составителями выпуска**, распространяются только на выпуск в целом и на те специально оговоренные в свидетельствах случаи, когда алгоритмы составлялись заново. Сделанные в процессе переработки изменения исходных алгоритмов, как правило, лишь улучшали их машинную реализацию (исправления, сокращения, оптимизация и т. д.) и удобства пользования ими, но не затрагивали сущности алгоритмов, их численных методов. Таким образом, алгоритмы, номера которых снабжены буквами «б» или «а», являются лишь вариантами исходных алгоритмов, публикуемыми на правах обычных подтверждений. Поэтому читателям и пользователям в их ссылках на источники нужно, как правило, указывать авторов первичных публикаций алгоритмов и номера журнала «CACM», из которых взяты эти алгоритмы, не зависимо от того, будут ли при этом читатели ссылаться на соответствующий выпуск данной серии или нет.

Контрольные решения и отладку по большинству алгоритмов данного выпуска проводил Ю. И. Марков, по алгоритмам 152, 154, 155, 160, 161, 167—169, 184 и 185 — М. Г. Грюнберг, по алгоритмам 159, 163, 191 и 192 — Г. М. Крапивина, по алгоритмам 151, 175 и 193 — В. М. Агеев, по алгоритму 171 — М. И. Агеев. Рукопись к изданию подготовил В. П. Алик. В проверке рукописи принимал участие В. М. Агеев. Основная переработка алгоритмов, написание текста свидетельств, компоновка выпуска и его общая редакция выполнялись М. И. Агеевым.

Составители выпуска выражают свою глубокую благодарность всем читателям, которые прислали свои замечания и подтверждения к ранее опубликованным алгоритмам, тем, кто дал в своих замечаниях высокую оценку проделанной работы, и тем, кто ссылками в своих опубликованных работах (см., например, работы [46, 85, 87, 88, 90, 91, 94, 104, 106]) подтвердил значение данной «Библиотеки алгоритмов» как нового справочного пособия.

---

* Вопрос об авторстве на алгоритмы широкого назначения, публикуемые в справочных пособиях, подобных данному, наиболее точно освещен в вышеупомянутом «Справочнике алгоритмов на языке АЛГОЛ» Уилкинсона и Райнша [85, с. 12], где сказано следующее: «Само содержание книги указывает на то, что это плод совместных усилий большого числа исследователей. Во многих случаях исходная основа алгоритма претерпевала изменения в течение длительного периода времени и многие математики принимали участие в улучшении первоначального варианта. Мы благодарны всем, кто принимал непосредственное участие в этом издании».

** Изменение в данном выпуске названия «авторы выпуска» (как это было принято в трех предыдущих выпусках [47—49]) на «составители выпуска» объясняется главным образом тем, что удельный вес (как по объему, так и по значению) проделанной ими работы заметно изменился за истекшие годы в пользу первичных публикаций алгоритмов в журнале «CACM».

АЛГОРИТМ 151б

Порядковый номер сочетания в лексикографически упорядоченном списке сочетаний [M1, G6]

Процедура *locate* (*locate* — определять местонахождение) определяет порядковый номер r данного вектора $(v_1, v_2, \ldots, v_c)$ в списке векторов без просмотра самого списка. В список входят всевозможные сочетания из n последовательных целых чисел, взятых по c. Наименьшее из этих чисел обозначается через *min*. Компоненты каждого вектора (сочетания) списка записываются в порядке возрастания слева направо, например 3 7 8, а векторы в списке размещаются в лексикографическом порядке, т. е. если эти векторы рассматривать как числа из c цифр, то они будут расположены в порядке возрастания. Например, при $min=1$, $c=3$ и $n=6$ векторы располагаются в следующем порядке: 123 124 125 126 134 135 ... 456. Для вектора $v=(1,3,4)$ процедура найдет в этом списке порядковый номер 5.

Процедура *locate* использует процедуру-функцию *comb*(w,p), определяющую число сочетаний из w по p, в качестве которой можно взять алгоритм 160б. Границы индексов вектора v нужно задавать от 0 до c, причем начальное значение $v(0)$ несущественно. После выполнения процедуры значение вектора v окажется измененным.

```
procedure locate(min,n,c,v,comb)result:(r);
      value min,n,c; integer min,n,c,r; integer array v;
      integer procedure comb;
begin integer i,max,p,w;
      r:=1; v[0]:=min-1;
      max:=min+n-2; c:=c-1;
      for i:=0 step 1 until c do
        begin p:=c-i;
test:     if v[i+1]-v[i]>1 then
            begin w:=max-v[i];
              r:=r+comb(w, p); v[i]:=v[i]+1;
              go to test
            end
        end i
end locate;
```

Свидетельство к алгоритму 151б

Алгоритм 151б получен из алгоритма 151а в результате следующих тождественных преобразований, имеющих целью ускорение его работы: во-первых, в теле процедуры *locate* добавлен оператор $c:=c-1$; и внесены соответствующие изменения в следующий затем оператор цикла; во-вторых, оператор $max:=min+n-1$ заменен оператором $max:=$

$min+n-2$, вследствие чего оказалось возможным вместо оператора $w:=max-v[i]-1$ писать $w:=max-v[i]$.

Алгоритм 151б был транслирован в системе БЭСМ-АЛГОЛ и для примера, приведенного в нижеследующем «Свидетельстве к алгоритму 151а», снова дал те же результаты.

Свидетельство к алгоритму 151а

Алгоритм 151а получен в результате исправления и ординарной переработки алгоритма 151 (Walter H. F. «CACM», 1963, № 2). В алгоритме 151 для числа элементов в сочетаниях было замечено несоответствие между обозначением, принятым в процедуре (идентификатор *c*), и обозначением, принятым в примечании (идентификатор *d*). Кроме того, в примечании к алгоритму 151 не было дано никаких указаний, что понимается под процедурой ***combinatorial*** (**в алгоритме 151а ей соответствует процедура** *comb*).

Алгоритм 151а был транслирован для $n=6$, $c=3$, $v=(3,5,6)$, и получены значения $r=21, 19, 14, 5$, соответствующие $min=0, 1, 2, 3$.

Этот результат легко проверяется, если выписать векторы списка в лексикографическом порядке. Например, для $min=3$ первые пять векторов списка имеют вид

345, 346, 347, 348, 356.

Для $min=2$ имеем следующее начало списка (до вектора 356):

234					
235	245			345	
236	246	256		346	356
237	247	257	267	347	... ...

АЛГОРИТМ 152б

Генератор перестановок нулей и единиц [G6]

Процедура *nexcom* (*next* — следующая, *composition* — перестановка) преобразует данный вектор ***c***, состоящий из *n* элементов, каждый из которых либо 0, либо 1, в другой вектор, содержащий то же количество нулей и единиц, но в другом порядке. Если начать с вектора $c[1:n]$, содержащего единицы в первых *r* компонентах и нули — во всех остальных, то повторными вызовами процедуры *nexcom* можно получить все перестановки вектора ***c***. Процесс нахождения всех перестановок заканчивается, как только будет получен вектор ***c***, имеющий нули во всех первых $n-r$ компонентах и единицы в остальных. В последнем случае выход из процедуры осуществляется через метку-параметр *complete*. Если был задан нулевой вектор ***c***, то выход из процедуры происходит через метку-параметр *null*.

```
procedure nexcom(n,complete,null) dataresult:(c);
    value n; integer n; array c; label complete,null;
begin integer p,m,k;
    for k:=1 step 1 until n do
      if c[k]=1 then go to aa;
```

```
        go to null;
aa:     p:=0;
        for m:=k+1 step 1 until n do
          if c[m]=1 then p:=p+1 else go to bb;
bb:     if p+k=n then go to complete;
        c[k+p+1]:=1;
        for m:=k+p step -1 until k do c[m]:=0;
        for m:=1 step 1 until p do c[m]:=1
end nexcom;
```

Свидетельство к алгоритму 152а

Алгоритм 152а получен в результате исправления и ординарной переработки алгоритма 152 (Hopley J. «CACM», 1963, № 7). В алгоритме 152 была исправлена путаница с идентификаторами N и n.

С помощью системы ТА-1М алгоритм 152а при $n=4$ и $c=(1100)$ выдал перестановки

1100 1010 0110 1001 0101 0011

Для $n=5$ и $c=(11100)$ получены перестановки

11100 10110 11001 01101 01011
11010 01110 10101 10011 00111

Замечание рецензента. Задачу получения всех перестановок описанного выше типа можно решить с помощью алгоритма 155а, который работает несколько медленнее, так как предназначен для более сложных задач.

АЛГОРИТМ 153б

Целочисленная задача линейного программирования [H]

Процедура *gomory1* [*Gomory* — Гомори (автор метода)] предназначена для определения целого решения $x[1], \ldots, x[n-1]$ задачи линейного программирования с целыми коэффициентами. Иными словами, задача состоит в нахождении целых чисел $x[1], \ldots, x[n-1]$, минимизирующих целевую функцию

$$f=a[0,1]\times x[1]+\ldots+a[0,n-1]\times x[n-1]$$

при условии, что $x[1]\geqslant 0, \ldots; x[n-1]\geqslant 0$ и $a[i,1]\times x[1]+\ldots+a[i,n-1]\times x[n-1]\leqslant a[i,n]$ для $i=1, 2, \ldots, m-n+1$ $(2\leqslant n\leqslant m)$.

Матрица a, используемая процедурой, имеет размерность $a[0:m, 1:n]$. Входные значения компонентов этой матрицы частично задаются самой задачей (см. выше). Остальным компонентам значения должны быть предварительно присвоены следующим образом:

$$a[0,n]:=0 \text{ и } a[i,j]:=\textbf{if } i=j+m-n+1 \textbf{ then} -1 \textbf{ else } 0$$

для $i=m-n+2, \ldots, m$ и $j=1, \ldots, n$. Столбцы матрицы a, за исключением последнего, должны быть лексикографически положительными *.

Алгоритм заканчивает работу, когда все значения в последнем столбце, кроме самого верхнего, не отрицательны. Тогда f является

* Т. е. первые ненулевые элементы этих столбцов должны быть положительными. (*Прим. ред.*)

значением целевой функции, и оптимальное решение записано в $x[1], \ldots, x[n-1]$.

Выход к метке *signal* осуществляется тогда, когда встречается строка, в которой все элементы, кроме, возможно, последнего, не отрицательны.

```
procedure gomory1 (a,m,n,signal)result: (x,f);
      value m,n; integer m,n,f; label signal;
      integer array a,x;
begin integer i,j,k,q,r,c,t,s,lambda1,lambda2;
      integer procedure euclid(u,v);
        value u,v; integer u,v;
        begin integer w;
          w:=entier(u/v);
n8:       if w×v>u then
            begin w:=w—1; go to n8 end;
n9:       if (w+1)×v≤u then
            begin w:=w+1; go to n9 end;
          euclid:=w
        end euclid;
n1:   for i:=1 step 1 until m do
        if a[i,n]<0 then begin r:=i; go to n2 end i;
      go to fin;
n2:   for k:=1 step 1 until n—1 do
        if a[r,k]<0 then go to n4;
      go to signal;
n4:   q:=k;
      for j:=k+1 step 1 until n—1 do
        if a[r,j]<0 then
          begin i:=0;
n3:         if a[i,j]<a[i,q] then q:=j else
              if a[i,j]=a[i, q] then
                begin i:=i+1; go to n3 end
          end j;
      s:=0;
n5:   if a[s,q]=0 then
        begin s:=s+1; go to n5 end;
      lambda1:=—a[r,q]; lambda2:=1;
      for j:=1 step 1 until q—1, q+1 step 1 until n—1 do
        if a[r,j]<0 then
          begin
            for i:=0 step 1 until s—1 do
              if a[i,j]≠0 then go to n7;
            t:=euclid(a[s,j],a[s,q]);
            if (t×a[s,q]=a[s,j])∧(t>1) then
              begin i:=s;
n6:             i:=i+1;
                if t×a[i,q]=a[i,j] then go to n6;
                if t×a[i,q]>a[i,j] then t:=t—1
              end;
            if —a[r,j]×lambda2>t×lambda1 then
              begin lambda1:=—a[r,j]; lambda2:=t end;
n7:       end j;
```

```
      for j:=1 step 1 until q-1, q+1 step 1 until n do
        begin c:=euclid(a[r,j]×lambda2,lambda1);
          if c≠0 then
            for i:=0 step 1 until m do
              a[i,j]:=a[i,j]+c×a[i,q]
        end;
      go to n1;
fin:  f:=-a[0,n];
      for i:=1 step 1 until n-1 do
        x[i]:=a[m-n+i+1,n]
end gomory1;
```

Свидетельство к алгоритму 153б

Алгоритм 153б получен в результате модификации и ординарной переработки алгоритма 263 (Langmaack H. «CACM», 1965, № 10). Модификация заключалась в том, что были добавлены параметры f и x, в которых получаются результаты решения, в то время как в алгоритме 263 результаты получались как значения некоторых элементов последнего столбца матрицы a. Данная модификация введена для удобства пользования алгоритмом.

Алгоритм 153б был транслирован на машине М-220 и дал правильные результаты для следующих примеров.

1. Минимизировать целевую функцию x_1+x_2 при условии

$$-x_1-x_2\leqslant -1,$$
$$2x_1-x_2\leqslant 1.$$

Были заданы следующие значения входных параметров:

$$m=4,\ n=3,\qquad a=\begin{Vmatrix} 1 & 1 & 0\\ -1 & -1 & -1\\ 2 & -1 & 1\\ -1 & 0 & 0\\ 0 & -1 & 0 \end{Vmatrix}.$$

Результат: $f=1$, $x=(0,1)$. Правильность результата очевидна.

2. Минимизировать целевую функцию $14x_1+25x_2$ при условии

$$3x_1-5x_2\leqslant -11,$$
$$-7x_1+2x_2\leqslant -13.$$

Результат трансляции: $f=142$, $x=(3,4)$ совпадает с результатом, приведенным в работе И. Л. Калихмана [52] на с. 242. Сама задача дана в этой работе на с. 239 как «двойственная задача 1′».

Метод Гомори подробно описывается, например, в работе А. А. Корбута и Ю. Ю. Финкельштейна [53]. См. также «Подтверждение к алгоритму 263» Л. Пролла, перевод которого помещен ниже. Номер алгоритма 263 заменен на номер 153б, потому что в журнале «CACM» под номером 263 были опубликованы два разных алгоритма, причем данный алгоритм был предложен в журнале «CACM» вместо алгоритма 153 («CACM», 1963, № 2).

Подтверждение к алгоритму 263 *

Л. Пролл (Prol1 L. G. «CACM», 1970, № 5)

Алгоритм 263 был использован в АЛГОЛ-программе для машины ICL 1907 и успешно прошел без каких-либо изменений. Время вычислений и количество итераций для 12 опубликованных задач даны в табл. 1.

Таблица 1

Задача	m	n	Число итераций	Время (с)
1	13	10	13	1
2	17	11	8	1
3	13	10	35	1
4	18	13	—	600*
5	14	8	9	1
6	14	8	16	1
7	10	8	16	1
8	30	16	17	2
9	30	16	2569	248
10	65	16	—	600*
11	41	32	5	2
12	31	27	5	2

* Выполненне процедуры не закончилось. (Прим. Л. Пролла.)

Задача 1 взята из работы К. Хейли [5i]. Задачи 2, 3 и 4 заимствованы из работы Е. Балаша [6i]. В этих задачах переменные могут принимать значения не только 0 или 1. Задачи 5—10 взяты из работы Дж. Халди [7i]. Задачи 11 и 12 принадлежат Дж. Пьерсу [8i].

Р. Вилсон в своей работе [9i] показал, что ценой небольшого увеличения количества вычислений можно получить более точные сечения, чем в методе Гомори.

Сечения Вилсона могут быть включены в процедуру *gomory*1 посредством следующих изменений **.

1. В начало тела процедуры вставить описание

```
Boolean null,nflag;
```

2. В строку

```
n4:     q:=k;
```

вставить оператор

```
        null:=true;
```

3. Заменить строку

```
n7:     end j;
```

следующими строками

```
n7:     end else
      null:=false;
      c:=euclid(a[r,n]×lambda2,lambda1);
      s:=-(c+1); t:=-a[r,n]; nflag:=true;
      if null then go to n10;
      for j:=1 step 1 until n-1 do
        if a[r,j]>0 then
```

* Это подтверждение относится к алгоритму 153б. (*Прим. ред.*)

** Здесь используются обозначения, приняты в алгоритме 153б. (*Прим. ред.*)

```
          begin c:=euclid(a[r,j]×lambda2,lambda1);
            if s×a[r,j]<t×c then
              begin t:=a[r,j]; s:=c; nflag:=false end
          end j;
n10:  if s×lambda1<t×lambd2 then
        begin lambda1:=if nflag then 100×t—1 else t;
          lambda2:=if nflag then 100×s else s
        end;
```

4. Заменить строку

```
    begin c:=euclid(a[r,j]×lambda2,lambda1);
```

на следующие строки:

```
    begin
      c:=if lambda2≠0 then
        euclid(a[r,j]×lambda2,lambda1) else
        if a[r,j]<0 then—1 else 0;
```

После этих изменений несколько уменьшилось количество итераций, необходимых для решения задач 7 и 9. Новое число итераций и новое время вычислений для этих задач дано в табл. 2. Время вычисления для задач, не указанных в табл. 2, осталось без изменений.

Таблица 2

Задача	Число итераций	Время (с)
7	7	1
9	2238	235

АЛГОРИТМ 154б

Генератор лексикографически упорядоченной последовательности сочетаний [G6]

Процедура *comb*1 (*combination* — сочетание) образует различные сочетания из целых чисел (1, 2, ..., *n*), взятых по *r*. Сочетания генерируются в лексикографическом порядке и запоминаются в массиве *c*[1 : *r*], причем первым образуется сочетание (1, 2, ..., *r*). Перед первым обращением к процедуре *comb*1 логический параметр *prim* должен иметь значение **true**, а значение массива *c* несущественно. Перед последующими обращениями как *prim*, так и *c* должны иметь значения, полученные ими в предыдущем обращении к процедуре *comb*1. Выходное значение параметра *prim* будет **false** до получения последнего сочетания *n*—*r*+1, *n*—*r*+2, ..., *n*. При последующем обращении к процедуре *comb*1 значение массива *c* не изменится, а параметр *prim* примет значение **true**.

```
procedure comb1(n,r)dataresult:(c,prim);
      value n,r; integer n,r; Boolean prim; integer array c;
begin integer i,j;
      if prim then
        begin prim:=false;
          for j:=1 step 1 until r do c[j]:=j;
          go to final
```

```
      end;
    if c[r]<n then
      begin c[r]:=c[r]+1; go to final end;
    for j:=r—1 step—1 until 1 do
      if c[j]<n—r+j then
        begin c[j]:=c[j]+1;
          for i:=j+1 step 1 until r do c[i]:=c[j]+i—j;
          go to final
        end j;
    prim:=true;
final: end comb1;
```

Свидетельство к алгоритму 154а

Алгоритм 154а получен в результате ординарной переработки алгоритма 154 (Mifsud Ch. J. «CACM», 1963, № 3).

Алгоритм 154а транслирован для $n=4$, $r=2$ и 3. Получены правильные результаты:

$r=2$		$r=3$	
(1,2)	(2,3)	(1,2,3)	(1,3,4)
(1,3)	(2,4)	(1,2,4)	(2,3,4)
(1,4)	(3,4)		

Подтверждение к алгоритму 154

К. Басворт (Bosworth K. M. «CACM», 1963, № 8)

Эта процедура была проверена для $r=1$, 2, 3, 4, 5, 6 и $n=6$ и дала правильные результаты.

АЛГОРИТМ 155б

Генератор сочетаний с повторениями [G6]

Каждое обращение к процедуре *comb2* (*combination* — сочетание) дает новое сочетание c (если оно существует) из n целых чисел массива a, взятых по r ($r\geqslant 1$), причем массив a составляется процедурой из $m[1]$ целых, каждое из которых равно $mm[1]$, из $m[2]$ целых, равных $mm[2]$, и т. д. до $m[s]$ включительно, т. е. $n=m[1]+m[2]+\ldots m[s]$. Перед первым обращением к процедуре *comb2* нужно задавать параметру *total* значение нуль, в дальнейшем после образования каждого нового сочетания процедура увеличивает значение *total* на единицу. При первом обращении значения $a[1:n]$, $b[1:r]$ и $c[1:r]$ несущественны. При последующих обращениях нужно оставлять эти массивы такими, какими они получились в результате предыдущего обращения к этой процедуре. Последнее относится и к параметру *prim*.

Выполнение процедуры на машине может быть ускорено, если перед первым обращением к процедуре расположить s целых в массиве mm так, чтобы $m[1]\geqslant m[2]\geqslant\ldots\geqslant m[s]$.

Процедура *comb2* использует глобальную процедуру *comb1* (алгоритм 154б). Значение **true** параметра *prim* означает, что получено последнее сочетание и новое обращение к процедуре его не изменит. Значение параметра *prim* перед первым обращением несущественно.

```
procedure comb2(m,mm,n,r,s)dataresult:(total,a,b,c,prim);
      value n,r,s; integer n,r,s,total; Boolean prim;
      integer array m, mm,a,b,c;
begin integer i,j,t,p;
      if total=0 then
        begin p:=0
          for j:=1 step 1 until s do
            begin t:=p+1; p:=p+m[j];
              for i:=t step 1 until p do a[i]:=mm[j]
            end j;
          prim:=true
        end;
n1:   comb1(n,r,b,prim);
      if prim then go to final;
      if b[1]=1 then go to n2;
      if a[b[1]]=a[b[1]-1] then go to n1;
n2:   for j:=2 step 1 until r do
        if (a[b[j]]=a[b[j]-1])∧(b[j]>b[j-1]+1) then go to n1;
      for j:=1 step 1 until r do c[j]:=a[b[j]];
      total:=total+1;
final:end comb2;
```

Свидетельство к алгоритму 155а

Алгоритм 155а получен в результате модификации, сокращения и ординарной переработки алгоритма 155 (Mifsud Ch. J. «CACM», 1963, № 3). Модификация заключалась в замене собственных массивов *I* и *J* формальными параметрами *b* и *a*, а собственной переменной *first* параметром *prim* для того, чтобы ограничить изобразительные средства алгоритма рамками стандартного сокращенного АЛГОЛа-60 [2].

Алгоритм 155а транслирован для исходных данных, указанных в нижеследующем «Подтверждении», т. е. $n=11$, $s=4$, $m=(4,3,2,2)$, $mm=(4,7,9,16)$.

Для $r=1$ получено четыре следующих сочетания:

(4) (7) (9) (16).

Для $r=2$ получено десять следующих сочетаний:

(4,4) (4,7) (4,9) (4,16) (7,7)
(7,9) (7,16) (9,9) (9,16) (16,16)

Подтверждение к алгоритму 155

К. Басворт (Bosworth K. M. «CACM», 1963, № 8)

Эта процедура была проверена для $r=1,2,3,4$ и

$m[1]=4$ $m[2]=3$ $m[3]=2$ $m[4]=2$
$mm[1]=4$ $mm[2]=7$ $mm[3]=9$ $mm[4]=16$.

В результате были правильно получены 4 сочетания из чисел 4, 7, 9, 16 для $r=1$, 10 сочетаний для $r=2$, 18 сочетаний для $r=3$ и 26 сочетаний для $r=4$.

При этом были сделаны следующие изменения в процедуре, вызванные ограничениями конкретного транслятора.

1. Систематические изменения прописных букв там, где могло возникнуть противоречие при использовании одного только регистра латинского алфавита.
2. Замена переменных **own** глобальными переменными.
3. Замена целочисленных меток идентификаторами.

АЛГОРИТМ 156б

Сумма знакопеременного ряда произведений из элементов сочетаний [G6]

Процедура *sumcomb* (*sum* — сумма, *combination* — сочетание) вычисляет сумму

$$sum = \Sigma_1 a_i - \Sigma_2 a_i a_j + \Sigma_3 a_i a_j a_k - \ldots (-1)^n a_1 a_2 \ldots a_n,$$

где символы Σ_1, Σ_2, ..., Σ_{n-1} означают суммирование произведений из всевозможных сочетаний чисел a_1, a_2, ..., a_n, взятых по одному, по два, по три и т. д. до $n-1$. Например,

$$\Sigma_1 a_i = a_1 + a_2 + \ldots + a_n,$$

$$\Sigma_2 a_i a_j = a_1 a_2 + a_1 a_3 + \ldots + a_2 a_3 + a_2 a_4 + \ldots + a_{n-1} a_n$$

и т. п.

```
procedure sumcomb(a,n) result: (sum);
    value n; real sum; integer n; array a;
begin integer i;
    sum := 1;
    for i := 1 step 1 until n do sum := sum × (1 — a[i]);
    sum := 1 — sum
end sumcomb;
```

Свидетельство к алгоритму 156а

Алгоритм 156а составлен заново в несколько раз более краткой и экономной форме по сравнению с алгоритмом 156 (Mifsud Ch. J. «CACM», 1963, № 3). Новая форма алгоритма настолько проста и очевидна, что не требует никакого контрольного решения. В соответствии с этим «Подтверждение к алгоритму 156» (Bosworth K. M. «CACM», 1963, № 8) здесь не приводится.

Возможность упрощения алгоритма 156 по формуле

$$sum = 1 - \prod_{i=1}^{n} (1 - a_i)$$

была указана рецензентом алгоритма 156а.

АЛГОРИТМ 157б

Аппроксимация рядами Фурье [E2]

Если заданы числа f_i, которые могут пониматься как значения некоторой функции $f(x)$ в точках $x_i = 2\pi i/k$, где $i = 0, 1, \ldots, k$, то процедура *fourier*157 вычисляет коэффициенты a_0, a_1, ..., a_n и b_1, b_2, ..., b_n (где $n = k \div 2$) такие, что при k нечетном удовлетворяются уравнения

$$f_i=\frac{a_0}{2}+\sum_{p=1}(a_p\cos x_ip+b_p\sin x_ip),$$

а при k четном — уравиения

$$f_i=\frac{a_0}{2}+\sum_{p=1}^{n-1}(a_p\cos x_ip+b_p\sin x_ip)+\frac{a_n}{2}\cos\pi i.$$

Массивы-параметры должны иметь размерность a, $b[0:n]$ и $f[0:k-1]$. Выходное значение коэффициента $b[0]$ (а при k четном и коэффициента $b[n]$) для решения задачи несущественно. В процедуре используется константа $pi=\pi=3.14\ldots$, необходимая точность представления которой может зависеть от машины.

О рядах Фурье см., например, работу Г. П. Толстова [10]. Автор исходного алгоритма 157 указывает, что использованный им метод изложен в работе Г. Гюрцеля [11].

Из нижеприведенной процедуры *fourier*157 видно, что метод нахождения коэффициентов a_p и b_p в алгоритме 157а заключается в последовательном (для $p=0, 1, \ldots, n$) выполнении рекуррентных соотношений

$$a_p=2(f_0+c_pu_{k-1}-u_{k-2})/k;$$
$$b_p=2s_pu_{k-1}/k;$$
$$c_{p+1}=c_p\cos 2\pi/k-s_p\sin 2\pi/k;$$
$$s_{p+1}=s_p\cos 2\pi/k+c_p\sin 2\pi/k,$$

где $c_0=1$ и $s_0=0$, а u_{k-1} и u_{k-2} для каждого нового значения p вычисляются по рекуррентной формуле

$$u_{j-1}=f_{k-j}+2c_pu_j-u_{j-1},$$

где $j=1, 2, \ldots, k-1$ и $u_0=u_1=0$.

```
procedure fourier157(k,f)result:(a,b);
      value k; integer k; array a,b,f;
begin real temp,pi,s1,s2,c1,c2,u0,u1,u2,r; integer i,p,n;
      pi:=3.141592 65359; r:=2/k;
      c1:=cos(r×pi);
      s1:=sin(r×pi);
      c2:=1; s2:=0; n:=k÷2;
      for p:=0 step 1 until n do
        begin u1:=u2:=0;
          for i:=k-1 step -1 until 1 do
            begin u0:=f[i]+2×c2×u1-u2;
              u2:=u1; u1:=u0
            end i;
          a[p]:=r×(f[0]+u1×c2-u2);
          b[p]:=r×u1×s2;
          temp:=c1×c2-s1×s2;
          s2:=c1×s2+s1×c2; c2:=temp
        end p
end fourier157;
```

Свидетельство к алгоритму 157а *

Алгоритм 157а получен в результате ординарной переработки алгоритма 157 (Mifsud Ch. J. «CACM», 1963, № 3) и усовершенствования его для возможности использования при любом k (четном или нечетном) без внесения каких-либо изменений в тело процедуры. Кроме того, алгоритм был оптимизирован согласно рекомендации Р. Георга, данной в нижеследующем его «Замечании к алгоритму 157».

Ручной прокруткой процедуры *fourier* 157 было установлено, что алгоритм дает следующие значения для коэффициентов.

1. Для $k=2$: $a_0=f_0+f_1$ и $a_1=f_0-f_1$.
2. Для $k=3$: $a_0={}^2/_3(f_0+f_1+f_3)$, $a_1={}^1/_3(2f_0-f_1-f_2)$ и $b_1={}^{\sqrt{3}}/_3(f_1-f_2)$.
3. Для $k=4$: $a_0={}^1/_2(f_0+f_1+f_2+f_3)$, $a_1={}^1/_2(f_0-f_2)$, $a_2={}^1/_2(f_0-f_1+f_2-f_3)$ и $b_1={}^1/_2(f_1-f_3)$.

В правильности этих результатов можно легко убедиться путем непосредственного решения систем уравнений, данных в тексте, предшествующем описанию процедуры, или путем подстановки в эти уравнения полученных в результате прокрутки выражений.

Алгоритм 157а был транслирован в следующих трех вариантах исходных данных.

Вариант 1. Для функции

$$f(x)=\begin{cases} x^2 & \text{при } 0\leqslant x\leqslant\pi, \\ (2\pi-x)^2 & \text{при } \pi<x\leqslant 2\pi \end{cases}$$

результаты трансляции приведены в табл. 3 (первые шесть коэффициентов a_p), где контрольные значения взяты из [9, с. 556, пример 8] и соответствуют ряду

$$f(x)=\frac{\pi^2}{3}-4\left(\frac{\cos x}{1^2}-\frac{\cos 2x}{2^2}+\frac{\cos 3x}{3^2}-\ldots\right),$$

Таблица 3

a_p	Результаты трансляции			Контрольные значения
	$k=51$	$k=201$	$k=801$	
a_0	6.57720656	6.57957335	6.57972601	6.5799736
a_1	—3.99746694	—3.99983718	—3.99998940	—4.0000000
a_2	0.997456832	0.999837117	0.999989569	1.0000000
a_3	—0.441884329	—0.444281461	—0.444434182	—0.4444444
a_4	0.247415972	0.249836919	0.249989719	0.2500000
a_5	—0.157384894	—0.159836790	—0.159989744	—0.1600000

Абсолютные значения коэффициентов b_p, которые должны равняться нулю, в результате трансляции для $k=51$ были $0.46\cdot10^{-7}$—$0.19\cdot10^{-6}$, а для $k=801$ были $0.16\cdot10^{-10}$—$0.59\cdot10^{-6}$.

Вариант 2. Хуже результаты были для функции $f(x)=x$ в интервале $0\leqslant x\leqslant 2\pi$, представляемой медленно сходящимся рядом [9, с. 555, пример 1]

$$f(x)=\pi-2\left(\frac{\sin x}{1}+\frac{\sin 2x}{2}+\frac{\sin 3x}{3}+\ldots\right),$$

* См. также «Подтверждения к алгоритмам 157а и 199а» в приложении 1 к выпуску [49]. (*Прим. ред.*)

Результаты аппроксимации этой функции приведены в табл. 4.

Таблица 4

b_p	Результаты трансляции			Контрольные значения
	$k=51$	$k=201$	$k=801$	
b_0	0.00000000	0.00000000	0.00000000	0.000000000
b_1	—1.99746966	—1.99983722	—1.99998934	—2.00000000
b_2	—0.994935482	—0.999674286	—0.999979397	—1.00000000
b_3	—0.659060286	—0.666178022	—0.666635831	—0.666666667
b_4	—0.489840023	—0.499348391	—0.499958928	—0.500000000
b_5	—0.387270800	—0.399185365	—0.399948798	—0.400000000
a_0	3.07999278	3.12596287	3.13767078	3.14159265

Абсолютные значения a_p $(p \geqslant 1)$, которые должны быть равны нулю, в результате трансляции были примерно равны 0.0078 для k=801, 0.0312 для k=201 и 0.1232 для k=51.

Вариант 3. Результаты трансляции процедуры для четного k с той же функцией, которая была использована в варианте 1, приведены в табл. 5.

Таблица 5

a_p	Результаты трансляции			Контрольные значения
	$k=100$	$k=200$	$k=800$	
a_0	6.58500005	6.58006518	6.57975662	6.5799736
a_1	—4.00526795	—4.00032910	—4.00002103	—4.00000000
a_2	1.00528045	1.00032998	1.00002058	1.00000000
a_3	—0.449745850	—0.444778568	—0.444465012	—0.444444444
a_4	0.255330960	0.250322251	0.250020572	0.250000000
a_5	—0.165369342	—0.160322333	—0.160020561	—0.160000000

Замечание к алгоритму 157

Р. Георг (George R. «CACM», 1963, № 9)

Этот алгоритм был записан на языке FAP для машины 32-K IBM 704. Он был проверен для зубчатой (пилообразной) ломаной. Зубцы были воспроизведены путем суммирования разложения по $2n+1$ константам с превосходными результатами.

Массивы s, c и u с переменными индексами нигде не используются. Для экономии машинного времени я предлагаю заменить их простыми переменными.

Ценой дополнительного описания еще одной переменной real можно вынести выражение $2/(2\times n+1)$ за цикл, поскольку n в теле процедуры не меняется. Это сэкономит $4n+2$ операции умножения.

Замечание к алгоритму 157

Г. Шуберт (Schubert G. R. «CACM», 1963, № 10)

Алгоритм 157 был модифицирован для аппроксимации $2n$ данных точек и успешно прошел на машине Burroughs 220 с использованием языка BALGOL. В этой модификации $2n$ коэффициентов определяются так, чтобы удовлетворялось уравнение

$$f_i = \frac{a_0}{2} + \sum_{p=1}^{n-1} \left(a_p \cos \frac{\pi i p}{n} + b_p \sin \frac{\pi i p}{n} \right) + \frac{a_n}{2} \cos \pi i.$$

В модифицированной процедуре операторы во второй и третьей строках после описаний должны иметь вид

c1 : =cos(pi/n); s1 : =sin(pi/n) *.

Второй заголовок цикла должен быть

for i : =2×n—1 step—1 until 1 do

Строки, содержащие коэффициенты a и b, должны читаться

a[p] : =(f[0]+u1×c2—u2)/n;

b[p] : =u1×s2/n;

См. работу Р. Хэмминга [2i, с. 68—73].

Свидетельство к алгоритму 158б [C1]

Алгоритм 158б (Возведение степенного ряда в степень) здесь не публикуется, потому что результат переработки соответствующего алгоритма 158 (Fettis H. E. «CACM», 1963, № 3) уже опубликован под номером 134б (см. [49]).

АЛГОРИТМ 159б

Вычисление определителя (рекурсивная процедура) [F3]

Процедура *determinant* вычисляет определитель матрицы $x[1:n, 1:n]$, используя его комбинаторное определение.

Данный алгоритм может служить примером рекурсивной процедуры, которая не столь тривиальна, как процедура *fact* алгоритма 33б.

```
real procedure determinant(x,n);
    value n; integer n; array x;
begin real d; integer i; Boolean array b[1 : n];
    procedure thread(p,e,i);
      value p,e,i; real p; integer e,i;
      if i>n then d : =d+p×(—1)↑e else
      if p≠0 then
        begin integer j,f;
          f : =0;
          for j : =n step—1 until 1 do
            if b[j] then f : =f+1 else
              begin b[j] : =true;
                thread(p×x[i,j],e+f,i+1);
                b[j] : =false
              end j
        end thread;
start: for i : =1 step 1 until n do b[i] : =false;
    d : =0;
```

* Здесь используются обозначения, принятые в алгоритме 157а. (*Прим. ред.*)

```
        thread(1,0,1);
        determinant:=d
end determinant;
```

Свидетельство к алгоритму 159а*

Алгоритм 159а получен в результате ординарной переработки алгоритма 159 (Digby D. W. «CACM», 1963, № 3). Алгоритм был транслирован на машине БЭСМ-6 и дал правильный результат для

$$x=\begin{vmatrix}1 & 2 & 3\\ 2 & 3 & 1\\ 3 & 1 & 2\end{vmatrix}=-18.$$

Подтверждение к алгоритму 159

А. Лапидус (Lapidus A. «CACM», 1963, № 12)

Алгоритм 159 был транслирован на машине IBM 7090 как составная часть контрольных подпрограмм на языке ФОРТРАН с целью реализации рекурсивных процедур. Как и ожидалось, результаты получились плохие. Для матрицы Гильберта с элементами $a_{ij}=1/(i+j-1)$ результаты были следующие:

n	Должно быть	Результат трансляции
2	8.333 333 3 (—2)	8.333 333 2 (—2)
3	4.629 629 6 (—4)	4.629 623 1 (—4)
4	1.653 439 2 (—7)	1.651 933 4 (—7)
5	3.749 295 1 (—12)	—2.910 383 0 (—11)

Были также вычислены определители 4- и 6-го порядков с целыми элементами. В этом случае алгоритм дал полную точность.

АЛГОРИТМ 160б

Число сочетаний [S03]

Процедура-функция *comb* (*combination* — сочетание) определяет число сочетаний из m элементов по n

$$C^n_m=\frac{m!}{n!\,(m-n)!}$$

по рекуррентной формуле. Если n больше половины m, то процедура вычисляет $C^{m-n}_m=C^n_m$.

Данный алгоритм дает то же самое, что и алгоритм 19б, но с несколько меньшими затратами машинного времени за счет несколько большей затраты машинной памяти.

```
integer procedure comb(m,n);
      value m,n; integer m,n;
begin integer p,r,i;
      p:= if n≤m—n then n else m—n;
      n:=m—p;
```

---

* См. также «Подтверждение к алгоритму 159а» в приложении 1 к выпуску [49]. (*Прим. ред.*)

```
    r := if p=0 then 1 else n+1;
    for i := 2 step 1 until p do r := (n+i)/i×r;
    comb := r
end comb;
```

Свидетельство к алгоритму 160а

Алгоритм 160а получен в результате сокращения и ординарной переработки алгоритма 160 (Wolfson M. L., Wright H. V. «CACM», 1963, № 4).

Оператор $r := (r\times(n+i))/i$ алгоритма 160 был заменен оператором $r := (n+i)/i\times r$ с целью расширения области применимости алгоритма путем ликвидации преждевременного переполнения при больших n.

Алгоритм 160а дал правильные результаты при трансляции с исходными данными $m=10$ и $n=1, 3, 7$.

Правильность алгоритма 160 подтверждают Д. Торо (Thoro D. «CACM», 1963, № 8) и Р. Блекли (Blakely R. F. «CACM», 1963, № 10), а алгоритма 160а — З. А. Шиншинова и Б. Л. Шмульян [48, с. 118].

АЛГОРИТМ 161б

Вектор чисел всевозможных сочетаний из m элементов

[G6, S03]

Процедура *combvector* дает числа сочетаний $r_1=C^1_m$, $r_2=C^2_m$, ..., $r_n=C^n_m$ для $m\geqslant n$.

```
procedure combvector(m,n)result:(r);
    value m,n; integer m,n; integer array r;
begin integer i;
    r[1] := m;
    for i := 2 step 1 until n do r[i] := (m—i+1)/i×r[i—1]
end combvector;
```

Свидетельство к алгоритму 161а

Алгоритм 161а получен в результате ординарной переработки алгоритма 161 (Wright H. V., Wolfson M. L. «CACM», 1963, № 4). Оператор $r[i] := (r[i-1]\times(m-i+1))/i$ был заменен оператором $r[i] := (m-i+1)/i\times r[i-1]$ с целью предохранения от преждевременного переполнения при больших m и n.

Алгоритм 161а был транслирован для $m=n=10$. Получен правильный результат $r=(10, 45, 120, 210, 252, 210, 120, 45, 10, 1)$.

Правильность алгоритма 161 подтверждают Д. Торо (Thoro D. «CACM», 1963, № 10) и Д. Коллинз (Collins D. H. «CACM», 1963, № 10).

АЛГОРИТМ 162б

Вычерчивание графиков [J6]

Процедура *xymove* (*move* — перемещение, *xy* — координаты) определяет последовательность кодов, необходимую для перемещения пера

цифрового дискретного графопостроителя от начальной точки (xz, yz) до конечной точки (xn, yn) путем «наилучшей» аппроксимации прямой, соединяющей эти точки. Элементарное допустимое перемещение пера — это перемещение к смежной точке в плоской декартовой координатной сетке, причем допускаются и диагональные перемещения. Восемь элементарных перемещений пера кодируются следующим способом:

$$1=+y \quad 2=+x+y \quad 3=+x \quad 4=+x-y$$
$$5=-y \quad 6=-x-y \quad 7=-x \quad 8=-x+y$$

что соответствует рис. 1.

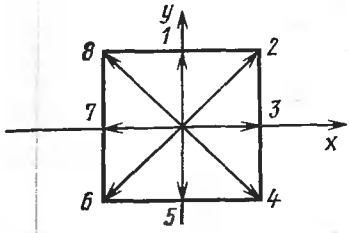

Рис. 1.

Аппроксимация считается «наилучшей» в том смысле, что каждая проходимая пером точка удалена от истинной линии, во всяком случае не дальше, чем любая другая точка того же самого перемещения.

Процедура *xymove* не пользуется умножением или делением.

Глобальная процедура *code*(*j*) принимает значения согласно следующей таблице:

j	1	2	3	4	5	6	7	8
code	1	2	3	2	3	4	5	4
j	9	10	11	12	13	14	15	16
code	5	6	7	6	7	8	1	8

Глобальная процедура *plot*(*move*) посылает значение переменной *move* на графопостроитель по его запросу.

```
procedure xymove(xz,yz,xn,yn);
      value xz,yz,xn,yn; integer xz,yz,xn,yn;
begin integer a,b,d,e,f,t,i,move;
      if xz=xn∧yz=yn then go to final;
      a:=xn—xz; b:=yn—yz;
      d:=a+b; t:=b—a; i:=0;
      if b≥0 then i:=2;
      if d≥0 then i:=i+2;
      if t≥0 then i:=i+2;
      i:=if a≥0 then 8—i else i+10;
      a:=abs(a); b:=abs(b);
      f:=a+b; d:=b—a;
      if d≥0 then
        begin t:=a; d:=—d end else t:=b;
      e:=0; move:=code(i—1); i:=code(i);
iter: a:=d+e;
      if t+e+a≥0 then
        begin e:=a; f:=f—2; plot(i) end else
```

```
    begin e:=e+t; f:=f—1; plot(move)end;
  if f>0 then go to iter;
final: end xymove;
```

Свидетельство к алгоритму 162а

Алгоритм 162а получен в результате ординарной переработки алгоритма 162 (Stockton F. G. «CACM», 1963, № 4), внесения в него поправки, указанной в нижеследующем «Подтверждении» В. Флетчера, и модификации согласно рекомендациям Д. Кейвина, указанным в его «Замечании» (см. ниже).

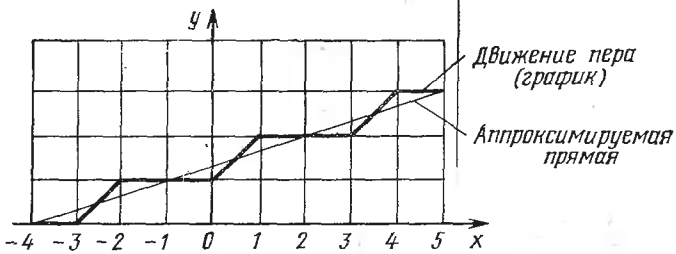

Рис. 2.

Алгоритм 162а транслирован с исходными данными $xz=-4$, $yz=0$, $xn=5$, $yn=3$, причем обращение к процедуре *plot* было заменено обращением к процедуре вывода на печать значений переменных *move* и *i*. В результате трансляции была получена следующая последовательность элементарных движений пера: 3, 2, 3, 3, 2, 3, 3, 2, 3. Правильность этих результатов показана на рис. 2.

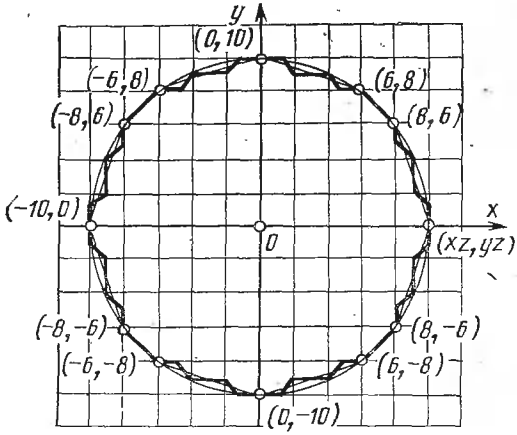

Рис. 3.

Далее таким же способом был аппроксимирован многоугольник с вершинами (10,0), (8,6), (6,8), (0,10), (—6,8), (—8,6), (—10,0), (—8, —6), (—6, —8), (0, —10), (6, —8), (8, —6) при $xz=10$ и $yz=0$. Результаты аппроксимации показаны на рис. 3 и сведены в табл. 6.

Таблица 6

Координаты вершин		Коды перемещений пера
xn	*yn*	
8	6	1, 8, 1, 1, 8, 1
6	8	8, 8
0	10	7, 8, 7, 7, 8, 7,
—6	8	7, 6, 7, 7, 6, 7
—8	6	6, 6
—10	0	5, 6, 5, 5, 6, 5
—8	—6	5, 4, 5, 5, 4, 5
—6	—8	4, 4
0	—10	3, 4, 3, 3, 4, 3
6	—8	3, 2, 3, 3, 2, 3
8	—6	2, 2
10	0	1, 2, 1, 1, 2, 1

Подтверждение к алгоритму 162

В. Флетчер (Fletcher W. E. «CACM», 1963, № 8)

В теле процедуры строка

```
if d≥ then i : =i+2;
```

была исправлена на строку

```
if d≥0 then i : =i+2;
```

С этим изменением тело процедуры было переведено на язык DECAL-BBN и успешно проверено на машине PDP-1 с использованием вывода на осциллоскоп для получения изображения траектории такого же, как на цифровом дискретном графопостроителе.

Замечания к алгоритму 162

Д. Кейвин (Cavin D. K. «CACM», 1964, № 8)

Для уменьшения среднего времени выполнения алгоритма 162 в нем были сделаны следующие изменения... *

Очевидно, что при любом движении, содержащем более двух элементарных перемещений пера, использование процедуры *code* в цикле излишне, поскольку для аппроксимации любой прямой необходимы не более двух из восьми допустимых перемещений пера. Следовательно, вынос обращения к кодовой процедуре из основного цикла уменьшает время выполнения всякий раз, когда движение требует более двух элементарных перемещений пера.

Эта процедура была закодирована на языке CODAP для машины CDC 1604-A и в этом модифицированном варианте работала примерно на 40% быстрее. При сравнении времени выполнения процедур использовались числа в интервале от —2000 до 2000, причем особое значение придавалось подынтервалу от —150 до 150.

Типографская опечатка, отмеченная в «Подтверждении» («CACM», 1963, № 8), была исправлена в обеих программах.

(Рецензент подтверждает, что алгоритм 162 с указанной модификацией действительно проходит — *Г. Форсайт.*)

---

* Приводится модификация девяти последних строк процедуры, учтенная при составлении алгоритма 162а. (*Прим. ред.*)

АЛГОРИТМ 163б

Модифицированная функция Ханкеля [S17]

Процедура *expk* (*exponential* — экспоненциальная, а *k* соответствует $K_p(x)$) вычисляет модифицированную функцию Ханкеля $e^x K_p(x)$ с заданной точностью *e* по интегральной формуле

$$e^x K_p(x) = \int_0^\infty e^{x(1-\operatorname{ch} t)} \operatorname{ch}(pt)\, dt$$

```
real procedure expk(p,x,e);
      value p,x,e; real p,x,e;
begin real f,g,h,r,s,u,z,zp;
      r := 0; h := 1;
iter: g := r; s := 0;
      z := exp(0.5×h); u := z×z;
intgr:zp := z↑p;
      f := 0.5×exp(x×(1—0.5×(z+1/z)))×(zp+1/zp);
      s := s+f; z := z×u;
      if f ⩾ e then go to intgr;
      r := h×s; h := 0.5×h;
      if abs(r—g) ⩾ e then go to iter;
      expk := r
end expk;
```

Свидетельство к алгоритму 163а

Алгоритм 163а получен в результате исправления, сокращения и ординарной переработки алгоритма 163 (Fettis H. E. «CACM», 1963, № 4).

Алгоритм 163а транслирован для исходных данных, указанных в нижеследующем «Подтверждении к алгоритму 163» Г. Тачера. Результаты трансляции даны в табл. 7 и 8. В этих таблицах $a = \frac{2}{\pi} \frac{\exp k(p,x,e)}{\exp(x)}$, где $\exp k(p,x,e)$ — результат трансляции, а значения $(2/\pi) K_p(x)$ взяты из [8, с. 256—262]. В последней работе значения

Таблица 7

x	$p=0$		$p=1$	
	$\frac{2}{\pi} K_0(x)$	a	$\frac{2}{\pi} K_1(x)$	a
0.2	1.1158	1.115771	3.0405	3.040384
0.4	0.7095	0.709509	1.3906	1.390560
0.6	0.4950	0.494970	0.8294	0.829385
0.8	0.3599	0.359900	0.5486	0.548610
1.0	0.2680	0.268024	0.3832	0.383174
3.0	0.02212	0.022115	0.02556	0.025564
6.0	0.000792	0.000792	0.0008556	0.0008555
9.0	0.00003239	0.0000323910	0.00003415	0.000034145
12.0	0.0000014011	0.0000014010	0.0000014583	0.00000145829
14.0	0.0000001758	0.00000017579	0.0000001820	0.00000018196

1.2716 и 0.9681 для p=0.6666667 были указаны, по-видимому, неточно (см. также нижеследующее «Подтверждение» Г. Тачера).

Таблица 8

x	$p = 0.3333333$		$p = 0.6666667$	
	$\frac{2}{\pi}K_{1/3}(x)$	a	$\frac{2}{\pi}K_{2/3}(x)$	a
0.2	1.2601	1.260048	1.7837	1.783620
0.3	0.9607	0.960701	1.2716	1.264681
0.4	0.7676	0.767589	0.9681	0.965805
0.6	0.5253	0.525254	0.6257	0.625700
0.8	0.3776	0.377618	0.4354	0.435352
1.0	0.27911	0.279105	0.3148	0.314783
3.0	0.022476	0.02247572	0.023591	0.02359053
6.0	0.0007988	0.000798764	0.0008196	0.000819618
9.0	0.00003258	0.0000325815	0.00003316	0.0000331596
10.0	0.000011379	0.0000113790	0.000011562	0.0000115614

Подтверждение к алгоритму 163

Г. Тачер (Thacher H. C. «CACM», 1963, № 9)

Поскольку этот алгоритм является описанием функции, то заголовок процедуры должен начинаться с описателя real. Кроме этого не было обнаружено никаких синтаксических ошибок.

Тело процедуры было транслировано и прошло на машине LGP-30 с использованием дартмутской системы SCALP. Результаты для e=0.0001; x=0.1, 0.2, ..., 1.0; p=0.0000000, 0.3333333, 0.6666667 и 1.000000 совпали со значениями из таблиц Янке, Эмде и Лёша для 3—4 цифр, за исключением ошибок, обнаруженных в этих таблицах для $(2/\pi)K_{2/3}(x)$.

При x=0 возникло переполнение. В самой процедуре или в обращении к ней должна содержаться проверка, не задана ли переменная меньшей, чем величина *eps*, которая выбирается для предупреждения переполнений.

Было найдено, что алгоритм слишком медленно работает. Затраты времени на машине LGP-30 были порядка 6 мин. Значительная экономия времени может быть получена путем улучшения квадратурной формулы. В алгоритме это центральная формула (от метки *intgr* до условного оператора), полностью повторяющаяся для каждой итерации. Более эффективен здесь был бы модифицированный алгоритм Ромберга. Процедура, основанная на этом методе, была опубликована в журнале «Communs ACM»*.

АЛГОРИТМ 164б

Приближение поверхности ортогональными полиномами по методу наименьших квадратов [E2]

Процедура *surfacefit* (*surface* — поверхность, *fit* — приближение, аппроксимация) аппроксимирует, в смысле метода наименьших квад-

* См. алгоритм 60б. (*Прим. ред.*)

ратов, полиномиальную функцию двух независимых переменных значениями некоторой зависимой переменной, определяемой в точках прямоугольной сетки, построенной в плоскости независимых переменных. Использование при этом ортогональных полиномов приводит к очень простой системе линейных уравнений, а не к плохо обусловленной системе, которая получилась бы из обычных нормальных уравнений. В процедуре предусмотрена также мера улучшений, получающихся от каждого нового включаемого члена, которая приводит в дальнейшем к автоматическому выбору полиномиальной функции «наилучшей» степени, как это определено критерием Гаусса. Во многих случаях предварительная нормализация переменных обеспечивает значительное уменьшение ошибок округления.

Входные параметры процедуры:

x[*i*] и *y*[*j*] — независимые переменные;

u[*i*] и *w*[*j*] — соответствующие им веса;

z[*i*,*j*] — зависимая переменная;

imax и *jmax* — максимальные значения переменных *i* и *j*;

nmax — величина, большая максимального значения показателя степени переменной *x*;

mmax — величина, большая максимального значения показателя степени переменной *y*.

Выходные параметры:

beta[*n*,*m*] — мера улучшения, получающегося от присоединения члена $x^n y^m$;

phi[*n*,*m*] — полиномиальный коэффициент при члене $x^n y^m$ (заметим, что степень результирующего полинома может быть меньше, чем максимальная степень, определяемая как результат применения критерия Гаусса);

zcomp — массив значений вычисляемой зависимой переменной;

$$minsqd = \left(\frac{\sum_{i,j} u[i] \times w[j] \times z[i,j]^2 - \sum_{n,m} beta[n,m]}{imax \times jmax} \right)^{1/2},$$

$$minsqdcomp = \left(\frac{\sum_{i,j} u[i] \times w[j] \times (z[i,j] - zcomp[i,j])^2}{imax \times jmax} \right)^{1/2},$$

$$sumdifcomp = \frac{\sum_{i,j} |z[i,j] - zcomp[i,j]|}{imax \times jmax},$$

$$maxdifcomp = max\,|z[i,j] - zcomp[i,j]|.$$

Если вычисления произведены точно, то *minsqd* и *minsqdcomp* равны между собой. Практически же они не будут равны из-за неточности вычислений. Большая разница между ними указывает на возникновение чрезмерных ошибок в вычислениях.

Размерность массивов: *x*,*u*[1 : *imax*]; *y*,*w*[1 : *jmax*]; *z*,*zcomp*[1 : *imax*, 1 : *jmax*]; *beta*,*phi*[1 : *nmax*, 1 : *mmax*].

Результирующий полином получается не в исходных, а в нормализованных переменных.

Автор алгоритма 164 ссылается на работы Дж. Кадвела [15i, 16i].

```
procedure surfacefit(x,u,y,w,z,nmax,mmax,imax,jmax)
        result:(beta,phi,zcomp,minsqd,minsqdcomp,
        sumdifcomp,maxdifcomp);
        value nmax,mmax,imax,jmax;
        real minsqd,minsqdcomp,sumdifcomp,maxdifcomp;
        integer nmax,mmax,imax,jmax;
        array x,u,y,w,z,phi,beta,zcomp;
begin real sumx,sumy,sumz,meanx,meany,meanz,numa,
        dena,denb,numc,denc,dend,alph,sumzsq,gausscrit,
        trialgausscrit,betasum,rescomp,poly;
        integer n,m,i,j,s,t,r;
        array a,b,denpa[1:nmax],c,d,denqa[1:mmax],
        alpha[1:nmax, 1:mmax],p[1:nmax, 1:imax],
        q[1:mmax, 1:jmax],pc[1:nmax, 1:nmax],qc[1:mmax, 1:mmax];
        comment Нормализация переменных;
start: sumx:=sumy:=sumz:=0;
        for i:=1 step 1 until imax do sumx:=sumx+x[i];
        meanx:=sumx/imax;
        for i:=1 step 1 until imax do x[i]:=x[i]—meanx;
        for j:=1 step 1 until jmax do sumy:=sumy+y[j];
        meany:=sumy/jmax;
        for j:=1 step 1 until jmax do y[j]:=y[j]—meany;
        for i:=1 step 1 until imax do
          for j:=1 step 1 until jmax do sumz:=sumz+z[i,j];
        meanz:=sumz/(imax×jmax);
        for i:=1 step 1 until imax do
          for j:=1 step 1 until jmax do z[i,j]:=z[i,j]—meanz;
        comment Вычисление ортогональных полиномов;
        numa:=dena:=0;
        for i:=1 step 1 until imax do
          begin p[1,i]:=1;
            numa:=numa+u[i]×x[i]; dena:=dena+u[i]
          end i;
        a[2]:=numa/dena;
        for i:=1 step 1 until imax do p[2,i]:=x[i]—a[2];
        for n:=3 step 1 until nmax do
          begin numa:=dena:=denb:=0; j:=n—1; m:=n—2;
            for i:=1 step 1 until imax do
              begin alph:=u[i]×p[j,i]↑2;
                numa:=numa+alph×x[i];
                dena:=dena+alph; denb:=denb+u[i]×p[m,i]↑2
              end i;
            a[n]:=numa/dena; b[n]:=dena/denb;
            for i:=1 step 1 until imax do
              p[n,i]:=(x[i]—a[n])×p[n—1,i]—b[n]×p[n—2,i]
          end n;
numc:=denc:=0;
        for j:=1 step 1 until jmax do
          begin q[1,j]:=1;
            numc:=numc+w[j]×y[j]; denc:=denc+w[j]
          end j;
        c[2]:=numc/denc;
```

```
for j := =1 step 1 until jmax do q[2,j] := =y[j]—c[2];
for m := =3 step 1 until mmax do
  begin numc := =denc := =dend := =0; i := =m—1; n := =m—2;
    for j := =1 step 1 until jmax do
      begin alph := =w[j]×q[i,j]↑2;
        numc := =numc+alph×y[j]; denc := =denc+alph;
        dend := =dend+w[j]×q[n,j]↑2
      end j;
    c[m] := =numc/denc; d[m] := =denc/dend;
    for j := =1 step 1 until jmax do
      q[m,j] := =(y[j]—c[m])×q[i,j]—d[m]×q[n,j]
  end m;
comment Вычисление вклада каждого ортогонального полинома
в минимизацию разностей;
for n := =1 step 1 until nmax do
  begin denpa[n] := =0;
    for i := =1 step 1 until imax do
      denpa[n] := =denpa[n]+u[i]×p[n,i]↑2
  end n;
for m := =1 step 1 until mmax do
  begin denqa[m] := =0;
    for j := =1 step 1 until jmax do
      denqa[m] := =denqa[m]+w[j]×q[m,j]↑2
  end m;
for n := =1 step 1 until nmax do
  for m := =1 step 1 until mmax do
    begin alph := =0;
      for i := =1 step 1 until imax do
        for j := =1 step 1 until jmax do
          alph := =alph+
            u[i]×w[j]×z[i,j]×p[n,i]×q[m,j];
      alpha[n,m] := =alph/(denpa[n]×denqa[m]);
      beta[n,m] := =alpha[n,m]×alph
    end n;
comment Приложение критерия Гаусса к определению степени
полинома, дающего наилучшую аппроксимацию исходных дан-
ных. Строго говоря, критерий Гаусса применим только в тех слу-
чаях, когда веса u(i) и w(j) равны единице;
sumzsq := =0;
for i := =1 step 1 until imax do
  for j := =1 step 1 until jmax do
    sumzsq := =sumzsq+u[i]×w[j]×z[i,j]↑2;
s := =t := =1;
for n := =1 step 1 until nmax do
  begin betasum := =0;
    for m := =1 step 1 until mmax do
      begin
        for r := =1 step 1 until n do
          betasum := =betasum+beta[r,m];
        trialgausscrit := =
          if betasum>sumzsq then 0 else
            (sumzsq—betasum)/(imax×jmax—n×m);
```

```
          if n=1∧m=1 then gausscrit : =trialgausscrit;
          if gausscrit=trialgausscrit∧n×m<s×t then
            begin s : =n; t : =m end;
          if gausscrit>trialgausscrit then
            begin gausscrit : =trialgausscrit;
              s : =n; t : =m
            end
        end m
    end n;
nmax : =s; mmax : =t;
minsqd : =sqrt(gausscrit×(imax×jmax—
    nmax×mmax)/(imax×jmax));
comment Вычисление коэффициентов ортогонального полинома;
for n : =1 step 1 until nmax do
    begin pc[n,n] : =1; i : =n—1; j : =n—2;
      for s : =1 step 1 until i do
        begin pc[n,s] : =—a[n]×pc[i,s];
          if s≠1 then pc[n,s] : =pc[n,s]+pc[i,s—1];
          if s≠i then pc[n,s] : =pc[n,s]—b[n]×pc[j,s]
        end s
    end n;
for m : =1 step 1 until mmax do
    begin qc[m,m] : =1; i : =m—1; j : =m—2;
      for t : =1 step 1 until i do
        begin qc[m,t] : =—c[m]×qc[i,t];
          if t≠1 then qc[m,t] : =qc[m,t]+qc[i,t—1];
          if t≠i then qc[m,t] : =qc[m,t]—d[m]×qc[j,t]
        end t
    end m;
comment Вычисление коэффициентов аппроксимирующего
полинома;
for s : =1 step 1 until nmax do
          for t : =1 step 1 until mmax do
      begin phi[s,t] : =0;
        for n : =s step 1 until nmax do
          for m : =t step 1 until mmax do
            phi[s,t] : =phi[s,t]
              +alpha[n,m]×pc[n,s]×qc[m,t]
      end s;
comment Вычисление зависимых переменных с использованием
аппроксимирующего полинома;
minsqdcomp : =sumdifcomp : =maxdifcomp : =0;
for i : =1 step 1 until imax do
    for j : =1 step 1 until jmax do
      begin zcomp[i,j] : =0;
        for s : =nmax step—1 until 1 do
          begin poly : =phi[s,mmax];
            for t : =mmax—1 step—1 until 1 do
              poly : =poly×y[j]+phi[s,t];
            zcomp[i,j] : =zcomp[i,j]×x[i]+poly
          end s;
        rescomp : =z[i,j]—zcomp[i,j];
```

```
            zcomp[i,j] := zcomp[i,j] + meanz;
            minsqdcomp := minsqdcomp + u[i] × w[j] × rescomp↑2;
            sumdifcomp := sumdifcomp + abs(rescomp);
            if abs(rescomp) > maxdifcomp then
              maxdifcomp := abs(rescomp)
          end i;
      minsqdcomp := sqrt(minsqdcomp/(imax × jmax));
      sumdifcomp := sumdifcomp/(imax × jmax)
end surfacefit;
```

Свидетельство к алгоритму 164б

Алгоритм 164б получен в результате некоторых тождественных преобразований в алгоритме 164а, имеющих целью сокращение его записи и ускорение его работы.

Алгоритм 164б был транслирован в системе ТА-1М, и с его помощью было получено правильное решение для функции, приведенной в нижеследующем «Подтверждении к алгоритму 164» К. Биттерли. При этом задавались параметры $nmax=mmax=3$, $imax=jmax=5$, $x=(0,1,2,3,4)$, $u=(1,1,1,1,1)$, $y=(0,1,2,3,4)$ и $w=(1,1,1,1,1)$. Значения аппроксимируемой функции вычислялись с помощью оператора

```
for i := 1 step 1 until imax do
  for j := 1 step 1 until jmax do
    z[i,j] := 1 - x[i] + y[j] - x[i] × y[j] + x[i]↑2 - y[j]↑2
```

Значения аппроксимирующего полинома были получены в виде

$$zcomp = \begin{Vmatrix} 1 & 1 & -1 & -5 & -11 \\ 1 & 0 & -3 & -8 & -15 \\ 3 & 1 & -3 & -9 & -17 \\ 7 & 4 & -1 & -8 & -17 \\ 13 & 9 & 3 & -5 & -15 \end{Vmatrix},$$

что в точности совпадает со значением входной матрицы z. Коэффициенты аппроксимирующего полинома были получены в форме

$$phi = \begin{Vmatrix} 0 & -5 & -1 \\ 1 & -1 & 0 \\ 1 & 0 & 0 \end{Vmatrix},$$

что соответствует полиному

$$z = 0x^0y^0 - 5x^0y^1 - 1x^0y^2 + 1x^1y^0 - 1x^1y^1 + 0x^1y^2 + 1x^2y^0 + 0x^2y^1 + 0x^2y^2 = \\ = -5y - y^2 + x - xy + x^2.$$

Остальные параметры были равны нулю.

Свидетельство к алгоритму 164а

Алгоритм 164а получен в результате исправления, некоторых очевидных сокращений и ординарной переработки алгоритма 164 (Clark R. E., Kubik R. N., Phillips L. P. «CACM», 1963, № 4). Кроме ошибок, указанных в нижеследующем «Подтверждении к алгоритму 164», была замечена опечатка, заключающаяся в том, что после последнего из **comment** тела процедуры первый оператор

```
minsqdcomp := sumdifcomp := maxidifcomp := 0.0;
```

должен иметь вид

```
minsqdcomp := sumdifcomp := maxdifcomp := 0.0;
```

Подтверждение к алгоритму 164

К. Биттерли (Bitterli C. V. «CACM», 1963, № 8)

Алгоритм *surfacefit* был переведен на ФОРТРАН и успешно прошел на машине IBM 7094. Потребовались следующие поправки... *

В качестве исходных данных была использована функция

$$z=1-x+y-xy+x^2-y^2$$

для $x=0,1,2,3,4$ и $y=0,1,2,3,4$. В результате был получен полином

$$z=x-5y-xy+x^2-y^2,$$

который верен для нормализованных переменных.

Нужно указать в примечании к этой процедуре, что результирующий полином получается не в исходных, а в нормализованных переменных.

АЛГОРИТМ 165б

Полные эллиптические интегралы [S21]

Процедура *elliptic* вычисляет полные эллиптические интегралы первого и второго рода в форме

$$K(m1)=\int_0^{\pi/2}(1-(1-m1)\sin^2\psi)^{-1/2}\,d\psi=K\left(k,\frac{\pi}{2}\right)$$ и

$$E(m1)=\int_0^{\pi/2}(1-(1-m1)\sin^2\psi)^{1/2}d\psi=E\left(k,\frac{\pi}{2}\right)$$

с помощью арифметико-геометрического среднего. Точность ограничивается только точностью используемой арифметики.

За исключением проверок на особые значения параметра, вычисление K является лишь незначительной модификацией второй процедуры алгоритма 149б. Эти интегралы могут быть аппроксимированы с ограниченной (до 6 цифр) точностью и с помощью алгоритмов 55б и 56б. Если на данной машине квадратный корень вычисляется не очень быстро, то для точности, ограничивающейся шестью цифрами, последние алгоритмы, вероятно, более эффективны.

В качестве независимой переменной выбран дополнительный параметр $m1$, а не параметр m, $|k|$ или угол α. Иначе возможна серьезная потеря точности при получении $m1$ через другие независимые переменные, когда $m1$ мало, а $dK/dm1$ очень велико. Вышеуказанные параметры связаны между собой соотношением

$$m1=1-m=1-k^2=\cos^2\alpha.$$

Формальный параметр *eps* определяет относительную точность результата. Для предотвращения входа в бесконечный цикл значение *eps* не должно быть меньше, чем удвоенная погрешность подпрограммы вычисления квадратного корня. Если $m1\leqslant 0$ или $m1>1$, то осуществляется выход из процедуры к метке *signal*.

---

* Указываются четыре поправки к алгоритму 164, учтенные в алгоритме 164а. (*Прим. ред.*)

Тело этой процедуры было проверено с использованием дартмутского транслятора SCALP на машине LGP-30. При $eps=5_{10}-7$ результаты совпали с табличными с точностью до трех единиц седьмой значащей цифры.

Процедура использует константу $\pi=3.14\ldots$, максимальная точность представления которой зависит от конкретного применения алгоритма.

```
procedure elliptic(m1,eps,signal)result:(kk,ee);
      value m1,eps; real m1,eps,kk,ee; label signal;
begin real a,b,c,sum,t; integer fact;
      if m1>1∨m1≤0 then go to signal;
      a:=1; fact:=1; b:=sqrt(m1);
      t:=1-m1; sum:=0;
iter: sum:=sum+t: c:=(a-b)/2;
      fact:=2×fact; t:=(a+b)/2;
      b:=sqrt(a×b); a:=t; t:=fact×c×c;
      if (abs(c)≥eps×a)∨(t>eps×sum) then go to iter;
      kk:=3.14159265359/(a+b);
      sum:=sum+t;    ee:=kk×(1-sum/2)
end elliptic;
```

Свидетельство к алгоритму 165б

Алгоритм 165б получен из алгоритма 165а в результате внесения в процедуру *elliptic* второй поправки, указанной в нижеследующем «Подтверждении к алгоритму 165» И. Фаркаса. Поскольку эта поправка элементарна и на результат вычисления не влияет, то новой трансляции алгоритма не делалось.

Свидетельство к алгоритму 165а

Алгоритм 165а получен в результате ординарной переработки алгоритма 165 (Thacher H. C. «CACM», 1963, № 4).

Результаты трансляции алгоритма 165а приведены в табл. 9, где контрольные значения K и E взяты из [8, с. 117].

