



**БИБЛИОТЕКА  
ТЕХНИЧЕСКОЙ  
КИБЕРНЕТИКИ**

**М. И. Агеев, В. П. Алик, Ю. И. Марков**

**БИБЛИОТЕКА  
АЛГОРИТМОВ  
516—1006**

**• СОВЕТСКОЕ РАДИО •**

БИБЛИОТЕКА  
ТЕХНИЧЕСКОЙ  
КИБЕРНЕТИКИ \_\_\_\_\_

М. И. АГЕЕВ, В. П. АЛИК, Ю. И. МАРКОВ

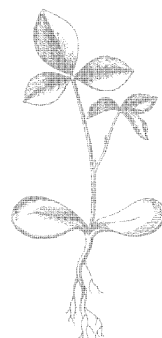
# БИБЛИОТЕКА АЛГОРИТМОВ 516—1006

СПРАВОЧНОЕ ПОСОБИЕ

**Выпуск 2**



МОСКВА  
«СОВЕТСКОЕ РАДИО»  
1976



Scan AAW

6Ф7.3  
А23

УДК 681.3.06

**Агеев М. И., Алик В. П., Марков Ю. И.** Библиотека алгоритмов 516—1006. (Справочное пособие.) Вып. 2. М., «Сов. радио», 1976, 136 с.

В книге приводятся описания алгоритмов по широкому кругу вопросов прикладной математики и программирования на алгоритмическом языке АЛГОЛ-60, публиковавшихся первоначально в журнале «Communications of the ACM» (США) под номерами 51—100, переведенных на русский язык, исправленных, улучшенных и отлаженных на ЭВМ авторами данного выпуска. Каждый алгоритм снабжен подтверждениями и **свидетельствами**, содержащими примеры применения, результаты отладки, критические оценки и сравнительные характеристики публикуемых алгоритмов.

В качестве приложений к выпуску приводится алгоритм «Как программировать игру в шахматы», а также замечания и подтверждения к ранее опубликованным алгоритмам, среди которых особое место занимает статья Р. Лондона «Доказательство алгоритмов — новый вид подтверждения».

Книга является настольной для специалистов различного уровня, связанных с работами на ЭВМ.

Рис. 7, табл. 32, библи. 77 назв.

#### Члены редакционного совета:

**Трапезников В. А.** (председатель), **Челюстин А. Б.** (зам. председателя), **Бусленко Н. П.**, **Виленкин С. Я.**, **Воронов А. А.**, **Гаазе—Рапорт М. Г.**, **Дудников Е. Г.**, **Ицкович Э. Л.**, **Копелович А. П.**, **Круг Г. К.**, **Мамионов О. Г.**, **Осколков И. И.**, **Пархоменко П. П.**, **Пинскер М. С.**, **Плискин Л. Г.**, **Поспелов Г. С.**, **Райбман Н. С.**, **Самойленко С. И.**, **Таль А. А.**, **Флешман Б. С.**, **Хургин Я. И.**, **Цыпкин Я. З.**, **Якобсон Б. М.**

#### Редакция кибернетической литературы

А  $\frac{30502-021}{046(01)-76}$  80-75

© Издательство «Советское радио», 1976 г.

## Предисловие

Данный выпуск служит продолжением серии, начатой выпуском «Библиотека алгоритмов 16—506» [38], и включает в себя алгоритмы, полученные в результате дальнейшего совершенствования алгоритмов выпуска «Алгоритмы (51—100)» [23]. Последний содержал алгоритмы 51a—100a, являющиеся, в свою очередь, результатом переработки соответствующих алгоритмов журнала «Communications of the ACM» (CASM) [17]. К каждому из алгоритмов данного выпуска прилагаются соответствующие «Подтверждения» и «Замечания» как из вышеуказанного журнала, так и советских пользователей алгоритмами, а также «Свидетельства», составленные авторами выпуска. Алгоритмы публикуются здесь на эталонном алгоритмическом языке АЛГОЛ-60 [21], описываемом во многих учебниках [13—16, 27, 28]. Там, где это возможно без заметного ухудшения алгоритмов, они предварительно перерабатывались авторами выпуска на сокращенный АЛГОЛ-60 [12] с некоторыми его расширениями в сторону полного АЛГОЛа [допускались: 1) возведение целых чисел в любую целую степень, 2) различение идентификаторов по всем содержащимся в них символам, 3) операция  $\div$  и 4) условное именуемое выражение]. В частности, все алгоритмы здесь записаны с использованием только строчных букв латинского алфавита\*. Прописные буквы используются только в алгоритме 50CJ (см. приложение 1) для идентификаторов, являющихся русскими словами.

В свидетельствах к алгоритмам указывается оригинал переработанного алгоритма, перечисляются виды работ, проведенных 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 относится перевод на русский язык комментариев, подтверждений и замечаний, придание алгоритму наглядной, удобочитаемой формы путем применения однотипной ступенчатой записи по правилам, опубликованным в статье [22], и использования 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 [12] и те случаи, когда алгоритмы составлялись заново, оговариваются в свидетельствах особо.

Предыдущий выпуск сопровождался тематическим указателем алгоритмов, появившихся в советской и зарубежной печати к моменту составления выпуска. Алгоритмы в этом тематическом указателе груп-

---

\* Из типографских соображений эти латинские буквы набраны в тексте курсивом, а в описаниях процедур — прямым шрифтом. Например, в алгоритме 51b *adjust* и *adjust* означают один и тот же идентификатор. (Прим. ред.)

пировались в соответствии с классификацией, принятой в журналах «САСМ». Названия алгоритмов в выпусках сопровождаются индексами, соответствующими этой классификации. Например, в заголовке «Алгоритм 726. Генератор композиций [A1]» индекс [A1] указывает, что алгоритм 726 относится к классу A1 («Арифметика. Теория чисел»). Расшифровка таких индексов имеется в вышеупомянутом тематическом указателе.

В выпусках серии «Библиотека алгоритмов» номера алгоритмов снабжаются буквой «б» (например, «Алгоритм 726б») для отличия их как от исходных, так и от алгоритмов, издававшихся в предыдущей серии [2, 23—26]. В ссылках на источники аббревиатура «САСМ» означает журнал «Communications of the ACM» [17]. 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. Контрольные решения по алгоритмам проводились с использованием транслятора \* TA-1M [10, 45] на машинах М-220 [44] и транслятора БЭСМ-АЛГОЛ [29, 45] на машине БЭСМ-6 (быстродействие — миллион операций в секунду, память — свыше 32 000 ячеек, длина мантиссы чисел — 10 цифр).

В серии «Библиотека алгоритмов» будут публиковаться только отлаженные алгоритмы. Однако, учитывая известный каждому программисту факт, что никакая отладка и даже многолетнее использование не гарантируют абсолютной безошибочности программ, а также и то, что пределов совершенствования алгоритмов практически не существует, авторы выпуска обращаются ко всем читателям и пользователям с просьбой присылать свои замечания и подтверждения в адрес издательства (для Агеева М. И.). Публикация таких замечаний в последующих выпусках будет продолжена, так же как это сделано в данном выпуске (см. приложение 2).

Работа по подготовке к изданию алгоритмов предыдущей серии [2, 23—26] была начата в соответствии с объявлением, регулярно (начиная с мая 1964 г.) публиковавшемся в разделе «Алгоритмы» журнала «САСМ»: «Репродукция алгоритмов данного раздела разрешается безвозмездно. Если репродукция делалась с целью публикации, то необходима ссылка на автора алгоритма и на выпуск «САСМ», в котором опубликован алгоритм». Первый выпуск «Алгоритмы (1—50)» сразу же после публикации был послан на отзыв редактору раздела «Алгоритмы» журнала «САСМ» президенту АСМ проф. Г. Форсайту и получил полное его одобрение.

Появление в печати предыдущей серии [2, 23—26] было встречено горячим одобрением не только подавляющего большинства читателей и пользователей, но 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»). Тем не менее, так же как и в предыдущем выпуске, здесь подчеркивается, что коллектив сотрудников, выполнявших одновременно работу составителей, переводчиков, переработчиков и отладчиков алгоритмов и собирательно именуемых здесь авторами выпуска, на авторство на отдельные алгоритмы отнюдь не претендует, за исключением разве толь-

---

\* В дальнейшем здесь наряду со словом «транслятор» будут употребляться слова «транслирующая система» или просто «система». (Прим. ред.)

ко тех специально оговоренных в свидетельствах случаев, 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 повышали 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 их подтверждений. Поэтому читателям и пользователям в их ссылках на источники нужно указывать авторов исходных алгоритмов и номера журнала «САСМ», из которых взяты эти алгоритмы, вне зависимости от того, будут ли при этом читатели упоминать и авторов соответствующего выпуска данной серии или нет.

Контрольные решения по большинству алгоритмов данного выпуска проводил Ю. И. Марков, по алгоритмам 51б—53б, 55б—58б, 66б, 70б, 75б, 78б, 96б и 99б — В. П. Алик, Л. В. Малюк и Р. М. Галис, по алгоритмам 63б—65б и 98б — Н. С. Путвинская, по алгоритму 85б — В. Ф. Дейкин, по алгоритму 50СJ—М. И. Агеев. Рукопись подготовил к изданию В. П. Алик. В проверке рукописи принимал участие Э. М. Каплинский. Основная переработка алгоритмов, написание текста свидетельств, компоновка выпуска и его общая редакция выполнены М. И. Агеевым.

## АЛГОРИТМ 516

### Корректировка обратной матрицы после изменения одного элемента в прямой матрице [F1]

Если дана матрица  $a = m^{-1}$ , имеющая размерность  $n \times n$ , а  $M$  — матрица, которая получается в результате увеличения одного элемента  $m[i, j]$  прямой матрицы  $m$  на величину  $d$ , то процедура *adjust* (*adjust* — корректировать, настраивать) может вычислить скорректированную матрицу  $b = M^{-1}$  по элементам матрицы  $a$  без обращения матрицы  $M$ .

```
procedure adjust(a,n,i,j,d)result: (b);  
  value n,i,j,d; real d; integer i,j,n; array a,b;  
begin real t; integer r,s;  
  t:=d/(a[j,i]×d+1);  
  for r:=1 step 1 until n do  
    for s:=1 step 1 until n do  
      b[r,s]:=a[r,s]—t×a[r,i]×a[j,s]  
end adjust;
```

### Свидетельство к алгоритму 516

Процедура *adjust* алгоритма 516 является стереотипным переизданием процедуры *adjust* алгоритма 51a. Правильность алгоритма 51a была подтверждена расчетами на машине Урал-2 И. Р. Гитманом [25, с. 171] и в расчетах на машине БЭСМ-6 Л. С. Кривонос и З. А. Шиншиновой [38, прил. 1].

### Свидетельство к алгоритму 51a

Алгоритм 51a получен в результате исправления, сокращения и ординарной переработки алгоритма 51 (Hendson J. R. «CACM», 1961, № 4).

Алгоритм 51a проверен с помощью транслятора ТА-1 для исходных данных

$$a = \begin{vmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{vmatrix}, \quad n=2, \quad i=1, \quad j=2, \quad d=3.$$

Получен результат

$$b = \begin{vmatrix} -0.363636363 & 0.454545454 \\ 0.272727272 & -0.0909090909 \end{vmatrix},$$

полностью совпадающий с точным решением

$$b = \begin{vmatrix} -\frac{4}{11} & \frac{5}{11} \\ \frac{3}{11} & -\frac{1}{11} \end{vmatrix}.$$

### Подтверждение к алгоритму 51

Р. Георг (Georg R. «CACM», 1962, № 7)

Эта процедура была запрограммирована на языке ФОРТРАН и переведена на машинный язык автоматически. Процедура была *проверена* на Аргонской вычислительной машине GEORGE. Использова-



лась программа интерпретации плавающей запятой, обеспечивающая максимальную точность до 31 разряда.

Процедура проверялась для матриц порядка  $n=2, 3, \dots, 10$ . Для каждого значения  $n$  делалось по 20 успешных опытов. Каждый опыт состоял из произвольного изменения произвольно выбранного элемента матрицы  $M$  с последующим применением процедуры *adjust*, а затем вычислением произведения  $N:=b \times M$ . Для каждого опыта погрешность вычислялась как

$$sum := \left\{ \sum_{i=1}^n \sum_{j=1}^n N[i, j] \right\} - n.$$

Для случайных приращений элемента, находящихся в пределах от  $-1.0$  до  $+1.0$ , значение *sum* никогда не превышало  $2.0 \times 10^{-8}$ .

В процедуре обнаружены две опечатки ... \*

## АЛГОРИТМ 526

### Генератор тест-матриц [F1]

Процедура *testmatr* вычисляет матрицу  $a$  любого порядка  $n$ , для которой заранее точно известна обратная матрица  $a^{-1}$  и собственные значения. В матрице  $a^{-1}$   $n$ -я строка и  $n$ -й столбец представляют собой последовательности чисел  $1, 2, \dots, n$ . После вычеркивания  $n$ -го столбца и  $n$ -й строки матрицы  $a^{-1}$  получается единичная матрица.

```

procedure testmatr (n) result: (a);
  value n; integer n; array a;
begin real c,d; integer i,j;
  c:=n×(n+1)×(2×n—5)/6;
  d:=1/c; a[n,n]:=—d;
  for i:=1 step 1 until n—1 do
    begin a[i,n]:=a[n,i]:=d×i;
      a[i,i]:=d×(c—i↑2);
      for j:=1 step 1 until i—1 do
        a[i, j]:=a[j,i]:=—d×i×j
    end
end testmatr;

```

### Свидетельство к алгоритму 526

Процедура *testmatr* алгоритма 526 является стереотипным переизданием процедуры *testmatr* алгоритма 52а. В «Свидетельство к алгоритму 52а» внесена поправка. Вместо фразы «Для контроля эта обращенная матрица обращалась снова ...» должно быть «Для контроля эта тест-матрица обращалась ...»

### Свидетельство к алгоритму 52а

Процедура *testmatr* алгоритма 52а получена в результатеординарной переработки процедуры TESTMATRIX, приведенной П. Науром в его «Замечании и подтверждении к алгоритму 52» («САСМ», 1963, № 1), перевод которого дается ниже.

---

\* Указываются две опечатки в алгоритме 51, учтенные при составлении алгоритма 51а. (Прим. ред.)



Процедура *testmatr* проверена на ТА-1, и с точностью до восьми десятичных разрядов получены следующие результаты.

Для  $n=4$

0.9	-0.2	-0.3	0.1
-0.2	0.6	-0.6	0.2
-0.3	-0.6	0.1	0.3
0.1	0.2	0.3	-0.1

Для контроля эта тест-матрица обращалась с помощью алгоритмов 58 и 66, в результате чего получилась матрица

1	0	0	1
0	1	0	2
0	0	1	3
1	2	3	4

что соответствует пояснительному тексту к процедуре *testmatr*.

#### Подтверждение к алгоритму 52

Х. Е. Гильберт (Gilbert H. E. «CASM», 1961, № 8)

Оператор  $c := t \times (t+1) \times (t+t-5)/6$ ; был заменен оператором  $c := n \times (n+1) \times (n+n-5)/6$ ; чтобы привести обращение к форме, указанной в описании алгоритма. Алгоритм был переведен на язык ФОРТРАН и проверен в UCSD с помощью программы, вычисляющей собственные значения матриц, на вычислительной машине CDC 1604. Для матрицы  $20 \times 20$  собственные значения следующие:

1. 1.000000;
2. 1.000000;
- . . . . .
19. 0.01636693;
20. -0.02493833.

#### Подтверждение к алгоритму 52

Дж. Хилмор (Hillmore J. S. «CASM», 1962, № 8)

Алгоритм был исправлен согласно рекомендации Х. Е. Гильберта в его «Подтверждении» («CASM», 1961, № 8) и затем успешно проверен на машине National-Elliott 803, использующей транслятор с языка Эллиот-АЛГОЛ. Полученные матрицы были использованы для проверки процедуры GJR обращения матриц, приведенной Х. Р. Шварцем в его статье «Введение в АЛГОЛ» («CASM», 1962, № 2).

#### Замечание и подтверждение к алгоритму 52

П. Наур (Naур P. «CASM», 1963, № 1)

Кроме внесения поправок, указанных Х. Е. Гильбертом («CASM», 1961, № 8), алгоритм был упрощен путем применения одновременного присваивания и исключения локализованных переменных  $t$  и  $f$ . В результате алгоритм имеет вид . . . \*

Этот вариант алгоритма был успешно пропущен в системе GIR ALGOL вместе с процедурами *INVERSION II* и *gjr* (см. «Подтверж-

\* Приводится новый вариант процедуры *testmatr*, соответствующий данному в алгоритме 52а. (Прим. ред.)

дение к алгоритму 120 [24]»). Из результатов, выданных процедурой *INVERSION II* видно, что определитель этих матриц задается выражением  $6/n(n+1)(5-2n)$ , которое является также значением элемента  $a[n,n]$ . Для  $n > 3$  наибольшим по модулю элементом является  $a[1,1] = 1 + a[n,n]$ .

### Дополнительное замечание к алгоритму 52

П. Наур (Naур Р. «САСМ», 1963, № 8)

Просматривая результаты отыскания собственных значений, я сделал вывод, что все, кроме двух, собственные значения тест-матриц равны единице, а два остальных задаются выражениями  $6/(p \times (n+1))$  и  $p/(n \times (5-2 \times n))$ , где  $p = 3 + \text{sqrt}((4 \times n - 3) \times (n-1) \times 3/(n+1))$ .

Эти выражения были использованы для определения абсолютных погрешностей в собственных значениях, вычисляемых процедурой *jacobi* (алгоритм 85), с помощью алгоритма «Триангуляризация по Хаусхолдеру» и т. д., как описано ниже. Они были использованы также для вычисления табл. 1 (применялась система GIER ALGOL с 29 значащими разрядами).

В табл. 1 цифры для  $n=20$  хорошо согласуются с результатами, приведенными в «Подтверждении к алгоритму 52» Х. Гильберта («САСМ», 1961, № 8).

Таблица 1

n	Определитель	Собственные значения, отличные от 1	
		(n-1)-е	n-е
3	-0.50000000	0.22474487	-2.2247449
4	-0.10000000	0.15311289	-0.65311289
5	-0.04000000	0.11323808	-0.35323808
6	-0.020408163	0.088290570	-0.23114771
7	-0.011904762	0.071428571	-0.16666667
8	-0.0075757576	0.059386081	-0.12756790
9	-0.0051282052	0.050422549	-0.10170460
10	-0.0036363636	0.043532383	-0.083532383
11	-0.0026737968	0.038097478	-0.070183039
12	-0.0020242915	0.033718770	-0.060034559
13	-0.0015698587	0.030128103	-0.052106125
14	-0.0012422360	0.027139206	-0.045772747
15	-0.0010000000	0.024619013	-0.040619013
16	-0.00081699347	0.022470157	-0.036359046
17	-0.00067613252	0.020619902	-0.032790288
18	-0.00056593096	0.019012916	-0.029765605
19	-0.00047846890	0.017606429	-0.027175807
20	-0.00040816327	0.016366903	-0.024938332

### АЛГОРИТМ 536

#### Извлечение корней n-й степени из комплексного числа [B4]

Процедура *nroot* (от *root* — корень) определяет  $n$  корней уравнения  $x^n = r + u \times i$ . Действительные части запоминаются в векторе *real*. Мнимые части запоминаются в соответствующих местах вектора *unreal*.

```
procedure nroot(n,r,u) result: (real,unreal);
  value n,r,u; real r,u; integer n; array real, unreal;
```

```

begin real c,m,pi,s,t; integer i;
  m:=1/n; pi:=3.14159265;
  s:=(r×r+u×u)↑(m/2);
  t:= if r=0 then sign (u) ×pi/2 else
      if r>0 then arctan(u/r) else arctan(u/r) +pi;
  t:=m×t; c:=2×pi×m;
  for i:=1 step 1 until n do
    begin real[i]:=s×cos(t);
      unreal[i]:=s×sin(t);
      t:=t+c
    end i
end nroot;

```

### Свидетельство к алгоритму 53б

Алгоритм 53б является стереотипным переизданием алгоритма 53а.

### Свидетельство к алгоритму 53а

Алгоритм 53а получен в результате сокращения и ординарной переработки алгоритма 53 (Hengdon J. R. «САСМ», 1961, № 4).

С помощью транслятора ТА-1 были извлечены корни 3-й и 5-й степеней из комплексных чисел  $8+6i$  и  $-8+6i$ . Результаты:

1.  $x^n = 8 + 6i$ ,  $n = 3$ ;

$$\begin{aligned}
 x &= 2.10506122 + 0.458591405i, \\
 x &= -1.44968241 + 1.59374079i, \\
 x &= -0.655378815 - 2.05233219i.
 \end{aligned}$$

2.  $x^n = -8 + 6i$ ,  $n = 3$ ;

$$\begin{aligned}
 x &= 1.44968242 + 1.59374079i, \\
 x &= -2.10506122 + 0.458591412i, \\
 x &= 0.655378793 - 2.05233220i.
 \end{aligned}$$

3.  $x^n = 8 + 6i$ ,  $n = 5$ ;

$$\begin{aligned}
 x &= 1.57178541 + 0.203413470i, \\
 x &= 0.292250701 + 1.55771498i, \\
 x &= -1.39116454 + 0.759307336i, \\
 x &= -1.15203768 - 1.08843723i, \\
 x &= 0.679166102 - 1.43199854i.
 \end{aligned}$$

4.  $x^n = -8 + 6i$ ,  $n = 5$ ;

$$\begin{aligned}
 x &= 1.39116455 + 0.759307332i, \\
 x &= -0.292250696 + 1.55771498i, \\
 x &= -1.57178541 + 0.203413475i, \\
 x &= -0.679166118 - 1.43199854i, \\
 x &= 1.15203766 - 1.08843724i.
 \end{aligned}$$

Эти результаты совпали шестью разрядами с результатами ручного счета.

### Замечание к алгоритму 53

К. В. Нестор (Nestor C. W. «САСМ», 1961, № 7)

Значительная экономия машинного времени для  $n \geq 3$  получилась

бы в результате использования рекуррентных формул для синуса и косинуса вместо обращений к подпрограммам *sin* и *cos* в цикле, соответствующем корням *n*-й степени из комплексного числа. Можно использовать формулы

$$\begin{aligned}\sin(n+1)\theta &= \sin n\theta \times \cos \theta + \cos n\theta \times \sin \theta. \\ \cos(n+1)\theta &= \cos n\theta \times \cos \theta - \sin n\theta \times \sin \theta\end{aligned}$$

ценой затраты некоторой дополнительной памяти.

Мы нашли, что эта процедура весьма эффективна в задачах, связанных с разложением в ряд Фурье, как это было предложено Г. Гюрзелем в гл. 24 работы «Математические методы для вычислительных машин» [11i].

#### **Свидетельство к алгоритму 546 [S14]**

Алгоритм 546 «Гамма-функция для интервала [1,2]» не публикуется здесь, потому что в журнале «САСМ» позднее были опубликованы более совершенные алгоритмы 80, 221 и 291 для вычисления гамма-функции. См. «Замечания к алгоритмам 34, 54, 80, 221 и 291» М. Пайка и И. Хилла, перевод которых опубликован вслед за «Свидетельством к алгоритму 34а» [38], а также алгоритмы 80б и 221а [26].

#### **АЛГОРИТМ 556**

##### **Полный эллиптический интеграл первого рода [S21]**

Процедура-функция *elliptic1* вычисляет эллиптический интеграл первого рода  $K(k, \pi/2)$ .

```
real procedure elliptic1(k);  
    value k; real k;  
begin real t;  
    t:=1-k×k;  
    elliptic1:=((0.032024666×t+0.054544409)×t  
        +0.097932891)×t+1.3862944  
        -(((0.010944912×t+0.060118519)×t  
        +0.12475074)×t+0.5)×ln(t)  
end elliptic1;
```

#### **Свидетельство к алгоритму 556**

Алгоритм 556 является стереотипным переизданием алгоритма 55а. Правильность алгоритма 55а была подтверждена в расчетах на машине БЭСМ-6. (См. «Подтверждение к алгоритмам 4а, 7а, ..., 238а» Л. С. Кривонос и З. А. Шиншиновой [38, прил. 1].)

#### **Свидетельство к алгоритму 55а**

Алгоритм 55а получен в результате исправления и ординарной переработки алгоритма 55 (Hendson J. R. «САСМ», 1961, № 4).

С помощью ТА-1 проведено контрольное решение, и для  $k=0.5$  получено  $K=1.68574994$ . Табличное значение [1] этого интеграла  $K=1.6858$ .

## Подтверждение к алгоритмам 55 и 149

Г. Тачер (Thacher Н. С. «САСМ», 1963, № 4)

Тела процедур алгоритмов 55 и 149 были проверены на машине LGP-30, использующей язык SCALP дартмутского транслятора для подмножества языка АЛГОЛ-60. Арифметика с плавающей запятой имеет семь значащих цифр.

Кроме модификаций, которые потребовались в связи с ограничениями, налагаемыми языком SCALP, оказались необходимыми следующие поправки ... \*

Параметры алгоритма 149 связаны с эллиптическим интегралом первого рода соотношением  $K = a \times \text{ELIP}(a, b)$ , где параметр  $m = k^2 = 1 - b^2/a^2$ . \*\*

Максимальная погрешность приближения в алгоритме 55 дается Хастингсом (Hastings) как  $0.6 \times 10^{-6}$ . Кроме того, может получиться существенная ошибка округления при формировании дополнительного параметра  $t = 1 - k \times k$ . Для  $k$ , близкого к 1, ошибка так велика, что достоверны только четыре значащие цифры. (В этой области дополнительный параметр  $t$  является гораздо более удовлетворительным приближением.)

Точность, достижимая в алгоритме 149, ограничивается только точностью арифметики и размером усилий, которые желательно затратить. Точность шесть цифр была получена с пятью вычислениями арифметическо-геометрического среднего для  $a = 1000$ ,  $b = 2$  и с одним вычислением для  $a = 500$ ,  $b = 500$ .

Ни тот, ни другой из алгоритмов неудовлетворителен при  $k = 1$ . Работа алгоритма 55 определяется ошибкой, получающейся при вычислении логарифма. При этих условиях (т. е. при  $k = 1$ ) алгоритм 149 входит в бесконечный цикл. Алгоритм 149 может перейти в бесконечный цикл также из-за величины предельной константы ( $10^{-8}$  в опубликованном алгоритме), оказывающейся слишком малой для используемой арифметики. Для арифметики SCALP было найдено необходимым увеличить это допустимое отклонение до  $5.0_{10} - 7$ . Результирующие значения эллиптических интегралов имели точность от двух до семи значащих цифр (до шести цифр в десятичной системе счисления).

Сравнительная эффективность этих двух алгоритмов сильно зависит от эффективности подпрограмм квадратного корня и логарифма. Для большинства вычислительных систем алгоритм 55 обеспечивает достаточную точность и более эффективен. Если же применяется операция извлечения квадратного корня или высокоэффективная подпрограмма вычисления квадратного корня, то алгоритм 149 может оказаться лучше.

## АЛГОРИТМ 566

### Полный эллиптический интеграл второго рода [S21]

Процедура-функция *elliptic2* вычисляет эллиптический интеграл второго рода  $E(k, \pi/2)$ , если задано  $t = 1 - k \times k$ . При  $k = 1$  происходит выход к глобальной метке *signal56*.

---

\* Указываются две ошибки в алгоритме 55 и одна ошибка в алгоритме 149. (Прим. ред.)

\*\* В оригинале напечатано  $1 - b/a$ . (Прим. ред.)

```

real procedure elliptic2(t);
    value t; real t;
begin if t=0 then go to signal56;
    elliptic2:=
        ((0.040905094×t+0.085099193)×t+0.44479204)×t+1.0
        —((0.01382999×t+0.08150224)×t+0.24969795)×t)×ln(t)
end elliptic2;

```

### Свидетельство к алгоритму 56б

Алгоритм 56б получен из алгоритма 56а путем следующих очевидных изменений.

1. Входной параметр  $k$  был заменен на  $t=1-k \times k$ . Соответственно был удален оператор  $t:=1-k \times k$ ; и скорректированы спецификации.

2. В начало тела процедуры добавлен оператор  
**if** t=0 **then go to** signal56;

3. Соответствующие этим модификациям коррективы внесены в пояснительный текст.

Правильность алгоритма 56б была подтверждена расчетами на машине БЭСМ-6 (см. «Подтверждение к алгоритмам 4а, 7а, ..., 238а» Л. С. Кривонос и З. А. Шиншиновой [38, прил. 1]).

### Свидетельство к алгоритму 56а

Алгоритм 56а получен в результате исправления и ординарной переработки алгоритма 56 (Hendon J. R. «САСМ», 1961, № 4). Были внесены следующие исправления.

1. Начало оператора *elliptic2:=*(( было заменено на *elliptic2:=*(( .

2. Идентификатор *log* был заменен на *ln*.

**С помощью транслятора ТА-1 для  $k=0.5$  получено  $E=1.46746165$ . Табличное значение [1] равно  $E=1.4675$ .**

### Подтверждение к алгоритму 56

Г. Ларсен (Larsen G. M. «САСМ», 1966, № 1)

Алгоритм 56 был проверен на машине Univac 1107 с использованием транслятора Univac 1107 ALGOL 60 (датированного 25 января 1965 г.). В этой машине числа представляются с плавающей запятой восемью значащими цифрами.

В алгоритме были исправлены две синтаксические ошибки...\*

Кроме того, оператор  $t:=1-k \times k$  был удален из алгоритма, а сам параметр  $t$  был принят в качестве входного параметра процедуры  
**real procedure** ELLIPTIC2(t); **value** t; **real** t;

во избежание погрешности, когда значение  $k$  близко к единице.

Было вычислено несколько значений полного эллиптического интеграла второго рода для  $1 \geq t > 0$ . Найдено, что максимальная погреш-

---

\* Указываются две ошибки в алгоритме 56, уже замеченные ранее авторами выпусков и исправленные при составлении алгоритма 56а. (Прим. ред.)

ность равна примерно  $7_{10}-7$ , если сравнивать результаты с таблицами А. М. Лежандра [12i]. Для  $t=0$  имел место выход по ошибке из программы вычисления  $ln$ .

## АЛГОРИТМ 576

### Функции Томсона $ber$ и $bei$ [S19]

Процедура-функция  $berbei$  вычисляет функцию Томсона [9, с. 997]  $ber(z)$ , если  $r$  положить равным нулю, или  $bei(z)$ , если  $r=1$ .

З а м е ч а н и е. В нижеследующей процедуре равенство  $s=s+k$  начнет выполняться с того момента, когда отношение  $k/s$  обратится в машинный нуль.

```

real procedure berbei(r,z);
    value r,z; real z; integer r;
begin real s,k,p; integer i;
    p:=z×z;
    k:=s:= if r=0 then 1 else p/4;
    p:=p×p;
    for i:=2 step 2 until 100 do
        begin
            if s=s+k then go to fin;
            k:=-0.0625×k×p/((i+r)×(i+r-1))↑2;
            s:=s+k
        end i;
    fin: berbei:=s
end berbei;

```

### Свидетельство к алгоритму 576

Алгоритм 576 является стереотипным переизданием алгоритма 57а.

### Свидетельство к алгоритму 57а

Алгоритм 57а получен в результате исправления, сокращения и ординарной переработки алгоритма 57 (Herndon J. R. «CACM», 1961, № 4).

Во избежание возможной неточности внутреннего машинного представления параметров  $r$  и  $i$  ( $r$  и  $c$  — в алгоритме 57), которая может привести к неверному выполнению условия **if**  $r=0$  **then** и оператора цикла, указанные параметры в алгоритме 57а были отнесены к типу **integer**.

Результаты решения с помощью алгоритма 57а в системе ТА-1 приведены в табл. 2.

Т а б л и ц а 2

Функция	Результаты трансляции	Табличные значения
$berbei(0, 1.65)=ber(1.65)$	0.884559782	0.8845598
$berbei(1, 1.65)=bei(1.65)$	0.671876813	0.6718768

Табличные значения функций  $ber$  и  $bei$  взяты из работы [9].



### Подтверждение к алгоритму 57

А. П. Релф (Relph A. P. «CACM», 1962, № 7)

Алгоритм 57 был транслирован с использованием транслятора DEUCE ALGOL. Никаких изменений не потребовалось, результаты были удовлетворительными.

### Подтверждение к алгоритму 57

Г. Тачер (Thacher H. C. «CACM», 1962, № 8)

Тело процедуры было проверено на машине LGP-30, использующей АЛГОЛ-транслятор, разработанный Вычислительным центром Дартмутского колледжа. Не было найдено никаких синтаксических ошибок. Для  $z=0.1, 0.2, \dots, 1.0$  с использованием семизначной десятичной арифметики программа дала результаты с погрешностями, меньшими пяти единиц (а для  $z=1, 2, \dots, 5$  меньшими 12 единиц) седьмого разряда. Для больших значений  $z$  могут быть серьезные погрешности округления. Например, для  $z=20$  таким путем может быть потеряно более двух десятых значения.

## АЛГОРИТМ 586

### Обращение матрицы методом Гаусса-Жордана [F1]

Процедура *invert58* обращает квадратную матрицу  $a$  порядка  $n$  методом исключения по Гауссу и Жордану. Попутно вычисляется определитель  $det$  матрицы  $a$ . Если матрица вырожденная или близка к вырожденной, то для предохранения от переполнения (при делении на значение главного элемента  $y$ ) выполняется условный оператор, обеспечивающий переход к нелокализованной метке *signal58*, как только значение главного элемента  $y$  по модулю станет меньше некоторого эмпирически данного значения  $eps$ .

```
procedure invert58(n,eps) dataresult: (a) result: (det);
  value n,eps; real eps,det; integer n; array a;
begin real y,w; integer i,j,k,r,p; array b,c[1:n];
  integer array z[1:n];
  det:=1;
  for j:=1 step 1 until n do z[j]:=j;
  for i:=1 step 1 until n do
    begin k:=i; y:=a[i,i];
      r:=i-1; p:=i+1;
      for j:=p step 1 until n do
        begin w:=a[i,j];
          if abs(w)>abs(y) then
            begin k:=j; y:=w end
        end;
      det:=y×det;
      if k≠i then det:=-det;
      if abs(y)<eps then go to signal 58;
      y:=1/y;
      for j:=1 step 1 until n do
```

```

begin c[j]:=a[j,k]; a[j,k]:=a[j,i];
a[j,i]:=-c[j]×y;
b[j]:=a[i,j]:=a[i,j]×y
end;
j:=z[i]; z[i]:=z[k]; z[k]:=j;
a[i,i]:=y;
for k:=1 step 1 until r, p step 1 until n do
  for j:=1 step 1 until r,p step 1 until n do
    a[k,j]:=a[k,j]-b[j]×c[k]
  end j;
for r:=1 step 1 until n do
  begin k:=z[r];
  for j:=r while k≠j do
    begin
      for i:=1 step 1 until n do
        begin w:=a[j,i]; a[j,i]:=a[k,i];
        a[k,i]:=w
        end i;
      i:=z[k]; z[k]:=z[j];
      k:=z[j]:=i
    end j
  end r
end invert58;

```

#### Свидетельство к алгоритму 58б

Алгоритм 58б отличается от алгоритма 58а только тем, что в операторе

if abs(y) < eps then go to signal58;

была исправлена опечатка: символ > был заменен символом <.

#### Свидетельство к алгоритму 58а

Алгоритм 58а получен в результате переработки алгоритма 58 (Cohen D. «САСМ», 1961, № 5) в соответствии с рекомендациями «Замечания к алгоритму 58» П. Г. Беренца («САСМ», 1962, № 12). Кроме того, в процедуру *invert58* введен оператор  $y:=1/y$  для экономии машинного времени вследствие замены  $2n^2$  делений умножениями, согласно рекомендации Р. Георга («Подтверждение к алгоритму 58». «САСМ», 1962, № 8). Перевод «Замечания к алгоритму 58» Г. Струбле («САСМ», 1962, № 8) здесь не приводится, поскольку это замечание не вносит ничего нового в приводимую информацию.

Результаты трансляции алгоритма 58а с тест-матрицей четвертого порядка приведены в «Свидетельстве к алгоритму 52а». Кроме того, обращалась матрица Вильсона

5	7	6	5
7	10	8	7
6	8	10	9
5	7	9	10

с помощью алгоритма 58а и алгоритма 66а. Для обеих процедур результаты совпадают до восьмого разряда включительно и равны

67.999999			
—40.999999	24.999999		
—16.999999	9.9999999	4.9999999	
9.9999999	—5.9999999	—2.9999999	1. 9999999

### Подтверждение к алгоритму 58

Р. А. Конджер (Conger R. A. «CASM», 1962, № 6)

Процедура была закодирована вручную на языке ФОРТРАН для машины IBM 1620. Были найдены необходимыми следующие поправки...\*

После этих исправлений программа была проверена обращением матрицы  $6 \times 6$  и обращением результата. Вторичный результат совпал с исходной матрицей с точностью до ошибок округления.

### Подтверждение к алгоритму 58

Р. Георг (George R. «CASM», 1962, № 8)

Эта процедура была запрограммирована на языке ФОРТРАН, переведена в машинный код автоматически и проверена на вычислительной машине George, построенной в Аргоне. Использовалась программа плавающей запятой, дающая точность максимум 31 разряд. Обнаружено несколько различных ошибок...\*\*

Пересмотренный алгоритм был проверен на вычислительной машине LGR-30 с использованием языка АЛГОЛ-30, малого подмножества языка АЛГОЛ. С ограничениями, принятыми в подмножестве, на проверенных матрицах программа работала удовлетворительно.

### Замечание к алгоритму 58

П. Г. Беренц (Behrenz P. G. «CASM», 1962, № 12)

Эта процедура была пропущена на машине FACIT EDB с использованием языка FASIT ALGOL 1. Были сделаны некоторые изменения в процедуре...\*\*\*.

### Свидетельство к алгоритму 59б [C2]

Алгоритм 59б не публикуется здесь, потому что соответствующий алгоритм 59 «Нахождение корней вещественного полинома по методу Бариса» был позднее заменен в журнале «CASM» более совершенным алгоритмом 340 («CASM», 68—11).

## АЛГОРИТМ 60б

### Вычисление интеграла по Ромбергу [D1]

Процедура-функция *rombint* вычисляет значение интеграла  $\int_a^b f(x) dx$

\* Указываются две поправки к алгоритму 58, учтенные при составлении алгоритма 58а. (Прим. ред.)

\*\* Рекомендуется восемь поправок к алгоритму 58, учтенных при составлении алгоритма 58а. (Прим. ред.)

\*\*\* Предлагается ряд поправок к алгоритму 58, учтенных при составлении алгоритма 58а. (Прим. ред.)

с погрешностью порядка  $2 \times k + 2$ , где  $k \geq 0$ . При увеличении  $k$  на единицу время вычисления, грубо говоря, удваивается.

```

real procedure rombint(f,a,b,k);
    value a,b,k; real a,b; integer k; real procedure f;
begin real d,s,h; integer m,i,j,n; array t[1:k+1];
    d:=b-a;
    t[1]:=(f(a)+f(b))/2;
    n:=1;
    for i:=1 step 1 until k do
        begin s:=0; n:=2×n;
            h:=d/n;
            for j:=1 step 2 until n do s:=s+f(a+j×h);
            t[i+1]:=(2×s/n+t[i])/2;
            m:=1;
            for j:=i step -1 until 1 do
                begin m:=4×m;
                    t[j]:=t[j+1] + (t[j+1]-t[j])/(m-1)
                end j
            end i;
        rombint:=t[1]×d
    end rombint;

```

#### Свидетельство к алгоритму 60б

Процедура-функция *rombint* алгоритма 60б является стереотипным переизданием процедуры-функции алгоритма 60а. В пояснительный текст внесены уточнения редакторского характера.

#### Свидетельство к алгоритму 60а

Алгоритм 60а получен в результате оптимизации и ординарной переработки алгоритма 60 (Вауег F. L. «САСМ», 1961, № 6). Оптимизация состояла в сокращении времени счета, благодаря тому, что после некоторых преобразований выражение  $2 \times n - 1$  в первом заголовке цикла по  $j$  было заменено на  $n$ .

После переработки проведено контрольное решение на ТА-1 для  $J = \int_2^5 \frac{x^3}{(4+x^2)^2} dx$ . Результат трансляции  $J = 0.0308139510$ . Ручным счетом получено значение  $J = 0.03078$ .

#### Подтверждение к алгоритму 60

Г. Тачер (Thacher H. C. «САСМ», 1962, № 3)

Эта процедура была переведена на язык транслятора АСТ-III для вычислительной машины Royal-Precision LGP-30. Эта система допускает семь значащих десятичных цифр.

Программа была использована для вычисления интегралов  $J_1 = \int_{0.01}^{1.1} x^n dx$

и  $J_2 = \int_{1.1}^{0.01} x^n dx$ . Результаты приведены в табл. 3.

Таблица 3

$k$	$n=0$	$n=+12$		$n=-1$		$n=-5$	
	$J_1$ и $ J_2 $	$J$	$J_2$	$J_1$	$J_2$	$J_1$	$J_2$
1	1.0899997	0.57076812	-0.57076842	19.641113	-19.641125	18.166655 $\times 10^8$	-18.166667 $\times 10^8$
2	1.0899997	0.30614608	-0.30614626	10.656923	-10.656929	8.4777719 $\times 10^8$	-8.4777766 $\times 10^8$
5	1.0899991	0.26555693	-0.26555818	4.9017590	-4.9017805	1.0408634 $\times 10^9$	-1.0408640 $\times 10^9$
10				4.7002345	-4.7004402	0.25000715 $\times 10^9$	-0.25000727 $\times 10^9$
12						0.24999291 $\times 10^9$	-0.25001311 $\times 10^9$
И. з.*	1.0900000	0.26555932	-0.26555932	4.7004831	-4.7004831	0.25000000 $\times 10^9$	-0.25000000 $\times 10^{9**}$

\* Истинное значение. (Прим. ред.)

\*\* В оригинале значения  $J_2$  для  $k=1$  и  $k=2$  перепутаны местами (Прим. ред.)

Очевидно, что процедура дает хорошие результаты для положительных степеней и что даже влияние близости особой точки для отрицательных степеней может быть преодолено.

Область применимости алгоритма могла бы быть расширена путем дополнения списка формальных параметров процедурой *check*, определяющей, получилась ли достаточная точность без проведения всех итераций. Эта процедура может иметь, например, следующую форму:

```

procedure check(t1,t2,f,exit);
  real t1,t2; integer f; label exit;
begin if abs((t2-t1)×f)/t1 < tolerance ∧ f > mink then
  goto exit
end

```

Глобальная переменная *tolerance*, которая является максимальной отрицательной разностью между приближенными значениями возрастающего порядка и минимально приемлемым порядком, должна выбираться программистом в соответствии с требованиями задачи. Проверка такого рода явно не так надежна, как предварительная оценка необходимого порядка, но часто служит приемлемым средством достижения цели.

Алгоритм квадратуры по Ромбергу описывается в работах [13i, 14i].

### Подтверждение к алгоритму 60

К. Х. Бухнер (Buchner К. Н. «САСМ», 1962, № 5)

С августа 1961 г. эта процедура успешно применялась к различным задачам на языке ФОРТРАН для машины IBM 1620. Вследствие своей изящности и экономного использования памяти метод вычисления интеграла по Ромбергу вытеснил другие методы в нашей библиотеке программ, например метод Ньютона — Котеса 10-го порядка.

Автор алгоритма 60 ссылается на работу Штифеля, в которой излагаются различные методы численного интегрирования, в том числе и алгоритм Ромберга.

### Замечание к алгоритму 60

Г. Тачер (Thacher Н. С. «САСМ», 1964, № 7)

Алгоритм квадратуры по Ромбергу с большим успехом использовался многими (Тачер Г. «Подтверждение к алгоритму 60» «САСМ»,

1962, № 3. Бухнер К. Х. «Подтверждение к алгоритму 60» «САСМ», 1962, № 5) и показал себя одним из удобных и надежных методов интегрирования. Поэтому имеет смысл указать, что алгоритм не вполне совершенен и что существует значительный класс подынтегральных функций, для которых экстраполяционные значения являются худшими оценками интеграла, чем соответствующие суммы формулы трапеций.

Точность процедуры Ромберга зависит от возможности разложения погрешности метода трапеций по степеням  $h^2$ , где  $h$  — величина шага. Разложением такого типа является ряд Эйлера — Маклорена. Другое выражение может быть получено с помощью рядов Фурье. Коэффициенты при  $h^{2r}$  в формуле Эйлера — Маклорена пропорциональны разности значений  $(2r+1)$ -й производной на двух концах интервала. Следовательно, любой интеграл, для которого нечетные производные подынтегральной функции на концах интервала либо равны нулю, либо равны между собой, не сходится при экстраполяции по Ромбергу. Простыми примерами таких интегралов являются интегралы периодических функций на интервале более длинном, чем период функции, и интегралы, для которых производные равны нулю на обоих концах интервала. Примером последнего типа служит интегральная аппроксимация модифицированной функции Ханкеля [34i]

$$e^x K_p(x) = \int_0^L e^{x(1-\cosh t)} \cosh(pt) dt,$$

где  $L$  берется таким большим, что частью интеграла от  $L$  до  $\infty$  можно пренебречь. Несколько других примеров даются под заголовком «Особые случаи» Бауэром, Рутисхаузером и Штифелем [15i]. Эта статья — наиболее обширное исследование метода Ромберга на английском языке.

Алгоритм непригоден также и тогда, когда разложение остаточного члена содержит другие степени  $h$  или хотя бы одну такую степень. Рутисхаузер [16i] рассматривает оценочные интегралы в форме

$$\int_0^a f(x) dx = \int_0^a (\varphi(x)/\sqrt{x}) dx.$$

Если такие интегралы вычисляются по формуле трапеций в предположении, что  $f(0)=0$ , то погрешность может быть выражена в форме  $\Sigma c_k h^{2k} + \sqrt{h} \Sigma d_k h^k$ . Хотя стандартная экстраполяция Ромберга непригодна для этой последовательности оценок, Рутисхаузер дает модифицированную процедуру, которая является эффективной. Экстраполяция непригодна также тогда, когда подынтегральная функция разрывна, хотя с точки зрения вычислительной математики это исключение тривиально.

Было также указано (McKeeman W. M. Личная переписка. Сент., 1963; Engeli M. Личная переписка. Янв., 1964), что процедура Ромберга может накапливать ошибки округления. Для большинства случаев применения потери ввиду их значительности предотвратить невозможно.

## АЛГОРИТМ 616

### Процедуры интервальной арифметики [A1]

Термин «интервальное число» был использован П. С. Двайером. Машинные процедуры интервальной арифметики разработал примерно в 1958 г. Р. Мор [11].

Если  $a \leq x \leq b$  и  $c \leq y \leq d$ , то процедура *rangsum* (сокращение от *range* — интервальная, *sum* — сумма) дает интервал  $[e, f]$  такой, что  $e \leq x + y \leq f$  \*. Из-за особенностей машинных операций (округления и усечения) машинные суммы  $a + c$  и  $b + d$  могут не обеспечивать сохранности концевых точек выходного интервала. Поэтому для процедуры *rangsum* нужна нелокализованная вещественная процедура *correcsum* (сокращение от *correction* — поправка и *sum* — сумма), которая компенсирует погрешности машинной арифметики. Тело процедуры *correcsum* будет зависеть от типа машины, для которой эта процедура написана, поэтому она здесь не приводится (однако ниже дается пример такой процедуры). Предполагается, что процедура *correcsum* имеет вещественные параметры  $v$  и  $u$  и целый параметр  $i$  и что ей сопутствует нелокализованная вещественная процедура *correction*, которая дает верхнюю границу величины погрешности, содержащейся в машинном представлении числа. Выходное значение функции *correcsum* дает левый конец интервала, выдаваемого процедурой *rangsum*, когда *correcsum* вызывается с параметром  $i = -1$ , и правый конец, когда  $i = 1$ .

Процедуры *rangsub*, *rangpro* и *rangdiv* (сокращение от *subtraction* — вычитание, *product* — произведение, *division* — деление) предназначены для остальных основных операций интервальной арифметики. Процедура *rangsqr* (сокращение от *square* — квадрат) дает интервал, внутри которого должен лежать квадрат интервального числа.

Процедуры *rangsumc*, *rangsubc*, *rangproc* и *rangdivc* предусмотрены для операций с комплексными интервальными аргументами, т. е. вещественные и мнимые части этих аргументов являются интервальными числами.

```
procedure rangsum(a,b,c,d) result: (e,f);  
  value a,b,c,d; real a,b,c,d,e,f;  
begin e:=correcsum(a,c,-1); f:=correcsum(b,d,1) end rangsum;
```

Процедура *rangsub* вычисляет разность интервальных чисел  $\{a, b\}$  и  $\{c, d\}$ , т. е. находит интервал  $[e, f]$  такой, что  $e \leq x - y \leq f$  для всех  $x$  и  $y$ , удовлетворяющих условиям  $a \leq x \leq b$  и  $c \leq y \leq d$ . В теле процедуры *rangsub* процедура *rangsum* не локализована.

```
procedure rangsub(a,b,c,d) result: (e,f);  
  value a,b,c,d; real a,b,c,d,e,f;  
rangsum(a,b,-d,-c,e,f);
```

Процедура *rangpro* находит произведение  $\{e, f\}$  интервальных чисел  $\{a, b\}$  и  $\{c, d\}$ , т. е. находит интервал  $[e, f]$  такой, что  $e \leq x \times y \leq f$  для всех  $x$  и  $y$ , удовлетворяющих условиям  $a \leq x \leq b$  и  $c \leq y \leq d$ . В теле процедуры *rangpro* имеется глобальная процедура *correcpro*, аналогичная вышеописанной процедуре *correcsum* (см. нижеследующий пример).