Таблица 9

α град.	$m1$	Результаты трансляции		Контр. значения	
		kk	ee	K	E
0	1.00000000	1.57079632	1.57079632	1.5708	1.5708
20	0.883022222	1.62002589	1.52379920	1.6200	1.5238
40	0.586824090	1.78676913	1.39314024	1.7868	1.3931
60	0.250000001	2.15651564	1.21105602	2.1565	1.2111
80	0.0301536902	3.15338524	1.04011439	3.1534	1.0401
85	0.00759612384	3.83174197	1.01266350	3.8317	1.0127
89	0.000304586574	5.43490968	1.00075157	5.4349	1.0008
90	$\sim 0.724\times10^{-17}$	21.1192370	0.999999999	∞	1

Подтверждение к алгоритму 165

И. Фаркас (Farkas I. «CACM», 1969, № 1)

В алгоритме 165 была обнаружена одна опечатка и одна семантическая ошибка.

1. В списке формальных параметров вместо *m1* должно быть *m*1 *.
2. Второй оператор тела процедуры

a : =fact : =1;

должен быть заменен на оператор

fact : =1; a : =1;

поскольку *fact* и *a* разных типов.

Алгоритм 165 был переведен на ФОРТРАН IV для машины IBM 7094-II, в которой числа имеют мантиссу из 27 двоичных значащих цифр (около 8 десятичных значащих цифр). Поскольку наша программа, вычисляющая значение функции *sqrt*, имеет относительную погрешность 0.75_{10}—8, то параметр *tol* (в алгоритме 165а параметр *eps*. — *Прим. ред.*) был выбран равным 3_{10}—8. Значения *K* и *E* были вычислены для *m1*=0.01(0.01)1.0, и результаты сравнивались с данными таблиц М. Абрамовича и И. Стегуна [8i]. Значения *E* для *m*1=0.01 отличались от контрольных двумя единицами последней цифры, для всех других значений *m*1 абсолютная погрешность не превышала одной единицы в последней цифре. Для вычисления 100 значений потребовалось 0.1 с.

АЛГОРИТМ 166б

Обращение матрицы методом Монте-Карло [F1]

Процедура *montecarlo* может вычислять отдельную строку матрицы, обратной по отношению к данной, используя метод Монте-Карло.

Входные параметры:

a — данная матрица. Размерность массива a[1 : n, 1 : n].

row — номер вычисляемой строки обратной матрицы.

tol — допустимое отклонение, а следовательно, и критерий окончания процесса.

mxm — максимально допустимое число тысяч случайных блужданий, после которого процесс оканчивается.

Выходные параметры:

inv — массив, содержащий обращенную строку. Размерность *inv*[1 : *n*].

test — массив, содержащий скалярные произведения вектора *inv* на столбцы матрицы *a*. Размерность массива *test*[1 : *n*]. Он должен совпадать со строкой единичной матрицы, имеющей номер *row*.

count — на выходе равно числу случайных блужданий, выполненных до окончания вычисления.

Глобальный параметр:

random — вещественная процедура-функция, получающая случайное значение в интервале [0,1].

Если *a* — матрица, подлежащая обращению, то для обеспечения сходимости наибольшее собственное значение матрицы *e*—*a* (где *e* — единичная матрица порядка *n*) по модулю должно быть меньше единицы.

Данная процедура легко может быть приспособлена для нахождения одного неизвестного из совместной системы линейных уравнений.

---

* Эта опечатка была уже замечена ранее авторами выпусков и исправлена при составлении алгоритма 165а. (*Прим. ред.*)

```
      procedure montecarlo(n,a,row,tol,mxm)result:(inv,test,count);
      value n,row,tol,mxm; real tol;
      integer n,row,mxm,count;
      array a,inv,test;
begin real res,p,g; integer i,k,nwk,lastwalk,walk;
      array sum[1 : n], v[1 : n, 1 : n];
start: p := (n—1)/n↑2;
      for i := 1 step 1 until n do
        for k := 1 step 1 until n do
          v[i,k] := if i≠k then —a[i,k]/p else
            (1—a[i,k])/p;
      nwk := 1000; count := res := 0;
      for k := 1 step 1 until n do test[k] := sum[k] := 0;
n1:   lastwalk := row; g := 1;
n2:   walk := entier(random/p+1);
      if walk > n then go to step;
      g := v[lastwalk,walk]×g; lastwalk := walk;
      go to n2;
step: count := count+1; sum[lastwalk] := sum[lastwalk]+g;
      if count < nwk then go to n1;
      for k := 1 step 1 until n do
      inv[k] := n×sum[k]/count;
      for i := 1 step 1 until n do
        for k := 1 step 1 until n do
          test[i] := inv[k]×a[k,i]+test[i];
      for i := 1 step 1 until row—1,
        row+1 step 1 until n do
          res := abs(test[i])+res;
      res := abs(test[row]—1)+res;
      if res < tol then go to exit;
      if count ≥ 1000×mxm then go to exit;
      nwk := nwk+1000; res := 0;
      for k := 1 step 1 until n do test[k] := 0;
      go to n1;
exit: end montecarlo;
```

Свидетельство к алгоритму 166б

Алгоритм 166б является стереотипным переизданием алгоритма 166а, если не считать того, что метка *stop* была заменена меткой *step*, поскольку идентификатор *stop* в некоторых АЛГОЛ-трансляторах (например, в ТА-1М) является стандартным и в качестве метки использоваться не может.

Алгоритм 166б был транслирован в системе ТА-1М на машине М-220, и для проверки его правильности была взята элементарная матрица

$$a = \begin{Vmatrix} 0.95 & 0.00 \\ 0.10 & 0.95 \end{Vmatrix}$$

такая, что собственное значение разности $e—a$ очевидно мало и равно 0.05. Точное значение обратной матрицы есть

$$a^{-1} = \begin{Vmatrix} 1.0526316 & 0,0000000 \\ -0.11080332 & 1.0526316 \end{Vmatrix}.$$

Задавались параметры: n=2, mxm=100, row=1; 2 и tol=10^{-2}, 10^{-3}, 10^{-4}. Результаты решения приведены в табл. 10.

Таблица 10

tol	*Result*	*row*=1		*row*=2	
10^{-2}	*inv*	1.0533504	0.00000000	—0.10403200	1.0508902
	test	1.0006828	0.00000000	0.0062586	0.99834572
	count	1000		3000	
10^{-3}	*inv*	1.0533504	0.00000000	—0.11094483	1.0520159
	test	1.0006828	0.00000000	—0.00019600	0.99941512
	count	1000		45000	
10^{-4}	*inv*	1.0525859	0.00000000	—0.11088951	1.0526463
	test	0.9995665	0.00000000	—0.00008039	1.0000142
	count	33000		66000	

О времени решения можно судить по значению *count*, поскольку на каждую 1000 блужданий затрачивалось примерно 4 с.

Для получения случайных чисел вместо процедуры-функции *random* использовалась процедура-код *p*1147 системы ТА-1М. Поэтому в процедуру *montecarlo* были временно внесены следующие изменения:

1. В начало тела процедуры добавлено описание

real al,x;

2. В качестве первого оператора тела процедуры добавлено обращение *p*1147(*al*).

3. Оператор с меткой *n*2 был заменен на операторы

n2: p1147(al,x); walk:=entier(x/p+1);

Свидетельство к алгоритму 166а

Алгоритм 166а получен в результате ординарной переработки и некоторых сокращений в алгоритме 166 (Rodman R. D. «CACM», 1963, № 4), а также внесения в него поправок, указанных в нижеследующем «Замечании» Р. Родмана.

Замечание к алгоритму 166

Р. Д. Родман (Rodman R. D. «CACM», 1963, № 9)

Алгоритм содержит две ошибки...*

После внесения этих исправлений процедура *montecarlo* была переведена на расширенный АЛГОЛ и успешно прошла на машине Burroughs B-5000. Сходимость имела место во всех случаях, когда мат-

* Указываются две ошибки, учтенные в алгоритме 166а. (*Прим. ред.*)

рица удовлетворяла условию, поставленному в примечании к алгоритму. Было найдено, что сходимость самая быстрая и программа наиболее практична для матриц с собственными значениями, малыми по сравнению с единицей *.

АЛГОРИТМ 167б

Разделенные разности с повторяющимися точками [E1]

Процедура *divdif* (*divided differences* — разделенные разности) вычисляет нисходящие («прямые») разделенные разности $[x_1, x_2, \ldots, x_n]$ функции $f(x)$, где n последовательно принимает значения 1, 2, 3, ... (см. [7]).

Формальные параметры:

x — вещественный массив размерностью $[1 : n]$. Значения аргумента $x[i]$ не обязательно различны и не обязательно упорядочены, но если массив x уже выбран, то он фиксирует содержание массивов *bb* и *v*. Если $x[1], x[2], \ldots, x[n]$ — монотонная последовательность, то погрешность округления в любой n-й разделенной разности не будет больше той, которая была бы вызвана изменением каждого из значений $f(x[i])$ максимум на n единиц последней значащей цифры. Но если $x[i]$ не образуют монотонную последовательность, то погрешность может быть катастрофической, когда некоторые из разделенных разностей сравнительно велики.

v — вещественный массив размерностью $[1 : n]$, содержащий значения функции $f(x)$ и, возможно, ее производных в точках x_i согласно формуле $v[i] = f^{(m_i)}(x_i)/m_i!$ для $i = 1, 2, 3, \ldots, n$, а m_i — номер повторного появления величины x_i в массиве x [если x_i появляется первый раз, то $v[i] = f^{(0)}(x_i)/0! = f(x_i)$].

bb — вещественный массив размерностью $[1 : n]$, содержащий восходящие («обратные») разделенные разности. Перед обращением к процедуре *divdif* нужно иметь $bb[i] = [x_i, x_{i+1}, \ldots, x_{n-1}]$ для $i = 1, 2, \ldots, n-1$. После выполнения процедуры будут найдены $bb[i] = [x_i, x_{i+1}, \ldots, x_{n-1}, x_n]$ для $i = 1, 2, \ldots, n-1, n$. Если $n = 1$, то начальное значение массива *bb* может быть любым.

m — максимальное значение m_i для $i = 1, 2, \ldots, n$.

Следующий оператор цикла показывает, как процедура *divdif* может быть использована для построения таблицы нисходящих и восходящих разностей:

```
for n := =1 step 1 until nn do
    begin x[n] := =...; v[n] := =...;
      ff[n] := =divdif(n,x,v,m,bb)
    end
```

Массив *ff* может быть использован в процедуре *fnewt*, а массив *bb* — в процедуре *bnewt* (см. алгоритмы 168б и 169б).

```
real procedure divdif(n,x,v,m)dataresult:(bb);
      value n,m; integer n,m; array x,v,bb;
begin real d; integer i,j,nk; array w[1 : m+2];
      if n=1 then
```

---

* По-видимому, в последней фразе Р. Родман имел в виду матрицы *e—a*. (*Прим. ред.*)

```
        begin bb[1] := =v[1]; go to final end;
      nk := =1;
      for i := =1 step 1 until n do
        if x[i] =x[n] then
          begin nk := =nk+1; w[nk] := =v[i] end i;
      for i := =n step —1 until 2 do
        begin w[1] := =bb[i—1]; bb[i] := =w[2];
          j := = if n—i+2<nk then n—i+2 else nk;
          for j := =j step —1 until 2 do
            begin d := =x[n] —x[i+j—3];
              if d≠0 then
                begin w[j] := =(w[j] —w[j—1])/d end else
                begin w[j] := =w[j+1];
                  if nk=j+1 then nk := =j
                end
            end j
        end i;
      bb[1] := =w[2];
final : divdif := =bb[1]
end divdif;
```

Свидетельство к алгоритму 167б

Алгоритм 167б не отличается от алгоритма 167а, если не считать того, что строка букв *result* в заголовке процедуры *divdif* была заменена строкой букв *dataresult*.

Правильность этого алгоритма была также подтверждена и в расчетах на машине БЭСМ-6 [47, с. 131].

Свидетельство к алгоритму 167а

Алгоритм 167а получен в результате исправления, сокращения и ординарной переработки алгоритма 167 (Kahan W., Farkas I. «CACM», 1963, № 4). Кроме того, для удобства пользования алгоритмом массив-параметр w был локализован в теле процедуры, а вместо него был добавлен формальный параметр m. В алгоритме 167 была обнаружена одна опечатка: вместо оператора

```
if NK—j—1≠0 then go Cont;
```

должен быть оператор

```
if NK—j—1≠0 then go to Cont;
```

Алгоритм 167а был транслирован с исходными данными, указанными в «Подтверждении» Г. Тачера, перевод которого публикуется ниже вслед за алгоритмом 169б. Были получены следующие результаты:

$ff[1] = 148.413200$	$bb[1] = 2.07542003$
$ff[2] = 148.413200$	$bb[2] = 11.4904000$
$ff[3] = 106.602400$	$bb[3] = 53.3012000$
$ff[4] = 41.8107999$	$bb[4] = 148.413200$
$ff[5] = 9.41497998$	$bb[5] = 255.015599$
$ff[6] = 2.07542003$	$bb[6] = 403.428800$

Значения *ff* соответствуют верхней диагонали, а значения *bb* — нижней строке табл. 14 в «Подтверждении» Г. Тачера: Некоторые несовпадения (максимальное расхождение имеет место для разности *ff*[5]=9.41497998, которая дана у Г. Тачера равной 9.415191), объясняется, по-видимому, ошибками в результатах Г. Тачера, поскольку в правильности результатов трансляции алгоритма 167а можно легко убедиться ручным счетом.

Кроме того, алгоритм 167а был транслирован для функции $y=x^2$. Результаты этой трансляции даны в «Дополнительном свидетельстве к алгоритмам 167а, 168а и 169а», приводимом ниже вслед за алгоритмом 169б.

АЛГОРИТМ 168б

Интерполяция по Ньютону с разделенными разностями в обратном направлении [E1]

Процедура *bnewt* (*backward* — обратно, *Newton* — Ньютон) находит значения $f(z)$ и $f'(z)$ с помощью интерполяции по таблице с неравным шагом аргумента x, используя восходящие разделенные разности, полученные, например, с помощью алгоритма 167б.

Формальные параметры:

x — вещественный массив размерностью $[1:n]$. Значения аргумента $x[i]$ не обязательно различны и не обязательно упорядочены, но если массив x уже выбран, то он фиксирует содержание массива *bb*.

bb — вещественный массив размерностью $[1:n]$, содержащий восходящие разделенные разности $bb[i]=[x_i, x_{i+1}, \ldots, x_n]$ для $i=1, 2, \ldots, n$ (см. алгоритм 167б и [7]). Если два или более значения $x[i]$ равны между собой, то некоторые из разностей $bb[i]$ должны быть разделенными разностями, соответствующими кратным точкам (см. алгоритм 167б).

p — значение сопутствующего полинома от z степени не выше $n-1$, имеющего вид

$$bb[n]+(z-x_n)\times\{bb[n-1]+(z-x_{n-1})\times$$
$$\times\{bb[n-2]+\ldots+(z-x_2)\times bb[1]\}\ldots\}.$$

Этот полином является интерполяционным полиномом, который (с точностью до ошибок округления) соответствует значениям функции $f(x)$ и любым ее производным, задаваемым процедуре *divdif* (алгоритм 167б), т. е. $p=f(z)$.

d — значение производной от предыдущего полинома, т. е. $d=f'(z)$.

e — максимальная погрешность в значении *p*, вызываемая округлением при выполнении процедуры *bnewt*. Оценка погрешности *e* основывается на предположении, что результат каждой арифметической операции с плавающей запятой подвергается усечению после 27-й значащей двоичной цифры, как это имеет место для ФОРТРАН-программы на машине IBM-7090.

```
procedure bnewt(z,n,x,bb)result:(p,d,e);
    value z,n; real z,p,d,e; integer n; array x,bb;
begin real z1; integer i;
    p:=d:=e:=0;
```

```
    for i := 1 step 1 until n do
      begin z1 := z—x[i]; d := p+z1×d;
        p := bb[i]+z1×p; e := abs(p)+e×abs(z1)
      end i;
    e := (1.5×e—abs(p))×3₁₀—8
end bnewt;
```

Свидетельство к алгоритму 168а *

Алгоритм 168а получен в результате ординарной переработки алгоритма 168 (Kahan W., Farkas I. «CACM», 1963, № 4).

Перевод «Подтверждения к алгоритмам 167, 168, 169» Г. Тачера приводится ниже вслед за алгоритмом 169б.

Алгоритм 168а был транслирован с исходными данными, использованными в приводимом ниже «Подтверждении» Г. Тачера, причем в качестве массива *bb* были взяты результаты решения по алгоритму 167а **.

Результаты трансляции алгоритма 168а даны в табл. 11.

Таблица 11

z	p	d	e
5.0	148.413200	148.413200	$\sim 0.137049000\times10^{-4}$
5.5	244.697343	244.692613	$\sim 0.125211780\times10^{-4}$
6.0	403.428800	403.428800	$\sim 0.605143200\times10^{-5}$

Значения p и d этой таблицы хорошо совпадают с соответствующими значениями табл. 16 в «Подтверждении» Г. Тачера.

Кроме того, были проведены расчеты для функции $y=x^2$, результаты которых даны в «Дополнительном свидетельстве к алгоритмам 167а, 168а и 169а», приводимом ниже вслед за алгоритмом 169б.

АЛГОРИТМ 169б

Интерполяция по Ньютону с разделенными разностями в прямом направлении [E1]

Процедура *fnewt* (*forward* — вперед, *Newton* — Ньютон) находит значения $f(z)$ и $f'(z)$ с помощью интерполяции по таблице с неравным шагом аргумента x, используя нисходящие разделенные разности, которые можно получить, например, с помощью алгоритма 167б.

Формальные параметры:

x — вещественный массив размерностью $[1:n]$. Значения аргумента x_i не обязательно различны и не обязательно упорядочены, но если массив x уже выбран, то он фиксирует содержание массива ff.

ff — вещественный массив размерностью $[1:n]$, содержащий нисходящие разделенные разности $ff[i]=[x_1, x_2, \ldots, x_n]$ для $i=1, 2, \ldots, n$ (см. алгоритм 167б и [7]). Если два или более значения $x[i]$ равны между собой, то некоторые из разностей $ff[i]$ должны быть раз-

* См. также подтверждение в [47, с. 131]. (*Прим. ред.*)

** Эти результаты не отличаются от результатов решения по алгоритму 167б. (*Прим. ред.*)

деленными разностями, соответствующими кратным точкам (см. алгоритм 167б).

r — значение сопутствующего полинома от z степени не выше $n-1$, имеющего вид

$$ff[1]+(z-x_1)\times\{ff[2]+(z-x_2)\times$$
$$\times\{ff[3]+\ldots+(z-x_n)\times ff[n]\}\ldots\}.$$

Этот полином является интерполирующим полиномом, который (с точностью до ошибок округления) соответствует значениям функции $f(x)$ и любым ее производным, задаваемым процедуре *divdif* (алгоритм 167б), т. е. $r=f(z)$.

d — значение производной от предыдущего полинома, т. е. $d=f'(z)$.

e — максимальная погрешность в значении *r*, вызываемая округлением при выполнении процедуры *fnewt*. Оценка погрешности основывается на предположении, что результат каждой арифметической операции с плавающей запятой подвергается усечению после 27-й значащей цифры, как это имеет место для ФОРТРАН-программ на машине IBM-7090.

```
procedure fnewt(z,n,x,ff)result:(r,d,e);
    value z,n; real z,r,d,e; integer n; array x,ff;
begin real z1; integer i;
    r:=d:=e:=0;
    for i:=n step -1 until 1 do
      begin z1:=z-x[i]; d:=r+z1×d;
        r:=ff[i]+z1×r; e:=abs[r]+abs(z1)×e
      end i;
    e:=(1.5×e-abs(r))×3₁₀-8
end fnewt;
```

Свидетельство к алгоритму 169а*

Алгоритм 169а получен в результате ординарной переработки алгоритма 169 (Kahan W., Farkas I. «CACM», 1963, № 4).

Алгоритм 169а был транслирован с исходными данными, использованными в приводимом ниже «Подтверждении» Г. Тачера, причем в качестве массива *ff* были взяты результаты, выданные для этих же точек алгоритмом 167а. Результат трансляции алгоритма 169а приведен в табл. 12.

Таблица 12

z	*r*	*d*	*e*
5.0	148.413200	148.413200	0.222619800×10^{-5}
5.5	244.697343	244.692613	0.923484033×10^{-5}
6.0	403.428800	403.428800	0.223242420×10^{-4}

Значения *r* и *d* в этой таблице хорошо совпадают с соответствующими значениями табл. 15 «Подтверждения к алгоритмам 167, 168, 169» Г. Тачера, перевод которого приводится ниже.

* См. также подтверждение в [47, с. 131]. (Прим. *ред.*)

Кроме того, были проведены расчеты для функции $y=x^2$, результаты которых даны в «Дополнительном свидетельстве к алгоритмам 167а, 168а и 169а», приводимом ниже.

Дополнительное свидетельство к алгоритмам 167а, 168а и 169а

Кроме тех контрольных решений, результаты которых были приведены в соответствующих свидетельствах к алгоритмам 167а, 168а и 169а, по этим алгоритмам были проведены решения для случая интерполирования функции $y=x^2$ в следующих вариантах значений аргумента.

I. В алгоритме 167а задавались параметры

$$x=(0,1,3,6),\ v=(0,1,9,36),\ n=4,\ m=0.$$

Результаты его трансляций с этими параметрами приведены в табл. 13.

Таблица 13

i	1	2	3	4
ff[i]	0	1	1	0
bb[i]	0	1	9	36

Результаты трансляции алгоритмов 168а и 169а с этими разностями приведены в табл. 14.

Таблица 14

z	Алгоритм 168а			Алгоритм 169а		
	p	d	e	r	d	e
1	1	2	$\sim 0.204\times10^{-5}$	1	2	$\sim 0.60\times10^{-7}$
2	4	4	$\sim 0.168\times10^{-5}$	4	4	$\sim 0.33\times10^{-6}$
5	25	10	$\sim 0.96\times10^{-6}$	25	10	$\sim 0.24\times10^{-5}$
6	36	12	$\sim 0.54\times10^{-6}$	36	12	$\sim 0.351\times10^{-5}$

Очевидно, что эти результаты совершенно точные.

II. В алгоритме 167а задавались параметры

$$x=(0,1,3,3,6,6,1,3),\ v=(0,1,9,6,36,12,2,1),\ n=8,\ m=2.$$

Результаты приведены в табл. 15.

Таблица 15

i	1	2	3	4	5	6	7	8
ff [i]	0	1	1	0	0	0	0	0
bb [i]	0	0	0	0	0	1	12	36

Результаты трансляции алгоритмов 168а и 169а с этими разностями приведены в табл. 16.

Таблица 16

z	Алгоритм 168а			Алгоритм 169а		
	p	d	e	r	d	e
1	1	2	$\sim 0.27\times10^{-5}$	1	2	$\sim 0.60\times10^{-7}$
2	4	4	$\sim 0.22\times10^{-5}$	4	4	$\sim 0.33\times10^{-6}$
5	25	10	$\sim 0.92\times10^{-6}$	25	10	$\sim 0.24\times10^{-5}$
6	36	12	$\sim 0.54\times10^{-6}$	36	12	$\sim 0.35\times10^{-5}$

Подтверждение к алгоритмам 167, 168, 169

Г. Тачер (Thacher H. C., «CACM», 1963, № 9)

Эти процедуры были проверены на машине LGP-30 с дартмутским транслятором SCALP. При трансляции и решении не было обнаружено никаких синтаксических или математических ошибок.

Нужно отметить, что хотя в алгоритме 169 уменьшение значения n по сравнению с тем, которое используется для образования ff, приводит к интерполяционному полиному, базирующемуся на меньшем числе точек, это не имеет места в алгоритме 168. Гибкость алгоритма можно повысить путем введения дополнительного формального параметра, например, *deg*, и путем замены заголовка цикла на заголовок

for i := =nn—deg **step** 1 **until** nn **do**

Логика оценки погрешности в алгоритмах 168 и 169 не вполне ясна. Однако представляется, что оценка может быть отрегулирована для различных точностей используемых арифметик путем соответствующего подбора константы 3_{10}—8. Для арифметики SCALP эта константа была заменена на 1_{10}—7.

Таблица 17

n	x [n]	v [n]	bb [n]	bb [n—1]	bb [n—2]	bb [n—3]	bb [n—4]	bb [n—5]
1	5.0	148.4132	148.4132					
2	5.0	148.4132	148.4132	148.4132				
3	6.0	403.4288	403.4287	255.0155	106.6023			
4	6.0	403.4288	403.4287	403.4287	148.4132	41.81091		
5	5.0	74.20658	148.4132	255.0155	148.4132	41.81091	9.415191	
6	6.0	201.7144	403.4287	255.0151	148.4132	53.30115	11.49023	2.075043

Алгоритмы были проверены на примерах, данных Милном — Томсоном [3i] и Милном [4i]. В обоих примерах алгоритм 167 выдавал таблицу разделенных разностей, а алгоритмы 168 и 169 использовали их как входные значения. Для проверки вычисления разделенных разностей с кратными точками были взяты значения первых двух производных экспоненциальной функции при x=5.0 и 6.0. Были получены разности, приведенные в табл. 17.

Таблица 18

z	bnewt			fnewt		
	p	d	e	r	d	e
5.000000	148.4132	148.4132	0.4567298×10^{-4}	148.4132	148.4132	0.7420658×10^{-5}
5.500000	244.6973	244.6924	0.4173722×10^{-4}	244.6973	244.6924	0.3078276×10^{-4}
6.000000	403.4287	403.4287	0.2017143×10^{-4}	403.4287	403.4287	0.7441404×10^{-4}

Нисходящие разности лежат на верхней диагонали. Использование этих разностей в процедурах *bnewt* и *fnewt* дало для $nn=6$ результаты, приведенные в табл. 18.

АЛГОРИТМ 170б

Определитель с полиномиальными элементами [F3]

Процедура *polymatrix* (*polynomial* — полином, *matrix* — матрица) раскрывает определитель общего вида, в котором каждый элемент является полиномом.

Эта программа полезна для исследования задач динамической устойчивости при использовании аппроксимации преобразующей функции. Здесь сначала выполняется один из процессов триангуляризации полиномиальной матрицы с вещественными коэффициентами, затем умножением диагональных элементов формируется детерминантный полином. Указанная здесь полиномиальная матрица имеет в качестве элементов полиномы вида $\sum_{i=0}^{n} a_i x^i$. После триангуляризации все элементы ниже главной диагонали равны нулю. Далее при раскрытии определителя там формируются ненулевые члены. В результате можно проверять, например, критерий устойчивости путем вычисления корней сформированного таким образом характеристического уравнения, пользуясь какой-нибудь подходящей программой нахождения корней.

Рассмотрим для примера полиномиальную матрицу с квадратичными элементами ($n=2$). В этом случае массив a должен иметь размерность $[1:p, 1:p, 1:m]$, где p — порядок матрицы, а $m=n\times p+1$. Здесь первый индекс соответствует строке, второй — столбцу, а третий — коэффициенту полинома.

Следовательно, перед обращением к процедуре постоянный член главного полиномиального элемента содержится в $a[i,j,1]$, линейный — в $a[i,j,2]$, а квадратичный — в $a[i,j,3]$ и т. д.

После выполнения программы коэффициенты детерминантного полинома содержатся в массиве $c[1:m]$. Постоянный коэффициент будет в $c[1]$, линейный — в $c[2]$, а квадратичный — в $c[3]$ и т. д. Переменная r будет равна числу коэффициентов детерминантного полинома. В общем случае $r\neq m$, так как при раскрытии определителя некоторые коэффициенты могут обратиться в нуль, но при обращении к процедуре нужно задавать $r=m$. Если полиномы, представляющие собой элементы матрицы, не все одного и того же порядка, то перед обращением к процедуре нужно брать n равным наивысшей из степеней полиномов.

Значение *eps* можно задавать, например, равным 2_{10}—8.

В некоторых случаях неправильного хода решения осуществляется выход из процедуры к глобальной метке *signal* 170.

```
procedure polymatrix(a,p,n,eps)dataresult:(r)result:(c);
        value p,n; integer p,n,r; array a,c; real eps;
begin real sa,sb;
        integer i,j,k,j1,j2,j3,j4,j5,j6,j7,j8,j9,j10,j11,m;
        array c1,c2[1 : n×p+1]; integer array mat[1 : p, 1 : p];
start : m : =n×p+1;
```

```
      for i := =1 step 1 until p do
        for j := =1 step 1 until p do
          begin j1 := =0;
            for k := =1 step 1 until m do
            if a[i,j,k] ≠0 then j1 := =k;
            mat[i,j] := =j1
        end i;
      j1 := =1;
n0:   j9 := =0;
      for i := =j1 step 1 until p do
        begin
          if mat[i,j1] <0 then go to signal 170;
          if mat[i,j1] >0 then
            begin j9 := =j9+1; j3 := =i end
        end i;
      if j9<1 then go to signal 170;
      if j9>1 then go to n2;
      if j3<j1 then go to signal 170;
      if j3≠j1 then
        for j := =j1 step 1 until p do
          begin j2 := =j4 := =mat[j3,j];
            if mat[j1,j] >j2 then j2 := =mat[j1,j];
            mat[j3,j] := =mat[j1,j]; mat[j1,j] := =j4;
            for k := =1 step 1 until j2 do
              begin sa := =a[j3,j,k]; a[j3,j,k] := =a[j1,j,k];
                a[j1,j,k] := =—sa
              end k
          end j;
      go to n12;
n2:   j3 := =j1+1;
      for i := =j3 step 1 until p do
        begin
n3:       if mat[i,j1] <0 then go to signal 170;
          if mat[i,j1] =0 then go to n11;
          if mat[j1,j1] <0 then go to signal 170;
          if mat[j1,j1] ≠0∧mat[i,j1] ≥mat[j1,j1] then go to n5;
n4:       for j := =j1 step 1 until p do
            begin j2 := =j4 := =mat[j1,j];
              if mat[i,j] >j2 then j2 := =mat[i,j];
              mat[j1,j] := =mat[i,j]; mat[i,j] := =j4;
              for k := =1 step 1 until j2 do
                begin sa := =a[i,j,k];
                  a[i,j,k] := =a[j1,j,k]; a[j1,j,k] := =—sa
                end k
            end j;
          go to n3;
          comment Перестановка строк i и j1;
n5:       j7 := =mat[i,j1]; j5 := =mat[j1,j1];
          j6 := =j7—j5; sb := =a[i,j1,j7] /a[j1,j1,j5];
          if abs(sb) <4 then go to n6;
          if j6<0 then go to signal 170;
          if j6=0 then go to n4;
```

```
n6:         for j := j1 step 1 until p do
              begin j5 := mat[j1,j];
                for k := 1 step 1 until j5 do
                  begin j7 := k+j6;
                    if j7>m then go to n10;
                  sa := a[i,j,j7]-sb×a[j1,j,k];
                  a[i,j,j7] := if abs(sa)<eps then 0 else sa
                  end k
              end j;
n10:        for j := j1 step 1 until p do
              begin j7 := mat[j1,j]+j6;
                if mat[i,j] ≥ j7 then j7 := mat[i,j];
                mat[i,j] := 0;
                for k := 1 step 1 until j7 do
                  if a[i,j,k] ≠ 0 then mat[i,j] := k
              end j;
n11:      end i;
        go to n0;
n12:    j1 := j1+1;
        if j1<p then go to n0;
        for j := 1 step 1 until p do
          begin j2 := mat[j,j];
            for k := 1 step 1 until j2 do c1[k] := a[j,j,k];
            if j=1 then go to n14;
            for k := 1 step 1 until r do c2[k] := c[k];
            for k := 1 step 1 until m do c[k] := 0;
            if j2<0 then go to signal 170;
            if j2=0 then go to n15;
            for k := 1 step 1 until j2 do
              for j10 := 1 step 1 until r do
                begin j11 := k+j10-1;
                  c[j11] := c[j11]+c1[k]×c2[j10]
                end k;
            r := j11;
            go to n15;
n14:        for k := 1 step 1 until j2 do c[k] := c1[k];
            r := j2;
n15:      end j
end polymatrix;
```

Свидетельство к алгоритму 170б

Алгоритм 170б получен из алгоритма 170а в результате внесения в него поправок, указанных в нижеприведенном «Замечании к алгоритму 170а» В. Д. Сычева и Ю. И. Маркова, а также в результате некоторых его улучшений. В частности, произведены преобразования ряда операторов между метками *n*6 и *n*10, а также между метками *n*10 и *n*11, имеющие целью ускорить выполнение алгоритма. Кроме того, была удалена метка *exit* перед последним **end** тела процедуры и все операторы **go to** *exit* были заменены операторами **go to** *signal*170 с целью обеспечения возможности контроля всех аварийных выходов из процедуры. Наконец, константа $2_{10}-8$ была заменена дополнительным формальным параметром *eps* с целью обеспечения возможности варьи-

рования этой константы без внесения каких-либо изменений в тело процедуры.

Алгоритм 170б был транслирован на машине ICL-4-70, и с его помощью был правильно решен пример, приведенный в вышеупомянутом «Замечании к алгоритму 170а». Кроме того, был получен правильный результат для определителя

$$\begin{vmatrix} 1+x+x^2+x^3+x^4 & 3-2x-x^3 \\ -1+x & 1 \end{vmatrix} = 4-4x+3x^2+2x^4.$$

При этом задавались параметры: $p=2$, $n=4$, $r=m=9$, $eps=2_{10}-8$, $a=(1,1,1,1,1,0,0,0,0,0;\ 3,-2,0,-1,0,0,0,0,0;\ -1,1,0,0,0,0,0,0,0;\ 1,0,0,0,0,0,0,0,0)$.

Результат имел вид $c=(4,-4,3,0,2,0,0,0,0)$.

Замечание к алгоритму 170а

В. Д. Сычев, Ю. И. Марков, Москва, сентябрь 1971

В процедуре *polymatrix* [27] необходимы две поправки.

1. На с. 63, стр. 4—6 сверху

```
begin j3 : =i;
  if mat[i,j1] <0 then go to exit;
  if mat[i,j1] >0 then j9 : =j9+1
```

должны быть заменены на строки

```
begin
  if mat[i,j1] <0 then go to exit;
  if mat[i,j1] >0 then
    begin j9 : =j9+1; j3 : =i end
```

2. На с. 63, стр. 2 снизу

```
if mat[j1,j1] ≠0∧mat[i,j1] <mat[j1,j1] then
```

должна быть заменена строкой

```
if mat[j1,j1] ≠0∧mat[i,j1] ⩾mat[j1,j1] then
```

Исправленный вариант этой процедуры использовался в многочисленных расчетах и всегда давал верные результаты. В частности, был решен пример для

$$\begin{vmatrix} 1 & 1-x \\ 1+x & 1 \end{vmatrix} = x^2,$$

т. е. для $p=2$, $n=1$, $m=r=3$ и

$$a = \left\| \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \right\|, \quad \left\| \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix} \right\|, \quad \left\| \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} \right\|.$$

Получен результат $c=(0,0,1)$. В этом случае массив a формировался следующим образом: $a_{111}=a_{121}=a_{211}=a_{212}=a_{221}=1$, $a_{112}=a_{113}=a_{123}=a_{213}=a_{222}=a_{223}=0$, $a_{122}=-1$.

Свидетельство к алгоритму 170а

Алгоритм 170а получен в результате сокращения и ординарной переработки алгоритма 170 (Hennion P. E. «CACM», 1963, № 4), а также внесения в него поправок, указанных в «Замечании к алгоритму 170» (Hennion P. E. «CACM», 1963, № 8) и в «Подтверждении к алгоритму 170» (Priebe K. B. «CACM», 1964, № 7). Кроме того, была исправлена опечатка во второй строке после метки «L4:», заключающаяся в использовании круглой скобки вместо индексной.

«Замечание» П. Хеньона здесь не публикуется, потому что, кроме перечисления ошибок в алгоритме 170, оно не содержит другой информации.

Подтверждение к алгоритму 170

К. Прайб (P r i e b e K. B. «CACM», 1964, № 7)

Алгоритм 170 был переведен на язык FAST для машины NCR/315 и дал удовлетворительные результаты после следующих исправлений *.

Свидетельство к алгоритму 171б [Z]

Поскольку в журнале «CACM» под номером 171 не было вообще опубликовано никакого алгоритма, то в данном выпуске под номером 171б публикуется новый алгоритм шахматного программирования. Этот алгоритм вынесен в приложение 1 по причине его особого тематического содержания и весьма большого объема.

АЛГОРИТМ 172б

Интерполяция табличной функции нескольких переменных (рекурсивная процедура) [E1]

Процедура-функция *inpol* (*interpolation* — интерполяция и *polynomial* — полином) вычисляет значение функции нескольких переменных с помощью полиномиальной интерполяции по таблице опорных значений. Опорные значения могут быть заданы в узлах многомерной прямоугольной сетки с произвольными интервалами. Интерполяция осуществляется с помощью процесса Невиля (N e v i l l), повторяющегося по каждому измерению.

Первое значение массива $t[0]=d$ — это число независимых переменных (число измерений); оно всегда должно быть целым (хотя и типа **real**), в противном случае будет выбираться его целая часть. Специальный случай $d=0$ означает, что табулированная функция постоянна, ее значение равно $t[1]$. Это же значение берется и в случае $d<0$. Элементы массива $t[1]$, $t[2]$, ..., $t[d]$ являются количествами значений независимых переменных $x1$, $x2$, ..., xd соответственно и, следовательно, должны быть целыми. Далее помещаются $t[1]$ значений переменной $x1$, $t[2]$ значений переменной $x2$, ... и $t[d]$ значений переменной xd. Все значения каждой из этих независимых переменных должны быть различными и составлять монотонную последовательность. В конце помещаются $t[1]\times t[2]\times\ldots\times t[d]$ значений зависимой переменной (т. е. функции) $f(x1, x2, \ldots, xd)$, расположенных в порядке: $t[d]$ раз по $t[d-1]$ раз по ... $t[2]$ раз по $t[1]$ значений f. Таблица представляется одномерным массивом, потому что использовать d-мерный массив, не зная заранее значение d, невозможно.

$x[i]$ — координаты ($i=1, 2, \ldots, d$) точки, в которой путем интерполирования нужно найти значение функции.

$n[i]$ — количество табулированных значений переменной $x[i]$, которые желательно использовать для интерполяции по i-му измерению.

---

* Далее указываются пять поправок к алгоритму 170, после чего следует замечание редактора Г. Форсайта, повторяющее поправки П. Хеньона (Hennion). (*Прим. ред.*)

Искомое значение интерполируемой функции присваивается указателю функции *inpol*. Фактический параметр, соответствующий параметру *expol*, должен быть выражением, значение которого присваивается указателю функции *inpol*, когда какое-либо из значений $x[i]$ окажется вне области, покрываемой массивом t. В этом случае параметр *xout* является частным значением этого $x[i]$. Переменные i, *out* и *xout* введены в список формальных параметров процедуры *inpol* для того, чтобы их можно было использовать в фактическом параметре, соответствующем формальному параметру *expol*.

Пусть, например, обращение к процедуре *inpol* записано следующим образом:

```
z := inpol(a,x,k,n,out,y,
  if k=1 then extrapolate(a,1,n,out,y) else
    if k=2 then limtab(a,2,out,y) else y-2)
```

Если $x[1]$ находится вне области, покрываемой массивом a, то этот оператор для получения значения функции *inpol* будет использовать экстраполяционную процедуру *extrapolate*, описание которой приведено ниже. Если вне области находится $x[2]$, то произойдет замена данного значения $x[2]$ его значением на ближайшем конце таблицы путем использования процедуры *limtab* (см. ниже). Если вне области окажется любая другая переменная, например $x[3]$, то значение *inpol* будет взято равным $x[3]-2$.

В общем случае предполагается, что выражение *expol* будет содержать один или несколько указателей функции, а также критерий для выбора между ними, как это показано в вышеприведенном примере. Выражение *expol* может включать в себя, например, любую из следующих альтернатив (в первой и в третьей из них важны побочные эффекты, а значение, присваиваемое выражению *expol*, в соответствии с разд. 5.4.4 пересмотренного сообщения по языку АЛГОЛ-60 [1] может быть только пустым).

1. Выражение *expol* может быть указателем функции, который для экстраполяции использует интерполяционную формулу путем возврата к процедуре *inpol* и выполнения оператора *out* := **false**. В формуле интерполяции используются последние $n[i]$ значений переменной $x[i]$, но выражение *expol* вместо этого может использовать первые $n[i]$ значений (что обычно более предпочтительно, если $x[i]$ находится за нижней границей таблицы) путем выполнения оператора $t[0] := n[i]-t[i]$ (в котором значение локальной переменной $n[i]$ предназначено для использования тогда, когда оно отличается от значения нелокальной переменной $n[i]$). Для этой цели может применяться процедура *extrapolate*, приведенная ниже.

2. Выражение *expol* может использовать какую-либо другую формулу экстраполяции, после вычисления которой управление может быть снова передано процедуре *inpol* без изменения значения логической переменной *out*. Если это все, что требуется, то фактический параметр, соответствующий параметру *expol*, может быть обычным арифметическим выражением, не содержащим указателя функции.

3. Выражение *expol* может быть указателем функции, который ограничивает значение $x[i]$ так, чтобы оно находилось внутри интервала, заменяя его тем значением i-й переменной, которое она имеет на ближайшей границе таблицы (или каким-либо другим значением). Для этого выражение *expol* должно оперировать со значением *xout*, а не

непосредственно с *x*[*i*]. Глобальный массив *x* при этом не должен затрагиваться. Перед возвращением в *inpol* при вычислении выражения *expol* должен выполняться оператор *out* := **false**. Для этой цели можно пользоваться процедурой *limtab*, приведенной ниже.

4. Выражение *expol* может требовать выполнения чего-либо другого и содержать программу без возвращения в *inpol* (например, отсылая к глобальной метке с помощью оператора перехода). Это должно рассматриваться как аварийный выход, поскольку значение *inpol* будет не определено (см. разд. 5.4.4 пересмотренного сообщения по языку АЛГОЛ-60 [1]).

Если на выходе из процедуры параметр *out* ≡ **true**, то это означает, что встречалась экстраполяция. Обратное утверждение не обязательно верно, поскольку значение *out* зависит от вида фактического параметра, соответствующего формальному параметру *expol*.

В теле процедуры *inpol* локализована процедура *oneway* (*one* — один, *way* — путь), которая выполняет интерполяцию по одному измерению.

Кроме того, в теле процедуры *inpol* используются две глобальные процедуры *fors*3 и *nev*, описания которых приведены ниже вслед за описанием процедуры *inpol*.

```
real procedure inpol(t,x,i,n,out,xout,expol);
      value x,n; real xout,expol; integer i;
      Boolean out;
      array t,x; integer array n;
begin integer d,j,k,r,m,q,xi;
      d := =entier(t[0]);
      if d<1 then inpol := =t[1] else
        begin xi := =1;
          for i := =1 step 1 until d do
            begin
              if n[i]<2 then n[i] := =2;
              if n[i]>t[i] then n[i] := =t[i];
              xi := =xi+n[i]
            end i;
          begin array f[1 : xi—n[1]];
            integer array v,xinit,yinc[1: d];
            procedure oneway;
              begin f[v[1]] := =nev(x[1],t,xinit[1],t,q,r);
                i := =1; m := =0;
                for k := =1 step 1 until d—1 do
                  begin q := =q+yinc[k];
                    if v[k]≠n[k] then go to continue;
                    m := =m+n[k];
                    f[m+v[k+1]] :=
                      nev(x[k+1],t,xinit[k+1],f,i,n[k]);
                    i := =i+n[k]
                  end k;
continue:       end oneway;
            q := =xi := =d+1; m := =1;
            for i := =1 step 1 until d do
              begin k := =xi+t[i]—1;
                out := =(x[i]—t[xi])×(x[i]—t[k])>0;
```

```
                if out then
                  begin xout:=x[i]; inpol:=expol;
                    x[i]:=xout;
                    if t[0]≤0 then
                      begin k:=k+t[0]; t[0]:=d end
                  end;
                if out then go to bb;
                j:=xi;
aa:             r:=(j+k)÷2;
                if (x[i]-t[j])×(x[i]-t[r])>0 then
                  j:=r else k:=r;
                if k-j>1 then go to aa;
                comment Поиск x[i] в таблице;
                r:=k-n[i]÷2;
                if r≤xi then r:=xi else
                  begin k:=xi+t[i]-n[i];
                    if r>k then r:=k
                  end корректировки вблизи края таблицы;
                q:=q+t[i]+(r-xi)×m;
                xinit[i]:=r; xi:=xi+t[i];
                yinc[i]:=m×(t[i]-(if i=1 then 0 else n[i]));
                m:=m×t[i]
              end i;
            v[d]:=1; r:=n[1];
            for i:=1 step 1 until d-1 do n[i]:=n[i+1];
            fors3(d-1,oneway,v,n);
            inpol:=f[m+1]
          end блока с массивом f
        end if d<1;
bb:   end inpol;
```

Процедура fors 3 *

Процедура *fors*3 (сокращение от *for statement* — оператор цикла) заменяет (свертывает) «*n*-слойный» оператор цикла, подоператором которого является обращение к процедуре *p* (известно, что любой оператор цикла может быть приведен к такому виду). Два массива *v* и *ub* дают значение параметра цикла и его верхней границы (соответственно) для каждого уровня. Размерность этих массивов *v*,*ub*[1 : *n*].

```
procedure fors3(n,p,v,ub);
      value n; integer n; integer array v,ub;
      procedure p;
begin integer j;
      if n=0 then p else
        for j:=1 step 1 until ub[n] do
          begin v[n]:=j; fors3(n-1,p,v,ub) end
end fors3;
```

Процедура nev

Процедура-функция *nev* (*Neville* — Невилль) осуществляет одномерную интерполяцию по Невиллю. Для интерполяции используются

---

* Приведенная здесь процедура *fors* 3 не отличается по существу от процедуры *fors* 1 алгоритма 1376 [49]. (*Прим. ред.*)

n значений независимой переменной, а именно, *n* последовательных элементов массива *ax*, начиная с индекса *sax*. Соответствующими значениями зависимой переменной являются *n* последовательных элементов массива *ay*, начиная с индекса *say*. Параметр *x* — это значение независимой переменной, для которого отыскивается путем интерполяции значение зависимой переменной (значение указателя функции *nev*).

```
real procedure nev(x,ax,sax,ay,say,n);
      value x,sax,say,n; real x; integer sax,say,n;
      array ax,ay;
begin integer i,j,nj,ki; array f[0:n—1];
      for j:=0 step 1 until n—1 do f[j]:=ay[say+j];
      for j:=1 step 1 until n—1 do
        begin nj:=n—j—1;
          for i:=0 step 1 until nj do
            begin ki:=sax+i;
              f[i]:=(f[i+1]—f[i])×(x—ax[ki])
                /(ax[ki+j]—ax[ki])+f[i]
          end i
        end j;
      nev:=f[0]
end nev;
```

Процедура extrapolate

Процедура-функция *extrapolate* предназначена для использования в фактическом параметре, соответствующем формальному параметру *expol* обращения к процедуре *inpol*. Параметры процедуры *extrapolate* имеют тот же смысл, что и параметры процедуры *inpol*.

Процедура *extrapolate* использует для экстраполяции по *i*-й переменной интерполяционную формулу. Если точка интерполяции находится за нижней границей таблицы, то процедура *extrapolate* берет первые *n*[*i*] значений *i*-й переменной вместо последних *n*[*i*] ее значений.

```
real procedure extrapolate(t,i,n,out,xout);
      real xout; integer i; Boolean out; array t;
      integer array n;
begin integer j,k;
      out:=false; extrapolate:=0; j:=1;
      for k:=0 step 1 until i—1 do j:=j+t[k];
      if t[i]=1 then xout:=t[j] else
      if abs(xout—t[j])<abs(xout—t[j+t[i]—1]) then
        begin k:=n[i];
          if k<2 then k:=2;
          if k>t[i] then k:=t[i];
          t[0]:=k—t[i]
        end
end extrapolate;
```

Процедура limtab

Процедура-функция *limtab* (*limit* — граница, край и *table* — таблица) предназначена для использования в фактическом параметре, соответствующем формальному параметру *expol* обращения к процедуре

inpol. Параметры процедуры *limtab* имеют тот же смысл, что и параметры процедуры *inpol*.

Процедура *limtab* заменяет значение *xout*, оказавшееся вне пределов таблицы, значением *i*-й переменной на ближайшем краю таблицы.

```
real procedure limtab(t,i,out,xout);
    real xout; integer i; Boolean out; array t;
begin integer j,k;
    j:=1;
    for k:=0 step 1 until i-1 do j:=j+t[k];
    k:=j+t[i]-1;
    limtab:=xout:=
      if abs(xout-t[j])>abs(xout-t[k]) then t[k] else t[j];
    out:=false
end limtab;
```

Автор алгоритма 264 указывает, что процедуры *inpol*, *extrapolate* и *limtab* были проверены на вычислительной машине ICT Atlas. Они были также проверены на машине National-Elliott 803 после внесения в них изменений, требуемых ограничениями входного языка АЛГОЛ-транслятора 803. Тесты брались для $d=0$, 1, 2 и 3 и включали все специальные случаи.

Свидетельство к алгоритму 172б

Алгоритм 172б получен в результате ординарной переработки, исправления, сокращения и модификации алгоритма 264 (Stafford J. «CACM», 1965, № 10). Модификация заключалась в том, что процедуры FORS3 и NEV, локализованные в теле процедуры INPOL алгоритма 264, были заменены глобальными процедурами *fors*3 и *nev* алгоритма 172б из следующих соображений.

1. Может потребоваться использование процедур *fors*3 и *nev* не только в теле процедуры *inpol*, но и для других целей.

2. Процедура *inpol* с глобальными процедурами *fors*3 и *nev* обычно транслируется более экономно, чем с локальными процедурами.

3. Описание процедуры *inpol* с глобальными процедурами *fors*3 и *nev* записывается более кратко и становится более обозримым и более удобным для перфорации и ввода в память машины.

Исправления состояли в следующем.

1. В процедуре ONEWAY алгоритма 264 была замечена опечатка. Вместо

```
for K:=1 step 1 until D-1 do
```

должно быть

```
for K:=1 step 1 until D-1 do
```

2. В комментарии к процедуре ONEWAY вместо N[2] должно быть N[1].

3. В теле процедуры NEV перед последним оператором

```
NEV:=F[0]
```

должна быть еще одна закрывающая операторная скобка

```
end;
```

Алгоритм 172б был транслирован на машине БЭСМ-6, и были получены верные результаты при интерполировании функций $y=x1^2$ и

Входные параметры				Результат	
Функция	*x1*, *x2*	*n*	*expol*	inpol	out
$y=x1^2$	0.5	2	*y*	0.5	**false**
	0.5	3	*y*	0.25	**false**
	4	3	*extrapolate* (*t*, 1, *n*, *out*, *y*)	16	**false**
	4	3	*limtab* (*t*, 1, *out*, *y*)	9	**false**
	4	3	*y*+6	10	**true**
	—1	3	*extrapolate* (*t*, 1, *n*, *out*, *y*)	1	**false**
$y=x1^2+x2^2$	0.5	2	*y*	1	**false**
	0.5	3	*y*	0.5	**false**
	4	3	*extrapolate* (*t*, 2, *n*, *out*, *y*)	32	**false**
	4	3	*limtab* (*t*, 2, *out*, *y*)	18	**false**
	4	3	*y*+6	10	**true**
	—1	3	*extrapolate* (*t*, 2, *n*, *out*, *y*)	2	**false**

$y=x1^2+x2^2$ для опорных значений аргументов $x1=x2=0,1,2,3$. Результаты трансляции приведены в табл. 19.

Массив *t* задавался следующим образом:

для $y=x1^2$ $t=(1,4,0,1,2,3,0,1,4,9)$;

для $y=x1^2+x2^2$ $t=(2,4,4,0,1,2,3,0,1,2,3,0,1,4,9,1,2,5,10,4,5,8,13,9,10,13,18)$.

В последнем массиве:

$t[0]=d=2$ — число независимых переменных;

$t[1]=4$ и $t[2]=4$ — число значений независимых переменных *x1* и *x2*;

$t[3]$, $t[4]$, $t[5]$, $t[6]$ — значения независимой переменной *x1*;

$t[7]$, $t[8]$, $t[9]$, $t[10]$ — значения независимой переменной *x2*;

$t[11]$, ..., $t[26]$ — 16 значений (так как $t[1]\times t[2]=16$) функции $f(x1, x2)$, расположенных в порядке $f(0,0)$, $f(0,1)$, $f(0,2)$, $f(0,3)$, $f(1,0)$, $f(1,1)$, ..., $f(3,3)$.

Номер алгоритма 264 был заменен на номер 172б, потому что в журнале «CACM» алгоритмы с номерами 171 и 172 не были опубликованы вообще, а под номером 264 были опубликованы два разных алгоритма.

АЛГОРИТМ 173б

Присваивание значений массивам разной размерности (рекурсивная процедура) [K2]

Очевидным применением процедуры *assign* (*assignment* — присваивание) может быть присваивание значения одного массива другому. Выигрыш состоит в том, что для массивов различной размерности может применяться одна и та же программа. Размерность массива может быть даже переменной. Так, процедура, по существу тождественная процедуре *assign*, была использована автором алгоритма 173 при реализации в АЛГОЛ-трансляторе рекурсивных процессов с **own**.

Однако, кроме того, процедура *assign* может иметь и другое применение, как это указано ниже. Например, обращение

assign((**if** i=1 **then false else** a)∨b[i,i],1,i,1,n,a,j)

вычисляет след логического двумерного массива *b*. Обращение

```
assign((if i[1]=1 then 0 else
  a[i[3],i[2]])+b[i[3],i[1]]×c[i[1],i[2]],3,i[j],1,
  if j=3 then n else if j=2 then m else p,a[i[3],i[2]],j)
```

присваивает массиву a значение произведения матриц b и c. Можно заметить, что в более общем случае процедура *assign* будет выполнять все тензорные операции, например тензорное умножение, альтернирование и т. д.

Расшифровка параметров процедуры.

a — переменная (простая или с индексами), представляющая собой общий вид элемента массива, которому должно присваиваться значение в процессе выполнения процедуры *assign*.

ind — переменная (простая или с индексом *j*), представляющая собой общий вид индексов переменной *a*, определяющих процесс.

j — идентификатор, служащий индексом переменных *ind*.

dim — количество переменных *ind*.

low, *up* — нижняя и верхняя (соответственно) границы изменения параметра *ind*.

b — выражение, значение которого должно присваиваться переменной *a*.

```
procedure assign(b,dim,ind,low,up)result:(a,j);
      value dim; integer dim,ind,low,up,j;
begin j:=dim;
      for ind:=low step 1 until up do
        if dim>1 then
          begin assign(b, dim—1, ind,low,up,a,j);j:=dim end else
          a:=b
end assign;
```

Свидетельство к алгоритму 173б

Алгоритм 173б получен из алгоритма 173а путем исправления пояснительного текста к нему и переноса параметра *j* в заголовке процедуры *assign* с последнего места в списке параметров на предпоследнее, поскольку этот параметр не является хранилищем результата выполнения процедуры.

Алгоритм 173б был транслирован в системе БЭСМ-АЛГОЛ (машина БЭСМ-6), и с его помощью были правильно решены следующие две задачи.

1. Присваивание матрице $a[1:2,1:2]$ значения матрицы

$$b=\begin{Vmatrix}1 & 2\\ 3 & 4\end{Vmatrix}.$$

При этом обращение к процедуре имело вид

```
assign(b[i[1],i[2]],2,i[j],1,2,j,a[i[1],i[2]]).
```

2. Умножение матрицы b на матрицу c, когда

$$b=\begin{Vmatrix}1 & 2\\ 3 & 4\end{Vmatrix} \text{ и } c=\begin{Vmatrix}2 & 1\\ 1 & 2\end{Vmatrix},$$

и обращение к процедуре

```
assign((if i[1]=1 then 0 else a[i[3],i[2]])+
+b[i[3],i[1]]×c[i[1],i[2]],3,i[j],1,2,j,a[i[3],i[2]]).
```

Результат:

$$a = \begin{Vmatrix} 4 & 5 \\ 10 & 11 \end{Vmatrix}.$$

Свидетельство к алгоритму 173а

Алгоритм 173а получен в результате ординарной переработки алгоритма 173 (Hajek O. «CACM», 1963, № 6).

При работе с алгоритмом 173а нужно иметь в виду, что средства, используемые в нем, выходят за рамки сокращенного АЛГОЛа [6] в следующих отношениях:

1) используется свойство рекурсивности процедур;
2) формальные параметры a и b не специфицированы;
3) фактические параметры, вызываемые по наименованию, могут быть выражениями, а не только идентификаторами.

Алгоритм 173а проверен вручную на примерах, данных в описании алгоритма. Поправка, указанная в нижеследующем «Подтверждении» З. Филсака и Л. Врховецкой, оказалась правильной и учтена в описании алгоритма 173а.

Подтверждение к алгоритму 173

Р. Скоуэн (Scowen R. S. «CACM», 1963, № 10)

Алгоритм 173 был успешно проверен с использованием транслятора DEUCE ALGOL 60. Потребовалось только добавление спецификаций для формальных параметров a и b (транслятор DEUCE ALGOL 60 требует спецификаций для всех формальных параметров). Авторский пример

```
assign((if i[3]=1 then 0.0 else a[i[1],i[2]])+
  b[i[1],i[2]]×c[i[3],i[2]],3,i[j],1,
  if j=1 then n else if j=2 then m else p,a[i[1],i[2]],j)
```

образовал матричное произведение $b \times c$ и запомнил его в a*.

Алгоритм был использован также для ввода в машину матрицы с помощью обращения

```
assign(read,2,i[j],1, if j=1 then n else p,b[i[1],i[2]],j)
```

(*read* — это вещественная процедура, которая принимает значение, задаваемое очередным числом на вводной ленте).

Для решения этих примеров требуется примерно в три раза больше времени, чем для выполнения эквивалентных им операторов

```
for i:=1 step 1 until n do
  for j:=1 step 1 until m do
    begin a[i,j]:=0.0;
      for k:=1 step 1 until p do
        a[i,j]:=a[i,j]+b[i,k]×e[k,j]
    end;
```

и

```
for j:=1 step 1 until p do
  for i:=1 step 1 until n do b[i,j]:=read;
```

---

* См., однако, нижеследующее «Подтверждение» З. Филсака и Л. Врховецкой. (*Прим. ред.*)

Подтверждение к алгоритму 173

З. Филсак, Л. Врховецка (Filsak Z., Vrchovecka L. «CACM», 1963, № 10)

Алгоритм был модифицирован для ввода в систему Elliott-ALGOL следующим образом. В системе Elliott-ALGOL вызываемые по наименованию параметры в рекурсивных процедурах специфицируются. К счастью, единственный параметр, изменяющийся во время рекурсивного обращения в теле процедуры *assign*, вызывается по значению (это параметр *dim*, который определяет глубину рекурсии). Вместо тела процедуры *assign* было поставлено: 1) описание процедуры *ass*(*dim*), тело которой такое же, как у первоначальной процедуры *assign*, но рекурсивное обращение к процедуре *assign* в котором было заменено на обращение к *ass*, и 2) отдельный оператор вызова *ass*(*dim*).

Модифицированная процедура была проверена (в вычислительном центре на машине National Elliott 803) на довольно большой последовательности примеров, включая примеры, приведенные в тексте алгоритма 173. Было обнаружено, что в последнем примере на умножение матриц индексы i_1 и i_3 нужно всюду поменять местами.

В самом алгоритме никаких изменений не потребовалось. По-видимому, описанная выше модификация, мотивированная ограничениями языка Elliott-ALGOL, повышает и эффективность алгоритма по крайней мере для большинства рассмотренных размерностей.

Свидетельство к алгоритму 174б [C2]

Алгоритм 174б («Границы корня полинома с интервальными коэффициентами») не публикуется здесь, потому что соответствующий алгоритм 174 («CACM», 1963, № 6) не был подтвержден ни в журнале «CACM», ни в расчетах составителей выпуска.

АЛГОРИТМ 175б

Сортировка последовательностей [M1]

Процедура *sort* сортирует последовательность чисел $x[1:m]$ по величине. Процедура проста, требует только одну ячейку t памяти для промежуточных значений и выполняется достаточно быстро, особенно для коротких последовательностей.

```
procedure sort(m)dataresult:(x);
        value m; integer m; array x;
begin real t; integer i,j,k;
        for i := =2 step 1 until m do
          begin k := =i; t := =x[k];
            for j := =i—1 step —1 until 1 do
              if x[j]⩽t then go to next else
                begin k := =j; x[j+1] := =x[j] end j;
next:       x[k] := =t
          end i
end sort;
```

Свидетельство к алгоритму 175б

Алгоритм 175б получен из алгоритма 175а в результате следующих тождественных преобразований, имевших целью ускорение выполнения процедуры *sort*. Заголовок цикла

for i := =1 **step** 1 **until** m—1 **do**

был заменен заголовком

for i := =2 **step** 1 **until** m **do**

а в следующие далее две строки были внесены соответствующие коррективы.

Алгоритм 175б был транслирован в системе БЭСМ-АЛГОЛ на машине БЭСМ-6 и для примеров, приведенных в нижеследующем «Свидетельстве к алгоритму 175а», дал правильные результаты.

Свидетельство к алгоритму 175а

Алгоритм 175а, составленный согласно предложению рецензента, при той же длине процедуры выполняется значительно быстрее, чем алгоритм 175 (Shaw C. J., Trimble T. N. «CACM», 1963, № 6), содержащий к тому же и ошибки (границы массива в спецификации массива N и нелокализованная переменная Temporary).

Замечания О. Юэлиха (Yueluch O. C. «CACM», 1963, № 12 и «CACM», 1964, № 5) здесь не публикуются, поскольку во втором «Замечании» отвергаются все предложения, высказанные в первом «Замечании». «Подтверждение» Г. Шуберта (Schubert G. R. «CACM», 1963, № 10), не публикуется здесь, поскольку оно потеряло свое значение после замены алгоритма 175 на алгоритм 175а.

Алгоритм 175а был транслирован и дал правильные результаты для $x=(3,5,7,2,4)$, $(7,3,9,6,9,3,1)$ и $(7,7,1,4,3,2,9,6,5)$.

АЛГОРИТМ 176б

Аппроксимация последовательности точек линейной комбинацией любых заданных функций [E2]

Процедура *surfit* по данной последовательности m ординат и соответствующих им значений от n наперед заданных основных функций f_i одного или нескольких линейно-независимых переменных аппроксимирует (в смысле метода наименьших квадратов) точки с помощью функции, имеющей форму $a_1f_1+a_2f_2+\ldots+a_nf_n$, где a_i — искомые коэффициенты.

Вычисляются также вектор разностей e_j и их средняя длина r. Должны быть предусмотрены также веса, соответствующие данным точкам. Алгоритм сводится по существу к решению матричного уравнения $f^T\cdot w\cdot f\cdot a=f^T\cdot w\cdot z$, где a — вектор искомых коэффициентов a_i; w — вектор, содержащий диагональные элементы матрицы размерностью $m\times m$ из весов данных точек; z — вектор значений ординат, а f — матрица размерностью $m\times n$ из соответствующих значений функции. Процедура *surfit* использует процедуру-параметр *invert*, заменяющую вещественную матрицу *gg* (рабочий массив) на обратную. Размерность массивов-параметров: $a[1:n],e,w,z[1:m],f[1:m,1:n]$.

```
procedure surfit(f,z,w,m,n,invert)result:(a,e,r);
      value m,n; real r; integer m,n; array a,f,z,w,e;
      procedure invert;
begin real s,g; integer i,j,k; array gg[1 : n,1 : n];
      for i : =1 step 1 until n do
        for j : =1 step 1 until n do
          begin s : =0;
            for k : =1 step 1 until m do
              s : =s+f[k,i]×f[k,j]×w[k];
            gg[i,j] : =s
          end i;
      invert(gg,n);
      for i : =1 step 1 until n do
        begin s : =0;
          for j : =1 step 1 until m do
            begin g : =0;
              for k : =1 step 1 until n do
                g : =g+gg[i,k]×f[j,k];
              s : =s+g×z[j]×w[j]
            end j;
          a[i] : =s
        end i;
      s : =0;
      for i : =1 step 1 until m do
        begin g : =z[i];
          for j : =1 step 1 until n do
            g : =g−a[j]×f[i,j];
          s : =s+g↑2; e[i] : =g
        end i;
      r : =sqrt(s/m)
end surfit;
```

Свидетельство к алгоритму 176б

Алгоритм 176б получен из алгоритма 176а в результате существенной корректировки пояснительного текста и достаточно очевидных (но существенных) тождественных преобразований тела процедуры *surfit*, имеющих целью ускорение ее выполнения. Глобальная процедура *invert*176 была заменена формальной процедурой *invert* ради удобства использования данного алгоритма в автоматизированных библиотеках, подобных архиву системы БЭСМ-АЛГОЛ.

Алгоритм 176б был транслирован в системе ТА-1М на машине М-220, и с его помощью были решены следующие задачи.

Задача 1. Аппроксимировать последовательность точек прямой $z=x$ с абсциссами 0.5(0.1)1.0 линейной комбинацией двух функций $f_1=\sin x$ и $f_2=\mathrm{sh}\, x$.

Задавались параметры $z=(0.5, 0.6, 0.7, 0.8, 0.9, 1.0)$, $m=6$, $n=2$, $w=(1, 1, 1, 1, 1, 1)$. Значения матрицы $f[1:6,1:2]$, а также функций $\sin x$ и $\mathrm{sh}\, x$ вычислялись на машине. Телом процедуры *invert* служило обращение к процедуре p0037 системы ТА-1М.

Были получены результаты $a=(0.53499527, 0.46819589)$, $e=(-0.46507855_{10}-3, -0.15964659_{10}-3, 0.18080925_{10}-3, 0.41030976_{10}-3, 0.31289157_{10}-3, -0.40737137_{10}-3)$ и $r=0.34319182_{10}-3$.

Для контроля качества аппроксимации вычислялись для $x=$ $=0.5(0.01)1.0$ максимальные по модулю значения разностей $z-f_1$, $z-f_2$ и $z-y$, где $y=a_1f_1+a_2f_2$ — аппроксимирующая функция. Были получены $(z-f_1)_{max}=0.15852901$ при $x=1.0$, $(z-f_2)_{max}=-0.17520119$ при $x=1.0$ и $(z-y)_{max}=-0.46507855_{10}-3$ при $x=0.5$.

Задача 2. Аппроксимировать последовательность точек прямой $z=x$ с абсциссами $0.0(0.1)1.0$ линейной комбинацией трех функций $f_1=fr\sin(x)$ (алгоритм 88а), $f_2=\operatorname{tg} x$ и $f_3=\operatorname{arctg} x$.

Задавались параметры $z=(0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)$, $m=11$, $n=3$, $w=(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$. Остальные параметры вычислялись так же, как в задаче 1.

Были получены результаты $a=(0.50780118, -0.020849070, 1.0304356)$, $e=(0.00000000, -0.876_{10}-3, -0.130_{10}-2, -0.105_{10}-2, -0.212_{10}-3, 0.760_{10}-3, 0.126_{10}-2, 0.836_{10}-3, -0.426_{10}-3, -0.138_{10}-2, 0.620_{10}-3)$, $r=(0.90266789_{10}-3)$.

Контроль качества аппроксимации производился так же, как в задаче 1. Были получены $(z-y)_{max}\approx-0.138_{10}-2$ при $x=0.9$, $(x-fr\sin(x))_{max}\approx0.562$, $(x-\operatorname{tg} x)_{max}\approx-0.557$, $(x-\operatorname{arctg} x)_{max}\approx0.215$. Три последних значения получены при $x=1.0$.

Свидетельство к алгоритму 176а

Алгоритм 176а получен в результате исправлений и ординарной переработки алгоритма 176 (Arthurs T. D. «CACM», 1963, № 6). Исправления заключались в следующем: 1) в алгоритме 176 спецификация **procedure** *Invert,sqrt*; является лишней и должна быть вычеркнута; 2) в последнем операторе цикла нужно заменить оператор присваивания $e[i]:=y[i]$ на $e[i]:=z[i]$.

Свидетельство к алгоритму 177б [E2, F4]

Алгоритм 177б («Решение переопределенной системы уравнений») не публикуется здесь, потому что соответствующий алгоритм 177 («CACM», 1963, № 6) не был подтвержден ни в журнале «CACM», ни в расчетах составителей выпуска.

Кроме ошибок, указанных в «Замечании к алгоритму 177» (Synge M. J. «CACM», 1963, № 7) и в «Свидетельстве к алгоритму 177а» [50], в этом алгоритме имеются и другие ошибки. В частности, элементы массива $b[1:n-r,1:m-r]$ в операторе цикла, следующем за меткой *next*, не определены, так как используются в границах $[r+1:n,r+1:m]$. Не определено и значение $ic[k]$ в операторе с меткой *repeat*.

Для решения переопределенных систем уравнений можно пользоваться более совершенным алгоритмом 328 («CACM», 68—6, 69—6).

АЛГОРИТМ 178б

Минимизация функции нескольких переменных методом прямого поиска (методом конфигураций) [E4]

Процедура *directsearch* (*direct* — прямой, *search* — поиск) может быть использована для поиска минимума функции s от k переменных. Авторы алгоритма 178 указывают, что применение этой процедуры рас-

сматривается в работе Р. Хука и Т. Дживса [10i]. Здесь используются по существу такие же обозначения, какие приняты в «Приложении В» указанной работы, за исключением очевидных модификаций, соответствующих требованиям сокращенного АЛГОЛа. См. также работу Дж. Уайлда [45, с. 202].