---

\* Другими словами это можно выразить так, что процедура *rangsum* находит сумму  $\{e, f\}$  двух интервальных чисел  $\{a, b\}$  и  $\{c, d\}$ . (Прим. ред.)



```

procedure rangpro(a,b,c,d) result: (e,f);
  value a,b,c,d; real a,b,c,d,e,f;
begin real r;
  e:=f:=a×c;
  for r:=a×d,b×c,b×d do
    if r<e then e:=r else
      if r>f then f:=r;
  e:=correcpro(e,—1);
  f:=correcpro(f,1)
end rangpro;

```

Процедура *rangdiv* находит частное от деления интервального числа  $\{a,b\}$  на интервальное число  $\{c,d\}$ , т. е. находит интервал  $[e,f]$  такой, что  $e \leq x/y \leq f$  для всех  $x$  и  $y$ , удовлетворяющих условиям  $a \leq x \leq b$  и  $c \leq y \leq d$ . В теле процедуры *rangdiv* используется глобальная вещественная процедура *correcdiv*, аналогичная (возможно идентичная) процедуре *correcpro*. Если интервальный делитель содержит нуль, то осуществляется выход из программы к нелокализованной метке *signal61*.

```

procedure rangdiv(a,b,c,d) result: (e,f);
  value a,b,c,d; real a,b,c,d,e,f;
begin if c≤0 ∧ d≥0 then go to signal61;
  e:=f:=a/c;
  for r:=a/d,b/c,b/d do
    if r<e then e:=r else
      if r>f then f:=r;
  e:=correcdiv(e,—1); f:=correcdiv(f,1)
end rangdiv;

```

Процедура *rangdiv* может быть записана короче, но (на многих машинах) с несколько бóльшим временем выполнения, как это указано ниже.

```

procedure rangdiv2(a,b,c,d) result: (e,f);
  value a,b,c,d; real a,b,c,d,e,f;
begin if c≤0 ∧ d≥0 then go to signal61;
  rangpro(a,b,1/d,1/c,e,f);
  e:=correcdiv(e,—1); f:=correcdiv(f,1)
end rangdiv2;

```

Процедура *rangsqr* находит квадрат интервального числа  $\{a,b\}$ , т. е. находит интервал  $[e,f]$ , такой что  $e \leq x^2 \leq f$  для всех  $x$ , удовлетворяющих условию  $a \leq x \leq b$ . В теле этой процедуры используется нелокализованная процедура *correcpro*.

```

procedure rangsqr(a,b) result: (e,f);
  value a,b; real a,b,e,f;
begin e:=a×a; f:=b×b;
  if a<0 then
    begin e:= if b<0 then f else 0;
      if —a>b then f:=a×a
    end;
  e:=correcpro(e,—1); f:=correcpro(f,1)
end rangsqr;

```

Процедура *rangsumc* находит сумму комплексных интервальных чисел  $\{a_1,a_2,b_1,b_2\}$  и  $\{c_1,c_2,d_1,d_2\}$ , т. е. находит интервалы  $[e_1,e_2]$  и

$[f1, f2]$  такие, что  $e1 \leq e \leq e2$  и  $f1 \leq f \leq f2$  для всех  $e$  и  $f$ , удовлетворяющих равенству  $e + fi = (a + bi) + (c + di)$ , где  $a1 \leq a \leq a2$ ,  $b1 \leq b \leq b2$ ,  $c1 \leq c \leq c2$  и  $d1 \leq d \leq d2$ .

Процедура *rangsumc* содержит глобальную процедуру *rangsum*.

```

procedure rangsumc(a1,a2,b1,b2,c1,c2,d1,d2) result: (e1,e2,f1,f2);
    value a1,a2,b1,b2,c1,c2,d1,d2;
    real a1,a2,b1,b2,c1,c2,d1,d2,e1,e2,f1,f2;
begin rangsum(a1,a2,c1,c2,e1,e2);
    rangsum(b1,b2,d1,d2,f1,f2)
end rangsumc;

```

Процедура *rangsubc* производит вычитание комплексных интервальных чисел, т. е. находит интервалы  $[e1, e2]$  и  $[f1, f2]$  вещественной и мнимой частей разности комплексных чисел  $(a + bi) - (c + di) = e + fi$  по данным интервалам  $a1 \leq a \leq a2$ ,  $b1 \leq b \leq b2$ ,  $c1 \leq c \leq c2$  и  $d1 \leq d \leq d2$ .

В теле процедуры *rangsubc* содержится глобальная процедура *rangsumc*.

```

procedure rangsubc(a1,a2,b1,b2,c1,c2,d1,d2) result: (e1,e2,f1,f2);
    value a1,a2,b1,b2,c1,c2,d1,d2;
    real a1,a2,b1,b2,c1,c2,d1,d2,e1,e2,f1,f2;
rangsumc(a1,a2,b1,b2,-c2,-c1,-d2,-d1,e1,e2,f1,f2);

```

Процедура *rangproc* производит умножение комплексных интервальных чисел, т. е. определяет интервалы  $[e1, e2]$  и  $[f1, f2]$  вещественной и мнимой частей произведения  $(a + bi) \times (c + di) = e + fi$  по данным интервалам  $a1 \leq a \leq a2$ ,  $b1 \leq b \leq b2$ ,  $c1 \leq c \leq c2$  и  $d1 \leq d \leq d2$ . В теле этой процедуры используются глобальные процедуры *rangpro*, *rangsub* и *rangsum*.

```

procedure rangsubc(a1,a2,b1,b2,c1,c2,d1,d2) result: (e1,e2,f1,f2);
    value a1,a2,b1,b2,c1,c2,d1,d2;
    real a1,a2,b1,b2,c1,c2,d1,d2,e1,e2,f1,f2;
begin real p1,p2,q1,q2;
    rangpro(a1,a2,c1,c2,p1,p2);
    rangpro(b1,b2,d1,d2,q1,q2);
    rangsub(p1,p2,q1,q2,e1,e2);
    rangpro(a1,a2,d1,d2,p1,p2);
    rangpro(b1,b2,c1,c2,q1,q2);
    rangsum(p1,p2,q1,q2,f1,f2)
end rangproc;

```

Процедура *rangdivc* производит деление комплексного интервального числа  $\{a1, a2, b1, b2\}$  на комплексное интервальное число  $\{c1, c2, d1, d2\}$ , т. е. определяет интервалы  $[e1, e2]$  и  $[f1, f2]$  вещественной и мнимой частей частного  $e + fi = (a + bi) / (c + di)$  по данным интервалам  $a1 \leq a \leq a2$ ,  $b1 \leq b \leq b2$ ,  $c1 \leq c \leq c2$  и  $d1 \leq d \leq d2$ . В теле процедуры *rangdivc* используются глобальные процедуры *rangsqr*, *rangproc*, *rangsum* и *rangdiv*.

```

procedure rangdivc(a1,a2,b1,b2,c1,c2,d1,d2) result: (e1,e2,f1,f2);
    value a1,a2,b1,b2,c1,c2,d1,d2;
    real a1,a2,b1,b2,c1,c2,d1,d2,e1,e2,f1,f2;
begin real p1,p2,q1,q2,r1,r2;
    rangsqr(c1,c2,p1,p2);
    rangsqr(d1,d2,q1,q2);

```

```

rangsum(p1,p2,q1,q2,r1,r2);
rangproc(a1,a2,b1,b2,c1,c2,—d2,—d1,p1,p2,q1,q2);
rangdiv(p1,p2,r1,r2,e1,e2);
rangdiv(q1,q2,r1,r2,f1,f2)
end rangdivc;

```

Пример. Нижеследующие операции *correction*, *correcsum* и *correcpro* предназначены для выполнения арифметических операций с единой точностью над нормализованными числами с плавающей запятой. Они пригодны для машин, в которых мантисса числа с плавающей запятой изображается  $s$  значащими цифрами  $b$ -й системы счисления. Предельные возможности машины или требования пользователей ограничиваются интервалом  $b^m \leq |p| < b^{n+1}$  для некоторых целых  $m$  и  $n$ . Если число  $p$  не лежит в указанном интервале, то из тела процедуры *correction* осуществляется выход к глобальной метке *signalcorr*.

Процедура *correction* дает верхнюю границу погрешности в машинном представлении числа. Когда эта процедура составляется для конкретной машины, то в ней переменные  $b$ ,  $s$ ,  $m$  и  $n$  нужно заменить соответствующими целыми числами.

```

real procedure correction(p);
  value p; real p;
begin integer i;
  p:=abs(p);
  for i:=m step 1 until n do
    if  $b \uparrow i \leq p \wedge p < b \uparrow (i+1)$  then
      begin correction:= $b \uparrow (i+1-s)$ ; go to fin end;
  go to signalcorr;
fin: end correction;

```

Процедура *correcsum* дана в качестве примера возможной процедуры для использования на машинах, которые (если сложение выполняется с плавающей запятой) просто отбрасывают цифры младших неиспользуемых разрядов. Здесь не делается никакой попытки проверить, не являются ли все отбрасываемые цифры нулями. В теле процедуры *correcsum* используется глобальная процедура *correction*.

```

real procedure correcsum(w,v,i);
  value w,v,i; real w,v; integer i;
begin real r,cw,cv,cr,m;
  correcsum:=r:=w+v; m:=sign(w)×sign(v);
  if  $m=0 \vee i \times m \times r < 0$  then go to fin;
  m:=cw:=correction(w);
  cv:=correction(v);
  cr:=correction(r);
  if  $cv=cw \wedge cr \leq cw$  then go to fin;
  if  $cv > m$  then m:=cv;
  if  $cr > m$  then m:=cr;
  correcsum:=r+i×m;
fin: end correcsum;

```

Процедура *correcpro* предназначена для машин, у которых младшие разряды произведения, выходящие за пределы разрядной сетки, просто отбрасываются. Процедура *correction* здесь не локализована.

```

real procedure correcpro(p,i);
  value p,i; real p; integer i;

```

correcpro:= if  $p \times i \leq 0$  then  $p$  else  $p + i \times \text{correction}(p)$ ;

Хотя обычно операции с округлением предпочтительнее операций с усечением, для этих процедур операции с усечением приводят к более точным границам, чем операции с округлением.

### Свидетельство к алгоритму 61б

Алгоритм 61б отличается от алгоритма 61а только тем, что 1) в процедуру *rangsqr* внесено исправление согласно «Замечанию к алгоритму 61а» Ю. И. Маркова [25, стр. 179]; 2) идентификатор второго варианта процедуры *rangdiv* был заменен на *rangdiv2*; 3) процедура *rangdiv* была записана в более краткой и наглядной форме.

Процедуры *rangdiv*, *rangdiv2* и *rangsqr* были транслированы в системе ТА-1М для машины М-220 и дали правильные результаты для следующих примеров:

1) процедуры *rangdiv* (в варианте, приведенном в алгоритме 61а) и *rangdiv2* для исходных данных

<i>a</i>	2	—2	—5	2	—2	—5
<i>b</i>	5	5	—2	5	5	—2
<i>c</i>	6	6	6	—9	—9	—9
<i>d</i>	9	9	9	—6	—6	—6

2) процедура *rangsqr* для исходных данных

<i>a</i>	2	—2	—5	—5
<i>b</i>	5	5	2	—2

Проверка правильности результатов производилась с помощью следующих операторов:

1) для процедур *rangdiv* и *rangdiv2*

```
for x:=a step 1 until b do
  for y:=c step 1 until d do
    if  $x/y < e \vee x/y > f$  then output('E',x,y,e,f);
```

2) для процедуры *rangsqr*

```
for x:=a step 1 until b do
  if  $x \uparrow 2 < e \vee x \uparrow 2 > f$  then output('E',x,e,f);
```

Поскольку машина М-220 округляет числа не так, как это предполагается в процедурах *correcsum* и *correcpro*, то последние отладкой проверены не были. Вследствие этого из процедур *rangdiv*, *rangdiv2* и *rangsqr* перед отладкой были вычеркнуты операторы  $e := \text{correcdiv}(e, -1)$ ;  $f := \text{correcdiv}(f, 1)$ ;  $e := \text{correcpro}(e, -1)$ ; и  $f := \text{correcpro}(f, 1)$ , т. е. отладка производилась с точностью до погрешностей округления.

Остальные процедуры алгоритма 61б вследствие их простоты и очевидности на машине не транслировались.

## Свидетельство к алгоритму 61a

Процедуры алгоритма 61a получены в результате сокращения и ординарной переработки соответствующих процедур алгоритма 61 (Gibb A. «САСМ», 1961, № 7). Процедуры *rangpro*, *rangdiv* и *rangsqr*, по существу, написаны заново.

Алгоритм проверен вручную для всех возможных сочетаний интервальных чисел.

## АЛГОРИТМ 626

### Последовательность присоединенных функций Лежандра второго рода [S16]

Процедура *legendr2* помещает последовательность значений  $Q_n^m(x)$  в массив *q* для целых значений *n* от 0 до *nmax* для частного значения *m* и некоторого значения *x*, которое должно быть вещественным, если  $ri=0$ , или чисто мнимым в противном случае. Массив *r* содержит ряд отношений последовательных значений *q*. Эти отношения могут быть особенно ценными, когда наименьшее  $Q_n^m(x)$  так мало, что выходит за пределы машинного представления чисел (например,  $10^{-25}$ , когда наименьшее представимое в машине число равно  $10^{-18}$ ).

Выход к нелокализованной метке *signal62* осуществляется, когда  $Q_n^m(x)$  теряет смысл (мнимые значения *x* не могут быть отрицательными, а вещественные значения *x* не могут быть менее 1).

Значения  $Q_n^m(x)$  могут быть легко вычислены с помощью гипергеометрических рядов (см. [1], с. 468), если *x* не слишком мало и  $(n-m)$  не слишком велико. Если же вещественное *x* близко к единице или мнимое *x* близко к нулю, то значения  $Q_n^m(x)$  получают по соответствующей последовательности значений  $P_n^m(x)$ . Потеря значащих цифр случается для *x* таких малых как 1.1, если  $n > 10$ . Потеря значащих цифр представляет большую трудность в использовании конечных полиномиальных представлений при  $n > m$ . Однако процедура *legendr2* была проверена как для больших, так и для малых *x* и *n*. В процедуре *legendr2* используется константа  $\pi/2 = 1.57 \dots$

```
procedure legendr2(m,nmax,x,ri) result: (r,q);
  value m,nmax,x,ri; real x; integer m,nmax,ri; array r,q;
begin real t,q0,s; integer i,n;
  n:= if nmax>13 then nmax+7 else 20;
  if ri=0 then
    begin
      if x=1 then go to signal62;
      if m=0 then q[0]:=0.5*ln((x+1)/(x-1)) else
        begin q[0]:=t:=-1/sqrt(x*x-1);
          q0:=0; t:=2*x*t;
          for i:=2 step 1 until m do
            begin s:=(i-1)*t*q[0]+(3*i-i*i-2)*q0;
              q0:=q[0]; q[0]:=s;
            end i
          end m;
          r[n+1]:=x-sqrt(x*x-1);
```

```

    for i:=n step—1 until 1 do
        r[i]:=(i+m)/((2×i+1)×x+(m—i—1)×r[i+1]);
    go to fin
end ri;
if m=0 then
    q[0]:=if x<0.5 then arctan(x)—1.570796327 else—arctan(1/x)
else
    begin q[0]:=t:=1/sqrt(x×x+1);
        q0:=0; t:=2×x×t;
        for i:=2 step 1 until m do
            begin s:=(i—1)×t×q[0]+(3×i+i×i—2)×q0;
                q0:=q[0]; q[0]:=s
            end i
        end m;
        r[n+1]:=x—sqrt(x×x+1);
        for i:=n step —1 until 1 do
            r[i]:=(i+m)/((i—m+1)×r[i+1]—(2×i+1)×x);
        for i:=1 step 2 until nmax do r[i]:=—r[i];
fin:   for i:=1 step 1 until nmax do q[i]:=q[i—1]×r[i]
end legendr2;

```

#### Свидетельство к алгоритму 62б

Алгоритм 62б получен из алгоритма 62а путем внесения в него следующих поправок:

1. На 32-й странице 3-я строка сверху [23] вместо

**begin** q[0]:=t:=1/sqrt(x×x—1);

должно быть

**begin** q[0]:=t:=1/sqrt(x×x+1);

2. Для ликвидации деления на нуль при  $ri=m=x=0$  оператор

**if** x=1 **then go to** signal62;

**был** перенесен из 15-й строки процедуры *legendr2* в ее 7-ю строку.

Алгоритм 62б был транслирован в системе ТА-1М для машины М-220. Результаты трансляции для  $ri=0$  сведены в табл. 4 и 5, а для  $ri=1$  — в табл. 6 и 7. Контрольные значения в табл. 4—7 взяты из сборника таблиц Колумбийского университета [32] (США).

Таблица 4

x	n=1			
	m=0		m=1	
	Результаты	Контрольные значения	Результаты	Контрольные значения
1.1	0.67448734	0.674487	—0.17028090 <sub>10</sub> 1	—0.170281 <sub>10</sub> 1
1.5	0.20707843	0.207078	—0.44193764	—0.441938
3.0	0.39720770 <sub>10</sub> —1	0.397208 <sub>10</sub> —1	—0.80402028 <sub>10</sub> —1	—0.804020 <sub>10</sub> —1
10.0	0.33534773 <sub>10</sub> —2	0.335348 <sub>10</sub> —2	—0.67137103 <sub>10</sub> —2	—0.671371 <sub>10</sub> —2

Таблица 5.

$x$	$n=5$			
	$m=0$		$m=4$	
	Результаты	Контрольные значения	Результаты	Контрольные значения
1.5	$0.24668237_{10}-2$	$0.246682_{10}-2$	$0.11134250_{10}2$	$0.111343_{10}2$
3.0	$0.19107860_{10}-4$	$0.191079_{10}-4$	$0.62220398_{10}-1$	$0.622204_{10}-1$
10.0	$0.11732753_{10}-7$	$0.117328_{10}-7$	$0.35700357_{10}-4$	$0.357004_{10}-4$

Таблица 6

$x$	$n=1$			
	$m=0$		$m=1$	
	Результаты	Контрольные значения	Результаты	Контрольные значения
1.0	$-0.21460183$	$-0.214602$	$0.40361395$	$0.403614$
3.0	$-0.34478336_{10}-1$	$-0.347483_{10}-1$	$0.68781292_{10}-1$	$0.687813_{10}-1$
5.0	$-0.13022200_{10}-1$	$-0.130222_{10}-1$	$0.25943135_{10}-1$	$0.259431_{10}-1$

Таблица 7

$x$	$n=3; m=2$	
	Результаты	Контрольные значения
1.0	$-0.43805509$	$-0.438055$
3.0	$-0.12250521_{10}-1$	$-0.122505_{10}-1$
5.0	$-0.17352158_{10}-2$	$-0.173522_{10}-2$

### Свидетельство к алгоритму 62а

Алгоритм 62а получен в результате исправления, сокращения и ординарной переработки алгоритма 62 (Herndon J. R. «САСМ», 1961, № 7).

Кроме опечаток, указанных в нижеследующем замечании Дж. Херндона («САСМ», 1961, № 12), в алгоритме 62 было исправлено следующее.

1. Параметры,  $m$ ,  $nmax$ ,  $ri$  и переменные  $i$ ,  $n$  были отнесены к типу **integer**.

2. Идентификатор  $log$  был заменен на  $ln$ .

3. Оператор **if**  $x=1$  **then**  $Q[0]:=9.9\uparrow 45$ , где число  $9.9\uparrow 45$  изображает «бесконечность», был заменен оператором



if  $x=1$  then go to signal62.

4. В двух местах произведение  $3i$  было заменено на  $3 \times i$ .

### Замечание к алгоритму 62

Дж. Р. Херндон (Herdon J. R. «САСМ», 1961, № 12)

При просмотре алгоритма 62 были найдены ошибки...\*

Эта процедура получена из стандартной рекуррентной формулы

$$(n+m-1)Q_{n-2}^m = (2n-1)xQ_{n-1}^m - (n-m)Q_n^m.$$

Обратим и умножим на  $(n+m-1)Q_{n-1}^m$ . Тогда

$$\frac{Q_{n-1}^m}{Q_{n-2}^m} = \frac{n+m-1}{(2n-1)x - (n-m)Q_n^m/Q_{n-1}^m} \quad \text{или} \quad R_{n-1}^m = \frac{n+m-1}{(2n-1)x - (n-m)R_n^m}.$$

Анализ показывает, что для большого  $n$  эта бесконечная непрерывная дробь должна быть вычислена примерно до восьми членов для получения восьми разрядов точности, если последний член оценивается асимптотическим значением, получаемым следующим образом:

$$R_{n-1}^m = R_n^m, \quad \lim_{n \rightarrow \infty} R_n^m = x \pm \sqrt{x^2 - 1}.$$

Выбирается знак минус, так как в общем случае  $Q_n^m < Q_{n-1}^m$ .

Значение  $Q_0^m(x) = 1/2 \ln[(x+1)/(x-1)]$ , тогда как  $Q_1^m(x) = xQ_0^m(x) - 1$ , и  $Q_1^0(x) = -1/\sqrt{x^2 - 1}$ .

Другие значения получаются путем использования отношений  $R_n^m(x)$  и (или) рекуррентной формулы

$$Q_n^m = -(2(m-1)x/\sqrt{x^2-1})Q_n^{m-1} + (n-m+2)(n+m-2)Q_n^{m-2}.$$

Получение выражения для  $Q_0^m(ix)$  нетривиально и производится следующим образом:

$$i \times Q_0^m(ix) = \frac{1}{2} \ln \frac{ix+1}{ix-1} = \frac{1}{2} \ln \left[ -\frac{x^2-1}{x^2+1} + \frac{2x}{x^2+1} \right]^{**}.$$

$$e^{a+ib} = e^a e^{ib} = e^a (\cos b + i \times \sin b).$$

Поэтому  $\operatorname{tg} b = -2x/(1-x^2)$  и  $Q_0^m(ix) = (\operatorname{arctg} x - \pi/2)i$ .

## АЛГОРИТМ 63б

### Разделение элементов массива [M1]

Процедура *partition* (*partition* — разделение) выбирает значение  $x$  случайного элемента из массива  $a[m:n]$  и переставляет значения элементов этого массива таким образом, чтобы существовали целые  $i$  и  $j$

\* Далее указываются две опечатки в алгоритме 62, исправленные в алгоритме 62а. (Прим. ред.)

\*\* По-видимому, в оригинале здесь ошибка. Нужно

$$= \frac{1}{2} \ln \left[ \frac{x^2-1}{x^2+1} - \frac{2xi}{x^2+1} \right].$$

со следующими свойствами:  $m \leq i < j \leq n$  при условии  $m < n$ . Далее если  $m \leq p < i$ ,  $i < r < j$ ,  $j \leq q \leq n$ , то  $a[p] \leq a[r] \leq a[q]$ .

Эта процедура использует нелокализованную целую процедуру-функцию  $random(m, n)$ , которая с равной вероятностью выбирает случайное целое число  $p$  между  $m$  и  $n$ , и нелокализованную процедуру  $exchange$ , описание которой имеет вид

```
procedure exchange(a,b);  
    value a,b; real a,b;  
begin real s; s:=a; a:=b; b:=s end;
```

Выходные параметры процедуры  $partition$ :  $a$ ,  $i$  и  $j$

```
procedure partition(m,n) data result: (a) result: (i,j);  
    value m,n; integer m,n,i,j; array a;  
begin real x; integer p;  
    p:=random(m,n); x:=a[p];  
    i:=n; j:=m;  
n01: for j:=j step 1 until n do  
    if a[j]>x then go to n02;  
    j:=n;  
n02: for i:=i step -1 until m do  
    if a[i]<x then go to n03;  
    i:=m;  
n03: if i>j then  
    begin exchange(a[i],a[j]);  
    i:=i-1; j:=j+1;  
    go to n01  
    end;  
    if j<p then  
    begin exchange(a[j],a[p]); j:=j+1 end else  
    if i>p then  
    begin exchange(a[i],a[p]); i:=i-1 end  
end partition;
```

### Свидетельство к алгоритму 63б

Процедура  $partition$  алгоритма 63б является стереотипным переизданием процедуры  $partition$  алгоритма 63а, за исключением первой строки, которая в алгоритме 63а имела вид

```
procedure partition(a,m,n,i,j);
```

Это изменение сделано ради удобства пользования процедурой. В пояснительный текст алгоритма также внесены некоторые изменения, повышающие ясность изложения.

Алгоритм 63б был транслирован в системе БЭСМ—АЛГОЛ машины БЭСМ-6 для примеров, приведенных в нижеследующем «Свидетельстве к алгоритму 63а», т. е. оператор  $p:=random(m,n)$  заменялся (ради удобства отладки) оператором  $p:=3$  или  $p:=4$ . Были получены такие же результаты, какие указаны в «Свидетельстве к алгоритму 63а».

Переводы «Подтверждений к алгоритмам 63, 64 и 65» помещены после алгоритма 65б.

### Свидетельство к алгоритму 63а

Алгоритм 63а получен в результате ординарной переработки алгоритма 63 (Н о а г е С. А. R. «САСМ», 1961, № 7).

Алгоритм 63а проверен вручную для  $a=(4, 1, 3, 5, 2)$ ;  $m=1$ ,  $n=5$  и  $p=3$  и 4. Получены результаты  
 для  $p=3$   $a=(2, 1, 3, 5, 4)$ ;  $i=2$ ;  $j=4$ ;  
 для  $p=4$   $a=(4, 1, 3, 2, 5)$ ;  $i=4$ ;  $j=5$ .

## АЛГОРИТМ 64б

### Быстрая сортировка [рекурсивная процедура] [М1]

Процедура *quicksort* (*quick* — быстрая, *sort* — сортировка) — это очень быстрый и удобный метод сортировки массива в памяти машины с генератором случайных чисел. Среднее число сравнений равно  $2(n-m)\ln(n-m)$ , а среднее число парных перестановок равно  $\frac{1}{3}(n-m)\ln(n-m)$ . Смысл параметров  $a$ ,  $m$  и  $n$  тот же, что и в алгоритме 63б.

```
procedure quicksort(m,n) dataresult:(a);
  value m,n; integer m,n; array a;
  if m < n then
    begin integer i,j;
      partition(m,n,a,i,j);
      quicksort(m,i,a);
      quicksort(j,n,a)
    end quicksort;
```

### Свидетельство к алгоритму 64б

Алгоритм 64б получен в результате модификации алгоритма 64а для повышения удобства пользования им. В первой строке описания процедуры *quicksort* был изменен порядок следования параметров, соответственно этому изменились и два рекурсивных обращения к процедуре *quicksort* в ее собственном теле. Кроме того, в связи с изменением порядка следования параметров процедуры *partition* алгоритма 63б изменилось и обращение к ней из тела процедуры *quicksort*.

Алгоритм 64б транслирован в системе БЭСМ—АЛГОЛ на машине БЭСМ-6 совместно с процедурой *partition* алгоритма 63б, в которой оператор  $p:=\text{random}(m,n)$  был заменен (для удобства отладки) на оператор  $p:=(m+n)/2$ . Алгоритм дал правильный результат при задании параметров  $m=1$ ,  $n=5$  и  $a=(4, 1, 3, 5, 2)$ .

Переводы «Подтверждений к алгоритмам 63, 64 и 65» помещены после алгоритма 65б.

### Свидетельство к алгоритму 64а

Алгоритм 64а получен в результате сокращения и ординарной переработки алгоритма 64 (Но аге С. А. Р. «САСМ», 1961, № 7). Данный алгоритм использует свойство рекурсивности процедур и может работать только на соответствующих трансляторах (например, на ТА-2 [18]).

## АЛГОРИТМ 656

### Поиск элемента в сортируемом массиве [рекурсивная процедура] [M1]

Процедура *find* (*find* — поиск) присваивает переменной  $a[k]$  такое значение, которое она имела бы, если бы массив  $a[m:n]$  был рассортирован. После выполнения процедуры *find* массив  $a$  будет частично рассортирован и последующие обращения к процедуре *find* будут выполняться быстрее, чем первое.

```
procedure find(m,n,k) dataresult:(a);  
  value m,n,k; integer m,n,k; array a;  
  if m < n then  
    begin integer i,j;  
      partition(m,n,a,i,j);  
      if k ≤ i then find(m,i,k,a) else  
        if k ≥ j then find(j,n,k,a)  
    end find;
```

#### Свидетельство к алгоритму 656

Алгоритм 656 получен в результате модификации алгоритма 65а для повышения удобств пользования им. В первой строке описания процедуры *find* был изменен порядок следования параметров, соответственно этому изменились и два рекурсивных обращения к процедуре *find* в ее собственном теле. Кроме того, в связи с изменением порядка следования параметров процедуры *partition* алгоритма 63б изменилось и обращение к ней из тела процедуры *find*.

Алгоритм 656 был транслирован в системе БЭСМ—АЛГОЛ машины БЭСМ-6 совместно с процедурой *partition* алгоритма 63б, в которой оператор  $p := \text{random}(m,n)$  был заменен (для упрощения отладки) на оператор  $p := (m+n)/2$ . Алгоритм дал правильные результаты при задании параметров  $m=1$ ,  $n=5$ ,  $a=(4, 1, 3, 5, 2)$  и  $k=1, 2, 3, 4, 5$ . После обращения к процедуре *find* для первых трех значений  $k$  массив  $a$  имел вид  $(1, 2, 3, 5, 4)$ , а для  $k=4$  и  $5$  — вид  $(1, 2, 3, 4, 5)$ , т. е., элемент  $a[k]$  действительно находился на месте  $k$  в массиве  $a$ .

#### Свидетельство к алгоритму 65а

Алгоритм 65а получен в результате сокращения и ординарной переработки процедуры, приведенной в нижеследующем «Подтверждении» Б. Рандела и Л. Рассела («CACM», 1963, № 8). Данный алгоритм можно использовать только на трансляторах, допускающих рекурсивные процедуры.

#### Подтверждение к алгоритмам 63, 64 и 65

Дж. Хилмор (Hillmore J. S. «CACM», 1962, № 8)

Тело процедуры *find* было исправлено на ...\*

Эти три процедуры были затем успешно проверены на вычислительной машине National Elliott 803, использующей транслятор с языка Elliott ALGOL.

---

\* Предлагается исправленный вариант алгоритма 65. (Прим. ред.)

Авторская оценка  $\frac{1}{3}(n-m)\ln(n-m)$  для числа парных перестановок, необходимого для сортировки случайной последовательности, была найдена правильной. Однако число сравнений оказалось в общем меньше чем  $2(n-m)\ln(n-m)$  даже без модификации, указанной ниже.

Эффективность процедуры *quicksort* была повышена путем изменения ее тела на \*

```
begin integer i,j;
  if m < n-1 then
    begin partition(m,n,a,i,j);
      quicksort(m,i,a);
      quicksort(j,n,a)
    end else
      if m = n-1 then
        begin if a[n] < a[m] then exchange(a[m],a[n]) end
      end quicksort;
```

#### Подтверждение к алгоритмам 63, 64 и 65

Б. Рандел и Л. Дж. Рассел (Randell B., Russell L. J.  
«CACM», 1963, № 8)

Алгоритмы 63, 64 и 65 были проверены с использованием транслятора Pegasus ALGOL 60, разработанного в Англии в г. Хатфилде компаний Havilland Aircraft.

В алгоритмах 63 и 64 не потребовалось никаких изменений, они работали удовлетворительно... \*\*

#### АЛГОРИТМ 666

##### Обращение симметричной матрицы методом квадратных корней [F1]

Процедура *invers66* обращает симметричную матрицу  $a[1:n, 1:n]$ , для которой ведущие миноры не нулевые, с помощью упрощенного варианта метода квадратных корней. Эта процедура заменяет  $n(n+1)/2$  диагональных и наддиагональных элементов матрицы  $a$  элементами матрицы  $a^{-1}$ , оставляя поддиагональные элементы неизменными.

Преимущества этого метода состоят в том, что требуется только  $n$  рабочих ячеек, не нужна никакой единичной матрицы, не извлекаются никакие квадратные корни, выполняется только  $n$  делений. Когда  $n$  возрастает, то число умножений приближается к  $n^3/2$ . Если  $a[1,1]=0$ , то осуществляется выход к нелокализованной метке *signal66*. (Подробное описание метода см. [4, с. 177—179].)

```
procedure invers66(n) dataresult: (a);
  value n; integer n; array a;
begin real y,p; integer i,j,k; array v[1:n-1];
  for k:=1 step 1 until n do
    begin if a[1,1]=0 then go to signal66;
      p:=1/a[1,1];
      for i:=2 step 1 until n do v[i-1]:=a[1,i];
```

\* Для удобства пользования нижеследующие операторы модифицированы в соответствии с алгоритмами 636 и 656. (Прим. ред.)

\*\* Далее указываются некоторые поправки и дается новый вариант алгоритма 65, использованный для алгоритма 65а. (Прим. ред.)

```

    for i:=1 step 1 until n-1 do
      begin a[i,n]:=y:=-v[i]×p;
        for j:=i step 1 until n-1 do
          a[i,j]:=a[i+1,j+1]+v[j]×y
        end j;
      a[n,n]:=-p
    end k;
    for i:=1 step 1 until n do
      for j:=i step 1 until n do a[i,j]:=-a[i,j]
    end invers66;

```

#### Свидетельство к алгоритму 66б

Алгоритм 66б получен из алгоритма 66а в результате одного очевидного усовершенствования. Ради экономии машинного времени заголовков цикла

```
for k:=0 step 1 until n-1 do
```

был заменен заголовком

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

Алгоритм 66а был подтвержден также и расчетами на машинах БЭСМ-6 (см. «Подтверждение к алгоритмам 4а, 7а, ..., 238а» Л. С. Кривонос и З. А. Шиншиновой [38]) и Урал-2 (см. «Подтверждение к алгоритмам 1а, 7а, ..., 143а» и И. Р. Гитмана [25]).

#### Свидетельство к алгоритму 66а

Алгоритм 66а получен в результате исправления и ординарной переработки алгоритма 66 (Caffrey J. «САСМ», 1961, № 7). Проведено контрольное решение на трансляторе ТА-1. Получены хорошие результаты, которые указаны в «Свидетельстве к алгоритму 58а».

#### Подтверждение к алгоритму 66

Б. Рандел, К. Г. Бройден (Randell B., Broyden C. G. «САСМ», 1962, № 1)

Эта процедура была транслирована с использованием транслятора DEUCE—ALGOL. Потребовались следующие исправления...\*

Программа, транслированная с использованием 20-разрядной мантиссы с плавающей запятой, была проверена в применении к матрице Вильсона

5	7	6	5
7	10	8	7
6	8	10	9
5	7	9	10

и дала результаты

67.9982	—40.9991	—16.9995	9.9997
—40.9991	24.9995	9.9997	—5.9998
—16.9995	9.9997	4.9998	—2.9999
9.9997	—5.9998	—2.9999	1.9999

(симметричная матрица дополнена программой вывода).

\* Указывается одна поправка к алгоритму 66, учтенная в алгоритме 66а. В данном и следующем переводах «Подтверждений» используются обозначения, принятые в алгоритме 66а. (Прим. ред.)

Данная процедура в действительности обращает не только положительно определенные симметричные матрицы (как это сказано у Дж. Каффри в описании алгоритма 66. — *Прим. ред.*). По-видимому, требуется только ограничение, чтобы ведущие миноры матрицы были не нулевыми. Переменная  $a[1,1]$  последовательно принимает значения отношения  $(r+1)$ -го к  $r$ -му ведущему минору матрицы, и если оно становится равным нулю, то переменная  $p$  не может быть вычислена.

Следующая матрица, в которой последовательными значениями  $a[1,1]$  были +2, —2, —1, —0.6, +5, дала результаты, верные с точностью до единицы пятой значащей цифры:

2	—3	1	—1	4
—3	2	—4	3	—2
1	—4	—3	2	4
—1	3	2	—2	—3
4	—2	4	—3	2

### Подтверждение к алгоритму 66

Дж. Каффри (Caffrey J. «CACM», 1962, № 6)

Эта процедура была транслирована с помощью системы BALCOM в Стенфордском университете при использовании 8-разрядной арифметики с плавающей запятой. Опечатка, указанная Ранделом и Бройденом («CACM», 1962, № 1), была исправлена, и в качестве теста использовался тот же пример (матрица Вильсона  $4 \times 4$ ). Результат обращения

68.0000	—41.0000	—17.0000	10.0000
	25.0000	10.0000	—6.0000
		5.0000	—3.0000
			2.0000

Полезно также отметить, что определитель матрицы может быть получен как результат последовательного перемножения значений главных элементов. Таким образом, если  $t_i$  (равный  $a[1,1]$ ) является главным элементом матрицы порядка  $n$ , то

$$\text{определитель} = \prod_{i=1}^n t_i.$$

Для вышеприведенного примера определитель равен единице.

Замечание Рандела и Бройдена относительно кажущегося ограничения этой процедуры случаями положительной определенности является правильным. Иначе говоря, любая невырожденная вещественная симметричная матрица (положительная, отрицательная или неопределенного знака) может быть обращена с помощью этого алгоритма. Следовательно, первоначальный вариант процедуры должен быть дополнен оператором

if  $p=0$  then go to signal.



Второй пример Рандела и Бройдена (матрица пятого порядка) был также использован в качестве теста со следующими результатами:

—0.0000	0.9999	0.0000	0.0000	0.9999
	1.5333	—0.7333	—0.1333	0.7999
		—0.8666	—1.0666	—0.5999
			—1.4666	—0.1999
				0.2000

Определитель равен —14.999999.

При попытке обратить обращенный сегмент  $4 \times 4$  матрицы Гильберта, как предложено Ранделом («CASM», 1962, № 1, с. 50), получены следующие результаты:

0.9999	0.4999	0.3333	0.2499
	0.3333	0.2499	0.1999
		0.1999	0.1666
			0.1428

Определитель равен 6048020.6.

## АЛГОРИТМ 676

### Умножение уплотненной симметричной матрицы на прямоугольную [F1]

Процедура *cram* [*cram* — уплотнение, втискивание (в память)] умножает симметричную матрицу  $e[1:n, 1:n]$  на прямоугольную  $b[1:n, 1:r]$  в предположении, что во входном одномерном массиве  $a[1:n \times (n+1)/2]$  построчно записана верхняя треугольная часть (включая главную диагональ) матрицы  $e$ . Результат помещается на место матрицы  $b$ .

В процессе выполнения процедуры *cram* значения элементов  $e[i, j]$  присваиваются элементам  $a[m_i + j]$ , где  $m_i = (2n - i) \cdot (i - 1) / 2$ . Из последнего соотношения можно найти, что  $m_{i+1} = m_i + n - i$ . Это позволяет вычислять значения  $m$  рекурсивно, имея  $m_1 = 0$ . Такое уплотнение симметричной матрицы может быть полезно, например, тогда, когда она не помещается целиком в свободную часть памяти машины.

```

procedure cram(a,n,r) dataresult: (b);
  value n,r; integer n,r; array a,b;
begin real s; integer i,j,k,m; array v[1:n];
  for j:=1 step 1 until r do
    begin
      for i:=1 step 1 until n do
        begin s:=0; m:=i—n;
          for k:=1 step 1 until n do
            begin m:=m+(if k≤i then n+1—k else 1);
              s:=s+a[m]×b[k,j]
            end k;
          v[i]:=s
        end i;
    end j;

```

```

      for i:=1 step 1 until n do b[i,j]:=v[i]
    end j
end scam;

```

### Свидетельство к алгоритму 67б

Алгоритм 67б получен в результате модификации алгоритма 67а, имевшей целью его упрощение и состоявшей в следующем.

1. Из заголовка процедуры был исключен параметр  $p$ . В соответствии с этим был вычеркнут оператор  $p:=n+1$ , а 1-я строка на 41 с. [6] была заменена на

```
begin m:=m+(if k≤i then n+1-k else 1);
```

2. Был вычеркнут второй оператор тела процедуры, обеспечивавший ввод массива  $a$ , поскольку такой ввод только сужал область применимости процедуры. Вследствие такого исключения стал ненужным и первый оператор тела процедуры  $k:=n \times (n+1)/2$ ;

После этих модификаций алгоритм был транслирован в системе ТА-1М для М-220 с примером, приведенным в «Свидетельстве к алгоритму 67а» и дал правильный результат. Массив  $a$  при этом задавался в виде  $a=(0, 4, 10, 8, 18, 40)$ .

### Свидетельство к алгоритму 67а

Алгоритм 67а получен в результате исправления, сокращения, модификации и ординарной переработки алгоритма 67 (Caffrey J. «CACM», 1961, № 7).

Модификация состояла в исключении массива  $c$  (для экономии памяти) и замене процедуры ввода *READ* процедурой *inreal*, рекомендованной журналом «CACM» в разделе «Алгоритмы» в мае 1964 г. [2, Приложение 1].

Алгоритм проверен с исходными данными

$$e = \begin{vmatrix} 0 & 4 & 10 \\ 4 & 8 & 18 \\ 10 & 18 & 40 \end{vmatrix}, \quad b = \begin{vmatrix} 5 & 3 \\ 2 & 6 \\ 1 & 8 \end{vmatrix}$$

и получен правильный результат

$$e \times b = \begin{vmatrix} 18 & 104 \\ 54 & 204 \\ 126 & 458 \end{vmatrix}.$$

### Подтверждение к алгоритму 67

А. П. Релф (Relf A. P. «CACM», 1962, № 6)

Процедура *scam* была транслирована при помощи DEUCE-ALGOL транслятора со следующими исправлениями...\*

### Свидетельство к алгоритму 68б [A1]

Алгоритм 68б не публикуется здесь, потому что соответствующий алгоритм 68 «Пополнение одного числа другим» (Rice H. G. «CACM», 1961, № 8) является примитивным, назначение его неясно и полезность его представляется сомнительной.

---

\* Указываются две опечатки в алгоритме 67 и предлагается изменить оператор ввода. (Прим. ред.)

## Свидетельство к алгоритму 696 [Н]

Алгоритм 696 не публикуется здесь, потому что соответствующий алгоритм 69 «Прослеживание цепочки» (Mausch H. «САСМ», 1961, № 9) не был подтвержден расчетами ни в журнале «САСМ», ни авторами данного выпуска.

## АЛГОРИТМ 706

### Интерполяция по Эйткуэну [Е1]

Если даны  $(n+1)$  абсцисс  $x_0, x_1, \dots, x_n$  и  $(n+1)$  значений функции  $f(x_0), f(x_1), \dots, f(x_n)$ , то процедура *aitken* строит полином Лагранжа  $ff(x)$  степени  $n$  такой, что  $ff(x_i) = f(x_i)$ . Следовательно, для любого данного значения  $xx$  получается значение  $ff(xx)$ . Для построения полинома  $ff(xx)$  применяется итеративная схема Эйткуэна. Поскольку массив  $f$  используется для хранения промежуточных значений, то его начальное значение затирается. Процедура *aitken* пригодна как для равных, так и для неравных интервалов  $[x_i, x_{i+1}]$ .

```
procedure aitken(x,f,n,xx) result: (ff);  
  value xx,n; real xx,ff; integer n; array x,f;  
begin integer i,j;  
  for j:=0 step 1 until n-1 do  
    for i:=j+1 step 1 until n do  
      f[i]:= ( (xx-x[j]) × f[i] - (xx-x[i]) × f[j]) / (x[i]-x[j]);  
    ff:=f[n]  
  end aitken;
```

## Свидетельство к алгоритму 706

Алгоритм 706 является стереотипным переизданием алгоритма 70а, который успешно используется сейчас на многих машинах, и в частности на БЭСМ-6.

При пользовании этим алгоритмом не следует задавать больших значений  $n$ , это может привести к переполнению или потере точности. Обычно достаточны значения  $n=2$  или  $n=3$ . Поэтому обычно приходится дополнять обращения к такой процедуре операторами, обеспечивающими выбор из данной таблицы трех или четырех точек таких, чтобы точка  $xx$  оказалась в интервале  $[x_0, x_n]$ .

## Свидетельство к алгоритму 70а

Алгоритм 70а получен в результате исправления, сокращения и ординарной переработки алгоритма 70 (Mifsud C. J. «САСМ», 1961, № 11).

Алгоритм 70а проверен с помощью транслятора ТА-1 для функции  $e^x$ . При  $xx=0.58$  для равных интервалов  $x$  получено значение  $ff=1.78605373$  (по справочнику [1]  $ff=1.7860$ ). При  $xx=0.86$  для неравных интервалов  $x$  получено  $ff=2.36311017$  (по справочнику [1]  $ff=2.3632$ ).

## Подтверждение к алгоритму 70

А. Релф (Relph A. P. «CASM», 1962, № 7)

Алгоритм 70 был транслирован с использованием транслятора DEUCE ALGOL и дал удовлетворительные результаты после добавления...\*

## Свидетельство к алгоритму 71б [G6]

Алгоритм 71б не публикуется здесь, потому что соответствующий алгоритм 71 («CASM», 1961, № 11) содержал ошибки и был в дальнейшем заменен в журнале «CASM» алгоритмами 86, 87, 102, 130, 202, дающими перестановки при более экономном расходовании машинной памяти и машинного времени.

## АЛГОРИТМ 72б

### Генератор композиций [A1]

Процедура *comp1* (сокращение от *composition* — композиция) преобразует данную композицию *c* из *k* частей целого положительного числа *n* в следующую по порядку композицию, если таковая существует. Если же процедура *comp1* выполняется с каждой новой композицией после ее получения, то образуются все композиции при условии, что в качестве первоначальной была взята композиция 1, 1, ..., 1,  $n-k+1$ . Если *c* имеет форму  $n-k+1, 1, 1, \dots, 1$ , то следующей композиции не существует и *c* заменяется вектором из *k* нулей [3, с 129].

```
procedure comp1(c,k);
  value k; integer k; integer array c;
begin integer j;
  j:=k;
test: if j>1 then
  begin
    if c[j]=1 then
      begin j:=j-1; go to test end;
    c[j-1]:=c[j-1]+1;
    c[k]:=c[j]-1;
    if j≠k then c[j]:=1
  end else
    for j:=1 step 1 until k do c[j]:=0
end comp1;
```

Процедура *comp2* образует и выдает на печать по каналу *k* всевозможные композиции из *p* частей целого положительного числа *n*.

```
procedure comp2(n,p,k);
  value n,p,k; integer n,p,k;
begin integer j; integer array c[1:p];
  for j:=1 step 1 until p-1 do c[j]:=1;
  c[p]:=n-p+1;
start: outarray(k,c);
  j:=p;
```

---

\* Указываются две поправки и возможность сокращения алгоритма 70, учтенные в алгоритме 70а. (Прим. ред.)

```

test: if j>1 then
  begin
    if c[j]=1 then
      begin j:=j-1; go to test end;
    c[j-1]:=c[j-1]+1;
    c[p]:=c[j]-1;
    if j≠p then c[j]:=1;
    go to start
  end
end comp2;

```

### Свидетельство к алгоритму 72б

Алгоритм 72б отличается от алгоритма 72а только тем, что в процедуре *comp2* оператор с меткой «start:» был заменен на

```
start: outarray(k,c);
```

Алгоритм 72б был транслирован в системе ТА—1М для М-220 с примером, приведенным в «Свидетельстве к алгоритму 72а», и дал такие же результаты.

### Свидетельство к алгоритму 72а

Алгоритм 72а написан в двух вариантах заново в более краткой, экономичной и наглядной форме, чем алгоритм 72 (Hellerman L., Ogden S. «CACM», 1961, № 11). Перевод «Подтверждения к алгоритму 72» (Collison D. M. «CACM», 1962, № 8) здесь не приводится, поскольку оно потеряло свое значение после написания алгоритма 72а.

Алгоритм 72а проверен для  $n=5$  и  $k=3$ . Получено шесть следующих трехкомпонентных композиций числа 5:

```

c [1]  1  1  1  2  2  3
c [2]  1  2  3  1  2  1
c [3]  3  2  1  2  1  1

```

## АЛГОРИТМ 73б

### Неполные эллиптические интегралы [S21]

Процедура *ellint* вычисляет значения неполных эллиптических интегралов первого и второго рода  $F(\phi, k)$  и  $E(\phi, k)$ , где  $\phi$  берется в радианах. Если  $|k| > 1$  или  $|\phi| > \pi/2$ , то  $E$  и  $F$  приравниваются  $10^8$ , иначе они будут содержать вычисленные интегралы. Алгоритм использует константы  $\pi/2 = 1.57 \dots$  и  $tg(1) = 1.557393 \dots$

Более полную информацию об этом алгоритме можно получить в работе Дидonato и Хершея [21].

```

procedure ellint(k,phi) result: (ee,ff);
  value k,phi; real k,phi,ee,ff;
begin real kp,sn,cs,h1,h2,a1,a2,s1,s2,s3,s4,p1,p2,m1,m2,n1,n2,
  t1,t2,d1,d2,d3,d4; integer n;
  s1:=s2:=s3:=s4:=0; n:=0;
  h1:=1; sn:=sin(phi); cs:=cos(phi);
  if abs(k×sn) ≤ 1.557393 then go to small;

```

```

    if (abs(k) ≤ 1) ∧ (abs(phi) ≤ 1.57079633) then go to large;
    ee:=ff:=108;
    go to fin;
small: al:=phi;
step1: n:=n+2; ee:=(n-1)/n; t1:=k↑2×h1;
    h2:=ee×t1;
    a2:=ee×a1—sn↑(n-1)×cs/n;
    d1:=h2×a2; d2:=-t1×a2/n;
    s1:=s1+d1; s2:=s2+d2;
    h1:=h2; a1:=a2;
    if abs((s1+d1)—s1) > 0 ∧ abs(phi×sn↑n) ≥ abs(a2) then
        go to step1;
    ff:=phi+s1; ee:=phi+s2;
    go to fin;
large: kp:=1—k↑2; al:=1;
    p1:=m1:=n1:=0; t1:=abs(k)×cs; t2:=1—(k×sn)↑2;
step2: n:=n+2; ee:=(n-1)/n;
    ff:=t1×t2↑((n-1)/2)/n;
    h2:=ee×h1;
    a2:=ee↑2×kp×a1;
    p2:=p1+1/((n-1)×(n/2));
    m2:=(m1—ff×h2)×((n+1)/(n+2))↑2×kp;
    n2:=(n1—ff×h1)×ee×(n+1)×kp/(n+2);
    d1:=m2—a2×p2;
    d2:=n2+kp×(a1/n↑2—ee×a1×p2);
    d3:=a2; d4:=(n+1)×a2/(n+2);
    s1:=s1+d1; s2:=s2+d2;
    s3:=s3+d3; s4:=s4+d4;
    h1:=h2; a1:=a2;
    p1:=p2; m1:=m2;
    n1:=n2;
    if abs((s1+d1)—s1) > 0 then go to step2;
    n1:=sqrt(t2); n2:=abs(k×sn);
    t2:=t1/n1; t1:=ln(4/(n1+t1));
    ff:=t1×(1+s3)+t2×ln(0.5×(1+n2))+s1;
    ee:=(0.5+s4)×kp×t1+1—t2×(1—n2)+s2;
    if phi ≥ 0 then go to fin;
    ee:=-ee; ff:=-ff;
fin: end ellint;

```

### Свидетельство к алгоритму 736

Алгоритм 736 получен в результате дальнейшей оптимизации алгоритма 73а путем замены некоторых неоднократно повторяющихся выражений простыми переменными. Например, в строке, начинающейся с метки «*step1*:», добавлен оператор  $t1:=k\uparrow 2 \times n1$ ; что дает возможность заменить в последующих строках выражение  $k\uparrow 2 \times n1$  на простую переменную  $t1$ . Аналогично простыми переменными были заменены выражения  $abs(k) \times cs$  и  $1 - (k \times sn)\uparrow 2$  (начиная со второй строки после метки «*large*:») и выражения  $\sqrt{1 - (k \times sn)\uparrow 2}$  и  $abs(k \times sn)$  (начиная со строки, следующей за оператором **if**  $abs((s1+d1)—s1) > 0$  **then go to step2**;). Кроме того, после метки «*large*:» оператор

$kp := \text{sqrt}(1 - k \uparrow 2)$  был заменен оператором  $kp := 1 - k \uparrow 2$ , что дало возможность всюду в процедуре заменить выражение  $kp \uparrow 2$  на  $kp$ .