Расшифровка параметров:

s — процедура, вычисляющая минимизируемую функцию *s*(*psi*).

k — количество переменных функции *s*.

d — доля начальных значений аргументов, используемая как начальный размер шага. На выходе из процедуры *d* — это конечный размер шага.

rho — множитель, уменьшающий размер шага.

delta — минимально допустимый размер шага (процедура заканчивается, если размер шага становится меньше *delta*).

max — максимально допустимое количество вычислений функции. На выходе из процедуры *max* — это фактически затраченное количество вычислений функции.

psi[1 : *k*] — на входе в процедуру — это начальная точка поиска. На выходе — это найденная точка минимума, т. е. совокупность значений аргументов, при которой функция *s* имеет минимум.

spsi — на выходе из процедуры — это значение функции *s* в точке *psi*[1 : *k*].

min — наименьшая степень числа 10, представимая в данной машине.

```
procedure directsearch(k,rho,delta,s,min)
      dataresult:(psi,d,max)result:(spsi);
      value k,rho,delta,min; real rho,delta,spsi,d,min,
      integer k,max; array psi; real procedure s;
begin real sphi,ss,theta; integer i,eval; array phi,s1[1:k];
      procedure test;
        if eval<max then eval:=eval+1 else
          go to exit;
      procedure ee;
        for i:=1 step 1 until k do
          begin phi[i]:=phi[i]+s1[i];
            sphi:=s(phi); eval:=eval+1;
            if sphi<ss then ss:=sphi else
              begin s1[i]:=-s1[i];
                phi[i]:=phi[i]+2.0×s1[i];
                test; sphi:=s(phi);
                if sphi<s then ss:=sphi else
                  phi[i]:=phi[i]-s1[i]
              end
          end ee;
start: for i:=1 step 1 until k do
        begin s1[i]:=d×abs(psi[i]);
          if s1[i]<min then s1[i]:=d
        end;
      spsi:=s(psi); eval:=1;
n1:   ss:=spsi;
      for i:=1 step 1 until k do phi[i]:=psi[i];
      ee;
```

```
        if ss<spsi then
          begin
n2:         for i:=1 step 1 until k do
              begin
                if phi[i]>psi[i]≡s1[i]<0 then
                  s1[i]:=—s1[i];
                theta:=psi[i]; psi[i]:=phi[i];
                phi[i]:=2.0×phi[i]—theta
              end i;
            spsi:=ss; test; ss:=sphi:=s(phi); ee;
              if ss≥spsi then go to n1;
            for i:=1 step 1 until k do
                if abs(phi[i]—psi[i])>0.5×abs(s1[i]) then
                  go to n2
            end;
          if d≥delta then
            begin if eval>max then go to exit;
              d:=rho×d;
              for i:=1 step 1 until k do s1[i]:=rho×s1[i];
              go to n1
            end if d;
exit:   max:=eval
end directsearch;
```

Свидетельство к алгоритму 178б

Алгоритм 178б получен из алгоритма 178а путем внесения в него модификаций, предложенных Р. Де-Вогелером, Ф. Томлиным и Л. Смитом в их «Замечаниях к алгоритму 178» (см. ниже), и модификаций, внесенных в процедуру *directsearch* составителями данного выпуска и заключающихся в следующем.

1. Иногда на выходе из процедуры нужно знать количество вычислений функции, фактически производившихся в процессе выполнения процедуры. Для этой цели нет необходимости вводить новый параметр, а можно использовать параметр *max*, служивший в алгоритмах 178 и 178а только для задания максимально допустимого количества вычислений функции. Поэтому в процедуре *directsearch* и в операторах, предложенных в замечаниях вышеуказанных авторов, были сделаны следующие изменения:

1) параметр *max* в заголовке процедуры был перенесен в группу параметров, следующих за строкой букв *dataresult*;

2) после метки «exit:» был вставлен оператор *max*: =*eval*;

3) оператор **go to** *n*2 был заменен оператором **go to** *exit*;

4) в пятой модификации Ф. Томлина и Л. Смита оператор **go to** *n*3 был заменен оператором **go to** *exit*.

2. После предыдущей модификации параметр *corr* в процедуре стал лишним, поскольку судить о недостаточности (для нахождения минимума) входного значения *max* стало возможно по выходному его значению. Поэтому из заголовка процедуры был исключен параметр *corr*, а из тела процедуры — операторы «corr: =**true**;» и «corr: =**false**;».

3. В заголовок процедуры *directsearch* был добавлен параметр *min*, означающий некоторое малое число, представимое еще в машине, а в операторе цикла с меткой *start* условие **if** s1[i]=0.0 **then** было за-

менено условием **if** s1[i]<min **then**, так как равенство $s1[i]=0.0$ практически невыполнимо.

Алгоритм 178б был транслирован в системе ТА-1М на машине М-220, и с его помощью была минимизирована функция $s=(\psi_1-0.01)^2+(\psi_2-1)^2+(\psi_3-100)^2$, которая рассматривалась в нижеприведенном замечании Л. Смита. Минимизация проводилась для следующих значений входных параметров: $k=3$, $rho=0.1$, $delta=0.001$, $d=0.2$, $psi=(0.05, 5, 500)$, $min=10^{-18}$ и $max=100, 50, 40, 30, 20$ и 10.

Во всех случаях, кроме $max=10$, были получены точные значения точки минимума $psi=(0.01, 1, 100)$. Выходное значение max либо равнялось 49 (для входных значений $max=100$ и 50), либо точно совпадало с его входным значением.

Свидетельство к алгоритму 178а

Алгоритм 178а получен в результате ординарной переработки модифицированной процедуры *directsearch*, приведенной в «Замечании к алгоритму 178» М. Белла и М. Пайка (Bell M., Pike M. C. «CACM», 1966, № 9), перевод которого здесь не приводится, поскольку это «Замечание», кроме указанных модификаций и их обоснования, другой информации не содержит.

Алгоритм 178а был транслирован для $k=2$, $d=10$, $rho=0.5$, $delta=0.001$, $max=1000$. В качестве s задавалась процедура

```
real procedure s(psi);
    array psi;
    s := psi[1]↑2+psi[2]↑2;
```

т. е. вычислялся минимум параболоида $z=x^2+y^2$.

При задании начальной точки (100, 200) были получены точные координаты минимума (0,0) и $corr\equiv$**true**. Правильные результаты были получены также для параболоидов $z=(x+1)^2+(y-1)^2-2$ и $z=\sum_{i=1}^{10} x^2_i$.

Время счета на машине М-20 равнялось 6—7 с*.

Замечание к алгоритму 178

Р. Де-Вогелер (De Vogelaer R. «CACM», 1968, № 7)

Вопреки утверждению автора алгоритма в процедуре не происходит выход к метке *exit*, когда число итераций достигает заданного значения *max*.

Для исправления этого нужно три оператора *eval*:=*eval*+1 поменять местами с предшествующими им операторами и заменить их обращением к процедуре *test*, описанной ниже:

```
procedure test;
    if eval<max then eval := eval+1 else
      begin corr := false; go to exit end test;
```

Оператор, следующий за меткой *n2*, нужно вычеркнуть.

---

*См. также «Подтверждение к алгоритму 178а». П. И. Гостева и С. И. Лаврушина в приложении 1 к выпуску 3 [49], а также «Подтверждение и замечания к алгоритмам 17б, ..., 178а» Ю. А. Михайлова в приложении 2 к данному выпуску. (*Прим. ред.*)

Замечание к алгоритму 178

Ф. Томлин, Л. Смит (Tomlin F. K., Smith L. B. «CACM», 1969, № 11)

Алгоритм 178 в том виде, как он был модифицирован М. Беллом и М. Пайком (Bell M., Pike M. C. «CACM», 1966, № 9), не всегда дает определенный минимум. Кроме того, в тот момент, когда количество вычислений функций больше или равно значению *max* и размер шага больше *delta*, максимально допустимое количество вычислений функции почти всегда оказывается уже превышенным. Наконец, метка 3 в алгоритме не используется. Чтобы обеспечить получение определенного минимума, тест проверки количества вычислений нужно переместить к той точке, где достигается минимум.

Р. Де-Вогелер в «Замечании к алгоритму 178» («CACM», 1968, № 7) правильно отметил, что выход из процедуры происходит не так, как это определено в пояснительном тексте к ней, и дал поправки, которые действительно приводят к тому, что процедура заканчивается, когда количество вычислений функции превышает заданный предел (а не после некоторого количества вычислений). Однако думается, что решение этой задачи, предложенной Де-Вогелером, приводит к излишним проверкам. Следовательно, процедура *test* должна выполняться не только после пробного продвижения, как это было у М. Белла и М. Пайка, но и тогда, когда уменьшается размер шага. Такой метод не соответствует описанию использования параметра *max*, данному в пояснительном тексте, но не противоречит принципу исключения лишних вычислений. Для получения определенного минимума, для уменьшения количества вычислений функции, когда размер шага больше *delta*, и для удаления лишней метки нужно внести следующие изменения.

1. Строки *

```
n2:    if eval⩾max then begin corr := =false; go to exit end;
```

нужно вычеркнуть.

2. Следующий за этими строками заголовок цикла

```
    for i := =1 step 1 until k do
```

нужно заменить на

```
n2:  for i := =1 step 1 until k do
```

3. После оператора

```
    spsi := =ss;
```

нужно вставить следующие строки:

```
      if eval⩾max then
n3:     begin corr := =false; go to exit end;
```

4. Строку **

```
n3:  if d⩾delta then
```

нужно заменить на строку

```
      if d⩾delta then
```

5. Строку

```
      begin d := =rho×d;
```

нужно заменить на строки

```
      begin
        if eval>max then go to n3;
        d := =rho×d;
```

---

* В данном переводе используются обозначения, принятые в алгоритме 178а. (*Прим. ред.*)

** Эта модификация была уже внесена в алгоритм 178а. (*Прим. ред.*)

Замечание к алгоритму 178

Л. Смит (Smith L. B. «CACM», 1969, № 11)

Алгоритм 178 в том виде, как он был модифицирован М. Беллом и М. Пайком (Bell M., Pike M. C. «CACM», 1966, № 9), был успешно использован автором данного «Замечания» для решения нескольких разных задач с применением различных языков (таких, например, как расширенный АЛГОЛ для машины B 5500, сокращенный АЛГОЛ для машины IBM 7090 и ФОРТРАН для машины IBM/360). Было найдено полезным внесение одной модификации, включающей выбор шага так, чтобы он стал эффективным для переменных, значительно различающихся по величине.

М. Беллом и М. Пайком было показано, что как только будет найден минимум, то каждая переменная увеличится (или уменьшится) на величину *d*. Если функция такова, что в районе минимума значения переменных различаются на несколько порядков, то универсальный шаг приведет к тому, что в процессе поиска некоторые параметры будут по существу игнорированы. Например, если функция двух переменных имеет минимум вблизи точки (100.0, 0.1), то для минимизации по первому параметру размер шага 10.0 будет полезным, но для минимизации по второму параметру — бесполезным до тех пор, пока этот шаг не будет уменьшен до величины 0.01. С другой стороны, размер шага 0.01 для второй переменной будет полезным, а для минимизации по первой переменной он будет требовать слишком много итераций.

Модификация процедуры *directsearch*, решающая задачу масштабирования, предусматривает использование для каждой переменной шага соответствующего размера. Такая модификация легко осуществляется, поскольку для запоминания размеров шага, снабженных знаками, уже используется массив. Изменение выполняется путем удаления оператора с меткой *start* и замены его оператором *

```
start: for i := 1 step 1 until k do

    begin s1[i] := d×abs(psi[i]);
      if s1[i] = 0.0 then s1[i] := d
    end;
```

Это изменение полагает размер шага по каждой переменной равным начальному значению, умноженному на *d*, или если начальное значение равно 0.0, то равным *d*. Следовательно, *d* — это доля первоначального значения каждой переменной, используемая как начальный размер шага. Соответствующие уменьшения размера шага делаются по-прежнему без дополнительных модификаций процедуры.

Для иллюстрации полезности вышеуказанной модификации рассмотрим функцию

$$f(x_1,x_2,x_3) = (x_1 - 0.01)^2 + (x_2 - 1.0)^2 + (x_3 - 100.0)^2$$

с минимумом в точке (0.01, 1.0, 100.0). В табл. 20 показаны результаты использования процедуры *directsearch* для этой функции, полученные с модификацией размера шага и без нее. Результаты были вычислены на машине IBM/360/75 с использованием арифметики ординарной точности при *rho*=0.1, *delta*=0.001, *d*=0.2 для модифицированного выбора шага (когда для начального размера шага задавался масштаб 20% от начального значения переменной), а для опубликованного алгоритма *d*

* Здесь используются обозначения, принятые в алгоритме 178а. (*Прим. ред.*)

Процедура	d	m	f_{min}	Конечные значения переменных		
				x_1	x_2	x_3
Для начальных значений (0.0, 0.0, 200.0)						
Без модиф.	66.6667	153	0.841_{10}—7	0.00999995	0.999995	100.000
С модиф.	0.2	112	0.579_{10}—7	0.00999998	0.999990	100.000
Для начальных значений (0.05, 5.0, 500.0)						
Без модиф.	168.35	174	0.934_{10}—7	0.0100263	0.998958	99.9999
С модиф.	0.2	75	0.559_{10}—6	0.00999988	0.999998	99.9992

задавалось равным среднему значению начальных приближений к переменным.

Заметим, что модифицированный метод будет давать для каждого параметра одинаковые относительные погрешности, в то время как фиксированный размер шага приводит к одинаковым абсолютным погрешностям. В большинстве случаев относительная точность более предпочтительна, чем абсолютная.

В табл. 20 буква m — количество вычислений функции f.

АЛГОРИТМ 179б

Отношение неполных бета-функций [S14]

Процедура *incbeta* (*incomplete* — неполная, *beta* — бета-функция) вычисляет отношение $B_x(p,q)/B_1(p,q)$, где $B_x(p,q)=\int_0^x t^{p-1}(1-t)^{q-1}dt$, $0\leqslant x\leqslant 1$ и $p,q>0$, но не обязательно целые. При недопустимых значениях аргументов происходит выход из процедуры к нелокализованной метке *signal*179. В теле процедуры *incbeta* используется нелокализованная процедура *fact*(p), вычисляющая гамма-функцию

$$\Gamma(p)=\int_0^\infty t^{p-1}e^{-t}dt.$$

Метки *n*1, *n*2, *n*3, *n*4, *n*5 на работу процедуры влияния не оказывают, служат лишь для ее пояснения и могут не перфорироваться и не вводиться в машину. Пояснения к операторам, соответствующим этим меткам, помещены ниже за описанием процедуры.

```
real procedure incbeta(x,p,q,eps);
      value x,p,q,eps; real x,p,q,eps;
begin real s1,s2,t,t1,r,r1,qr,i; Boolean xx;
      if x>1∨x<0∨p≤0∨q≤0 then go to signal179;
      if x=0∨x=1 then
        begin incbeta:=x; go to fin end;
      xx:=x>0.5;
n1:   if xx then
        begin t:=p; p:=q; q:=t; x:=1—x end;
n2:   s2:=0; r:=1; t:=1—x; qr:=i:=q;
```

```
      for i := =i—1 while i>0 do
        begin qr := =i;
          r := r×(qr+1)/(t×(p+qr));
          s2 := =s2+r
        end i;
n3:   s1 := =r := =1; i := =0;
      for i := =i+1 while r>eps×s1 do
        begin r := =r×x×(i—qr)×(p+i—1)/(i×(p+i));
          s1 := =s1+r
        end i;
n4:   t := =t1 := =fact(qr);
      r := =r1 := =fact(qr+p);
      for i := =qr step 1 until q—0.5 do
        begin t1 := =t1×i; r1 := =r1×(i+p) end i;
n5:   t := =x↑p×(s1×r/(p×t)+s2×r1×(1—x)↑q/(q×t1))/fact(p);
      incbeta := = if xx then 1—t else t;
fin: end incbeta;
```

Пояснения к операторам процедуры:

*n*1 — этот оператор при $x>0.5$ делает перестановку значений аргументов для улучшения сходимости нижеприведенного ряда.

*n*3 — эта часть процедуры дает сумму степенного ряда для нецелых значений q и дает единицу для целых q.

В следующем операторе цикла критерий сходимости $r>eps\times s1$ можно заменить на $r>eps$, так как $s1$ всегда больше 1. В результате будет сэкономлено одно умножение на каждом витке цикла ценою, возможно, некоторого увеличения числа циклов.

*n*4 — эта часть процедуры вычисляет большинство из необходимых значений функции *fact*, уменьшая тем самым количество обращений к процедуре *fact*.

*n*5 — в этой части из частных результатов комплектуется конечный результат.

Свидетельство к алгоритму 179а

Алгоритм 179а получен в результате ординарной переработки и исправления алгоритма 179 (Ludwig O. G. «CACM», 1963, № 6).

Таблица 21

x	$p=0.5;\ q=0.5$		$p=2;\ q=1.5$	
	Алг. 179а	Контрольные значения	Алг. 179а	Контрольные значения
0.1	0.204832761	0.2048328	0.0181127862	0.0181128
0.2	0.295167225	0.2951672	0.0697957196	0.0697957
0.3	0.369010029	0.3690101	0.150790065	0.1507901
0.4	0.435905592	0.4359058	0.256387161	0.2563872
0.5	0.499999798	0.5000000	0.381281476	0.3812816
0.6	0.564094408	0.5640942	0.519333796	0.5193338
0.7	0.630989971	0.6309899	0.663150627	0.6631506
0.8	0.704832774	0.7048328	0.803226018	0.8032260
0.9	0.795167239	0.7951672	0.925686475	0.9256865
1.0	1.000000000	1.0000000	1.000000000	1.0000000.

Результаты трансляции алгоритма 179а при $eps=10^{-6}$ приведены в табл. 21, где контрольные значения были взяты из работы К. Пирсона [27].

Такое же хорошее совпадение результатов трансляции с контрольными значениями было получено для случаев: ($p=1$, $q=0.5$), ($p=1$, $q=1$) и ($p=1.5$, $q=1.5$). При этом для вычисления гамма-функции использовался алгоритм 80а.

При составлении алгоритма 179а в алгоритме 179 была сделана замена идентификаторов, указанная в табл. 22.

Таблица 22

incompletebeta	incbeta	term	r
epsilon	eps	term1	r1
infsum	s1	qrecur	qr
finsum	s2	index	i
temp	t	alter	xx
temp1	t1	factorial	fact
alarm	signal 179	end	fin

Замечание к алгоритму 179

М. Пайк, И. Хилл (Pike M. C., Hill I. D. «CACM», 1967, № 6)

В алгоритме 179 имеются две опечатки...*

С этими поправками алгоритм 179 успешно прошел на машине ICT Atlas с использованием алгоритма 221 (Gautschi W. «CACM», 1964, № 3) для вычисления гамма-функции. Можно сделать одно небольшое улучшение, если вызывать *eps* по значению.

В алгоритме, как он составлен, интервал допустимых значений p и q определяется опасностью возникновения переполнения при вычислении гамма-функции. Для большинства машин это означает, что $p+q$ должно быть меньше 70. В статистических функциях распределения, приводимых ниже, это ограничение весьма существенно. Однако эти гамма-функции появляются по существу только в форме отношений, и, пользуясь этим фактом, интервал допустимых значений p и q можно существенно расширить. Это легко выполняется, если пользоваться алгоритмом 291 (Pike M. C., Hill I. D. «CACM», 1966, № 9) и внести в алгоритм 179 следующее изменение: операторы, начиная с помеченного меткой *n*4 до помеченного меткой *n*5 включительно, нужно заменить на оператор

```
t:=x↑p×(s1×exp(loggamma(qr+p)—loggamma(qr)
  —loggamma(p+1))+s2×(1—x)↑q×exp(loggamma(p+q)
  —loggamma(p)—loggamma(q+1)));
```

Это означает также и то, что описания переменных *t*1 и *r*1 не нужны. Для четных и достаточно больших значений p и q это приведет также к ускорению работы алгоритма (см. перевод «Замечания к алгоритмам 34, 54, 80, 221 и 291» М. Пайка и И. Хилла [47, с. 89]).

Следующие процедуры-функции используют алгоритм 179 для вычисления наиболее часто встречающихся статистических функций распределения.

Процедура-функция *Ftail* вычисляет вероятность того, что случайная переменная, подчиняющаяся F-распределению с параметрами (сте-

* Указываются поправки, учтенные в алгоритме 179а. (*Прим. ред.*)

пенями свободы) $f1$ и $f2$, окажется больше положительного значения k.

```
real procedure Ftail(k,f1,f2,eps);
    value k,f1,f2,eps; real k,f1,f2,eps;
Ftail:=incbeta(f2/(f2+f1×k),0.5×f2,0.5×f1,eps);
```

Процедура-функция *student* вычисляет вероятность того, что абсолютное значение случайной переменной, подчиняющейся t-распределению (распределению Стьюдента) с f степенями свободы, окажется больше положительного значения k.

```
real procedure student(k,f,eps);
    value k,f,eps; real k,f,eps;
student:=incbeta(f/(f+k↑2),0.5×f,0.5,eps);
```

Процедура-функция *binomial* вычисляет вероятность того, что случайная переменная, подчиняющаяся биномиальному распределению с параметрами n и p, окажется не меньше k.

```
real procedure binomial(k,n,p,eps);
    value k,n,p,eps; real k,n,p,eps;
binomial:=incbeta(p,k,n—k+1,eps);
```

Замечание рецензента к алгоритму 179а

Очевидно, что так составленный алгоритм не является наилучшим: уже при $p=22$ функция $\Gamma(22)>10^{19}$ и на таких машинах, как М-20, возникает переполнение, в то время как вся функция

$$\frac{\Gamma(p+q)}{\Gamma(p)\,\Gamma(q)}\int_0^x t^{p-1}(1-t)^{q-1}\,dt$$

изменяется в пределах от 0 до 1. Таким образом, алгоритм может работать в очень небольшой области изменения p и q, даже если учесть «Замечание к алгоритму 179» М. Пайка и И. Хилла, где говорится об использовании нелокализованной процедуры *loggamma*(p). Алгоритм нуждается в существенном улучшении.

Замечание к алгоритму 179

О. Людвиг (Ludwig O. G. «CACM», 1974, № 3)

Алгоритм 179, модифицированный согласно замечанию М. Пайка и И. Хилла*, вычисляет отношение неполных бета-функций, пользуясь равенством

$$I_x(p,q)=\frac{\mathrm{INFSUM}\,x^p\Gamma(PS+p)}{\Gamma(PS)\,\Gamma(p+1)}+\frac{x^p(1-x)^q\,\Gamma(p+q)\times\mathrm{FINSUM}}{\Gamma(p)\,\Gamma(q+1)},$$

INFSUM и FINSUM — это суммы двух рядов, определяемые следующим образом:

$$\mathrm{INFSUM}=\sum_{i=0}^{\infty}\frac{(1-PS)_i\,p}{p+i}\cdot\frac{x^i}{i!},$$

где $(1-PS)_i=\begin{cases}1 \text{ для } i=0,\\ (1-PS)(2-PS)\ldots(i-PS)=\dfrac{\Gamma(1+i-PS)}{\Gamma(1-PS)} \text{ для } i>0;\end{cases}$

---

* Перевод этого замечания см. выше. (*Прим. ред.*)

$$\text{FINSUM} = \sum_{i=1}^{[q]} \frac{q(q-1)\ldots(q-i+1)}{(p+q-1)(p+q-2)\ldots(p+q-i)} \cdot \frac{1}{(1-x)^i},$$

где $[q]$ равно наибольшему целому, меньшему q. Если $[q]=0$, то FINSUM=0. Значение PS определяется как

$$PS = \begin{cases} 1, & \text{если } q \text{ целое,} \\ q - [q] & \text{в противном случае.} \end{cases}$$

Если переделать алгоритм 179 так, чтобы было введено масштабирование, то интервал аргументов p и q может быть расширен, а точность повышена.

Поскольку $I_x(p,q)$ — вероятность и, следовательно, заключена в интервале [0,1], а INFSUM и FINSUM — ряды только с положительными членами, то $I_x(p,q)$ — это совокупность членов, каждый из которых положителен и заключен в интервале [0,1], если, во-первых, каждый член ряда INFSUM умножается на $(x^p\Gamma(PS+p))/(\Gamma(PS)\times\Gamma(p+1))$ и, во-вторых, каждый член ряда FINSUM умножается на $(x^p(1-x)^q\Gamma(p+q))/(\Gamma(p)\Gamma(q+1))$.

Зная этот факт, мы можем дополнить алгоритм процедурой масштабирования. INFSUM — это ряд с убывающими членами. Если произведение первого члена ряда INFSUM на его коэффициент обратится в машинный нуль, то сумму этого ряда можно положить равной нулю и можно избежать выполнения всех вычислений, дающих машинный нуль. Это осуществлено в нижеприведенной модификации алгоритма. Но поскольку INFSUM — это ряд с убывающими членами, то обращение в машинный нуль может встретиться в более поздних вычислениях. Здесь не делается никакой попытки манипулировать этим.

Второй ряд более сложный. Этот ряд будет убывающим, если $q/((q+p-1)(1-x))$ меньше единицы. Если отдельный член ряда станет меньше чем $_{10}$—6 от предыдущей суммы, то вычисления на законном основании можно заканчивать, поскольку заметного прибавления суммы уже не будет. Вычисления заканчиваются и тогда, когда член ряда окажется меньше, чем произвольно заданная малая константа (EPS2). Это делается для того, чтобы предотвратить превращения в машинный нуль последующих членов.

Если ряд возрастающий, то обратиться в нуль может первый член. В этом случае из каждого члена ряда FINSUM (помноженного на его коэффициент) можно выделить в качестве множителя степень ε_1 (машинная точность $_{10}$—78 для IBM 360/370). Эти члены нельзя добавить к сумме, поскольку они меньше машинной точности, но они полезны для сохранения точности начальных членов, которые используются затем рекурсивно. По смыслу задачи мы знаем, что любой член ряда FINSUM (помноженный на его коэффициент) должен быть не больше единицы, но мы вынесли за скобки степени ε_1. Следовательно, если член ряда FINSUM становится больше единицы, то мы знаем, что ремасштабирование путем умножения члена на ε_1 будет правильным.

Апробирование на машине IBM 360/195 показало, что в результате переработки исходного алгоритма 179 и включения в него масштабирования входной интервал алгоритма может быть значительно расширен, причем с высокой степенью точности.

Нижеприведенная подпрограмма MDBETA требует функцию двойной точности DLGAMA, вычисляющую логарифм от гамма-функции. Можно пользоваться алгоритмом 291. Подпрограмма MDBETA была проверена совместно с программой SSP BDTR, приведенной в отчете [11i]. MDBETA работала в 3.5 раза быстрее, чем BDTR, и с большей точностью. Например, для случая $x=0.5$, $p=2000$ и $q=2000$ MDBETA дала результат 0.5, тогда как BDTR дала ответ 0.497026. Для дополнительного сравнения была использована подпрограмма IMSL, имеющая название MDBIN. Эта подпрограмма работает с одинарной точностью машины IBM 370/360 (приблизительно шесть значащих цифр). Для выполненных тестов максимальное расхождение встречалось в пятой значащей цифре, когда p и q были меньше 200. Для p и q, достигающих значения 2000, можно ожидать четыре точных значащих цифры.

Благодарности. Вышеуказанные идеи являются приложением идей покойного Хирондо Куки (Hirondo Kuki). Первоисточником программы MDBETA является программа в машинном коде, принадлежащая лаборатории 1 IMSL. Мы благодарны В. Фуллерторну (W. Fullerton) из Лос-Аламосской лаборатории Калифорнийского университета за рецензирование этой работы...*

Свидетельство к алгоритмам 180б и 181б [S15]

Алгоритмы 180б и 181б не публикуются здесь, потому что соответствующие алгоритмы 180 и 181 («CACM», 1963, № 6), а также алгоритмы 180а и 181а нужно считать устаревшими по причинам, указанным в «Замечаниях к алгоритмам 123, 180, 181, 209, 226, 272 и 304» И. Хилла и С. Джойса, перевод которых опубликован в предыдущем выпуске [49] вслед за свидетельством к алгоритму 123б. Интеграл вероятности можно вычислять с помощью более совершенных алгоритмов 209а и 304 («CACM», 1967, № 6).

АЛГОРИТМ 182б

Вычисление интеграла по Симпсону с заданной мерой погрешности [D1]

Процедура-функция *simpson* приближенно вычисляет интеграл от функции $f(x)$ в пределах от a до b методом Симпсона с заданной допустимой относительной мерой погрешности *eps*.

Данный алгоритм является нерекурсивным вариантом алгоритма 145б. Использованный здесь метод преобразования рекурсивной процедуры в нерекурсивную может применяться для широкого класса алгоритмов.

```
real procedure simpson(f,a,b,eps);
      value a,b,eps; real a,b,eps; real procedure f;
begin real absarea,est,fa,fm,fb,da,sx,est1,sum,f1;
      integer lvl; array dx,epsp,x2,x3,f2,f3,f4,fmp,
      fbp,est2, est3[1 : 30],pval[1 : 30,1 : 3];
```

* Далее приводится подпрограмма MDBETA на языке ФОРТРАН, которая здесь опущена. Читатель может при желании найти эту подпрограмму непосредственно в журнале «CACM». (*Прим. ред.*)

```
        integer array rtrn[1 : 30];
        switch return := r1,r2,r3;
start : lvl := 0; da := b - a;
        absarea := est := 1.0;
        fa := f(a); fb := f(b);
        fm := 4.0 × f((a+b)/2.0);
recur : lvl := lvl+1; dx[lvl] := da/3.0;
        sx := dx[lvl]/6.0; f1 := 4.0 × f(a+dx[lvl]/2.0);
        x2[lvl] := a+dx[lvl]; f2[lvl] := f(x2[lvl]);
        x3[lvl] := x2[lvl]+dx[lvl]; f3[lvl] := f(x3[lvl]);
        epsp[lvl] := eps; f4[lvl] := 4.0 × f(x3[lvl]+dx[lvl]/2.0);
        fmp[lvl] := fm; fbp[lvl] := fb;
        est1 := (fa+f1+f2[lvl]) × sx;
        est2[lvl] := (f2[lvl]+f3[lvl]+fm) × sx;
        est3[lvl] := (f3[lvl]+f4[lvl]+fb) × sx;
        sum := est1+est2[lvl]+est3[lvl];
        absarea := absarea - abs(est)+abs(est1)+
          abs(est2[lvl])+abs(est3[lvl]);
        if (abs(est-sum) ≤ epsp[lvl] × absarea)
          ∧ (est ≠ 1.0) ∨ (lvl ≥ 30) then
          begin
up:         lvl := lvl-1;
            pval[lvl,rtrn[lvl]] := sum;
            go to return[rtrn[lvl]]
          end;
        rtrn[lvl] := 1; da := dx[lvl];
        fm := f1; fb := f2[lvl];
        eps := epsp[lvl]/1.7; est := est1;
        go to recur;
r1:     rtrn[lvl] := 2; da := dx[lvl];
        fa := f2[lvl]; fm := fmp[lvl]; fb := f3[lvl];
        eps := epsp[lvl]/1.7; est := est2[lvl]; a := x2[lvl];
        go to recur;
r2:     rtrn[lvl] := 3; da := dx[lvl];
        fa := f3[lvl]; fm := f4[lvl]; fb := fbp[lvl];
        eps := epsp[lvl]/1.7; est := est3[lvl]; a := x3[lvl];
        go to recur;
r3:     sum := pval[lvl,1]+pval[lvl,2]+pval[lvl,3];
        if lvl > 1 then go to up;
        simpson := sum
end simpson;
```

Свидетельство к алгоритму 182a

Алгоритм 182a получен в результате исправления и ординарной переработки алгоритма 182 (Mckeeman W. M., Tesler L. «CACM», 1963, № 6). Кроме ошибок, указанных в нижеследующем «Подтверждении» Х. С. Батлера, была исправлена синтаксическая ошибка, заключавшаяся в том, что список значений в алгоритме 182 был помещен после спецификаций.

Алгоритм 182а был транслирован для вычисления интеграла $\int_0^3 x^2 dx$ при $eps=10^{-2}$ и 10^{-6}. В обоих случаях было получено точное значение интеграла.

Перед трансляцией в алгоритм 182а была внесена модификация, вызванная ограниченностью входного языка транслятора ТА-1, а именно, оператор

go to return[rtrn[lvl]]

был заменен операторами

i := =rtrn[lvl]; **go to** return[i]

и добавлено описание **integer** i;

Кроме того, правильные результаты были получены при вычислении интегралов

$$\int_0^{90} \sin(x/k)\, d(x/k) = 1 \quad \text{и} \quad \int_{-90}^{90.1} \sin(x/k)\, d(x/k) = 0.00174532829,$$

где $k=57.2957795$ и x выражены в градусах.

Подтверждение к алгоритму 182

Х. С. Батлер (Butler H. S. «CACM», 1964, № 4)

В Стенфорде авторы процедуры *simpson* подготовили транслитерацию алгоритма на язык BALGOL и использовали ее в некоторых задачах, содержащих численное интегрирование. Ценность процедуры наиболее убедительно проявилась, когда она была использована для вычисления тройного интеграла, в котором конечное интегрирование производилось над функцией с острым пиком, измеряемой величиной седьмого порядка.

Процедура *simpson* эффективно уменьшала количество вычислений и выполняла интегрирование в пять раз быстрее, чем это делалось другими методами. Значения интеграла хорошо согласовывались в пределах допустимой погрешности с независимо проведенными расчетами.

В опубликованном алгоритме необходимы следующие изменения... *

АЛГОРИТМ 183б

Преобразование ленточной симметричной матрицы в трехдиагональную [F2]

Процедура *bandred* (сокращение от *band* — лента, *reduction* — преобразование) преобразует вещественную симметричную матрицу ленточной формы (порядок n, $a[i,k]=0$ для $|i-k|>m$) к трехдиагональной форме с помощью последовательности ортогональных преобразований подобия. Данная процедура представляет собой обобщение алгоритма $m21$ Х. Рутисхаузера.

---

* Далее указываются четыре поправки к алгоритму 182, учтенные в алгоритме 182а. (*Прим. ред.*)

Благодаря симметрии нужно задавать только верхнюю часть ленточной матрицы, и элементы этой части для удобства обозначаются следующим образом: $a[i,0]$ (для $i=1, 2, \ldots, n$) представляет собой диагональный элемент i-й строки, а $a[i,k]$ для $k=1, 2, \ldots, m$ и $i=1, 2, \ldots, n-k$ представляет собой ленточный элемент i-й строки, находящийся в k-й позиции вправо от диагонали. Значение $a[i,k]$ для $k=1, 2, \ldots, m$ и $i>n-k$ несущественно. После выполнения преобразования элементы трехдиагональной симметричной матрицы задаются в виде $a[i,0]$ (для $i=1, 2, \ldots, n$) и $a[i,1]$ (для $i=1, 2, \ldots, n-1$).

В теле процедуры *bandred* составной оператор, выполняющийся в цикле по j, т. е. помещенный после заголовка цикла

for j := k **step** r **until** n—r **do**

описывает вращение, включающее в себя i-ю и $(i+1)$-ю строки и столбцы и выполняемое для того, чтобы привести к нулю либо $a[j,r]$, либо внеленточный элемент g соответственно. Это вращение порождает новый внеленточный элемент g (вообще говоря, отличный от нуля) при условии, что $i+r<n$.

Размерность массива-параметра: $a[1:n,0:m]$.

О преобразованиях симметричных матриц к трехдиагональным см. работу Д. К. Фаддеева и В. Н. Фаддеевой [9, с. 333].

```
procedure bandred(n,m) dataresult:(a);
    value n,m; integer n,m; array a;
begin real b,g,c,s,c2,s2,cs,u,v; integer i,j,k,p,r,rr;
    for r := m step —1 until 2 do
      for k := 1 step 1 until n—r do
        begin
          for j := k step r until n—r do
            begin b := if j=k then a[j,r] else g;
              s := if j=k then —a[j,r—1] else —a[j—1,r];
              if b=0 then go to endk;
              b := s/b; s := 1/sqrt(1+b×b); c := b×s;
              c2 := c×c; s2 := s×s; cs := c×s; i := j+r—1;
              u := a[i,0]; v := a[i,1]; b := a[i+1,0];
              a[i,0] := c2×u—2×cs×v+s2×b;
              a[i+1,0] := s2×u+2×cs×v+c2×b;
              a[i,1] := cs×(u—b)+v×(c2—s2);
              comment Преобразование столбцов;
              for p := j step 1 until i—1 do
                begin u := c×a[p,i—p]—s×a[p,i—p+1];
                  a[p,i—p+1] := s×a[p,i—p]+c×a[p,i—p+1];
                  a[p,i—p] := u
                end p;
              if j≠k then a[j—1,r] := c×a[j—1,r]—s×g;
              comment Преобразование строк;
              rr := if r≤n—i then r else n—i;
              for p := 2 step 1 until rr do
                begin u := c×a[i,p]—s×a[i+1,p—1];
                  a[i+1,p—1] := s×a[i,p]+c×a[i+1,p—1];
                  a[i,p] := u
                end p;
```

```
                if i+r<n then
                  begin comment Получение нового g;
                    g := —s×a[i+1,r];
                    a[i+1,r] := c×a[i+1,r]
                  end ifi
              end j;
endk:     end k
end bandred;
```

Свидетельство к алгоритму 183а

Алгоритм 183а получен в результате ординарной переработки алгоритма 183 (Schwarz H. R. «CACM», 1963, № 6).

Алгоритм 183а транслирован с исходными данными, указанными в «Свидетельстве к алгоритму 122а» [49]. При этом матрица a задавалась в форме

$$a = \begin{Vmatrix} 4 & 3 & 2 & 1 \\ 3 & 2 & 1 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Vmatrix},$$

а результат был получен в виде

$$a = \begin{Vmatrix} 4.00000000 & 3.74165738 & 0 & 0 \\ 5.00000000 & 0.462910050 & \sim 0.18\times 10^{-11} & 0 \\ 0.666666666 & 0.0890870806 & 0 & 0 \\ 0.333333333 & 0 & 0 & 0 \end{Vmatrix}.$$

Этот результат совпадает с результатом, приведенным в указанном «Свидетельстве». С такой же точностью и в такой же форме был получен результат для второго примера «Свидетельства».

Кроме того, алгоритмы 183а и 122а дали одинаковые результаты для матрицы

$$a = \begin{Vmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 2 & 5 & 6 & 7 & 0 & 0 \\ 3 & 6 & 10 & 11 & 12 & 0 \\ 0 & 7 & 11 & 16 & 17 & 18 \\ 0 & 0 & 12 & 17 & 21 & 22 \\ 0 & 0 & 0 & 18 & 22 & 25 \end{Vmatrix}.$$

АЛГОРИТМ 184б

Табулирование закона распределения Эрланга [S22]

Процедура *erlang* вычисляет вероятность Эрланга для i-го интервала по формуле

$$p_i = \int_0^{x_i} f(x)\,dx - \int_0^{x_{i-1}} f(x)dx,$$

где $f(x)=\frac{(x-x_0)^{k-1}}{(k-1)!}\left(\frac{k}{m}\right)^k e^{-\frac{k}{m}(x-x_0)}$;

m — среднее значение в распределении Эрланга;
$k=(m-x_0)^2/vars$ — верхняя граница интервалов группировки;
x_0 — нижняя граница первого интервала группировки;
$vars$ — дисперсия, уточненная с помощью поправки Шеппарда (см. [35, 36]);
c — число интервалов группировки;
p — вычисляемые значения вероятности; размерность массива $p[1:c+1]$;
x — массив значений x_i, имеющий размерность $[1:c]$.

О распределении Эрланга см., например, работу Д. Кокса и У. Смита [17].

```
procedure erlang(x,x0,m,vars,c)result:(p);
      value x0,m,vars,c; real x0,m,vars; integer c;
      array x,p;
begin real me,sp,u,sum1,sum2,z1,z2; integer i,j,k,f;
      array xe[0:c];
      for i:=1 step 1 utnil c do xe[i]:=x[i]-x0;
      xe[0]:=0; me:=m-x0;
      k:=me↑2/vars;
      u:=k/me; sp:=0;
      for i:=1 step 1 until c do
        begin sum1:=sum2:=1; f:=1;
          z1:=u×xe[i-1]; z2:=u×xe[i];
          for j:=1 step 1 until k-1 do
            begin f:=f×j;
              sum1:=sum1+z1↑j/f;
              sum2:=sum2+z2↑j/f
            end j;
          p[i]:=sum1×exp(-u×xe[i-1])-sum2×exp(-u×xe[i]);
          sp:=sp+p[i]
        end i;
      p[c+1]:=1.0-sp
end erlang;
```

Свидетельство к алгоритму 184а

Алгоритм 184а получен в результате исправления, оптимизации и ординарной переработки алгоритма 184 (Colker A. «CACM», 1963, № 7). В алгоритм 184 были внесены следующие исправления:

1) параметры $x0$, m, $vars$, вызываемые по значению, были специфицированы;

2) переменные типа **real** были локализованы в теле процедуры;

3) оператор $k:=0.5+(me\uparrow 2)/vars$ был заменен оператором $k:=me\uparrow 2/vars$.

Оптимизация алгоритма 184 состояла в следующем:

1) операторы, вычисляющие $z1$ и $z2$, были вынесены за пределы цикла по j;

2) процедура $factorial(j)$ была заменена непосредственным вычислением $j!$.

Алгоритм 184а был транслирован для следующих исходных данных:
1) $x0=0$, $m=2$, $vars=1$, $c=10$, $x=0.05(0.1)0.95$;
2) $x0=0$, $m=3$, $vars=1$, $c=10$, $x=0.5(0.1)1.4$;
3) $x0=1.5$, $m=6.5$, $vars=2.5$, $c=10$, $x=5(0.125)6.125$.
Некоторые из полученных результатов приведены в табл. 23.

Таблица 23

i	1		2		3	
	x_i	p_i	x_i	p_i	x_i	p_i
1	0.05	$0.384614395\cdot10^{-5}$	0.5	$0.277356157\cdot10^{-4}$	5.000	0.169504062
2	0.15	$0.261965744\cdot10^{-3}$	0.6	$0.820085989\cdot10^{-4}$	5.125	$0.262283211\cdot10^{-1}$
3	0.25	$0.148581085\cdot10^{-2}$	0.7	$0.227561831\cdot10^{-3}$	5.375	$0.278600043\cdot10^{-1}$
4	0.35	$0.400183482\cdot10^{-2}$	0.8	$0.524672680\cdot10^{-3}$	5.600	$0.292888226\cdot10^{-1}$
10	0.95	$0.321083524\cdot10^{-1}$	1.4	$0.939943409\cdot10^{-2}$	6.125	$0.329013462\cdot10^{-1}$
11	—	0.874702214	—	0.972067809	—	0.554506887

В статистике распределение Эрланга называется распределением Пирсона III типа [10, с. 39]. Последнее связано с распределением χ^2 следующим образом: если θ распределено по закону Эрланга, то случайная величина 2θ распределена по закону χ^2 с $2k$ степенями свободы.

Таблица 24

№ вар.	$n=2k$	Результаты счета		Контрольные значения	
		$u\times xe\,[i-1]$	$sum1\times\exp(-u\times xe\,[i-1])$	x	p
1	8	0.1	0.999996154	0.2	1.00000
		0.3	0.999734188	0.6	0.99973
		0.5	0.998248377	1.0	0.99825
		0.7	0.994246543	1.4	0.99425
		1.9	0.874702214	3.8	0.87470
2	18	1.5	0.999972265	3.0	0.99997
		1.8	0.999892256	3.6	0.99989
		2.1	0.999662694	4.2	0.99966
		2.4	0.999138021	4.8	0.99914
		4.2	0.972067809	8.4	0.97207
3	20	7.0	0.830495937	14.0	0.83050
		7.25	0.804267616	14.5	0.80427
		7.50	0.776407612	15.0	0.77641
		7.75	0.747118789	15.5	0.74712
		9.25	0.554506897	18.5	0.55451

Поскольку в распоряжении авторов данного выпуска не было таблиц, соответствующих распределению Эрланга, то были использованы таблицы для интеграла вероятностей χ^2 [8, с. 204—210]. Для приведения табличных данных (контрольных значений) в соответствие с выходными данными алгоритма 184а кроме

$$p[i]=sum1\times exp(-u\times xe[i-1])-sum2\times exp(-u\times xe[i])$$

на печать выводились последовательные значения

$$sum1 \times exp(-u \times xe[i-1]) = \int_0^{x_{i-1}} f(x)\,dx$$

и соответствующие им значения k и $u \times xe[i-1]$.

Согласно вышесказанному табличным случайным значениям x соответствуют значения $2 \times u \times xe[i-1]$, а табличное n равно $2k$. Совпадение результатов счета с контрольными значениями показано в табл. 24.

Более подробно о распределении Эрланга см. в работе Жюль Мота [11].

АЛГОРИТМ 185б

Табулирование функции нормального распределения [S15]

Процедура *normal* вычисляет вероятность p попадания нормально распределенной случайной величины в интервал $[x_{i-1}, x_i]$ по формуле

$$p_i = \int_0^{x_i} f(x)\,dx - \int_0^{x_{i-1}} f(x)\,dx,$$

где $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-m)^2}{2\sigma^2}\right]$ — плотность вероятности, используя для аппроксимации нормальной функции распределения формулу Гастингса

$$\Phi(x_{ni}) = 0.5 - 0.5 \cdot (1 + a_1 x_{ni} + a_2 x^2_{ni} + a_3 x^3_{ni} + a_4 x^4_{ni} + a_5 x^5_{ni})^{-8},$$

где $a_1 = 0.09979268$, $a_2 = 0.04432014$, $a_3 = 0.00969920$, $a_4 = -0.00009862$ и $a_5 = 0.00058155$;

x_{ni} — нормализованные значения x_i; $x_{ni} = (x_i - m)/\sqrt{vars}$;

m — среднее значение;

$vars$ — дисперсия, уточненная с помощью поправки Шеппарда (см. [35, 36]);

c — число интервалов группировки;

p_i — вычисляемые значения вероятности; размерность массива $p[1:c+1]$;

x — массив значений x_i; размерность массива $x[1:c]$;

hastings(*xn*[*i*]) — обращение к процедуре-функции *hastings*, вычисляющей значение $\Phi(x)$ по формуле Гастингса.

```
procedure normal(x,m,vars,c,hastings)result:(p);
    value m,vars,c; real m,vars; integer c;
    array x,p; real procedure hastings;
begin integer i; array xn[1:c];
    for i:=1 step 1 until c do
      xn[i]:=(x[i]-m)/sqrt(vars);
    p[1]:=0.5+sign(xn[1])×hastings(abs(xn[1]));
    for i:=2 step 1 until c do
      p[i]:=(if xn[i]<0 then -1 else 1)×hastings(abs(xn[i]))
        -(if xn[i-1]<0 then -1 else 1)×hastings(abs(xn[i-1]));
```

```
    p[c+1] : =0.5+sign(xn[c])×hastings(abs(xn[c]))
end normal;
```

Свидетельство к алгоритму 185а

Алгоритм 185а получен в результате исправления, сокращения и ординарной переработки алгоритма 185 (Colker A. «CACM», 1963, № 7). В алгоритме 185 были обнаружены следующие ошибки:

1) параметры *m*, *vars* и *c*, вызываемые по значению, не были специфицированы;

2) перед символом **else** в двух местах алгоритма имелся символ «;».

3) вместо

$$p[1] := 0.5 - hastings(abs(xn[1]));$$

должно быть

$$p[1] := 0.5 + sign(xn[1]) \times hastings(abs(xn[1]));$$

4) вместо

$$p[c+1] := 0.5 - hastings(xn[c]);$$

должно быть

$$p[c+1] := 0.5 + sign(xn[c]) \times hastings(abs(xn[c]));$$

Алгоритм 185а был транслирован с параметрами $m=0$, $vars=1$, $c=4$ для сегментов [—3,—2.6], [—1,—0.5] и [2,2.4] и для сегмента [—0.5,0.5] при $c=10$. Результаты трансляции даны в табл. 25.

Таблица 25

x_i	p_i	Контрольные значения	x_i	p_i	Контрольные значения
—3.0	0.0013553	0.0014	2.0	0.977254	0.9772
—2.9	0.0005140	0.0005	2.1	0.004888	0.0049
—2.8	0.0006870	0.0007	2.2	0.003962	0.0040
—2.7	0.0009093	0.0009	2.3	0.003180	0.0032
	0.0034657	0.0035		0.989284	0.9893

Такое же хорошее совпадение вычисленных результатов с контрольными значениями было получено в остальных случаях. Контрольные значения были взяты из работы Е. С. Вентцель [14, табл. 1].

При этом для вычисления нормальной функции распределения по формуле Гастингса использовалась процедура

```
real procedure hastings(t);
    value t; real t;
begin real sum; integer j;
    sum : =1;
    for j : =1 step 1 until 5 do sum : =sum+t↑j×a[j];
    hastings : =(1—sum↑(—8))×0.5
end hastings;
```

Значение p_1, выдаваемое алгоритмом,— это вероятность попадания в интервал $(-\infty, x_1]$, а значение p_{c+1} — это вероятность попадания в интервал $(-\infty, c]$. В частности, для приведенных в табл. 25 интервалов $p[-2,6]=0.0034657$, а $p[2,4]=0.989284$. Контрольные значения соответственно равны 0.0035 и 0.9893 *.

* См. также подтверждение в [47, с. 131]. (*Прим. ред.*)

АЛГОРИТМ 186б

Комплексная арифметика [А2]

Данный алгоритм содержит систему процедур *p*, *q*, *s*, *t*, *v* и *u*, позволяющих записывать любое арифметическое выражение с комплексными переменными аналогично тому, как оно записывается с вещественными переменными языка АЛГОЛ-60. В конце алгоритма дается пример процедуры, вычисляющей значения некоторых двух выражений с комплексными переменными.

Очевидно, что эти процедуры легко расширить для использования в кватернионной арифметике или в общем векторном и тензорном исчислении.

Следует обратить внимание на то, что параметры процедур *p*, *q*, *s*, *u* вызываются по значению, а процедур *t* и *v* — только по наименованию; здесь это существенно. Кроме того, нужно заметить, что глубина (или высота) *n* массива-накопителя *h*[1 : *n*,1 : 2] равна максимальному числу расположенных подряд закрывающих скобок в обращениях к этим процедурам, если не считать при этом скобок, которые являются частями ограничителей параметров.

В алгоритме принято называть *k*-м элементом накопителя *h* пару переменных (*h*[*k*,1], *h*[*k*,2]), которым присваиваются значения вещественной и мнимой частей некоторого комплексного числа.

Изобразительные средства алгоритма 186а выходят за рамки сокращенного АЛГОЛа [6], поскольку в нем используются функции с побочными эффектами и выражения для параметров, вызываемых по имени.

Умножение элементов

Процедура-функция *p* присваивает (*k*+1)-му элементу накопителя *h* значение произведения *i*-го и *j*-го элементов, принимая при этом значение *k*+1.

```
integer procedure p(i,j);
      value i,j; integer i,j;
begin p := =k := =k+1;
      h[k,1] := =h[i,1]×h[j,1]—h[i,2]×h[j,2];
      h[k,2] := =h[i,1]×h[j,2]+h[i,2]×h[j,1]
end p;
```

Деление элементов

Процедура-функция *q* присваивает (*k*+1)-му элементу накопителя *h* значение частного от деления *i*-го элемента на *j*-й элемент, принимая при этом значение *k*+1.

```
integer procedure q(i,j);
      value i,j; integer i,j;
begin real b;
      q := =k := =k+1;
      b := =h[j,1]↑2+h[j,2]↑2;
      h[k,1] := =(h[i,1]×h[j,1]+h[i,2]×h[j,2])/b;
      h[k,2] := =(h[i,2]×h[j,1]—h[i,1]×h[j,2])/b
end q;
```

Сложение элемеитов

Процедура-функция *s* присваивает $(k+1)$-му элементу накопителя *h* значение суммы *i*-го и *j*-го элементов, принимая при этом значение $k+1$.

```
integer procedure s(i,j);
      value i,j; integer i,j;
begin s:=k:=k+1;
      h[k,1]:=h[i,1]+h[j,1];
      h[k,2]:=h[i,2]+h[j,2]
end s;
```

Запись переменной в накопитель

Процедура-функция *t* присваивает $(k+1)$-му элементу накопителя *h* значение комплексной переменной *a* (т. е. массива $a[1:2]$), принимая при этом значение $k+1$.

```
integer procedure t(a);
      array a;
begin t:=k:=k+1; h[k,1]:=a[1]; h[k,2]:=a[2] end t;
```

Формирование элемента накопителя

Процедура-функция *v* формирует $(k+1)$-й элемент накопителя *h*, присваивая его компонентам значение выражения *expi* и принимая при этом значение $k+1$.

```
integer procedure v(i,expi);
      real expi; integer i;
begin v:=k:=k+1;
      i:=1; h[k,1]:=expi;
      i:=2; h[k,2]:=expi
end v;
```

Выборка переменной из накопителя

Процедура *u* присваивает комплексной переменной *r* (т. е. массиву $r[1:2]$) значение *i*-го элемента, приводя *k* к нулю.

```
procedure u(i,r);
      value i; integer i; array r;
begin r[1]:=h[i,1]; r[2]:=h[i,2]; k:=0 end u;
```

Пример

Следующая процедура с помощью процедур комплексной арифметики вычисляет $rr=a^2+b^2$ и $r=(a+ib)/(a-ib)$, где *a*, *b*, *r* и *rr* — комплексные числа. Размерность массивов *a*, *b*, *r*, $rr[1:2]$.

```
procedure complex(a,b)result:(r,rr);
      array a,b,r,rr;
begin integer i,k; array h[1:4,1:2];
      k:=0;
      u(s(p(t(a),t(a)))plus:(p(t(b),t(b))),rr);
      comment Вычислено выражение rr=a^2+b^2;
      u(q(s(t(a))plus:(p(v(i,i-1),t(b))),
        s(t(a))plus:(p(v(i,1-i),t(b)))),r);
      comment Вычислено выражение r=(a+(i×b))/(a+(-i×b));
end complex;
```

Как указывает автор алгоритма 186, описание этого алгоритма опубликовано в работе [14i].

Свидетельство к алгоритму 186б

Алгоритм 186б получен из алгоритма 186а в результате некоторых сокращений записи процедур *p* и *q*. Вследствие элементарности алгоритма и очевидности сделанных в нем изменений контрольного решения на машине не проводилось.

Свидетельство к алгоритму 186а

Алгоритм 186а получен в результате ординарной переработки и модификации алгоритма 186 (Van de Riet R. P. «CACM», 1963, № 7). Модификация состояла в отделении процедур интервальной арифметики от процедуры *complex*, данной в качестве примера их применения.

В алгоритме 186 были исправлены ошибки, состоящие в том, что в процедурах *p*, *q*, *s*, *t*, *v* вместо оператора $k := k+1$ использовался оператор $k := k-1$.

Алгоритм 186а был проверен вручную для вышеприведенного примера.

АЛГОРИТМ 187б

Разности и производные (рекурсивные процедуры) [E1]

Нижеприведенная программа вычисляет (с целью демонстрации применения процедур *delta* и *der*) третью производную от экспоненциальной функции e^x по схеме разностей шестого порядка для 50 значений аргумента $x = 1/50, 2/50, \ldots, 1$.

Как было проверено на машине X1 Математического центра, эти программы работают медленно вследствие явно выраженной рекурсивности. В реальном программировании обычно берут на себя труд расписывать хорошо известную формулу Грегори. Если же пользоваться процедурами *delta* и *der*, то этот труд перекладывается на машину. Но главное назначение данного алгоритма состоит в демонстрации гибкости языка АЛГОЛ-60 при пользовании свойством рекурсивности процедур.

delta — процедура-функция, вычисляющая прямую разность *n*-го порядка по данной последовательности значений функции $f(k)$ с равноотстоящими значениями аргумента.

der — процедура-функция, вычисляющая производные порядка *or* по данной последовательности значений функции $f(k)$ с равноотстоящими значениями аргумента. Теоретически погрешность будет порядка $h \uparrow (n+1-or)$, где h — длина шага по аргументу (согласно [14i]).

*k*0 — точка, в которой вычисляется производная.

В этих процедурах используется вызов по наименованию выражений, и, следовательно, они выходят за рамки сокращенного АЛГОЛа [6] и в этом отношении.

```
begin real h; integer i,k; array a[1 : 50];
      real procedure sum(i,h,k,ti);
        value k; real ti; integer i,k,h;
        begin real s;
          s : =0;
          for i : =h step 1 until k do s : =s+ti;
```

```
        sum := s
      end sum;
    real procedure delta(n,k,k0,fk);
      value n,k0; real fk; integer n,k,k0;
      begin integer i;
        delta := if n=1 then
          sum(k,k0,k0+1,(-1)↑(k+1-k0)×fk) else
          delta(1,i,k0,delta(n-1,k,i,fk))
      end delta;
    real procedure der(or,n,h,k,k0,fk);
      value or,n,h,k0; real fk,h; integer or,n,k,k0;
      begin integer i;
        der := if or=1 then
          sum(i,1,n,delta(i,k,k0,fk)×(-1)↑(i+1)/i)/h else
          der(1,n+1-or,h,i,k0,der(or-1,n-1,h,k,i,fk))
      end der;
start: for i := 1 step 1 until 50 do a[i]=exp(i/50);
    for i := 1 step 1 until 25 do
      a[i] := der(3,6,0.02,k,i,a[k]);
      outarray(1,a)
end
```

Свидетельство к алгоритму 187б

Алгоритм 187б отличается от алгоритма 187а только тем, что в пояснительные тексты к нему внесены некоторые изменения редакторского характера.

Алгоритм 187б был транслирован на машине БЭСМ-6 в системе БЭСМ-АЛГОЛ, и были получены правильные результаты с погрешностями от 4.6×10^{-5} (получено $a[7]=1.150227301$ вместо 1.150273799) до 30×10^{-5} (получено $a[18]=1.433632378$ вместо 1.433329415). Общее время вычисления 25 значений производной было 18 мин 28 с.

Свидетельство к алгоритму 187а

Алгоритм 187а получен в результате ординарной переработки алгоритма 187 (Van de Riet R. P. «CACM», 1963, № 7) и добавления оператора вывода результатов в конце программы.

АЛГОРИТМ 188б

Сглаживание по трем точкам [E3]

Процедура *smooth*13 (*smooth* — сглаживать) использует трехточечные формулы Грама (Gram) первой степени, описанные Гилдебрандом в работе [12i], для сглаживания последовательности n значений функции (при равноотстоящих значениях аргумента), записанных в массиве $x[1:n]$. Если обращение к процедуре производится с $n<3$, то происходит выход к глобальной метке *signal*188. Более подробно о сглаживании см. в работе В. Милна [13, с. 209].

```
procedure smooth13(n)dataresult:(x);
    value n; integer n; array x;
begin real a,b,c; integer i;
```

```
    if n<3 then go to signal188;
    a:=x[1]; c:=x[n—2];
    x[1]:=(5×a+2×x[2]—x[3])/6;
    for i:=2 step 1 until n—1 do
      begin b:=x[i]; x[i]:=(a+b+x[i+1])/3; a:=b end;
    x[n]:=(—c+2×b+5×x[n])/6
end smooth13;
```

Свидетельство к алгоритму 188а *

Алгоритм 188а получен в результате усовершенствования алгоритма 188 (Rodriguez-Gil F. «CACM», 1963, № 7), которое заключалось в том, что массив $xp[1:n]$ был заменен простыми переменными a, b и c для экономии памяти машины, а периодические дроби, представляющие коэффициенты, были заменены простыми дробями для обеспечения независимости алгоритма от конкретной машины.

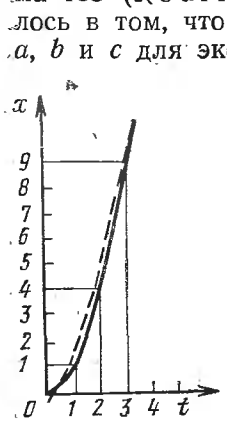

Рис. 4.

Алгоритм 188а был транслирован для $x=(0,1,4,9,16)$, т. е. «сглаживанию» подвергалась функция $x=t^2$ для $t=(0,1,2,3,4)$. Были получены результаты (—0.333333333, 1.66666666, 4.66666666, 9.66666666, 15.6666666), которым на рис. 4 соответствует штриховая линия.

Точно такие же результаты были получены после трансляции алгоритма 188. Для $t=(-4,-3,-2,-1,0)$ были получены результаты, симметричные вышеприведенным. Для $t=(-2,-1,0,1,2)$ было получено (3.66666666, 1.66666666, 0.666666666, 1.66666666, 3.66666666). При «сглаживании» функции $x=t^3$ для $t=(0,1,2,3,4)$ было получено (—1.00000000, 3.00000000, 12.0000000, 33.0000000, 61.0000000).

АЛГОРИТМ 189б

Сглаживание по пяти точкам [E3]

Процедура *smooth*35 (*smooth* — сглаживать) использует пятиточечные формулы Грама (Gram) третьей степени, описанные Гилдебрандом в работе [12i], для сглаживания последовательности n значений функции (при равноотстоящих значениях аргумента), записанных в массиве $x[1:n]$. Если обращение к процедуре производится с $n<5$, то происходит выход к глобальной метке *signal*189.

Более подробно о сглаживании см. в работе В. Э. Милна [13, с. 299].

```
procedure smooth35(n)data:(x)result:(x);
      value n; integer n; array x;
begin real a,b,c,s,t; integer i;
      if n<5 then go to signal189;
      a:=x[1]; b:=x[2];
      s:=x[n—4]; t:=x[n—1];
```

* См. также подтверждение в [47, с. 131]. (*Прим. ред.*)

```
      x[1] := (69×a+4×b−6×x[3]+4×x[4]−x[5])/70;
      x[2] := (2×a+27×b+12×x[3]−8×x[4]+2×x[5])/35;
      for i := 3 step 1 until n−2 do
        begin c := x[i];
          x[i] := (−3×a+12×b+17×c+12×x[i+1]−3×x[i+2])/35;
          a := b; b := c
        end i;
      x[n−1] := (−8×a+12×b+2×s+27×t+2×x[n])/35;
      x[n] := (4×a−6×b−s+4×t+69×x[n])/70
end smooth35;
```

Свидетельство к алгоритму 189а *

Алгоритм 189а получен в результате усовершенствования алгоритма 189 (Rodriguez-Gil F. «CACM», 1963, № 7), заключавшегося в том, что массив $xp[1:n]$ был заменен переменными a, b, c, s и t для экономии памяти, а периодические дроби, представляющие коэффициенты, были заменены простыми дробями для обеспечения независимости алгоритма от конкретной машины.

Алгоритм 189а был транслирован для случая «сглаживания» функции $x=t^2$ при $t=(0,1,2,3,4)$, и были получены результаты ($0.363767881\times\times10^{-11}$, 0.999999999, 4.00000000, 9.00000000, 16.0000000). При «сглаживании» функции $x=t^3$ для $t=(0,1,2,3,4)$ были получены точные значения функции (0, 1, 8, 27, 64). Такие же результаты были получены для алгоритма 189.

АЛГОРИТМ 190б

Комплексная степень комплексного числа [B4]

Процедура *compow* (*complex* — комплексный, *power* — степень) вычисляет $x+iy=(a+ib)\uparrow(c+id)$ по формуле $x+iy=e^{cr-dp}(cosv+isinv)$, где $v=cp+dr$, $r=\ln\sqrt{a^2+b^2}$, $i=\sqrt{-1}$. В комплексной области, разрезанной вдоль действительной оси от 0 до $-\infty$, значение p, вычисляемое в теле данной процедуры, — это сумма главного значения аргумента числа $(a+ib)$ и числа $2\pi n$, где n — целое число, задаваемое в зависимости от условий задачи. При анализе алгоритма нужно помнить, что $-\pi/2\leqslant\arctan(x)\leqslant\pi/2$. В процедуре использована константа $pi=3.14\ldots$

Общая степенная функция описывается, например, в работе М. А. Лаврентьева и Б. В. Шабата [12].

```
procedure compow(a,b,c,d,n)result:(x,y);
      value a,b,c,d,n; real a,b,c,d,x,y; integer n;
begin real p,r,v,w,pi;
      x := y := 0; pi := 3.14159265359;
      if a=0∧b=0 then go to fin;
      p := 2×pi×n+
        (if a=0 then sign(b)×pi/2 else arctan(b/a)+
          (if a>0 then 0 else
          (if b≥0 then pi else −pi)));
```

---

* См. также подтверждение в [47, с. 131]. (*Прим. ред.*)

```
    r : =0.5×ln(a×a+b×b);
    v : =c×p+d×r; w : =exp(c×r—d×p);
    x : =w×cos(v); y : =w×sin(v);
fin: end compow;
```

Свидетельство к алгоритму 190а *

Алгоритм 190а получен в результате сокращения и ординарной переработки алгоритма 190 (Relph A. P. «CACM», 1963, № 7).

Результаты трансляции алгоритма 190а для $n=0$ приведены в табл. 26.

Таблица 26

$a+bi$	$c+di$	Контрольные значения		Результаты трансляции	
		x	y	x	y
$-2-2i$	$1-i$	—0.259455916	0.0674459475	—0.259455916	0.0674459477
$-2+i$	i	0.0476335649	0.0495107027	0.0476335649	0.0495107028
$-1-i$	$1-i$	—0.121339466	—0.0569501185	—0.121339466	—0.0569501185
-1	$-1-i$	—23.1406925	-0.1036746×10^{-6}	—23.1406925	-0.8357444×10^{-7}
$-2i$	$-1-i$	0.0664135000	0.0799545287	0.0664135000	0.0799545285
$-i$	$1+i$	0.8790207×10^{-8}	—4.81047736	0.1295031×10^{-7}	—4.81047737
i	$-1+i$	0.3781307×10^{-9}	—0.207879577	0.5596335×10^{-9}	—0.207879576
$2i$	$1+i$	—0.265653998	0.319818115	—0.265653998	0.319818115
$1-i$	$-1-i$	0.291850379	0.136978627	0.291850379	0.136978627
1	$1+i$	0.999999999	0.8731149×10^{-9}	1.000000000	0.000000000
$2-2i$	$-1+i$	—0.195093245	0.750498700	—0.195093245	0.750498696
$2+i$	$1-i$	3.35025931	—1.18915021	3.35025932	—1.18915022

Контрольные значения в табл. 26 были получены по формуле $(a+bi)^{(c+di)}=e^{(c+di)ln(a+bi)}$, где вычисление экспоненты и логарифма комплексного числа производилось с помощью алгоритмов 46а и 48а соответственно. В алгоритм 48а предварительно было внесено исправление, указанное в «Замечании и подтверждении к алгоритму 48а», приведенному в приложении 1 к выпуску [50].

АЛГОРИТМ 191б

Гипергеометрическая функция [S22]

Процедура *hypergeom* вычисляет гипергеометрическую функцию $s1+i\times s2={}_2F_1(a,b;c;z)$ с комплексными параметрами ($a=a_1+ia_2$, $b=b_1+ib_2$ и т. д.). В теле этой процедуры локализована процедура *compmult* (сокращение от *complex* — комплексный и *multiplication* — умножение), которая вычисляет произведение $c_1+ic_2=(a_1+ia_2)(b_1+ib_2)$. Если прибавление последующего члена ряда к сумме не влияет на предыдущее значение суммы, то вычисление заканчивается и параметр *corr* (сокращение от *correct* — правильный) получает значение **true**. Если же после *max* итераций такая точность не достигается, то процедура заканчивается со значением параметра *corr*≡**false**.

Более подробно о гипергеометрических рядах см. в работе В. В. Голубева [11] или в работе Янке и др. [8].

---

* Алгоритм 190а был подтвержден также в первом выпуске [47, с. 132]. (*Прим. ред.*)

```
procedure hypergeom(a1,a2,b1,b2,c1,c2,z1,z2,max)
      result:(s1,s2,corr);
      value a1,a2,b1,b2,c1,c2,z1,z2,max;
      real a1,a2,b1,b2,c1,c2,z1,z2,max,s1,s2; Boolean corr;
begin real d,y1,y2; integer n;
      procedure compmult(a1,a2,b1,b2,c1,c2);
        value a1,a2,b1,b2; real a1,a2,b1,b2,c1,c2;
        begin c1 := =a1×b1—a2×b2; c2 := =a2×b1+a1×b2
        end compmult;
start: s1 := =y1 := =1; s2 := =y2 := =0; corr := = true;
      for n := =1 step 1 until max do
        begin d := =n×((c1+n—1)↑2+c2↑2);
          compmult(a1+n—1,a2,y1/d,y2/d,y1,y2);
          compmult(y1,y2,b1+n—1,b2,y1,y2);
          compmult(y1,y2,c1+n—1,—c2,y1,y2);
          compmult(y1,y2,z1,z2,y1,y2);
          if s1=s1+y1∧s2=s2+y2 then go to fin;
          s1 := =s1+y1; s2 := =s2+y2;
        end n;
      corr := = false;
fin: end hypergeom;
```

Свидетельство к алгоритму 191a

Алгоритм 191a получен в результате ординарной переработки и модификации алгоритма 191 (Relph A. P. «CACM», 1963, № 7). Модификация состояла в обеспечении сигнализации о правильности результата путем присваивания дополнительному параметру *corr* значения **false** в том случае, когда процедура заканчивается по достижении предельного числа итераций *max* и, следовательно, выдает результат без гарантированной точности. Кроме того, предельное число итераций 100 было заменено дополнительным параметром *max* во избежание необходимости внесения изменений в тело процедуры в случае, когда требуется более 100 итераций.

Перевод «Подтверждения к алгоритмам 191 и 192» см. ниже вслед за алгоритмом 192б.

Алгоритм 191a был транслирован для следующих вариантов обращения к процедуре hypergeom:

I. *hypergeom*(0.5,0,0.5,0,1,0,k^2,0,30000,*s1*,*s2*,*corr*),

т. е. вычислялась ${}_2F_1(0.5,0.5;1;k^2)=2/\pi K(k)$, где $k=\sin\alpha=\sin(0.01745329252\alpha^\circ)$ и α° выражены в градусах.

Таблица 27

α° град.	Вариант I			Вариант II		
	s1	$(\pi/2)\,s1$	$K(k)$	*s1*	$(\pi/2)\,s1$	$E(k)$
0	1.00000000	1.57079633	1.57079633	1.00000000	1.57079633	1.570796327
30	1.07318200	1.68575035	1.68575035	0.934215458	1.46746221	1.467462209
60	1.37288050	2.15651565	2.1565156	0.770082213	1.21105603	1.21110560
80	2.00750739	3.15338524	3.1534	0.662157391	1.04011439	1.0401
85	2.43936269	3.83174196	3.8317	0.644681619	1.01266352	1.0127
89	3.45962565	5.43436728	5.4349	0.637172194	1.00086774	1.0008
90	4.34772536	6.82939103	∞	0.636778867	1.00024990	1.0000

Таблица 28

x	Вариант III		Вариант IV			
	$s1$	$s1\,(1-x)$	x	$s1$	$s1 \times x$	$\ln(1+x)$
0	1.00000000	1.000000000	0.1	0.953101798	0.0953101798	0.0953101798
0.1	1.11111111	1.000000000	0.2	0.91160784	0.182321556	0.182321557
0.2	1.25000000	1.000000000	0.3	0.874547548	0.262364264	0.262364264
0.3	1.42857142	0.999999999	0.4	0.841180591	0.336472237	0.336472237
0.4	1.66666666	0.999999999	0.5	0.810930216	0.405465108	0.405465108
0.5	2.00000000	1.000000000	0.6	0.783339382	0.470003629	0.470003629
0.6	2.49999999	1.000000000	0.7	0.758040359	0.530628251	0.530628251
0.7	3.33333333	0.999999999	0.8	0.734733331	0.587786665	0.587786665
0.8	4.9999999_9	0.999999999	0.9	0.713170984	0.641853886	0.641853886
			1.0	0.693163845	0.693163845	0.693147180

II. *hypergeom*(0.5,0,—0.5,0,1,0,k^2,0,1000,*s1*,*s2*,*corr*),
т. е. вычислялась ${}_2F_1(0.5,-0.5;1;k^2)=2/\pi E(k)$, где $k=\sin\alpha$.
III. *hypergeom*(1,0,1,0,1,0,*x*,0,100,*s1*,*s2*,*corr*),
т. е. вычислялась ${}_2F_1(1,1;1;x)=1/(1-x)$.
IV. *hypergeom*(1,0,1,0,2,0,—*x*,0,100,*s1*,*s2*,*corr*),
т. е. вычислялась ${}_2F_1(1,1;2;-x)=\ln(1+x)/x$.
V. *hypergeom*(—*n*,0,1,0,1,0,—*x*,0,100,*s1*,*s2*,*corr*),
т. е. вычислялась ${}_2F_1(-n,1;1;-x)=(1+x)^n$.

Результаты трансляции даны в табл. 27—29.

Таблица 29

x	Вариант V					
	$n=1$		$n=5$		$n=10$	
	$s1$	$(1+x)^n$	$s1$	$(1+x)^n$	$s1$	$(1+x)^n$
1	2	2	31.9999847	32.0000	1024.0000	1024.0000
5	6	6	7775.99630	7776.00	60466176.0	60466176.0
10	11	11	161050.921	161051	25937424600	25937424601

Контрольные значения в табл. 27 взяты из работ [8, 39, 40].
Контрольные значения $\ln(1+x)$ в табл. 28 взяты из работы [43].
Контрольные значения $(1+x)^n$ в табл. 29 взяты из работы [44].
Для всех случаев во всех вариантах было *corr*≡ **true** и *s2*=0*.

Подтверждение и замечание к алгоритму 191

Х. Копеляр (Koppelaar H. «CACM», 1974, № 10)

В алгоритме были сделаны следующие изменения.

1. Была исключена процедура *compmult*.

2. В соответствии с предыдущим пунктом и со стандартным обозначением ${}_2F_1$ начало и конец процедуры были заменены на

real procedure hyp2geom1 (a,b,c,z);
end hyp2geom1;

* См. также подтверждение в [47, с. 131]. (*Прим. ред.*)

3. Исключение процедуры *compmult* привело к следующей модификации алгоритма:

```
real procedure hyp2geom1(a,b,c,z);
    value a,b,c,z; real a,b,c,z;
begin real s,y; integer i;
    s:=y:=1;
    for i:=0 step 1 until 100 do
      begin y:=y×((a+i)/(c+i))×(b+i)×z/(i+1);
        if s=s+y then go to exit;
        s:=s+y
      end;
exit: hyp2geom1:=s
end hyp2geom1;
```

Неэффективность исходного алгоритма для вещественных аргументов, указанная Г. Тачером в его «Подтверждении к алгоритмам 191 и 192»*, этими модификациями в значительной степени снижается, поскольку они существенно уменьшают затраты машинного времени.

С этими модификациями алгоритм был транслирован на машине CDC-6500 при использовании транслятора Control Data Algol 3 и дал результаты, как это указано ниже, удовлетворительные только отчасти.

Были выполнены два следующих теста (а) и (б) с использованием тождеств 1 и 2 и алгоритма 160 («CACM», 1963, № 4) ** для вычисления $C^n{}_m$. Эти тождества были следующими:

1) $(1/B(a,b))(z^a/a)\,{}_2F_1(a,1-b;a+1;z)=I_z(a,b)$,

2) $a\times B(a,b)=1/C^a{}_{a+b-1}$, где

$$B(a,b)=\int_0^1 t^{a-1}(1-t)^{b-1}\,dt$$

полная бета-функция и

$$I_z(a,b)=(1/B(a,b))\int_0^z t^{a-1}(1-t)^{b-1}\,dt$$

неполная бета-функция, $a\geqslant 1$, $b\geqslant 1$, a и b целые.

Тест (а). Вычисление $C^a{}_{a+b-1}\times$*hyp2geom1*$(a,1-b,a+1,z)\times z\uparrow a$ дало результаты, правильные (согласно таблицам Пирсона [17i]) в семи цифрах для следующих значений a, b и z:

$b=7$, $a=7,8,9$, $z=0.61\,(0.01)\,0.97$;

$b=17$, $a=17$, $z=0.13$;

$b=17$, $a=17,18$, $z=0.14$;

$b=17$, $a=17,18,19$, $z=0.15$;

$b=17$, $a=17\,(1)\,34$, $z=0.50$.

Общее время вычисления этого теста на машине Control Data 6500 было меньше 10 с.

Тест (б). Такой же тест, как и тест (а), был выполнен для следующих значений a, b и z: $b=17$, $a=17\,(1)\,34$ и $z=0.51\,(0.01)\,0.60$. Алгоритм дал результаты, правильные (согласно [17i]) в пяти цифрах.

---

* Перевод этого подтверждения см. ниже. (*Прим. ред.*)

** См. соответствующий алгоритм 160б. (*Прим. ред.*)

Для значений $b=17$, $a=17(1)34$ и $z=0.61(0.01)0.89$ результаты ухудшались с увеличением z. Эта погрешность вызывается ухудшением сходимости ряда

$$ {}_2F_1(a, b; c; z) = \sum_{n=0}^{\infty} (a)_n (b)_n / (c)_n z^n / n! $$

при возрастании z и при возрастании b. Символ $(a)_n$ означает

$$ (a)_n = \Gamma(a+n)/\Gamma(a). $$

Более точно

$$ (1/B(a, b))(z^a/a)\,{}_2F_1(a, 1-b; a+1; z) = $$

$$ = \sum_{n=0}^{\infty} C^a{}_{a+b-1} ((a)_n (1-b)_n / (a+1)_n) z^{n+a} / n! $$

Из этого выражения видно, что скорость сходимости этого разложения, вообще говоря, зависит главным образом от значений b и z. Это объясняет, почему в тесте (а) для $b=7$ результаты при возрастании z оставались точными и почему в тесте (б) для $b=17$ результаты при возрастании z ухудшались.

АЛГОРИТМ 192б

Конфлюентная гипергеометрическая функция [S22]

Процедура *confhypergeom* вычисляет конфлюентную (вырожденную) гипергеометрическую функцию $s1+i\times s2={}_1F_1(a,c;z)$ с комплексными параметрами ($a=a1+i\times a2$ и т. д.). В теле этой процедуры локализована процедура *compmult*, вычисляющая произведение двух комплексных чисел $c1+i\times c2=(a1+i\times a2)(b1+i\times b2)$. Если прибавление следующего члена гипергеометрического ряда к сумме не влияет на предыдущее значение суммы, то вычисление заканчивается и параметр *corr* получает значение **true**. Если же после *max* итераций такая точность не достигается, то процедура заканчивается со значением параметра *corr*= **false**.

Более подробно о гипергеометрических рядах см. в работе В. В. Голубева [11] или в работе Янке и др. [8].

```
procedure confhypergeom(a1,a2,c1,c2,z1,z2,max)
        result:(s1,s2,corr);
        value a1,a2,c1,c2,z1,z2,max;
        real a1,a2,c1,c2,z1,z2,max,s1,s2; Boolean corr;
begin real d,y1,y2; integer n;
        procedure compmult(a1,a2,b1,b2,c1,c2);
          value a1,a2,b1,b2; real a1,a2,b1,b2,c1,c2;
          begin c1 := =a1×b1−a2×b2;
            c2 := =a2×b1+a1×b2
          end compmult;
start: s1 := =y1 := =1; s2 := =y2 := =0; corr := = true;
        for n := =1 step 1 until max do
          begin d := =n×((c1+n−1)↑2+c2↑2);
            compmult(a1+n−1,a2,y1/d,y2/d,y1,y2);
            compmult(y1,y2,c1+n−1,−c2,y1,y2);
            compmult(y1,y2,z1,z2,y1,y2);
```

```
        if s1=s1+y1∧s2=s2+y2 then go to fin;
        s1 : =s1+y1; s2 : =s2+y2
      end n;
      corr : = false;
fin: end confhypergeom;
```

Свидетельство к алгоритму 192а

Алгоритм 192а получен в результате исправления, модификации и ординарной переработки алгоритма 192 (Relph A. P. «CACM», 1963, № 7). Была исправлена одна опечатка в заголовке цикла, который в алгоритме 192 имел вид

for n : = **step** —1 **until** 100 **do**,

и сделана такая же модификация, как в алгоритме 191а.

Алгоритм 192а был транслирован при $a1=1$, $a2=0$, $c1=1$, $c2=0$, $z1=x$, $z2=0$ и $max=1000$, т. е. вычислялась функция ${}_1F_1(1,1;x)=e^x$. Результаты трансляции даны в табл. 30.

Таблица 30

x	$s1$	e^x	x	$s1$	e^x
—10	0.454002824×10^{-4}	$0.4539992976\times10^{-4}$	0.5	1.64872127	1.648721271
—5	0.673791656×10^{-2}	$0.6737947000\times10^{-2}$	1	2.71828182	2.718281829
—1	0.367879441	0,3678794412	2	7.38905610	7.389056099
—0.5	0.606530689	0.6065306597	5	148.413158	148.4131591
0	1.00000000	1.00000000	10	22026.4658	22026.46579

Во всех случаях было $corr\equiv$ **true**. Контрольные значения взяты из [41].

Затем была вычислена функция Лагерра

$$l_n(x)=e^{-x/2}L_n(x)=e^{-x/2}{}_1F_1(-n,1;x),$$

для чего использовалось обращение к процедуре вида

$$confhypergeom(-n,0,1,0,x,0,1000,s1,s2,corr).$$

Результаты трансляции даны в табл. 31.

Таблица 31

x	$l_0(x)$		$l_5(x)$	
	Вычисленное	Контрольное	Вычисленное	Контрольное
0	1.00000000	1.0000	1.00000000	1.0000
0.5	0.778800763	0.7788	—0.347010576	—0.3470
1.0	0.606530650	0.6065	—0.283047641	—0.2830
2.0	0.367879440	0.3679	0.269778256	0.2698
5.0	0.820840986×10^{-1}	0.8208×10^{-1}	—0.259935829	—0.2599
20.0	0.453999897×10^{-4}	0.4540×10^{-4}	—0.216369971	—0.2164

Для всех случаев было $corr\equiv$ **true**. Контрольные значения в табл. 31 взяты из работы Янке и др. [8].

Наконец, алгоритм 192а был транслирован для случая обращения к процедуре

$$confhypergeom(0.5,0,1,0,0,2\times x,max,s1,s2,corr)$$

при max=100, т. е. вычислялась функция

$$_1F_1(0.5,1;\ i\times 2x)=e^x I_0(x)=I_0(x)\cos x+iI_0(x)\sin x$$

(см. работу Л. Дж. Слейтера [15], с. 13). Результаты трансляции даны в табл. 32, где $r1=s1/\cos x$, $r2=s2/\sin x$.

Таблица 32

x	$s1$	$s2$	$r1$	$r2$	I_0
0	1	0	1	Не определено	1
1	0.413438074	0.643891651	0.765197680	0.765197680	0.7651977
2	—0.0931714395	0.203583309	0.223890779	0.223890779	0.2238908
3	0.257449484	—0.0366985341	—0.260051954	—0.260051954	—0.2600520
4	0.259594439	0.300563967	—0.397149809	—0.397149810	—0.3971498
5	—0.0503774946	0.170301862	—0.177596793	—0.177596779	—0.1775968
6	0.144645163	—0.0420926169	—0.150645323	0.150645247	0.1506453
7	0.226230290	0.197148867	0.300079074	0.300080500	0.3000793
8	—0.0249719439	0.169822436	0.171628440	0.171649083	0.1716508
9	0.0822951090	—0.0372383562	—0.0903220016	—0.0903583740	—0.0903336
10	0.206158562	0.133744797	—0.245698435	—0.245844866	—0.2459358
11	—0.000818276091	0.171498298	—0.184891951	—0.171499977	—0.1711903
12	0.0369388614	—0.0174281717	0.0437739979	0.0324805281	0.0476896
13	0.194693562	0.0374090314	0.214550943	0.0890337130	0.2069213

Для всех случаев $corr\equiv$ **true**. Контрольные значения I_0 взяты из работы [42].

Подтверждение к алгоритмам 191 и 192

Г. Тачер (Thacher H. C. «CACM», 1964, № 4)

Тела этих двух процедур были проверены на трансляторе Dartmouth SCALP вычислительной машины LGP-30. Не было обнаружено никаких синтаксических ошибок, и программы дали результаты, совпадающие до семи десятичных цифр с табличными значениями для следующих специальных случаев: $_2F_1(0.5,0.5;1;k^2)=(2/\pi)K(k)$, $_2F_1(0.5,\ -0.5;1;k^2)=(2/\pi)E(k)$, $_1F_1(0.5,1;iy)=J_0(x)$, $_1F_1(-1,0.1;x)$, $_1F_1(-0.5,0.1;x)$ и $_1F_1(-0.5,0.5;x)$, где K и E — полные эллиптические интегралы первого и второго рода.

Следует заметить, что функция, вычисляемая алгоритмом 191, есть $_2F_1(a,b;c;z)$, а не $_1F_2(a,b;c;z)$, как это утверждается в примечании к процедуре. Эти программы вычисляют функции путем прямого суммирования гипергеометрического ряда. Следовательно, они являются сравнительно общими, но не эффективными. Нужно также предостеречь против попыток вычислений вне интервала эффективной сходимости ряда.

АЛГОРИТМ 193б

Обращение степенного ряда [C1]

Процедура *serirevers* (сокращение от *series* — ряд, *reversion* — обращение) выдает коэффициенты $b[i]$ ряда

$$x=y+\sum_{i=2}^{n} b[i]\times y\uparrow i,$$

если заданы коэффициенты $a[i]$ ряда

$$y = x + \sum_{i=2}^{n} a[i] \times x \uparrow i.$$

Процедура пользуется последовательными приближениями по формуле

$$y_{p+1} = x - \sum_{i=2}^{p+2} b[i] \times y_p \uparrow i,$$

где $p=0, 1, \ldots, n-2$ и $y_0=x$. Размерность массивов: $a[1:n]$, $b[0:n]$.

```
procedure serirevers(a,n)result:(b);
      value n; integer n; array a,b;
begin real s; integer i,j,k,m; array q,r[0:n];
      a[1]:=b[0]:=0; b[1]:=1;
      for k:=2 step 1 until n do
        begin b[k]:=0;
          for i:=0 step 1 until k do r[i]:=0;
          for j:=k step -1 until 1 do
            begin q[0]:=r[0]-a[j];
              for i:=1 step 1 until k do q[i]:=r[i];
              for i:=0 step 1 until k do
                begin s:=0;
                  for m:=0 step 1 until i do
                    s:=s+b[m]×q[i-m];
                  r[i]:=s
                end i
            end j;
          for i:=2 step 1 until k do b[i]:=r[i]
        end k
end serirevers;
```

Свидетельство к алгоритму 193б

Алгоритм 193б получен из алгоритма 193а в результате модификации, имеющей целью ускорение выполнения процедуры и заключающейся в том, что первый заголовок цикла **for** k:=1 **step** 1 **until** n−1 **do** был заменен заголовком **for** k:=2 **step** 1 **until** n **do** и всюду в теле процедуры $k+1$ было заменено на k.

Алгоритм 193б был транслирован на машине БЭСМ-6 в системе БЭСМ-АЛГОЛ, и с ним было успешно повторено решение примера, приведенного в нижеследующем «Свидетельстве к алгоритму 193а». Были заданы $n=14$ и $a[i]=1/i!$ для $i=1, 2, \ldots, 14$. С точностью до одной единицы последней (т. е. десятой) значащей цифры были получены $b[i]=(-1)^{i+1}/i$, являющиеся коэффициентами разложения в ряд функции $x=\ln(1+y)$ [18, с. 327].

Свидетельство к алгоритму 193а

Алгоритм 193а получен в результате ординарной переработки алгоритма 193 (Fettis H. E. «CACM», 1963, № 7). В отличие от алгоритма 193 массив a был исключен из списка значений для экономии машинного времени.

Алгоритм 193а был транслирован для случая вычисления первых 14 коэффициентов ряда $\ln(1+y)$ по первым четырнадцати коэффициентам ряда $y=e^x-1$. Результаты точно совпали с теоретическими значениями этих коэффициентов.

Подтверждение к алгоритму 193

Г. Тачер (Thacher H. C. «CACM», 1963, № 12)

Алгоритм 193 был проверен на машине LGP-30 с использованием АЛГОЛ-транслятора, разработанного Вычислительным центром дартмутского колледжа. Не было обнаружено никаких синтаксических ошибок. Программа успешно вычислила первые четыре коэффициента ряда $\ln(1+y)$ по первым четырем коэффициентам ряда $y=e^x-1$.

АЛГОРИТМ 194б

Корни решения системы дифференциальных уравнений [D2]

Процедура *zersol* (сокращение от *zero* — корень, *solution* — решение) находит простые корни решения $y_i(y_0)$ системы m дифференциальных уравнений первого порядка

$$y'_j=f_j(y_0, y_i, \ldots, y_m),$$

интегрируемой по методу Рунге — Кутта с шагом h.

Параметры процедуры:

h — шаг интегрирования.

eps — допустимая погрешность в значении искомых корней (без учета погрешностей округления).

f(*ys*,*j*,*v*) — процедура, вычисляющая правые части уравнений (т. е. значение функции f_j). Входные параметры процедуры *f* — это номер уравнения *j* и массив *ys*. Выходной параметр — *v*.

mr — матрица размерностью [1 : 4,1 : 4], содержащая коэффициенты, соответствующие методу Рунге — Кутта. Например, *mr* может содержать следующие значения: 1/2, 1, 1/2, 0, $1-\sqrt{2}$, $1-\sqrt{2}$, $1-\sqrt{2}$, 1/2, $1+\sqrt{2}$, $1+\sqrt{2}$, $1+\sqrt{2}$, 0, 1/6, 1/3, 1/2, 1/2.

m — число уравнений и искомых функций.

yi — массив размерностью [0 : *m*]. Перед выполнением процедуры *zersol* этот массив должен содержать начальные значения функций, а после выполнения в нем содержатся конечные значения искомых функций.

ff — значение, задаваемое пользователем. Поиск корней прекращается, как только станет $yi[0]>ff$.

z — массив, содержащий найденные корни. Размерность $z[1:nn]$, где *nn* должно быть не меньше предполагаемого числа корней.

```
procedure zersol(h,m,eps,f,ff,mr)dataresult:(yi)result:(z);
    value h,m,eps,ff; real h,eps,ff; integer m;
    array yi,z,mr; procedure f;
begin real v,r,d; integer j,s,n,k;
    array q,yt[1 : m],ys,ya[0 : m];
    n:=1;
```

```
        for d := =h while yi[0] ≤ff do
          begin s := =1;
r1:         for j := =1 step 1 until m do
              begin q[j] := =0.0; ys[j] := =yt[j] := =yi[j] end j;
            ys[0] := =yi[0];
r2:         for k := =1 step 1 until 4 do
              begin ys[0] := =ys[0]+mr[k,4]×d;
                for j := =1 step 1 until m do
                  begin f(ys,j,v); v := =v×d;
                    r := =mr[k,1]×v—mr[k,2]×q[j];
                    yt[j] := =yt[j]+r;
                    q[j] := =q[j]+3.0×r—mr[k,3]×v
                  end j;
                for j := =1 step 1 until m do ys[j] := =yt[j]
              end k;
            if s=2 then go to zer;
            if sign(yi[1]) ≠ sign(ys[1]) then go to it;
tr:         for j := =0 step 1 until m do yi[j] := =ys[j];
            go to if yi[0] > ff then fin else r2;
it:         s := =2;
            for j := =0 step 1 until m do yal[j] := =ys[j];
zer:        d := =d/2;
            if d > eps then
              go to if sign(yi[1]) = sign(ys[1]) then tr
                else r1;
            z[n] := =yi[0]+d; n := =n+1;
            for j := =0 step 1 until m do yi[j] := =yal[j]
          end d;
fin: end zersol;
```

Свидетельство к алгоритму 194а

Алгоритм 194а получен в результате исправления, сокращения, модификации и ординарной переработки алгоритма 194 (Domingo C. «CACM», 1963, № 8). Модификация заключалась в замене локализованного массива *MR* формальным параметром *mr* для того, чтобы сделать процедуру независимой от конкретных значений этого массива, выбираемых пользователем процедуры.

В алгоритме 194 была исправлена ошибка, заключавшаяся в том, что после нахождения всех корней процедура зацикливалась на отрезке от метки *r2* до метки *it* (в алгоритме 194 им соответствуют метки *R2* и *IT*). Зацикливание происходило тогда, когда все корни были меньше *ff*. Для исправления этой ошибки в теле процедуры *zersol* оператор **go to** *r2*; был заменен оператором

go to if yi[0] > ff **then** fin **else** r2;

а последняя строка «**end** zersol;» была заменена на следующий текст:

; fin: **end** zersol;

Нужно заметить также, что данный алгоритм находит только корни функции $y_1(y_0)$, т. е. корни только одного из решений системы дифференциальных уравнений. Для отыскания корней других решений системы нужно последовательно менять нумерацию уравнений (и соответ-

ственно их решений) и повторять выполнение процедуры так, чтобы при каждом выполнении процедуры *zersol* первой считалась та функция, для которой в данный момент отыскиваются корни.

Корни отыскиваются в интервале от начального значения переменной интегрирования y_0 до конечного ее значения $y_0 > ff$. Интегрирование ведется с постоянным шагом h, и если расстояние по аргументу между двумя корнями меньше h, то эти корни могут быть пропущены.

Кроме того, нужно иметь в виду, что погрешность в значениях отыскиваемых корней в значительной степени зависит от величины шага интегрирования, что можно видеть по нижеприведенным результатам трансляции алгоритма 194а.

Алгоритм 194а был транслирован сначала для системы

$$y'_0 = 1,$$
$$y'_1 = 2y_0$$

при начальных условиях $y_0 = -2$, $y_1 = 3$, решением которой является, очевидно, функция $y_1 = y^2_0 - 1$ с корнями $z = \pm 1$. При этом задавались параметры $ff = 2$ и $yi = (-2,3)$. Результаты даны в табл. 33.

Таблица 33

h	eps	$z_1 = -1$	$z_2 = 1$
0.05	10^{-4}	—1.00001831	1.00000610
0.05	10^{-8}	—1.00000001	1.00000000
0.10	10^{-8}	—0.999999994	1.00000000

Затем интегрировалась система

$$y'_0 = 1,$$
$$y'_1 = {}^3/_2 y^2_0 - 2,$$
$$y'_2 = 3y_0.$$

Функция $y_1(y_0)$ имеет корни $z_1 = -2$, $z_2 = 0$ и $z_3 = 2$. Задавались параметры $m = 2$, $ff = 2.5$, $yi = (-2.5, -2.8125, 7.375)$, $h = 0.1$, 0.05, 0.005, 0.0005.

Результаты трансляции этой системы даны в табл. 34.

Таблица 34

h	eps	$z_1 = -2$	$z_2 = 0$	$z_3 = 2$
0.1000	10^{-8}	—2.01860405	0.122324639	1.97148118
0.0500	10^{-8}	—2.00992979	0.0598181785	1.98765672
0.0050	10^{-8}	—2.00104857	0.0058711358	1.99892747
0.0005	10^{-8}	—2.00010546	0.0005860114	1.99989440
0.0050	10^{-5}	—2.00104492	0.00586914262	1.99891601
0.0050	10^{-3}	—2.00187500	0.00562500200	1.99812500

Замечание рецензента к алгоритму 194а

Алгоритм полезен. К недостаткам алгоритма можно отнести отсутствие автоматического выбора шага интегрирования и необходимость вводить матрицу *mr*. Целесообразнее, по-видимому, было бы остановиться на каком-либо определенном варианте метода Рунге — Кутта.

АЛГОРИТМ 1956

Система линейных уравнений с ленточной матрицей [F4]

Процедура *bandsolve* (*band* — лента, *solve* — решать) находит решение матричного уравнения $ax=b$, где a — матрица большого порядка, все ненулевые элементы которой содержатся в узкой полоске (ленте), сконцентрированной вдоль главной диагонали. Параметр n — порядок матрицы a, m — ширина ленты (обязательно нечетное число).

Процедура *bandsolve* весьма эффективна, потому что оперирует только с ленточной частью матрицы a, заданной в массиве $c[1:n,1:m]$. Ленточные элементы i-й строки матрицы a размещаются в i-й строке матрицы c так, что элемент $a[i,j]$ переходит в элемент $c[i,j-i+(m+1)/2]$.

Все ленточные элементы матрицы a, как нулевые, так и ненулевые, должны присутствовать в массиве c. Значения неопределенных (указанным способом) элементов массива c, таких как $c[1,1]$ или $c[n,m]$, могут быть произвольными. Массив v вначале содержит вектор b. После выполнения процедуры *bandsolve* в массиве $v[1:n]$ содержится решение (вектор x). Значение массива c затирается в процессе решения, выполняемого методом исключения по Гауссу с перестановкой строк, после которой следует обратная подстановка.

```
procedure bandsolve(c,n,m,eps,signal)dataresult:(v);
      value n,m,eps; real eps; integer n,m;
      label signal; array c,v;
begin real t; integer i,j,jm,lr,piv,r;
      lr:=(m+1)÷2;
      for r:=1 step 1 until lr-1 do
        for i:=1 step 1 until lr-r do
          begin
            for j:=2 step 1 until m do c[r,j-1]:=c[r,j];
            c[r,m]:=c[n+1-r,m+1-i]:=0
          end сдвига строки и засылки нулей;
      for i:=1 step 1 until n-1 do
        begin piv:=i;
          for r:=i+1 step 1 until lr do
            if abs(c[r,1])>abs(c[piv,1]) then piv:=r;
          if abs(c[piv,1])<eps then go to signal;
          if piv≠i then
            begin t:=v[i];
              v[i]:=v[piv]; v[piv]:=t;
              for j:=1 step 1 until m do
                begin t:=c[i,j];
                  c[i,j]:=c[piv,j]; c[piv,j]:=t
                end j
            end перестановки строк;
          v[i]:=v[i]/c[i,1];
          for j:=2 step 1 until m do c[i,j]:=c[i,j]/c[i,1];
          for r:=i+1 step 1 until lr do
          begin t:=c[r,1];
            v[r]:=v[r]-t×v[i];
            for j:=2 step 1 until m do c[r,j-1]:=c[r,j]-t×c[i,j];
```

```
            c[r,m] := =0
          end r;
        if lr≠n then lr := =lr+1
      end триангуляризации;
    v[n] := =v[n]/c[n,1];
    jm := =2;
    for r := =n—1 step —1 until 1 do
      begin
        for j := =2 step 1 until jm do
          v[r] := =v[r]—c[r,j]×v[r—1+j];
        if jm≠m then jm := =jm+1
      end обратной подстановки
end bandsolve;
```

Свидетельство к алгоритму 195б

Алгоритм 195б получен из алгоритма 195а в результате внесения в него поправки, предложенной Э. Шуграфом в его «Замечании к алгоритму 195» (см. ниже). Поскольку внесенные в алгоритм 195а изменения являются элементарными и очевидными, то новой трансляции алгоритма не делалось.

Свидетельство к алгоритму 195а

Алгоритм 195а получен в результате ординарной переработки алгоритма 195 (Thurnau D. H. «CACM», 1963, № 8) и исправления в нем одной опечатки, заключавшейся в отсутствии точки с запятой после оператора $t := =c[r,1]$ в 16-й строке снизу от конца процедуры.

Алгоритм 195а был транслирован с исходными данными

$$a = \begin{Vmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 5 & 0 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 2 & 6 & 0 \\ 0 & 0 & 0 & 4 & 5 \end{Vmatrix} \rightarrow c = \begin{Vmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 7 & 8 & 9 \\ 2 & 6 & 0 \\ 4 & 5 & 0 \end{Vmatrix}$$

(т. е. при $n=5$, $m=3$) и $v=(5,26,74,45,41)$.

Получен правильный результат

$$v=(6,—0.5,2,6.83333333,2.73333333).$$

Затем было проведено другое решение

$$a = \begin{Vmatrix} 1 & 2 & 3 & 0 & 0 \\ 4 & 3 & 2 & 1 & 0 \\ 5 & 4 & 3 & 2 & 1 \\ 0 & 1 & 2 & 3 & 5 \\ 0 & 0 & 3 & 2 & 1 \end{Vmatrix} \rightarrow c = \begin{Vmatrix} 0 & 0 & 1 & 2 & 3 \\ 0 & 4 & 3 & 2 & 1 \\ 5 & 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 5 & 0 \\ 3 & 2 & 1 & 0 & 0 \end{Vmatrix}$$

(т. е. при $n=5$, $m=5$) и $v=(22,40,55,20,14)$.

Получен правильный результат

$$v=(4,5.25,2.50,3.25,0).$$

В обоих случаях преобразование матрицы *a* в матрицу *c* производилось с помощью оператора

```
for i := =1 step 1 until n do
  for j := =1 step 1 until n do
    begin k := =j—i+(m+1)/2;
      if 1≤k∧k≤m then c[i,k] := =a[i,j]
    end i
```

Замечание к алгоритму 195

Э. Шуграф (Schuegraf E. «CACM», 1972, № 12)

Алгоритм 195 был переведен на ФОРТРАН-IV для машины IBM 360/50. Были использованы различные матрицы с различными значениями n и m*. Регистрировались затраты машинного времени, и проверялась точность результатов. Время выполнения процедуры (в секундах) приведено в табл. 35.

Таблица 35

n \ m	11	15	21	25
50	0.2	0.7	1.1	1.9
100	0.6	1.6	2.5	4.2

Можно считать, что время выполнения процедуры пропорционально $((m-1)\div 2)^2 \times n$ (обратите внимание на определение m). При проверке результатов было обнаружено, что для вырожденных и близких к вырожденным матриц алгоритм дает неверные решения. Этот деффект можно устранить, если для проверки на вырожденность ввести параметр *eps* и метку-параметр *signal*. Для этого в описании процедуры первые две строки нужно заменить на следующие:

```
procedure bandsolve(c,n,m,eps,signal) dataresult:(v);
    value n,m; real eps; integer n,m;
    label signal; array c,v;
```

После оператора

```
    if abs(c[r,1])>abs(c[piv,1]) then piv := =r;
```

вставить оператор

```
    if abs(c[piv,1])<eps then go to signal;
```

АЛГОРИТМ 196б

Метод Мюллера нахождения корней произвольной функции [C5]

Процедура *muller* находит вещественные и комплексные корни произвольной функции.

---

* Здесь используются обозначения алгоритма 195а. В алгоритме 195 эти буквы были прописными. (*Прим. ред.*)

Значения параметров:

p1,p2,p3 — начальные значения. Корни, ближайшие к этим значениям, находятся в первую очередь.

mxm — максимально допустимое число итераций, которое может быть сделано при нахождении любого корня.

ep1,ep2 — допустимые погрешности. Если $abs((x_{i+1}-x_i)/x_{i+1})<ep1$ или если и значение функции, и значение модифицированной функции меньше *ep2*, то корень считается найденным.

ep3 — оценка погрешности в представлении чисел типа **real** в данной машине. Эта оценка не должна быть меньше двух единиц последнего разряда машинного слова.

sw1 — логический параметр. Если *sw1*≡ **true**, то печатается каждая итерация каждого корня.

sw2 — логический параметр. Если *sw2*≡ **true**, то печатается значение каждого корня сразу после его нахождения.

sw3 — логический параметр. Если *sw3*≡ **true**, то (когда это приемлемо) в число корней включается и печатается комплексное сопряженное каждого найденного корня.

swr — логический параметр. Если *swr*≡**true**, то отыскиваются только вещественные корни.

rrt,irt — выходные параметры-массивы, содержащие вещественные и мнимые части (соответственно) каждого найденного корня. Размерности этих массивов [1 : *nrts*], где *nrts* — целое, не меньшее, чем предполагаемое число корней.

bool[*i*] — выходной параметр, значение которого **false**, если для *i*-го корня достигнут критерий сходимости. В противном случае — **true**. Размерность [1 : *nrts*].

*fun*196 — процедура, вычисляющая значения функции. Обращение имеет вид *fun*196(*r,i,fr,fi*), где *r,i* — вещественная и мнимая части аргумента, а *fr,fi* — вещественная и мнимая части значения функции.

В теле процедуры *muller* локализована процедура *complex*, выполняющая необходимые комплексные операции. Для вывода результатов на печать используется стандартная процедура *outreal* [1].

```
procedure muller(p1,p2,p3,mxm,nrts,ep1,ep2,ep3,sw1,sw2,sw3,swr,fun)
        result:(rrt,irt,bool);
        value p1,p2,p3,mxm,nrts,ep1,ep2,ep3,sw1,sw2,sw3,swr;
        real p1,p2,p3,ep1,ep2,ep3; integer mxm,nrts;
        Boolean sw1,sw2,sw3,swr; array rrt,irt;
        Boolean array bool; procedure fun;
begin real rx1,rx2,rx3,ix1,ix2,ix3,rroot,iroot,rdnr,idnr,
        t1,it1,frroot,firoot,rfx1,rfx2,rfx3,ifx1,ifx2,ifx3,
        rh,ih,rlam,ilam,rdel,idel,t2,it2,t3,it3,t4,it4,rg,ig,
        rden,iden,rfunc,ifunc; integer cl,rtc,i,itc;
        switch j:=m2,m3,m4,m7,m11;
        procedure complex(a,ia,b,ib,k,c,ic);
          value a,ia,b,ib,k; real a,ia,b,ib,c,ic; integer k;
          begin real t; switch j:=mpy,dvd,sqt;
            go to j[k];
mpy:        c:=a×b−ia×ib; ic:=a×ib+ia×b;
            go to exit;
dvd:        t:=b↑2+ib↑2;
```

```
            if t<ep3 then
              begin ic := =0; c := =1; go to exit end;
            c := =(a×b+ia×ib)/t; ic := =(ia×b—a×ib)/t;
            go to exit;
sqt:        t := =abs(ia);
            if t<ep3∧a<0 then
              begin c := =0; ic := =sqrt(—a) end else
              if t<ep3 then
                begin c := =sqrt(a); ic := =0 end else
                begin t := =sqrt((abs(a)+sqrt(a↑2+ia↑2))/2);
                  ic := =0.5×ia/t;
                  if a≥0 then c := =t else
                    begin c := =abs(ic); ic := =sign(ia)×t end
                  end;
            if (b+c)↑2+(ib+ic)↑2<(b—c)↑2+(ib—ic)↑2 then
              begin c := =b—c; ic := =ib—ic end else
              begin c := =b+c; ic := =ib+ic end;
exit:     end complex;
start: for i := =1 step 1 until nrts do rrt[i] := =irt[i] := =0;
       rtc := =1;
m0:    ix1 := =ix2 := =ix3 := =iroot := =0; c1 := =itc := =0;
       rroot := =p1;
m1:    rdnr := =1; idnr := =0; bool[rtc] := = false;
       for i := =1 step 1 until rtc—1 do
         begin
           if abs(rroot—rrt[i])<ep3∧iroot<irt[i] then
             begin
               if c1=0 then p1 := =rroot := =2×rroot+ep1 else
                 if c1=1 then p2 := =rroot := =2×rroot+ep1 else
                   if c1=2 then p3 := =rroot := =2×rroot+ep1 else
                     go to fin1;
               comment В последнем случае мы приблизились к предварительно найденному корню и принимаем его приближение без проверки;
               go to m1
             end if rroot;
           complex(rdnr,idnr,rroot—rrt[i],iroot—irt[i],1,t1,it1);
           rdnr := =t1; idnr := =it1
         end i;
       c1 := =c1+1;
       fun196(rroot,iroot,t1,it1);
       complex(t1,it1,rdnr,idnr,2,frroot,firoot);
       go to j[c1];
m2:    rfx1 := =frroot; ifx1 := =firoot; rroot := =p2;
       go to m1;
m3:    rfx2 := =frroot; ifx2 := =firoot; rroot := =p3;
       go to m1;
m4:    rfx3 := =frroot; ifx2 := =firoot; rx1 := =p1;
       rx2 := =p2; rx3 := =p3; rh := =rx3—rx2;
       ih := =ix3—ix2;
       complex(rh,ih,rx2—rx1,ix2—ix1,2,rlam,ilam);
       rdel := =rlam+1; idel := =ilam;
```

```
m9:   if abs(rfx1—rfx2)<ep3∧abs(rfx2—rfx3)<ep3∧
        abs(ifx1—ifx2)<ep3∧abs(ifx2—ifx3)<ep3 then
        begin rlam :=1; ilam :=0; go to m8 end;
      complex(rfx1,ifx1,rlam,ilam,1,t1,it1);
      complex(rfx2,ifx2,rdel,idel,1,t2,it2);
      t1 :=t1—t2+rfx3; it1 :=it1—it2+ifx3;
      complex(rdel,idel,rlam,ilam,1,t2,it2);
      complex(t1,it1,t2,it2,1,t3,it3);
      complex(rfx3,ifx3,t3,it3,1,t1,it1);
      t1 :=—4×t1; it1 :=—4×it1;
      complex(rfx3,ifx3,rlam+rdel,ilam+idel,1,t2,it2);
      complex(rdel↑2—idel↑2,2×rdel×idel,rfx2,ifx2,1,t3,it3);
      complex(rlam↑2—ilam↑2,2×rlam×ilam,rfx1,ifx1,1,t4,it4);
      rg :=t4—t3+t2; ig :=it4—it3+it2;
      if swr∧(rg↑2+t1)<0 then
        begin rden :=rg; iden :=ig :=0 end else
        complex(rg↑2—ig↑2+t1,2×rg×ig+it1,rg,ig,3,
          rden,iden);
      complex(—2×rfx3,—2×ifx3,rdel,idel,1,t1,it1);
      complex(t1,it1,rden,iden,2,rlam,ilam);
m8:   itc :=itc+1;
      rx1 :=rx2; rx2 :=rx3; rfx1 :=rfx2; rfx2 :=rfx3;
      ix1 :=ix2; ix2 :=ix3; ifx1 :=ifx2; ifx2 :=ifx3;
      complex(rlam,ilam,rh,ih,1,t1,it1);
      rh :=t1; ih :=it1;
m6:   rdel :=rlam+1; idel :=ilam; rx3 :=rx2+rh;
      ix3 :=ix2+ih; c1 :=3; rroot :=rx3; iroot :=ix3;
      go to m1;
m7:   rfx3 :=frroot; ifx3 :=firoot;
      fun196(rx3,ix3,rfunc,ifunc);
      complex(rfx3,ifx3,rfx2,ifx2,2,t1,it1);
      if t1↑2+it1↑2>100 then
        begin rlam :=rlam/2; rh :=rh/2;
          ilam :=ilam/2; ih :=ih/2;
          go to m6
        end;
      if sw1 then
        begin outreal(1,rtc); outreal(1,itc);
          outreal(1,rx3); outreal(1,ix3)
        end;
      comment Вывод каждой итерации каждого корня;
      t1 :=rx3—rx2; it1 :=ix3—ix2;
      complex(t1,it1,rx2,ix2,2,t2,it2);
      if sqrt(t2↑2+it2↑2)≤ep1∨sqrt(rfx3↑2+ifx3↑2)≤ep2
        ∧sqrt(rfunc↑2+ifunc↑2)≤ep2 then go to fin1;
      go to if itc≥mxm then fin3 else m9;
fin1: if sw2 then
        begin outreal(1,rtc); outreal(1,itc);
          outreal(1,rx3); outreal(1,ix3)
        end;
      comment Вывод корня;
      go to m12;
```

```
fin3:  if sw2 then
         begin outreal(1,rtc); outreal(1,itc);
           outreal(1,rx3); outreal(1,ix3)
         end;
       comment Нет сходимости. Вывод последней итерации;
       bool[rtc] := = true;
m12:   rrt[rtc] := =rx3; irt[rtc] := =ix3;
       if rtc⩾nrts then go to exit;
       if abs(ix3)>ep1∧sw3∧¬bool[rtc] then
         begin ix3 := =—ix3;
           fun196(rx3,ix3,rfunc,ifunc);
           rroot := =rx3; iroot := =ix3; c1 := =4; go to m1;
m11:       rtc := =rtc+1; rrt[rtc] := =rx3; irt[rtc] := =ix3;
           if sw3 then
             begin outreal(1,rtc); outreal(1,rx3);
               outreal(1,ix3)
             end;
           comment Здесь выполняется вывод комплексного сопряженного найденному корню, поскольку оно оказалось приемлемым;
         end;
       if rtc<nrts then
         begin rtc := =rtc+1; go to m0 end;
exit: end muller;
```

Свидетельство к алгоритму 196б.

Алгоритм 196б получен из алгоритма 196а в результате внесения в него следующих исправлений.

1. Была исправлена опечатка, заключавшаяся в том, что метка *dvd* находилась строкой ниже, чем это нужно. Таким образом, третья после метки *mpy* строка

```
t := =b↑2+ib↑2;
```

должна иметь вид

```
dvd: t := =b↑2+ib↑2;
```

2. В алгоритме 196а (так же, как и в алгоритме 196) некоторые условия выполнялись путем сравнения с нулем значений типа **real**. Из-за неточности машинной арифметики вещественные значения нужно сравнивать не с нулем, а с некоторым малым числом, большим, однако, возможных погрешностей округления. В противном случае может возникнуть зацикливание. Поэтому в список формальных параметров, в список значений и в совокупность спецификаций был добавлен параметр *ep*3, а в теле процедуры были сделаны следующие корректуры:

а) строка

```
dvd: if t=0 then
```

была заменена строкой

```
if t<ep3 then
```

б) строка

```
sqt: if (ia=0)∧(a<0) then
```

была заменена строками

```
sqt: t := =abs(ia);
     if t<ep3∧a<0 then
```

в) двумя строками ниже условие

```
if ia=0 then
```

было заменено условием

```
if t<ep3 then
```

г) четвертая строка после метки *m*1, имевшая вид

```
if rroot=rrt[i]∧iroot<irt[i] then
```

приняла вид

```
if abs(rroot—rrt[i])<ep3∧iroot<irt[i] then
```

д) строка с меткой *m*9 была заменена строками

```
m9:  if abs(rfx1—rfx2)<ep3∧abs(rfx2—rfx3)<ep3∧
       abs(ifx1—ifx2)<ep3∧abs(ifx2—ifx3)<ep3 then
```

3. В третьей строке после метки *m*12 значение переменной *bool*[*rtc*] оказывалось неопределенным. Для исправления этого были сделаны следующие корректировки:

а) во второй строке после метки *start* оператор *rtc* : =0 был заменен оператором *rtc* : =1;

б) три строки ниже метки *m*0, первая из которых имела вид

```
rroot : =p1; bool[rtc] : = false;
```

были заменены строками

```
      rroot : =p1;
m1:   rdnr : =1; idnr : =0; bool[rtc] : = false;
      for i : =1 step 1 until rtc—1 do
```

в) из строки с меткой *m*12 был исключен оператор *rtc* : =*rtc*+1;

г) предпоследняя строка (перед меткой *exit*) была заменена строками

```
    if rtc<nrts then
      begin rtc : =rtc+1; go to m0 end;
```

д) две строки выше, имевшие вид

```
        end; else
        go to m0;
```

были заменены одной строкой

```
        end;
```

4. Чтобы комплексное сопряженное печаталось именно для только что полученного корня, а не для предыдущего, пятая строка ниже метки *m*11 была помещена непосредственно вслед за меткой *m*11.

5. Вывод значений переменных *rrt*[*rtc*] и *irt*[*rtc*] при выполнении условий **if** *sw*1 **then** и **if** *sw*2 **then** был заменен выводом значений *rtc*, *itc*, *rx*3 и *ix*3 для повышения содержательности выводимой информации.

6. Глобальная процедура *fun*196 была заменена формальной процедурой *fun* ради удобства использования данного алгоритма в автоматизированных библиотеках, подобных архиву системы БЭСМ-АЛГОЛ.

Алгоритм 196б был транслирован в системе ТА-1М на машине М-220, и с его помощью были успешно решены следующие задачи.

Задача 1. Решить уравнение

$$\ln z = z - 1.195281 - 0.536353i,$$

корень которого $z = 2+i$ известен из результатов отладки алгоритма 48б (см. значение $\ln(2+i)$ в [47, табл. 20]).

Для решения этой задачи процедура *fun* задавалась в следующем виде:

```
procedure fun(r,i,fr,fi);
      value r,i; real r,i,fr,fi;
begin real lnr,lni;
      logc(r,i,infin,lnr,lni);
      fr := lnr—r+1.195281; fi := ln i—i+0.536353;
      go to fin;
infin: stop;
fin: end;
```

где *logc* — процедура алгоритма 48б. Остальные входные параметры были следующими: $p1 = -0.5$, $p2 = 0.8$, $p3 = 1.5$, $mxm = 1000$, $nrts = 1$, $ep1 = ep2 = 10^{-7}$, $ep3 = 10^{-8}$, $sw1 \equiv sw2 \equiv$ **true**, $sw3 \equiv swr \equiv$ **false**.

Были получены результаты $rrt = 2$, $irt = 1$ и $bool \equiv$ **false**. Для ускорения выполнения алгоритма в системе ТА-1М процедура *complex* была вынесена из тела процедуры *muller* и локализована во внешней программе. Затем процедуры *complex*, *fun*, *logc*, *compow*, *ln*, *exp*, *sin*, *cos*, *arctg* и *p*1041 были зафиксированы с помощью процедуры *p*0717 на рабочем поле программы ИС-2, входящей в транслирующую систему ТА-1М. Время решения задачи при этом было примерно 4с.

Задача 2. Определить два корня выражения

$$z^{-z/2} - z + 2.195093245 - 2.750498700i,$$

для которого один корень $z = 2 - 2i$ уже известен, поскольку значение $(2-2i)^{(-1+i)}$ было вычислено ранее в алгоритме 190б (см. табл. 26).

Процедура *fun* задавалась в следующем виде:

```
procedure fun(r,i,fr,fi);
      value r,i; real r,i,fr,fi;
begin real pr,pi;
      compow(r,i,—r/2,—i/2,0,pr,pi);
      fr := pr—r+2.195093245; fi := pi—i—2.7504987
end fun
```

где *compow* — процедура алгоритма 190б. Значение *nrts* задавалось равным 2, а остальные параметры были такими же, как в задаче 1.

Были получены результаты $rrt = (2, 3.0924916)$, $irt = (-2, -8.4395384)$ и $bool \equiv$ (**false**,**false**).

Для проверки правильности второго из этих корней был выполнен оператор

```
for i:=1 step 1 until nrts do
  begin fun(rrt[i],irt[i],fr,fi);
      outreal(1,fr); outreal(1,fi)
  end
```

Полученные значения (*fr*,*fi*), которые теоретически должны быть нулями, были равны $(0.6_{10}-10,\ 0.3_{10}-8)$ для $i=1$ и $(-0.12_{10}-9,\ 0.19_{10}-8)$ для $i=2$. Время решения (при таком же фиксировании процедур, как в задаче 1) было 13—14 с.

Задача 3. Найти корни полиномов

1) $z^4 - 3z^3 + 20z^2 + 44z + 54$,
2) $z^5 + z^4 - 8z^3 - 16z^2 + 7z + 15$,
3) $z^6 - 14z^4 + 49z^2 - 36$,
4) $z^6 - 2z^5 + 2z^4 + z^3 + 6z^2 - 6z + 8$,

которые были использованы ранее для откладки алгоритма 3б.

Для решения этой задачи редактор выпуска составил специальную процедуру *polcomp*, вычисляющую значение $u+iv$ комплексного полинома $\sum_{i=0}^{n} a_i z^i$ от комплексного аргумента $z=x+iy$ по схеме Горнера.

```
procedure polcomp(x,y,n,a)result:(u,v);
    value x,y,n; real x,y,u,v; integer n; array a;
begin real s; integer i;
    u:=v:=0;
    for i:=n step—1 until 0 do
      begin s:=u;
        u:=s×x—v×y+a[i]; v:=s×y+v×x
      end i
end polcomp
```

Поскольку эта процедура может иметь широкое применение не только в алгоритме 196б, но и во многих других задачах, она оформлена здесь независимо от процедуры *fun*, которая в данной задаче имела вид

```
procedure fun(r,i,fr,fi);
    value r,i; real r,i,fr,fi;
polcomp(r,i,n,a,fr,fi)
```

Значение *nrts* задавалось равным степени соответствующего полинома, а остальные параметры процедуры *muller* были такими же, как в задаче 1.

Полученные значения корней не отличались больше чем на одну единицу последнего разряда от результатов отладки алгоритма 3б и были такими, как указано в табл. 36.

Таблица 36

Номер полинома	u	v	Номер полинома	u	v
1	—0.97063897 2.4706389	±1.0058075 ±4.6405331	3	±1.0000000 ±2.0000000 ±3.0000000	0 0 0
2	±1.0000000 3.0000000 —2.0000000	0 0 ±1.0000000	4	0.50000000 —1.0000000 1.50000000	±0.86602540 ±1.0000000 ±1.3228756

Для ускорения работы программы на рабочем поле ИС-2 фиксировались процедуры *complex*, *fun*, *polcomp* и *p*1041. Время решения для вышеуказанных четырех полиномов было при этом равно 8, 13, 18 и 13 с соответственно.

Свидетельство к алгоритму 196а

Алгоритм 196а получен в результате ординарной переработки алгоритма 196 (Rodman R. D. «CACM», 1963, № 8), внесения в него поправок, предложенных В. Уитли в нижеследующем его «Подтверждении», и исправления в нем следующих очевидных ошибок.

1. После метки «m0:» нельзя писать оператор

```
ix1 : =ix2 : =ix3 : =c1 : =iroot : =itc : =0;
```

поскольку в нем переменные *c1* и *itc* типа integer, а остальные переменные типа real. Кроме того, в этом операторе после переменной *c1* не должно быть запятой.

2. Нужно убрать точку с запятой после переменной *c1* в операторе

m1:c1; :=c1+1

3. В третьей строке после метки «m12:» нужно заменить идентификатор *ABS* на *abs*.

4. После метки «m11:» вместо условия if sw2 then ... должно быть условие if sw3 then ...

Подтверждение к алгоритму 196

В. Уитли (Whitley V. W. «CACM», 1968, № 1)

Алгоритм 196 был переведен на ФОРТРАН IV и транслирован на машинах CDC-3600 и IBM-7090 с одинарной и двойной точностью. В вариантах одинарной точности использовались подпрограммы комплексной арифметики, в вариантах двойной точности — подпрограммы, входящие в транслирующую систему, по возможности близко соответствующие подпрограммам, описанным в работе [19i]. Следовательно, проверявшийся алгоритм отличается от опубликованного только подпрограммой извлечения квадратных корней из комплексных величин.

К алгоритму 196 имеются пять следующих замечаний.

1. Если одно из значений *p1*, *p2* или *p3* совпадает со значением корня уравнения и если имеется более чем один корень, то процедура приводит к ошибке вида 0/0 при вычислении

$$F_r(z) = f(z) \Big/ \prod_{i=1}^{rtc} (z - z_i)$$

Мы решили в этом случае прекращать выполнение процедуры с выдачей сигнала пользователю. Рецензент предложил следующую альтернативу ...*

4. Франк в своей работе [20i] утверждает: «Эта процедура (метод Мюллера) для функций, имеющих простые корни, работает быстро. С другой стороны, если функция имеет кратный корень ξ, то $F_r(z)$ не определена, когда z приближается к значению ξ, которое может быть уже найдено. Однако даже и в этом случае никогда не получается ошибки. Действительно, были успешно найдены корни кратности шесть и более. Это является следствием главным образом того факта, что кратные корни отыскиваются с гораздо меньшей точностью, чем простые, и ведут себя по существу подобно близко расположенным корням». В личной переписке Франк пояснил, что подпрограмма, описанная в вышеуказанной его работе, включает в себя шаги, которые несколько изменяли найденные уже корни, заставляли их вести себя подобно близко расположенным корням. Замечание Франка для алгоритма 196 несправедливо. Это можно продемонстрировать простым тестом для функции $(z-2)^2$, имеющей двойной корень.

5. Извлечение квадратного корня из комплексных величин внутри процедуры *complex* содержит по крайней мере две ошибки, не самой

---

* Далее приводится отрывок процедуры взамен пяти строк, следующих за меткой *m1*, использованный авторами выпуска в алгоритме 196а. Следующие замечания 2 и 3 также опущены здесь, потому что указанные в них поправки учтены в алгоритме 196а. (*Прим. ред.*)

малой из которых является ошибка, связанная с вычислением местоположения комплексного числа, подлежащего извлечению корня. Рецензент указал вторую ошибку: «Пятая строка после метки *sqt* должна иметь вид

$$\mathrm{e}:=\mathrm{sqrt}((\mathrm{temp}+\mathrm{a})/2)\times\mathrm{sign}(\mathrm{ia})$$

Следующая за этим оператором конструкция

if t<a **then** 0 **else**

не нужна, так как отношение $t<a$ не может быть истинным. Кроме того, даже с этими поправками извлечение квадратного корня из комплексных величин в данной процедуре неудовлетворительно, потому что если ia мало, то либо $t-a$, либо $t+a$ окажется разностью между двумя почти равными числами и произойдет потеря значащих цифр».

Предлагается составной оператор, начинающийся с пятой строки после метки *sqt*, заменить на *

```
begin t:=sqrt((abs(a)+sqrt(a↑2+ia↑2))/2);
  ic=0.5×ia/t;
  if a≥0 then c:=t else
    begin c:=abs(ic); ic:=sign(ia)×t end
end;
```

Для некоторых вычислительных систем случай $ia=-0$ вызывает затруднения. Возможно, за исключением случая $ia=-0$, вышеприведенный оператор будет выбирать квадратный корень, у которого вещественная часть положительна.

Модификация алгоритма. В обоих вариантах, как с двойной, так и с одинарной точностью, были внесены изменения согласно предложению Трауба [21i]:

$$z_{i+1}=z_i-2f_i/p_i,\ \text{где}\ f_i=f(z_i),$$

$$p_i=\omega_i+\{\omega^2{}_i-4f_i f[z_i,z_{i-1},z_{i-2}]\}^{1/2},$$

$$\omega_i=f[z_i,z_{i-1}]+(z_i-z_{i-1})f[z_i,z_{i-1},z_{i-2}],$$

$$f[z_i,z_{i-1},z_{i-2}]=\frac{f[z_i,z_{i-1}]-f[z_{i-1},z_{i-2}]}{z_i-z_{i-2}},$$

$$[z_i,z_{i-1}]=(f_i-f_{i-1})/(z_i-z_{i-1}).$$

Как алгоритм 196, так и итерационная формула Трауба определяют знак квадратного корня для максимизации модуля знаменателя.

Хотя обе итерационные формулы эквивалентны, формула Трауба требует меньше операций (8 сложений, 5 умножений и 3 деления, в то время как по Мюллеру 10 сложений, 15 умножений и 2 деления), меньше памяти и меньше машинного времени.

Работа запрограммированного варианта формулы Трауба мало отличается от работы алгоритма 196. При одинаковых входных значениях оба метода сходятся за одинаковое число итераций, даже если первые итерации дают иногда разные результаты.

В табл. 37 примера 1 результаты вычислений по методам Мюллера и Трауба с одинарной точностью сравниваются с результатами вычисления с двойной точностью. Различие между первыми итерациями при

---

* Указанная здесь замена выполнена в алгоритме 196а. (*Прим. ред.*)

Таблица 37

Номер итерации	Номер варианта	z	$f(z)$
1	1	0.8959044683	—0.8890227259
	2	0.8959064917	—0.8890177130
	3	0.89590562323582572	—0.88901986473910256
2	1	1.009184101	0.2006266970
	2	1.009182792	0.2005955638
	3	1.0091833539604292	0.20060892685123016
3	1	0.9915438059	—0.1562027260
	2	0.9915451528	—0.1561798028
	3	0.99154457471920591	—0.15618964171373523
4	1	0.9996899988	$-0.6181798671\times10^{-2}$
	2	0.9996900961	$-0.6179863674\times10^{-2}$
	3	0.99969005435603457	$-0.61806941810976141\times10^{-2}$
5	1	0.9999986299	$-0.2740146010\times10^{-4}$
	2	0.9999986307	$-0.2738516196\times10^{-4}$
	3	0.99999863036901786	$-0.27392263226776617\times10^{-4}$
6	1	1.000000000	$0.3492459655\times10^{-8}$
	2	1.000000000	$0.3492459655\times10^{-8}$
	3	1.0000000001972048	$0.34940963812509629\times10^{-8}$

одинарной точности является результатом округления, вызванного тем, что начальные функции $f(p1)$, $f(p2)$ и $f(p3)$ очень близки друг к другу. При вычислениях с двойной точностью оба метода дают результаты, совпадающие в 17 значащих цифрах (машина CDC-3600 при двойной точности дает примерно 24 десятичные цифры).

Результаты счета с двойной точностью сравнивались с результатами счета с одинарной точностью на одной и той же машине, и, кроме того, сравнивались друг с другом результаты счета с одинаковой точностью (как с двойной, так и с одинарной) на двух разных машинах. В каждом случае различия могли быть удовлетворительно объяснены следующими факторами: 1) различными реализациями двойной арифметики на различных машинах, 2) различной длиной слова или 3) различиями в библиотечных подпрограммах на различных машинах.

Пример 1. $f(z)=z^{20}-1$.

$p1=0.1875$, $f(p1)=-0.99999999999999715799543$, $ep1=0.5\times10^{-6}$,

$p2=0.375$, $f(p2)=-0.99999999697569662195778 5$, $ep2=0.5\times10^{-6}$,

$p3=0.5$, $f(p3)=-0.99999904632468349375000 0$, $nrts=2$.

В табл. 37 вариант 1 — это алгоритм 196 с одинарной точностью, вариант 2 — метод Трауба с одинарной точностью и вариант 3 — счет с двойной точностью (цифры, совпадающие для обоих методов).

Пример 2. Акустическая волновая функция

$$f(z)=P\frac{\sin(2\pi S)}{S}+\rho\times\sin(2\pi S),\ \rho=2.50,$$

$$P=\sqrt{k^2-z^2},\ \mathrm{Re}\,(P)<0,\ S=\sqrt{(z/\varepsilon)^2-k^2},\ \mathrm{Re}\,(S)<0,$$

$k=0.0$, $\varepsilon=0.288$, $p1=0.0$, $p2=0.02$, $p3=0.04$. См. табл. 38.

Таблица 38

Итерация	$z=x+iy$		$f(z)=u+iv$	
	x	y	u	v
1	0.67672247×10^{-1}	0.44008261×10^{-2}	0.24001462	0.49014466×10^{-1}
2	0.71677143×10^{-1}	0.51274250×10^{-2}	0.23259824×10^{-1}	0.98558515×10^{-2}
3	0.72102452×10^{-1}	0.53056952×10^{-2}	0.20655826×10^{-3}	0.19331276×10^{-7}
4	0.72106262×10^{-1}	0.53092607×10^{-2}	0.39539112×10^{-7}	0.40978193×10^{-7}

Пример 3. 20 корней из единицы.

Начальными значениями были 19-, 27- и 35-я степень. Сопряженные значения принимались в качестве корней. Параметры $ep1=ep2=0.5\times10^{-7}$. Корни равны $e^{ir\pi/10}$, $r=0, \ldots, 19$ (см. табл. 39).

Таблица 39

Корни	r	Число итераций	Корни	r	Число итераций
0	3	5	10	1	10
1	17	*	11	19	*
2	4	15	12	2	7
3	16	*	13	18	*
4	7	22	14	6	10
5	13	*	15	14	*
6	9	14	16	0	5
7	11	*	17	8	8
8	5	14	18	12	*
9	15	*	19	10	2

* Звездочка в 3-й колонке таблицы 39 указывает на то, что в качестве данного корня принимались сопряженные значения, т. е.

$$z_1=\bar{z}_0,\ z_3=\bar{z}_2 \text{ и т. д.}$$

Различные замечания. Оба варианта метода Мюллера (алгоритм 196 и итерационная формула Трауба) имеют преимущество перед другими одноточечными методами, по которым можно отыскивать комплексные корни, пользуясь вещественными начальными значениями. Имеется, конечно, возможность неоправданной затраты времени на выполнение комплексной арифметики. Если не рассматривать библиотечную программу как предназначенную главным образом для комплексных корней, то существуют и другие итерационные формулы, которые требуют меньше места в памяти и имеют тот же порядок; существуют также и другие формулы, требующие меньше места и обладающие более высоким порядком, хотя обычно они включают в себя вычисление производных. Для наших целей траубовский вариант метода Мюллера оказался вполне удовлетворительным.

Пример 2 включен здесь для того, чтобы показать поведение алгоритма. при пользовании неалгебраической функцией. Для этой функции результаты алгоритма 196 и модификации Трауба точно совпали, за исключением различия в последней цифре для первой итерации.

Пример 3 приведен для того, чтобы показать, в каком порядке отыскиваются 20 корней из единицы и число итераций, требующихся для достижения заданной точности. Полученные результаты сверялись с таблицами [18i]. Все, кроме двух корней, показали точность до 9 значащих цифр, а эти два корня имели точность 8 значащих цифр.

Автор данного подтверждения выражает благодарность комиссии по атомной энергии (Contract AT (29—2)—1163) за оказанную поддержку. Кроме вышеуказанной литературы, автор пользовался также работой Мюллера [13i].

АЛГОРИТМ 197б

Деление матрицы на матрицу [F1]

Процедура *matrdiv* (сокращение от *matrix* — матрица и *division* — деление) делит прямоугольную матрицу $c[1{:}m,\ 1{:}n]$ на симметричную положительно-определенную матрицу $b[1{:}m,\ 1{:}m]$ с помощью метода квадратных корней [19, §20]. Верхний треугольник матрицы b заменяется на верхний треугольник матрицы N такой, что $N'\times N=b$. Остальные элементы матрицы b остаются неизменными. Матрица c заменяется матрицей $b^{-1}\times c$. Логическая переменная *solve* используется для переключения. Если задано ее значение **true**, то предполагается, что после предыдущего обращения к процедуре *matrdiv* сохранилась матрица N (записанная в массиве b) и требуется только вычисление матрицы $b^{-1}\times c$.

В теле процедуры *matrdiv* локализована процедура *dot*, вычисляющая скалярное произведение векторов a и b. Поскольку процедура *dot* вызывает по наименованию выражения, то алгоритм 197б выходит за рамки сокращенного языка АЛГОЛ-60 [2].

```
procedure matrdiv(m,n,solve)dataresult:(b,c);
        value m,n,solve; integer m,n; Boolean solve;
        array b,c;
begin integer i,j,k;
        real procedure dot(a,b,p,q);
          value q; real a,b; integer p,q;
          begin real s;
            s:=0;
            for p:=1 step 1 until q do s:=s+a×b;
            dot:=s
          end dot;
start: if solve then go to back;
        for i:=1 step 1 until m do
          begin b[i,i]:=sqrt(b[i,i]—dot(b[j,i]↑2,1,j,i—1));
            for j:=i+1 step 1 until m do
              b[i,j]:=(b[i,j]—dot(b[k,i],b[k,j],k,i—1))/b[i,i]
          end формирования верхней треугольной матрицы;
back:  for j:=1 step 1 until n do
```

```
begin
  for i:=1 step 1 until m do
    c[i,j]:=(c[i,j]—dot(b[k,i],c[k,j],k,i—1))/b[i,i];
  for i:=m step—1 until 1 do
    c[i,j]:=(c[i,j]
      —dot(b[i,m+1—k],c[m+1—k,j],k,m—i))/b[i,i]
  end двойной обратной подстановки
end matrdiv;
```

Свидетельство к алгоритму 197а

Алгоритм 197а получен в результате ординарной переработки алгоритма 197 (Wells M. «CACM», 1963, № 8) и внесения в него исправлений, указанных в нижеследующем «Подтверждении» М. Уэллса.

С помощью алгоритма 197а при использовании транслятора ТА-1 была решена система трех уравнений с тремя вариантами правых частей:

	I	II	III
$4x+2y+2z=$	2	—1	3
$2x+2y+2z=$	3	1	2
$2x+2y+3z=$	4	2	3

т. е. процедура *matrdiv* выполнялась для матрицы

$$b=\begin{Vmatrix}4&2&2\\2&2&2\\2&2&3\end{Vmatrix}$$

и для матрицы c, принимающей значения столбцов I, II и III. При *solve*≡**false** были получены результаты

$$b=\begin{Vmatrix}2&1&1\\2&1&1\\2&2&1\end{Vmatrix},$$

$c=(-0.5,1,1)$, $c=(-1,0.5,1)$ и $c=(0.5,-0.5,1)$ для I, II и III вариантов правых частей соответственно. Правильность этих результатов легко проверяется подстановкой в исходную систему уравнений.

Затем эта же система с вариантом I правых частей была последовательно решена при *solve*≡**false** и *solve*≡**true** и дала правильные результаты*.

Подтверждение к алгоритму 197

М. Уэллс (Wells M. «CACM», 1964, № 3)

Процедура была проверена на машине Ferranti Pegasus с помощью АЛГОЛ-транслятора, разработанного компанией Havilland Aircraft в городе Хэтфилде...**. С этими изменениями программа успешно решила несколько небольших контрольных задач.

Данная процедура применима только к симметричным положительно-определенным матрицам, и не делается даже попытки оценить точность результата.

---

* См. также подтверждение в [47, с. 131]. (*Прим. ред.*)

** Далее указываются поправки к алгоритму 197, учтенные при составлении алгоритма 197а. (***Прим. ред.***)

В примечании перед словами «положительно-определенную» нужно вставить слово «симметричную».

Интересно заметить, что первонанальный неверный вариант процедуры может делить одну симметричную матрицу на другую и может быть использован для обращения матриц.

Свидетельство к алгоритму 198б [D1]

Алгоритм 198б («Вычисление кратных интегралов по формулам Ньютона—Котеса») не публикуется здесь, потому что соответствующий алгоритм 198 («CACM», 1963, № 8) не был подтвержден ни в журнале «CACM», ни в расчетах составителей выпуска.

АЛГОРИТМ 199б

Переход от календарной даты к порядковому номеру дня и обратно [Z]

Данный алгоритм включает в себя четыре процедуры, позволяющие превращать календарную дату современного летосчисления в порядковый номер дня и обратно. В двух первых процедурах *jday* и *jdate* порядковый номер дня отсчитывается от начала текущего юлианского периода, начавшегося 1 января 4713 года до н. э., а в двух последних процедурах *kday* и *kdate* — от начала текущего столетия.

Юлианский период — это промежуток времени в 7980 лет, используемый в астрономических и хронологических расчетах. Счет времени в днях юлианского периода предложен в XVI в. французским ученым Ж. Скалигером. Благодаря этому длительность того или иного астрономического или исторического события, выраженная в сутках, может быть определена путем простого вычитания номеров дней, соответствующих концу и началу события.

Процедура *jday* (сокращение от *Julian* — юлианский, *day* — день) по дате григорианского календаря вычисляет порядковый номер дня в юлианском периоде.

Задаются параметры: число *d*, месяц *m* и год *y*. Результат вычисления *j* — порядковый номер дня. Процедура применима для любой даты современного григорианского календаря.