Алгоритм 73б был транслирован на машине М-220 в системе ТА-1М, и были вычислены значения  $F(\varphi, \sin \alpha)$ ,  $E(\varphi, \sin \alpha)$  для  $\varphi = 0^\circ, 1^\circ, 10^\circ, 20^\circ, \dots, 90^\circ$  и  $\alpha = 0^\circ, 1^\circ, 10^\circ, 20^\circ, \dots, 80^\circ, 89^\circ, 90^\circ$ . Результаты трансляции совпали с контрольными значениями взятыми из семизначных таблиц В. П. Ветчинкина [39], за исключением  $F(60^\circ, \sin 60^\circ) = 1.212597$  вместо 1.212537 у Ветчинкина, что объясняется, по-видимому, опечаткой. Некоторые из полученных значений приведены в табл. 8. Следует заметить, что при  $\varphi = 1^\circ$  и  $\alpha = 89^\circ$  не наблюдалось никакого заикливания, о котором говорится в нижеследующем «Подтверждении к алгоритму 73» Р. П. Ван де Рита, и были получены правильные результаты.

Таблица 8

$\varphi^\circ$	$\alpha^\circ$	$E(\varphi, \sin \alpha)$	$F(\varphi, \sin \alpha)$	$\varphi^\circ$	$\alpha^\circ$	$E(\varphi, \sin \alpha)$	$F(\varphi, \sin \alpha)$
30	0	0.52359877	0.52359877	1	89	0.017452406	0.017454178
30	1	0.52359187	0.52360567	1	30	0.017453071	0.017453514
30	10	0.52291511	0.52428401	10	30	0.17431249	0.17475385
30	40	0.51408861	0.53342745	40	30	0.68506023	0.71164727
30	70	0.50286804	0.54593191	70	30	1.1631768	1.2853005
30	90	0.50000000	0.54930614	90	30	1.4674622	1.6857503

### Свидетельство к алгоритму 73а

Алгоритм 73а получен в результате исправления, оптимизации и ординарной переработки алгоритма 73 (Jefferson D. K. «САСМ», 1962, № 12). Перевод замечания Д. Джефферсона («САСМ», 1962, № 10) не приводится, поскольку оно не добавляет новой информации к остальным материалам по алгоритму 73.

Кроме ошибок, указанных в приводимых ниже «Подтверждениях» Н. А. Мейера и Р. П. Ван де Рита, в алгоритме 73 исправлено следующее.

1. Выражение  $\pi/2$  заменено его значением 1.57079633, а  $\tanh(1)$  заменен на  $tg(1) = 1.557393$  согласно справочнику [19].

2. Переменная  $n$  отнесена к типу **integer**, иначе для  $\text{phi} < 0$  выражения, подобные  $sn \uparrow (n-1)$ , были бы не определены.

Переменные алгоритма 73а соответствуют переменным алгоритма 73 согласно следующему:

$n1 - N [1]$	$d1 - del [1]$	$sn - \sin \text{phi}$	$s1 - \text{sigma} [1]$
$n2 - N [2]$	$d2 - del [2]$	$cs - \cos \text{phi}$	$s2 - \text{sigma} [2]$
$t1 - T [1]$	$d3 - del [3]$	$h1 - H [1]$	$s3 - \text{sigma} [3]$
$t2 - T [2]$	$d4 - del [4]$	$h2 - H [2]$	$s4 - \text{sigma} [4]$
$a1 - A [1]$	$p1 - L [1]$	$m1 - M [1]$	$ee - E$
$a2 - A [2]$	$p2 - L [2]$	$m2 - M [2]$	$ff - F$

### Подтверждение к алгоритму 73

Д. Крайбл (Kriebel D. C. «САСМ», 1961, № 12)

Этот алгоритм был первоначально запрограммирован на языке машины NORC, и были получены таблицы неполного эллиптического интеграла первого и второго рода [2i].

Алгоритм был закодирован для транслятора MAD точно так, как он написан на языке АЛГОЛ и проверен на машине IBM 7090. Было сосчитано 40 случаев для  $k$  и  $\phi$ , меняющихся от  $0^\circ$  до  $90^\circ$ . Результаты выдавались с восемью значащими цифрами и совпали с таблицами Дидonato и Хершея с точностью от нуля до двух единиц восьмой цифры. (Это могло произойти из-за переводов из десятичной в двоичную и из двоичной в десятичную систему счисления при вводе и выводе, используемых в двоичной вычислительной машине. Результаты сравнивались с результатами прямого десятичного счета на машине NORC.)

### Подтверждение к алгоритму 73

Н. А. Мейер (Me yer N. A. «САСМ», 1963, № 2)

Процедура *ellint* была вручную закодирована на языке ФОРТРАН для машины IBM 7070. Были сделаны следующие исправления... \*

Исправленный алгоритм дал удовлетворительные результаты при сравнении с таблицами Дидonato и Хершея. Отличия встречались в восьмой значащей цифре, как показано в табл. 9 (интеграл  $F$ ) и 10 (интеграл  $E$ ).

Таблица 9

$\Phi$ , град	$\theta$ , град			
	0	30	60	90
0	0	0	0	0
30	$-1 \times 10^{-8}$	$-1 \times 10^{-8}$	$-1 \times 10^{-8}$	$-3 \times 10^{-8}$
60	$1 \times 10^{-8}$	$1 \times 10^{-8}$	$2 \times 10^{-8}$	$-3 \times 10^{-8}$
90	0	$2 \times 10^{-8}$	$6 \times 10^{-8}$	0

Таблица 10

$\Phi$ , град	$\theta$ , град			
	0	30	60	90
0	0	0	0	0
30	$-1 \times 10^{-8}$	$-1 \times 10^{-8}$	$-1 \times 10^{-8}$	$-1 \times 10^{-8}$
60	$1 \times 10^{-8}$	$1 \times 10^{-8}$	$-7 \times 10^{-8}$	$3 \times 10^{-8}$
90	0	0	$1 \times 10^{-8}$	0

### Подтверждение к алгоритму 73

Р. П. Ван де Рит (Van de Riet R. P. «САСМ», 1963, № 4)

Алгоритм содержал три опечатки... \*\*

\* Далее указываются три ошибки и возможность усовершенствования алгоритма 73, учтенные в алгоритме 73а. (Прим. ред.)

\*\* Далее указываются три опечатки в алгоритме 73, учтенные в алгоритме 73а. (Прим. ред.)



Программа была проверена на вычислительной машине X1 математического центра. Для  $\phi=45^\circ$ ,  $k=\sin 10^\circ, \sin 20^\circ, \dots, \sin 180^\circ$  были вычислены  $E$  и  $F$ . Результаты содержали 12 значащих цифр.

Сравнение с 12-разрядными десятичными таблицами Лежандра — Эмде (1931 г.) показало, что 12-я цифра была получена с ошибкой самое большое в четыре единицы. После 10 мин счета (т. е. после более чем 100 циклов) не было получено никаких результатов для  $k=\sin 89^\circ$ ,  $\phi=1^\circ$ , так как произошло заикливание.

Замечание. Поскольку  $\phi$  не меняется во время вычисления, то мы поставили оператор  $\cos \phi := \cos(\phi)$  в начало программы, чтобы предотвратить вычисление косинуса 30 или более раз. Кроме того, в выражении для  $T[1]$  и  $T[2]$  значение  $\sqrt{1-\sin^2 \phi}$  было заменено на  $\cos \phi$ , так как потери значащих цифр не происходит.

Выражение  $2 \times n$  было заменено новой переменной для ускорения работы программы.

## АЛГОРИТМ 746

**Аппроксимация с помощью полиномиальной кривой данной степени, проходящей через данные точки (метод наименьших квадратов) [E2]**

Процедура *curfit* (*curve* — кривая, *fitting* — вычерчивание по точкам) находит методом наименьших квадратов полином степени  $n$  ( $k \leq n \leq k+m$ ), график которого проходит через точки  $(a_1, b_1), \dots, (a_k, b_k)$  и аппроксимирует точки  $(x_1, y_1), \dots, (x_m, y_m)$ , причем  $w_i$  — это вес, приписываемый точке  $(x_i, y_i)$ . Процедура *curfit* использует нелокализованные метки *signal74* и *signal74g* и стандартную процедуру *outreal*(1, <A>), выводящую из машины по каналу 1 значение арифметического выражения <A>.

Детальное описание алгоритма приводится в работе Пека [10i], где используются такие же обозначения, что и в данной процедуре\*.

```

procedure curfit(k,a,b,m,x,y,w,n,gamma,x0,z,r)
  result:(alpha,beta,s,sgmsq,c);
  value k,m,n,r,x0,gamma; real x0,gamma; integer k,m,n,r;
  array a,b,x,y,w,alpha,beta,s,sgmsq,c,z;
begin real p,f,lamdba; integer i,j; array w1[1:k];
  comment Сначала определяются несколько процедур, которые
  используются в основной программе, начинающейся с метки
  start;
  procedure evaluate(x,nu);
    value x,nu; real x; integer nu;
    comment Процедура evaluate вычисляет значение  $f = s_0 p_0 +$ 
     $s_1 p_1 + \dots + s_v p_v$ , где  $p_{i+1}(x) = (x-a_i)p_i(x) - \beta_i p_{i-1}(x)$  для
     $i=0, 1, \dots, v-1$ ,  $p_{-1}(x)=0$ ,  $p_0(x)=1$ ,  $\beta_0=0$ . Значение  $p_v(x)$ 
    остается в p;
    begin real p0,t; integer i;
    p0:=0; p:=1; f:=s[0];

```

\* Назначение основных параметров процедуры *curfit* раскрывается также и в нижеследующем «Свидетельстве к алгоритму 746». (Прим. ред.)

```

for i:=0 step 1 until nu-1 do
  begin t:=p;
    p:=(x-alpha[i])×p-beta[i]×p0;
    p0:=t; f:=f+p×s[i+1]
  end i
end evaluate;
procedure coda(n,c);
  value n; integer n; array c;
  comment Эта процедура находит коэффициенты c такие, что
   $c_0 + c_1x + \dots + c_nx^n = s_0p_0(x) + \dots + s_np_n(x)$ ;
  begin real t1,t2; integer i,r; array pm,p[0:n];
    for r:=1 step 1 until n do
      c[r]:=pm[r]:=p[r]:=0;
      c[0]:=s[0]; pm[0]:=0; p[0]:=1;
      for i:=0 step 1 until n-1 do
        begin t2:=0;
          for r:=0 step 1 until i+1 do
            begin
              t1:=(t2-alpha[i]×p[r]-beta[i]×pm[r])/lambda;
              t2:=pm[r]:=p[r]; p[r]:=t1;
              c[r]:=c[r]+t1×s[i+1]
            end r
          end i
        end coda;
  procedure gefyt(n,n0,x,y,w,m);
    value n, n0,m; integer n,n0,m; array x,y,w;
    comment Это сердце основной процедуры. Процедура gefyt
    вычисляет  $\alpha_i, \beta_i, s_i, \sigma_i^2$ , используя метод ортогональных полино-
    мов, описанный в вышеуказанной работе Пека;
    begin real dsq,wpp,wpp0,wxpp,wyp,t;
      integer i,j,freedom; Boolean exact; array p,p0[1:m];
      if n-n0>m ∨ n<n0 then go to signal74g;
      beta[n0]:=dsq:=wpp:=0;
      exact:=n-n0≥m-1;
      for j:=1 step 1 until m do
        begin p[j]:=1; p0[j]:=0;
          wpp:=wpp+w[j];
          if ¬exact then dsq:=dsq+w[j]×y[j]×y[j]
        end расчета начальных условий;
      for i:=n0 step 1 until n do
        begin freedom:=m-1-(i-n0);
          wyp:=wxpp:=0;
          for j:=1 step 1 until m do
            begin t:=w[j]×p[j];
              if i<n then wxpp:=wxpp+t×x[j]×p[j];
              if freedom≥0 then wyp:=wyp+t×y[j]
            end j;
          if freedom≥0 then s[i]:=wyp/wpp;
          if ¬exact then
            begin dsq:=dsq-s[i]×s[i]×wpp;
              sgmsq[i]:=dsq/freedom
            end if;
          if i<n then

```

```

begin alpha[i]:=wxpp/wpp;
wpp0:=wpp; wpp:=0;
for j:=1 step 1 until m do
begin
t:=(x[j]-alpha[i])×p[j]-beta[i]×p0[j];
wpp:=wpp+w[j]×t×t;
p0[j]:=p[j]; p[j]:=t
end j;
beta[i+1]:=wpp/wpp0
end if
end i
end gefyt;
comment Здесь начинается основная программа;
start: for j:=1 step 1 until k do
begin w1[j]:=1; a[j]:=(a[j]-x0)/gamma end j;
gefyт(k,0,a,b,w1,k);
comment Здесь вычисляется полином степени k-1, график ко-
торого проходит через точки (a1,b1),..., (ak,bk), и αi, βi, si, где
0≤i≤k;
begin real rho;
rho:=0;
for j:=1 step 1 until m do
begin rho:=rho+w[j]; x[j]:=(x[j]-x0)/gamma end j;
rho:=m/rho;
comment Множитель ρ используется для нормализации весов.
Теперь для вычисления значения rk(x) и одновременно поли-
нома степени k-1 положим sk=0;
s[k]:=0;
for j:=1 step 1 until m do
begin evaluate(x[j],k);
if p=0 then go to signal74;
y[j]:=(y[j]-f)/p; w[j]:=w[j]×p×p×rho
end j
end rho;
comment Теперь веса нормализованы, и отрегулированные веса
и ординаты готовы для аппроксимации методом наименьших
квадратов;
gefyт(n,k,x,y,w,m);
comment Коэффициенты αi, βi (0≤i<n) и si(0≤i≤n) теперь
готовы. Полином может быть вычислен для x=z1, z2, ..., zг, но
переменная должна быть сначала отрегулирована. Заметим,
что мы можем, уменьшая n, вычислить наилучший полином
меньшей степени;
begin real x1;
for j:=1 step 1 until г do
begin x1:=(z[j]-x0)/gamma;
evaluate(x1,n);
outreal(1,z[j]);
outreal(1,f)
end j;
comment Теперь мы можем отрегулировать весовые коэф-
фициенты и затем найти коэффициенты для степенного ряда
c0+c1x+...+cnxn=s0p0(x)+...+snpn(x);

```

```

for i:=0 step 1 until n-1 do
  begin alpha[i]:=alpha[i]×gamma+x0;
    beta[i]:=beta[i]×gamma
  end i;
lambda:=gamma;
coda(n,c);
comment Мы можем теперь заново вычислить полином из
степенного ряда;
for j:=1 step 1 until r do
  begin x1:=z[j]; f:=c[n];
    for i:=n-1 step -1 until 0 do f:=f×x1+c[i];
      outreal(1,x1);
      outreal(1,f)
    end j
  end x1
end curfit;

```

#### Свидетельство к алгоритму 74б

Процедура *curfit* алгоритма 74б не отличается от процедуры *curfit* алгоритма 74а, за исключением того, что был изменен порядок расположения параметров в заголовке процедуры, а именно выходные параметры помещены в конце списка формальных параметров и отделены от остальных параметров разделителем параметров

)result:(

В комментариях к процедуре *curfit* были сделаны некоторые уточнения. Название алгоритма заменено более полным.

Алгоритм был транслирован в системе ТА-1М на машине М-220 с предварительной заменой в процедуре *curfit* двух операторов

outreal(1,z[j]); outreal(1,f)

на оператор

p1041(z[j],f)

и операторов

outreal(1,x1); outreal(1,f)

на оператор

p1041(x1,f)

в соответствии с системой ввода—вывода транслятора ТА-1М. Входные массивы были описаны в начале ведущей программы следующим образом:

```

array a,b[1:k],x,y,w[1:m],alpha,beta[0:n-1],z[1:r],
s,c[0:n],sgmsq[k:n];

```

Были проведены следующие варианты контрольных решений.

1. Аппроксимация точек, лежащих на одной прямой, полиномом первой степени.

Задавались  $n=k=1$ ;  $a[1:1]=2$ ;  $b[1:1]=3$ ;  $r=1$ ;  $z[1:1]=1$ ;  $x_0=0$ ;  $gamma=1$  при следующих вариантах значений остальных параметров.

1а. Аппроксимация одной точки, т. е.  $m=1$ ;  $x[1:1]=0$ ;  $y[1:1]=1$ ;  $w[1:1]=1$ .

Получено  $d[0:1]=(1,1)$ , т. е. точное уравнение  $y=x+1$  прямой, проходящей через две заданные точки  $(a_1,b_1)=(2,3)$  и  $(x_1,y_1)=(0,1)$ . В процессе своего выполнения процедура дважды напечатала пару чисел (1,2), соответствующую  $z[1]=1$  и  $f=2$ .

1б. Аппроксимация четырех точек, т. е.  $m=4$ ;  $x[1:4]=(0,1,3,5)$ ;  $y[1:4]=(1,2,4,6)$ ;  $w[1:4]=(1,1,1,1)$ .

Получено  $c[0:1]=(1,1)$ , т. е. то же, что и в предыдущем случае. О точности аппроксимации в этом случае можно было бы судить и по значению  $sgmsq=0$ .

1в. Вариация масштабных коэффициентов  $x_0$  и  $gamma$ .

Значения  $x_0$  и  $gamma$  на результатах аппроксимации не отражаются, и их можно задавать любыми (кроме  $gamma=0$ ). Так результаты трансляции для тех же параметров, что и в п. 1б, но при  $(x_0, gamma)=(-1,-1)$ ,  $(-1,2)$  и  $(1,1)$ , ничем не отличались от результатов п. 1б. Однако нужно иметь в виду, что при задании  $(x_0, gamma) \neq (0,1)$  выходные значения  $x$ ,  $y$  и  $w$  будут отличаться от входных. Масштабные коэффициенты  $(x_0, gamma) \neq (0,1)$  можно использовать, например, для ликвидации переполнения или потери точности, которые могут возникнуть в процессе выполнения процедуры.

2. Аппроксимация точек параболы  $y=x^2+1$  полиномом второй степени.

Задавались  $n=k=2$ ;  $a[1:2]=(0.5,4.5)$ ;  $b[1:2]=(1.25,21.25)$ ;  $r=1$ ;  $z[1:1]=1$ ;  $x_0=0$ ;  $gamma=1$  при следующих вариантах остальных параметров.

2а. Аппроксимация шести точек одной из ветвей параболы, т. е.  $m=6$ ;  $x[1:6]=(0,1,2,3,4,5)$ ;  $y[1:6]=(1,2,5,10,17,26)$ ;  $w[1:6]=(1,1,1,1,1,1)$ .

Получено  $c[0:2]=(0.96021126, 0.11619718, 0.92676056)$ , в то время как точные коэффициенты параболы должны быть  $(1,0,1)$ . Неудовлетворительность аппроксимации видна и по значению  $sgmsq=0.47114436$ . Улучшить аппроксимацию можно подбором весовых коэффициентов, как это видно из нижеследующего пункта.

2б. Подбор весовых коэффициентов.

После замены весов на  $w[1:6]=(0.5,1,1,1,1,0.5)$  при остальных параметрах тех же, что и в п. 2а, были получены точные коэффициенты полинома  $c[0:2]=(1,0,1)$  и соответственно  $sgmsq=0$ . Такие же результаты были и для значений  $w[1:6]=(0,0.5,1,1,0.5,0)$ ,  $(0.5,0.75,1,1,0.75,0.5)$   $(0.5,1,1,1,1,1)$ .

2в. Аппроксимация 11 точек обеих ветвей параболы, т. е.  $m=11$ ;  $x[1:11]=(-5,-4,-3,-2,-1,0,1,2,3,4,5)$ ;  $y[1:11]=(26,17,10,5,2,1,2,5,10,17,26)$ ;  $w[1:11]=(0.5,0.75,1,1,1,1,1,1,0.75,0.5)$ .

Получено точное решение  $c[0:2]=(1,0,1)$ ,  $sgmsq=0$ . Но при задании всех весов равными единице, было получено столь же неудовлетворительное решение, что и в п. 2а.

Таблица 11

$i$	$z[i]$	$f$ при $w=w1$	$f$ при $w=w2$	$exp(z[i])$
1	0.25	1.0873588	1.0893908	1.2840254
2	0.75	2.1870061	2.1860578	2.1170000
3	1.25	3.5216607	3.5209833	3.4903429
4	1.75	5.7459572	5.7466345	5.7546026
5	2.25	9.5145303	9.5154786	9.4877358
6	2.75	15.482014	15.479932	15.642631

3. Аппроксимация точек экспоненты  $y=e^x$ , заданной четырьмя точками, т. е.  $m=4$ ;  $x[1:4]=(0,1,2,3)$ ;  $y[i]=exp(x[i])$  для  $i=1,2,3,4$ ;  $x_0=0$ ;  $y=1$ ;  $r=6$ ;  $z[1:6]=(0.25,0.75,1.25,1.75,2.25,2.75)$  параболой  $n=3$ , прохо-

дящей через три точки  $a[1:3] = (0.5, 1.5, 2.5)$  и  $b[i] = \exp(a[i])$  для  $i = 1, 2, 3$  при  $\omega = \omega 1 = (1, 1, 1)$  и  $\omega = \omega 2 = (0.5, 1, 0.5)$ .

Полученные в результате трансляции ординаты  $f$  аппроксимирующей кривой в заданных точках  $z[i]$  приведены в табл. 11.

Значения  $sgmsq$  были 0.22944672 при  $\omega = \omega 2$  и 0.15392303 при  $\omega = \omega 1$ .

### Свидетельство к алгоритму 74а

Алгоритм 74а получен в результате ординарной переработки алгоритма 74 (Р е с к J. E. L. «CASM», 1962, № 1).

В алгоритме 74 была обнаружена опечатка во второй строке после метки *start*, а именно, символ «=» должен быть заменен символом «:=». Кроме того, проведена модификация алгоритма 74, заключающаяся в замене всех указаний о выдаче параметров на обращения к процедуре *outreal*, рекомендованной международным АЛГОЛ-комитетом в 1964 г. [21].

### Подтверждение к алгоритму 74

К. Исода (I s o d a К. «CASM», 1963, № 6)

Алгоритм 74 был вручную переведен на язык SOAP IIa для машины IBM 650 и успешно прошел без всяких исправлений, кроме случая, когда начало (0,0) дается как условие или как проба.

## АЛГОРИТМ 756

### Разложение многочлена на множители [C2]

Процедура *factors* (*factors* — множители) находит все рациональные линейные множители  $(u_i x + v_i)$ , где  $1 \leq i \leq r$ , многочлена  $a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n$  с целыми коэффициентами. Кроме того, находится общий наибольший делитель  $c$  коэффициентов  $a_i$ .

Метод заключается в том, что находятся все делители  $p$  коэффициента  $a_0$  и все делители  $q$  коэффициента  $a_n$  (причем  $1 \leq p \leq |a_0|$  и  $1 \leq q \leq |a_n|$ ). Составляются всевозможные пары из найденных  $p$  и  $q$  и проверяется, не является ли двучлен  $(px - q)$  множителем многочлена.

```

procedure factors(n,a) result: (u,v,r,c);
  value n; integer n,r,c; integer array a,u,v;
begin integer i,f,g,p,q;
  r:=0; c:=1;
  comment Сначала исключаются множители вида  $(1 \times x - 0)$ ;
zero: if a[n]=0 then
    begin n:=n-1; r:=r+1;
      u[r]:=1; v[r]:=0;
    go to zero
  end zero;
  for p:=1 step 1 until abs(a[0]) do
    if (a[0]÷p)×p=a[0] then
      begin comment Найден p, являющийся делителем  $a_0$ ;
        for q:=1 step 1 until abs(a[n]) do

```

```

begin comment Отыскивается  $q \neq 1$ , являющийся
множителем  $a_n$ ;
if  $q = 1$  then go to varia;
const: if  $(a[n] \div q) \times q = a[n]$  then
begin comment Далее проверка: не является
ли  $q$  общим множителем для всех  $a_i$ ;
for  $i := 0$  step 1 until  $n - 1$  do
if  $(a[i] \div q) \times q \neq a[i]$  then go to varia;
for  $i := 0$  step 1 until  $n$  do  $a[i] := a[i]/q$ ;
 $c := c \times q$ ;
go to const
end const. Затем проверка: является ли  $(px - q)$ 
множителем многочлена;
varia:  $f := a[0]$ ;  $g := 1$ ;
for  $i := 1$  step 1 until  $n$  do
begin  $g := g \times p$ ;  $f := f \times q + g \times a[i]$  end i;
if  $f = 0$  then
begin comment Данный двучлен  $(px - q)$  яв-
ляется множителем. Далее делим на него
наш полином;
 $r := r + 1$ ;  $u[r] := p$ ;  $v[r] := q$ ;
for  $i := 0$  step 1 until  $n$  do
begin
 $a[i] := f := (a[i] + f)/p$ ;  $f := f \times q$ 
end i;
 $n := n - 1$ ;
go to if  $n = 0$  then fin else varia
end f;
 $q := -q$ ;
if  $q < 0$  then go to varia
end q
end p;
fin: if  $n = 0$  then  $c := c \times a[0]$ 
end factors;

```

### Свидетельство к алгоритму 75б

Алгоритм 75б является стереотипным переизданием алгоритма 75а.

### Свидетельство к алгоритму 75а

Алгоритм 75а получен в результате сокращения и ординарной переработки процедуры, приведенной в «Подтверждении к алгоритму 75» А. П. Релфа (Relph A. P. «САСМ», 1962, № 7). Перевод подтверждения Дж. Хилмора («САСМ», 1962, № 8) здесь не приводится как потерявший свое значение после публикации алгоритма 75а.