При использовании этого алгоритма в конкретных трансляторах нужно быть уверенным, что в данной машине представимы целые числа до 3×10^6.

```
procedure jday(d,m,y)result:(j);
      value d,m,y; integer d,m,y,j;
begin integer c,ya;
      if m>2 then m:=m—3 else
        begin m:=m+9; y:=y—1 end;
      c:=y÷100; ya:=y—100×c;
      j:=(146097×c)÷4+(1461×ya)÷4
        +(153×m+2)÷5+d+1721119
end jday;
```

Процедура *jdate* (сокращение от *Julian* — юлианский, *date* — дата) превращает порядковый номер для юлианского периода в дату григорианского календаря. Поскольку *j* есть целый параметр процедуры,

то процедура астрономически верна для начала данных суток. Процедура определяет число d, месяц m и год y. При использовании этого алгоритма в конкретном трансляторе нужно быть уверенным, что в данной машине представимы числа до 3×10^6.

```
procedure jdate(j)result:(d,m,y);
      value j; integer j,d,m,y;
begin j:=j—1721119;
      y:=(4×j—1)÷146097; d:=(4×j—1—146097×y)÷4;
      j:=(4×d+3)÷1461; d:=(4×d+7—1461×j)÷4;
      m:=(5×d—3)÷153; d:=(5×d+2—153×m)÷5; y:=100×y+j;
      if m<10 then m:=m+3 else
        begin m:=m—9; y:=y+1 end
end jdate;
```

Процедура *kday* по дате григорианского календаря вычисляет порядковый номер k дня от начала двадцатого века. Входные параметры: d — число, m — месяц, ya — две последние цифры года. Процедура применима для дат от 1 марта 1900 года ($k=1$) до 31 декабря 1999 года ($k=36465$). Юлианский порядковый номер дня (действительный по полуночи) можно получить по формуле $j=k+2415079$.

```
procedure kday(d,m,ya)result:(k);
      value d,m,ya; integer d,m,ya,k;
begin if m>2 then m:=m—3 else
        begin m:=m+9; ya:=ya—1 end;
      k:=(1461×ya)÷4+(153×m+2)÷5+d
end kday;
```

Процедура *kdate* превращает порядковый номер дня от начала данного столетия в дату григорианского календаря. Процедура определяет число d, месяц m и последние две цифры года ya. Процедура применима к интервалу от $k=1$ (1 марта 00 года) до $k=36465$ (31 декабря 99 года) для одного и того же любого столетия. Для XX столетия соотношение между k и юлианским порядковым номером дня есть $j=k+2415079$.

```
procedure kdate(k)result:(d,m,ya);
      value k; integer k,d,m,ya;
begin ya:=(4×k—1)÷1461; d:=(4×k+3—1461×ya)÷4;
      m:=(5×d—3)÷153; d:=(5×d+2—153×m)÷5;
      if m<10 then m:=m+3 else
        begin m:=m—9; ya:=ya+1 end
end kdate;
```

Свидетельство к алгоритму 199а *

Алгоритм 199а получен в результате ординарной переработки алгоритма 199 (Tantzen R. G. «CACM», 1963, № 8) и модификации его согласно нижеприведенному «Подтверждению» Д. К. Оппенгейма.

Алгоритм 199а был транслирован для даты 7 ноября 1957 года. В процедуре *jday* были заданы: $d=7$, $m=11$, $y=1957$. Результат

---

* См. также «Подтверждения к алгоритмам 157а и 199а и замечания к алгоритмам 1а—250а» А. Е. Колесникова, помещенные в приложении 1 к выпуску [49]. (*Прим. ред.*)

j=2436150 совпал с приведенным в БСЭ [18] на с. 391. По этому значению j процедура *jdate* правильно определила дату. При задании в процедуре *kday* параметров d=7, m=11, ya=57 был получен результат k=21071, который удовлетворяет вышеуказанному соотношению $j=k+2415079$. При входе в процедуру *kdate* со значением k=21071 была правильно определена исходная дата.

Аналогичные расчеты были проведены и дали правильные результаты для дат 1 ноября 1962 года и 1 мая 1967 года, для которых контрольные значения k, равные 2437666 и 2439612 соответственно, были получены с помощью таблиц, приведенных на с. 390 БСЭ [18]. Эти три даты проверяют все различные разветвления в процедурах алгоритма (для $m\leqslant 2$, $2<m<10$ и $m\geqslant 10$).

Подтверждение к алгоритму 199

Д. К. Оппенгейм (Oppenheim D. K. «CACM» 1964, № 11)

Алгоритм 199 был переведен на язык Jovial J3 и проверен на машине Philco 2000. Входные данные были получены с помощью датчика случайных чисел, который выдал равномерно распределенные даты от 1583 до 2583 года. Результаты были проверены для 50 различных дат в этом интервале.

Процедуры в том виде, как они описаны, налагают на некоторые параметры ограничения, не вызванные необходимостью. Выражения никогда не могут использоваться в качестве входных параметров процедур. Кроме того, первоначальные значения входных параметров данных процедур могут измениться в процессе их выполнения. Нужно также заметить, что во многих конкретных реализациях АЛГОЛа использование параметров, вызываемых по наименованию, может обойтись гораздо дороже, чем использование вызываемых по значению. Вызов по наименованию является гораздо более мощным инструментом, чем это нужно для большинства параметров этой процедуры. По этим соображениям предлагаются следующие изменения...*

АЛГОРИТМ 200б

Генератор нормально распределенных случайных чисел [G5]

Процедура-функция *normrand* (сокращение от *normal* — нормальный, *random* — случайный) выдает случайное значение по нормальному закону распределения, используя глобальную вещественную процедуру *random*, которая выдает случайное значение согласно равномерному закону распределения на отрезке (—1,1).

Для достаточно точной аппроксимации нормального распределения значение n должно быть больше 10. Однако слишком большие значения n могут вызвать существенное увеличение машинного времени. Параметр m означает математическое ожидание, а *sigma* — среднеквадратическое отклонение.

Автор алгоритма 200 ссылается на работу Р. Хэмминга [2i].

---

* Далее перечисляются четыре модификации, учтенные при составлении алгоритма 199а. (*Прим. ред.*)

```
real  procedure normrand(m,sigma,n);
      value m,sigma,n; real m,sigma; integer n;
begin real sum; integer i;
      sum:=0;
      for i:=1 step 1 until n do sum:=sum+random;
      normrand:=m+sigma×sum×sqrt(3/n)
end normrand;
```

Свидетельство к алгоритму 200а

Алгоритм 200а получен в результате исправления и ординарной переработки алгоритма 200 (George R. «CACM», 1963, № 8). Кроме ошибок, указанных в «Подтверждении к алгоритму 200» (Pike M. C. «CACM», 1965, № 9), было исправлено следующее: в заголовке цикла отсутствовало начальное значение параметра цикла, равное 1. Вышеуказанное «Подтверждение» не публикуется здесь, потому что оно не содержит никакой другой информации.

Алгоритм 200а был транслирован с контрольной программой, генерирующей 5000 случайных чисел x и подсчитывающей количества p_i генерированных чисел, значения которых заключены в интервалах $[x_i, x_{i+1})$, где $x_{i+1}=x_i+0.1$, $i=1, 2, 3, \ldots, 19$, $x_1=-2$. Результаты приведены на рис. 5 и в табл. 40 (для правой ветви кривой распределения).

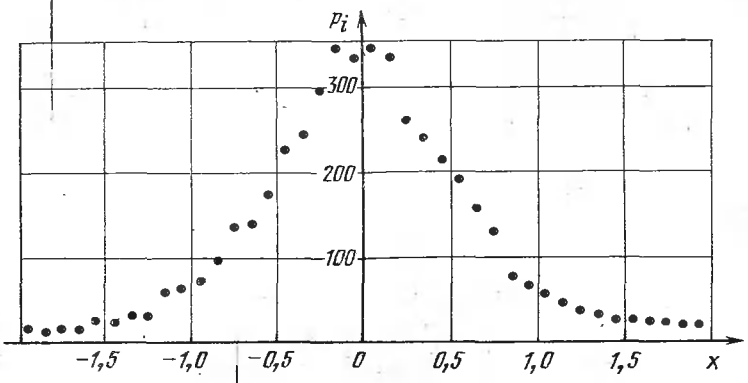

Рис. 5.

Расчеты проводились для $m=0$ и $sigma=1$. В качестве процедуры *random* использовалась процедура *p*1147 транслятора ТА-1М.

Таблица 40

x	p_i	x	p_i	x	p_i	x	p_i
0	346	0.5	191	1.0	59	1.5	27
0.1	316	0.6	156	1.1	46	1.6	24
0.2	264	0.7	130	1.2	38	1.7	23
0.3	243	0.8	78	1.3	32	1.8	20
0.4	217	0.9	73	1.4	23	1.9	18

Замечание рецензента к алгоритму 200а

Наиболее точным и экономичным методом подобного типа является метод «суммирования с поправкой», который позволяет получать нормально распределенное случайное число по сумме не 10 и более равномерно распределенных чисел (как это требуется в предлагаемом алгоритме 200), а всего лишь 5 чисел, причем с большей точностью [37, с. 100], пользуясь формулой $\eta = \omega_5 + 0.01(\omega^3_5 - 3\omega_5)$, где ω_5 — сумма пяти равномерно распределенных на интервале $[-\sqrt{3/5}, \sqrt{3/5}]$ чисел; η — получающееся нормально распределенное число с $m=0$ и $\sigma^2=1$. Алгоритм суммирования с поправкой позволяет более чем в два раза сократить время генерации случайного числа по сравнению с предложенным алгоритмом.

АЛГОРИТМЫ ШАХМАТНОГО ПРОГРАММИРОВАНИЯ

В данном приложении публикуются два алгоритма для решения шахматных задач на мат в n ходов: алгоритм 171б и алгоритм 68CJ. Первый предлагается для непосредственного практического применения к решению задач и исследованию их на однозначность, а также в качестве основы для дальнейшего развития и разработки более сложных алгоритмов шахматного программирования. Поэтому он приведен здесь полностью и с подробным пояснительным текстом.

Второй алгоритм в своих деталях может, по-видимому, заинтересовать только гораздо более узкий круг специалистов. Широкому кругу читателей будет достаточно (как станет ясно из приведенного к алгоритму свидетельства) ознакомиться с пояснительным текстом к этому алгоритму. Поэтому алгоритм 68CJ представлен здесь только полным переводом пояснительного к нему текста и свидетельством, в котором дается перечень обнаруженных в нем ошибок и перевод всех комментариев, встречающихся в АЛГОЛ-программе этого алгоритма. Желающие ознакомиться с этой программой детально, могут найти ее в одной из центральных библиотек, а материалы приведенного здесь свидетельства к алгоритму 68CJ помогут им разобраться в программе даже при отсутствии у них соответствующих знаний английского языка.

АЛГОРИТМ 1716 [Z]

ЭРА-77 — нерекурсивная программа решения шахматных многоходовок

Введение

В выпуске «Библиотека алгоритмов 51б—100б» [48] был опубликован алгоритм 50CJ «Как программировать игру в шахматы», полученный в результате перевода на русский язык, исправления и отладки на ЭВМ алгоритма 50 А. Белла [55]. В этом алгоритме содержалось описание на языке АЛГОЛ-60 программы решения шахматных задач на мат в два хода. Алгоритм 50CJ был предназначен (как это видно из его названия) в основном для того, чтобы помочь начинающим шахматным программистам войти в курс дела. Практическое значение приведенной в нем программы для массового решения шахматных задач не могло быть велико уже по причине ограниченности ее назначения.

Программа решающая шахматные задачи на любое заданное число ходов, могла бы принести существенную пользу, во-первых, шахматным композиторам, во-вторых, людям, готовящим задачи к публикации или для конкурсов, и, наконец, любому решателю шахматных задач, желающему проверить правильность своего решения задачи, ответ к которой отсутствует или же вызывает сомнение. АЛГОЛ-программа, разрабатывавшаяся с такой целью, была опубликована в 1971 году (см. ниже алгоритм 68CJ). Однако, как это видно из ее описания и нижеследующего свидетельства к алгоритму 68CJ, автор алгоритма своей основной цели не достиг, и проблема разработки программы решения многоходовок осталась тогда нерешенной.

С другой стороны, как это уже отмечалось в свидетельстве к алгоритму 50CJ, программа А. Белла после ее исправления и отладки могла служить неплохой основой для дальнейшего совершенствования. В качестве первого шага в этом направлении в выпуске «Библиотека алгоритмов 101б—150б» было опубликовано «Подтверждение к алгоритму 50CJ» [49, приложение 2], содержащее перечень изменений к программе алгоритма 50CJ, делающих ее способной решать задачи на мат в два, три или четыре

хода. В таком четырехходовом варианте алгоритм 50CJ стал пригоден для практического применения к решению большинства шахматных задач на современных машинах.

Однако уже сегодня пользователям шахматной программой может оказаться нужным решать с ее помощью хотя бы простейшие задачи на мат в пять или даже шесть-семь ходов. Кроме того, быстродействие ЭВМ продолжает возрастать, а в ближайшие годы вступят в строй многопроцессорные машины и многомашинные комплексы, которые дадут (при наличии соответствующего алгоритма) возможность решать задачи на мат в любое число ходов. Все это определяет целесообразность публикации шахматной программы решения задач на мат в n ходов, пригодной к практическому применению на большинстве современных ЭВМ и в то же время могущей служить удобной основой для разработки новых еще более сложных алгоритмов шахматного программирования. В качестве таковой и предлагается описываемая ниже программа ЭРА-77 (т. е. Экспериментальный Решающий Алгоритм 1977 года) *.

Хотя программа ЭРА-77 была получена в результате ряда последовательных усовершенствований алгоритма 50CJ, объем и значение внесенных в него изменений настолько велики, что перечисление их стало практически невыполнимым, и возникла необходимость описания всего алгоритма от начала до конца как нового. Фактически в программе алгоритма 50CJ не осталось ни одного места, которое не подвергалось бы существенной переработке и усовершенствованию, а многие участки программы ЭРА-77 написаны заново и являются полностью оригинальными **. Целесообразность столь фундаментальной переработки алгоритма 50CJ становится ясной уже из нижеследующего перечисления полученных при этом усовершенствований.

Кроме главного и очевидного результата, заключающегося в превращении «двухходовой» программы в нерекурсивную «n-ходовую», переработка дала следующее.

1. Повышено быстродействие алгоритма более чем в два раза, и намечены пути его дальнейшего ускорения.

2. Сокращен расход оперативной памяти машины более чем на 2000 машинных слов. В результате стало возможным использовать алгоритм на широко распространенных сегодня машинах с малой памятью, таких как машины типа М-20 [59, 60].

3. Ликвидирована рекурсивность процедур алгоритма 50CJ, возникавшая при включении в него операторов рокировки. Это сделало возможным использование алгоритма в системах трансляции с сокращенного варианта языка АЛГОЛ-60, такого, например, как получивший в нашей стране широкое распространение АЛГАМС [58, 92, 93].

4. Обеспечено наглядное печатание результатов решения не только в форме одного ключевого хода ***, записанного в принятой в шахматной практике нотации, но (при желании пользователя) и в форме полного перечня вторых ходов белыми, обеспечивающих мат в n ходов при любом ответе черных на ключевой ход.

5. Повышено удобство пользования алгоритмом, в частности, введено автоматическое определение характеристики заданной рокировки и обеспечена возможность произвольного порядка ввода в память машины кодов рокируемых фигур.

6. Разработан набор стандартных тестов, позволяющих быстро отыскивать ошибки, которые могут возникать в процессе дальнейших усовершенствований алгоритма.

7. Обеспечена наглядность и удобочитаемость алгоритма несмотря на некоторое неизбежное его усложнение, возникшее в связи с вышеперечисленными усовершенствованиями. Это может дать читателю возможность достаточно быстро разобраться в программе ЭРА-77 и использовать ее в качестве основы для разработки других шахматных алгоритмов ****.

---

* В настоящее время существуют и другие более сложные шахматные программы (см., например, [70, 75, 84]), которые после соответствующего приспособления могли бы использоваться для массового решения шахматных задач. Однако эти программы не публикуются и, следовательно, широкому кругу читателей сейчас не доступны, по-видимому потому, что их авторы преследуют несколько иные цели, а для вышеуказанного приспособления программ и тем более для подготовки их к изданию требуются значительные затраты труда и времени. (*Прим. авт.*)

** Так, полностью оригинально построены процедуры ЕСТЬ ЛИ ШАХ, ПЕЧАТЬ ХОДА, ВЫЧИСЛЕНИЕ ТАБЛИЦ и ОПРЕДЕЛЕНИЕ РОКИРОВКИ (разд. 11, 9, 12 и 8 соответственно) и группы операторов, вычисляющих ходы всеми фигурами (разд. 3.1—3.4), кроме слона. Эти участки программы можно рассматривать как новые самостоятельные алгоритмы. (*Прим. авт.*)

*** В алгоритме 50CJ результат решения печатался в форме позиции, возникавшей на доске после выполнения ключевого хода [48, с. 116]. (*Прим. авт.*)

**** В настоящее время (1978 год) автором программы ЭРА-77 разрабатывается на ее основе алгоритм проблемно-шахматной композиции, публикация которого предполагается в одном из следующих выпусков.

Как и все другие опубликованные алгоритмы решения шахматных задач, данный алгоритм решает задачи путем полного перебора всех ходов, которые могут быть сделаны как белыми, так и черными. Ограничение перебора, какое делается в игровой шахматной программе, в алгоритмах решения задач недопустимо, поскольку ход, который в обычной игре представляется наименее выгодным, чаще всего как раз и оказывается здесь ключевым, ибо в этом и состоит интригующая красота шахматной задачи. В программах решения шахматных задач можно исключать из рассмотрения только такие ходы, которые являются бесспорно излишними. Так, в алгоритме 50CJ для определения того, что после выполнения некоторого хода белые объявили шах черному королю, вычислялись в возникшей при этом позиции все возможные ходы белых с целью проверки, нет ли среди них такого, который сопровождается взятием черного короля. Для ограничения такого перебора ходов автором алгоритма 171б была специально составлена новая процедура ЕСТЬ ЛИ ШАХ, (описание этой процедуры приведено ниже в разд. 11), определяющая наличие шаха по взаимному расположению белых фигур и черного короля без вычисления каких-либо ходов белыми фигурами. Использование этой процедуры в программе ЭРА-77 дало ускорение работы алгоритма на 25—30% для тех 40 контрольных задач, на которых отлаживался алгоритм (эти задачи приведены в разд. 14).

Нижеследующее описание алгоритма 171б построено так, чтобы читателю, желающему воспользоваться программой ЭРА-77 для решения задач, но не собирающемуся вносить в алгоритм никаких изменений не было необходимости разбирать весь пояснительный текст до конца. Такому читателю достаточно ознакомиться только с нижеследующим разд. 1 пояснительного текста.

1. Кодировка позиции и ввод ее в память машины

При осуществлении перебора ходов электронная цифровая машина может, естественно, оперировать только с числовыми кодами, поэтому разработка любого шахматного алгоритма должна начинаться с выбора кодировки для шахматных фигур и полей шахматной доски.

Коды белых фигур в алгоритме 171б совпадают с кодами фигур алгоритма 50CJ, т. е. пешке (+П) соответствует код 1, коню (+К) — код 2, слону (+С) — код 3, ладье (+Л) — код 4, ферзю (+Ф) — код 5 и королю (+КР) — код 6. Но в отличие от алгоритма 50CJ коды черных фигур алгоритма 171б отличаются от кодов белых фигур знаками, т. е. черной пешке (—П) соответствует код —1, черному коню (—К) — код — 2 и т. д.*

Такое различение кодов фигур по цвету дало возможность отображать любую позицию в одном массиве ДОСКА [1 : 64] **. Кодировка фигур последовательными целыми числами от 1 до 6 дает возможность рационально строить программу, и в частности, пользоваться переключателем К ХОДАМ (в процедуре ПЕРЕПИСЬ ХОДОВ, описанной в разд. 3).

Кодировка полей доски в алгоритме 171б не отличается от кодировки полей, принятой как в алгоритме 50CJ, так и во многих других шахматных алгоритмах. Она изображена на рис. 6. Однако в отличие от других алгоритмов здесь кроме номера (кода) каждого поля введены в рассмотрение его квазикоординаты (см. на рис. 6 квазиабсциссы и квазиординаты). Сумма $i+j$ таких квазикоординат всегда равна коду соответствующего им поля. Наоборот, по коду поля легко определить его квазикоординаты $j=$(ПОЛЕ—1)$\div$8$\times$8 и $i=$ПОЛЕ—j. Это нововведение позволило намного рациональнее построить процедуры ПЕРЕПИСЬ ХОДОВ, ЕСТЬ ЛИ ШАХ и ПЕЧАТЬ ПОЛЯ, обеспечило экономию машинного времени и машинной памяти.

Пользуясь вышеуказанной кодировкой, любую шахматную позицию можно ввести в память машины в виде последовательности целых чисел ***, которая будет именовать-

---

* Здесь в круглых скобках после названий фигур указываются их символические обозначения, которые будут использоваться в дальнейшем для изображения позиции и ходов. (*Прим. авт.*)

** В алгоритме 50CJ для изображения позиции использовались два массива: ДОСКА БЕЛЫХ [1 : 65] и ДОСКА ЧЕРНЫХ [1 : 65]. Замена этих двух массивов одним массивом ДОСКА дала возможность в алгоритме 171б исключить из процедуры ОБЗОР ХОДОВ параметры-массивы СВОЯ ДОСКА и ЧУЖАЯ ДОСКА, а из процедуры ВЫПОЛНЕНИЕ ХОДА параметры-массивы ДОСКА СВОИХ и ДОСКА ЧУЖИХ. Это обеспечило экономию памяти и повысило быстродействие алгоритма. (*Прим. авт.*)

*** Процедуру ввода можно было бы записать для системы БЭСМ-АЛГОЛ и так, чтобы позиция вводилась в виде последовательности обычных буквенных обозначений фигур и полей, но такая процедура была бы гораздо более сложной и, следовательно, менее удобной для первичного ознакомления. (*Прим. авт.*)

ся в дальнейшем «информацией о задаче». Например, первая из двух задач, приведенных в пояснительном тексте к алгоритму 50CJ [48, с. 88, рис. 3] (в дальнейшем эта задача будет здесь именоваться, как «отладочная № 1»), записываемая в обычной шахматной нотации следующим образом:

белые (11 фигур): Крb2, Фf2, Ла5, Сс7, Са8, Кb3, Кf8, d2, f3, h4, g5,

черные (7 фигур): Крf5, Фе5, Кd5, d3, d4, f4, f6; может подаваться в читающее устройство в виде следующей информации о задаче:

2; 11; 6; 10; 5; 14; 4; 33; 3; 51; 3; 57; 2; 18;
2; 62; 1; 12; 1; 22; 1; 32; 1; 39;
7; 6; 38; 5; 37; 2; 36; 1; 20; 1; 28; 1; 30; 1; 46;

Программа ЭРА-77 позволяет вводить коды фигур данного цвета в произвольном порядке следования, т. е. пары чисел в вышеприведенной информации о задаче, начиная с пары 6; 10; и кончая парой 1; 39; можно менять местами. Точно так же взаимозаменяемы пары, начиная с 6; 38; и кончая 1; 46;. Однако первое число в информации о задаче всегда должно означать число ходов в задаче, второе — количество белых фигур, затем должна идти последовательность пар чисел, первое из которых означает код белой фигуры и второе — код занимаемого ею поля. За последовательностью пар чисел с кодами белых фигур должно вводиться число черных фигур и, наконец, последовательность числовых пар, состоящих из абсолютных значений кодов черных фигур и занимаемых ими полей. Знак минус нижеприведенная процедура присваивает кодам черных фигур перед записью их в массив ДОСКА. Так после ввода вышеуказанной позиции в машину этот массив будет иметь значение

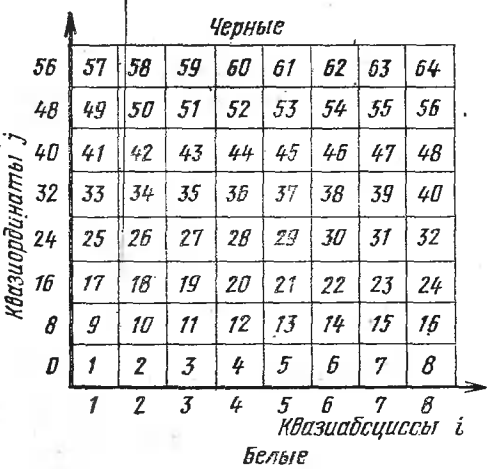

Рис. 6. Кодировка полей шахматиой доски и квазикоординаты.

0,0,0,0,0,0,0,0, 0,6,0,1,0,5,0,0, 0,2,0, —1,0,1,0,0,
0,0,0, —1,0, —1,0,1, 4,0,0, —2, —5, —6,1,0, 0,0,0,0,0, —1,0,0,
0,0,3,0,0,0,0,0, 3,0,0,0,0,2,0,0

Если расположить значения элементов массива ДОСКА по соответствующим полям шахматной доски, то получится закодированная диаграмма исходной позиции вышеуказанной задачи, как это изображено на рис. 7, которая полностью соответствует исходной позиции данной задачи (рис. 8).

Для того чтобы программа ЭРА-77 могла быстро находить на доске фигуры (без просмотра всех полей доски), адреса вводимых фигур (т. е. коды занимаемых ими полей) запоминаются процедурой ВВОД ПОЗИЦИИ в массиве АДРЕСА [—16 : 16] *. При этом адреса белых фигур записываются в порядке их ввода в «положительной ветви» [1 : 16] массива АДРЕСА, а адреса черных фигур — в «отрицательной ветви» [—1 : —16] этого массива. Так, после ввода информации о дачной задаче (в вышеуказанном порядке следования фигур) массив АДРЕСА будет иметь значение, показанное на рис. 9.

Заполнение массивов ДОСКА и АДРЕСА информацией о задаче легко проследить непосредственно по описанию процедуры ВВОД ПОЗИЦИИ, приведенному ниже. (При этом нужно иметь в виду, что число n ходов в задаче вводится в память машины

* В алгоритме 50CJ для этой цели использовались два массива: ПОЛЯ БЕЛЫХ [0 : 16] и ПОЛЯ ЧЕРНЫХ [0 : 16]. Замена этих двух массивов одним массивом АДРЕСА дала возможность исключить из процедур ОБЗОР ХОДОВ и ВЫПОЛНЕНИЕ ХОДА параметры-массивы СВОИ ПОЛЯ, ПОЛЯ СВОИХ и ПОЛЯ ЧУЖИХ. (*Прим. авт.*)

перед обращением к процедуре ВВОД ПОЗИЦИИ.) При вводе позиции (как и в ряде других мест программы ЭРА-77) используются процедуры ввода — вывода системы БЭСМ-АЛГОЛ [62, 90].

```
procedure ВВОД ПОЗИЦИИ;
begin input (ЧИСЛО БЕЛЫХ);
      for i:=1 step 1 until ЧИСЛО БЕЛЫХ do
        begin input (ФГ, ПОЛЕ); ДОСКА[ПОЛЕ]:=ФГ; АДРЕСА[i]:=ПОЛЕ;
          if ФГ=КОРОЛЬ then БКР:=i else
            if ФГ=ЛАДЬЯ∧ПОЛЕ=8 then БЛК:=i else
              if ФГ=ЛАДЬЯ∧ПОЛЕ=1 then БЛД:=i
        end i;
      input (ЧИСЛО ЧЕРНЫХ);
      for i:=1 step 1 until ЧИСЛО ЧЕРНЫХ do
        begin input (ФГ, ПОЛЕ); ДОСКА [ПОЛЕ]:=—ФГ; АДРЕСА [—i]:=ПОЛЕ;
          if ФГ=КОРОЛЬ then ЧКР:=—i else
            if ФГ=ЛАДЬЯ∧ПОЛЕ=64 then ЧЛК:=—i else
            if ФГ=ЛАДЬЯ∧ПОЛЕ=57 then ЧЛД:=—i
        end i;
КОНТРОЛЬ ВВОДА:
      output('×/','T','РЕШЕНИЕ—ПО—ПРОГРАММЕ—','T', ИМЯ ПРОГРАММЫ[1],'//',
      'T','ИСХОДНАЯ—ПОЗИЦИЯ—ЗАДАЧИ— 'T',ИМЯ ЗАДАЧИ[1], '//',
        'T','НА —МАТ—B','Z3DB',n);
      if n=1 then output('T','ХОД','/') else
      if n<5 then output('T','ХОДА','/') else output('T','ХОДОВ','/');
      ПЕЧАТЬ ПОЗИЦИИ; output('/')
end ВВОД ПОЗИЦИИ;
```

Используемые в процедуре ВВОД ПОЗИЦИИ переменные КОРОЛЬ и ЛАДЬЯ в начале работы программы, получают и в процессе работы программы сохраняют одно и то же значение (6 и 4 соответственно). Такие переменные предназначены для обеспечения наглядности программы и будут именоваться в дальнейшем «иллюстративными переменными».

3					2			64
		3						56
					-1			48
4			-2	-5	-6	1		40
			-1		-1		1	32
	2		-1		1			24
	6		1		5			16
								8
1	2	3	4	5	6	7	8	

Рис. 7. Двумерное изображение массива ДОСКА после ввода позиции отладочной задачи № 1.

8	С					К		
7			С					
6						-П		
5	Л			-К	-Ф	-Кр	П	
4				-П		-П		П
3		К		-П		П		
2		Кр		П		Ф		
1								
	a	b	c	d	e	f	g	h

Рис. 8. Диаграмма исходной позиции отладочной задачи № 1.

Поскольку места адресов фигур в массиве АДРЕСА остаются в процессе выполнения программы неизменными, то фигуры, участвующие в данной задаче, после выполнения этой процедуры можно считать пронумерованными в порядке ввода их кодов в память машины. В процессе поиска ключевого хода программа ЭРА-77 рассматривает все возможные первые ходы каждой из белых фигур. При этом фигуры выбираются в порядке убывания их номеров. Так, например, после вышеуказанного ввода информации об отладочной задаче № 1 программа сначала рассмотрит все возможные ходы пешкой g5, затем пешкой h4 и т. д. Учитывая это свойство программы, пользователь может управлять поиском ключевого хода с целью сокращения времени

решения задачи, если он будет вводить в память машины в последнюю очередь коды тех фигур, один из ходов которыми представляется ему с наибольшей вероятностью ключевым *.

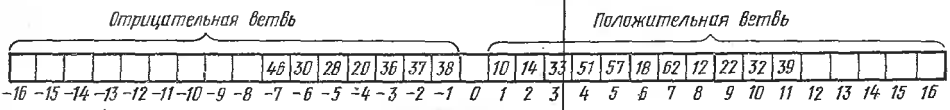

Рис. 9. Массив АДРЕСА после ввода отладочной задачи № 1.

В приведенной выше процедуре ВВОД ПОЗИЦИИ два условных оператора, начинающихся с условия if ФГ=КОРОЛЬ then, предназначены для запоминания номеров короля и ладей, находящихся в позициях, допускающих рокировку. Значения этих номеров (БКР, БЛК и т. д.) используются позже в процедурах ЗАПИСЬ РОКИРОВКИ и ЕСТЬ ЛИ ШАХ, описываемых ниже. Пояснения к этим операторам даны в описании первой из этих процедур (разд. 8).

Последняя группа операторов в процедуре ВВОД ПОЗИЦИИ, начинающаяся с метки КОНТРОЛЬ ВВОДА, предназначена для обеспечения контроля правильности введенной информации о задаче. Для этого печатается (путем обращения к процедуре ПЕЧАТЬ ПОЗИЦИИ) диаграмма введенной позиции. Кроме того, для удобства пользования алгоритмом перед диаграммой задачи печатается информация об используемом в данный момент варианте программы и название самой задачи.

РЕШЕНИЕ ПО ПРОГРАММЕ ЭРА-77 от 9.8.77
ИСХОДНАЯ ПОЗИЦИЯ ЗАДАЧИ ОТЛАДОЧНАЯ № 1
НА МАТ В 2 ХОДА

+С	0	0	0	0	+К	0	0
0	0	+С	0	0	0	0	0
0	0	0	0	0	—П	0	0
+Л	0	0	—К	—Ф	—КР	П	0
0	0	0	—П	0	—П	0	П
0	+К	0	—П	0	+П	0	0
0	+КР	0	+П	0	+Ф	0	0
0	0	0	0	0	0	0	0

Рис. 10. Пример информации, печатаемой процедурой ВВОД ПОЗИЦИИ.

Печать информации о варианте программы («имя программы») окажется полезной для тех пользователей, которые будут вносить в программу изменения с целью ее усовершенствования или приспособления к применяемой ими транслирующей системе. Новое имя программы задается пользователем обычно при каждом новом изменении в программе и вводится в память машины в виде строки, содержащей не более 20 символов (включая кавычки).

Печать названия задачи («имя задачи») полезна каждому пользователю программой. Она задается пользователем одновременно с информацией о задаче и вводится в память машины в виде строки, содержащей не более 30 символов (включая кавычки). Так, если при решении вышеприведенной отладочной задачи № 1 пользователь введет в память машины в качестве имени программы строку

'ЭРА—77— ОТ— 9.8.77'

а в качестве имени задачи строку

'ОТЛАДОЧНАЯ_№ 1'

то перед началом решения задачи процедура ВВОД ПОЗИЦИИ напечатает информацию, изображенную на рис. 10.

* Более подробно такой метод ускорения решения задач описан в приложении 2 к третьему выпуску [49] данной серии. (*Прим. авт.*)

Таким образом, кроме программы ЭРА-77 и информации о задаче пользователь должен вводить в память машины еще и другие данные. Общий порядок ввода полной информации, необходимой для решения задачи с помощью программы ЭРА-77, определяется схемой, показанной на рис. 11.

«Вспомогательные символы», указанные в схеме на рис. 11, необходимы для процедур ПЕЧАТЬ ПОЗИЦИИ, ПЕЧАТЬ ПОЛЯ и ПЕЧАТЬ ХОДА. Они вводятся во всех случаях решения задач с помощью двух операторов с меткой ВВОД ВСПОМО-

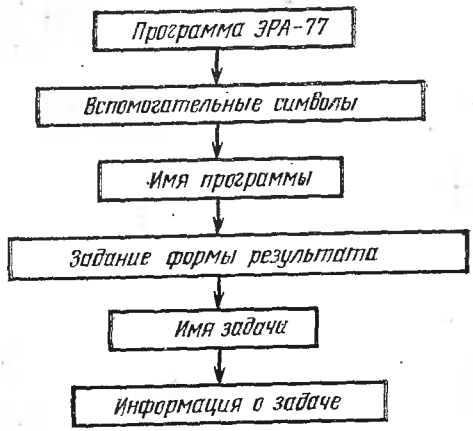

Рис. 11. Порядок ввода полной информации, необходимой для решения по программе ЭРА-77.

ГАТЕЛЬНЫХ СИМВОЛОВ (записанных в начале ведущей программы в разд. 7) в форме следующих двух последовательностей строк:

'П␣'‘К␣'‘С␣'‘Л␣'‘Ф␣'‘КР'

'A''B''C''D''E''F''G''H'

Легко заметить, что строки первой из этих последовательностей (вводимой в массив ИМЯ ФИГУРЫ [4:27]) содержат символические обозначения фигур, а строки второй последовательности (вводимой в массив БУКВА [3:26]) — буквенные обозначения вертикальных линий доски.

«Задание формы результата» в схеме на рис. 11 означает ввод символа «true;», если пользователь желает, чтобы кроме ключевого хода программа выдавала в качестве результата и все вторые ходы белыми (приводящие к мату при любом ответе черных на ключевой ход). Если же нужен только ключевой ход, то задание формы результата осуществляется вводом символа «false;». (Более подробно вывод вторых ходов описан в разд. 10.)

Если вся вышеперечисленная информация была составлена правильно и введена в память машины в указанном на рис. 11 порядке, то дальнейшее выполнение программы происходит уже автоматически.

2. Процедура ПЕЧАТЬ ПОЗИЦИИ

Для печатания диаграммы позиции в алгоритме 171б была составлена более короткая процедура, чем описанная в алгоритме 50CJ. Сокращение было достигнуто, во-первых, благодаря использованию одного массива ДОСКА вместо двух: ДОСКА БЕЛЫХ и ДОСКА ЧЕРНЫХ, а во-вторых, благодаря использованию вспомогательных символов (см. разд. 1).

```
procedure ПЕЧАТЬ ПОЗИЦИИ;
begin for j:=56 step -8 until 0 do
    begin output('///');
      for i:=1 step 1 until 8 do
        begin output('B'); k:=4×ДОСКА[i+j];
          if k>0 then
            output('T','+','T', ИМЯ ФИГУРЫ [abs(k)]) else
```

```
            if k=0 then output('T',—0— ') else
              output('T','—','T',ИМЯ ФИГУРЫ[abs(k)])
        end i
    end j;
    output('//')
end ПЕЧАТЬ ПОЗИЦИИ;
```

Эта процедура по коду фигуры (ДОСКА [*i*+*j*]) отыскивает в массиве ИМЯ ФИГУРЫ соответствующую строку, содержащую символ этой фигуры, и печатает его на соответствующем поле (*i*+*j*) диаграммы (см, например, рис. 10). Первый заголовок цикла вычисляет квазиординату *j* горизонтальной линии доски (начиная с верхней линии). Второй заголовок вычисляет квазиабсциссу *i* поля на данной горизонтальной линии. Первый оператор *output* ('//') обеспечивает пробел между горизонтальными линиями печатаемой диаграммы, а оператор *output* ('B') дает пробел между вертикальными линиями этой диаграммы.

3. Процедура ПЕРЕПИСЬ ХОДОВ

При осуществлении перебора ходов данной задачи шахматная программа обнаруживает непригодность того или иного испытуемого хода чаще всего только после выполнения серии последующих ходов. Таким образом, после серии изменений на шахматной доске программа бывает вынуждена неоднократно возвращаться к каждой данной позиции для апробирования новых возможных в этой позиции ходов.

Программа ЭРА-77 вычисляет с помощью процедуры ПЕРЕПИСЬ ХОДОВ все возможные в данной позиции ходы фигурами данного цвета один раз сразу же после возникновения этой позиции, т. е. сразу же после хода противника, и запоминает их в одномерном массиве СПИСОК*. В результате этого при отходе от данной позиции (для выполнения последующих ходов) программе достаточно запоминать только значение индекса (в массиве СПИСОК) выполненного хода. После возврата к данной позиции программа уже не вычисляет никаких ходов, а продолжает апробировать ранее вычисленные ходы путем выборки их из массива СПИСОК, начиная именно с этого вышеуказанного значения индекса. Вычисление всех возможных ходов фигурами данного цвета сразу же после хода противника дает возможность совместить это вычисление с проверкой законности сделанного противником хода поскольку одновременно с вычислением ходов проверяется, не сопровождаются ли они взятием короля противника. Такое совмещение ускоряет работу программы.

Поскольку процедура ПЕРЕПИСЬ ХОДОВ работает как для белых, так и для черных, то первый выполняемый оператор этой процедуры (см. описание процедуры ниже) определяет значение переменной ЧИСЛО СВОИХ, равное (по модулю) количеству фигур рассматриваемого цвета. Используемый в этом операторе параметр ЦВЕТ имеет значение 1, если процедура вычисляет ходы фигур белого цвета, и значение — 1, если черного цвета. Иллюстративная переменная БЕЛ в процессе всего выполнения программы имеет одно и то же значение, равное 1.

Новые позиции возникают на доске одна за другой после выполнения каждого нового полухода. Для каждой из этих позиций процедура ПЕРЕПИСЬ ХОДОВ вычисляет соответствующую группу возможных в ней ходов и размещает такие группы в массиве СПИСОК одну за другой, начиная каждую элементом с индексом *r*, получающим в начале вычисления группы значение, равное входному значению параметра РУБЕЖ. Эту передачу значения осуществляет второй оператор процедуры (*r* := =РУБЕЖ). После вычисления и записи в СПИСОК каждого нового хода процедура увеличивает значение *r* на единицу так что в конце выполнения процедуры *r* будет равно индексу первого свободного элемента массива СПИСОК, начиная с которого будет записывать новую группу ходов для следующего полухода. Это выходное значение *r* (которое будет здесь называться границей вычисленой группы ходов) передается параметру РУБЕЖ последним оператором процедуры (РУБЕЖ := =*r*). Поскольку при обращении к процедуре ПЕРЕПИСЬ ХОДОВ формальный параметр РУБЕЖ заменяется фактическим параметром РУБЕЖ ХОДОВ [ПОЛУХОД], то в результате последовательных обращений к этой процедуре в массиве РУБЕЖ ХОДОВ окажутся записанными все границы вычисленных групп ходов, соответствующие каждому выполненному полуходу. По значениям этих границ

---

* Форма массива СПИСОК в алгоритме 171б такая же, как в алгоритме 50CJ, однако вычисление ходов на иоля, заполняющие СПИСОК, производится совершенно по-новому, так, чтобы программа могла обходиться без использования специальных таблиц ходов, отнимавших в алгоритме 50CJ значительную часть оперативной памяти. (*Прим. авт.*)

программа определяет в дальнейшем моменты окончания апробирования ходов данной группы.

Третий оператор процедуры ПЕРЕПИСЬ ХОДОВ, т. е. оператор цикла с заголовком

for p : =ЧИСЛО СВОИХ step — ЦВЕТ until ЦВЕТ do

осуществляет последовательный перебор всех фигур данного цвета, участвующих в задаче, вычисляет для каждой фигуры подгруппу всех возможных в данной позиции ее ходов и записывает эту подгруппу в массив СПИСОК.

При обращении к процедуре ПЕРЕПИСЬ ХОДОВ для вычисления ходов белыми фигурами (т. е. при ЦВЕТ=1) отладочной задачи № 1 этот заголовок будет выполняться как

for p : =11 step — 1 until 1 do

а при вычислении ходов черными (т. е. при ЦВЕТ=—1) будет работать как

for p : =—7 step 1 until —1 do

В результате первый оператор тела этого цикла (т. е. оператор ПОЛЕ : =АДРЕСА [p]) будет последовательно присваивать переменной ПОЛЕ коды полей, занимаемых каждой из фигур рассматриваемого цвета. Второй оператор тела цикла вычисляет код ФГ фигуры, занимающей вышеуказанное поле. Следующий далее условный оператор определяет, занято ли сейчас это поле «своей» фигурой, т. е. фигурой рассматриваемого цвета. Если окажется, что ЦВЕТ×ФГ≤0, то фигура с номером p была ранее взята и, следовательно, нужно переходить к попытке вычисления ходов следующей фигуры (с номером p — ЦВЕТ). Величина ЦВЕТ×ФГ будет в дальнейшем именоваться «относительным кодом» фигуры, занимающей данное поле.

Если поле занято «своей» фигурой (относительный код ее положителен), то формирование подгруппы выполнимых этой фигурой ходов начинается с того, что в СПИСОК заносится информация о фигуре: ее номер p, абсолютное значение ее кода ФГ и код занимаемого ею поля. Затем для вычисления выполнимых фигурой ходов определяются ее квазиордината jo и квазиабсцисса io, и с помощью оператора **goto** К ХОДАМ [ФГ] совершается переход к вычислению самих выполнимых ходов.

При решении вышеприведенной отладочной задачи № 1 сначала рассматривается одиннадцатая фигура. Она занимает в массиве ДОСКА поле 39, содержащее пешку (код 1). Поэтому в начале первой группы ходов записываются числа 11, 1, 39. Затем вычисляются и заносятся в СПИСОК выполнимые ходы этой пешки, т. е. 47 и 46.

Вычисление выполнимых ходов любой фигуры всегда заканчивается переходом к метке ДАЛЕЕ. Оператор с этой меткой проверяет, был ли вычислен для данной фигуры хотя бы один ход. Если фигура не может сделать ни одного хода, то информация о ней затирается (вследствие выполнения оператора r : =r—2) информацией о новой фигуре. Если фигура может сделать хотя бы один ход, то в конце подгруппы ее ходов (после их вычисления) оператором СПИСОК [r+1] : =—ПОЛЕ записывается код исходного поля фигуры, взятый со знаком минус (в данном примере записывается —39). Этот отрицательный код будет в дальнейшем использоваться программой в качестве признака окончания подгруппы ходов (т. е. перечня ходов данной фигуры). После засылки в СПИСОК кода — ПОЛЕ процедура переходит к рассмотрению следующей в массиве АДРЕСА фигуры (с номером p — ЦВЕТ). Так продол-

Таблица 41

Название фигуры	Первая группа ходов в массиве СПИСОК				
	Номер фигуры p	Код фигуры	Исходное поле	Поля выполнимых ходов	Признаки конца подгруппы
Пешка	11	1	39	47, 46	—39
Пешка	10	1	32	40	—32
Конь	7	2	62	47, 45, 56, 52	—62
Конь	6	2	18	35, 3, 1, 28	—18
Слон	5	3	57	50, 43, 36	—57
Слон	4	3	51	60, 42, 58, 44, 37	—51
Ладья	3	4	33	34, 35, 36, 41, 49, 25, 17, 9, 1	—33
Ферзь	2	5	14	15, 16, 13, 6, 23, 5, 21, 28, 7	—14
Король	1	6	10	2, 9, 1, 17, 11, 3, 19	—10

жается до исчерпания соответствующей (т. е. «положительной» при ЦВЕТ=1 или «отрицательной» при ЦВЕТ=—1) ветви массива АДРЕСА. Получающийся в результате список чисел (перед выполнением первого полухода) в отладочной задаче № 1 приведен в табл. 41.

В табл. 41 пропущены две пешки (p=8 и p=9), потому что они не имеют выполнимых ходов. В массив СПИСОК заносятся и незаконные ходы (такие, как ходы королем на поля 11 и 19). Незаконность таких ходов обнаружится программой позже, когда будут делаться попытки их выполнения процедурой ВЫПОЛНЕНИЕ ХОДА (описание ее приведено ниже в разд. 4).

Формирование подгруппы выполнимых ходов лучше всего можно проследить непосредственно по нижеприведенному описанию процедуры ПЕРЕПИСЬ ХОДОВ. В этой процедуре для наглядности и удобочитаемости группы операторов с метками ПЕШКОЙ, КОНЕМ, КОРОЛЕМ, ЛАДЬЕЙ, ФЕРЗЕМ и СЛОНОМ заменены комментариями, в которых указаны разделы, где дается описание этих групп операторов. Перед трансляцией программы эти группы операторов нужно, разумеется, вставить на места соответствующих им комментариев.

```
procedure ПЕРЕПИСЬ ХОДОВ(ЦВЕТ, РУБЕЖ, ПРЕРЫВАНИЕ);
    value ЦВЕТ; integer ЦВЕТ, РУБЕЖ; label ПРЕРЫВАНИЕ;
begin integer i0, j0, ЧИСЛО СВОИХ,r;
switch К ХОДАМ:=ПЕШКОЙ,КОНЕМ,СЛОНОМ,ЛАДЬЕЙ,ФЕРЗЕМ,КОРОЛЕМ;
    ЧИСЛО СВОИХ:= if ЦВЕТ=БЕЛ then ЧИСЛО БЕЛЫХ else
      —ЧИСЛО ЧЕРНЫХ;
    r:=РУБЕЖ;
    for p:=ЧИСЛО СВОИХ step —ЦВЕТ until ЦВЕТ do
      begin ПОЛЕ:=АДРЕСА[p]; ФГ:=ДОСКА[ПОЛЕ];
        if ЦВЕТ×ФГ≤0 then go to НОВАЯ ФИГУРА;
        СПИСОК[r]:=p; СПИСОК[r+1]:=ФГ:=abs(ФГ);
        r:=r+2; СПИСОК[r]:=ПОЛЕ;
        j0:=(ПОЛЕ—1)÷8×8; i0:=ПОЛЕ—j0;
        go to К ХОДАМ [ФГ];
ПЕШКОЙ:; comment См. раздел 3.1; go to ДАЛЕЕ;
КОНЕМ:; comment См. раздел 3.2; go to ДАЛЕЕ;
КОРОЛЕМ:; comment См. раздел 3.3; go to ДАЛЕЕ;
ЛАДЬЕЙ:ФЕРЗЕМ:; comment См. раздел 3.4;
        if ФГ=ЛАДЬЯ then go to ДАЛЕЕ;
СЛОНОМ:; comment См. раздел 3.5;
ДАЛЕЕ: if СПИСОК[r]=ПОЛЕ then r:=r—2 else
          begin СПИСОК[r+1]:=—ПОЛЕ; r:=r+2 end;
НОВАЯ ФИГУРА:
    end p;
    РУБЕЖ:=r
end ПЕРЕПИСЬ ХОДОВ;
```

3.1. Вычисление ходов пешкой

Ходы пешкой, выполнимые в данной позиции, вычисляются с помощью нижеприведенной группы операторов, помещающейся в процедуре ПЕРЕПИСЬ ХОДОВ вслед за меткой ПЕШКОЙ.

```
    k:=ПОЛЕ; u:=8×ЦВЕТ;
ЕЩЕ ХОД: k:=k+u;
    if ДОСКА[k]≠0 then go to ПРОБА ВЗЯТИЯ;
    r:=r+1; СПИСОК[r]:=k; w:=k+12×ЦВЕТ;
    if w≥29∧w≤36 then go to ЕЩЕ ХОД;
ПРОБА ВЗЯТИЯ: j:=j0+u; m:=НА ПРОХОДЕ[ПОЛУХОД—1];
    for i:=i0—1,i0+1 do
      begin k:=i+j; w:=ЦВЕТ×ДОСКА[k];
        if i≥1∧i≤8∧(w<0∨k=m) then
          begin if w=—КОРОЛЬ then go to ПРЕРЫВАНИЕ;
            r:=r+1; СПИСОК[r]:=k
          end
      end i;
```

Оператор с меткой ЕЩЕ ХОД вычисляет код поля, на которое может сходить пешка (без взятия фигуры противника), добавляя к коду или вычитая (в зависимости от цвета пешки) из кода исходного поля число 8. Следующий далее условный оператор проверяет, свободно ли вычисленное поле k. Если да, то код этого поля записывается в очередной свободный элемент массива СПИСОК, а затем начинается проверка, не находится ли пешка в своей начальной позиции, т. е. имеет ли она право сделать

еще и ход на два поля вперед. Для сокращения такой проверки вычисляется вспомогательная величина $w=k+12\times$ЦВЕТ. Для пешки, находящейся в начальной позиции, эта величина всегда заключена в интервале [29, 36] независимо от цвета пешки. В результате определение права пешки на удвоенный ход сводится к проверке, находится ли w в указанном интервале. Если да, то совершается возврат к метке ЕЩЕ ХОД, где к вычисленному перед этим коду поля k еще раз добавляется число ± 8. Снова делается проверка, свободно ли вычисленное поле, и если да, то и это второе поле заносится в СПИСОК. После этого неизбежно начинают выполняться операторы с меткой ПРОБА ВЗЯТИЯ.

Для определения возможности хода пешки со взятием сначала к квазиординате пешки $j0$ добавляется $u=\pm 8$. [Например, в отладочной задаче № 1 для пешки, находящейся на поле 39, будет вычислено значение $j=j0+u=32+8=40$, поскольку $j0=(39-1)+8\times 8=32$.] Далее для проверки возможности взятия пешкой «влево» оператор цикла сначала от квазиабсциссы пешки $i0$ вычитает 1. (Для пешки, находящейся на поле 39, $i0=$ПОЛЕ$-j0=39-32=7$, следовательно, оператор цикла вычислит $i=7-1=6$.) Сумма $i+j$ вычисленных таким образом величин дает код поля k, на которое пешка может сходить со взятием (для рассматриваемой пешки $k=46$). Далее вычисляется относительный код w фигуры, занимающей поле k [для рассматриваемого примера $w=1\times(-1)=-1$]. Затем выясняется, не находится ли пешка на краю доски (проверкой неравенства $1\leqslant i\leqslant 8$, занято ли поле k фигурой противника (проверкой неравенства $w<0$) и может ли данная пешка взять пешку противника на проходе (проверкой равенства $k=m$). При удовлетворительном результате таких проверок проверяется еще, не является ли фигура на поле k королем противника. Если нет, то код поля k заносится в СПИСОК. Если же на поле k находится король, то происходит выход из процедуры ПЕРЕПИСЬ ХОДОВ к формальной метке ПРЕРЫВАНИЕ. Значение m здесь равно коду поля, через которое «перепрыгнула» пешка противника на предыдущем полуходе, если тогда был совершен какой-либо ход пешкой на два поля вперед. Если же на предыдущем полуходе удвоенного хода пешкой не было, то $m=0$. Заполнение массива НА ПРОХОДЕ происходит в процедуре ВЫПОЛНЕНИЕ ХОДА (описание ее приведено ниже в разд. 4).

Если при выполнении предыдущих операций не было выхода на метку ПРЕРЫВАНИЕ, то вышеуказанный оператор цикла вычисляет $i=i0+1$ для проверки возможности взятия пешкой «вправо» и для нового значения поля k снова выполняет все вышеперечисленные операции.

3.2. Вычисление ходов конем

Ходы конем, выполнимые в данной позиции, вычисляются с помощью нижеприведенной группы операторов, помещающейся в процедуре ПЕРЕПИСЬ ХОДОВ вслед за меткой КОНЕМ.

```
for u:=1,2 do
  for i:=i0+u,i0—u do
    if i⩾1∧i⩽8 then
      for v:=16/u,—v do
        for j:=j0+v do
          if j⩾0∧j<56 then
            begin k:=i+j; w:=ЦВЕТ×ДОСКА[k];
              if w⩽0 then
                begin if w=—КОРОЛЬ then
                  go to ПРЕРЫВАНИЕ;
                  r:=r+1; СПИСОК[r]:=k
                end
            end;
```

Вычисление ходов конем начинается с вычисления квазиабсциссы i поля назначения (т. е. поля, на которое делается ход), равной $i0\pm 1$ или $i0\pm 2$ (сначала берется $i=i0+1$). Затем проверяется, не выйдет ли поле назначения за правый ($i\leqslant 8$) или левый ($i\geqslant 1$) край доски. Если да, то вычисляется квазиабсцисса другого поля назначения (например, $i=i0-1$). Если же нет, то вычисляется квазиордината j данного поля назначения, равная $j0\pm 16$ (если было $i=i0\pm 1$) или $j0\pm 8$ (если $i=i0\pm 2$), и проверяется, не выйдет ли поле назначения за верхний ($j\geqslant 56$) или нижний ($j\geqslant 0$) край доски. Если нет, то вычисляется код k поля назначения и относительный код w занимающей это поле фигуры. Если вычисленное поле занято своей фигурой (т. е. $w>0$), то ищется другое поле назначения (для других значений i и j). Если же нет (т. е. $w\leqslant 0$), то проверяется, не пронзойдет ли при этом ходе взятие короля против-

ника. Если нет, то вычисленный код поля заносится в СПИСОК. Если да, то происходит выход из процедуры к формальной метке ПРЕРЫВАНИЕ.

3.3. Вычисление ходов королем

Ходы королем, выполнимые в данной позиции, вычисляются с помощью нижеприведенной группы операторов, помещающейся в процедуре ПЕРЕПИСЬ ХОДОВ вслед за меткой КОРОЛЕМ.

```
for i:=i0,i—1,i+2 do
  if i≥1∧i≤8 then
    for j:=j0,j—8,j+16 do
      if j≥0∧j≤56 then
        begin k:=i+j; w:=ЦВЕТ×ДОСКА[k];
          if w≤0 then
            begin if w=—КОРОЛЬ then go to ПРЕРЫВАНИЕ;
              r:=r+1; СПИСОК[r]:=k
            end
        end;
```

Легко видеть, что вычисление ходов королем отличается от вычисления ходов конем только методом определения квазикоординат (i и j) поля назначения k. При этом ради сокращения записи операторов в качестве поля назначения временно принимается и самое исходное поле (с квазикоординатами $i0$ и $j0$). Поскольку исходное поле всегда занято «своей фигурой» (т. е. самим королем), то для него будет $w>0$ и, следовательно, код этого поля в СПИСОК занесен не будет.

3.4. Вычисление ходов ладьей и ортогональных ходов ферзем

Ходы ладьей и ортогональные ходы ферзем, выполнимые в данной позиции, вычисляются с помощью нижеприведенной группы операторов помещающейся в процедуре ПЕРЕПИСЬ ХОДОВ вслед за двойной меткой ЛАДЬЕЙ : ФЕРЗЕМ.

```
    for  u:=1,2,3,4  do
      begin h:=hh[u]; m:=mm[u];
        v:=  if u<3 then i0 else j0; u:=ПОЛЕ—v;
        for v:=v+h step h until m do
          begin k:=u+v; w:=ЦВЕТ×ДОСКА [k];
            if w>0 then go to uu;
            r:=r+1; СПИСОК[r]:=k;
            if  w<0  then
              go to if w=—КОРОЛЬ then ПРЕРЫВАНИЕ else uu
          end v;
uu:     end u;
```

В этой группе операторов главный оператор цикла, задавая переменной u четыре значения 1, 2, 3, 4, обеспечивает тем самым вычисление ходов по четырем ортогональным направлениям: вправо, влево, вверх и вниз от исходного поля соответственно. Оператор $h:=hh[u]$ из массива $hh=(1, —1, 8, —8)$ выбирает шаг h соответственно рассматриваемому направлению. Оператор $m:=mm[u]$ из массива $mm=(8, 1, 56, 0)$ выбирает предельное значение той из двух ортогональных координат поля назначения, которая изменяется в рассматриваемом направлении. (В частности, при $u=1$ будет выбран шаг $h=1$ и предельное значение абсциссы $m=8$, что соответствует направлению вправо от исходного поля.) Третий оператор присваивания переменной v присваивает значение $i0$ или $j0$ так, чтобы для горизонтальных направлений ($u=1, 2$) следующий далее оператор цикла вычислял квазиабсциссы полей назначения, а для вертикальных направлений ($u=3, 4$) — квазиординаты этих полей. В любом из этих случаев оператор $u:=$ПОЛЕ$—v$ присвоит переменной u значение той из двух квазикоординат исходного поля, которая при движении ладьи остается постоянной. [В частности первый раз (при $u=1$) переменная v получит значение $i0$, а u — значение $j0$; следующий далее заголовок цикла будет работать как **for** $v:=i0+1$ **step** 1 **until** 8 **do**, а оператор $k:=u+v$ будет вычислять $j0+v$, т. е. действительные поля назначения при движении ладьи вправо от исходного поля.]

Затем вычисляется относительный код w фигуры, возможно занимающей поле k, и проверяется, не занято ли это поле своей фигурой (если $w>0$). Если да, то делается переход на метку uu для вычисления ходов по другому ортогональному направлению. Если нет, то код поля k заносится в СПИСОК и делается проверка, не занято ли поле чужой (если $w<0$) фигурой. Если нет, то продолжается движение по данному ортогональному направлению (т. е. вычисляется следующее значение v). Если же да,

то проверяется, не является ли эта чужая фигура королем. Если да, то осуществляется выход из процедуры к формальной метке ПРЕРЫВАНИЕ. Если же нет, то делается переход к вычислению ходов в другом направлении.

Когда вычисление ходов по всем четырем ортогональным направлениям закончено, то делается проверка, была ли рассматриваемая фигура ФГ ладьей или ферзем. Если это была ладья, то вычисление ее ходов закончено и делается переход к метке ДАЛЕЕ. Если же это был ферзь, то начинается выполнение группы операторов с меткой СЛОНОМ для вычисления диагональных ходов ферзя. Заполнение массивов *hh* и *mm* производится один раз в начале работы программы с помощью процедуры ВЫЧИСЛЕНИЕ ТАБЛИЦ (ее описание дано ниже в разд. 12).

3.5. Вычисленне ходов слоном и диагональных ходов ферзем

Ходы слоном и диагональные ходы ферзем, выполнимые в данной позиции, вычисляются с помощью нижеприведенной группы операторов, помещающейся в процедуре ПЕРЕПИСЬ ХОДОВ вслед за меткой СЛОНОМ.

```
for u:=0,1,2,3 do
  begin h:=ТАБСЛОН[u]; m:=ТАБСЛОН[4×ПОЛЕ+u];
    for k:=ПОЛЕ+h step h until m do
      begin w:=ЦВЕТ×ДОСКА[k];
        if w>0 then go to uu2;
        r:=r+1; СПИСОК[r]:=k;
        if w<0 then
          go to if w=—КОРОЛЬ then ПРЕРЫВАНИЕ else uu2
      end k;
uu2:  end u;
```

Главный оператор цикла, задавая здесь переменной *u* четыре значения 0, 1, 2, 3, обеспечивает тем самым вычисление ходов по четырем диагональным направлениям: вправо — вверх, влево — вниз, влево — вверх, вправо — вниз от исходного поля соответственно. Оператор *h*:=ТАБСЛОН[*u*] из массива ТАБСЛОН выбирает шаг *h* (равный 9, —9, 7 или —7) соответственно рассматриваемому направлению. Следующий далее оператор выбирает из этого же массива значение *m* поля, являющегося предельным при движении по данному направлению. (Например, для слона на поле 51 при *u*=0 будут выбраны *h*=9 и *m*=60, а при *u*=1 будут выбраны *h*=—9 и *m*=33.)

Затем оператор цикла вычисляет коды *k* полей назначения по диагональному направлению, определяемому данным значением *u*. (Например, для поля 51 и *u*=1 заголовок этого оператора будет работать как **for** *k*:=51+9 **step** 9 **until** 60 **do**, а для *u*=2 — как **for** *k*:=51—9 **step** —9 **until** 33 **do**). Далее с вычисленными кодами полей назначения производятся точно такие же операции, какие производятся при вычислении ходов ладьей (см. разд. 3.4).

4. Процедура ВЫПОЛНЕНИЕ ХОДА

После того как в результате первого обращения к процедуре ПЕРЕПИСЬ ХОДОВ будет составлен список первых ходов белыми, выполнимых в исходной позиции (первая группа ходов в массиве СПИСОК), начинает работать процедура ВЫПОЛНЕНИЕ ХОДА (см. описание ее ниже). Эта процедура при каждом обращении к ней имитирует перестановку фигуры с поля СПИСОК[*p*] на поле СПИСОК[*p*+1], где *p* — это первый параметр процедуры. Цвет выполняющей ход фигуры задается вторым параметром ЦВЕТ, значение которого равно 1 для хода белыми и —1 для хода черными. Для того чтобы программа могла быстро отыскивать абсолютное значение кода выполняющей ход фигуры, ее номер и ее исходное поле, эти три величины перед выполнением очередной подгруппы ходов массива СПИСОК запоминаются не только в начале подгруппы, но и в переменных ФИГУРА[ПОЛУХОД], НОМЕР[ПОЛУХОД] и ОБРАТНО К[ПОЛУХОД] соответственно (см. ниже в процедуре ВЫПОЛНЕНИЕ ХОДА операторы с метками ПОДГОТОВКА ПРОБЫ ХОДОВ НОВОЙ ФИГУРОЙ и ПОДГОТОВКА ОТВЕТА).

В начале процедуры (для сокращения ее записи и времени ее выполнения) значения неоднократно употребляемых (в ее теле) переменных с индексами временно запоминаются в простых переменных. Так, переменная НА становится равной коду поля, *на* которое будет делаться данный ход. Переменная ОТ получает значение кода поля, на котором находилась фигура перед апробированием данного хода, т. е. поля, *от* которого будет делаться перестановка испытуемой фигуры на поле НА. Переменная ФГ станет кодом *фигуры*, ход которой будет апробироваться при данном обра-

щении к процедуре. В переменной ВЗ будет записан код фигуры, *взятой* ранее на поле ОТ. (Если такого взятия не было, то ВЗ имеет значение нуль.)

```
procedure ВЫПОЛНЕНИЕ ХОДА(p,ЦВЕТ,СМЕНА ФИГУРЫ);
      value ЦВЕТ; integer p,ЦВЕТ; label СМЕНА ФИГУРЫ;
begin go to ПОДГОТОВКА ХОДА;
ОТМЕНА ХОДА: ПОЛУХОД:=ПОЛУХОД—1; p:=p+1;
ПОДГОТОВКА ХОДА: ОТ:=СПИСОК[p]; НА:=СПИСОК[p+1];
      ФГ:=ФИГУРА[ПОЛУХОД]; ВЗ:=ВЗЯТАЯ[ПОЛУХОД];
      if НА>0 then
        begin
          if ФГ=ПЕШКА В ПРЕВРАЩЕНИИ then
            go to ПРЕВРАЩЕНИЕ;
          if ВЗ=ПЕШКА НА ПРОХОДЕ then
            begin ДОСКА[ОТ]:=0;
              ДОСКА[ОТ—8×ЦВЕТ]:=—ЦВЕТ
            end else
            ДОСКА [ОТ]:=—ЦВЕТ×ВЗ;
          ВЗЯТАЯ[ПОЛУХОД]:=abs(ДОСКА[НА]);
          НА ПРОХОДЕ[ПОЛУХОД]:=0;
          if ФГ=ПЕШКА then
            begin
              if НА>56∨НА<9 then
                begin
                  ФИГУРА[ПОЛУХОД]:=ФГ:=ПЕШКА В ПРЕВРАЩЕНИИ;
ПРЕВРАЩЕНИЕ: ДОСКА[НА]:=if ЦВЕТ×ДОСКА[НА]<0 then
                    ФЕРЗЬ×ЦВЕТ else ДОСКА[НА]—ЦВЕТ;
                  if ДОСКА[НА]≠КОНЬ×ЦВЕТ then p:=p—1 else
                    ФИГУРА[ПОЛУХОД]:=ФГ:=ПЕШКА;
                  go to ХОД СДЕЛАН
                end;
              k:=ОБРАТНО К[ПОЛУХОД];
              if abs(k—НА)=16 then
                begin НА ПРОХОДЕ[ПОЛУХОД]:=k+8×ЦВЕТ;
                  go to ОБЫЧНЫЙ ХОД
                end;
              if НА=НА ПРОХОДЕ[ПОЛУХОД—1] then
                begin ДОСКА[НА—8×ЦВЕТ]:=0;
                  ВЗЯТАЯ[ПОЛУХОД]:=ПЕШКА НА ПРОХОДЕ
                end
            end;
ОБЫЧНЫЙ ХОД: ДОСКА[НА]:=ФГ×ЦВЕТ;
ХОД СДЕЛАН: АДРЕСА[НОМЕР[ПОЛУХОД]]:=НА;
          ПОЛУХОД:=ПОЛУХОД+1;
          r:=РУБЕЖ ХОДОВ[ПОЛУХОД]:=РУБЕЖ ХОДОВ[ПОЛУХОД—1];
ПРОВЕРКА ЗАКОННОСТИ ХОДА И ЗАПИСЬ ОТВЕТНЫХ ХОДОВ:
          ПЕРЕПИСЬ ХОДОВ(—ЦВЕТ,РУБЕЖ ХОДОВ[ПОЛУХОД],
            ОТМЕНА ХОДА);
          if ЗАДАНА РОКИРОВКА then
            begin
              if ЦВЕТ×ЦВЕТ РОКИРОВКИ≤0∧ПОЛУХОД<пп then
                ЗАПИСЬ РОКИРОВКИ(—ЦВЕТ)
            end;
ПОДГОТОВКА ОТВЕТА: НОМЕР[ПОЛУХОД]:=СПИСОК[r];
          ФИГУРА[ПОЛУХОД]:=СПИСОК[r+1];
          ОБРАТНО К[ПОЛУХОД]:=СПИСОК[r+2]
        end  else
        begin
ВОЗВРАТ ХОДИВШЕЙ: ДОСКА[—НА]:=ФГ×ЦВЕТ;
          АДРЕСА[НОМЕР[ПОЛУХОД]]:=—НА;
          if ВЗ=ПЕШКА НА ПРОХОДЕ then
            begin ДОСКА[ОТ]:=0;
              ДОСКА[ОТ—8×ЦВЕТ]:=—ЦВЕТ;
            end else
            ДОСКА[ОТ]:=—ЦВЕТ×ВЗ;
          ВЗЯТАЯ[ПОЛУХОД]:=0;
```

```
ПОДГОТОВКА ПРОБЫ ХОДОВ НОВОЙ ФИГУРОЙ:
        p:=p+4; НОМЕР[ПОЛУХОД]:=СПИСОК[p—2];
        ФИГУРА[ПОЛУХОД]:=СПИСОК[p—1];
        ОБРАТНО К[ПОЛУХОД]:=СПИСОК[p];
        go to СМЕНА ФИГУРЫ
    end
end ВЫПОЛНЕНИЕ ХОДА;
```

Условие **if** НА>0 **then** проверяет, не закончилось ли апробирование ходов данной фигуры. Если нет, то проверяется, не является ли испытуемая фигура пешкой, находящейся в процессе превращения. Если да, то совершается переход к операторам с меткой ПРЕВРАЩЕНИЕ, выполнение которых будет описано ниже. Если же нет, то следующим оператором (начиная с **if** ВЗ=...) на доску возвращается фигура противника, возможно взятая с поля ОТ при апробировании предыдущего (в массиве СПИСОК) поля. При этом сначала проверяется, не была ли эта взятая фигура пешкой на проходе. Если да, то возврат ее на доску делается по-особому, т. е. не на поле ОТ (которое в этом случае просто очищается оператором ДОСКА[ОТ]:=0), а на поле ОТ+8×ЦВЕТ. Иллюстративные переменные ПЕШКА НА ПРОХОДЕ и ПЕШКА В ПРЕВРАЩЕНИИ в начале работы программы получают, а затем всегда сохраняют значение —1.

Далее в освободившейся (в результате возврата на доску взятой ранее фигуры противника) переменной ВЗЯТАЯ[ПОЛУХОД] запоминается абсолютное значение кода фигуры противника, находящейся, возможно, на поле НА, т. е. выполняется взятие этой фигуры. Затем переменная НА ПРОХОДЕ[ПОЛУХОД] очищается от кода поля, возможно записанного в нее при апробировании предыдущего по списку хода (если это был ход пешкой на два поля вперед).

Если в результате проверки условия **if** ФГ=ПЕШКА **then** окажется, что рассматриваемая фигура не является пешкой, то выполняется обычный ход, т. е. на поле НА заносится код ФГ со знаком переменной ЦВЕТ (иллюстративная переменная ПЕШКА всегда имеет значение 1). Если же рассматриваемая фигура — пешка, то для нее приходится еще выполнять ряд особых операций, зависящих от положения этой пешки на доске.