Алгоритм 75а проверен с помощью транслятора ТА-1 для многочленов  $3x^3 - 29x^2 + 78x - 40$  и  $x^3 - 6x^2 + 32$ . Получены правильные результаты:

для первого многочлена  $1 \times (x - 4) (x - 5) (3x - 2)$ ;

для второго многочлена  $1 \times (x + 2) (x - 4) (x - 4)$ .

## Подтверждение к алгоритму 75

А. П. Релф (Relf A. P. «CACM», 1962, № 7)

Алгоритм 75 был транслирован с использованием транслятора DEUCE ALGOL и дал удовлетворительные результаты после следующих исправлений...\*

## Свидетельство к алгоритму 76б [M1]

Алгоритм 76б «Процедуры сортировки» здесь не приводится, поскольку соответствующий алгоритм 76 («CACM», 1962, № 1) не пригоден ни к какой переработке по причинам, указанным в нижеследующем «Замечании» Б. Рэнделла.

## Замечание к алгоритму 76

Б. Рэнделл (Randell B. «CACM», 1962, № 6)

В процедурах сортировки были обнаружены следующие типы ошибок.

1. Описания процедур не начинаются с символа **procedure**.
  2. В спецификациях массивов даются границы индексов.
  3. Символ «=» используется вместо символа «:=» в операторах присваивания и в заголовках циклов.
  4. Большое число точек с запятой опущено (обычно после **end**).
  5. Выражения в списках граничных пар в описаниях массивов зависят от локализованных переменных.
  6. В некоторых операторах процедур отсутствуют правые скобки.
  7. Условные операторы следуют за **then**.
  8. Нет описаний для A или Z, что является, по-видимому, опечаткой.
  9. В некоторых процедурах делается попытка использовать некоторые идентификаторы для двух различных величин, а иногда попытка дважды описать один идентификатор в том или ином начале блока.
  10. В процедуре *Presort quadratic selection* массив, описанный как двумерный, используется как одномерный. д
  11. В одном месте переменная с индексами дается как фактический параметр, соответствующий формальному параметру, специфицированному как массив.
  12. В некоторых процедурах идентификаторы, используемые в качестве формальных параметров, вновь описываются и все же используются как параметры.
  13. В каждой процедуре в совокупности спецификаций дается *k*, не заданное в списке формальных параметров.
- Не было сделано никакой попытки транслировать или даже понять логику этих процедур. Думается, что такая чрезвычайно неаккуратная попытка использования языка АЛГОЛ никогда не должна появляться в качестве алгоритма в журнале «CACM».

---

\* Далее указываются две поправки к алгоритму 75 и приводится новый вариант процедуры *factors*, в результате переработки которого получен алгоритм 75а. (Прим. ред.)



## АЛГОРИТМ 776

### Интерполяция, дифференцирование и интегрирование функций [D1, D4, E1]

Процедура *difint* может в зависимости от способа обращения к ней выполнять интерполяцию, дифференцирование или интегрирование функций одной переменной, которые на всем или на интересующей нас части интервала адекватно описываются с помощью дупараболической кривой.

Процедура основана на использовании интерполяционной схемы Лагранжа, приспособленной для осредненных парабол второго порядка. По данному методу вычисляется производная от функции, численно определенной в точках 1, 2, 3 и 4, путем проведения одной параболы через точки 1, 2, 3 и второй параболы через точки 2, 3 и 4. Тогда производная в точке 2 равна среднему значению аналитических производных двух парабол, т. е. коэффициенты параболы  $(a_1x^2 + b_1x + c_1)$ , проходящей через точки 1, 2 и 3, и коэффициенты параболы  $(a_2x^2 + b_2x + c_2)$ , проходящей через точки 2, 3 и 4, определяются с помощью уравнений Лагранжа, как показано ниже. Арифметические средние значения этих коэффициентов

$$a = (a_1 + a_2)/2, \quad b = (b_1 + b_2)/2, \quad c = (c_1 + c_2)/2$$

используются для получения производной на интервале от точки 2 до точки 3 по формуле  $(2ax + b)$ .

Интерполяция производится аналогично, за исключением того, что конечной формулой является парабола  $(ax^2 + bx + c)$ .

Интегрирование производится подобно построению кривой по точкам. Например, интеграл между некоторыми двумя точками, скажем, 2 и 3, является осредненным интегралом двух парабол между границами независимых координат для точек 2 и 3. Осреднение производится для каждого интервала вдоль абсциссы, так как полученные результаты накапливаются для вычисления определенного интеграла.

С помощью уравнений Лагранжа находятся коэффициенты  $a_k$ ,  $b_k$  и  $c_k$

путем вычисления  $t_j = y_j \left| \prod_{i=1, i \neq j}^m (x_j - x_i) \right|$ , где  $y_j = f(x_j)$ ,  $j = 1, 2, \dots, m$ .

Тогда  $a_k = \sum_{i=1}^m t_i$ ;  $b_k = \sum_{i=1}^m t_i \times \sum_{j=1, j \neq i}^m x_j$ ;  $c_k = \sum_{i=1}^m t_i \times \prod_{j=1, j \neq i}^m x_j$ . В данной процедуре эти формулы применяются для  $m=3$ ,  $k=1$  и 2.

Значения формальных параметров поясняются ниже:

$x$  — массив табличных значений аргумента, имеющий размерность  $[1:n]$ ; все значения аргумента различны и образуют монотонно возрастающую последовательность;

$y$  — массив табличных значений функций, имеющий размерность  $[1:n]$ ;

$jt$  — параметр, задающий режим выполнения процедуры *difint*. Для интерполяции нужно положить  $jt=1$ ; для дифференцирования  $jt=2$ , а для интегрирования  $jt=3$ ;

$xarg$  — значение аргумента, при котором нужно производить интерполяцию или дифференцирование. Если  $xarg$  находится вне интервала

( $x[1], x[n-1]$ ), то формулы Лагранжа работают как экстраполяционные. При интегрировании значение  $xarg$  несущественно;

$x1, x2$  — нижний и верхний пределы интегрирования. При интерполяции и дифференцировании значения  $x1$  и  $x2$  несущественны.

```

procedure difint(n,jt,xarg,x1,x2,x,y) result: (res);
    value n,jt,xarg,x1,x2; real xarg,x1,x2,res;
    integer n,jt; array x,y;
begin real ca,cb,cc,a,b,c,s1,s2,t1,t2,t3,sum;
    integer j,js,j2,i,j1; switch beta:=no9,no5,no6;
start: if jt=3 then go to intgr;
    if xarg  $\geq$  x[n-1] then
        begin j:=n-1; js:=1; go to term end;
    if xarg  $\leq$  x[2] then
        begin j:=2; js:=1; go to term end;
comment Определение места аргумента в таблице;
    js:=2;
    for i:=2 step 1 until n do
        begin if xarg < x[i] then go to term; j:=i end;
no5:   ca:=a; cb:=b; cc:=c;
    js:=3; j:=j+1;
    go to term;
no6:   a:=(ca+a)/2; b:=(cb+b)/2; c:=(cc+c)/2;
no9:   if jt=2 then go to diff;
    intrp:res:=a $\times$ xarg2+b $\times$ xarg+c;
    go to fin;
    diff: res:=2 $\times$ a $\times$ xarg+b;
    go to fin;
    intgr:sum:=0; s1:=x1;
    j2:=n; j1:=2;
    for i:=1 step 1 until n do
        begin
            if x1  $\leq$  x[i] then go to no17;
            j1:=j1+1
        end;
no17: for i:=1 step 1 until n do
        begin j2:=j2-1;
            if x2  $\geq$  x[j2+1] then go to lagr
        end i;
    term: j1:=j2:=j;
    lagr: for j:=j1,j1+1 step 1 until j2 do
        begin a:=x[j-1]-x[j];
            b:=x[j-1]-x[j+1]; c:=a-b;
            t1:=y[j-1]/(a $\times$ b);
            t2:=y[j]/(a $\times$ c);
            t3:=-y[j+1]/(b $\times$ c);
            a:=t1+t2+t3;
            b:=- (x[j]+x[j+1]) $\times$ t1-(x[j-1]+x[j+1]) $\times$ t2
                -(x[j-1]+x[j]) $\times$ t3;
            c:=x[j] $\times$ x[j+1] $\times$ t1+x[j-1] $\times$ x[j+1] $\times$ t2+x[j-1] $\times$ x[j] $\times$ t3;
            if jt $\neq$ 3 then go to beta[js];
            if j=j1 then
                begin ca:=a; cb:=b; cc:=c; end else

```

```

begin
  ca:=(a+ca)/2; cb:=(b+cb)/2; cc:=(c+cc)/2
end;
s2:=x[j];
sum:=sum+ca×(s2↑3—s1↑3)/3+cb×(s2↑2—s1↑2)/2
+cc×(s2—s1);
ca:=a; cb:=b; cc:=c;
s1:=s2
end j;
res:=sum+ca×(x2↑3—s1↑3)/3+cb×(x2↑2—s1↑2)/2+cc×
(x2—s1);
fin: end difint;

```

### Свидетельство к алгоритму 77б

Алгоритм 77б получен из алгоритма 77а путем внесения в него модификации, предложенной Э. М. Каплинским в его «Замечании к алгоритму 77а» [38].

Алгоритм 77б был транслирован на машине М-220 в системе ТА—1М, и с ним были проведены расчеты для значений параметров  $x_1=0$ ;  $x_2=2$ ;  $x[i]=x_1+(i-1)(x_2-x_1)/n$ , где  $n=5, 10, 20, 50, 100, 200$ ;  $y[i]=\exp(x[i])$  для  $i=1, 2, \dots, n+1$ ;  $xarg=0.5, 0.9, 1, 1.077, 1.57, 2.1, 2.5$ ;  $jt=1, 2, 3$ . Некоторые из результатов приведены в табл. 12 (для  $n=200$ ) и 13 (для  $xarg=1.57$ ).

Таблица 12

$xarg$	Результаты трансляции		Контрольные значения
	Интерполяция	Дифференцирование	
0.5	1.6487213	1.6487073	1.6487212
0.9	2.4596031	2.4595822	2.4596031
1.0	2.7182819	2.7182586	2.7182818
1.077	2.9358582	2.9358646	2.9358587
1.57	4.8066479	4.8066064	4.8066481
2.1	8.1640434	8.1120029	8.1661699
2.5	11.988269	11.009126	12.182493

Таблица 13

$n$	Результаты трансляции		
	Интерполяция	Дифференцирование	Интегрирование
5	4.8118901	4.6542744	6.3520786
10	4.8068337	4.8055027	6.3859727
20	4.8066721	4.8078594	6.3888324
50	4.8066456	4.8067138	6.3890498
100	4.8066484	4.8067283	6.3890554
200	4.8066479	4.8066064	6.3890563
Контрольные значения	4.8066481	4.8066481	6.3890561

Контрольные значения для табл. 12 были получены в результате вычисления значений  $e^{xarg}$  машиной М-220 по стандартной программе

системы ТА—1М. Последние две строчки в табл. 12 показывают гораздо большие погрешности интерполяции и дифференцирования вследствие того, что в этих случаях ( $xarg=2.1$  и  $2.5$ ) формулы алгоритма работают как экстраполяционные.

Контрольные значения для интегрирования были получены в результате вычисления машиной М-220 значения интеграла  $\int_0^2 e^x dx = e^2 - 1$  по стандартной программе системы ТА-1М. Точно такое же значение  $e^2$  можно найти в таблицах Б. И. Сегала и К. А. Семендяева [37, с. 450].

### Свидетельство к алгоритму 77а

Алгоритм 77а получен в результате исправления, сокращения и ординарной переработки алгоритма 77 (Hennion Р. Е. «САСМ», 1962, № 2). Кроме ошибок, указанных в нижеследующих «Подтверждении» и «Замечании», в алгоритме 77 было исправлено следующее.

1. Метка *start* стояла перед описанием. По-видимому, она должна стоять перед первым оператором процедуры.

2. Оператор, следующий после

**comment** End of loop on [jm] index;

должен иметь вид

$sum := sum + ca \times (xup \uparrow 3 - syl \uparrow 3) / 3 + cb \times (xup \uparrow 2 - syl \uparrow 2) / 2 + cc \times (xup - syl);$

В алгоритме 77а этому оператору соответствует оператор в седьмой строке от конца процедуры.

3. В примечании к алгоритму 77 отсутствовали сведения о значениях формальных параметров в заголовке процедуры.

Идентификаторы алгоритма 77а соответствуют идентификаторам алгоритма 77 следующим образом:

$n - nop$	$i - ia$	$s1 - syl$	$no17 - L17$
$x1 - xlo$	$t1 - term1$	$s2 - syu$	$intrp - L10$
$x2 - xup$	$t2 - term2$	$no5 - L5$	$diff - L11$
$x - xa$	$t3 - term3$	$no6 - L6$	$intgr - L12$
$y - ya$	$j1 - ib$	$no9 - L9$	$larg - L16$
$j - jm$	$j2 - jul$		

Перевод «Замечания к алгоритму 77» (Hennion Р. Е. «САСМ», 1963, № 8) здесь не приводится, поскольку в дальнейшем был опубликован его исправленный вариант («САСМ», 1963, № 11).

Алгоритм 77а проверен с помощью транслятора ТА—1 для функций  $e^x$ ,  $\log x$  при  $xarg=2.5$ ,  $n=21$ ,  $x1=1$ ,  $x2=2$  с равным шагом по аргументу\*.

\* Результаты трансляции, приводившиеся ранее в «Свидетельстве к алгоритму 77а» [23], здесь опущены, поскольку в «Свидетельстве к алгоритму 77б» даны более точные, детальные и показательные результаты решения по этому алгоритму. (Прим. ред.)

## Подтверждение к алгоритму 77

В. Уиттер (Whitter V. E. «САСМ», 1962, № 6)

Эта процедура была переведена на язык ВАС-220 (диалект языка АЛГОЛ-60) и проверена на машине Burroughs 220. Были обнаружены следующие небольшие ошибки... \*

После внесения вышеуказанных поправок процедура была проведена для интерполяции, дифференцирования и интегрирования функций  $e^x$ ,  $\log X$  и  $\sin X$  в интервале  $1 \leq X \leq 5$ . Двадцать одно значение каждой из функций, взятые с равным шагом по  $X$  и с точностью до семи значащих цифр, были сведены в таблицу на вышеуказанном интервале. Затем была проверена процедура; табл. 14 показывает полученную точность.

Таблица 14

Функция	Число правильных значащих цифр		
	Интерполирование	Дифференцирование	Интегрирование
$e^x$	$\geq 4^*)$	$\geq 2$	$\geq 4$
$\log X$	$\geq 4^*)$	$\geq 2$	$\geq 3$
$\sin X$	$\geq 4^*)$	$\geq 2$	$\geq 4$

\*) За исключением случая интерполяции между первыми двумя точками.  
(Прим. В. Уиттера)

Эти результаты вполне приемлемы, если учесть сравнительно большой шаг по  $X$ . Тесты, использующие меньшие и неравные шаги по  $X$ , тоже были удовлетворительными.

Также было обнаружено, что для интегрирования существуют следующие ограничения:

- 1)  $xlo \leq xa[1]$ ;
- 2)  $xup \leq xa[nor]$ .

## Замечание к алгоритму 77

П. Е. Хенъон (Henpion P. E. «САСМ», 1963, № 11)

Мое внимание было привлечено к словам в «Подтверждении к алгоритму 77» (Whitter V. E. «САСМ», 1962, № 6) о том, что существуют ограничения на верхнюю и нижнюю границы интегрирования, т. е. 1)  $xlo \leq xa[1]$ , 2)  $xup \leq xa[nor]$ . Чтобы снять эти ограничения, нужно сделать следующие два изменения... \*\*

## АЛГОРИТМ 786

### Корни полиномов с целыми коэффициентами, получаемые в форме простых дробей [С2]

Процедура *ratfact* (сокращение от *rational* — рациональный и *factor* — множитель) предназначена для нахождения рациональных корней полиномов с целыми коэффициентами. Здесь используется предло-

\* Далее указываются три поправки к алгоритму 77, учтенные при составлении алгоритма 77а. (Прим. ред.)

\*\* Далее указываются два изменения, учтенные при составлении алгоритма 77а. (Прим. ред.)

женное Дж. Пеком расширение хорошо известного метода вычисления полиномов по схеме Горнера. Полином  $f(x) = a_0 + a_1x + \dots + a_nx^n$  с целыми коэффициентами и с  $a_0 \times a_n \neq 0$  имеет в качестве корня несократимую дробь  $p/q$  тогда и только тогда, когда  $a_0q^n + a_1q^{n-1}p + \dots + a_{n-1}qp^{n-1} + a_np^n = 0$ . Кроме того,  $q$  должно быть множителем  $a_n$ , а  $p$  — множителем  $a_0$ . Ненулевые рациональные корни  $p/q$  выводятся на внешний носитель информации с помощью процедуры *outreal* через канал с номером  $k$ . Массив  $a$  имеет размерность  $[0:n]$ .

```

procedure ratfact(a,n,k);
  value n,k; integer n,k; integer array a;
begin integer i,p,q,r,t,f,g,a0,an;
  a0:=abs(a[0]); an:=abs(a[n]);
  for p:=1 step 1 until a0 do
    begin comment Если p не множитель коэффициента a[0]
      или q не множитель коэффициента a[n], то переходим
      на конец цикла для продвижения в соответствующем
      списке цикла;
      if (a0÷p)×p≠a0 then go to finp;
      for q:=1 step 1 until an do
        begin if (an÷q)×q≠an then go to finq;
        comment Для проверки, является ли p/q корнем,
        вычисляем полином. Определение значения г дает
        экономиию одного вычисления переменной с индексом;
        f:=g:=a[0]; t:=p;
        for i:=1 step 1 until n do
          begin r:=a[i]×t;
            f:=f×q+r; g:=-g×q+r;
            t:=t×p
          end i;
          if f=0 then
            begin outreal(k,p); outreal(k,q) end;
          if g=0 then
            begin outreal(k,-p); outreal(k,q) end;
        end q;
      finq;
    finp;
  end p
end ratfact;

```

### Свидетельство к алгоритму 78б

Процедура *ratfact* алгоритма 78б является стереотипным переизданием процедуры *ratfact* алгоритма 78а. В пояснительные тексты и в «Подтверждения» были внесены некоторые уточнения редакторского характера.

### Свидетельство к алгоритму 78а

Алгоритм 78а получен в результате исправления, ординарной переработки и некоторой модификации алгоритма 78 (Perry C. «САСМ», 1962, № 2).

Модификация состояла в добавлении двух переменных  $a0$  и  $an$  и двух операторов  $a0:=abs(a[0])$ ; и  $an:=abs(a[n])$ ; что дает сокращение времени работы алгоритма. Кроме того, процедура *print* была заменена стандартной процедурой *outreal*, что заставило ввести новый фор-

мальный параметр  $k$  — номер канала. Исправлена ошибка: удалены границы индексов в спецификации массива  $a$ .

Алгоритм был проверен с помощью транслятора TA—1 для следующих полиномов:

1)  $3 + 14x + 3x^2 - 36x^3$ ,  $n = 3$ .

Результат:  $x_1 = -1/3$ ,  $x_2 = 3/4$ ,  $x_3 = -3/9$ .

2)  $3 - x - 22x^2 + 24x^3$ ,  $n = 3$ .

Результат:  $x_1 = 1/2$ ,  $x_2 = -1/3$ ,  $x_3 = 3/4$ ,  $x_4 = 3/6$ .

В обоих случаях последний корень совпадает по значению с первым.

### Подтверждение к алгоритму 78

М. Халстид (Halstead M. H. «CACM», 1962, № 3)

Процедура *ratfact* была переведена на язык NELIAC и проверена на машине UNIVAC-M-190 Countess и на машине CDC 1604. Были вычислены корни полиномов порядка от двух до шести. Не потребовалось никаких исправлений. Было замечено, что для полинома, коэффициенты которого имеют общий множитель, будут выдаваться избыточные значения  $p/q$ , в которых эта дробь была действительным корнем, но лишь одно, в котором значения  $p$  и  $q$  содержат общий множитель.

### Подтверждение к алгоритму 78

Д. М. Коллисон (Collision D. M. «CACM», 1962, № 8)

Алгоритм был успешно проверен с помощью транслятора Elliott ALGOL на машине National Elliott 803. Было замечено, что кратный рациональный корень может быть напечатан процедурой только один раз.

## АЛГОРИТМ 796

### Коэффициенты полиномиальной аппроксимации производной любого порядка от табличной функции [D4]

Процедура *dicol* выдает коэффициенты *coef* для  $n$  ординат (соответствующих абсциссам *x<sub>tab</sub>*) в  $n$ -точечном конечно-разностном выражении для  $k$ -й производной, вычисляемой в точке *x<sub>p</sub>*. Используемый метод состоит в определении аналитического выражения для  $k$ -й производной каждого коэффициента в  $n$ -точечной интерполяционной формуле Лагранжа и вычислении этого выражения в точке *x<sub>p</sub>*. При  $k = 0$  будут вычисляться сами интерполяционные коэффициенты Лагранжа.

```
procedure dicol(k,n,xp,xtab) result: (coef);  
  value k,n,xp; real xp; integer k,n; array xtab,coef;  
begin real factk,sum,denom,part; integer i,terms,j,m,high;  
  integer array xuse[1:n-1];  
  factk:=1;  
  for i:=2 step 1 until k do factk:=i×factk;  
  terms:=n-k-1;  
  if terms<0 then go to zz;  
  for j:=1 step 1 until n do  
    begin sum:=0; denom:=part:=1;
```

```

for i:=1 step 1 until n do
  if i≠j then denom:=denom×(xtab[j]—xtab[i]);
  if terms=0 then go to yy;
  m:=high:=1;
aa:  if (high=j) ∨ (xtab[high]=xp) then
      begin high:=high+1; go to aa end;
  if high>n then
      begin m:=m—1;
        if m≤0 then go to xx;
        high:=xuse[m]+1; go to aa
      end;
  xuse[m]:=high; m:=m+1;
  if m≤terms then
      begin high:=high+1; go to aa end;
  for i:=1 step 1 until terms do
      part:=part×(xp—xtab[xuse[i]]);
  sum:=sum+part; m:=terms;
  part:=1; high:=xuse[terms]+1;
  go to aa;
yy:  sum:=1;
xx:  coef[j]:=sum×factk/denom
      end j;
go to fin;
zz:  for i:=1 step 1 until n do coef[i]:=0;
fin:  end dicol;

```

### Свидетельство к алгоритму 796

Процедура *dicol* алгоритма 796 является стереотипным переизданием процедуры *dicol* алгоритма 79а. В пояснительный текст алгоритма 796 было внесено несколько изменений редакторского характера.

Алгоритм 796 был транслирован в системе ТА—1М для М-220 с входными параметрами  $n=3$ ;  $k=0, 1, 2$ ;  $xp=0, 2$ ;  $xtab=1, 3, 6$ . Результаты трансляции даны в табл. 15.

Выражения для коэффициентов Лагранжа при  $n=3$  ( $k=0$ ) даны, например, в работе П. В. Мелентьева [33, с. 127]. Выражения для  $coef[i]$  при  $k=1$  и  $k=2$  легко получаются из выражений для  $coef[i]$  при  $k=0$  обычным дифференцированием.

Имея табличные значения функции  $y[i]$  в опорных (узловых) точках  $xtab[i]$  и вычислив  $coef[i]$  с помощью процедуры *dicol*, легко получить значения как самой функции так и любой ее производной в точке  $xp$  по формуле

$$y^{(n)} = \sum_{i=1}^n coef[i] y[i].$$

Например, для функции  $y=x^2$  в точке  $xp=2$  получаем

$$y=y^{(0)}=2/5 \times 1 + 2/3 \times 9 - 1/15 \times 36 = 4,$$

$$y'=y^{(1)}=-1/2 \times 1 + 1/2 \times 9 + 0 \times 36 = 4,$$

$$y''=y^{(2)}=1/5 \times 1 - 1/3 \times 9 + 2/15 \times 36 = 2.$$

Это полностью совпадает с контрольными значениями  $y(xp)=2^2=4$ ;  $y'(xp)=2 \times 2=4$   $y''(xp)=2$ .



Таблица 15

$k$	Аналитические выражения для coef[i]	Контрольные значения coef [i]	Результаты трансляции	
			$xp=0$	$xp=2$
0	$\text{coef [1]} = \frac{(x-x_3)(x-x_2)}{(x_1-x_3)(x_1-x_2)}$	$1 \frac{4}{5} \frac{2}{5}$	1.8000000	0.4000000
	$\text{coef [2]} = \frac{(x-x_3)(x-x_1)}{(x_2-x_3)(x_2-x_1)}$	$-1 \frac{2}{3}$	-1.0000000	0.6666666
	$\text{coef [3]} = \frac{(x-x_2)(x-x_1)}{(x_3-x_2)(x_3-x_1)}$	$\frac{1}{5} -\frac{1}{15}$	0.2000000	-0.0666666
1	$\text{coef [1]} = \frac{2x-x_2-x_3}{(x_1-x_3)(x_1-x_2)}$	$-\frac{9}{10} -\frac{1}{2}$	-0.9000000	-0.5000000
	$\text{coef [2]} = \frac{2x-x_3-x_1}{(x_2-x_3)(x_2-x_1)}$	$-1 \frac{1}{6} \frac{1}{2}$	-1.1666666	0.5000000
	$\text{coef [3]} = \frac{2x-x_1-x_2}{(x_3-x_2)(x_3-x_1)}$	$-\frac{4}{15} 0$	-0.2666666	0
2	$\text{coef [1]} = \frac{2}{(x_1-x_3)(x_1-x_2)}$	$\frac{1}{5} \frac{1}{5}$	0.2000000	0.2000000
	$\text{coef [2]} = \frac{2}{(x_2-x_3)(x_2-x_1)}$	$-\frac{1}{3} -\frac{1}{3}$	-0.3333333	-0.3333333
	$\text{coef [3]} = \frac{2}{(x_3-x_2)(x_3-x_1)}$	$\frac{2}{15} \frac{2}{15}$	0.1333333	0.1333333

### Свидетельство к алгоритму 79a

Алгоритм 79a получен в результате ординарной переработки и некоторых очевидных сокращений алгоритма 79 (Giampio T. P. «CASM», 1962, № 2).

### Подтверждение к алгоритму 79

Ева С. Кларк (Clark Eva S. «CASM», 1963, № 3)

Эта процедура была переведена на язык ФОРТРАН и проверена на машине CDC 1604. Приемлемая точность была получена для  $k=0$ ,  $4 \leq n \leq 12$ . При возрастании  $n$  и  $k$  точность снижается. Было обнаружено, что при возрастании  $n$  время выполнения процедуры быстро растет. Для  $k=0$  получены результаты, приведенные в табл. 16.

Таблица 16

$n$	Примерное количество машинных операций
4	$1.3_{10}3$
6	$6.9_{10}3$
8	$3.8_{10}4$
10	$1.8_{10}5$
12	$8.6_{10}5$

Автор алгоритма 79 указал в письме, что процедура была составлена для использования с малым  $n$  и малым  $k$ .

## АЛГОРИТМ 806

### Вычисление обратной гамма-функции с точностью до 10 цифр [S14]

Процедура *rgr80* вычисляет обратную вещественную гамма-функцию  $1/\Gamma(x)$  для любых вещественных значений  $x$  при использовании схемы Горнера для вычисления аппроксимирующего полинома. Погрешность метода аппроксимации равна  $\pm 2^{-35}$ .

```
real procedure rgr80(x);
  value x; real x;
begin real y;
  y:=1;
aa:  if x>1 then
    begin x:=x-1; y:=y×x; go to aa end;
  y:=1/y;
cc:  if x<-1 then
    begin y:=y×x; x:=x+1; go to cc end;
rgr80:= if x=1 then y else
  y×(((((((((-0.00000018122×x+0.000001328554)×x
    -0.000002625721)×x-0.000017527917)×x
    +0.000145624324)×x-0.000360851496)×x
    -0.000804341335)×x+0.008023278113)×x
    -0.017645242118)×x-0.024552490887)×x
    +0.191091101162)×x-0.233093736365)×x
    -0.422784335092)×x+1)×x×(x+1)
end rgr80;
```

### Свидетельство к алгоритму 806

Алгоритм 806 составлен, по существу, заново в значительно более краткой и экономной форме, чем алгоритм 80а. Для экономии памяти и машинного времени тело процедуры *rgam* алгоритма 80а было использовано внутри процедуры *rgr80* в последнем из ее операторов. Это тело было предварительно модифицировано так, чтобы не требовался вспомогательный массив  $b[0:12]$ , и не вычислялись бы переменные с индексами.

Алгоритм 806 был транслирован на машине М-220 в системе ТА-1М и дал результаты, приведенные в табл. 17.

Таблица 17

$x$	$rgr(80)$	$x$	$rgr(80)$	$x$	$rgr(80)$
2,3	0.85710962	5/3	1.1077321	0,5	0.56418958
7/4	1.0880652	4/3	1.1198465	-0,2	-0.17178740
5/4	1.1032626	2/3	0.73848811	-0,5	-0.28209479
3/4	0.81604893	1/3	0.37328217	-1,2	0.20614488
1/4	0.27581566	1	1.0000000	-1,5	0.42314218

Эти результаты полностью совпали с контрольными значениями  $1/\Gamma(x)$ , взятыми из работы Е. Янке и др. [36, с. 54 и 59] и справочника Б. И. Сегала и К. А. Семендяева [37, с. 450] (в последнем для значений  $1/\sqrt{\pi}$  и  $1/2\sqrt{\pi}$ ).

Смотрите также «Замечание к алгоритмам 34, 54, 80, 221 и 291» М. Пайка и И. Хилла, помещенное в предыдущем выпуске [38] вслед за «Свидетельством к алгоритму 34б».

### Свидетельство к алгоритму 80а

Алгоритм 80а получен в результате исправления, сокращения и ординарной переработки алгоритма 80 (Holsten W. «CASM», 1962, № 3).

В алгоритме 80 были обнаружены следующие ошибки.

1. В процедуре RGR спецификация **real procedure** RGAM не нужна и должна быть вычеркнута.

2. В процедуре RGAM описания **integer i;** и **real array B[0:13];** должны быть перенесены из совокупности спецификаций в начало блока тела процедуры.

3. В примечании к процедуре RGR формула (2) должна иметь такой вид, какой она имеет в описании алгоритма 80а (см. [1, с. 162]).

4. В процедуре RGR после первых двух условных операторов должны быть точки с запятой.

Алгоритм 80а проверен с помощью транслятора ТА-1 для трех значений  $x$  и получены следующие результаты (табл. 18).

Таблица 18

$x$	Результаты трансляции	Результаты, вычисленные по справочнику [1]
—0.2	—0.171787403	—0.17178
0.3	0.334272752	0.33427
0.5	0.564189583	0.56418

### Свидетельство к алгоритмам 81б, 82б, 83б [G7]

Алгоритмы 81б, 82б и 83б не публикуются здесь, потому что правильность соответствующих им алгоритмов 81, 82 и 83 («CASM», 1962, № 3), равно как и алгоритмов 81а, 82а, 83а, не была подтверждена ни в журнале «CASM» ни в расчетах авторов данного выпуска.

При новом просмотре алгоритма 81а в нем была обнаружена опечатка. На с. 67 [23] в 12-й строке снизу вместо

$C[k] := \text{true};$

должно быть

$E: C[k] := \text{true};$

Кроме того, как алгоритм 81, так и алгоритм 81а содержит явную ошибку: на с. 67 [23] в последней строке условие

$\text{if prices}[j] < i \text{ then}$

при  $j=1$  не выполняется, так как вначале  $i=0$ . Следовательно, встречающаяся далее переменная  $l$  остается неопределенной и последний в теле процедуры оператор цикла выполняться не может.

## АЛГОРИТМ 846

### Вычисление интеграла по Симпсону от таблично заданной функции [D1]

Процедура *sim1* дает приближенное значение определенного интеграла непрерывной функции  $y=f(x)$  на интервале  $[a,b]$ , заданной в виде таблицы значений  $y_0, y_1, \dots, y_n$  с постоянным шагом, где  $n$  — четное число,  $y_0=f(a)$  и  $y_n=f(b)$ . Вычисление ведется по формуле Симпсона

$$\int_a^b f(x) dx \cong \frac{b-a}{3n} (y_0 + 4y_1 + 2y_2 + \dots + 4y_{n-1} + y_n).$$

```
real procedure sim1 (n,a,b,y);  
  value a,b,n; real a,b; integer n; array y;  
begin real s; integer i;  
  s:=(y[0]-y[n])/2;  
  for i:=2 step 2 until n do s:=s+2×y[i-1]+y[i];  
  sim1:=2×(b-a)×s/(3×n)  
end sim1;
```

#### Свидетельство к алгоритму 846

Процедура *sim1* алгоритма 846 получена в результате замены в процедуре *sim* алгоритма 84а оператора

```
for i:=1 step 2 until n-1 do s:=s+2×y[i]+y[i+1];
```

на оператор

```
for i:=2 step 2 until n do s:=s+2×y[i-1]+y[i];
```

для ускорения выполнения процедуры.

Алгоритм 846 был транслирован на машине М-220 в системе ТА-1М. С помощью процедуры *sim1* вычислялся интеграл

$$\int_0^{\pi/2} \cos x dx = 1.$$

Результаты приведены ниже.

$n$	2	4	10	20
<i>sim1</i>	1.0022798	1.0001345	1.0000033	1.0000002

#### Свидетельство к алгоритму 84а

Алгоритм 84а получен в результате ординарной переработки алгоритма 84 (Hennion P. E. «CACM», 1962, № 4). Переводы «Замечания к алгоритму 84» («CACM», 1962, № 8) и «Подтверждения к алгоритму 84» («CACM», 1962, № 11) здесь не приводятся, поскольку они не добавляют новой информации к нижеследующему «Подтверждению» А. П. Релфа.

#### Подтверждение к алгоритму 84

А. П. Релф (Relph A. P. «CACM», 1962, № 7)

Этот алгоритм был транслирован с помощью транслятора DEUCE ALGOL и дал удовлетворительные результаты без всяких исправлений.

В примечании к алгоритму не оговаривается, что  $n$  должно быть обязательно четным.

### АЛГОРИТМ 856

#### Вычисление собственных значений и собственных векторов симметричной матрицы методом Якоби [F2]

Процедура *jacobi* находит все собственные значения и собственные векторы данной симметричной матрицы  $a[1:n, 1:n]$  с помощью модифицированного итерационного метода Якоби [17i].

На выходе из процедуры  $k$ -й столбец матрицы  $s$  содержит  $k$ -й из  $n$  собственных векторов данной матрицы, диагональный элемент  $a[k, k]$  массива  $a$  является соответствующим  $k$ -м собственным значением. Начальное значение массива  $s$  несущественно. Параметр  $rho$  (требуемая точность) введен в вышеуказанной работе, где дается детальная схема метода. Назначение параметра  $rho$  состоит в том, что итерация заканчивается, когда для каждого недиагонального элемента  $a[i, j]$  выполняется неравенство  $abs(a[i, j]) < (rho/n) \times norm1$ , где  $norm1$  является функцией только недиагональных элементов первоначальной матрицы.

```

procedure jacobi (n, rho) data result: (a) result: (s);
    value n, rho; real rho; integer n; array a, s;
begin real norm1, norm2, thr, mu, omega, sint, cost, intl, v1, v2, v3;
    integer i, j, p, q, ind;
    comment Формирование в массиве s единичной матрицы;
    for i:=1 step 1 until n do
        for j:=1 step 1 until i do
            if i=j then s[i, j]:=1 else s[i, j]:=s[j, i]:=0;
    comment Вычисление начальной нормы (norm1), конечной нормы (norm2) и порога (thr);
    intl:=0;
    for i:=2 step 1 until n do
        for j:=1 step 1 until i-1 do intl:=intl+2×a[i, j]↑2;
    if intl=0 then go to fin;
    norm1:=thr:=sqrt(intl);
    norm2:=(rho/n)×norm1;
    ind:=0;
main: thr:=thr/n;
    comment Здесь начинается обработка недиагональных элементов;
main1: for q:=2 step 1 until n do
    for p:=1 step 1 until q-1 do
        if abs(a[p, q]) ≥ thr then
            begin ind:=1;
                v1:=a[p, p]; v2:=a[p, q]; v3:=a[q, q];
                mu:=0.5×(v1-v3);
                omega:= if mu=0 then -1 else
                    -sign(mu)×v2/sqrt(v2↑2+mu↑2);
                sint:=omega/sqrt(2×(1+sqrt(1-omega↑2)));
                cost:=sqrt(1-sint↑2);
                for i:=1 step 1 until n do

```

```

begin
  if i≠p ∧ i≠q then
    begin int1:=a[i,p]; mu:=a[i,q];
      a[q,i]:=a[i,q]:=int1×sint+mu×cost;
      a[p,i]:=a[i,p]:=int1×cost—mu×sint
    end;
    int1:=s[i,p]; mu:=s[i,q];
    s[i,q]:=int1×sint+mu×cost;
    s[i,p]:=int1×cost—mu×sint
  end i;
  mu:=sint↑2; omega:=cost↑2; int1:=sint×cost;
  a[p,p]:=v1×omega+v3×mu—2×v2×int1;
  a[q,q]:=v1×mu+v3×omega+2×v2×int1;
  a[p,q]:=a[q,p]:=(v1—v3)×int1+v2×(omega—mu)
end p;
comment Теперь проверка достижения конечной точности;
if ind=1 then begin ind:=0; go to main1 end;
if thr>norm2 then go to main;
fin: end jacobi;

```

#### Свидетельство к алгоритму 85б

Алгоритм 85б получен из алгоритма 85а путем внесения в него поправки, предложенной В. Ф. Дейкиным в нижеследующем его «Замечании к алгоритму 85а», и модификации, предложенной П. Науром в нижеследующем его «Подтверждении к алгоритму 85». Кроме того, было сделано несколько тождественных преобразований для экономии машинного времени.

#### Свидетельство к алгоритму 85а

Алгоритм 85а получен в результате исправления, некоторого сокращения и ординарной переработки алгоритма 85 (Evans T. G. «САСМ», 1962, № 4).

Более подробные сведения о методе Якоби можно получить, например, в работе Д. К. Фаддеева и В. Н. Фаддеевой [4, § 81].

Алгоритм 85а транслирован с исходными данными  $n=4$ ,  $\rho=10^{-8}$  и

$$a = \begin{pmatrix} 1.00 & 0.42 & 0.54 & 0.66 \\ 0.42 & 1.00 & 0.32 & 0.44 \\ 0.54 & 0.32 & 1.00 & 0.22 \\ 0.66 & 0.44 & 0.22 & 1.00 \end{pmatrix}.$$

Результаты трансляции:

$$s = \begin{pmatrix} 0.579642502 & -0.0503284495 & -0.718845953 & 0.380449881 \\ 0.459996665 & 0.237226458 & -0.0956989810 & -0.850275473 \\ 0.433459111 & -0.812846170 & 0.387435463 & -0.0358896058 \\ 0.514325614 & 0.529595844 & 0.569206432 & 0.361941215 \end{pmatrix};$$

$$a = \begin{pmatrix} 2.32274880 & 0.100051435_{10}-11 & 0.479925056_{10}-14 & 0.000000000 \\ 0.100051435_{10}-11 & 0.796706689 & 0.000000000 & 0.179516047_{10}-14 \\ 0.479925056_{10}-14 & 0.000000000 & 0.242260708 & 0.000000000 \\ 0.000000000 & 0.179516047_{10}-14 & 0.000000000 & 0.638283803 \end{pmatrix}.$$

Согласно работе Фаддеева Д. К. и Фаддеевой В. Н. [4, с. 575—577], исходная матрица должна иметь собственные значения 2.32274884, 0.63828379, 0.79670672, 0.24226073 и третий собственный вектор (—0.71884900, —0.09569928, 0.38743715, 0.5692089).

### Замечание к алгоритму 85а

Г. Г. Дядюша, Киев, октябрь 1966

Алгоритм 85а может быть существенно усовершенствован, если учесть следующие обязательства.

1. Для вычисления тригонометрических функций угла элементарного вращения  $\varphi$  достаточно двух операций извлечения квадратного корня.

2. Вместо  $\sin \varphi$  можно применять  $\operatorname{tg} \varphi$ , который удобно вычисляется по формуле

$$\operatorname{tg} \varphi = -a_{ij} / [\mu + \operatorname{sign}(\mu + \delta(\mu)) \mu_1],$$

$$\text{где } \mu = (a_{ij} - a_{ii})/2; \mu_1 = \sqrt{\mu^2 + a_{ij}^2}; \delta(\mu) = \begin{cases} 0, & \mu \neq 0, \\ 1, & \mu = 0. \end{cases}$$

3. Диагональные элементы преобразованной матрицы удобно вычисляются по формулам  $a'_{ii}, a'_{jj} = a_{ii} + \mu \pm \mu_1$ .

4. Элемент  $a'_{ij}$  должен быть равен нулю.

5. Целесообразно при каждой итерации выбирать преграду чуть ниже среднего квадратичного недиагональных элементов.

6. Достаточно хранить в памяти машины только верхнюю половину симметричной матрицы (как в алгоритме 67а).

7. Для многих программирующих программ удобно применять только одномерные массивы.

В приводимой ниже процедуре *jacobi2* исходная матрица записывается в массиве  $a[1:n \times (n+1)/2]$  в следующем порядке: в первых  $n$  ячейках — диагональные элементы, в последующих — элементы наддиагональной верхней треугольной матрицы построчно. Матрица  $s$  записывается в массиве  $s[1:n \times n]$  по столбцам.

**procedure** jacobi2(n,rho) dataresult:(a) result:(s);

value n,rho; real rho; integer n; array a,s;

**begin** real norm1,norm2,mu,mul,t,cos,d;

integer i,j,k,p,q,r,il,kl,r1,n1,n2,n3; Boolean g;

p:=0;

**for** i:=1 **step** 1 **until** n **do**

**for** j:=1 **step** 1 **until** n **do**

**begin** p:=p+1;

s[p]:= **if** i=j **then** 1 **else** 0

**end** i;

n1:=n-1; n2:=n1-1;

n3:=n×(n+1)/2; g:= **false**;

thr: norm1:=0;

**for** i:=n+1 **step** 1 **until** n3 **do** norm1:=norm1+a[i]↑2;

norm1:=sqrt(2×norm1)/n;

**if** g **then** g:= **false** **else** norm2:=norm1×rho;

**if** norm1<norm2 **then** norm1:=norm2;

k:=n;

**for** q:=0 **step** 1 **until** n2 **do**

**begin** d:=a[q+1];

```

for p:=q+1 step 1 until n1 do
  begin k:=k+1;
    if abs(a[k])>norm1 then
      begin t:=a[k]; a[k]:=0; g:= true;
        mu:=(a[p+1]-d)/2; mul:=sqrt(t↑2+mu↑2);
        if mu<0 then mul:=-mul;
        mu:=mu+mul; t:=t/mu;
        a[p+1]:=d:=d+mu; mul:=2×mul;
        d:=d-mul; cost:=sqrt(mu/mul);
        i:=q+n; r:=p+n;
        il:=n×q; r1:=n×p;
        for j:=0 step 1 until n1 do
          begin
            if j≠q ∧ j≠p then
              begin mu:=cost×a[i]; mul:=cost×a[r];
                a[r]:=mu×t+mul; a[i]:=mu-t×mul
              end ifj;
            il:=il+1; r1:=r1+1; k1:=n2-j;
            mu:=cost×s[il]; mul:=cost×s[r1];
            s[r1]:=mu×t+mul; s[il]:=mu-t×mul;
            i:= if j<q then i+k1 else i+1;
            r:= if j<p then r+k1 else r+1
          end j;
        end ifabs
      end p;
    a[q+1]:=d
  end q;
  if g then go to thr
end jacobi2; *

```

### Подтверждение к алгоритму 85а

Г. Г. Дядюша, Киев, октябрь 1967

Алгоритм 85а в разных вариантах неоднократно транслировался с помощью транслятора ТА-2 в виде составной части программ, предназначенных для квантово-химических расчетов, и всегда работал безотказно.

Кроме ординарной переработки на входной язык транслятора ТА-2 эти варианты отличались от процедуры *jacobi2*, приведенной в предыдущем «Замечании», следующими деталями.

1. Конструкции «...:=...:=...» и «if ... ∧ ... then) расписывались подробнее из-за того, что транслятор ТА-2 переводит их недостаточно экономно.

2. *norm2* не вычислялась, а задавалась вместо *rho*.

3. *norm1* вычислялась по другим формулам, например,

$$norm1 = \left\{ [1 + n \times (n - 1) \div 2]^{-1} \sum_{i < j} a_{ij}^2 \right\}^{1/2},$$

---

\*Процедура *jacobi2*, приведенная выше, отличается от представленной Г. Дядюшей тем, что в ней исправлены некоторые очевидные ошибки. После исправления процедура *jacobi2* была транслирована в системе ТА-1М и для примера, приведенного в «Свидетельстве к алгоритму 85а», дала точно такие же результаты, как и процедура *jacobi*. (Прим. ред.)



или же для вычисления *norm1* использовалась выборка из элементов  $a_{ij}$ , а не все недиагональные элементы, например

$$norm1 = \left( \frac{1}{n} \sum_{i=2}^n a_{1,i}^2 \right)^{1/2}.$$

4. Изменялся порядок присваивания значений элементам преобразованной матрицы, например, так, чтобы диагональные элементы после преобразования располагались примерно в порядке возрастания.

Заметим, что в процедуре *jacobi2*, приведенной в предыдущем «Замечании», все элементарные вращения являются собственными и определитель матрицы *s* всегда остается равным 1. Вариант, описанный в п. 4, этим свойством не обладает, и определитель матрицы *s*, полученной после такой процедуры, может оказаться равным  $-1$  (т. е. ортогональная матрица может оказаться несобственной).

#### Замечание к алгоритму 85а

В. Ф. Дейкин, Москва, май 1973

Алгоритм 85а (так же как и исходный алгоритм 85) дает правильное решение, строго говоря, не для всех симметричных матриц. Так, для тривиального случая, когда матрица *a* заведомо диагональна, алгоритм закикливается.

Действительно, для указанного вида матриц

$$int1 = \sum_{i=2}^n \sum_{j=1}^{i-1} 2 \times a[i, i] \uparrow 2 = 0,$$

поэтому и  $norm1 = thr = sqrt(int1) = 0$ , а также

$$norm2 = (rho/n) \times norm1 = 0.$$

Вследствие этого условие **if**  $abs(a[p,q]) \geq thr$ , находящееся в начале цикла после метки *main1*, удовлетворяется всегда, и величине *ind* тоже всегда присваивается значение единицы. Поэтому в результате проверки достижения конечной точности управление неизбежно передается на метку *main1*, т. е. происходит закикливание.

Поскольку при решении задач не всегда бывает известно, с каким видом матриц придется иметь дело, то, по-видимому, целесообразно внести в алгоритм следующие изменения.

1. После оператора

$norm1 := thr := sqrt(int1);$

добавить оператор

**if**  $norm1 = 0$  **then go to main2;**

2. Конец процедуры

**end jacobi;**

заменить на

**main2: end jacobi;**

В этом случае на выходе из процедуры матрица *a* сохраняет свой первоначальный вид, а матрица *s* превращается в единичную.

#### Подтверждение к алгоритму 85

Дж. Хилмор (Hillmore J. S. «CACM», 1962, № 8)

Оператор

$\omega ga := (\text{if } \mu = 0.0 \text{ then } 1 \text{ else } \text{sign}(\mu)) \times (-v2) / \sqrt{v2 \uparrow 2 + \mu \uparrow 2};$

был заменен на оператор  
 $\text{omega} := \text{if } \mu = 0.0 \text{ then } -1.0 \text{ else } -\text{sign}(\mu) \times v2 / \sqrt{v2^2 + \mu^2};$

Если  $\mu = 0$ , то первый оператор сводился к  $\text{omega} := -v2 / \sqrt{v2^2}$ ; и ошибка округления при вычислении квадратного корня могла сделать значение  $\text{omega}$  несколько большим единицы. В результате происходил аварийный останов при вычислении следующего оператора, когда делается попытка вычислить значение  $\sqrt{1 - \text{omega}^2}$ .

В модифицированной форме алгоритм был успешно проверен с использованием транслятора Elliott ALGOL на машине National Elliott 803. Задавались матрицы до 15-го порядка, для которых получились собственные значения и собственные векторы с общей точностью равной семи десятичным разрядам.

### Подтверждение к алгоритму 85

П. Наур (H a u r P. «САСМ», 1963, № 8)

Первоначально мы проверили алгоритм в системе GIER ALGOL с включением следующих поправок...\*

При более близком ознакомлении обнаружилось, однако, что значительное число лишних операций может быть исключено в самом внутреннем цикле путем замены двух операторов цикла в центре алгоритма одним оператором цикла следующим образом...\*\*

Эта переделка особенно существенна в системах, имеющих сравнительно медленно работающий механизм вычисления индексов, таких как GIER ALGOL, так как она исключает более чем три из восьми ссылок на индексные переменные.

Процедура *jacobi* была опробована на двух различных последовательностях матриц с известными собственными значениями. В обоих случаях отладочная программа была составлена для отыскания интервала погрешностей в собственных значениях, вычисленных процедурой *jacobi*. Дополнительно для проверки были использованы отношения  $a \times v - \lambda \times v = 0$  ( $a$  — данная матрица,  $v$  — собственный вектор,  $\lambda$  — соответствующее собственное значение) и  $a - (st) \times \lambda \times s = 0$  ( $s$  — матрица, столбцами которой служат собственные векторы,  $st$  — результат транспонирования матрицы  $s$ , а  $\lambda$  — диагональная матрица собственных значений). Для отладки были использованы тест-матрицы пересмотренного алгоритма 52, приведенного в журнале «САСМ», 1963, № 1, р. 39, и матрица, предложенная Х. Б. Хансеном (H a n s e n H. B.), с элементами

$$HBN[j,i] = HBN[i,j] = n+1-j \text{ для } j \geq i,$$

имеющая собственное значение

$$0.5 / (1 - \cos((2 \times i - 1) \times \pi / (2 \times n + 1))).$$

Результаты приведены в табл. 19 и 20 (табл. 19 — для тест-матрицы Хансена; табл. 20 — для тест-матрицы алгоритма 52). GIER ALGOL использует числа с плавающей запятой и 29 значащими цифрами.

\* Далее указываются три поправки к алгоритму 85, учтенные при составлении алгоритма 85а. (Прим. ред.)

\*\* Далее приводится участок процедуры *jacobi*, использованный при составлении алгоритма 85б. (Прим. ред.)