Так, если окажется, что пешка выполняет ход на самую верхнюю (при НА>56) или самую нижнюю (при НА<9) горизонталь, то начинают выполняться операции, связанные с превращением пешки в фигуру. При этом сначала переменным ФИГУРА[ПОЛУХОД] и ФГ присваивается значение иллюстративной переменной ПЕШКА В ПРЕВРАЩЕНИИ (т. е. —1). Это делается для того, чтобы при следующих пробах хода (в данной позиции) обходить ряд операторов, которые не должны тогда выполняться, и сразу попадать на метку ПРЕВРАЩЕНИЕ с помощью оператора

if ФГ=ПЕШКА В ПРЕВРАЩЕНИИ **then go to** ПРЕВРАЩЕНИЕ

Оператор с меткой ПРЕВРАЩЕНИЕ при первом своем выполнении записывает на поле НА ферзя, поскольку тогда ЦВЕТ×ДОСКА[НА]≤0, так как поле НА либо свободно, либо занято чужой фигурой. При последующих пробах хода (в данной позиции) на поле НА каждый раз будет записываться фигура, код которой на единицу меньше (по модулю) кода ранее записанной туда фигуры (т. е. за ферзем будет записываться ладья, затем слон и, наконец, конь), поскольку тогда ЦВЕТ×ДОСКА[НА]>0, так как поле НА будет уже занято своей фигурой (превращенной пешкой).

Каждый раз после такой записи проверяется, не закончились ли превращения, т. е. не записан ли уже на поле НА конь). Если еще не закончились, то значение параметра p уменьшается на единицу для того, чтобы следующий (в данной позиции) пробный ход снова выполнялся на поле НА. Если же превращения закончились, то переменным ФИГУРА[ПОЛУХОД] и ФГ снова возвращается значение переменной ПЕШКА (т. е. 1). В результате этого при последующих пробах хода (в данной позиции) снова начнут выполняться все операторы, предшествующие метке ПРЕВРАЩЕНИЕ. После каждого превращения совершается переход (ради экономии машинного времени) на метку ХОД СДЕЛАН, поскольку операторы, предшествующие этой метке, в таком случае выполнять не требуется.

Если же рассматриваемая пешка не совершает хода на последнюю горизонталь доски, то проверяется, не выполняет ли сейчас эта пешка ход на два поля вперед. Если да [т. е. если $abs(k-НА)=16$, где k — исходное поле], то код поля, через которое «перепрыгивает» сейчас пешка (равный $k+8\times ЦВЕТ$), запоминается в переменной НА ПРОХОДЕ[ПОЛУХОД] и совершается переход к выполнению обычного хода. Если же нет, то делается еще проверка, не совершает ли данная пешка взятие

пешки противника на проходе. Если да (т. е. если НА=НА ПРОХОДЕ[ПОЛУХОД—1]), то поле НА—8×ЦВЕТ, где находится сейчас пешка противника, очищается, а переменной ВЗЯТАЯ[ПОЛУХОД] присваивается значение переменной ПЕШКА НА ПРОХОДЕ (т. е. —1). Это значение будет использовано при следующей пробе хода в данной позиции для возврата на доску этой взятой на проходе пешки (см. в начале данной процедуры оператор, начинающийся с **if** ВЗ=...).

После выполнения хода (т. е. после записи на поле НА кода соответствующей фигуры) адрес ходившей фигуры, хранящийся в массиве АДРЕСА, корректируется оператором с меткой ХОД СДЕЛАН. Затем производится проверка законности сделанного хода и пополнение массива СПИСОК группой ответных ходов, которые может сделать противник в возникшей при этом позиции. Для этого сначала значение переменной ПОЛУХОД увеличивается на единицу. Затем для нового полухода начальное значение переменной РУБЕЖ ХОДОВ[ПОЛУХОД] (равное первому свободному индексу переменных массива СПИСОК) устанавливается равным последнему значению переменной РУБЕЖ ХОДОВ[ПОЛУХОД—1] и одновременно запоминается в переменной *r*. Затем выполняется обращение к процедуре ПЕРЕПИСЬ ХОДОВ, которая в процессе записи в массив СПИСОК группы ответных ходов проверяет, нет ли среди них хода, сопровождающегося взятием короля. Если такой ход есть, то произойдет выход из процедуры ПЕРЕПИСЬ ХОДОВ к фактической метке ОТМЕНА ХОДА (которая соответствует формальной метке ПРЕРЫВАНИЕ). Операторы, следующие за этой меткой, возвращают переменной ПОЛУХОД то значение, которое она имела при выполнении отменяемого хода, и увеличивают значение *p* на единицу для апробирования следующего в массиве СПИСОК хода.

Если апробированный ход оказался законным, то делается проверка, не была ли в начальной позиции задана рокировка (в этом случае переменная ЗАДАНА РОКИРОВКА будет иметь значение **true**, как это станет видно после рассмотрения процедуры ОПРЕДЕЛЕНИЕ РОКИРОВКИ). Решение задач с заданной рокировкой будет рассмотрено позднее в разд. 8. Здесь же рассматривается случай, когда в начальной позиции решаемой задачи рокировка была невозможна (в этом случае переменная ЗАДАНА РОКИРОВКА имеет значение **false**). В рассматриваемом случае начинают выполняться операторы с меткой ПОДГОТОВКА ОТВЕТА, которые из начала (т. е. с индекса *r*) только что записанной в СПИСОК группы ходов переписывают значения номера фигуры, ее кода и ее исходного поля в переменные НОМЕР[ПОЛУХОД], ФИГУРА[ПОЛУХОД] и ОБРАТНО К[ПОЛУХОД] соответственно. Эти значения будут использованы при апробировании ответного хода. На этом работа процедуры ВЫПОЛНЕНИЕ ХОДА в случае НА>0 заканчивается, и происходит выход из нее через последний символ **end** ее тела.

Если же в начале работы процедуры было НА<0 (т. е. апробирование ходов данной фигурой закончено), то фигура возвращается на исходное поле (—НА) и в массиве АДРЕСА восстанавливается ее исходный адрес. На доску возвращается (оператором, начинающимся с **if** ВЗ=...) фигура противника, возможно взятая при последней пробе хода, и выполняется ПОДГОТОВКА ПРОБЫ ХОДОВ НОВОЙ ФИГУРОЙ. Для этого индекс *p* рассматриваемых элементов массива СПИСОК, увеличивается на 4, так чтобы он стал соответствовать элементу, содержащему код исходного поля новой фигуры, и информация о новой фигуре переписывается в соответствующие элементы массивов ФИГУРА, НОМЕР и ОБРАТНО К. После этого осуществляется выход из процедуры через формальную метку СМЕНА ФИГУРЫ.

5. Процедура ВОЗВРАТ ХОДА

В процессе поиска решения задачи в ряде случаев как белым, так и черным приходится брать свой ход назад, т. е. восстанавливать позицию на доске такой, какой она была до ответного хода противника. Поскольку перед выполнением каждого полухода (т. е. перед каждым обращением к процедуре ВЫПОЛНЕНИЕ ХОДА) вся информация о фигуре, выполняющей этот полуход, запоминается в переменных с индексом ПОЛУХОД, то для известного значения этого индекса и индекса *p* (в массиве СПИСОК) выполненного противником хода восстановить позицию можно всегда за одним исключением, которое состоит в следующем. Если перед взятием хода назад ответным ходом противника была рокировка, т. е. восстанавливать на доске нужно не одну фигуру, а две (ладью и короля), то кроме вышеуказанных значений двух индексов необходимо еще знать дополнительную информацию о короле противника (поле короля ПКР и номер короля НКР). Восстановление позиции после рокировки противника будет изложено в разд. 8. Здесь рассматриваются лишь случаи, не связанные с рокировкой.

Выполнение приведенной ниже процедуры ВОЗВРАТ ХОДА начинается с того, что информация о восстанавливаемой (на доске) фигуре противника, т. е. ее код,

ход взятой ею фигуры, код поля, на которое она сделала ход (перед возвратом), и код ее исходного поля запоминаются в простых переменных ФГ, ВЗ, НАЧ и КОН соответственно. Это делается для сокращения записи процедуры и расходуемого ею машинного времени. При этом для выборки из массива СПИСОК кода поля НАЧ, на которое был ранее сделан противником ход, приходится учитывать случай, когда фигура противника была пешкой, совершившей на этом ходе превращение. В этом случае индекс p нужно брать на единицу большим, чем в остальных случаях, поскольку перед выходом из процедуры ВЫПОЛНЕНИЕ ХОДА этот индекс был на единицу уменьшен (см. описание процедуры ВЫПОЛНЕНИЕ ХОДА, второй оператор после метки ПРЕВРАЩЕНИЕ).

Далее код фигуры записывается на поле КОН массива ДОСКА, код этого поля фиксируется в массиве АДРЕСА, и переменная ВЗЯТАЯ[ПОЛУХОД] очищается (ее значение теперь уже хранится в переменной ВЗ). Затем взятая (возможно) противником фигура возвращается на доску таким же способом, как и в процедуре ВЫПОЛНЕНИЕ ХОДА, и делается проверка, не была ли в исходной позиции задана рокировка. Для рассматриваемого здесь случая (когда рокировка не задана) работа процедуры на этом заканчивается.

```
procedure ВОЗВРАТ ХОДА(ЦВЕТ,p,ПКР,НКР);
      value ЦВЕТ,p,ПКР,НКР; integer ЦВЕТ,p,ПКР,НКР;
begin integer НАЧ,КОН;
      ФГ:=ФИГУРА[ПОЛУХОД]; ВЗ:=ВЗЯТАЯ[ПОЛУХОД];
      НАЧ:=СПИСОК[p+
        (if ФГ=ПЕШКА В ПРЕВРАЩЕНИИ then 2 else 1)];
      КОН:=ОБРАТНО К [ПОЛУХОД];
      ДОСКА[КОН]:=ЦВЕТ×abs(ФГ);
      АДРЕСА[НОМЕР[ПОЛУХОД]]:=КОН;
      ВЗЯТАЯ[ПОЛУХОД]:=0;
      if ВЗ=ПЕШКА НА ПРОХОДЕ then
        begin ДОСКА[НАЧ]:=0;
          ДОСКА[НАЧ—8×ЦВЕТ]:=—ЦВЕТ
        end else
        ДОСКА[НАЧ]:=ЦВЕТ×ВЗ;
      if ЗАДАНА РОКИРОВКА then
        begin d:=КОН—НАЧ; k:=НАЧ+sign(d);
          if ДОСКА[k]≠0∧abs(d)<4∧k≠КОН then
ВОЗВРАТ КОРОЛЯ:begin ДОСКА[ПКР]:=КОРОЛЬ×ЦВЕТ;
                ДОСКА [k]:=0; АДРЕСА[НКР]:=ПКР
              end
        end
end ВОЗВРАТ ХОДА;
```

6. Процедуры подготовки решения

В данном разделе описываются процедуры, которые выполняются только один раз перед началом решения задачи и (в соответствии с их названием) предназначены для его подготовки. Выполняемые ими операторы оформлены в виде самостоятельных процедур только ради обеспечения краткости и наглядности описываемой ниже (в разд. 7) ведущей части программы. Это сделано потому, что ознакомление с работой этой части является решающим для понимания работы всей программы ЭРА-77, и, естественно, желательно процесс такого ознакомления по возможности облегчить.

Целям подготовки решения служат следующие процедуры: ПОДГОТОВКА ПАМЯТИ, ВВОД ПОЗИЦИИ, ОПРЕДЕЛЕНИЕ РОКИРОВКИ, ВЫЧИСЛЕНИЕ ТАБЛИЦ и ЗАПИСЬ ПЕРВЫХ ХОДОВ. Из них процедура ВВОД ПОЗИЦИИ была уже описана ранее (в разд. 1), а процедуры ОПРЕДЕЛЕНИЕ РОКИРОВКИ и ВЫЧИСЛЕНИЕ ТАБЛИЦ по методическим соображениям описываются позже (в разд. 8 и 12 соответственно). Таким образом, в данном разделе остается привести описания только двух процедур ПОДГОТОВКА ПАМЯТИ и ЗАПИСЬ ПЕРВЫХ ХОДОВ, без ознакомления с которыми понимание работы ведущей части программы было бы существенно затруднено.

Процедура ПОДГОТОВКА ПАМЯТИ присваивает ряду простых переменных и массивов начальные значения, необходимые для организации правильного использования их в дальнейшей работе программы. При этом значения иллюстративных переменных ПЕШКА, КОНЬ, СЛОН, ЛАДЬЯ, ФЕРЗЬ, КОРОЛЬ, БЕЛ, ЧЕРН, КОРОТ, ДЛИН, ПЕШКА НА ПРОХОДЕ и ПЕШКА В ПРЕВРАЩЕНИИ, присвоенные им в данной процедуре, сохраняются ими в процессе всего выполнения программы ЭРА-77. Описание процедуры приведено ниже.

```
procedure ПОДГОТОВКА ПАМЯТИ;
begin БЛК:=БЛД:=ЧЛК:=ЧЛД:=t1:=t2:=t3:=t4:=t5:=t6:=
      НА ПРОХОДЕ[0]:=0;
      for i:=1 step 1 until nn+1 do
        ВЗЯТАЯ[i]:=НОМЕР[i]:=ОБРАТНО К[i]:=0;
      for i:=1 step 1 until 64 do ДОСКА[i]:=0;
      ПЕШКА:=БЕЛ:=КОРОТ:=ХОД:=ПОЛУХОД:=
        РУБЕЖ ХОДОВ[0]:=РУБЕЖ ХОДОВ[1]:=1;
      КОНЬ:=2; СЛОН:=3; ЛАДЬЯ:=4; ФЕРЗЬ:=5; КОРОЛЬ:=6;
      ЧЕРН:=ДЛИН:=ПЕШКА НА ПРОХОДЕ:=
        ПЕШКА В ПРЕВРАЩЕНИИ:=—1;
      ПОИСК ХОДОВ 2:= false
end ПОДГОТОВКА ПАМЯТИ;
```

Процедура ЗАПИСЬ ПЕРВЫХ ХОДОВ вычисляет и засылает в начало массива СПИСОК все первые ходы белыми, возможные в данной позиции (см. разд. 3). Если в начальной позиции была допустима рокировка белыми, то список первых ходов дополняется еще ходами короля и соответствующей ладьи (или ладей), необходимыми для выполнения заданного вида рокировки (или рокировок) (операторы, выполняющие эту запись, приведены в разд. 8). Если при составлении этого списка ходов будет обнаружено, что белые могут первым ходом взять короля черных, то происходит выход из процедуры к фактической метке ОШИБКА, описанной в конце ведущей части программы (разд. 7), с выводом соответствующего сигнального текста и последующим остановом работы программы. Описание этой процедуры приведено ниже.

```
procedure ЗАПИСЬ ПЕРВЫХ ХОДОВ;
begin ПЕРЕПИСЬ ХОДОВ(БЕЛ,РУБЕЖ ХОДОВ[1],ОШИБКА);
      if ЗАДАНА РОКИРОВКА∧ЦВЕТ РОКИРОВКИ≠ЧЕРН then
        ЗАПИСЬ РОКИРОВКИ(БЕЛ)
end ЗАПИСЬ ПЕРВЫХ ХОДОВ;
```

7. Ведущая часть программы ЭРА-77

Ниже приведена ведущая часть программы ЭРА-77. Для получения полной программы достаточно второй комментарий в этой части заменить описаниями процедур, указанных в этом комментарии, и операторы с метками *n1*, *n2* и *n3* — соответствующими операторами из разд. 10. Подробные пояснения к операторам ведущей части приведены вслед за ее текстом.

```
begin comment Программа ЭРА-77;
      integer a,b,c,d,h,i,j,k,m,n,nn,p,q,r,s,t,t1,t2,
      t3,t4,t5,t6,u,v,w,ХОД,ПОЛУХОД,ХОД МАТА,ОТ,НА,
      ВЗ,ФГ,Б,Ч,БКР,ЧКР,БЛК,ЧЛК,БЛД,ЧЛД,ЧИСЛО БЕЛЫХ
      ЧИСЛО ЧЕРНЫХ,ЦВЕТ РОКИРОВКИ,ПОЛЕ,ВИД РОКИРОВКИ БЕЛЫХ,
      ВИД РОКИРОВКИ ЧЕРНЫХ,ПЕШКА,КОНЬ,СЛОН,ЛАДЬЯ,
      ФЕРЗЬ,КОРОЛЬ,ПЕШКА НА ПРОХОДЕ,ПЕШКА В ПРЕВРАЩЕНИИ,
      БЕЛ,ЧЕРН,КОРОТ,ДЛИН;
      Boolean ЗАДАНА РОКИРОВКА,С ХОДАМИ 2,ПОИСК ХОДОВ 2;
      integer array ИМЯ ПРОГРАММЫ[1:20],ТАБСЛОН[0:259],
      АДРЕСА [—16:16],ДОСКА[1:64],ИМЯ ФИГУРЫ[4:27],
      БУКВА [3:26],ИМЯ ЗАДАЧИ [1:30],hh,mm [1:4];
ВВОД ВСПОМОГАТЕЛЬНЫХ СИМВОЛОВ:
      for i:=4 step 4 until 24 do input(ИМЯ ФИГУРЫ[i]);
      for i:=3 step 3 until 24 do input(БУКВА[i]);
ВВОД ИНФОРМАЦИИ О ПРОГРАММЕ И ЗАДАЧЕ:
      input(ИМЯ ПРОГРАММЫ[1],С ХОДАМИ 2, ИМЯ ЗАДАЧИ[1],n);
      nn:=2×n;
      begin integer array ФИГУРА,ВЗЯТАЯ,НОМЕР,
        ОБРАТНО К[1:nn+1],НА ПРОХОДЕ,РУБЕЖ ХОДОВ[0:nn+1],
        СПИСОК[1:(nn+1)×130],БМ,ЧМ[1:n];
        Boolean array ВОЗМОЖЕН ПАТ[0:n];
        comment На место данного комментария нужно по-
        местить описания процедур, которые перечислены
        в табл. 42, приведенной в конце данного раздела;
НАЧАЛО: ПОДГОТОВКА ПАМЯТИ; ВВОД ПОЗИЦИИ;
        ОПРЕДЕЛЕНИЕ РОКИРОВКИ; ВЫЧИСЛЕНИЕ ТАБЛИЦ;
        ЗАПИСЬ ПЕРВЫХ ХОДОВ; НОМЕР[1]:=СПИСОК[1];
```

```
        ФИГУРА[1]:=СПИСОК[2]; ОБРАТНО К[1]:=СПИСОК[3];
ОЧЕРЕДНОЙ ХОД: Б:=РУБЕЖ ХОДОВ[ПОЛУХОД—1]+2;
ХОД БЕЛЫХ:if Б≤РУБЕЖ ХОДОВ[ПОЛУХОД] then
          begin ВОЗМОЖЕН ПАТ[ХОД—1]:=false;
            ВОЗМОЖЕН ПАТ[ХОД]:=true;
            if ХОД=1∨ХОД>ХОД МАТА then ХОД МАТА:=ХОД;
            ВЫПОЛНЕНИЕ ХОДА(Б,БЕЛ,ХОД БЕЛЫХ);
            БМ[ХОД]:=Б;
            if ХОД=n then
              begin ЕСТЬ ЛИ ШАХ(ПОИСК ЗАЩИТЫ);
ШАХА НЕТ:       go to БЕЛЫЕ МЕНЯЮТ ХОД
              end;
ПОИСК ЗАЩИТЫ: Ч:=РУБЕЖ ХОДОВ[ПОЛУХОД—1]+2;
ХОД ЧЕРНЫХ: if Ч≤РУБЕЖ ХОДОВ[ПОЛУХОД] then
              begin ВЫПОЛНЕНИЕ ХОДА(Ч,ЧЕРН,ХОД ЧЕРНЫХ);
n2:;
                ЧМ[ХОД]:=Ч; ХОД:=ХОД+1;
                go to if ХОД≤n then ОЧЕРЕДНОЙ ХОД else
                  БЕЛЫЕ БЕРУТ НАЗАД ХОД
              end ХОД ЧЕРНЫХ;
ХОДЫ ЧЕРНЫХ ИСЧЕРПАНЫ:
            if ВОЗМОЖЕН ПАТ[ХОД] then
              begin ЕСТЬ ЛИ ШАХ(ЭТО МАТ);
ЭТО ПАТ:        go to БЕЛЫЕ МЕНЯЮТ ХОД
              end;
ЧЕРНЫЕ БЕРУТ НАЗАД ХОД: ЭТО МАТ:
            ПОЛУХОД:=ПОЛУХОД—1; Б:=БМ[ХОД];
n3:         if ХОД=1 then go to РЕШЕНИЕ;
            ВОЗВРАТ ХОДА(БЕЛ,Б,5,БКР);
            ХОД:=ХОД—1; ПОЛУХОД:=ПОЛУХОД—1;
            Ч:=ЧМ[ХОД]+1;
            go to ХОД ЧЕРНЫХ;
РЕШЕНИЕ: ПЕЧАТЬ РЕШЕНИЯ;
n1:;
ОДНО РЕШЕНИЕ НАЙДЕНО: go to КОНЕЦ;
БЕЛЫЕ МЕНЯЮТ ХОД: Б:=Б+1; ПОЛУХОД:=2×ХОД—1;
            go to ХОД БЕЛЫХ
          end ХОД БЕЛЫХ;
        if ХОД=1 then
          begin
            output('//','Т','ХОДЫ БЕЛЫХ ИСЧЕРПАНЫ','/');
            go to КОНЕЦ
          end;
БЕЛЫЕ БЕРУТ НАЗАД ХОД: ХОД:=ХОД—1;
        ПОЛУХОД:=ПОЛУХОД—1; Ч:=ЧМ[ХОД];
        ВОЗВРАТ ХОДА(ЧЕРН,Ч,61,ЧКР);
        Б:=БМ[ХОД]+1; ПОЛУХОД:=ПОЛУХОД—1;
        go to ХОД БЕЛЫХ;
ОШИБКА: output('//','Т',
          'ПЕРВЫМ ХОДОМ БЕРЕТСЯ КОРОЛЬ ЧЕРНЫХ','/');
КОНЕЦ: end
end
```

Программа ЭРА-77 начинает свою работу с ввода исходной информации (необходимой для ее выполнения) операторами с метками ВВОД ВСПОМОГАТЕЛЬНЫХ СИМВОЛОВ и ВВОД ИНФОРМАЦИИ О ПРОГРАММЕ И ЗАДАЧЕ. Эта информация (см. разд. 1) должна быть подготовлена к вводу (в данном варианте программы должна быть набита на перфокартах) и размещена (в данном случае — в колоде перфокарт) в указанном на рис. 11 порядке. В частности, в результате этого ввода логическая переменная С ХОДАМИ 2 получает значение **true** или **false** в зависимости от того, какой из этих двух символов был принят пользователем в качестве «задания формы результата» (см. рис. 11 и пояснения к нему в разд. 1). Целая переменная *n* после ввода исходной информации получает значение количества ходов, заданных в решаемой задаче.

Затем после метки НАЧАЛО выполняются обращения к пяти процедурам подготовки решения (см. разд. 6), после которых в массиве СПИСОК оказывается записанной первая группа ходов (см., например, табл. 41). Информация о последней вве-

денной в память машины фигуре, ходы которой будут апробироваться программой первыми, сразу же переписывается в переменные НОМЕР[1], ФИГУРА[1] и ОБРАТНО К[1]. [Для отладочной задачи № 1 это будут числа 11, 1 и 39 соответственно (см. табл. 41).]

После метки ОЧЕРЕДНОЙ ХОД параметр цикла Б получает сначала значение 3 (поскольку РУБЕЖ ХОДОВ[0]=1; см. разд. 6, описание процедуры ПОДГОТОВКА ПАМЯТИ), что соответствует индексу (в массиве СПИСОК) исходного поля первой испытуемой фигуры [для отладочной задачи № 1 это поле 39 (см. табл. 41)].

За меткой ХОД БЕЛЫХ следует условие, проверяющее, не закончилось ли апробирование ходов данной группы массива СПИСОК (т. е. ходов, возможных в данной позиции). {Так, например, для отладочной задачи № 1 после записи первых ходов значение переменной РУБЕЖ ХОДОВ[ПОЛУХОД] будет равно 81 (количество ходов в табл. 41 плюс единица), и данное условие будет, очевидно, выполнено.} Если апробирование ходов данной группы закончилось, то проверяется (после строки end ХОД БЕЛЫХ), не была ли эта группа ходов первой группой массива СПИСОК. Если да (т. е. если при этом ХОД=1), то печатается текст ХОДЫ БЕЛЫХ ИСЧЕРПАНЫ, который означает, что рассмотрение всех возможных ходов, которые могли бы привести к решению задачи, закончено. Если же рассмотренная группа ходов не была первой в массиве СПИСОК, то начинают выполняться операторы, следующие за меткой БЕЛЫЕ БЕРУТ НАЗАД ХОД, которые обеспечивают возможность поиска решения апробированием предыдущего (в порядке выполнения) хода. Работа этих операторов описывается ниже.

Если же апробирование ходов данной группы массива СПИСОК не закончилось, то перед выполнением хода белыми переменная ВОЗМОЖЕН ПАТ[ХОД—1] получает значение false, поскольку сам переход к выполнению хода белыми означает, что в позиции, рассматривавшейся на предыдущем полуходе, у черных был хотя бы один законный ход, который они и выполнили (т. е. пат в этой позиции был невозможен). Однако в позиции, рассматриваемой на данном ходе, возникновение пата для черных еще не исключено. В соответствии с этим переменной ВОЗМОЖЕН ПАТ[ХОД] временно (до рассмотрения следующего хода) присваивается здесь значение true.

Следующий далее условный оператор определяет ход, на котором белые сделают мат фактически (для некорректно поставленной задачи значение этого хода может оказаться меньше заданного в задаче числа ходов). Значение этого хода (т. е. значение переменной ХОД МАТА) печатается в конце работы программы вслед за ключевым ходом (операторы, выполняющие это печатание, приведены в разд. 9) при выполнении обращения к процедуре ПЕЧАТЬ РЕШЕНИЯ, следующего за меткой РЕШЕНИЕ (см. выше в ведущей части программы ЭРА-77). Переменная ХОД всегда имеет значение рассматриваемого в данный момент хода, а переменная ПОЛУХОД — значение рассматриваемого полухода.

Затем обращением к процедуре ВЫПОЛНЕНИЕ ХОДА проверяется, закончилось ли апробирование ходов рассматриваемой фигуры белых (т. е. апробирование рассматриваемой подгруппы ходов массива СПИСОК). Если да (т. е. при выполнении этой процедуры оказалось, что НА=СПИСОК[Б+1]<0), то происходит выход из процедуры ВЫПОЛНЕНИЕ ХОДА к метке ХОД БЕЛЫХ для апробирования хода новой фигурой. Если же апробирование ходов рассматриваемой фигуры не закончено, то (для апробирования нового хода этой фигуры с поля ОБРАТНО К[ПОЛУХОД] на поле СПИСОК[Б+1]) процедуры ВЫПОЛНЕНИЕ ХОДА переставляет белую фигуру с поля СПИСОК[Б] на поле СПИСОК[Б+1] и значение индекса Б запоминается в переменной БМ[ХОД] на случай возврата к данной позиции после серии последующих ходов.

Затем проверяется, не является ли выполненный белыми ход последним в задаче. Если да (т. е. ХОД=n), то обращением к процедуре ЕСТЬ ЛИ ШАХ (описание этой процедуры приведено в разд. 11) проверяется, сопровождается ли этот последний ход шахом. Если шаха нет, то осуществляется переход к метке БЕЛЫЕ МЕНЯЮТ ХОД. Если же шах есть, то происходит выход из пропедуры к метке-параметру ПОИСК ЗАЩИТЫ. На эту же метку осуществляется переход и в случае, когда выполненный белыми ход не является последним (т. е. ХОД≠n).

Поиск защиты от выполненного белыми хода начинается с присвоения параметру цикла Ч значения, соответствующего индексу кода исходного поля первой фигуры в группе ответных (на этот выполненный белыми ход) ходов черными массива СПИСОК (в отладочной задаче № 1 переменная Ч первый раз получает значение 83).

За меткой ХОД ЧЕРНЫХ следует условие, проверяющее, не закончилось ли апробирование рассматриваемой группы ответных ходов черных. Если не закончилось (т. е. Ч≤РУБЕЖ ХОДОВ[ПОЛУХОД]), то обращением к процедуре ВЫПОЛНЕНИЕ ХОДА проверяется, закончилось ли апробирование ходов рассматриваемой фигуры черных. Если да, то происходит выход из процедуры к метке-параметру ХОД ЧЕР-

НЫХ и начинается апробирование следующего (в массиве СПИСОК) ответного хода. Если же апробирование ходов рассматриваемой фигуры черных еще не закончилось, то индекс Ч этого хода запоминается в переменной ЧМ[ХОД] на случай возврата к данной позиции. Затем значение переменной ХОД увеличивается на единицу и проверяется, не был ли выполненный ход последним в данной задаче. Если этот ход не был последним (т. е. ХОД$\leqslant n$), то выполняется возврат к метке ОЧЕРЕДНОЙ ХОД, после которой снова начинают выполняться вышеописанные действия, но уже для нового (увеличенного на единицу) значения переменной ХОД и для нового (увеличенного на два *) значения переменной ПОЛУХОД.

Если же выполненный чернымн ход был последним ходом в данной задаче (т. е. после увеличения на единицу значение переменной ХОД стало больше n), то это означает, что у черных есть защита от сделанного им на последнем ходе шаха, т. е. на этом последнем ходе черные не получили мат. Поэтому совершается переход к метке БЕЛЫЕ БЕРУТ НАЗАД ХОД, после которой значения переменных ХОД И ПОЛУХОД уменьшаются на единицу, и для возврата черной фигуры на прежнее место (где она была до выполнения последнего хода) переменной Ч возвращается то значение, которое она имела при выполнении предыдущего хода черных.

Далее обращением к процедуре ВОЗВРАТ ХОДА на доске восстанавливается позиция, которая была до последнего хода черных. Затем переменной Б присваивается значение индекса следующего (за выполненным) в массиве СПИСОК хода, значение переменной ПОЛУХОД еще раз уменьшается на единицу, и выполняется возврат к метке ХОД БЕЛЫХ для апробирования нового хода белых. Если при проверке условия, следующего за меткой ХОД ЧЕРНЫХ, окажется, что апробирование рассматриваемой группы ходов черными законнилось (т. е. Ч>РУБЕЖ ХОДОВ[ПОЛУХОД]), то это означает, что черные не нашли удовлетворительной защиты от предыдущего хода белых. В этом случае после метки ХОДЫ ЧЕРНЫХ ИСЧЕРПАНЫ делается проверка, возможен ли на данном ходе пат черным. Если да (т. е. ВОЗМОЖЕН ПАТ[ХОД]$\equiv$**true** и, следовательно, в данной позиции у черных вообще нет ни одного законного хода), то проверяется, ЕСТЬ ЛИ ШАХ черным. Если при этом окажется, что черный король находится под шахом, то это означает, что на данном ходе черные получили мат и соответственно происходит выход к метке ЭТО МАТ.

От мата черные могут попытаться избавиться, взяв ход назад. Поэтому для восстановления позиции (для возврата белой фигуры на прежнее место) значение переменной ПОЛУХОД уменьшается на единицу, а переменной Б возвращается значение индекса (в массиве СПИСОК) последнего апробированного хода белых. Эти же операции выполняются и тогда, когда после метки ХОДЫ ЧЕРНЫХ ИСЧЕРПАНЫ оказывается, что ВОЗМОЖЕН ПАТ[ХОД]$\equiv$**false**, и совершается переход к метке ЧЕРНЫЕ БЕРУТ НАЗАД ХОД.

Затем делается проверка, не был ли апробированный ход первым ходом в данной задаче. Если ХОД=1 (т. е. черные уже не могут взять ход назад), то выполняется переход к метке РЕШЕНИЕ, где процедурой ПЕЧАТЬ РЕШЕНИЯ (ее описание приведено в разд. 9) печатается ключевой ход. Если же апробированный ход не был первым ходом в задаче, то у черных имеется еще возможность взять ход назад, что они и делают обращением к процедуре ВОЗВРАТ ХОДА, восстанавливающей позицию, которая была на доске до последнего хода белых. Затем значения переменных ХОД и ПОЛУХОД уменьшаются на единицу, переменной Ч присваивается значение индекса (в массиве СПИСОК) апробированного ранее хода, и делается возврат к метке ХОД ЧЕРНЫХ для апробирования другого хода черными.

После печатания ключевого хода тот вариант программы ЭРА-77, который приведен выше, свою работу заканчивает. Однако если из вышеприведенной программы удалить оператор

ОДНО РЕШЕНИЕ НАЙДЕНО: **go to** КОНЕЦ;

то после печатания ключевого хода программа будет продолжать свою работу до полного исчерпания всех ходов белых. При этом если задача имеет несколько решений, то программа напечатает все соответствующие ключевые ходы, после чего выдаст на печать текст ХОДЫ БЕЛЫХ ИСЧЕРПАНЫ. В таком варианте программа удобна для проверки шахматных задач на однозначность решения.

В табл. 42 перечислены (по алфавиту) процедуры, описания которых нужно поместить на место второго комментария ведущей части программы ЭРА-77, а также разделы, где приведены их описания.

---

* Значение переменной ПОЛУХОД увеличивается на единицу внутрн тела процедуры ВЫПОЛНЕНИЕ ХОДА (см. разд. 4). После двух обращений к этой процедуре оно возрастет на два. (*Прим. авт.*)

Таблица 42

Номера п/п	Идентификаторы процедур	Номера разделов
1	ВВОД ПОЗИЦИИ	1
2	ВОЗВРАТ ХОДА	5
3	ВЫПОЛНЕНИЕ ХОДА	4
4	ВЫЧИСЛЕНИЕ ТАБЛИЦ	12
5	ЕСТЬ ЛИ ШАХ	11
6	ЗАПИСЬ ПЕРВЫХ ХОДОВ	6
7	ЗАПИСЬ РОКИРОВКИ	8
8	ОПРЕДЕЛЕНИЕ РОКИРОВКИ	8
9	ПЕРЕПИСЬ ХОДОВ	3
10	ПЕЧАТЬ ПОЗИЦИИ	2
11	ПЕЧАТЬ ПОЛЯ	9
12	ПЕЧАТЬ РЕШЕНИЯ	9
13	ПЕЧАТЬ ХОДА	9
14	ПОДГОТОВКА ПАМЯТИ	6
15	ПОДГОТОВКА ПОИСКА ХОДОВ 2	10

8. Решение задач с рокировкой

Алгоритм 171б (в отличие от алгоритма 50CJ [48]) построен так, чтобы решение задач, допускающих рокировки, ничем (в эксплуатации программы ЭРА-77) не отличалось от решения обычных задач (не допускающих рокировки), не требовало от пользователя программой выполнения каких-либо дополнительных операций и не налагало бы на применение программы каких-либо новых ограничений *. Для достижения этих целей (а также для повышения быстродействия и сокращения записи) операторы рокировки алгоритма 50CJ подверглись радикальной переработке и изменениям.

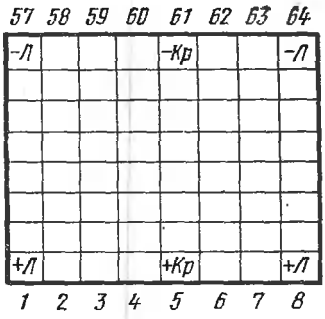

Рис. 12. Положение фигур, участвующих в рокировке.

Так, для обеспечения произвольности порядка ввода кодов рокируемых фигур в память машины уже в процедуре ВВОД ПОЗИЦИИ (см. разд. 1) предусмотрены специальные операторы. В частности, в теле первого оператора цикла этой процедуры предусмотрен условный оператор (начинающийся с if ФГ=КОРОЛЬ...), обеспечивающий запоминание номеров рокируемых белых фигур, а именно: номер белого короля запоминается в переменной БКР (т. е. номер Белого КоРоля), номер ладьи, если она расположена на поле $h1$, запоминается в БЛК (т. е. номер Белой Ладьи Короткой рокировки), а номер ладьи, расположенной на $a1$, запоминается в БЛД (т. е. номер Белой Ладьи Длинной рокировки). Аналогично в теле второго оператора цикла процедуры ВВОД ПОЗИЦИИ запоминаются номера рокируемых черных фигур (ЧКР — номер Черного КоРоля, ЧЛК — номер Черной Ладьи Короткой рокировки и ЧЛД — номер Черной Ладьи Длинной рокировки). Значения этих шести переменных используются позже в процедуре ЗАПИСЬ РОКИРОВКИ (описанной ниже в данном разделе).

Главной задачей операторов рокировки является дополнение группы ходов (возможных в данной позиции) массива СПИСОК соответствующими последовательностями кодов, обеспечивающими выполнение рокировок при обычной работе процедуры ВЫПОЛНЕНИЕ ХОДА.

---

* При решении задач с рокировкой пользователь алгоритмом 50CJ [48] должен был каждый раз вносить в свою программу изменения (для задания значений четырем логическим переменным, таким как КОРОТКАЯ РОКИРОВКА БЕЛЫХ и др.), вводить в память машины коды рокируемых фигур строго в определенном порядке и пользоваться только транслятором, допускающим рекурсивные процедуры. (*Прим. авт.*)

Так, для выполнения короткой рокировки белыми соответствующая группа ходов массива СПИСОК дополняется следующей последовательностью кодов (состоящей из трех подгрупп ходов, соответствующих рис. 12):

НКР	6	5	—7	БЛК	4	8	6	—8	НКР	6	7	—5
	КР	e1	g1		Л	h1	f1	h1		Кр	g1	e1
	Сдвиг короля			Сдвиг ладьи				Возврат рокировки				

где на местах НКР и БЛК подразумеваются их числовые значения. Легко заметить, что подгруппа ходов, обеспечивающая сдвиг короля, отличается от обычной подгруппы ходов массива СПИСОК тем, что в ней сразу за исходным полем 5 следует признак конца подгруппы —7. В результате этого процедура ВЫПОЛНЕНИЕ ХОДА (см. разд. 4) сразу же начнет выполняться для НА<0, и белый король будет переставлен с поля *e1* на поле *g1* без увеличения значения переменной ПОЛУХОД на единицу. Затем начнут выполняться ходы следующей подгруппы массива СПИСОК, обеспечивающие сдвиг ладьи с поля *h1* на поле *f1*. Таким образом при одном и том же значении полухода будет совершено перемещение («сдвиг») двух фигур: короля и ладьи, т. е. будет выполнена рокировка.

Если после выполнения серии последующих ходов обнаружится, что выполненная рокировка белых не обеспечивает решения данной шахматной задачи, то (в результате выборки из массива СПИСОК следующего хода) ладья будет возвращена на поле 8 (т. е. на поле *h1*) и на том же полуходе король будет переставлен с поля 7 на поле 5 (т. е. с *g1* на *e1*). Этим и будет обеспечено восстановление на доске позиции, существовавшей до рокировки (т. е. будет обеспечен «возврат рокировки»).

Аналогично этому для выполнения длинной рокировки белыми массив СПИСОК дополняется следующей последовательностью кодов:

НКР	6	5	—3	БЛД	4	1	4	—1	НКР	6	3	—5
	Кр	e1	c1		Л	a1	d1	a1		Кр	c1	e1

Для короткой рокировки черных нужен список

НКР	6	61	—63	БЛД	4	64	62	—64	НКР	6	63	—61
	Кр	e8	g8		Л	h8	f8	h8		Кр	g8	e8

Для длинной рокировки черных составляется список

НКР	6	61	—59	БЛД	4	57	60	—57	НКР	6	59	—61
	Кр	e8	c8		Л	a8	d8	a8		Кр	c8	e8

Вышеприведенные четыре последовательности кодов обеспечивают возврат соответствующих рокировок во всех случаях, кроме одного, когда рокировка сопровождалась матом черному королю. В этом случае черные должны взять назад свой предыдущий ход, чтобы поискать другую защиту от предпоследнего хода белых. Но для этого (т. е. для восстановления позиции, бывшей на доске перед последним ходом белых) должна выполняться процедура ВОЗВРАТ ХОДА. Поскольку при этом возвращаемый ход белых является рокировкой, то процедура ВОЗВРАТ ХОДА, кроме обычных операций, обеспечивающих восстановление на доске фигуры, ходившей последней (в данном случае ладьи), должна еще вернуть на место и короля. Эту задачу выполняет в процедуре ВОЗВРАТ ХОДА оператор, начинающийся с условия **if** ЗАДАНА РОКИРОВКА **then** (см. разд. 5).

Логической переменной ЗАДАНА РОКИРОВКА в начале работы программы ЭРА-77 присваивается (с помощью процедуры ОПРЕДЕЛЕНИЕ РОКИРОВКИ, описание которой приведено ниже) значение **true**, если в исходной позиции данной шахматной задачи какая-либо из пар король — ладья находилась в позиции, допускающей рокировку, и значение **false** в противном случае. Таким образом, последний оператор тела процедуры ВОЗВРАТ ХОДА начинает работать только тогда, когда в исходной позиции данной задачи был допустим какой-либо из четырех возможных видов рокировки. При этом сначала проверяется, был ли возвращаемый ход рокировкой, и только в этом случае начинает выполняться оператор с меткой ВОЗВРАТ КОРОЛЯ (см. разд. 5).

Процедура ОПРЕДЕЛЕНИЕ РОКИРОВКИ кроме присваивания вышеуказанных значений переменной ЗАДАНА РОКИРОВКА задает значения следующим трем целым переменным, которые будут в дальнейшем именоваться «индикаторами рокировки».

Переменная ЦВЕТ РОКИРОВКИ получает значение 1, если в исходной позиции была допустима рокировка только для белых, значение —1, если была допустима рокировка только для черных, и значение 0, если были допустимы рокировки как для белых, так и для черных.

Переменная ВИД РОКИРОВКИ БЕЛЫХ получает значение 1, если в начальной позиции была допустима короткая рокировка белых, значение —1, если была допустима длинная рокировка белых, и значение 0, если для белых были допустимы как короткая, так и длинная рокировка. Аналогично переменная ВИД РОКИРОВКИ ЧЕРНЫХ получает значения 1, —1 или 0 в зависимости от вида рокировки черных *. Описание этой процедуры имеет следующий вид:

```
procedure ОПРЕДЕЛЕНИЕ РОКИРОВКИ;
begin Boolean КБ,ДБ,КЧ,ДЧ,РБ,РЧ;
      КБ:=ДОСКА[8]=ЛАДЬЯ; ДБ:=ДОСКА[1]=ЛАДЬЯ;
      КЧ:=ДОСКА[64]=—ЛАДЬЯ; ДЧ:=ДОСКА[57]=—ЛАДЬЯ;
      РБ:=ДОСКА[5]=КОРОЛЬ∧(КБ∨ДБ);
      РЧ:=ДОСКА[61]=—КОРОЛЬ∧(КЧ∨ДЧ);
      ЗАДАНА РОКИРОВКА:=РБ∨РЧ;
      ЦВЕТ РОКИРОВКИ:=
        (if РБ then 1 else 0)—(if РЧ then 1 else 0);
      ВИД РОКИРОВКИ БЕЛЫХ:=
        (if РБ∧КБ then 1 else 0)—(if РБ∧ДБ then 1 else 0);
      ВИД РОКИРОВКИ ЧЕРНЫХ:=
        (if РЧ∧КЧ then 1 else 0)—(if РЧ∧ДЧ then 1 else 0)
end ОПРЕДЕЛЕНИЕ РОКИРОВКИ;
```

Индикаторы рокировки сохраняют полученные здесь значения до конца работы программы и используются для правильной и экономной организации записи в массив СПИСОК вышеуказанных четырех последовательностей кодов, обеспечивающих выполнение рокировок.

Дополнение массива СПИСОК вышеуказанными четырьмя последовательностями кодов осуществляется с помощью процедуры ЗАПИСЬ РОКИРОВКИ, описание которой приведено ниже **.

```
procedure ЗАПИСЬ РОКИРОВКИ(ЦВЕТ);
      value ЦВЕТ; integer ЦВЕТ;
begin integer r,НКР,НЛК,НЛД,НЛ,ВИД РОКИРОВКИ;
      if ЦВЕТ=БЕЛ then
        begin НКР:=БКР; НЛК:=БЛК; НЛД:=БЛД; s:=0;
          ВИД РОКИРОВКИ:=ВИД РОКИРОВКИ БЕЛЫХ
        end else
        begin НКР:=ЧКР; НЛК:=ЧЛК; НЛД:=ЧЛД; s:=56;
          ВИД РОКИРОВКИ:=ВИД РОКИРОВКИ ЧЕРНЫХ
        end;
      r:=РУБЕЖ ХОДОВ[ПОЛУХОД]; q:=5+s;
      if ДОСКА[q]≠ЦВЕТ×КОРОЛЬ then go to ВЫХОД;
      if ВИД РОКИРОВКИ=ДЛИН then go to ДЛИННАЯ;
КОРОТКАЯ: НЛ:=НЛК; a:=6+s; b:=7+s; c:=8+s;
      go to if ДОСКА[a]≠0∨ДОСКА[b]≠0 then ТЕСТ else БЫЛ ЛИ ХОД;
ДЛИННАЯ: НЛ:=НЛД; a:=4+s; b:=3+s; c:=1+s;
      if ДОСКА[a]≠0∨ДОСКА[b]≠0∨ДОСКА[2+s]≠0 then go to ВЫХОД;
БЫЛ ЛИ ХОД:
      for j:=ПОЛУХОД—2 step —2 until 1 do
        if НОМЕР[i]=НКР∨НОМЕР[j]=НЛ then go to ТЕСТ;
```

---

* Индикаторы рокировки получают нулевые значения и тогда, когда никакая из рокировок не задана, но это не приводит к неоднозначности, поскольку индикаторы рокировки используются в программе лишь в том случае, когда ЗАДАНА РОКИРОВКА=true. Для решения задач, в которых недопустимость какого-либо из видов рокировки обнаруживается ретроанализом, операторы присваивания значений индикаторам рокировки нужно заменить операторами ввода этих значений. (*Прим. авт.*)

** В алгоритме 50CJ [48] операторы рокировки являлись частью процедуры ОБЗОР ХОДОВ (так называлась в алгоритме 50CJ процедура, аналогичная процедуре ПЕРЕПИСЬ ХОДОВ). В алгоритме 171б операторы рокировки оформлены в виде самостоятельной процедуры ради ликвидации рекурсивности процедуры ОБЗОР ХОДОВ. (*Прим. авт.*)

```
БИТЫ ЛИ ПОЛЯ:
      ДОСКА[a]:=ДОСКА[b]:=ЦВЕТ×КОРОЛЬ; t:=r;
      ПЕРЕПИСЬ ХОДОВ (—ЦВЕТ,t,ОЧИСТКА ДОСКИ);
ХОДЫ РОКИРОВКИ: СПИСОК[r]:=СПИСОК[r+9]:=НКР;
      СПИСОК[r+1]:=СПИСОК[r+10]:=КОРОЛЬ;
      СПИСОК[r+2]:=q; СПИСОК[r+3]:=—b;
      СПИСОК[r+4]:=НЛ; СПИСОК[r+5]:=ЛАДЬЯ;
      СПИСОК[r+6]:=c; СПИСОК[r+7]:=a;
      СПИСОК[r+8]:=—c; СПИСОК[r+11]:=b;
      СПИСОК[r+12]:=—q; r:=r+13;
ОЧИСТКА ДОСКИ: ДОСКА[a]:=ДОСКА[b]:=0;
ТЕСТ: if ВИД РОКИРОВКИ=0∧НЛ=НЛК then go to ДЛИННАЯ;
ВЫХОД: РУБЕЖ ХОДОВ[ПОЛУХОД]:=r
end ЗАПИСЬ РОКИРОВКИ;
```

Первый оператор этой процедуры переменным НКР (т. е. Номер КоРоля), НЛК т. е. Номер Ладьи Короткой рокировки) и НБД (т. е. Номер Ладьи Длинной рокировки) присваивает значения номеров короля и ладей соответствующего цвета. В соответствии с цветом рассматриваемой рокировки здесь же присваивается значение переменным ВИД РОКИРОВКИ и *s*. Следующий условный оператор проверяет, находится ли король на месте (т. е. на поле $q=5+0$ для белых или на поле $q=5+56=61$ для черных) в момент выполнения процедуры.

Условия, следующие за метками КОРОТКАЯ и ДЛИННАЯ, проверяют, свободны ли поля, расположенные между королем и соответствующей ладьей. Оператором с меткой БЫЛ ЛИ ХОД делается проверка, не был ли сделан ход королем (если НОМЕР[*j*]=НКР) или соответствующей ладьей (если НОМЕР[*j*]=НЛ) на одном из предыдущих ходов *j*. Затем для проверки того, что ни одно из трех полей (на котором король помещается, на которое и через которое он пойдет) не находится под ударом неприятельских фигур, оператор с меткой БИТЫ ЛИ ПОЛЯ временно помещает на доску еще двух королей, так чтобы королями оказались заняты эти три поля. Далее обращением к процедуре ПЕРЕПИСЬ ХОДОВ вычисляются все возможные ходы неприятельских фигур, и если среди этих ходов окажется такой, который сопровождается взятием хотя бы одного из вышеуказанных трех королей, то проверяемый вид рокировки невозможен и произойдет переход к метке ОЧИСТКА ДОСКИ.

После этого два лишних короля снимаются с доскн, и оператором с меткой ТЕСТ проверяется, не были ли в исходной позиции заданы оба вида рокировки (т. е. короткая и длинная). Если да и перед последней проверкой выполнялись операции для короткой рокировки (т. е. если НЛ=НЛК), то совершается переход к метке ДЛИННАЯ, после которой снова выполняются вышеуказанные проверки, соответствующие длинной рокировке. Если же при обращении к процедуре ПЕРЕПИСЬ ХОДОВ окажется, что вышеупомянутые три поля не находятся под боем неприятельских фигур, то начинают выполняться операторы с меткой ХОДЫ РОКИРОВКИ, которые запишут в массив СПИСОК (начиная с индекса *r*) одну из четырех вышеописанных последовательностей кодов, обеспечивающую выполнение соответствующей рокировки.

9. Печатание результатов решения

Как уже говорилось выше, результаты решений шахматных задач программа ЭРА-77 печатает в наглядной, привычной для шахматистов форме. Первичной процедурой, обеспечивающей наглядность печати результатов, является нижеприведенная процедура ПЕЧАТЬ ПОЛЯ:

```
procedure ПЕЧАТЬ ПОЛЯ(ПОЛЕ);
      value ПОЛЕ; integer ПОЛЕ;
      begin j:=(ПОЛЕ—1)÷8;
      output('T',БУКВА[3×(ПОЛЕ—j×8)],'ZD',j+1)
end;
```

Эта процедура по значению параметра ПОЛЕ вычисляет и печатает буквенное обозначение вертикали этого поля и номер его по горизонтали. Например, если ПОЛЕ=39, то эта процедура напечатает G5, так как при этом $j=(39-1)\div 8=4$ и $j+1=5$, а $3\times(ПОЛЕ-j\times 8)=3\times(39-32)=21$ соответствует индексу открывающей кавычки в строке 'G' массива БУКВА (см. содержание этого массива в разд. 1).

Процедура ПЕЧАТЬ ПОЛЯ используется прежде всего в нижеследующей процедуре ПЕЧАТЬ ХОДА:

```
procedure ПЕЧАТЬ ХОДА(ЦВЕТ,ВЫП,ПЛХ);
    value ЦВЕТ,ВЫП,ПЛХ; integer ЦВЕТ,ПЛХ; Boolean ВЫП;
begin integer НАЧ,КОН; Boolean ОБХОД;
    ФГ:=ФИГУРА[ПЛХ]; НАЧ:=ОБРАТНО К[ПЛХ];
    КОН:=if ВЫП then АДРЕСА[НОМЕР[ПЛХ]] else abs(НА);
    d:=КОН—НАЧ;   ВЗ:=ВЗЯТАЯ[ПЛХ];
    ОБХОД:=ДОСКА[КОН—sign(d)]=КОРОЛЬ∧abs(d)<4;
    if ЦВЕТ=БЕЛ then output('Т','+') else
      if ЦВЕТ=0 then output('В') else output('Т','—');
ПЕЧАТЬ РОКИРОВКИ:
    if ФГ=КОРОЛЬ∧abs(d)=2 then go to ВЫХ;
    if ОБХОД∧d<0 then output('Т','————0—0—') else
      if ОБХОД∧d>0 then output('Т','  —0—0—0—') else
ПЕЧАТЬ ОБЫЧНОГО  ХОДА:
        begin output('Т',ИМЯ ФИГУРЫ[4×abs(ФГ)],'В');
          ПЕЧАТЬ ПОЛЯ(НАЧ); w:=abs(ДОСКА[КОН]);
          if ¬|ВЫП∧(w=0∨ФГ=ПЕШКА В ПРЕВРАЩЕНИИ∧ВЗ=0))∨
            (ВЫП∧ВЗ=0) then output('Т', '—') else output('Т',':3');
          ПЕЧАТЬ  ПОЛЯ(КОН);
ПЕЧАТЬ ПРЕВРАЩЕНИЯ:
          if ФГ=ПЕШКА∧(КОН<9∨КОН>56) then
            begin
              if ВЫП∧w=КОНЬ then output('Т','К—') else output('Т','Ф—')
            end  else
            if ФГ≠ПЕШКА В ПРЕВРАЩЕНИИ then output ('ВВ') else
              output ('Т', ИМЯ ФИГУРЫ[4×(if ВЫП then w else w—1)])
        end;
    output  ('В');
ВЫХ: end ПЕЧАТЬ ХОДА;
```

Процедура ПЕЧАТЬ ХОДА печатает ход, который был только что выполнен (если задано ВЫП≡**true**), или ход, который сейчас будет выполнен (если задано ВЫП≡**false**) при данном значении полухода ПЛХ. При этом вначале печатается знак + или —, если задано значение параметра ЦВЕТ, равное 1 или —1 соответственно, или не печатается ничего («печатается пробел»), если задано ЦВЕТ=0.

Печатание хода при ВЫП≡**false** используется в стандартных отладочных тестах (описанных в разд. 13) для прослеживания промежуточных попыток хода. Результаты решения печатаются всегда при ВЫП≡**true**. При печатании обычного хода всегда печатается буквенное обозначение фигуры (ФГ), исходное (НАЧ) и конечное (КОН) ее поля. Например, ход белой пешкой с поля *e2* на *e4* при ЦВЕТ=1 будет напечатан как +П E2—E4. Если ход сопровождается взятием фигуры противника, то тире между исходным и конечным полями заменяется двоеточием. Если пешка превращается в фигуру, то после конечного поля печатается буквенное обозначение этой фигуры (например, П D7—D8К). Однако, если ход сопровождается шахом или матом, то соответствующие символы + или × данной процедурой не печатаются.

Для правильного печатания хода, являющегося рокировкой (0—0 или 0—0—0), в начале процедуры вычисляется логическая переменная ОБХОД, которая получает значение **true** только тогда, когда ладья при своем ходе «перепрыгивает» через короля («обходит» короля), т. е. этот ход является рокировкой. Короткая рокировка отличается от длинной по знаку переменной d=КОН—НАЧ.

Оператор **if** ФГ=КОРОЛЬ∧abs(d)=2 **then go to** ВЫХ нужен здесь для того, чтобы при выводе на печать подряд всех промежуточных проб ходов (при ВЫП≡**false**) в случае рокировки печатался бы только символ 0—0 или 0—0—0, но не печатался бы ход королем на два поля по горизонтали.

Результат решения печатается ведущей программой (см. разд. 7) путем обращения к специальной процедуре ПЕЧАТЬ РЕШЕНИЯ, описание которой приведено ниже.

```
procedure ПЕЧАТЬ РЕШЕНИЯ;
begin output('Т',':/:/КЛЮЧЕВОЙ—ХОД—');
    ПЕЧАТЬ ХОДА(0, true, 1);
    output('Т',';—МАТ—НА','Z3D', ХОД МАТА, 'Т','—М—ХОДЕ','//')
end;
```

Например, в результате решения вышеприведенной отладочной задачи № 1 вслед за информацией, печатаемой процедурой ВВОД ПОЗИЦИИ (см. рис. 10), будет выведена на печать строка

КЛЮЧЕВОЙ ХОД Ф F2 : D4; МАТ НА 2-М ХОДЕ

10. Печатание вторых ходов

При решении многоходовых задач часто бывает нужно знать не только первый ключевой ход, но и все вторые ходы белых, обеспечивающие заданный мат при любом ответе черных на этот ключевой ход. По желанию пользователя программа ЭРА-77 может выдавать на печать вслед за ключевым ходом и такие вторые ходы. Для этого в ведущей части программы (разд. 7) достаточно сделать следующие изменения.

1. Пустой оператор *n*1:; нужно заменить оператором

```
n1:     if С ХОДАМИ 2∧ХОД МАТА>1 then
          begin ПОДГОТОВКА ПОИСКА ХОДОВ 2;
            go to ПОИСК ЗАЩИТЫ;
ДАЛЬШЕ: end;
```

Тогда если пользователь программой в качестве «задания формы результата» (см. рис. 11) введет в память машины символ «**true**;», то оператор ввода, следующий за меткой ВВОД ИНФОРМАЦИИ О ПРОГРАММЕ И ЗАДАЧЕ (см. раздел 7), присвоит переменной С ХОДАМИ 2 значение **true**. Следовательно, вышеприведенный оператор с меткой *n*1 вызовет обращение к процедуре ПОДГОТОВКА ПОИСКА ХОДОВ 2, описание которой приведено ниже:

```
procedure ПОДГОТОВКА ПОИСКА ХОДОВ 2;
begin ПОИСК ХОДОВ 2:=true; r:=РУБЕЖ ХОДОВ[1];
      НОМЕР[2]:=СПИСОК[r]; ФИГУРА[2]:=СПИСОК[r+1];
      ОБРАТНО К[2]:=СПИСОК[r+2]; ПОЛУХОД:=2;
      output('/','T','ВТОРЫЕ ХОДЫ БЕЛЫХ','/')
end;
```

Эта процедура вернет программу ЭРА-77 к тому состоянию, в котором она была непосредственно после выполнения ключевого хода с той лишь разницей, что теперь уже значение переменной ПОИСК ХОДОВ 2 будет **true**, тогда как до этого момента ее значение было **false** (см. в разд. 6 процедуру ПОДГОТОВКА ПАМЯТИ). Затем произойдет переход к метке ПОИСК ЗАЩИТЫ, после которой программа снова повторит все операции, которые она уже выполняла, начиная с ключевого хода до момента печатания решения, с той лишь разницей, что теперь в процессе выполнения такого вторичного поиска защиты она уже будет печатать (по ходу решения) все ответные первые ходы черных и вторые ходы белых (если, разумеется, в программу внесены и следующие два изменения).

2. Пустой оператор *n*2:; нужно заменить оператором

```
n2:   if ПОИСК ХОДОВ 2 then
        begin if ХОД=1 then ПЕЧАТЬ ХОДА(ЧЕРН,true,2) end;
```

Этот оператор обеспечивает печатание всех первых ходов черными до тех пор, пока ПОИСК ХОДОВ 2 будет иметь значение **true**.

3. Оператор

```
n3:   if ХОД=1 then go to РЕШЕНИЕ;
```

нужно заменить следующими двумя операторами:

```
n3:   if ХОД=1 then
        begin if ПОИСК ХОДОВ 2 then go to РЕШЕНИЕ;
          ПОИСК ХОДОВ 2:= false; go to ДАЛЬШЕ
        end;
      if ПОИСК ХОДОВ 2 then
        begin if ХОД=2 then
          begin ПЕЧАТЬ ХОДА(БЕЛ, true,3); output('3B') end
        end;
```

Второй из этих операторов будет обеспечивать печатание вторых ходов белых до тех пор, пока ПОИСК ХОДОВ 2≡**true**. Первый из этих операторов начнет выполняться тогда, когда черные закончат свой вторичный поиск защиты от ключевого хода белых и выяснится, что защиты у них нет. Поскольку при этом ПОИСК ХОДОВ 2 еще имеет значение **true**, то перехода к метке РЕШЕНИЕ не будет (ключевой ход уже был напечатан перед выполнением оператора с меткой *n*1), переменной ПОИСК ХОДОВ 2 будет присвоено значение **false** и произойдет переход к метке ДАЛЬШЕ. Если оператор с меткой ОДНО РЕШЕНИЕ НАЙДЕНО не был удален из программы, то на этом работа программы закончится. Если же такой оператор был удален, то после печатания вторых ходов программа будет продолжать перебор всех ходов с выдачей других решений, если таковые в задаче имеются. Работа программы в этом случае закончится печатанием строки

ХОДЫ БЕЛЫХ ИСЧЕРПАНЫ

Например, для задачи 3 ТАНДЕМ (условие задачи см. в разд. 14) при задании формы результата символом «true;» программа ЭРА-77 в качестве результата выдала следующую информацию:

КЛЮЧЕВОЙ ХОД Ф G7—F8; МАТ НА 3-М ХОДЕ

ВТОРЫЕ ХОДЫ БЕЛЫХ

—К G8—H6	+ Ф F8:E7	—П E7—E6	+ 0—0—0
—П E7:D6	+ 0—0	—КР E5:D6	+ Ф F8—B8

Для четырехходовой задачи А. Верле из еженедельника «64» за 1975 год № 29 (условие задачи см. в разд. 14) было напечатано

КЛЮЧЕВОЙ ХОД П E7—E8Л; МАТ НА 4-М ХОДЕ

ВТОРЫЕ ХОДЫ БЕЛЫХ

—П D2—D1Ф	+ Л E8—H8	—П D2—D1Л	+ Л E8—H8
—П D2—D1C	+ Л E8—F8	—П D2—D1К	+ КР F2—G3
—КР H1—H2	+ Л E8—H8		

Пользователь программой ЭРА-77 должен учитывать, что вышеописанный вывод на печать вторых ходов происходит за счет существенного замедления работы программы в среднем на 14% (для конкретных задач замедление может колебаться в пределах от 2 до 30%), поскольку при этом программа снова повторяет весь поиск защиты от ключевого хода. Поэтому для экономии машинного времени целесообразно обычные решения задач производить с заданием формы результата символом «false;». Если же встретится задача, ключевой ход которой требует еще и разъяснений путем выдачи вторых ходов, то решение этой задачи можно повторить с заданием формы результата символом «true», но порядок ввода в память машины кодов фигур целесообразно при этом изменить так, чтобы код фигуры, выполняющей ключевой ход, вводился последним (см. разд. 1). Тогда затраты времени на это повторное решение задачи будут минимальными.

Разумеется, вышеописанный метод программной реализации печатания вторых ходов не является единственно возможным. Это печатание можно запрограммировать и более экономно по затратам машинного времени, если обеспечить в программе регулярное запоминание вторых ходов белых, однако для такого запоминания потребуются дополнительная затрата машинной памяти и гораздо большие переделки в данной программе.

11. Процедура ЕСТЬ ЛИ ШАХ

Описание процедуры ЕСТЬ ЛИ ШАХ, используемой в ведущей части программы ЭРА-77 (см. разд. 7), отнесено в один из последних разделов алгоритма 171б, потому что детальный разбор этой процедуры для понимания работы программы ЭРА-77 не необходим. Роль этой процедуры, ее связь с остальными частями программы весьма просты и очевидны, поскольку результатом ее выполнения является только один из двух выходов из процедуры: либо к метке-параметру ШАХ этой процедуры (если в момент обращения к ней в позиции на доске есть шах черному королю), либо через последний **end** тела процедуры (если такого шаха нет). Детальный разбор этой процедуры может понадобиться, по-видимому, только тем пользователям программой, которые возьмутся за ее дальнейшую модификацию с целью разрешения других проблем шахматного программирования, с целью усовершенствования самой программы ЭРА-77 или для приспособления ее к другим машинам, к другим языкам программирования и к другим транслирующим системам. Описание процедуры имеет следующий вид:

```
procedure ЕСТЬ ЛИ ШАХ(ШАХ);
    label ШАХ;
begin integer di,dj,sg,ПКР,iКР,jКР,ПФ,iФ, jФ;
    switch sw:=ПЕШКОЙ,КОНЕМ,СЛОНОМ,ЛАДЬЕЙ,ФЕРЗЕМ,КОРОЛЕМ;
    ПКР:=АДРЕСА[ЧКР]; jКР:=(ПКР—1)÷8; iКР:=ПКР—8×jКР;
    for q:=1 step 1 until ЧИСЛО БЕЛЫХ do
      begin ПФ:=АДРЕСА[q]; jФ:=(ПФ—1)÷8;
        iФ:=ПФ—8×jФ; d:=ПКР—ПФ; m:=ДОСКА[ПФ];
        di:=iКР—iФ; dj:=jКР—jФ; sg:=sign(di×dj);
        di:=abs(di); dj:=abs(dj);
        go to if m>0 then sw[m] else ДРУГОЙ ФИГУРОЙ;
```

```
ПЕШКОЙ: go to if di×dj=1∧d>0 then ШАХ else
          ДРУГОЙ ФИГУРОЙ;
КОНЕМ: go to if di×dj=2 then ШАХ else ДРУГОЙ ФИГУРОЙ;
СЛОНОМ:ФЕРЗЕМ:
        if di=dj then
          begin h:=sign(d)×(8+sg); go to ЛИНИЯ end;
        if m=СЛОН then go to ДРУГОЙ ФИГУРОЙ;
ЛАДЬЕЙ: if sg=0 then
          begin h:=sign(d); if dj≠0 then h:=8×h;
ЛИНИЯ:      r:=ПКР—h;
            for k:=ПФ+h step h until r do
              if ДОСКА[k]≠0 then go to ДРУГОЙ ФИГУРОЙ;
            go to ШАХ
          end ЛАДЬЕЙ;
КОРОЛЕМ: ДРУГОЙ ФИГУРОЙ:
      end q
end ЕСТЬ ЛИ ШАХ;
```

Данная процедура с помощью оператора цикла с заголовком

for q:=1 **step** 1 **until** ЧИСЛО БЕЛЫХ **do**

осуществляет перебор всех белых фигур в порядке возрастания их номеров q (или, что то же самое, индексов полей в массиве АДРЕСА, занимаемых белыми фигурами) и по взаимному расположению на доске каждой из этих фигур и черного короля определяет наличие шаха. Взаимное расположение фигур характеризуется соотношением их «квазикоординат», причем в качестве «квазиординаты» в данной процедуре используется величина, в восемь раз меньшая, чем величина квазиординаты, определенной в разд. 1 (см. рис. 6). Поэтому термин «квазикоординаты» и «квазиординаты» будут в данном разделе употребляться в кавычках.

Перед главным циклом по q определяется поле черного короля ПКР и его «квазикоординаты» iКР и jКР. В начале цикла по q определяется поле ПФ, занимаемое очередной рассматриваемой фигурой, и его «квазикоординаты» iФ и jФ. Далее определяются код m рассматриваемой фигуры, разность d между кодами полей черного короля и белой фигуры и разности di и dj между их соответствующими «квазикоординатами».

Если рассматриваемая фигура не была взята и в данный момент находится на доске (т. е., если $m>0$), то совершается переход к определению шаха этой фигурой, если же этой фигуры уже нет на доске (т. е. при $m\leqslant 0$), то происходит переход к рассмотрению другой белой фигуры (для следующего значения q).

При определении шаха пешкой проверяется, находится ли пешка от короля на расстоянии одного шага по диагонали (при $di=1\wedge dj=1$) и расположен ли черный король на доске выше белой пешки (при $d>0$).

Определение шаха конем основано на том факте, что произведение двух натуральных чисел (в данном случае di и dj) может равняться двум тогда и только тогда, когда одно из них равно единице, а второе двум. В данном случае это условие равносильно тому, что белый конь находится на расстоянии одного хода от черного короля.

Определение шаха линейной фигурой (т. е. слоном, ладьей или ферзем) начинается с определения, находится ли черный король на линии хода (т. е. линии, по которой данная фигура может ходить) данной фигуры. Для слона это выражается проверкой наличия у фигуры общей с королем диагонали (когда $di=dj$), а для ладьи — общей с королем ортогонали (когда $sg=0$, а следовательно, либо $di=0$, либо $dj=0$).

Для ферзя выполняются обе эти проверки. Если черный король оказался на линии хода данной фигуры, то для нее сначала вычисляется шаг h, равный приращению кода поля, необходимому для продвижения на одно поле по этой линии хода. Для слона h может иметь значение 7, —7, 9, —9, а для ладьи 1, —1, 8, —8 в зависимости от направления движения от фигуры к королю. Это направление процедура определяет по значениям $sign(d)$ и sg. [Например, если король находится на поле 61, а слон на поле 47, то $sign(d)=sign(61-47)=1$ и $sg=sign(di\times dj)=sign(-2\times 2)=-1$. Следовательно, $h=1\times(8-1)=7$, т. е. к коду поля фигуры нужно добавлять (для продвижения от фигуры к королю на одно поле) число 7.]. После вычисления шага h начинают выполняться операторы с меткой ЛИНИЯ, которые проверяют, свободна ли линия, соединяющая черного короля с рассматриваемой белой фигурой, ибо только в этом случае объявлен шах.

Если очередная рассматриваемая белая фигура оказалась королем, то сразу совершается переход к рассмотрению другой фигуры, поскольку шах королем не объявляется.

12. Процедура ВЫЧИСЛЕНИЕ ТАБЛИЦ

В алгоритме 50CJ для вычисления ходов, возможных в данной позиции, использовались семь таблиц (таких, как ТАБЛИЦА КОНЯ, ТАБЛИЦА КОРОЛЯ и т. п.) общим объемом в 2056 машинных слов. Большие затраты машинной памяти на хранение этих таблиц были причиной того, что программисты, желающие воспользоваться алгоритмом, но располагающие только машинами с ограниченной памятью (такими, как М-220), вынуждены были для ликвидации таблиц предпринимать самостоятельную переработку алгоритма (см., например, «Подтверждение к алгоритму 50CJ» Г. А. Клугермана и М. А. Ожегова [49, приложение 1]) или отказываться от его применения.

При отсутствии у программистов достаточного опыта и достаточно глубокого знакомства с алгоритмом такая переработка может не только потребовать значительных затрат труда и времени, но и привести к существенному замедлению работы алгоритма. Поэтому для ликвидации таблиц автором алгоритма 171б была по существу заново разработана процедура ПЕРЕПИСЬ ХОДОВ, использующая при вычислении ходов вместо вышеуказанных семи таблиц одну только таблицу ТАБСЛОН[0 : 259] * и два массива *hh*[1 : 4] и *mm*[1 : 4]. Путем тщательного и продуманного подбора вычислительных средств эту переработку удалось осуществить без замедления алгоритма.

Содержание массива ТАБСЛОН не отличается от содержания массива ТАБЛИЦА СЛОНА алгоритма 50CJ. Начало массива ** имеет вид

9	—9	7	—7
64	1	1	1
56	2	9	2
48	3	17	3
.	.	.	.

Первая строчка этого массива содержит значения шагов по диагоналям при движении слона от данного поля вправо — вверх (шаг=9), влево — вниз (шаг=—9), влево — вверх (шаг=7) и вправо — вниз (шаг=—7). Вторая строчка содержит коды предельных полей при движении слона от поля 1 (т. е. *a*1) с соответствующим шагом. Третья строчка содержит предельные поля для исходного поля 2 (т. е. *b*1) и т. д. Например, при движении с поля 3 с шагом 9 (т. е. вправо — вверх) предельным является поле 48 (см. рис. 6), а при движении с шагом 7 (т. е. влево — вверх) — предельное поле 17. Использование этого массива для вычисления возможных в данной позиции ходов описано в разд. 3.5.

Массив ТАБСЛОН [так же, как и массивы *hh*=(1, —1, 8, —8) и *mm*==(8, 1, 56, 0)] заполняется в начале работы программы (при обращении к процедуре ВЫЧИСЛЕНИЕ ТАБЛИЦ) и сохраняет свое значение до конца ее выполнения. В алгоритме 50CJ заполнение таблицы слона производилось с помощью оператора ввода. В алгоритме 171б для сокращения объема вводимой информации ввод массива ТАБСЛОН был заменен его вычислением с помощью нижеприведенной процедуры ВЫЧИСЛЕНИЕ ТАБЛИЦ.

```
procedure ВЫЧИСЛЕНИЕ ТАБЛИЦ;
begin hh[1]:=1; hh[2]:=—1; hh[3]:=8; hh[4]:=—8;
      mm[1]:=8; mm[2]:=1; mm[3]:=56; mm[4]:=0;
      ТАБСЛОН[0]:=9; ТАБСЛОН[1]:=—9;
      ТАБСЛОН[2]:=7; ТАБСЛОН[3]:=—7;
      for k:=1 step 1 until 64 do
        begin m:=(k—1)÷8; j:=8×m; i:=k—j;
          v:=i—m; w:=i+m; m:=4×k;
          ТАБСЛОН[m]:= if v>0 then 72—8×v else 63+v;
          ТАБСЛОН[m+1]:=if v>0 then v else 9—8×v;
          ТАБСЛОН[m+2]:= if w<8 then 8×w—7 else 49+w;
          ТАБСЛОН[m+3]:= if w<8 then w else 8×w—56
        end k
end ВЫЧИСЛЕНИЕ ТАБЛИЦ;
```

---

* Ликвидировать и таблицу слона (как называлась в алгоритме 50CJ таблица ТАБСЛОН) без замедления алгоритма не представляется возможным. Ввиду небольшого объема этой таблицы решено было ее сохранить. (*Прим. авт.*)

** Полностью этот массив приведен в выпуске 2 [48, с. 103] данной серии. (*Прим. авт.*)

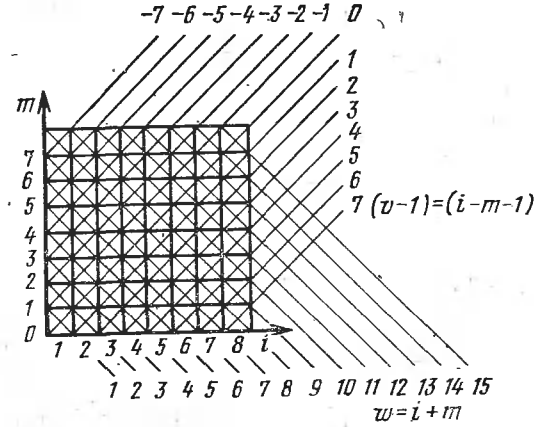

Рис. 13. Нумерация диагоналей доски и связь ее с «квазикоординатами».

Хотя процедура ВЫЧИСЛЕНИЕ ТАБЛИЦ **является совершенно новой, нигде ранее не описанной, подробного пояснения к ее операторам здесь не дается, поскольку, во-первых, правильность работы ее легко проверяется путем проверки составленного ею массива ТАБСЛОН и, во-вторых,** при переводе программы ЭРА-77 на другие языки программирования формулы вычисления предельных полей массива ТАБСЛОН остаются неизменными. Однако для программистов, которые возьмутся за дальнейшее развитие данного алгоритма, полезно будет заметить, что вычисляемые в данной процедуре значения переменной $v=i-m$ однозначно определяют диагонали доски, поднимающиеся вправо — вверх, а значения переменной $w=i+m$ определяют диагонали, поднимающиеся влево — вверх (см. рис. 13).

В данной процедуре (а также на рис. 13) i — значения квазиабсцисс вертикалей доски, а m — значения «квазиординат» (как они определены в разд. 11) горизонталей доски.

13. Отладочные тесты к программе ЭРА-77

Освоение любой новой программы обычно сопровождается ошибками, даже если программа готовится для трансляции в той же системе, в которой она первоначально отлаживалась (для программы ЭРА-77 это система БЭСМ-АЛГОЛ [62, 63, 90]). Вероятность появления ошибок существенно возрастает, если программа переводится на другой язык программирования или приспосабливается для другой транслирующей системы. И практически неизбежны ошибки тогда, когда производится модификация программы с целью ее дальнейшего развития и совершенствования.

Выявление и устранение ошибок в достаточно сложной программе (т. е. ее отладка) — процесс обычно довольно длительный и трудоемкий. Поэтому при публикации сложной программы, предназначенной для дальнейшего ее развития, представляется целесообразным, чтобы авторы делились с читателями опытом отладки, накопленным в процессе создания такой программы. Особенно полезным представляется набор отладочных тестов, часто употреблявшихся автором программы при ее разработке.

По вышесказанным соображениям ниже приводятся несколько тестов, наиболее часто использовавшихся при отладке программы ЭРА-77. Для того чтобы вставление этих тестов в программу сопровождалось минимальными в ней изменениями, в начале программы (см. разд. 7) специально описаны шесть вспомогательных переменных $t1$, $t2$, $t3$, $t4$, $t5$ и $t6$, а в процедуре ПОДГОТОВКА ПАМЯТИ (см. разд. 6) этим переменным присваиваются начальные нулевые значения (независимо от того, будут ли эти значения использованы при данном выполнении программы или нет). Поскольку такое присваивание делается только один раз, то затратами машинного времени на это можно пренебречь.

Тест 1. Ограничение количества выполнений некоторой группы операторов данным числом раз. При отладке часто бывает нужно ограничить число выполнений какой-либо группы операторов, которая без такого ограничения выполнялась бы циклически чрезмерно бо́льшое число раз. Для этого удобно пользоваться тестом 1, состоящим из следующих двух частей (обозначенных буквами А и Б).

А. Перед группой операторов, число выполнений которой при работе программы пользователь желает ограничить значением k, вставляется строка вида

```
if t1<k then begin t1:=t1+1;
```

где вместо k должно стоять его конкретное числовое значение. При работе с перфокартами удобно иметь заготовленные заранее карты, на каждой из которых набита подобная строка с различными значениями k (например, k=1, 2, 3, 5, 10, 30, 100, 500). В частности, если нужно, чтобы группа операторов выполнялась только три раза,

то перед ней вставляется строка

```
if t1<3 then begin t1:=t1+1;
```

Б. Вслед за вышеуказанной группой операторов вставляется строка

```
end;
```

если нужно, чтобы после выполнения данной группы операторов k раз программа продолжала работать, или строка

```
end else go to КОНЕЦ;
```

если после этого работа программы должна прерваться.

Тест 2. Печатание последовательности ходов. Этот тест обеспечивает печатание каждой пробы хода перед ее выполнением. При этом пробы ходов печатаются группами, в начале каждой из которых печатается текст, указывающий порядковый номер хода, на котором делаются эти пробы. Тест имеет вид следующего составного оператора:

```
begin if t2=0 then
        begin t2:=—1; output('/') end;
      if ХОД<t2 then output('/');
      if ХОД≠t2 then
        begin t2:=ХОД;
          output('/','Z—3D',ХОД,'T','—И—ХОД—')
        end;
      ПЕЧАТЬ ХОДА(ЦВЕТ, false, ПОЛУХОД)
end;
```

Этот тест вставляется в процедуру ВЫПОЛНЕНИЕ ХОДА непосредственно перед условием **if** НА>0 **then**. При работе с перфокартами удобно иметь этот тест всегда наготове набитым на две стандартные карточки.

Обычно печатание всех проб ходов, выполняемых от начала решения задачи до его конца, бывает нежелательно или невозможно из-за колоссального объема выводимой при этом информации. Поэтому этот тест обычно используется в комбинации с другими тестами, ограничивающими моменты начала и конца работы теста 2.

В качестве такого ограничивающего теста можно пользоваться, например, тестом 1 или составлять ограничения нестандартным образом, обеспечивая выполнение теста 2 только на интересующем программиста участке работы программы. Лишь в специально составленных отладочных задачах можно печатать все пробы ходов, выполняемые программой. Примером такой задачи может служить вторая задача А. Белла, приведенная в алгоритме 50CJ [48, с. 91, рис. 4]. Позиция этой задачи записывается следующим образом.

Белые (11 фигур): Kph1, Jh2, Jg1, Cf1, Ch3, e2, g2, g3, g4, e6, g6.
Черные (11фигур): Kph8, Jg8, Cf8, e3, e4, e5, f5, g5, f6, e7, g7.

Для этой задачи при пользовании тестом 2 было напечатано следующее:

```
1—Й ХОД  +П G4:F5  —КР H8—H7  —КР H8—H8  —П G5—G4
2—Й ХОД  +С H3:G4  —КР H8—H7  —КР H8—H8
1—Й ХОД  —П G5—G5
```

КЛЮЧЕВОЙ ХОД П G4:F5; МАТ НА 2-М ХОДЕ

Для задачи 1 типа Валодао [48, с. 116]
Белые: Kpe1, Ja1, Cd6, Ke2, Ke3, a7, d3, g2, h7.
Черные: Kph1, Cb1, c3, g5, h4.
при пользовании тестом (в комбинации с тестом 1) были напечатаны 60 следующих проб ходов.

```
1-Й ХОД  +П H7—H8Ф  —П G5—G4
2-Й ХОД  +Ф H8—G8  +Ф H8—F8  +Ф H8—E8  +Ф H8—D8
+Ф H8—C8  +Ф H8—B8  +Ф H8—A8  +Ф H8—H7  +Ф H8—H6
+Ф H8—H5  +Ф H8:H4  —П G4—G3  —П G4—G4  —П C3—C2
—П C3—C3  —КР H1—H2  —КР H1—G1  —КР H1:G2  —КР H1—H1
—С B1—C2  —С B1:D3  —С B1—A2  —С B1—B1
```

```
        1-Й ХОД —П G5—G5  —П  H4—H3
        2-Й ХОД +Ф H8—G8  +Ф  H8—F8  +Ф  H8—E8  +Ф  H8—D8
+Ф H8—C8  +Ф H8—B8  +Ф  H8—A8  +Ф  H8—H7  +Ф  H8—H6
+Ф H8—H5  +Ф H8—H4  +Ф  H8:H3  —П  G5—G4  —П  G5—G5
—П C3—C2  —П C3—C3  —КР H1—H2  —КР H1—G1  —КР H1:G2
—КР H1—H1  —С B1—C2  —С  B1:D3  —С  B1—A2  —С  B1—B1
        1-Й ХОД —П H4—H4  —П C3—C2
        2-Й ХОД +Ф H8—G8  +Ф H8—F8  +Ф H8—E8  +Ф H8—D8
+Ф H8—C8  +Ф H8—B8  +Ф H8—A8
```

Тест 3. Вывод полной информации из процедуры ВЫПОЛНЕНИЕ ХОДА. Иногда бывает необходимо перед выполнением каждого хода выводить из процедуры ВЫПОЛНЕНИЕ ХОДА значения всех используемых в ней переменных. Тест 3 обеспечивает такой вывод, причем информация печатается в виде таблицы с заголовками, содержащими сокращенные назнания печатаемых переменных. Тест имеет следующий вид:

```
begin if t3=0 then
        begin t3:=1;
          output('///','12B','T','— ХОД—','T','ПОЛУХ','3B','T','p—','T','— НОМ',
          'T','— ПРОХ','3B','T','B3','3B','T','Б—','T','—РУБ—','3B','T','Ч—')
          end;
      if ПОЛУХОД=1 then output('/');
      output('/');
      ПЕЧАТЬ ХОДА(ЦВЕТ, false, ПОЛУХОД);
      output('Z—3DB',ХОД,ПОЛУХОД,p,НОМЕР[ПОЛУХОД],
        НА ПРОХОДЕ[ПОЛУХОД],ВЗЯТАЯ[ПОЛУХОД],Б,
        РУБЕЖ ХОДОВ[ПОЛУХОД],Ч)
end;
```

		ХОД	ПОЛУХ	Р	НОМ	ПРОХ	ВЗ	Б	РУБ	Ч
+П	H7—H8Ф	1	1	3	9	0	0	3	76	0
—П	G5—G4	1	2	78	—5	0	0	2	105	78
+Ф	H8—G8	2	3	107	9	0	0	107	194	78
+Ф	H8—F8	2	3	108	9	0	0	108	194	78
+Ф	H8—E8	2	3	109	9	0	0	109	194	78
+Ф	H8—D8	2	3	110	9	0	0	110	194	78
+Ф	H8—C8	2	3	111	9	0	0	111	194	78
+Ф	H8—B8	2	3	112	9	0	0	112	194	78
+Ф	H8—A8	2	3	113	9	0	0	113	194	78
+Ф	H8—H7	2	3	114	9	0	0	114	194	78
+Ф	H8—H6	2	3	115	9	0	0	115	194	78
+Ф	H8—H5	2	3	116	9	0	0	116	194	78
+Ф	H8 : H4	2	3	117	9	0	0	117	194	78
—П	G4—G3	2	4	196	—5	0	0	117	218	196
—П	G4—G4	2	4	197	—5	0	0	117	218	197
—П	C3—C2	2	4	201	—3	0	0	117	218	201
—П	C3—C3	2	4	202	—3	0	0	117	218	202
—КР	H1—H2	2	4	206	—2	0	0	117	218	206
—КР	H1—G1	2	4	207	—2	0	0	117	218	207
—КР	H1 : G2	2	4	208	—2	0	0	117	218	208
—КР	H1—H1	2	4	209	—2	0	1	117	218	209
—С	B1—C2	2	4	213	—1	0	0	117	218	213
—С	B1 : D3	2	4	214	—1	0	0	117	218	214
—С	B1—A2	2	4	215	—1	0	1	117	218	215
—С	B1—B1	2	4	216	—1	0	0	117	218	216
—П	G5—G5	1	2	79	—5	0	0	117	105	79
—П	H4—H3	1	2	83	—4	0	0	117	105	83
+Ф	H8—G8	2	3	107	9	0	0	107	197	83
+Ф	H7—F8	2	3	108	9	0	0	108	197	83
+Ф	H8—E8	2	3	109	9	0	0	109	197	83

Так же, как и тест 2, этот тест вставляется в процедуру ВЫПОЛНЕНИЕ ХОДА перед условием **if** НА>0 **then** и обычно используется в комбинации с другими тестами, ограничивающими объем выводимой им информации.

В качестве примера выше приведен результат вывода информации с помощью теста 3 при решении задачи 1 типа Валодао (см. ее текст выше) для тридцати первых проб ходов.

Тест 4. Печатание первых ходов белыми. Очень часто бывает необходимо следить за ходом решения задачи путем вывода в процессе ее решения первых ходов белых. Такой вывод выполняется с помощью строки

if ПОЛУХОД=2 **then** ПЕЧАТЬ ХОДА (0, **true**, 1);

вставляемой в ведущую часть программы (см. разд. 7) непосредственно после оператора ВЫПОЛНЕНИЕ ХОДА(Б, БЕЛ, ХОД БЕЛЫХ);

Например, с помощью теста 4 при решении задачи 2 ТАНДЕМ (см. ее текст в разд. 14) были напечатаны следующие первые ходы белыми, предшествующие выдаче ключевого хода:

П B6—B7	П B6:A7	К G5—E4	К G5—E6	К G5—F3	К G5—F7
К G5:H7	Ф F5—E5	Ф F5—D5	Ф F5—F6	Ф F5—F7	Ф F5—F8
Ф F5—F4	Ф F5—F3	Ф F5—F2	Ф F5—F1	Ф F5—G6	Ф F5:H7
Ф F5—F4					

Тест 5. Печатание списка. Для проверки правильности работы процедуры ПЕРЕПИСЬ ХОДОВ чаще всего приходится пользоваться обращением к нижеприведенной процедуре ПЕЧАТЬ СПИСКА.

```
procedure ПЕЧАТЬ СПИСКА;
begin output('/','T','ПОЛУХОД=','Z3D',ПОЛУХОД,'/');
    for i:=1 step 1 untii РУБЕЖ ХОДОВ[ПОЛУХОД] do
      begin output('Z—3DB',СПИСОК[i]);
        if СПИСОК[i]<0 then output('/')
      end;
    output('/')
end ПЕЧАТЬ СПИСКА;
```

Описание этой процедуры вставляется в начало ведущей программы (вслед за любым другим описанием), а обращение к ней — вслед за обращением к процедуре ПЕРЕПИСЬ ХОДОВ (например, в процедуре ЗАПИСЬ ПЕРВЫХ ХОДОВ, описанной в разд. 6).

Например, список первых ходов белыми (первая группа ходов в массиве СПИСОК), возможных в исходной позиции задачи 1 Валодао (см. ее текст выше), был напечатан тестом 5 следующим образом:

ПОЛУХОД=1

9	1	56	64	—56										
8	1	49	57	—49										
7	3	44	53	62	35	26	17	51	58	37	30	23	16	—44
6	2	21	38	6	36	4	31	27	11	—21				
5	1	20	28	—20										
4	2	13	30	28	23	7	19	3	—13					
3	4	1	2	9	17	25	33	41	—1					
2	1	15	23	31	—15									
1	6	5	4	12	6	14	—5							

Разумеется, использование вышеприведенных пяти тестов не может охватить все возможные случаи отладки, и программисту приходится иногда дополнять эти «стандартные тесты» вновь составляемыми им временными тестами. Однако, как показала практика, вышеприведенные тесты охватывают подавляющее большинство случаев отладки и оказываются полезными при выявлении почти любой ошибки в данной программе.

Кроме вышеприведенных тестов, программисту, работающему с перфокартами, полезно иметь наготове наборы и других, более простых карточек. Например, полезно

иметь под рукой карточку с обращением к процедуре «ПЕЧАТЬ ПОЗИЦИИ;», которая вставляется в различные места программы в зависимости от выявляемой ошибки.

Часто (например, при «зацикливании» программы) бывает необходимо проследить порядок выполнения операторов на некотором участке программы. Для этого полезно держать наготове набор карточек, на каждой из которых набито по одной из следующих строк:

```
output ('T','N1—');
output ('T','N2—');
output ('T','N3—');
. . . . . .
```

Тогда для выявления ошибки достаточно будет только вставлять эти карточки между операторами, порядок выполнения которых нужно проследить.

14. Результаты отладки

Как уже говорилось выше, алгоритм 171б был получен из алгоритма 50CJ в результате ряда последовательных изменений, каждое из которых преследовало те или иные конкретные цели (например, ускорение работы программы, сокращение ее записи или расходуемой ею памяти, повышение наглядности программы и удобств пользования ею). Для исключения возникавших при этом ошибок после каждого такого изменения программа проверялась на большом числе специально подобранных контрольных задач, которые можно разбить на следующие группы.

I. Задачи конкурсного типа, т. е. задачи, публикуемые обычно в открытой печати с целью организации конкурса по их решению. В качестве таковых были взяты те 20 задач, которые публиковались в газете «Вечерняя Москва» с декабря 1972 по февраль 1973 года под рубрикой «Конкурс-26». Разумеется, для этой цели можно было взять задачи и из любого другого конкурса. «Конкурс-26» был использован в данном случае лишь потому, что публикация его задач совпала с началом отладки алгоритма 50CJ, а в дальнейшем в процессе совершенствования алгоритма важно было следить за тем, чтобы каждое новое изменение сопровождалось его ускорением или же во всяком случае не сопровождалось его замедлением. Поэтому для проверки быстродействия важно было пользоваться все время одними и теми же задачами.

Тексты первых десяти двухходовых задач из «Конкурса-26» были уже опубликованы в материалах алгоритма 50CJ [48, с. 115], а тексты десяти трехходовок из этого конкурса были опубликованы в «Подтверждении к алгоритму 50CJ» [49, приложение 2]. Время исследования на однозначность решения этих 20 задач с помощью алгоритмов 171б и 50CJ приведено в табл. 43.

Таблица 43

Двухходовые задачи			Трехходовые задачи		
Номер задачи	Время исследования		Номер задачи	Время исследования	
	Алгоритм 50CJ	Алгоритм 171б		Алгоритм 50CJ	Алгоритм 171б
1	39 с	17 с	11	9 мин 28 с	4 мин 09 с
2	45	19	12	20 11	7 13
3	2 мин 21	1 мин 01	13	9 36	4 03
4	3 29	1 47	14	19 32	7 58
5	35	17	15	8 04	3 10
6	3 22	1 40	16	19 07	8 36
7	3 29	37	17	23 11	6 38
8	56	29	18	1 ч 37 07	43 18
9	4 21	1 58	19	13 56	5 14
10	31	19	20	19 35	7 25
Среднее значение	1 мин 53 с	42 с	Среднее значение	23 мин 29 с	9 мин 34 с

II. Задачи типа Валодао, т. е. задачи, использующие все три вида специальных ходов: превращение пешки, взятие пешки на проходе и рокировку. Поскольку задачи такого типа являются редкостью, то возможностей для их выбора у автора алгоритма

171б не было. Поэтому для отладки программы ЭРА-77 были взяты только те восемь задач Валодао — таск, которые использовались и для отладки алгоритма 50CJ [48, с. 116] и [49, приложение 2], тем более, что это способствовало сравнению двух алгоритмов по быстродействию. Эти задачи были ранее опубликованы в еженедельнике «64» № 21 от 25—31 мая 1973 года в статье «Таких три разных хода».

Время исследования на однозначность решения этих восьми задач (первые шесть из них двухходовки, а две последние — трехходовки) с помощью алгоритмов 171б и 50CJ приведены в табл. 44.

Таблица 44

Номер задачи	Время исследования		Номер задачи	Время исследования	
	Алгоритм 50CJ	Алгоритм 171б		Алгоритм 50CJ	Алгоритм 171б
1	1 мин 14 с	33 с	5	1 мин 41 с	44 с
2	1 24	35	6	1 18	38
3	3 02	1 мин 30	7	2 ч 7 56	48 мин 03
4	3 00	1 29	8	7 5 00	2 ч 27 28

III. Задачи типа тандем, т. е. задачи, содержащие рокировку и взятые из статьи В. Карпова «Играет тандем», опубликованной в еженедельнике «64» № 6 (293) от 8—14 февраля 1974 года. Задачи этого типа были взяты для отладки алгоритма 171б ради более надежной проверки операторов рокировки. В статье В. Карпова было приведено шесть таких задач, но для отладки алгоритма 171б использовались только пять из них: двухходовые задачи 1 и 2 и трехходовые задачи 3, 4 и 6. Четырехходовая задача 5 для отладки не использовалась, потому что для ее решения требовалось очень много машинного времени. Текст использованных задач приведен ниже.

1. Белые: Крe1, Фe7, Ла1, Лh1, Сc3, Сc4, Кd3, Кe6, g2.
   Черные: Крe3, Фb6, Сh6, Кf5, b5, c6, g3.
2. Белые: Крh3, Фf5, Сb5, Сc5, Кg5, b6, h6.
   Черные: Крe8, Фa4, Ла8, Лh8, Сe7, a7, b3, c4, d7, h7.
3. Белые: Крe1, Фg7, Ла1, Лh1, Сf7, Кd6, a5, c2, c3, f5, g3, g6.
   Черные: Крe5, Кg8, d7, e7, f6.
4. Белые: Крh5, Лh7, Кb6, Кc5.
   Черные: Крe8, Ла8, Сe1, a5, b7, f5, f6.
6. Белые: Крe1, Фa8, Ла1, Лh1.
   Черные: Крf7, b7.

Время исследования этих задач на однозначность решения было равно 5 мин 50 с, 2 мин 04 с, 23 мин 39 с, 11 мин 06 с, и 13 мин 12 с соответственно.

IV. Четырехходовые задачи. Для отладки алгоритма 171б были взяты, во-первых, три первые четырехходовые задачи из книги «Альбом ФИДЕ. 1914—1944/II» с номерами 839—841, которые уже использовались ранее для отладки четырехходовой модификации алгоритма 50CJ [49, приложение 2], и, во-вторых, две следующие простые четырехходовые задачи.

1. Задача из еженедельника «64» № 29 за 1975 г., с. 14.
   Белые: Крf2, e7.
   Черные: Крh1, d2.
2. Задача из журнала «Наука и жизнь» № 8 за 1975 год, с. 138.
   Белые: Крf8, Лh6.
   Черные: Крh8, Кa7, h7.

Время исследования этих двух последних задач было равно 1 мин 38 с и 21 мин 14 с соответственно. Время исследования задач из «Альбома ФИДЕ» приведено в табл. 45.

Таблица 45

Номер задачи	Время исследования	
	Алгоритм 50CJ	Алгоритм 171б
839	12 мин 00 с	4 мин 34 с
840	5 ч 58 28	1 ч 48 38
841	2 36 56	1 01 04

V. Пятиходовые задачи. В качестве таковых для отладки были взяты две следующие задачи-малютки.

1. Задача из еженедельника «64» № 29 за 1975 год, с. 14.
   Белые: Крf2; а6.
   Черные: Крh1, f6.
2. Задача из еженедельника «64» № 6 за 1975 год, (раздел «Публикуется впервые»).
   Белые: Крс5, а7, с6.
   Черные: Кра8.

Время исследования этих задач было равно 7 мин 29 с и 7 мин 04 с соответственно.

Во всех вышеуказанных случаях решения по алгоритму 171б указывается время его работы, которое выдавалось на печать вариантом транслятора БЭСМ-АЛГОЛ от 20.5.76 с диспетчером ДИСПАК от 12.3.77. При этом от общего использованного времени всегда отнималось 10 с, примерно затрачиваемых на трансляцию и подготовку к печати исходной позиции.

15. Предметный указатель алгоритма 171б

В данный указатель включены идентификаторы и термины, неоднократно употребляемые в описании алгоритма, и разделы, в которых даны их определения.

АДРЕСА — 1
БЕЛ — 3, 6
БКР — 8
БЛД — 8
БЛК — 8
ВВОД ПОЗИЦИИ — 1, 7
ВОЗВРАТ КОРОЛЯ — 5, 8
ВОЗВРАТ ХОДА — 5, 7
ВСПОМОГАТЕЛЬНЫЕ СИМВОЛЫ — 1
ВЫПОЛНЕНИЕ ХОДА — 4, 7
ВЫЧИСЛЕНИЕ ТАБЛИЦ — 7, 12
Группа ходов — 3
ДЛИН — 6
ДОСКА — 1
ЕСТЬ ЛИ ШАХ — 7, 11
ЗАДАНА РОКИРОВКА — 7, 8
Задание формы результата — 1
ЗАПИСЬ ПЕРВЫХ ХОДОВ — 6, 7
ЗАПИСЬ РОКИРОВКИ — 7, 8
Иллюстративные переменные — 1, 6
ИМЯ ЗАДАЧИ — 1
ИМЯ ПРОГРАММЫ — 1
ИМЯ ФИГУРЫ — 1
Индикаторы рокировки — 8
Информация о задаче — 1
Квазиабсцисса — 1
Квазикоординаты — 1
Квазиордината — 1
Кодировка полей — 1
Кодировка фигур — 1
КОНЬ — 6
КОРОЛЬ — 6
КОРОТ — 6
ЛАДЬЯ — 1, 6
НКР — 5, 8
НОМЕР — 1
Номер фигуры — 1
ОПРЕДЕЛЕНИЕ РОКИРОВКИ — 7, 8
Отладочная задача № 1 — 1
Относительный код фигуры — 3
ПЕРЕПИСЬ ХОДОВ — 3, 7
ПЕЧАТЬ ПОЗИЦИИ — 2, 7
ПЕЧАТЬ ПОЛЯ — 7, 9
ПЕЧАТЬ РЕШЕНИЯ — 7, 9
ПЕЧАТЬ ХОДА — 7, 9
ПЕШКА — 4, 6
ПЕШКА В ПРЕВРАЩЕНИИ — 4, 6
ПЕШКА НА ПРОХОДЕ — 4, 6
ПКР — 5, 8
ПОДГОТОВКА ПОИСКА ХОДОВ 2—7, 10
ПОДГОТОВКА ПАМЯТИ — 6, 7
Подгруппа ходов — 3
ПОЛЕ — 3
ПОЛУХОД — 7
РУБЕЖ — 3
РУБЕЖ ХОДОВ — 3
СЛОН — 6
СПИСОК — 3
С ХОДАМИ 2—7, 10
ФЕРЗЬ — 6
ХОД — 7
ХОД МАТА — 7, 9
ЦВЕТ — 3
ЧЕРН — 6
ЧИСЛО БЕЛЫХ — 1
ЧИСЛО ЧЕРНЫХ — 1
ЧКР — 1, 8
ЧЛД — 1, 8
ЧЛК — 1, 8

Подтверждение к алгоритму 171б

В. М. Агеев, С. В. Шехматов, Москва, декабрь 1978

С помощью алгоритма 171б на машине БЭСМ-6 в системе БЭСМ-АЛГОЛ были проверены решения всех 243 (вместе с вариантами-«близнецами») двухходовок с номерами от 1 до 220, опубликованных в «Альбоме ФИДЕ 1968—1970» [95]. Были получены правильные решения по всем этим задачам, за исключением тех, которые были приведены в «Альбоме» с дефектами. Так, оказалось, что не имеют решений задачи 1 и 8. В задаче 56 кроме основного решения Фf2—f1 обнаружено второе решение Кf7 : e5, а в задаче 165b кроме основного решения Фе3—с5 найдено еще решение

Таблица 46

n	$t_{ср}$	n	$t_{ср}$	n	$t_{ср}$
6	24 с	14	1 мин 26 с	21	3 мин 17 с
7	22	15	1 26	22	3 18
8	36	16	1 40	23	3 22
10	35	17	3 40	24	4 09
11	45	18	2 56	25	3 38
12	59	19	3 09	26	4 50
13	1 мин 30	20	2 51		

Ke8—c7+. Для трех задач истинные решения не совпадают с приведенными в «Альбоме». А именно, для задачи 37 решением является Ld1—d8 (в ответе Ld1—d2), для задачи № 40 — Фc7—a5 (в ответе Фc7—a4?), для задачи 153 — Ca2—c4 (в ответе Kc4). Задача 171 решается матом в один ход (Kc5—e6), а в 193 возможны два мата в один ход (Фf7—g7 и Фf7—f6).

Таблица 47

Номер задачи	n	t	Номер задачи	n	t	Номер задачи	n	t
1	5	19 с	97	18	4 мин 53 с	212a	23	1 мин 26 с
2	6	31	98	18	3 05	212b	23	1 23
3a	6	19	99	18	4 51	213	23	4 58
3b	6	25	100	18	3 32	214	23	49
3c	6	22	101	18	3 43	215	23	5 28
3d	6	22	102	18	3 54	216	24	4 09
4	7	27	103	18	3 05	217	25	5 03
5	7	17	104	18	1 22	218	25	2 56
6	8	36	105	18	2 17	219	25	2 54
7	10	37	106	18	54	220	26	4 50

Среднее время решения задач

25 с		3 мин 10 с	3 мин 24 с

Среднее время ($t_{ср}$), затраченное на решение задач с данным числом фигур (n), указано в табл. 46. В табл. 47 приведены количества фигур (n) и времена решений (t) каждой из первых десяти задач с номерами от 1 до 7, десяти задач с номерами от 97 до 106 и последних десяти задач с номерами от 212a по 220. Во всех случаях решение задач проводилось с полным перебором всех ходов, возможных в заданной позиции.

Замечание к алгоритму 171б

В. М. Агеев, Москва, март 1979

Для удобства пользователя были сделаны две модификации алгоритма 171б, заключающиеся в следующем.

1. Цифровой ввод начальной позиции решаемой задачи заменен буквенным вводом. Например, вводимая в машину информация о начальной позиции задачи № 3* раньше записывалась

2; 8; 5; 61; 1; 54; 2; 48; 2; 15; 1; 12; 4; 8; 6; 5; 3; 3;

6; 3; 41; 1; 29; 4; 25; 1; 24; 1; 23; 6; 22;

* См. [48, с. 116], где для исправления записи позиции надо добавить две белые пешки на d2 и на f7.

После сделанной модификации эту начальную позицию стало возможным записывать в гораздо более привычной для шахматистов форме

2; 8; 'Крe1, Фе8, Лh1, Сc1, Kg2, Кh6, d2, f7'

6; 'Крf3, Ла4, Са6, e4, g3, h3'

В общем случае, при буквенном вводе, как и при цифровом, сначала указывается число ходов в задаче, затем — число белых фигур, далее, в кавычках, обычная для шахматистов запись позиций белых фигур. После этого указывается число черных фигур, а затем, в кавычках, обычная, для шахматистов запись позиций черных фигур. Порядок следования позиций фигур в этих записях произвольный.

Изменения в программе заключались в следующем:

а) в начале программы был дополнительно описан целый массив ТЗ[0 : 65],

б) начало процедуры ВВОД ПОЗИЦИИ до метки КОНТРОЛЬ ВВОДА включительно заменено следующими строками:

```
procedure БУКВЕННЫЙ ВВОД;
begin integer ЦВЕТ;
      input (ЧИСЛО БЕЛЫХ, ТЗ[0]);
      k:=ЧИСЛО БЕЛЫХ; ЦВЕТ:=БЕЛ;
n1:   j:=1;
      for i:=ЦВЕТ step ЦВЕТ until k do
        begin if ТЗ[j+1]<10 then
          begin ФГ:=ПЕШКА; j:=j—1; go to n2 end;
          if ТЗ[j+1]=48 then
            begin ФГ:=КОРОЛЬ; j:=j+1; go to n2 end;
          t:=ТЗ[j];
          ФГ:=if t=42 then КОНЬ else
            if t=49 then СЛОН else
              if t=43 then ЛАДЬЯ else ФЕРЗЬ;
n2:       t:=ТЗ[j+1];
          ПОЛЕ:=(ТЗ[j+2]—1)×8+
            (if t=32 then 1 else
              if t=34 then 2 else
                if t=49 then 3 else
                  if t=63 then 4 else
                    if t=37 then 5 else
                      if t=64 then 6 else
                        if t=65 then 7 else 8);
          ДОСКА[ПОЛЕ]:=ФГ×ЦВЕТ; АДРЕСА[i]:=ПОЛЕ;
          if ЦВЕТ=ЧЕРН then go to n3;
          if ФГ=КОРОЛЬ then БКР:=i else
            if ФГ=ЛАДЬЯ∧ПОЛЕ=8 then БЛК:=i else
              if ФГ=ЛАДЬЯ∧ПОЛЕ=1 then БЛД:=i;
          go to n4;
n3:       if ФГ=КОРОЛЬ then ЧКР:=i else
            if ФГ=ЛАДЬЯ∧ПОЛЕ=64 then ЧЛК:=i else
              if ФГ=ЛАДЬЯ∧ПОЛЕ=57 then ЧЛД:=i;
n4:       j:=j+4
        end i;
      if ЦВЕТ=БЕЛ then
        begin input (ЧИСЛО ЧЕРНЫХ, ТЗ[0]);
          k:=—ЧИСЛО ЧЕРНЫХ; ЦВЕТ:=ЧЕРН; go to n1
        end;
КОНТРОЛЬ ВВОДА:
```

В этих операторах цифры 48, 42, 49, 43, 32, 34, 63, 37, 64, 65 являются кодами (в системе БЭСМ — АЛГОЛ) букв Р, К, С, Л, А, В, D, E, F, G соответственно, записанными в десятичной системе,

в) обращение к процедуре ВВОД ПОЗИЦИИ (после метки НАЧАЛО) заменено на обращение к процедуре БУКВЕННЫЙ ВВОД.

2. В программу внесены изменения, в результате которых стало возможным одновременное решение нескольких задач при однократном вводе программы и однократной ее трансляции. Изменения заключались в следующем:

а) в начале программы были дополнительно описаны две целые переменные g и ЧИСЛО ЗАДАЧ (параметр цикла и количество решаемых задач, соответственно),

б) после метки ВВОД ИНФОРМАЦИИ О ПРОГРАММЕ и ЗАДАЧЕ вместо оператора

input(ИМЯ ПРОГРАММЫ[1], С ХОДАМИ2, ИМЯ ЗАДАЧИ[1],n);
вставлены следующие строки:

```
input(ИМЯ ПРОГРАММЫ[1], С ХОДАМИ2, ЧИСЛО ЗАДАЧ);
for g:=1 step 1 until ЧИСЛО ЗАДАЧ do
  begin input(ИМЯ ЗАДАЧИ[1],n);
```

в) в конце программы добавлен символ **end**.

Модифицированная программа была проверена на машине БЭСМ-6 для решения четырех задач. При этом перед информациями о задачах добавлялась карта с набитыми на ней символами «4;».

Подтверждение к алгоритму 171б

Н. М. Кимлик, Рига, апрель 1979

Алгоритм 171б был переведен на язык ПЛ-1 и проверен на машине ЕС-1050 (быстродействие 1000000 оп./с) почти для всех задач, приведенных в выпусках «Библиотека алгоритмов». При этом никаких ошибок в программе обнаружено не было.

Общее время трансляции программы было десять минут.* Время решения задач на машине ЕС-1050 оказалось примерно в 1,5 раза большим, чем на машине БЭСМ-6 по программе на языке АЛГОЛ-60.

Все процедуры этого алгоритма транслировались один раз и заносились в личную библиотеку объектных модулей на магнитных дисках.

АЛГОРИТМ 68CJ

Белые начинают и дают мат в *n* ходов (рекурсивные процедуры)

Пояснительный текст к алгоритму 68

Дж. Р. Маннинг (Manning J. R. «The Comp. J.», 1971, № 2) **

Алгоритм 50 А. Белла [55] для программирования шахматной игры побудил меня представить на рассмотрение законченную программу решения шахматных задач типа «Белые начинают и дают мат в *n* ходов».

В некоторых отношениях данный подход к решению задачи отличается от подхода А. Белла. Например, данная программа пользуется «окаймленной доской» так, как это показано на рис. 14. 64 поля в центре соответствуют фактической доске и называются фактическими, внешние поля именуются формальными. Это упрощает вычисление ходов: например, если белый конь находится на поле m, то он может ходить на любое из полей $m\pm21$, $m\pm19$, $m\pm12$ или $m\pm8$ при условии, что удовлетворяются следующие требования.

1. Поле назначения является фактическим.
2. Поле назначения не занято белой фигурой.
3. Белый король не находится под шахом.

В начале программы элементам логического массива *val*[—20 : 99] *** присваиваются значения **true** для фактических полей и **false** для формальных. Перед выполнением каждого предполагаемого хода делается проверка (по массиву *val*), не является ли поле назначения формальным.

В шахматной задаче должны быть опробованы все возможные ходы и все возможные ответы на каждый сделанный ход и т. д. Следовательно, бывает много возвратов к предыдущей позиции (см. на рис. 15 блок-схему для трехходовой шахматной задачи), и нужно запоминать состояние шахматной доски после каждого хода. В случае n-ходовой задачи нужно запоминать $2n+1$ позиций (включая начальную). Состояние доски описывается двумерным целым массивом $a[1 : 2\times n+1, -20 : 99]$. Значение $a[x, m]$ указывает, какой фигурой занято поле m непосредственно перед полуходом x (А. Белл называет значение x уровнем). Значение $a[x, m]$ бывает равным нулю, если

---

* Поскольку быстродействие машин ЕС-1050 и БЭСМ-6 примерно одинаково, то представляет интерес сравнение времени трансляции программы ЭРА-77 на языке ПЛ-1 (около 10 мин.) со временем трансляции ее на языке АЛГОЛ-60 (около 15 с). (*Прим. ред.*)

** См. [56]. Русский перевод выполнен М. И. Агеевым. Индекс CJ у номера алгоритма указывает на источник, в котором был опубликован исходный вариант этого алгоритма (журнал «The Computer Journal»). (*Прим. пер.*)

*** Сокращение от слова valed — фактический (*Прим. пер.*)

	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0
-1	0	1	2	3	4	5	6	7	8	9	10
9	10	11	12	13	14	15	16	17	18	19	20
19	20	21	22	23	24	25	26	27	28	29	30
29	30	31	32	33	34	35	36	37	38	39	40
39	40	41	42	43	44	45	46	47	48	49	50
49	50	51	52	53	54	55	56	57	58	59	60
59	60	61	62	63	64	65	66	67	68	69	70
69	70	71	72	73	74	75	76	77	78	79	80
79	80	81	82	83	84	85	86	87	88	89	90
89	90	91	92	93	94	95	96	97	98	99	

Рис. 14. Нумерация полей шахматной доски

поле m свободно, равным —1, —3, —4, —7, —8, —9, если оно занято неприятельской пешкой, конем, слоном, ладьей, ферзем или королем соответственно, и +1, +3, +4, +7, +8, +9, если оно занято своей пешкой конем, слоном, ладьей, ферзем или королем.

После каждого пробного хода делается обращение к логической процедуре *ownch* * для проверки, не оставлен ли собственный король под шахом. Если не оставлен, то шахматная доска «переворачивается» (путем $a[x+1, m] := -a[x, 79-m]$) и вычислительная машина готова играть за противника.

Логический массив *stal* ** используется для предотвращения пата.

Основная идея программы — чередование процедур *scanmo* *** и *trym* ****. После ввода начальной позиции программа обращается к процедуре *scanmo* со значением полухода $x=1$ и обозревает все возможные ходы белых. Когда найден законный ход, машина выполняет его с помощью процедуры *trym*, переворачивает доску и вновь рекурсивно обращается к процедуре *scanmo* со значением полухода, увеличенным на единицу, для обзора возможных ответов черных. Процедуры *scanmo* и *trym* взаимнорекурсивные, но поскольку язык Эллнот-АЛГОЛ не разрешает использовать процедуру до ее описания, то приходится прибегать к помощи формального параметра *r* процедуры *trym*, заменяемого при каждом обращении к процедуре *trym* фактическим параметром *scanmo*.

Данная программа допускает превращение пешки в фигуру (причем не обязательно в ферзя) и взятие пешки на проходе (за исключением взятия на проходе первым ходом), но не допускает рокировку. Хотя в шахматных задачах рокировка допустима, все же, если не известна предыстория данной позиции, никогда нельзя быть уверенным, что рокировка законна *****.

Данная программа была транслирована на машине ICL-Elliott 4120 и была проверена на нескольких задачах, взятых из книги Липтона, Мэтьюза и Райса. Решение двухходовых задач (№№ 40, 80, 280 и 320) заняло 10, 36, 7 и 46 мин, а простая трехходовая задача (№ 267) потребовала почти 2 ч. Следовательно, данная программа работает медленно: время, затрачиваемое на решение четырехходовой задачи, может измеряться неделями, пятиходовой — годами, а шестиходовой — столетиями!

---

* От own — собственный и check — шах. (*Прим. пер.*)

** Сокращение от слова stalmate — пат. (*Прим. пер.*)

*** Сокращение от слова scan — обзор и moves — ходы. (*Прим. пер.*)

**** Сокращение от слова try — пробовать и moves — ходы. (*Прим. пер.*)

***** В настоящее время принято (см., например, [57]) считать, что рокировка законна, если нельзя доказать обратное. (*Прим. пер.*)

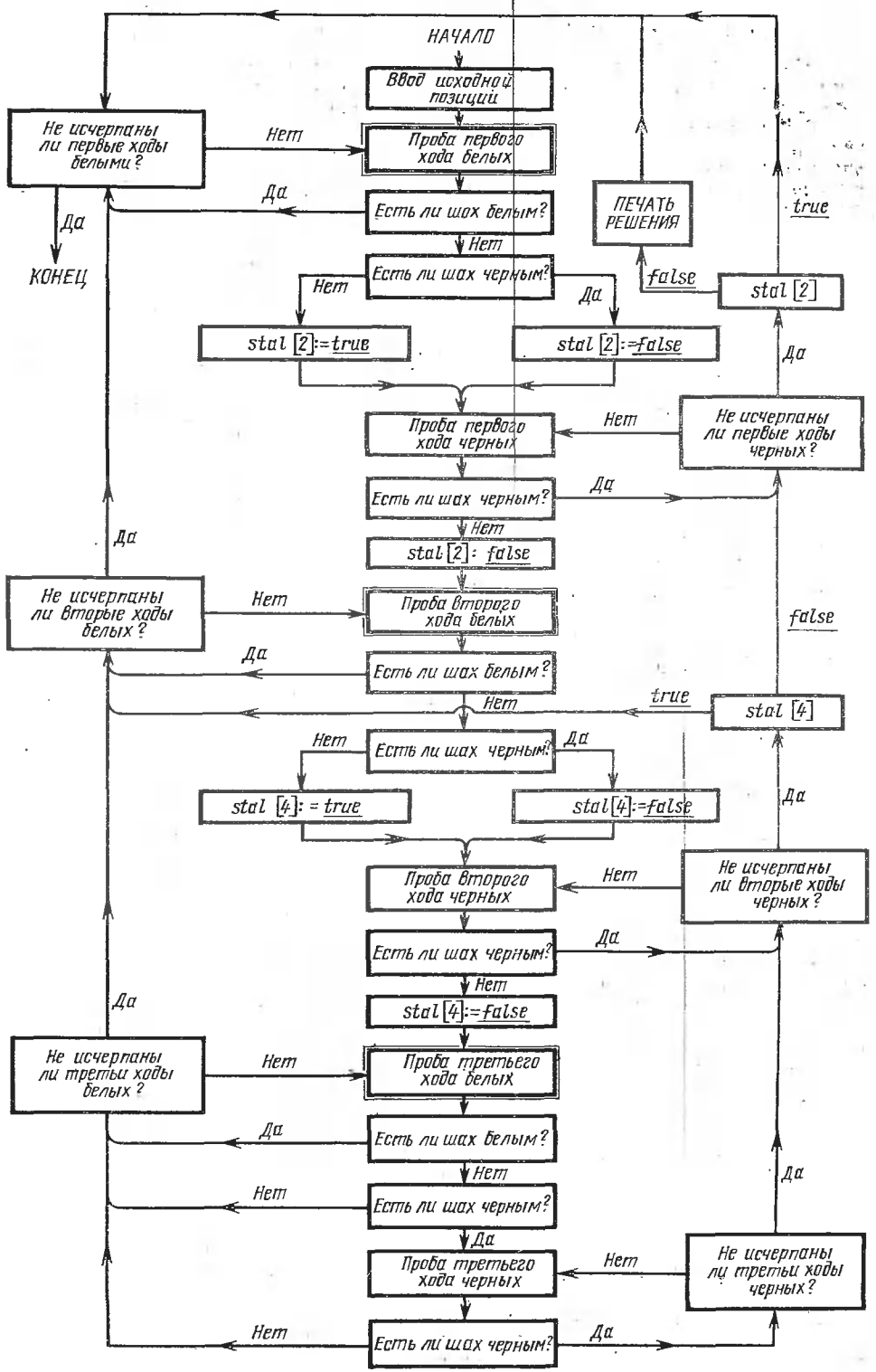

Рис. 15. Блок-схема решения трехходовой задачи.

Свидетельство к алгоритму 68CJ

Пояснительный текст алгоритма 68CJ является дословным переводом пояснительного текста к алгоритму 68 (Дж. Р. Маннинг. «The Comp. J.», 1971, № 2). Программа этого алгоритма не публикуется здесь по следующим соображениям.

1. Как видно из вышеприведенного пояснительного текста, программа Дж. Маннинга работает с чрезмерно большими затратами машинного времени и позволяет практически решать на большинстве современных машин только двухходовые и лишь некоторые простейшие трехходовые задачи. Поэтому она не может быть рекомендована широкому кругу пользователей, тем более что выше была описана программа ЭРА-77, позволяющая уже сегодня решать любые трехходовые, большинство четырехходовых и даже простейшие пятиходовые задачи.

2. Процедуры алгоритма Дж. Маннинга являются рекурсивными и, следовательно, не могут использоваться в транслирующих системах, работающих с сокращенного языка АЛГОЛ-60 такого, например, как получивший в нашей стране широкое распространение алгоритмический язык АЛГАМС [58, 92, 93].

3. Алгоритм Дж. Маннинга предназначен для решения только таких задач, которые не используют рокировку, в то время как алгоритм 171б решает любую шахматную задачу.

4. Попытка решить на машине БЭСМ-6 с помощью программы Маннинга даже такую простейшую задачу, как вторая задача А. Белла [55, с. 91, рис. 4], оказалась безрезультатной. Не исключено, что кроме перечисленных ниже в алгоритме Дж. Маннинга имеются еще какие-то другие ошибки, затрачивать время на выявление которых было сочтено здесь нецелесообразным ввиду бесперспективности практического применения алгоритма.

Тем не менее некоторые специалисты, посвятившие себя проблеме шахматного программирования, могут пожелать разобраться в алгоритме Дж. Маннинга более детально с целью выявления и заимствования из него тех или иных технических приемов *. Для облегчения их работы ниже дается не только перечень ошибок, уже обнаруженных в программе Дж. Маннинга, но и перевод всех комментариев и некоторых идентификаторов, используемых этой программой.

В алгоритме 68 [56] были замечены следующие очевидные ошибки.

1. с.211, fig.2, Block diagram, второй прямоугольник снизу. Вместо «Try a first move for black» должно быть «Try a third move for white».

2. с.211, fig.2, четвертый ромбик снизу. Вместо «NO» должно быть «YES».

3. с.212, левая колонка, 4-я стр. сверху. Вместо

x:=x—1: white:=¬ white;

должно быть

x:=x—1; white:=¬white;

4. с.212, правая колонка, 29-я стр. снизу. Вместо

if w=7∨w=9 then

должно быть

if w=7∨w=8 then

5. На с. 212 в двух местах (левая колонка, стр. 11—18 снизу и правая колонка, стр. 9—16 сверху) оператор цикла с заголовком for k:=3, 4, 7, 8 do должен быть заключен в операторные скобки.

Кроме тех формальных параметров, смысл которых был уже разъяснен выше в пояснительном тексте, русскому читателю может оказаться полезной расшифровка следующих идентификаторов, используемых в алгоритме Дж. Маннинга:

bishop — слон,
ep — на проходе (от en passent),
exh — исчерпан (от exhausted),
king — король,
knight — конь,
pawn — пешка,
print — отпечатать,
queen — ферзь,
retrace — возвратиться,
rook — ладья,
white — белые.

В АЛГОЛ-программе алгоритма 68 используются следующие пояснительные тексты и комментарии.

---

* В частности, ознакомление с алгоритмом Дж. Маннинга навело автора алгоритма 171б на идею различения знаками кодов белых и черных фигур и составление новой процедуры ЕСТЬ ЛИ ШАХ (хотя конструкция этой процедуры ничего общего с алгоритмом Дж. Маннинга не имеет). (*Прим. пер.*)

Третья строка программы
print Сколько в задаче ходов'
В процедуре ownch
comment Получает значение **true**, если свой король под шахом;
comment mk — это поле, заиятое королем;
В процедуре trym
comment Данная процедура делает пробиый ход, передвигая фигуру с поля *p* на поле *q*. Фактический параметр, соответствующий параметру *r*, всегда *scanmo*. Процедуру *scanmo* нельзя вызывать как глобальную, поскольку ее описание помещено после обращения к ней;
comment Доска переворачивается так, чтобы машина играла за противника;
print 'мат на', j÷2. 'ходе', ':ключевой ход', p, q;
В процедуре scanmo
После заголовка процедуры
comment Эта процедура обозревает один за другим все возможиые ходы. Когда найден очередной возможный ход, вызывается процедура trym;
После оператора w:=ep[x—1];
comment Взятие пешки иа проходе;
После оператора **if** w<1 **then go to** L4;
comment Ход пешкой;
После заголовка **for** k:=3,4,7,8 **do begin**
comment Превращение пешки в коня, слона, ладью или ферзя;
После условия **if** m<19 **then**
begin comment Взятие пешкой;
После заголовка **for** k:=3,4,7,8 **do**
begin comment Превращение пешки при взятии фигуры противника;
После строки **end** pawn move;
comment Ход конем;
После строки **end** knight move;
comment Диагональный ход слоном или ферзем;
После строки Q7: **end** diagonal move;
comment Ортогональный ход ладьей или ферзем;
После строки Q8: **end** orthogonal move;
comment Ход королем;
В ведущей части программы
comment На первом ходе белыми взятие пешки на проходе запрещено;
print'другого решения нет'.

ПРИЛОЖЕНИЕ 2

Подтверждения и замечания к алгоритмам, опубликованным в предыдущих выпусках

Подтверждение к алгоритмам 4б, 5б, 7б, 8б, 10б, 11б, 12б, 13б, 14б, 17б, 18б, 19б, 20б, 22б, 24б, 29б, 36б, 39б, 46б, 48б, 49б, 50б, 51б, 52б, 53б, 55б, 56б, 57б, 66б, 67б, 70б, 72б, 75б, 78б, 80б, 84б, 94б, 96б, 97б, 99б

М. П. Решетник, Житомир, март 1979

Выражая искреннюю благодарность за Вашу работу* по редактированию алгоритмов, сообщаю, что перечисленные в заголовке данного подтверждения алгоритмы были переведены нами на алгоритмический язык АЛМИР, проверены на примерах, опубликованных в соответствующих свидетельствах, и длительное время используются в работе на ЭВМ «МИР-1».

В некоторые процедуры были внесены изменения, связанные с тем, что нумерация массивов в языке АЛМИР начинается не с нуля, а с единицы. Кроме того, в некоторых случаях процедуры были упрощены, ибо для машины «МИР» переполнения не страшны.

---

* Данное подтверждение было направлено в форме письма редактору выпусков. Ко всем подтвержденным здесь алгоритмам М. П. Решетник представил листинги соответствующих расчетов. (*Прим. ред.*).

При этом были обнаружены следующие опечатки.

1. В алгоритме 4б на с. 17, 15-я строка снизу напечатано

x: real procedure f;

должно быть

x; real procedure f;

2. В алгоритме 20б на с. 44, 3-я строка снизу напечатано

—Ei(—10)=0,415696012

должно быть

—Ei(—10)=0.4156969012$_{10}$—5

3. В алгоритме 39б на с. 100, 4-я строка сверху напечатано

0,97352692

должно быть

0.97352692

4. В алгоритме 40б на с. 101, 13-я строка сверху напечатано (2,4)) должно быть (2, 4).

5. В алгоритме 75б на с. 50, 7-я строка сверху напечатано

for i:=0 step 1 until n — do

должно быть

for i:=0 step 1 until n do

Смею надеяться, что издание выпусков «Библиотеки алгоритмов» будет продолжено.

Подтверждение к алгоритмам 7б, 8б, 35б, 43б, 50CJ, 72б, 130б

В. С. Рукавица, г. Лозовая, Харьковской обл., апрель 1980

Перечисленные в заголовке алгоритмы были транслированы и успешно используются нами в системе АЛГАМС машины МИНСК-32 (транслятор ТАМ-32, ред. 6-77). Некоторые алгоритмы были переведены на ФОРТРАН и используются в системе ТФ-1.

Текст алгоритма 50CJ пришлось несколько изменить, чтобы обойти недостатки транслятора ТАМ-32. В частности, процедуры ХОДЫ ПЕШЕК, ХОДЫ СЛОНА И ЛАДЬИ, ХОДЫ КОНЯ И КОРОЛЯ пришлось сделать глобальными и ввести в программу дополнительно некоторые переменные.

С помощью алгоритма 50CJ было решено несколько задач, причем время решения первой задачи, на которой отлаживался алгоритм 50 А. Белла, равнялось 50 с.

Считаю нужным отметить здесь, что значение Ваших выпусков трудно переоценить. Для нас, практиков-прикладников, это и школа программирования, и библиотека алгоритмов, и пища для ума.

Подтверждение и замечания к алгоритмам 17б, 24б, 26б, 26б, 58б, 63б, 64б, 65б, 77б и 178а

Ю. А. Михайлов, Киев, октябрь 1980

Перечисленные в заголовке алгоритмы были переведены на язык ПЛ-1 уровня F ОС ЕС ЭВМ и используются в нашей работе. Результаты решений контрольных задач, приведенных за описаниями этих алгоритмов, совпали с опубликованными. К алгоритмам 24б, 26б и 178а имеются следующие замечания.

Алгоритм 24б. В пояснительном тексте алгоритма нет достаточной ясности, каким именно переменным соответствуют коэффициенты диагоналей выше и ниже главной. Так, например, в книге А. Н. Тихонова и А. А. Самарского «Уравнения математической физики» (М.: Наука, 1977) на с. 590 приведена запись

$$A_i y_{i-1} - C_i y_i + B_i y_{i+1} = -F_i. \qquad (1)$$

В книге Н. Н. Калитина «Численные методы» (М.: Наука, 1978) на с. 133 читаем

$$A_i x_{i-1} - B_i x_i + C_i x_{i+1} = d_i. \qquad (2)$$

В алгоритме 24б решается система уравнений, записанная в форме

$$\begin{aligned} -b_1 x_1 + c_1 x_2 &= d_1, \\ a_1 x_1 - b_2 x_2 + c_2 x_3 &= d_2, \end{aligned} \qquad (3)$$

$$a_{i-1}x_{i-1}-b_ix_i+c_ix_{i+1}=d_i \quad (i=3, 4, \ldots, n-1),$$
$$a_{n-1}x_{n-1}-b_nx_n=d_n. \qquad (3)$$

Однако в практической работе (например, при решении дифференциальных уравнений второго порядка в частных производных) формирование коэффициентов прогонки в таком виде (индекс при a на единицу меньше, чем соответствующие индексы при b и c) несколько неудобно. Поэтому представляется целесообразным видоизменить тело алгоритма 24б так, чтобы он удовлетворял форме записи уравнений (2). При этом длина вектора a увеличится на один элемент, а в теле процедуры $a[i-1]$ заменится на $a[i]$ при условии, что $a[1]=0$, $a[2]=a_1$, $a[3]=a_2$, ..., $a[n]=a_{n-1}$, где a_1, a_2, ..., a_{n-1} — коэффициенты в записи (3).

Расход памяти при этом возрастает на одно машинное слово, однако тот факт, что ие нужно будет дважды вычислять разность $i-1$, это компенсирует. А кроме того повысится удобство пользования алгоритмом. С такими изменениями алгоритм 24б был проверен в ЕС ЭВМ, и результаты решения совпали с приведенными в [47].

Алгоритм 26б. Целесообразность наличия в процедуре *root* оператора if f(0)=0 then go to fin вызывает сомнение. Проверку равенства $f(0)=0$ можно выполнять отдельно до входа в процедуру. Наличие этой проверки в теле процедуры значительно сужает область применимости алгоритма 26б. Так, например, с ним нельзя решать уравнение $x=\mathrm{ctg}(x)$ или искать отличные от нуля корни уравнения $x=\sin(x)$.

Алгоритм 178а. При решении системы уравнений (система Пауэлла)

$$-13+x_1+((5-x_2)x_2-2)x_2=0;$$
$$-29+x_1+((1+x_2)x_2-14)x_2=0$$

было получено *corr*≡true, а *spsi*=48 (вместо должного *spsi*≈0), т. е. решение попало в «крутой овраг». Поэтому для контроля правильности выполнения процедуры *directsearch* недостаточно знать, что *corr*≡true. Надо еще проверить условие $abs(spsi)\leqslant acc$, где *acc* — норма решения.

В заключение пользуюсь случаем, чтобы выразить авторам выпусков «Библиотеки алгоритмов» свою глубокую благодарность за большую и очень нужную работу. Ценность этих выпусков бесспорна. «Библиотека алгоритмов» — это настольная книга программистов. Желательно только, чтобы скорость издания очередных выпусков была более высокой.

Замечания к алгоритму 23б

Ю. В. Гендельман, Волгоград, ноябрь 1979

В подтверждениях к алгоритму 23б С. В. Ратафьева и Ю. Д. Красильникова справедливо указано, что для сортировки числовых массивов этот алгоритм мало пригоден. Однако следует заметить, что при достаточном числе классов k число элементов исходного массива в каждом классе много меньше n (общего числа элементов массива), и алгоритм 23б можно применять для предварительной сортировки, а внутри каждого класса использовать более простой метод. Именно так рекомендует, в частности, использовать этот алгоритм Д. Кнут [102, с. 99—100].

Для упорядочения массива по возрастанию в соответствии с вышесказанным можно предложить следующую процедуру.

```
procedure matsort1(x,n,k,f)result:(y, t);
    value n,k; integer n,k; array x,y; integer array t; integer procedure f;
begin real s; integer i,i1,j,m,p,q;
    for i:=k-1 step-1 until 0 do t[i]:=0;
    for i:=1 step 1 until n do
      begin j:=f(x[i]); t[j]:=t[j]+1 end;
    for i:=k-2 step-1 until 0 do t[i]:=t[i]+t[i+1];
    for i:=1 step 1 until n do
      begin j:=f(x[i]);
        y[n+1-t[j]]:=x[i]; t[j]:=t[j]-1
      end i;
    comment Далее производится сортировка внутри каждого класса, если число
    элементов в нем больше 1;
    q:=0;
    for i:=0 step 1 until k-1 do
      begin p:=q+1; q:=n-t[i];
        comment p — номер первого, а q — номер последнего элемента класса i
        в массиве y, причем, если в классе нет ни одного элемента, то p>q, если
        в классе один элемент, то p=q, если же более одного элемента, то p<q;
        if q>p then
```

```
        for j:=p step 1 until q—1 do
          begin i1:=0; s:=y[j];
            for m:=j+1 step 1 until q do
              if s>y[m] then begin i1:=m; s:=y[m] end;
              if i1≠0 then begin y[i1]:=y[j]; y[j]:=s end
          end j
      end i
end matsort;
```

Для упорядочения массива по убыванию достаточно в условии **if** s>y[m] **then** заменить знак > на знак < в соответствующим выборов функции *f**.

В цитированной работе Д. Кнута приведены оценки времени работы и требуемой памяти, из которых следует, что этот алгоритм имеет смысл применять при $n \geqslant 1000$.

Замечание к алгоритму 32б

Ю. Д. Красильников, Ю. А. Старостин, Москва, февраль 1979

Несмотря на то, что алгоритм 32б, а также соответствующие ему алгоритмы 32а и 32 были подтверждены как в «CACM», так и в выпусках «Библиотеки алгоритмов», он имеет некоторые недостатки, основной из которых — излишне усложненная логическая организация программы. Действительно, алгоритм 32б содержит четыре условных и один безусловный переход и оператор цикла, при программировании которого также используются команды безусловного и условного переходов. В то же время возможно перепрограммировать алгоритм так, чтобы его модифицированная версия содержала всего четыре условных перехода, при этом эффективность алгоритма не ухудшилась, его текст несколько сократился (и соответственно сократилась длина результирующей программы в машинном коде), а логическая структура программы стала более простой и наглядной.

Ниже приведен текст модифицированной версии, алгоритма 32б — процедуры *multint*1, а также ее рекурсивного аналога — процедуры *multintr*. Параметры этих процедур совпадают с параметрами процедуры *multint* алгоритма 32б с единственным исключением — формальный массив s специфицирован как имеющий тип **integer**, а не **real**, так как интервал интегрирования может быть поделен только на целое число подынтервалов. Кроме того, если массив s будет иметь тип **real**, то вследствие приближенного представления в ЭВМ действительных чисел теоретически возможно неверное сравнение переменных с индексами s[j] и h[j], что приведет к неверному вычислению интеграла.

```
real procedure multint1(n,aa,bb,ff,s,p,u,w);
      value n,p; integer n,p; array u,w;
      integer array s;
      real procedure aa,bb,ff;
begin real recurint; integer j; array a,b,c,d,r,x[1:n];
      integer array k,h[1:n];
      j:=1;
entry: a[j]:=aa(j,x); b[j]:=bb(j,x);
      d[j]:=(b[j]—a[j])/s[j];
      r[j]:=0.0; h[j]:=1;
looph: c[j]:=a[j]+(h[j]—0.5)×d[j];
      k[j]:=1;
loopk: x[j]:=c[j]+u[k[j]]×0.5×d[j];
      if j≠n then
        begin j:=j+1; go to entry end;
      recurint:=1.0;
exit: r[j]:=r[j]+recurint×ff(j,x)×w[k[j]];
      k[j]:=k[j]+1;
      if k[j]≤p then go to loopk;
      h[j]:=h[j]+1;
      if h[j]≤s[j] then go to looph;
      recurint:=r[j]×0.5×d[j];
    if j≠1 then
      begin j:=j—1; go to exit end;
      multint1:=recurint
end multint1;
```

* В подтверждение приведенной здесь процедуры Ю. Б. Гендельман представил листинги проведенных им расчетов. (*Прим. ред.*)

```
real procedure multintr(n,aa,bb,ff,s,p,u,w);
      value n,p; integer n,p; array u,w;
      integer array s;
      real procedure aa,bb,ff;
begin array x[1:n];
      real procedure recurint(j);
        value j; integer j;
        begin real a,b,c,d,r; integer k;
          a:=aa(j,x); b:=bb(j,x);
          d:=(b—a)/s[j];
          r:=0.0;
          for c:=a+0.5×d step d until b do
            for k:=1 step 1 until p do
              begin x[j]:=c+u[k]×0.5×d;
                r:=r+(if j=n then 1.0 else
                  recurint(j+1))×ff(j,x)×w[k]
              end;
          recurint:=r×0.5×d
        end recurint;
      multintr:=recurint(1)
end multintr;
```

Эти процедуры, а также исходная процедура multint алгоритма 32б были транслированы с использованием АЛГОЛ-транслятора в мониторной системе «Дубна» на ЭВМ БЭСМ-6. С их помощью был вычислен второй из интегралов, приведенных в «Подтверждении к алгоритму 32» Г. Тачера («Библиотека алгоритмов 1б—50б»), с учетом исправлений согласно «Замечаниям к алгоритму 32» и к «Подтверждению к алгоритму 32» К. Кельбига («Библиотека алгоритмов 1б—50б»). Все три процедуры дали результаты, совпадающие с приведенными в «Подтверждении к алгоритму 32а» В. Е. Цейтлина («Библиотека алгоритмов 1б—50б») не менее чем в восьми десятичных цифрах, за исключением значения интеграла при $p=3$, $k=2$, $s=2$. Для этого случая В. Е. Цейтлин приводит значение 1.129425434, в то время как нами получен результат 1.109429434, что совпадает с данными, приведенными Г. Тачером и К. Кёльбигом. Причиной этого несовпадения результатов стал, очевидно, сбой устройства вывода при вычислении В. Е. Цейтлиным значения интеграла на ЭВМ.

Для сравнительной оценки эффективности модификаций алгоритма с их помощью вычислялся восьмикратный интеграл

$$\int_0^2\int_0^2\int_0^2\int_0^2\int_0^2\int_0^2\int_0^2\int_0^2 dx_1dx_2dx_3dx_4dx_5dx_6dx_7dx_8 = 2^8 = 256$$

при $p=2$, $s=2$. Вид подынтегральной функции *ff* и пределов интегрирования *aa* и *bb* был выбран возможно более простым с тем, чтобы время, затрачиваемое на их вычисление, было минимальным. Все три процедуры вычислили верное значение 256.0. Время, затраченное на вычисление, приведено в табл. 48. В ней также указаны время трансляции процедур и их длина после трансляции в машинных словах.

Таблица 48

Процедура	Время трансляции, с	Время счета, с	Длина (в машинных словах)
multint	1.84	38.58	110
multint1	1.30	34.83	88
multintr	1.40	26.88	66

Приведенные данные позволяют сделать вывод, что модификации алгоритма 32б *multint*1 и *multintr* по сравнению с исходной процедурой *multint* имеют более короткий текст, порождают более короткую результирующую программу, их логическая структура более проста и наглядна, а эффективность несколько выше, чем у исходного алгоритма. (Следует, однако, отметить, что в случае, когда подынтегральная функция имеет достаточно сложный вид, время работы всех трех процедур будет практически одинаковым, так как основная часть этого времени будет затрачена именно на вычисление подынтегральной функции.) Поэтому использование вместо процедуры *multint*

ее модифицированной версии *multint*1, а в случае, если транслятор допускает рекурсивные процедуры и *multintr*, представляется более предпочтительным.

Подтверждение к алгоритмам 40б, 51б, 53б, 58б, 66б, 112б, 116б—118б, 120б и 126б

В. В. Варивода, Ленинград, февраль 1980

Перечисленные в заголовке алгоритмы используются в нашей лаборатории наиболее часто. Все они проверялись на машине БЭСМ-6; кроме того, алгоритмы 40б, 51б, 53б, 58б, 116б и 118б были проверены на машине ЕС-1050 и показали высокую эффективность. Некоторые из них переведены нами на ФОРТРАН-IV.

Алгоритм 53б был переведен также и на язык ПЛ-1 для системы ДОС ЕС. По этому алгоритму были получены, в частности, следующие правильные результаты.

1. Для $\sqrt[5]{-8+6i}$ получено

$$1.391165+0.7593070i, \quad -0.6791691-1.431996i,$$
$$-0.2922487+1.557715i, \quad 1.152035-1.088440i.$$
$$-1.571785+0.2034159i,$$

2. Для $\sqrt[10]{3+7i}$ получено

$$1.216782+0.1425113i, \quad -1.216783-0.1425072i,$$
$$0.9006320+0.8305005i, \quad -0.9006349-0.8304973i,$$
$$0.2404715+1.201266i, \quad -0.2404757-1.201265i,$$
$$-0.5115405+1.113191i, \quad 0.5115367-1.113193i,$$
$$-1.068162+0.5999154i, \quad 1.068160-0.5999191i.$$

[В подтверждение этих результатов т. Варивода прислал листинг соответствующих расчетов. (*Прим. ред.*)]

Пользуюсь случаем выразить нашу большую признательность всему авторскому коллективу за выпуск «Библиотеки алгоритмов».

Замечания к алгоритмам 50CJ, 57б, 126б, 143б и 145б

М. И. Агеев, Москва, декабрь 1978

В материалах второго выпуска «Библиотека алгоритмов 51б—100б» [48] нужно исправить следующее.

На с. 116, 17-я строка сверху напечатано

Белые: Крe1, Фе8, Лh1, Сс1, Кg2, Кh6.

Должно быть (эта поправка предложена В. М. Агеевым, г. Москва)

Белые: Крe1, Фе8, Лh1, Сс1, Кg2, Кh6, d2, f7.

В материалах третьего выпуска «Библиотека алгоритмов 101б—150б» [49] нужно сделать следующие исправления.

1. На с. 46, 14-я строка снизу напечано

ритма 126 (Counts J. W. «CACM», 1962, № 7).

Должно быть

ритма 126 (Counts J. W. «CACM», 1962, № 10).

2. На с. 73, 16-я строка снизу напечатано

simpson (f, a, b—a, f(a), 4×f((a+b)/2, f(b), absarea, 1, eps)

Должно быть

simpson (f, a, b—a, f(a), 4×f((a+b)/2)/2, f(b), absarea, 1, eps)

3. На с. 96, 21-я строка снизу напечатано

Черные: Крe6, аб.

Должно быть (эта поправка предложена В. Н. Антонцевым, г. Москва)

Черные: Крe6, Ла8, аб.

4. На с. 98, 8-я строка снизу напечатано

begin output ('/', 'Т', 'РАССМОТРЕНЫ ХОДЫ ФИГУРОЙ',

Должно быть

begin output ('/', 'Т', 'РАССМОТРЕНЫ—ХОДЫ—ФИГУРОЙ',

5. На с. 117, 14-я строка сверху напечатано

143а. Сортировка с помощью графа.

Должно быть

143б. Сортировка с помощью графа.

6. На с. 120, 6-я строка сверху напечатано

37б. Функции Томсона ber и bei.

Должно быть

57б. Функции Томсона ber и bei.

7. На с. 125, 22-я строка сверху напечатано

98. АЛГАМС ДОС ЕС ЭВМ. М., «Статистика», 1977. Авт.: Бородич и др.

Должно быть

87. АЛГАМС ДОС ЕС ЭВМ. М., «Статистика», 1977. Авт.: Бородич и др.

Подтверждение к алгоритму 50CJ

В. Н. Антонцев, Москва, май 1979

Алгоритм 50CJ [48] с некоторыми изменениями был переведен на автокод БЭМШ [100] для машины БЭСМ-6 *. Длина программы (без таблиц и констант) составила 243 слова. В программе были реализованы все ходы фигур, кроме рокировки. Алгоритм был расширен для решения задач с числом ходов до пяти. Были решены задачи № 1—20 на мат в два и три хода, приведенные в выпусках «Библиотеки алгоритмов» [48] и [49], а также задачи № 839—842 [49, с. 97] на мат в четыре хода.

Время исследования задач (т. е. полного перебора ходов) с помощью составленной программы (автокод) и программы, опубликованной в выпусках [48] и [49] (АЛГОЛ), приведено в табл. 49.

Таблица 49

Номер задачи	Время		Номер задачи	Время	
	Автокод	АЛГОЛ		Автокод	АЛГОЛ
1	30″	38″	11	1′15″	9′28″
2	15″	45″	12	4′57″	20′11″
3	32″	2′21″	13	2′41″	9′36″
4	45″	3′29″	14	4′09″	19′32″
5	10″	35″	15	1′07″	8′04″
6	23″	3′22″	16	4′02″	19′07″
7	52″	1′59″	17	3′04″	23′11″
8	22″	56″	18	8′29″	1^{h}37′07″
9	17″	4′21″	19	1′44″	13′56″
10	12″	31″	20	2′56″	19′35″
Среднее значение	26″	1′53″	Среднее значение	3′36″	23′29″

При решении задачи № 18 программа нашла 9 решений. Как выяснилось, это произошло из-за опечатки в выпуске алгоритмов [49, с. 96]: в задаче № 18 не была указана черная ладья на поле a8. Как видно из табл. 49 среднее время решения задач на автокоде в 4—7 раз меньше, чем среднее время их решения на языке АЛГОЛ-60.

Для четырехходовых задач № 839, № 840, № 841 время исследования было 6′48″, 24′15″, 19′38″ соответственно. Программа на языке АЛГОЛ-60 исследовала эти задачи за время 12′00″, 5^{h}58′28″ и 2^{h}36′56″ соответственно [49, с. 97].

При исследовании задачи № 842 [49, с. 96] с помощью составленной программы было обнаружено, что эта задача имеет побочное решение Лc3 : d3.

Подтверждение и замечание к алгоритму 60б

Ю. В. Гендельман, Волгоград, ноябрь 1979

Алгоритм 60б был переведен в автокод БМ-4/220 для машины БЭСМ-4 и использовался как подпрограмма в различных задачах. При $k=5$ (что обеспечивало абсолютную погрешность примерно 10^{-6}) время работы алгоритма в 5—10 раз мень-

* О переводе алгоритма 50CJ также и на ФОРТРАН сообщается в статье Р. Х. Эренштейна [104]. (*Прим. ред.*)

n	k	$J_1 = -J_2$	Абсолютная погрешность J_1	Абсолютная погрешность J_2	„Условная" погрешность
12	1	0.57076847	0.31		0.85
	2	0.30614627	0.41×10^{-1}		0.25
	3	0.26671003	0.12×10^{-2}		0.39×10^{-2}
	4	0.26556334	0.40×10^{-5}		0.11×10^{-2}
	5	0.26555932	0.72×10^{-9}	0.74×10^{-9}	0.40×10^{-5}
	„точное"	0.26555932	—	—	—
—1	1	19.641127	14.9		26.5
	2	10.656933	6.0		8.4
	5	4.9017647	0.20		0.56
	8	4.7007182	0.24×10^{-3}		0.39×10^{-2}
	9	4.7004866	0.62×10^{-5}		0.23×10^{-3}
	10	4.7004804	0.67×10^{-7}	0.66×10^{-7}	0.62×10^{-5}
	„точное"	4.7004803	—	—	—
—5	1	18.166666×10^{8}	18×10^{8}		27×10^{8}
	2	8.4777799×10^{8}	8.2×10^{8}		9.1×10^{8}
	5	1.0408651×10^{8}	0.79×10^{8}		1.0×10^{8}
	10	0.25001312×10^{8}	0.13×10^{4}		0.57×10^{5}
	11	0.25000010×10^{8}	0.10×10^{2}		0.13×10^{4}
	12	0.24999999×10^{8}	0.36×10^{-1}	0.65×10^{-1}	0.10×10^{2}
	„точное"	0.24999999×10^{8}	—	—	—

ше, чем при использовании стандартной программы интегрирования методом Симпсона с автоматическим выбором шага. Кроме того, память, занимаемая подпрограммой по Ромбергу, намного меньше, чем память, требуемая для стандартной программы по Симпсону. Результат вычисления интеграла $\int_2^5 \frac{x^3dx}{(4+x^2)^3}$ при $k=2$ в точности совпал с приведенным в «Свидетельстве к алгоритму 60а».

Ввиду того, что выбор величины k во многих случаях затруднителен, предлагается модифицировать этот алгоритм, введя в него оценку абсолютной погрешности, рекомендуемую в работе Н. С. Бахвалова [101, с. 178]. Вычисление интеграла прекращается, если $\min_j |t_{j+1}^{(i)} - t_j^{(i-1)}| < \varepsilon/d$, где $t_j^{(i)}$ — значение $t[j]$, полученное после i-го выполнения внешнего цикла в теле процедуры. Модифицированная таким образом процедура может иметь следующий вид.

```
real procedure rombint1 (f,a,b,k,eps);
      value a,b,k,eps; real a,b,eps; integer k;
      real procedure f;
begin real d,s,h,v,eps1; integer m,i,j,n; Boolean u;
      array t[1:k+1];
      d:=b—a; t[1]:=0.5×(f(a)+f(b));
      n:=1; eps1:=eps/d; u:= false;
      for i:=1 step 1 until k do
        begin s:=0; n:=n+n; h:=d/n;
          for j:=1 step 2 until n do s:=s+f(a+j×h);
          t[i+1]:=s/n+0.5×t[i]; m:=1;
          for j:=i step —1 until 1 do
            begin m:=4×m; v:=t[j+1]—t[j];
              if abs(v)<eps1 then u:= true;
              t[j]:=t[j+1]+v/(m—1)
            end j;
          if u then go to fin
```

```
        end i;
fin:  rombint1:=t[1]×d
end rombint1;
```

В этой процедуре k — максимально допустимое число итераций, а eps — абсолютная погрешность.

В табл. 50 приведены результаты работы процедуры на следующих примерах, приведенных Тачером в его подтверждении к алгоритму 60,

$$J_1 = \int_{0.01}^{1.1} x^n dx \text{ и } J_2 = \int_{1.1}^{0.01} x^n dx.$$

В табл. 50 «точные решения» получены на ЭВМ по аналитическим формулам. «Условные» погрешности оценивались по рекомендуемой здесь методике, причем для их получения процедура *rombint1* была дополнена операторами, запоминающими и печатающими для каждой итерации значения $t[1]\times d$ и $\min abs(v)$. При этом для $n=0$ при любом k (даже при $k=0$) получалось точное решение 1.09.

Из приведенных в табл. 50 данных* следует, что начиная с некоторого k, «условные» погрешности приблизительно совпадают с точными погрешностями, определенными на предыдущем шаге. Следовательно, задача выбора k_{max} полностью не снимается. Но, как показали практические расчеты, даже при тщательном выборе k_{max} в 5—10% случаев фактически выполненное число итераций меньше k_{max}.

По вопросу применения процедуры для подынтегральных функций с особенностями заметим, что хотя по словам Тачера «... влияние близости особой точки ... может быть преодолено», все же в практических расчетах лучше пользоваться процедурами с автоматическим выбором шага или, по крайней мере, разбивать интервал интегрирования, выделяя особую точку (хотя это достаточно очевидно). Вышеуказанную фразу Тачера не следует рассматривать как практический совет.

И, наконец, оценка погрешности вида $d\times|t_1^{(i)}-t_1^{(i-1)}|$ оказалась, как и следовало ожидать, заниженной.

Подтверждение к алгоритмам 63б, 64б и 65б

А. М. Букреев, Москва, ноябрь 1978

После исправления процедуры *exchange* (исключения из нее списка значений) алгоритмы 63б—65б были транслированы на машине БЭСМ-6 в системе АЛГОЛ-ГДР (версия от 6.6.75) и работали правильно.

Недостаток алгоритма 64б — неизвестная глубина рекурсии, которая может оказаться весьма значительной. Глубина рекурсии нижеследующей, предлагаемой мною процедуры quicksort 1 ** не превышает $\log_2(2\times(n-m+2)/3)$.

```
procedure quicksort1(m,n)dataresult:(a);
      value m,n; integer m,n; array a;
begin integer i,j;
iter:  if m<n then
         begin partition(m,n,a,i,j);
           if i-m<n-j then
             begin quicksort1(m,i,a); m:=j end else
               begin quicksort1(j,n,a); n:=i end;
           go to iter
         end
end quicksort1;
```

Поскольку глубина рекурсии в этом варианте процедуры незначительна, то можно написать нерекурсивный вариант алгоритма 64б, организовав запоминание необходимых данных в массивах. Предлагаемый вариант процедуры приводится ниже:

```
procedure quicksort2(m,n)dataresult:(a);
      value m,n; integer m,n; array a;
begin integer i,j,ij;
      array m1,n1[1:ln(2×(n-m+2)/3)/ln(2)];
      ij:=0;
iter: if m<n then
        begin partition(m,n,a,i,j); ij:=ij+1;
          if i-m<n-j then
```

---

* В подтверждение проведенных расчетов Ю. В. Гендельман представил листинги соответствующих расчетов. (*Прим. ред.*)

** В подтверждение этой и нижеследующих процедур тов. А. М. Букреев представил листинги с результатами проверки этих процедур на машине. (*Прим. ред.*)

```
            begin m1[ij]:=j; n1[ij]:=n; n:=i end else
            begin m1[ij]:=m; n1[ij]:=i; m:=j end;
          go to iter
        end;
      m:=m1[ij]; n:=n1[ij]; ij:=ij—1;
      if ij≥0 then go to iter
end quicksort2;
```

Далее приведен предлагаемый мною нерекурсивный вариант процедуры *find* алгоритма 65б:

```
procedure  find1(m,n,k)dataresult:(a);
      value m,n,k; integer m,n,k; array a;
begin integer i,j;
iter: if m<n then
        begin partition(m,n,a,i,j);
          if k≤i then n:=i else
            if k≥j then m:=j else go to final;
          go to iter
        end;
final: end find 1;
```

Замечания к алгоритмам 115б, 135б, 137б, 139б, 141б, 143б и 149б

Г. С. Рашкович, Одесса, февраль 1979

Прежде всего хочется выразить большую признательность всему авторскому коллективу за подготовку выпусков «Библиотека алгоритмов» и особенно за непрерывную работу по их улучшению и выявлению ошибок, опечаток и неточностей. Выпуски «Библиотеки» пользуются огромной популярностью и быстро раскупаются.

Желая быть полезным в дальнейшем улучшении алгоритмов «Библиотеки», предлагаю несколько исправлений опечаток, перечисленных в табл. 51.

Таблица 51

Номер алгоритма	Страница	Строка	Напечатано	Должно быть
115б	21	19 св.	k : =0;	k : =0;
135б	56	18 сн.	if m<then 1	if m<1 then 1
135б	59	3 сн.	value s, f; integer s, f, ex; array factors;	value s, f; integer s, f, ex; array factors;
137б	65	2 си.	begin integer j;	begin integer j;
139б	67	6 сн.	... end esle	... end else
141б	71	6 св.	if a [i, j] ∧ ...	if a [i, j]∧ ...
143б	72	3 св.	... end esle	... end else
149б	78	17 св.	... then 1.57079633/a else	... then 1.57079633/a else

Кроме того, в процедуре *linearsystem* алгоритма 135б ([49], с. 57, 22-я строка сверху) описания типов, находящиеся в начале оператора цикла по *k*

```
for k:=1 step 1 until m do
  begin real normy,kr; integer count,limit;
```

имеет смысл перенести в начало процедуры. Это позволит избавиться от *m*-кратного выполнения машинных операций, связанных с выделением памяти под эти переменные.

Далее в «Дополнительных сведениях об алгоритмах 108 и 109», в «Свидетельстве к алгоритму 110б» и в «Свидетельстве к алгоритму 111б» [49, с. 16—17] говорится о языке АЛГОЛ-58. Желательно указать, что это за язык и где он описан *.

---

* Описание языка АЛГОЛ-58 в русском переводе опубликовано в [97], где он именовался как АЛГОЛ. Название АЛГОЛ-58 стало употребляться специалистами позже для отличия его от языка АЛГОЛ-60. Язык АЛГОЛ-58 не получил широкого распространения и в настоящее время почти забыт, однако до появления языка АЛГОЛ-60 на языке АЛГОЛ-58 был опубликован ряд алгоритмов, в частности алгоритмы 1—7 журнала «САСМ». (*Прим. ред.*)

Замечания к алгоритмам 121а и 133б

Ю. С. Вагин, В. П. Логвиненко, Москва, ФИАН СССР, февраль 1981

Алгоритмы 121а и 133б были переведены на язык ФОРТРАН 5 и проверены на машине ECLIPSE. Оказалось, что алгоритм 133б имеет существенный недостаток — наличие областей корреляции между близко расположенными числами в генерируемой последовательности. В то же время алгоритм 121а, не включенный в выпуск «Библиотека алгоритмов 101б—150б», поскольку он не был подтвержден в расчетах авторов выпуска, дает вполне удовлетворительные результаты, если использовать в нем для генерации псевдослучайных чисел не алгоритм 133б, а модифицированный алгоритм 133m.

1. *Корреляция случайных чисел в последовательности, выдаваемой алгоритмом 133б.* На рис. 16 изображен график функции $y=(5\times x)\bmod 2^{35}$, используемой в алгоритме 133б. На этом графике последовательности псевдослучайных чисел (ППСЧ) соответствуют точки, имеющие целочисленные нечетные значения x и y. Процесс перехода от числа x_n к следующему числу x_{n+1} представляется как движение с прямой $y_n=(5\times x_n)\bmod 2^{35}$ на прямую $y_n=x_{n+1}$ с тем же значением ординаты, а затем возврат назад с тем же значением абсциссы. Видно, что для выхода из углов квадрата требуется несколько проходов. Так, для $x_0=2^{35}-1$ получим $x_1=2^{35}-5$; $x_2=2^{35}-25$; ... Эта подпоследовательность заканчивается на k-м проходе. Значение k можно оценить следующим образом: $x_k=2^{35}-5^k\approx(4/5)2^{35}$, $k\approx 35\log_5 2-1\approx 14$.

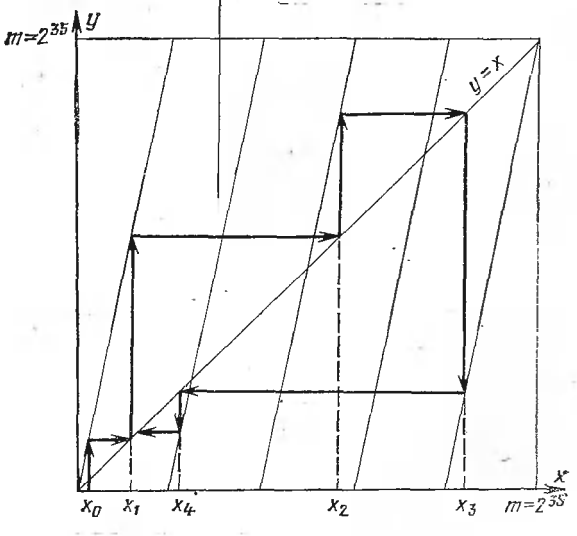

Рис. 16.

Таким образом, для $x_0=1$ и $x_0=2^{35}-1$ выявляется подпоследовательность неслучайных чисел (ПНЧ) из 15 элементов. Для значений $x_0=3$ и $x_0=2^{35}-3$ имеются аналогичные ПНЧ, состоящие из 14 элементов. Такие ПНЧ существуют в пределах областей (1, 1374389531) и ($2^{35}-1$, $2^{35}-1374389531$), число элементов в них убывает от 15 до 2. Кроме того, внутри интервала $(0, 2^{35})$ вокруг точек $x_0=2^{33}$, 2^{34}, 3×2^{33} также находятся области ПНЧ. Например, для $x_0=2^{34}+1$ образуется ПНЧ из 14 элементов: $x_1=2^{34}+5$, $x_2=2^{34}+25$, ...

Помимо ПНЧ, локализованных на участке одной из наклонных прямых графика (см. рис. 16), существуют подпоследовательности, связывающие две или больше наклонных прямых. Так, при $x_0=1431655765$ получаем $x_1=7158278825$, $x_2=1431655757=x_0-8$, ... Таким образом, ППСЧ, вырабатываемая алгоритмом 133б, разбивается на ряд областей ПНЧ. Анализ показывает, что области ПНЧ в сумме перекрывают большую часть интервала $(0, 2^{35})$. При вычислении средних значений x, x^2, вообще x^α, эту особенность заметить невозможно, так как вычисление средних позволяет судить только о равномерности заполнения случайными числами выбранного интервала, но не дает сведений о корреляции. Корреляция между соседними числами может быть выявлена, например, при вычислении средних значений $x_n\times x_{n+1}$, $x_n\times x_{n+1}\times x_{n+2}$, $(x_{n+1}-x_n)^2$. Следует заметить, что корреляция соседних чисел и равномерность заполнения выбранного интервала являются независимыми характеристиками ППСЧ, и для их контроля необходимо использовать разные тесты.

2. *Модификация алгоритма 133б.* Приведенные выше соображения показывают, что для уменьшения длины ПНЧ и размеров областей неслучайности необходимо увеличить число a в общей формуле для линейной конгруэнтной последовательности $x_{n+1}=(a\times x_n+c)\bmod m$, частным случаем которой является алгоритм 133б. В [107] указывается, что «множитель a должен превосходить величину $\sqrt{m}$, желательно, чтобы он был больше $m\times 10^{-2}$, но меньше $m-\sqrt{m}$. Последовательность разрядов в двоичном или десятичном представлении a не должна иметь простого, регулярного вида».

Однако, практически реализовать алгоритм с большим a не совсем удобно, так как произведение $a \times x$ выходит за пределы допустимого числа разрядов ЭВМ. Приходится вычислять произведение по частям, это увеличивает время счета.

Для модифицированного алгоритма 133m нами были выбраны значения $a=5^5$, $c=0$ и $m=2^{37}$, что соответствует 15 десятичным разрядам для произведения $a \times x$. Фактически этот алгоритм является просто пятикратным повторением первоначального алгоритма 133б (с поправкой на разные числа m в них). Числа в последовательности, выдаваемой модифицированным алгоритмом, являются выборкой каждого пятого числа из ППСЧ, производимой прежним алгоритмом (с той же поправкой). Длина ПНЧ при этом также уменьшается в 5 раз, а размер областей неслучайности — в $1/4 5^8$ раза (если бы в модифицированном алгоритме стояло $m=2^{35}$, то уменьшение составило бы 5^8). Значение $a=5^5$ не удовлетворяет требованию $a > \sqrt{m}$, такая модификация не ликвидирует полностью корреляцию соседних чисел в производимой ППСЧ, однако при проверке алгоритма результаты получились вполне удовлетворительные.

Кроме того, был испытан усложненный алгоритм 133у, в котором были выполнены требования, рекомендованные в [107]. Усложненный алгоритм имеет вид

$$x_{n+1} = (5^9 \times x_n + 29044268343) \bmod 2^{37}.$$

Значение c выбиралось из условия $c \approx m(1/2 - \sqrt{3}/6)$ [107].

3. Результаты проверки алгоритмов. Все три алгоритма были переведены на язык ФОРТРАН 5 и реализованы на машине ECLIPSE (быстродействие — около 1 млн. опер./с). Для проверки алгоритмов вычислялись средние значения

$$\frac{1}{N}\Sigma R_n,\ \frac{1}{N}\Sigma R^2_n,\ \frac{1}{N}\Sigma R_n \times R_{n+1},\ \frac{1}{N}\Sigma R_n \times R_{n+1} \times R_{n+2},$$

где $R_n = x_n \times m^{-1}$. Результаты представлены в табл. 52.

Запись алгоритма 133m на ФОРТРАНе имеет следующий вид:

```
      SUBROUTINE URAN37(R)
C     RANDOM UNIFORM DISTRIBUTION
      COMMON X
      REAL R
      DOUBLE PRECISION X,DM37
      DM37=137438953472
      X=X*3125
      X=X—IDINT(X/DM37)*DM37
      R=X/DM37
      RETURN
      END
```

Таблица 52

Алгоритм	Число испытаний N	$\frac{1}{N}\sum R_n$	$\frac{1}{N}\sum R^2_n$	$\frac{1}{N}\sum R_n \times R_{n+1}$	$\frac{1}{N}\sum R_n \times R_{n+1} \times R_{n+2}$	Время счета, с
133б	700	.5236910	.3508849	.2856995	.1567200	
	30 000	.5003107	.3334049	.2662969	.1430143	19
133m	700	.5225139	.3548063	.2718254	.1385781	
	30 000	.5000737	.3341070	.2501774	.1255972	20
133у	700	.4866381	.3207265	.2402948	.1181728	
	30 000	.4959531	.3295804	.2467715	.1227638	23
Теоретическое значение		.5	.3333333	.25	.125	

Запись алгоритма 133у отличается от приведенной двумя строчками, добавленными после 8-й строки алгоритма 133m:

$$X=X*625+29044268343$$

$$X=X-IDINT(X/DM37)*DM37$$

Вычисление алгоритма 133у последовательно в два действия было вызвано недостатком разрядов ЭВМ для произведения $5^9\times2^{37}$. Для алгоритма 133б задавалось начальное значение $x_0=13000000001$, для алгоритмов 133m и 133у выбиралось $x_0=130000000001$.

Усреднения производились для общего числа испытаний $N=700$ и $N=30000$. Как видно из табл. 52, алгоритмы 133m и 133у во всех случаях дают почти одинаковые результаты, алгоритм 133б дает заметное отличие для $\frac{1}{N}\Sigma R_n\times R_{n+1}\times R_{n+2}$. Время счета алгоритмов 133б и 133m было почти одинаковое, время счета алгоритма 133у в 1.2 раза больше.

Сравнение результатов показывает, что все три алгоритма выдают ППСЧ с хорошим равномерным заполнением выбранного интервала (0,1), однако генерируемая алгоритмом 133б ППСЧ имеет заметную корреляцию между соседними числами. Алгоритм 133m является наиболее предпочтительным, так как имеет меньшее время счета по сравнению с алгоритмом 133у.

4. Замечание и подтверждение к алгоритму 121а. Особенно сильно корреляция соседних чисел ППСЧ будет заметна при разыгрывании неравномерно распределенной случайной величины методом Неймана [108]. В этом случае может быть получен просто неправильный результат. Именно это произошло при использовании в алгоритме 121а в качестве вспомогательного алгоритма 133б для получения псевдослучайных чисел, равномерно распределенных в интервале (0,1). Сам алгоритм 121а не является ошибочным, так как он состоит в применении метода Неймана для разыгрывания случайной величины ζ с известной плотностью распределения вероятности:

$$P(x)=(2^{-1}\times\pi)^{0.5}\exp(-2^{-1}\times x^2)$$

Рассмотрим работу алгоритма 121а. Для получения нормально распределенной случайной величины ζ используется случайная величина γ, равномерно распределенная в интервале (0,1). Сначала проводится сравнение γ_1 с числом $a=0.68268949$. При $\gamma_1<a$ принимают, что ζ лежит в интервале (0,1) и для ее разыгрывания используется обычный метод Неймана. Вычисляют два значения случайной величины γ: γ_2 и γ_3. Если $\gamma_2\leqslant\exp(-\gamma_3^2/2)$, то ζ присваивают значение $\zeta=\gamma_3$ и работа алгоритма окончена, в противном случае значения γ_2 и γ_3 отбрасываются, вычисляются новые значения γ_2, γ_3 и сравнение повторяется. При $\gamma_1>a$ принимают, что ζ лежит в интервале $(1,\infty)$ и метод Неймана используется для разыгрывания вспомогательной случайной величины η, расположенной в интервале (0,1) и связанной с ζ формулой

$$\zeta=F(\eta)=\sqrt{1-2\ln\eta}.$$

Случайная величина η имеет плотность распределения вероятности

$$q(x)=p[F(x)]\times\left|\frac{dF}{dx}\right|=\sqrt{2\times\pi^{-1}\times e^{-1}\times(1-2\ln x)^{-1}}.$$

Для разыгрывания η вычисляют два значения случайной величины γ: γ_4 и γ_5. Если $\gamma_4\leqslant(1-2\ln\gamma_5)^{-0.5}$, то ζ присваивают значение $\zeta=(1-2\ln\gamma_5)^{0.5}$, в противном случае значения γ_4, γ_5 отбрасываются, вычисляются новые значения и сравнение повторяется.

Из изложенного ясно, что алгоритм 121а позволяет проводить разыгрывание случайной величины ζ, распределенной нормально в интервале $(0,\infty)$, и ошибки в работе алгоритма могут быть связаны только с несовершенством исходного генератора случайных чисел, дающего равномерное распределение в интервале (0,1).

Для проверки алгоритма 121а были проведены его испытания с использованием различных алгоритмов: 133б, 133m, 133у. Алгоритм 121а был переведен на язык ФОРТРАН 5. Вычислялись средние значения $\frac{1}{N}\Sigma r_n$, $\frac{1}{N}\Sigma r^2_n$, $\frac{1}{N}\Sigma r_n\times r_{n+1}$ и $\frac{1}{N}\Sigma r_n\times r_{n+1}\times r_{n+2}$, где $r_n=\zeta_n$. Усреднения проводились для общего числа испытаний $N=700$ и $N=30\,000$. Результаты представлены в табл. 53. Сравнение показывает, что использование алгоритма 133б приводит к ошибочным результатам, а использование алгоритмов 133m и 133у дает во всех случаях почти одинаковые результаты. Время счета для алгоритмов 133б и 133m почти одинаковое, время счета для алгоритма

Алгоритм	Число испытаний N	$\frac{1}{N}\sum r_n$	$\frac{1}{N}\sum r^2_n$	$\frac{1}{N}\sum r_n \times r_{n+1}$	$\frac{1}{N}\sum r_n \times r_{n+1} \times r_{n+2}$	Время счета, с
133б	700	.7732376	.8530287	.6073838	.4799986	—
	30 000	.7451329	.8352420	.5558938	.4126467	34
133т	700	.8282247	1.0347480	.6959469	.5778156	
	30 000	.7947559	.9955303	.6349522	.5094135	36
133у	700	.8324369	1.0942520	.6853759	.5580371	
	30 000	.7980368	1.0032010	.6348202	.5013317	44
Теоретическое значение		.7978845	1.0	.6366196	.5079489	—

133у в 1.3 раза больше. На рис. 17 представлены те же результаты для N=30 000, показывающие частоту 0.05 попадания случайной величины ζ в избранный интервал. Интервал (0,1) был разбит на 20 равных частей. Видно, что для алгоритма 133б имеются области аномального малого или большого выпадания случайной величины ζ. Это объясняется ошибками при сравнении γ_2 и $\exp(-2^{-1}\times\gamma^2_3)$, где γ_2 и γ_3 — два соседних коррелирующих друг с другом случайных числа, выдаваемых алгоритмом 133б.

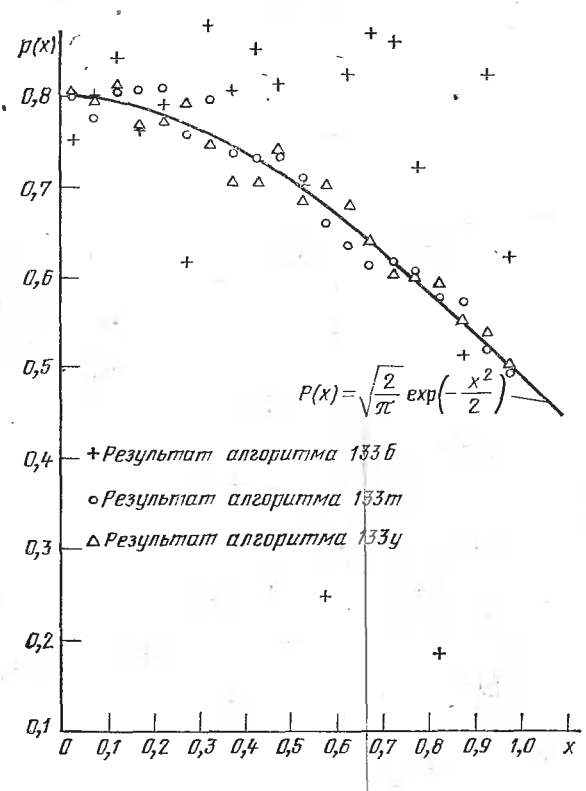

Рис. 17.

В заключение считаем нужным отметить, что выпускаемые сборники алгоритмов имеют неоценимое значение для работы программистов. Экономится много времени и сил на поиски требуемой программы, а публикация замечаний позволяет исправить в конце концов все неточности. Хотелось бы пожелать авторам выпусков успешного продолжения этой трудной работы по подготовке изданий.

Подтверждение к алгоритму 126б

Г. С. Рашкович, Одесса, март 1979

Алгоритм 126б был переведен на язык АЛМИР-65 для ЭВМ «МИР-1» и на входной язык клавишной программно-управляемой ЭВМ «Искра-125» и неоднократно использовался на этих машинах, а также в системе ТА-1М ЭВМ БЭСМ-4М для решения систем линейных уравнений. Как правило, решались довольно плохо обусловленные системы, возникающие при полиномиальной аппроксимации экспериментальных данных. Несмотря на принципиальные различия машинных арифметик, результаты решений на всех трех машинах различались не более чем в шестом знаке после запятой (расчеты проводились на БЭСМ-4М с обычной разрядностью — длина мантиссы 36 бит, а на ЭВМ «Мир» и «Искра» — с восемью десятичными знаками).

Подтверждение и замечание к алгоритмам 126б и 133б

А. М. Васильев, Хабаровск, НПО ДАЛЬСТАНДАРТ, июль 1979

Алгоритм 126б и алгоритм 133б были переведены на язык ПЛ-1 ДОС ЕС версии 2.1 и проверены на ЭВМ ЕС-1022 на примерах, приведенных вслед за описаниями этих алгоритмов.

Для алгоритма 126б результаты проверки полностью совпали с результатами, приведенными в «Свидетельстве к алгоритму 126а».

Для алгоритма 133б результаты вычислений полностью совпали с результатами, приведенными в замечании Д. Лауглина [49, с. 54] и с точностью до 8—9 цифр после десятичной точки — с результатами Дж. Пура [49, с. 54].

Однако для $x0=28395423107$ были получены результаты 0.5024063793, 0.4965200937, 0.5012649627 и 0.3414107437, 0.3286334174, 0.3335242210, отличающиеся от результатов, приведенных в пояснительном тексте алгоритма 133б, уже во второй — третьей цифрах.

Подтверждение и замечание к алгоритму 130б

С. А. Терещенко, Москва, ноябрь 1979

Было обнаружено, что при попытке получить все $4!=24$ перестановки из 4 различных элементов алгоритм 130б дает неверный результат — только 8 перестановок. Для исправления оператор

```
if b[k]>0∧b[k]<b[i] then a[k]:=b[k];
```

был заменен оператором

```
if b[k]>0∧b[k]<b[i] then
  begin a[k]:=b[i]; b[i]:=b[k] end;
```

После такого исправления алгоритм 130б был переведен на язык ФОРТРАН IV ЕС ЭВМ и дал правильный результат как для четырех различных элементов ($n=4$, $x=1$), так и для исходных данных, приведенных в описании алгоритма.

[Правильность этого замечания была проверена на машине и подтверждена составителями данного выпуска на многочисленных примерах для различных значений входных параметров. (*Прим. ред.*).]

Мне очень приятно, что моя скромная помощь, выразившаяся в данной заметке, пригодилась для Вашей исключительно полезной «Библиотеки алгоритмов». К сожалению, должен отметить недостаточность тиража издания подобного рода. Если третий выпуск «Библиотеки» еще можно кое-где найти, то первый выпуск стал уже библиографической редкостью.

Подтверждение к алгоритму 133б

В. К. Иванов, М. З. Слуцкий, Ю. Л. Титов, Днепропетровск, декабрь 1980

Алгоритм 133б был промоделирован на ЭВМ ЕС-1022 на языке ФОРТРАН-IV и показал удовлетворительные результаты *. В связи с ограниченностью разрядной сетки машины ЕС-1022 в алгоритм были внесены следующие изменения: $m35:=134217728$; $m36:=268435436$; $m37:=536870912$. Начальное значение x_0 принималось равным 526337. Результаты моделирования приведены в табл. 54.

---

* См. также «Замечания к алгоритмам 121а и 133б» Ю. С. Вагина и В. П. Логвинейко, помещенные выше. (*Прим. ред.*)

Количество значений	500	10 000	5000
Математическое ожидание M	0.478 344 977	0.487 551 749	0.501 203 120
Дисперсия D	0.083 403 230	0.079 990 327	0.082 671 523

Подтверждение к алгоритму 133б

Ю. В. Дементьев, Н. К. Шендрик, Новосибирск, декабрь 1980

Алгоритм 133б был преобразован к виду, пригодному для использования на ЭВМ, представление целых чисел в которых ограничено семью (или более) десятичными цифрами.

Каждое псевдослучайное целое число x, имеющее 12 десятичных разрядов, в измененном алгоритме представляется парой целых шестизначных чисел $x1$ и $x2$, связанных между собой соотношением $x=x1\times10^6+x2$. При первом обращении к процедуре *random*2 значения $x1$ и $x2$ должны находиться в интервалах $10\,000<x1\leqslant34\,359$; $0<x2<999999$, если $x1<34359$; $0<x2<738368$, если $x1=34359$, причем $x2$ должно быть нечетным.

Проверка процедуры *random*2, описание которой приведено ниже, производилась на машине М-222 с транслятором ТА-1М и прошла успешно. Были повторены все приведенные в справочном пособии [49] варианты расчетов. Результаты проверки совпали (в пределах ошибок округления) с приведенными в пособии во всех случаях, кроме варианта для машины FASIT EDB.

```
real procedure random2(a,b,x1,x2);
      value a,b; real a,b; integer x1, x2;
begin real r; integer d,p1,p2;
      p1 := 34359; p2 := 738368;
      r := (x1+x2×10—6)/(p1+p2×10—6);
      x1 := 5×x1; x2 := 5×x2; d := entier(5×r);
      x1 := x1—d×p1; x2 := x2—d×p2; d := entier(x2/10 6);
      x1 := x1+d; x2 := x2—d×10 6;
      random2 := r×(b—a)+a
end random2;
```

Замечания к алгоритмам 137б и 138б

А. А. Гинзбург, В. М. Губочкин, Ленинград, март 1979

Прежде всего следует заметить, что применение алгоритма 137б имеет смысл, по-видимому, только тогда, когда число вложений циклов заранее неизвестно. Непосредственный счет с использованием фиксированного числа вложений циклов происходит значительно быстрее.

Мы внесли в алгоритм 137б следующие изменения.

1. Для повышения универсальности алгоритма в список параметров процедуры *fors1* был включен целочисленный массив $v\,[1:n]$, в котором пользователем задаются нижние границы изменений параметров циклов.

2. Для экономии машинного времени в тело модифицированной процедуры *fors1m* была включена процедура *forsm* с одним формальным параметром *nloc*.

3. Рекурсия в процедуре *fors1m* ведется с увеличением *nloc*, в результате чего нумерацию границ параметров циклов пользователь будет теперь производить более естественным способом «сверху вниз», а не «снизу вверх», как это было в алгоритме 137б. Это снижает вероятность ошибки, которую пользователь может допустить при нумерации *.

Текст модифицированной таким образом процедуры приведен ниже.

---

* В частности, такого рода ошибка была допущена в «Свидетельстве к алгоритму 137б» [49, с. 66, 16-я строка сверху], где вместо $u[1]=3$, $u[2]=4$ должно быть $u[1]=4$, $u[2]=3$. Вследствие симметричности приведенного там примера эта ошибка не отразилась на правильности полученной матрицы. (*Прим. ред.*)

```
procedure fors1m(n,p,v,u,i);
      value n; integer n; integer array v,u,i; procedure p;
begin procedure forsm(nloc);
      value nloc; integer nloc;
      begin integer j;
        for j:=v[nloc] step 1 until u[nloc] do
          begin i[nloc]:=j;
            if nloc=n then p else forsm(nloc+1)
          end
      end forsm;
      forsm(1)
end   fors1m;
```

Скорость свертывания оператора цикла еще больше увеличится, если пользоваться для этого нерекурсивной процедурой *forsn*, приведениой ниже.

```
procedure  forsn(n,p,v,u,i);
      value n; integer n; integer array v,u,i;
      procedure p;
begin integer j;
      for j:=1 step 1 until n do i[j]:=v[j];
loop: p;
      for j:=n step —1 until 1 do
        if i[j]<u[j] then
          begin i[j]:=i[j]+1; go to loop end else i[j]:=v[j]
end forsn;
```

Аналогичные изменения можно сделать и в алгоритме 138б.

Процедуры *fors1* (с дополнительным параметром *v*), *fors1m* и *forsn* были проверены на машине БЭСМ-6 в системе ГДР — АЛГОЛ [99, с. 32—34]. Производился подсчет числа «счастливых» билетов с шестизначными номерами, т. е. свертыванию подвергался шестислойный оператор цикла

```
for  i1:=0  step  1  until  9  do
  for i2:=0 step 1 until 9 do
    for i3:=0 step 1 until 9 do
      for i4:=0 step 1 until 9 do
        for i5:=0 step 1 until 9 do
          for i6:=0 step 1 until 9 do
            if i1+i2+i3=i4+i5+i6 then n:=n+1;
```

Во всех случаях был получен правильный результат $n=55252$ *. Время счета без свертывания оператора цикла было равно 29 с, для процедуры *fors1* — 287 с, для *fors1m*—131 с и для *forsn*—123 с.

Замечания к алгоритму 145б

Л. А. Кнелер, Москва, май 1979

В тексте алгоритма 145б [49] замечены следующие опечатки.

1. На с. 73, 21-я строка сверху напечатано

```
if abs(est—sum)⩽eps×absarea∧est≠1)∨level⩾20 then
```

Должно быть

```
if (abs(est—sum)⩽eps×absarea∧est=1)∨level⩾20 then
```

2. На с. 73, 24-я строка сверху напечатано

```
+simpson(f,x1,dx,f2,fm,f3,absarea,est,eps/1.7)
```

Должно быть

```
+simpson(f,x1,dx,f2,fm,f3,absarea,est2,eps/1.7)
```

3. На с. 73, 30-я строка сверху напечатано

```
simpson(f,a,b—a,f(a),4×f((a+b)/2),f(b),absarea,1,eps)
```

Должно быть

```
simpson(f,a,b—a,f(a),4×f((a+b)/2),f(b),absarea,1,eps)
```

Алгоритм 145б был переведен на язык ПЛ-1 и после трансляции в системе ОС ЕС 4.0 дал правильный результат для примера, приведенного в «Подтверждении к алгоритму 145» В. М. Мак-Кимена [49, с. 74].

---

* В подтверждение этих данных авторами данного «Замечания» был представлен листинг соответствующих расчетов. (*Прим. ред.*)

Замечание к алгоритму 149

О. Сковгард (Skovgaard Ove. «TOMS», 1978, № 1)

Текст, следующий за двоеточием в конце пятого абзаца в замечании Г. Тачера [25i]*, должен иметь следующий вид: $K=a\times ELIP1(a,b)$ или $K=a\times ELIP2(a,b)$, где $m=k^2=1-(b/a)^2$.

Процедура более высокого качества опубликована в [22i, с. 86, *procedure cel1*]. Для некоторых машин эффективность этой процедуры может быть несколько повышена путем исключения последнего в цикле оператора присваивания $m:=m/2$, замены второго оператора присваивания $m:=kc+m$ на $m:=(kc+m)\times 0.5$ и замены последнего оператора присваивания $cel1:=pi/m$ на $cel1:=(pi/2)/m$. Следует заметить, что переменную m не нужно путать с параметром $m=k^2$.

Более эффективная, но менее компактная процедура приведена в [23i] и использована, например, в [24i].

Подтверждение к алгоритму 266

А. М. Васильев, Хабаровск, август 1980

Алгоритм 266 («САСМ», 1965, № 10) был переведен на язык ПЛ-1 ДОС ЕС версии 2.2 и успешно проверен на ЭВМ ЕС-1022. Результаты покер-теста для чисел, которые приведены вслед за описанием алгоритма 266, полностью совпали с результатами М. Пайка и И. Хилла. Кроме того, для тех же начальных значений y_1 были вычислены среднее значение и среднее значение квадрата величины y в интервале (0,1) для количества чисел N=500, 1000, 5000. Вычислялась также статистика $\frac{1}{N/100}\sum_{1}^{100}(N_k-N/100)^2$ по гистограмме, составленной при разбивке всего интервала на 100 равных отрезков (N_k — количество чисел, попавших в k-й отрезок) для проверки равномерности распределения по критерию χ^2. Результаты испытаний приведены в табл. 55.

Таблица 55

y_1	$\frac{1}{N}\sum_{1}^{N} y_n$	$\frac{1}{N}\sum_{1}^{N} y^2_n$	$\frac{1}{N/100}\sum_{1}^{100}(N_k-N/100)^2$	N
13421773	0.50713935	0.34871551	83.6	500
	0.50262788	0.34202247	80.8	1000
	0.49882091	0.33336589	105.6	5000
22369621	0.50343442	0.34114721	94.4	500
	0.49395354	0.32891638	79.4	1000
	0.49689849	0.33118059	94.1	5000
33554433	0.49769674	0.33111839	83.2	500
	0.49213939	0.32511541	98.0	1000
	0.49330454	0.32651454	87.7	5000
8426219	0.48298171	0.31112848	95.6	500
	0.49152076	0.32251831	94.2	1000
	0.50422396	0.33703290	92.3	5000
42758321	0.51057687	0.34527101	106.4	500
	0.50235966	0.34041476	99.0	1000
	0.49916112	0.33244271	103.3	5000
56237485	0.52515916	0.36063137	103.6	500
	0.51860225	0.35210391	97.4	1000
	0.50709660	0.33760251	86.2	5000
62104023	0.51424426	0.34909148	92.0	500
	0.49939685	0.33258051	82.8	1000
	0.50329809	0.33720839	98.3	5000

* В русском переводе замечания Г. Тачера [48, с. 12] этот текст (он находится там в третьем абзаце) был уже исправлен авторами выпуска. (*Прим. ред.*)

Замечание к алгоритму 394

Д. Маршалл (Marshall D. R. T. «CACM», 1972, № 12)

Слова «colums» и «row» в четвертом и пятом предложениях первого комментария процедуры *split* перепутаны между собой местами. Эти предложения должны читаться следующим образом: «If the input tables has no colums, then split returns zero, ... If the table is entirely dashes or has no rows, then split...».

В главной программе оператор обращения к процедуре *split* использует параметр *l*, значение которого не определено. Этому параметру нужно задавать значение, равное номеру первого столбца в подлежащем обработке массиве. Естественно это значение положить равным единице.

После вышеуказанных исправлений алгоритм был переведен на язык ПЛ-1 и работал правильно.

Подтверждение и замечания к алгоритму 394

Н. А. Селянко, Киев, май 1979

Предполагаются известными замечания Д. Маршалла («CACM», 1972, № 12) и Ю. Д. Красильникова [49, с. 91].

Данный алгоритм был реализован на языке АЛГОЛ-60 в системе ОС MFT версий 4.0 и 4.1 и MVT версии 4.1 на ЕС ЭВМ.

При этом из текста алгоритма были удалены переменная N и оператор N:=—1; как дублирующие и неиспользуемые, дополнительно введены отсутствовавшие во внешней программе описания типа **integer** и **integer array** для соответствующих фактических параметров и переменных, а во внутренних процедурах были добавлены спецификации таких же типов для соответствующих формальных параметров. Были также учтены следующие особенности данной версии языка ОС АЛГОЛ (уровня F).

1. Отсутствие реализации описателя **own**.
2. Употребление несколько отличной символики подмножества служебных знаков и знаков логических операций.
3. Использование управляющей процедуры SYSACT для редактирования протокола распечатки выходного символа *col* и избыточных правил в процедуре *split*.

Кроме того, следует отметить, что так как порядковые номера решающих правил в выходной матрице (элементы массивов *yes* и *no*) маркируются знаком «—», то для единообразия этим же знаком целесообразно отмечать и избыточные правила. Для этого перед оператором

```
outinteger (1, col[i]);
```

в процедуре *split* следует вставить оператор

```
col[i]:=—col[i];
```

и заключить оба в операторные скобки.

На базе данного алгоритма был разработан генератор с языка таблиц решений, характерной особенностью которого является использование динамической структуры вызываемых управляющей программой модулей (фазы идентификации и распределения памяти, загрузки исходного текста, генерации, редактирования и вывода) с целью высвобождения дополнительного объема оперативной памяти для размещения таблицы (а соответственно и выходной матрицы) максимально возможных размерностей.

Поскольку алгоритм 394 был реализован до выхода в свет «Библиотеки алгоритмов 101б—150б» [49], то последнее утверждение 4 в замечаниях Ю. Д. Красильникова [49, с. 92] представляется спорным.

Список литературы, которой пользовались составители выпуска

1. **Naur R. (ed.)** Revised Report on the Algorithmic Language ALGOL 60. — «Communication of the ACM», 1963, № 1. Русский перевод: Алгоритмический язык АЛГОЛ-60. Пер. с англ.— М.: Мир, 1965.
2. **Атман Р. Е.** Сообщение о сокращенном АЛГОЛе-60 (ИФИП). — Журн. вычисл. мат. и мат. физ., 1965, № 3.

3. **Шура-Бура М. Р., Любимский Э. З.** Транслятор АЛГОЛ-60.— Журн. вычисл. мат. и мат. физ., 1964, № 1.
4. **Агеев М. И., Алик В. П., Галис Р. М.** Алгоритмы (1—50). — М.: ВЦ АН СССР, 1966.
5. **Алгоритмы** (51—100)./ Агеев М. И., Алик В. П., Малюк Л. В., Марков Ю. И. — М.: ВЦ АН СССР, 1966.
6. **Атман Р. Е.** Сообщение о процедурах ввода—вывода в языке АЛГОЛ-60. — Журн. вычисл. мат. и мат. физ., 1964, № 5.
7. **Гельфонд Л. О.** Исчисление конечных разностей. — М.: Физматгиз, 1959.
8. **Янке Е., Эмде Ф., Леш Ф.** Специальные функции. — М.: Наука, 1964.
9. **Бронштейн И. Н., Семендяев К. А.** Справочник по математике для инженеров и учащихся ВТУЗов. — М.: Физматгиз, 1962.
10. **Толстов Г. П.** Ряды Фурье. — М.: Физматгиз, 1960.
11. **Голубев В. В.** Лекции по аналитической теории дифференциальных уравнений. — М.: Гостехиздат, 1950.
12. **Лаврентьев М. А., Шабат Б. В.** Методы теории функций комплексного переменного. — М.: Наука, 1965.
13. **Милн В. Э.** Численный анализ. — М.: ИЛ, 1951.
14. **Вентцель Е. С.** Теория вероятностей.— М.: Наука, 1964.
15. **Слейтер Л. Дж.** Вырожденные гипергеометрические функции. — М.: ВЦ АН СССР, 1966.
16. **Гончаров В. Л.** Теория интерполирования и приближения функций.— М.: Гостехиздат, 1954.
17. **Кокс Д., Смит У.** Теория очередей. — М.: Мир, 1966.
18. **Большая** советская энциклопедия. — 2-е изд. — М.: БСЭ, 1957. — Т. 49.
19. **Фаддеев Д. К., Фаддеева В. Н.** Вычислительные методы линейной алгебры.— М.: Физматгиз, 1963.
20. **Агеев М. И., Кривонос Л. С., Марков Ю. И.** Алгоритмы (101—150). — М.: ВЦ АН СССР, 1967.
21. **Communications** of the ACM, 1960—1967. — Algorithms.
22. **Лавров С. С.** Универсальный язык программирования (АЛГОЛ-60). — М.: Наука, 1972.
23. **Агеев М. И.** Основы алгоритмического языка АЛГОЛ-60. — М.: ВЦ АН СССР, 1965.
24. **Боттенбрух Г.** Структура АЛГОЛ-60 и его использование. — М.: ИЛ, 1963.
25. **Мак—Кракен Дж.** Программирование на АЛГОЛе.— М.: Мир, 1964.
26. **Попов В. Н., Степанов В. А., Стишева А. Г., Травникова Н. А.** Программирующая программа. Журн. вычисл. мат. и мат. физ., 1964, № 1.
27. **Pearson K.** Tables of the incomplete beta-function. Biometrica Office. — London: University College, 1934.
28. **Numerische** Mathematik. — Berlin — Heidelberg — New York: Springer-Verlag, 1962—1966.
29. Журнал вычислительной математики и математической физики, 1962—1965.
30. **The computer** journal. — London. — British comp. society, 1961—1966.
31. **The computer** bulletin — London: British comp. society, 1964, 1965.
32. **Journal** of the ACM. — Canada: University of Toronto, 1962, 1964.
33. **Computing.** — Wien/New York: Springer-Verlag, 1966.
34. **Алгоритмы** и алгоритмические языки/ ВЦ АН СССР. — М., 1968, вып. 3.
35. **Крамер Г.** Математические методы статистики. — М.: ИЛ, 1948.
36. **Кендалл М., Стьюарт А.** Теория распределений. — М.: Наука, 1966.
37. **Голенко Д. И.** Моделирование и статистический анализ псевдослучайных чисел на электронных вычислительных машинах. — М.: Наука, 1965.
38. **Агеев М. И.** Единая форма наглядной записи алгоритмов. — Алгоритмы и алгоритмические языки. — М.: ВЦ АН СССР, 1968, вып. 3.
39. **Fletcher A.** A table of complete elliptic integrals. — Philosophical Magazine, 1940, v. 30, p. 517—519.
40. **Беляков В. М., Кравцова Р. И., Раппопорт М. Г.** Таблицы эллиптических интегралов/ ВЦ АН СССР. — М., 1962, т. 2.
41. **Таблицы** e^x и e^{-x}.— М.: Изд-во АН СССР, 1955.
42. **Люстерник Л. А., Акумский Н. Я., Диткин В. А.** Таблицы бесселевых функций. — М.: Гостехиздат, 1949.
43. **Таблицы** натуральных логарифмов/Т. 1: Логарифмы чисел от 0 до 5. Под ред. К. А. Карпова. ВЦ АН СССР. — М., 1960. — (Библиотека мат. таблиц).
44. **Таблицы** степеней целых чисел/Под ред. К. А. Карпова. ВЦ АН СССР. М., 1963. (Библиотека мат. таблиц).
45. **Уайлд Д. Дж.** Методы поиска экстремума.— М.: Наука, 1967.

46. **Zielke G.** ALGOL—Katalog. Matrizenrechnung. — Leipzig: Teubner V., 1972.
47. **Библиотека** алгоритмов 1б—506/ Агеев М. И., Алик В. П., Галис Р. М., Марков Ю. И. — М.: Сов. радио, 1975.
48. **Агеев М. И., Алик В. П., Марков Ю. И.** Библиотека алгоритмов 51б—100б. — М.: Сов радио, 1976. — (Техническая кибернетика).
49. **Агеев М. И., Алик В. П., Марков Ю. И.** Библиотека алгоритмов 101б—150б. — М.: Сов. радио, 1978.— (Техническая кибернетика).
50. **Алгоритмы** (151—200)/ Агеев М. И., **Грюнберг** М. Г., Марков Ю. И. и др./ ИПУ—ВЦ АН СССР. — М., 1970.
51. **Агеев М. И., Марков Ю. И., Швакова Г. М.** Алгоритмы (201—250)/ ИПУ — ВЦ АН СССР. — М., 1971.
52. **Калихман И. Л.** Линейная алгебра и программирование. — М.: Высшая школа, 1967.
53. **Корбут А. А., Финкельштейн Ю. Ю.** Дискретное программирование. — М.: Наука, 1969.
54. **Семенчук А. В.** Расчет генерации гармоник в нелинейных электрических и магнитных трехфазных цепях. — Алгоритмы и программы/ ВНТИЦентр — Кишинев, 1975, № 1.
55. **Bell A. G.,** Algorithm 50. How to program a computer to play legal chess. — The Computer Journal, 1970, v. 13, № 5.
56. **Manning J. R.** Algorithm 68. White to move and mate in n moves. — The Computer Journal, 1971, v. 14, № 2.
57. **Карпов В.** Играет тандем. — «64», 1974, № 6.
58. **Описание** языка АЛГАМС. — В кн.: Алгоритмы и алгоритмические языки/ — ВЦ АН СССР. — М., 1968, вып. 3.
59. **Ляшенко В. Ф.** Программирование для цифровых вычислительных машин М-20, БЭСМ-3М, БЭСМ-4, М-220. — М.: Сов. радио, 1967.
60. **Доброневский О. В., Борохович Я. П., Самченко К. В.** Справочник по электронным вычислительным машинам. — Киев: Вища школа, 1976.
61. **Дроздов Е. А., Комарницкий В. А., Пятибратов А. П.** Электронные вычислительные машины единой системы. — М.: Машиностроение, 1976.
62. **Система** «БЭСМ—АЛГОЛ». Методическое руководство по программированию/ Курочкин В. М., Подшивалов Д. Б., Срагович А. И. и др. — М., 1969. — (Ротапринт ВЦ МГУ).
63. **Система** «БЭСМ—АЛГОЛ»/ Курочкин В. М., Подшивалов Д. Б. и др. Труды 2-й Всесоюзной конференции по программированию./ Ин-т геологии и геофизики СО АН СССР. — Новосибирск, 1965.
64. **Эшби У.** Что такое разумная машина. — Зарубежная электроника, 1962, № 3.
65. **Ботвинник М. М.** Алгоритм игры в шахматы. — М.: Наука, 1968.
66. **Битвинник М. М.** Люди и машины за шахматной доской. — В кн.: Кибернетика ожидаемая и неожиданная. — М.: Наука, 1968.
67. **Адельсон-Вельский Г. М.** и др. О программировании игры ВМ в шахматы. — Успехи мат. наук, 1970, 25, № 2.
68. **Браун.** Шахматы и планирование. — Зарубежная радиоэлектроника, 1971, № 9.
69. **Адельсон-Вельский Г. М., Арлазаров В. Л.** Методы усиления шахматных программ. — Проблемы кибернетики. — М., 1974, вып. 29.
70. **Донской М. В.** О программе, играющей в шахматы. — Проблемы кибернетики. М., 1974, вып. 29.
71. **Гельфанд Я. Ю., Футер А. Л.** Реализация дебютной справочной для шахматной программы. — Проблемы кибернетики. — М., 1974, вып. 29.
72. **Комиссарчик Э. А., Футер А. Л.** Об анализе ферзевого эндшпиля при помощи ЭВМ. — Проблемы кибернетики. — М.: 1974, вып. 29.
73. **Туманов В.** Сказка стала былью. — Шахматы в СССР, 1974, № 10.
74. **Хеикии В.** Ночи «Каиссы». — «64», 1974, № 33.
75. **Хенкин В.** «Каисса» — чемпион мира. — Наука и жизнь, 1975, № 1.
76. **Авербах Ю.** Четыре победы «Каиссы». — Знание — сила, 1975, № 1.
77. **Ботвинник М. М.** О кибернетической цели игры. — М.: Сов. радио, 1975.
78. **Юдин А.** Библиотека эндшпиля ЭВМ. — «Шахматы в СССР», 1975, № 7.
79. **Хенкин В.** Соперник «Каиссы»? — Комс, правда, 15 июля 1975.
80. **Штильман Б.** Машина учится. — Шахматы в СССР, 1976, № 4.
81. **Юдин А. Д.** Программа поиска информации в двумерной таблице с субординацией входов (библиотека позиций эндшпиля). — Программирование, 1976, № 4.
82. **Гик Е. Я.** Математика на шахматной доске. — М.: Наука, 1976.
83. **Александров А. Г.** и др. Обработка больших массивов информации на примере анализа ладейного эндшпиля. — Программирование/ АН СССР. — М., 1977, № 4.
84. **Ботвинник М. М.** «Пионер» готовится к чемпионату. — «Правда», 24 ноября 1977.

85. **Уилкинсон, Райнш.** Справочник алгоритмов на языке АЛГОЛ (линейная алгебра). — М.: Машиностроение, 1976.
86. **Белявский Е. И., Деген А. Б., Этин Ю. Б.** АЛГОЛ-60. — Л.: Гидромет. изд-во, 1966.
87. **Брудно А. Л.** АЛГОЛ. — М.: Наука, 1971.
88. **Савинков В. М., Цальп В. Д.** Программирование на АЛГОЛе. — М.: Высшая школа, 1975.
89. **Kerner I. O., Zielke G.** Einfürung in die algorithmische Sprache ALGOL. — DDR, Leipzig: Teubner Verlag, 1966.
90. **Демин В. Ф., Добролюбов Л. В., Степанов В. А.** Системы программирования на АЛГОЛе. — М.: Наука, 1977.
91. **Гемст В. К., Стукен А. А.** Процедуры на АЛГОЛ-60: Учебное пособие. — Рига: РПИ, 1973.
92. **АЛГАМС** ДОС ЕС ЭВМ/ Бородич Л. И., Жевняк О. Н., Касько Н. К. и др.— М.: Статистика, 1977.
93. **Алгамс ЭВМ «Минск-32»**/ Емельянов В. А., Колесник А. М., Куракина Л. Б. и др. — М.: Статистика, 1976.
94. **Халилов А. И., Ющенко А. А.** АЛГОЛ-60 (программированное учебное пособие). Киев: Вища школа, 1975.
95. **FIDE Album** 1968—1970. — Zagreb: Šahovska naklada, 1977.
96. **Адельсон-Вельский Г. М., Арлазаров В. Л., Донской М. В.** Программирование игр. — М.: Наука, 1978.
97. **Автоматизация** программирования: Пер. с англ./ Под ред. А. П. Ершова. — М.: ФизматГИЗ, 1961.
98. **Пярнпуу А. А.** Программирование на АЛГОЛе и ФОРТРАНе. — М.: Наука, 1978.
99. **Мазной Г. Л.** Программирование на БЭСМ-6 в системе «Дубна». — М.: Наука, 1978.
100. **Автокод БЕМШ** М1970. — М.: ИПМ АН СССР, 1970. — **(Ротапринт)**.
101. **Бахвалов Н. С.** Численные методы в 2-х т.— М.: Наука, 1973.
102. **Кнут Д.** Искусство программирования для ЭВМ в 3-х т. — М.: Мир, 1978. — Т. 3. Сортировка и поиск.
103. **Нивергельт Ю., Фаррар Дж., Рейнгольд Э.** Машинный подход к решению математических задач.— М.: Мир, 1977.
104. **Эренштейн Р. Х.** ЭВМ и шахматная композиция.— Наука и техника, 1979, № 11.
105. **Агеев В. М.** Альбом ФИДЕ и ЭВМ.— Шахматы в СССР, 1979, № 6.
106. **Гурьянов А. Е.** АЛГОЛ-60 в упражнениях.— Л.: ЛГУ, 1978.
107. **Кнут Д. Искусство** программирования для ЭВМ в 3-х т.— М.: Мир, 1977.— Т. 2. Получисленные алгоритмы, с. 101, 193—195.
108. **Ермаков С. М.** Метод Монте-Карло и смежные вопросы.— М.: Наука, 1971.
109. **Романовский И. В.** Алгоритмы решения экстремальных задач.— М.: Наука, 1977.
110. **Вергань А. Ф., Сизиков В. С.** Методы решения интегральных уравнений с программами для ЭВМ (справочное пособие).— Киев: Наукова думка, 1978.
111. **Беляева М. А., Лесик Н. И.** АЛГОЛ-60.— Харьков: Вища школа, 1977.

Список литературы, на которую ссылаются авторы исходных алгоритмов

1i. **Goertzel G.** Mathematical Methods for Digital Computer. — John Wiley and Sons, Inc., 1960.
2i. **Hamming R. W.** Numerical methods for Scientists and Engineers. — McGraw-Hill, 1962.
3i. **Miln-Thomson.** The Calculus of Finite Differences. — Macmillan, 1951, p. 4.
4i. **Milne.** Numerical Calculus. — Princeton, 1949, p. 204.
5i. **Haley K. B.** Mathematical Programming for Business and Industry. — New York, Macmillan; 1968, 127.
6i. **Balas E.** An additive algorithm for solving linear programs with zero — one variables. — Oper. Res. 1965, 13, p. 517—545, ex. 2, 3, 4.
7i. **Haldi J.** 25 integer programming test problems. Working Paper № 43, Grad. Sch. of Bus. Stanford, Calif., 1964, test problems 1—5 and 9.
8i. **Pierce J. F.** Applications of combinatorial programming to a class of all zero-one integer programming problems. — Man. Sci. 1968, 15, 191—209, ex. 1.2.
9i. **Wilson R. B.** Stronger cuts in Gomory's all-integer programming algorithm. — Oper. Res. 1967, 15, 155—156.
10i. **Hooke R., Jeeves T. A.** Direct Search Solution of Numerical and Statistical Problems. — J. ACM, 1961, 8, 2, p. 212—229.

11i. System/360 Scientific Subroutine Package (360A-CM-03X). Version III Programmer's Manual, H20—0205.
12i. Hildebrand. Introduction to Numerical Analysis. Ch. 7.
13i. Muller David E. A method for solving algebraic equations using an automatic computer. — MTAC 1956, 10, p. 208—215.
14i. Technical Note TN 27. — Mathematical Centre, Nov., 1962.
15i. Cadwell J. H. Least squares surface fitting program. — The computer J., 1961, p. 226.
16i. Cadwell J. H., Williams D. E. Some orthogonal methods of curve and surface fitting. — The computer J, 1961, p. 260.
17i. Pearson K. (Ed). Tables of the incomplete beta function. — Cambridge U. Press, 1948.
18i. NBS Tables of Sines and Cosines to 15 Decimal Places.
19i. IBJOB Manual. IBM Systems Reference Library, File No. 7090—27, Form C28-6389-2. IBM 7090/7094 IBSYS Operating System, Version 13, IBJOB Procesor.
20i. Frank, Werner L. Finding zeros of arbitrary Functions. — J. ACM, 1958, 5, p. 154—169.
21i. Traub J. F. Iterative Methods for Solution of Equations. — Prentice-Hall, Englewood Cliffs, N. J.: 1964, 212.
22i. Bulirsch R. Numerical calculation of elliptic integrals and elliptic functions. — Numer. Math., 1965, v. 7, 78—90.
23i. Cody W. J. Complete elliptic integrals. In Hart J. F. et al. Computer Approximations. — New York: Wiley, 1968, p. 150—154, and p. 335—339..
24i. Cody W. J. The FUNPACK package of special function subroutines. — ACM Trans. Mathem. Software, 1975, 1, p. 13—25.
25i. Thacher H. C. Jr. Certification of Algorithm 149. Complete elliptic integral. — CACM, 1963, 4, 166—167.

Содержание

Предисловие 3
Алгоритм 151б. Порядковый номер сочетания в лексикографически упорядоченном списке сочетаний [M1, G6] 6
Алгоритм 152б. Генератор перестановок нулей и единиц [G6] 7
Алгоритм 153б. Целочисленная задача линейного программирования [H] . . 8
Алгоритм 154б. Генератор лексикографически упорядоченной последовательности сочетаний [G6] 12
Алгоритм 155б. Генератор сочетаний с повторениями [G6] 13
Алгоритм 156б. Сумма знакопеременного ряда произведений из элементов сочетаний [G6] 15
Алгоритм 157б. Аппроксимация рядами Фурье [E2] 15
Свидетельство к алгоритму 158б [C1] 19
Алгоритм 159б. Вычисление определителя (рекурсивная процедура) [F3] . . 19
Алгоритм 160б. Число сочетаний [S03] 20
Алгоритм 161б. Вектор чисел всевозможных сочетаний из m элементов [G6, S03] 21
Алгоритм 162б. Вычерчивание графиков [J6] 21
Алгоритм 163б. Модифицированная функция Ханкеля [S17] 25
Алгоритм 164б. Приближение поверхности ортогональными полиномами по методу наименьших квадратов [E2] 26
Алгоритм 165б. Полные эллиптические интегралы [S21] 32
Алгоритм 166б. Обращение матрицы методом Монте-Карло [F1] 34
Алгоритм 167б. Разделенные разности с повторяющимися точками [E1] . . . 37
Алгоритм 168б. Интерполяция по Ньютону с разделенными разностями в обратном направлении [E1] 39
Алгоритм 169б. Интерполяция по Ньютону с разделенными разностями в прямом направлении [E1] 40
Алгоритм 170б. Определитель с полиномиальными элементами [F3] 44
Свидетельство к алгоритму 171б [Z] 48
Алгоритм 172б. Интерполяция табличной функции нескольких переменных (рекурсивная процедура) [E1] 48
Алгоритм 173б. Присваивание значений массивам разной размерности (рекурсивная процедура) [K2] 54

ББК 32.97
Б 59
УДК 681.3.06

Б 59 **Библиотека** алгоритмов 151б—200б: Справочное пособие. Вып. 4/ М. И. Агеев, В. П. Алик, Ю. И. Марков; Под ред. М. И. Агеева. — М.: Радио и связь, 1981. — 184 с., с ил. — (Библиотека технической кибернетики).

1 р. 10 к.

Приведены переводы на русский язык алгоритмов на языке АЛГОЛ-60 по вопросам прикладной математики и программирования, публиковавшихся в журнале «САСМ» (США) под номерами 151—200, исправленных, улучшенных и отлаженных на ЭВМ, а также снабженных подтверждениями и свидетельствами. Как приложения приводятся описание нерекурсивной программы решения шахматных многоходовок, а также подтверждения и замечания к алгоритмам, опубликованным в предыдущих выпусках.

Предназначается для специалистов различного уровня, связанных с работами на ЭВМ.

Б $\frac{30502—156}{046(01)—81}$ 59—81 (С. р.) 1502000000

ББК 32.97
6Ф7.3