Таблица 19

Абсолютные погрешности в собственных значениях				Отклонение от равенства $a \times v - \lambda \times v = 0$						Отклонение от равенства $a - (st) \times \lambda \times s = 0$						
Порядок	$j$	Ошибка	$j$	Ошибка	Элемент	Вектор	Ошибка	Элемент	Вектор	Ошибка	Элемент	Вектор	Ошибка	Элемент	Вектор	Ошибка
$\rho = 1.0_{10} - 3$																
5	1	$-1.1_{10} - 6$	3	$5.2_{10} - 8$	1	1	$-1.7_{10} - 4$	1	3	$2.0_{10} - 4$	1	1	$-2.5_{10} - 4$	5	5	$1.0_{10} - 4$
10	9	$-7.9_{10} - 5$	8	$3.5_{10} - 5$	7	2	$-3.3_{10} - 3$	6	6	$3.0_{10} - 3$	1	1	$-4.2_{10} - 3$	6	7	$3.2_{10} - 3$
15	15	$-9.2_{10} - 5$	12	$3.7_{10} - 5$	6	3	$-1.7_{10} - 3$	11	13	$1.7_{10} - 3$	9	15	$-1.5_{10} - 3$	8	9	$1.8_{10} - 3$
$\rho = 1.0_{10} - 5$																
5	1	$-1.1_{10} - 6$	3	$6.0_{10} - 8$	2	5	$-1.3_{10} - 7$	5	2	$4.1_{10} - 8$	1	2	$-1.6_{10} - 7$	4	5	$4.5_{10} - 8$
10	1	$-1.2_{10} - 5$	2	$2.2_{10} - 7$	7	3	$-2.7_{10} - 5$	2	8	$2.2_{10} - 5$	7	7	$-2.4_{10} - 5$	2	8	$2.3_{10} - 5$
15	1	$-3.5_{10} - 5$	4	$3.9_{10} - 7$	11	9	$-6.4_{10} - 6$	7	2	$4.8_{10} - 6$	11	12	$-5.3_{10} - 6$	12	12	$4.7_{10} - 6$
$\rho = 1.0_{10} - 8$																
5	1	$-1.1_{10} - 6$	3	$6.0_{10} - 8$	2	5	$-1.3_{10} - 7$	4	2	$6.5_{10} - 9$	2	2	$-1.3_{10} - 7$	4	4	$3.0_{10} - 8$
10	1	$-1.2_{10} - 5$	2	$2.2_{10} - 7$	1	10	$-1.1_{10} - 6$	4	2	$6.4_{10} - 8$	1	2	$-5.7_{10} - 7$	9	9	$8.2_{10} - 8$
15	1	$-3.5_{10} - 5$	4	$3.9_{10} - 7$	1	14	$-3.4_{10} - 6$	4	2	$3.9_{10} - 7$	2	2	$-1.3_{10} - 6$	15	15	$8.9_{10} - 8$

Таблица 20

Абсолютные погрешности в собственных значениях					Отклонение от равенства $a \times v - \lambda \times v = 0$					Отклонение от равенства $a - (st) \times \lambda \times s = 0$						
Порядок	$i$	Ошибка	$j$	Ошибка	Элемент	Вектор	Ошибка	Элемент	Вектор	Ошибка	Элемент	Вектор	Ошибка			
$\rho = 1.0_{10}-5$																
5	4	$-1.0_{10}-8$	4	0.0	5	5	$-3.3_{10}-8$	5	4	$4.3_{10}-8$	5	5	$-5.1_{10}-8$	4	4	$3.9_{10}-8$
10	8	$-1.1_{10}-8$	4	0.0	7	7	$-1.2_{10}-8$	9	6	$1.3_{10}-8$	7	8	$-5.1_{10}-9$	6	6	$2.0_{10}-8$
15	13	$-1.1_{10}-8$	6	0.0	14	14	$-9.3_{10}-9$	10	10	$9.4_{10}-9$	8	9	$-1.9_{10}-9$	10	10	$1.3_{10}-8$
$\rho = 1.0_{10}-8$																
3	3	$-7.5_{10}-9$	1	$3.7_{10}-9$	3	1	$-2.8_{10}-9$	2	2	$9.3_{10}-9$	1	3	0.0	1	2	$1.9_{10}-8$
4	4	$-5.6_{10}-9$	3	0.0	2	2	$-4.5_{10}-9$	3	4	$3.3_{10}-9$	2	2	0.0	2	3	$9.3_{10}-9$
5	4	$-1.0_{10}-8$	1	0.0	5	4	$-4.9_{10}-9$	4	4	$5.8_{10}-9$	1	1	$-7.5_{10}-9$	3	4	$7.5_{10}-9$
6	4	$-4.7_{10}-9$	4	0.0	4	3	$-2.8_{10}-9$	5	4	$3.6_{10}-9$	1	6	$-2.3_{10}-10$	4	5	$9.3_{10}-9$
7	4	$-5.1_{10}-9$	5	0.0	6	6	$-2.8_{10}-9$	4	4	$3.4_{10}-9$	5	7	$-1.2_{10}-10$	5	6	$7.5_{10}-9$
8	7	$-7.5_{10}-9$	5	0.0	5	5	$-6.0_{10}-9$	5	6	$3.2_{10}-9$	3	8	$-1.2_{10}-10$	7	7	$9.3_{10}-9$
9	6	$-4.4_{10}-9$	7	0.0	6	5	$-5.1_{10}-9$	7	6	$3.2_{10}-9$	5	5	$-7.5_{10}-9$	8	8	$1.5_{10}-8$
10	8	$-1.5_{10}-8$	8	0.0	8	9	$9.3_{10}-9$	9	7	$7.0_{10}-9$	6	7	$-2.3_{10}-9$	9	9	$2.0_{10}-8$
11	10	$-7.5_{10}-9$	1	0.0	9	10	$-6.5_{10}-9$	8	11	$3.0_{10}-9$	1	1	$-3.1_{10}-9$	8	8	$7.5_{10}-9$
12	8	$-5.0_{10}-9$	11	0.0	10	6	$-7.0_{10}-9$	10	8	$2.4_{10}-9$	6	6	$-1.7_{10}-8$	4	4	$1.3_{10}-8$
13	12	$-1.1_{10}-8$	10	0.0	10	11	$-6.9_{10}-9$	12	10	$9.1_{10}-9$	7	7	$-3.0_{10}-8$	12	12	$3.2_{10}-8$
14	10	$-1.5_{10}-8$	4	0.0	13	13	$-1.1_{10}-8$	10	10	$6.7_{10}-9$	9	10	$-3.5_{10}-9$	6	6	$1.7_{10}-8$
15	13	$-1.1_{10}-8$	6	0.0	14	14	$-1.1_{10}-8$	11	10	$3.5_{10}-9$	8	9	$-3.0_{10}-9$	6	11	$7.5_{10}-9$

Трансляция программы, которая выдает одну из таких таблиц, заняла около 40 с. Время решения указано в табл. 21.

Таблица 21

$\rho$	$n$	Время решения		
		Для первоначального алгоритма, сек		Для пересмотренного алгоритма, сек
		тест-матрица алгоритма 52	тест-матрица Хансена	тест-матрица Хансена
$10^{-3}$	5			3
	10			22
	15			70
$10^{-5}$	5	3		5
	10	5	41	29
	15	13	148	99
$10^{-8}$	5	4	7	6
	6	5	12	
	7	5	18	
	8	5	25	
	10	13		38
	15	22		116

Из табл. 21 видно, что тест-матрица алгоритма 52 нетипична по отношению к обычному решению с помощью процедуры *jacobi*. Гораздо более высокая точность, полученная для этой матрицы, по сравнению с матрицей Хансена говорит о том же.

Алгоритмы, опубликованные Дж. Вилкинсоном («Num. Math.», 1962, 4, 354—376) тоже были успешно проверены в системе GIER ALGOL. Алгоритмы Вилкинсона приводят матрицу к трехдиагональной форме посредством метода Хаусхолдера и используют последовательности Штурма для нахождения собственных значений и обратную итерацию для нахождения собственных векторов. В GIER ALGOL этот метод примерно в 1.3 раза быстрее, чем алгоритм *jacobi* для рассмотренного здесь ряда матриц. Процедура *jacobi* имеет то преимущество, что собственные векторы строго ортогональны даже при кратных собственных значениях, кроме того, процедура *jacobi* имеет более простую логику. С другой стороны, алгоритмы Вилкинсона дают гораздо более высокую скорость, если отыскиваются только некоторые из собственных значений и собственных векторов, тогда как процедура *jacobi* всегда находит их все.

#### Свидетельство к алгоритмам 86б и 87б [G6]

Алгоритмы 86б и 87б не публикуются здесь, потому что для получения перестановок компонент вектора вместо алгоритмов 86 и 87 (Реек J. E., Schrack G. F. «CACM», 1962, № 4) позднее в журнале «CACM» были опубликованы более совершенные алгоритмы. См., например, «Замечания к алгоритмам 87, 102, 130 и 202» Р. Орд-Смита [26, с. 10].

### Свидетельство к алгоритмам 886, 896 и 906 [S20]

Алгоритмы 886, 896 и 906 не публикуются здесь, потому что для вычисления интегралов Френеля вместо алгоритмов 88, 89 и 90 (C u p - d i f f J. L. «САСМ», 1962, № 5) позднее в журнале «САСМ» был опубликован более совершенный алгоритм 244 (G r a y M. «САСМ», 1964, № 11), которому соответствует русский вариант — алгоритм 244а [26].

### Свидетельство к алгоритму 916 [E2]

Алгоритм 916 не публикуется здесь, потому что для аппроксимации таблиц с помощью полинома Чебышева вместо алгоритма 91 (N e w - h o u s e A. «САСМ», 1962, № 5) позднее в журнале «САСМ» был опубликован более совершенный алгоритм 318 (B o o t h r o y d J. «САСМ», 1967, № 12).

## АЛГОРИТМ 926

### Решение системы линейных алгебраических уравнений и обращение матрицы [F4]

Процедура *simult* (сокращение от *simultaneous* — совместная) решает уравнение  $u'x' = b'$  относительно вектора  $x'$ , где ' означает транспонирование. Число итераций задается параметром *kount*. Результат транспонирования матрицы  $u'$  задается в массиве  $u[1:n, 1:n]$ , а вектор-строка  $b'$  — в массиве  $b[1:n]$ . Автор алгоритма 92 не рекомендует задавать  $kount > 6$ . Решение уравнения, т. е. вектор-строка  $x'$  содержится в массиве  $x[1:n]$ . Матрица  $c[1:n, 1:n]$  служит для контроля точности. После выполнения процедуры *simult* в качестве первой строки матрицы  $c$  должен быть вектор  $b'$ ; вдоль главной ее диагонали должен стоять первый элемент  $b_1$  вектора  $b'$ , а остальные ее элементы должны быть равны нулю. Число *eps* (типа *real*) служит для проверки того, насколько действительный результат близок к теоретическому. Если взять  $b' = (1, 0, \dots, 0)$ , то данная процедура найдет матрицу  $w[1:n, 1:n]$ , являющуюся обращенной матрицей  $u$ . В нулевом цикле первой итерации процедура определяет в качестве строк матрицы  $w$  столбцы матрицы  $u$ . Для всех следующих итераций строки матрицы  $w$ , вычисленные в  $n$ -м цикле последней итерации, являются строками матрицы  $w$  нулевого цикла.

```
procedure simult(u,b,n,kount,eps)result: (w,x,c);  
  value eps,n,kount; real eps; integer n,kount;  
  array u,c,b,w,x;  
begin real bh,z; integer i,j,p;  
  for j:=1 step 1 until n do  
    for i:=1 step 1 until n do w[j,i]:=u[i,j];  
nol:  for j:=1 step 1 until n do  
    for i:=1 step 1 until n do c[i,j]:=0;  
    bh:=b[1];  
    for j:=1 step 1 until n do  
      begin  
        for i:=1 step 1 until n do  
          c[j,j]:=c[j,j] + w[j,i] × u[i,j];
```

```

z:=bh/c[j,j];
for i:=1 step 1 until n do w[j,i]:=z×w[j,i];
for i:=1 step 1 until n do
  if i≠j then
    begin
      for p:=1 step 1 until n do
        c[i,j]:=c[i,j]+u[p,j]×w[i,p];
      z:=(if i=1 then b[j] else 0)—c[i,j];
      for p:=1 step 1 until n do
        w[i,p]:=bh×w[i,p]+z×w[j,p]
      end i;
    end j;
  end j;
z:=abs(b[1]);
for j:=1 step 1 until n do
  if abs(abs(c[j,j])—z)>eps then
    begin kount:=kount—1;
      if kount≥0 then go to no1
    end;
  for j:=1 step 1 until n do x[j]:=w[1,j]
end simult;

```

#### Свидетельство к алгоритму 926

Алгоритм 926 получен в результате ряда существенных преобразований алгоритма 92а, имеющих целью сокращение его записи и расходуемой им памяти и ускорение его выполнения.

Алгоритм 926 был транслирован на машине М-220 в системе ТА-1М для решения системы уравнений

$$\begin{aligned}
 2x_1 + x_2 + 3x_3 &= 9, \\
 x_1 - 2x_2 + x_3 &= -2, \\
 3x_1 + 2x_2 + 2x_3 &= 7.
 \end{aligned}$$

В обращении к процедуре *simult* задавались параметры

$$u = \begin{vmatrix} 2 & 1 & 3 \\ 1 & -2 & 2 \\ 3 & 1 & 2 \end{vmatrix};$$

$b=(9,-2,7)$ ;  $n=3$ ;  $kount=1, 3, 6, 10, 50$  и  $eps=10^{-3}, 10^{-5}, 10^{-8}$ . Во всех вариантах были получены одни и те же правильные результаты  $x=(-1,2,3)$  и

$$c = \begin{vmatrix} 9 & -2 & 7 \\ -0.36379788_{10}-11 & 9 & 0.21827872_{10}-10 \\ 0 & -0.58207661_{10}-10 & 9 \end{vmatrix}.$$

При задании  $b=(1,0,0)$  для всех вышеуказанных вариантов остальных параметров была получена одна и та же матрица

$$w = \begin{vmatrix} -0.46153846 & 0.076923076 & 0.61538461 \\ 0.30769230 & -0.38461538 & -0.076923076 \\ 0.53846153 & 0.076923076 & -0.38461538 \end{vmatrix},$$

точно совпадающая с результатом обращения матрицы  $u$  с помощью процедуры  $r0037$  системы ТА-1М.

Далее такие же расчеты были проведены и для системы уравнений

$$\begin{aligned} 2x_1 + x_2 + 3x_3 &= 9, \\ x_1 - 2x_2 + 2x_3 &= -2, \\ 3x_1 + x_2 + 2x_3 &= 7, \end{aligned}$$

матрица которой является транспонированной матрицей от первой системы уравнений. Как и для первой системы уравнений, результаты не зависели от  $kount$  и  $eps$  и были  $x = (-0.42536367_{10} - 10, 3, 2)$  и

$$c = \begin{vmatrix} 9 & -2 & 7 \\ -0.14551915_{10} - 10 & 9 & 0.72759576_{10} - 11 \\ 0.29103830_{10} - 10 & 0 & 9 \end{vmatrix}.$$

Точное решение системы  $x = (0, 3, 2)$ .

При задании  $b = (1, 0, 0)$  была получена матрица

$$w = \begin{vmatrix} -0.46153846 & 0.30769230 & 0.53846153 \\ 0.076923076 & -0.38461538 & 0.076923076 \\ 0.61538461 & -0.076923076 & -0.38461538 \end{vmatrix},$$

точно совпавшая с результатом обращения матрицы  $u$  с помощью процедуры  $r0037$  системы ТА-1М.

### Свидетельство к алгоритму 92а

Алгоритм 92а получен в результате исправления, сокращения и ординарной переработки алгоритма 92 (R o e k D. J. «САСМ», 1962, № 5).

В алгоритме 92 были обнаружены следующие ошибки.

1. Процедура, вычисляющая абсолютное значение от своего параметра, имеет в языке АЛГОЛ стандартный идентификатор  $abs$ , следовательно, параметр  $absf$  не нужен.

2. После метки S3: должно следовать

**for**  $j := 1$  **step** 1 **until**  $n$  **do**  
if  $abs(abs(c[j,j]) - abs(b[1])) > eps$  **then**

3. После метки S6: должно быть

**for**  $j := 1$  **step** 1 **until**  $n$  **do**

Алгоритм проверен с помощью транслятора ТА-1 для системы линейных алгебраических уравнений третьего порядка...\*

### Свидетельство к алгоритму 93б [А1]

Алгоритм 93б «Обобщенные арифметические операции» не публикуется здесь, потому что с помощью соответствующих алгоритмов 93 (Perstein M. H. «САСМ», 1962, № 6) и 93а [23] авторы выпуска не получили удовлетворительного решения.

---

\* Результаты трансляции не приводятся здесь как потерявшие свое значение после публикации «Свидетельства к алгоритму 92б». (Прим. ред.)

## АЛГОРИТМ 946

### Генератор сочетаний [G6]

Процедура *combination* (*combination* — сочетание) образует следующее по порядку сочетание из  $n$  целых чисел по  $k$ , если заданы  $n$ ,  $k$  и предшествующее сочетание. В векторе  $j[1], \dots, j[k]$  целые числа меняются в пределах от 0 до  $n-1$  и всегда при входе и выходе из процедуры составляют монотонную строго возрастающую последовательность. Если входной вектор  $j$  состоит из нулей, то в качестве первого сочетания будет получено  $n-k, \dots, n-1$ . Это начальное сочетание получится также и после сочетания 0, 1, ...,  $k-1$ , являющегося последним значением вектора  $j$  в этом цикле.

```
procedure combination(n,k) data result: (j);  
  value n,k; integer n,k; integer array j;  
begin integer a,b,m;  
  for b:=1 step 1 until k do  
    if j[b] ≥ b then  
      begin a:=j[b]-b-1;  
        for m:=1 step 1 until b do j[m]:=m+a;  
        go to final  
      end;  
    b:=n-k+1;  
    for m:=1 step 1 until k do j[m]:=b+m;  
  final: end combination;
```

### Свидетельство к алгоритму 946

Алгоритм 946 получен из алгоритма 94а путем внесения в него поправки, предложенной А. Витеком в его «Замечании к алгоритму 94а» [25, с. 184], и замены оператора

```
  for m:=1 step 1 until k do j[m]:=n-k-1+m;  
на операторы  
  b:=n-k-1;  
  for m:=1 step 1 until k do j[m]:=b+m;
```

Алгоритм 946 был транслирован на машине М-220 в системе ТА-1М, и с ним успешно были повторены расчеты, указанные в ниже-следующем «Свидетельстве к алгоритму 94а».

### Свидетельство к алгоритму 94а

Алгоритм 94а получен в результате исправления, сокращения и переработки алгоритма 94 (Kurtzberg J. «САСМ», 1962, № 6).

Алгоритм проверен для  $n=4$  и  $k=3$ . В качестве начального вектора  $j$  был взят вектор (0,0,0). В результате четырех обращений к процедуре получены следующие сочетания: (1,2,3); (0,2,3); (0,1,3); (0,1,2).

### Подтверждение к алгоритму 94

Р. В. Мей (May R. W. «САСМ», 1962, № 11)

Алгоритм 94 был переведен на язык ФОРТРАН для машины IBM 1620 и прошел удовлетворительно без всяких исправлений. Однако переменная  $A$  не была описана.



## Подтверждение к алгоритму 94

Р. Е. Гренч (Grench R. E. «CASM», 1962, № 12)

В алгоритме потребовалось четыре изменения...\*

После вышеуказанных изменений тело алгоритма 94 было проверено на машине LGP 30, использующей транслятор Дартмутского колледжа с языка АЛГОЛ-30. Процедура работала удовлетворительно, а время, необходимое для получения одного вектора  $j$  при  $k=5$  и  $n=15$ , было равно 30с.

Если алгоритм предназначен для использования в виде процедуры, то нужно добавить различные тесты. Эти тесты могут включать оператор, проверяющий, правильно ли начальное значение вектора  $j$ , и не является ли  $k > n$ . Эти две возможности были исследованы, и обнаружено, что получаются неверные векторы  $j$ .

## Свидетельство к алгоритму 956 [A1]

Алгоритм 956 не публикуется здесь, потому что с помощью соответствующих алгоритмов 95 (Stockmал F. «CASM», 1962, № 6) и 95a [23] авторы выпусков не получили удовлетворительного решения на машине. Генерацию разбиений можно производить более совершенным алгоритмом 263 («CASM», 65—8).

## АЛГОРИТМ 966

### Матрица причинно-следственных отношений [H]

Процедура *ancestor* (*ancestor* — прародитель) работает следующим образом. Первоначально (т. е. во входной матрице  $m[1:n, 1:n]$ ) элемент  $m[i, j] \equiv \text{true}$ , если индивидуум  $i$  является родителем (непосредственной причиной) индивидуума  $j$ . После выполнения процедуры  $m[i, j] \equiv \text{true}$ , если индивидуум  $i$  является прародителем (косвенной причиной) индивидуума  $j$ , т. е. после выполнения процедуры имеет место эквивалентность  $m[i, j] \equiv \text{true}$  тогда, когда существует последовательность  $k, l, \dots, p$  такая, что во входной матрице все  $m[i, k], m[k, l], \dots, m[p, j]$  эквивалентны true.

```
procedure ancestor(n) dataresult:(m);  
  value n; integer n; Boolean array m;  
begin integer i, j, k;  
  for i:=1 step 1 until n do  
    for j:=1 step 1 until n do  
      if m[j, i] then  
        for k:=1 step 1 until n do  
          if m[i, k] then m[j, k]:=true  
end ancestor;
```

---

\* Далее указываются четыре поправки к алгоритму 94, учтенные в алгоритме 94a. (Прим. ред.)

### Свидетельство к алгоритму 96б

Алгоритм 96б является стереотипным переизданием алгоритма 96а, правильность которого была подтверждена также и расчетами Г. В. Пелледова [25, с. 174]. Автор алгоритма 96 ссылается на работу С. Уаршала [3i].

### Свидетельство к алгоритму 96а

Алгоритм 96а получен в результате ординарной переработки алгоритма 96 (Floyd R. W. «САСМ», 1962, № 6).

Путем трансляции подтверждена правильность результатов для первой и третьей матрицы нижеследующего «Подтверждения к алгоритму 96» («САСМ», 1963, № 3). Вторая матрица в этом подтверждении задана неверно, поскольку она должна быть квадратной.

### Подтверждение к алгоритму 96

Г. Тачер (Thacher H. C. «САСМ», 1963, № 3)

Тело этой процедуры было проверено на машине LGP 30 с использованием дартмутского транслятора. После заключения условных операторов в скобки **begin** и **end** (это необходимо для данного транслятора) процедура работала удовлетворительно для следующих матриц\*:  
 $n=5$ , время 8 мин 15 с

$$\left\| \begin{array}{ccccc} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right\| \rightarrow \left\| \begin{array}{ccccc} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right\| ;$$

$n=9$ , время 31 мин 2 с

$$\left\| \begin{array}{cccccccc} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array} \right\| \rightarrow \left\| \begin{array}{cccccccc} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array} \right\| .$$

Правильность этих результатов\*\* была подтверждена при исследовании сетевых диаграмм.

### АЛГОРИТМ 97б

#### Кратчайший путь [N]

Первоначально  $m[i, j]$  — это длина прямой связи от точки  $i$  сети до точки  $j$ . Если прямой связи между этими точками не существует, то

\* В переводе ради наглядности символы  $T$  и  $F$  заменены на 1 и 0 соответственно. (Прим. ред.)

\*\* Вторая матрица (для  $n=6$ ) здесь не приводится, поскольку в оригинале она задана неправильно. (Прим. ред.)

$m[i,j]$  первоначально равно  $10^{10}$ . После выполнения процедуры  $m[i,j]$  будет длиной кратчайшего пути от  $i$  до  $j$ . Если такового не существует, то  $m[i,j]=10^{10}$ . Массив  $m$  имеет размерность  $[1:n,1:n]$ . Более подробно см. в работе С. Уаршала [31].

```

procedure shortest path(n) dataresult:(m);
    value n; integer n; array m;
begin real inf,s; integer i,j,k;
    inf:= $10^{10}$ ;
    for i:=1 step 1 until n do
        for j:=1 step 1 until n do
            if m[j,i]<inf then
                for k:=1 step 1 until n do
                    if m[i,k]<inf then
                        begin s:=m[j,i]+m[i,k];
                            if s<m[j,k] then m[j,k]:=s
                        end
    end shortest path;

```

### Свидетельство к алгоритму 976

Алгоритм 976 является стереотипным переизданием алгоритма 97а. Алгоритм 976 был транслирован на машине М-220 в системе ТА-1М, и с его помощью была решена следующая простая задача: найти кратчайшие пути между узловыми точками сети, изображенной на рис. 1.

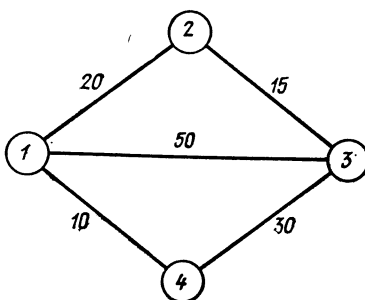


Рис. 1.

Задавались параметры  $n=4$  и

$$m = \begin{vmatrix} 0 & 20 & 50 & 10 \\ 20 & 0 & 15 & 10^{10} \\ 50 & 15 & 0 & 30 \\ 10 & 10^{10} & 30 & 0 \end{vmatrix}.$$

Были получены следующие правильные результаты:

$$m = \begin{vmatrix} 0 & 20 & 35 & 10 \\ 20 & 0 & 15 & 30 \\ 35 & 15 & 0 & 30 \\ 10 & 30 & 30 & 0 \end{vmatrix}$$

Другой алгоритм определения кратчайшего пути описан в работе Алексеевой С. М. и Алексеева О. Г. [43].

### Свидетельство к алгоритму 97а

Процедура *shortest path* алгоритма 97а, по существу, не отличается от процедуры алгоритма 97 (Floyd R. W. «CACM», 1962, № 6).

## АЛГОРИТМ 986

### Комплексный криволинейный интеграл [D1]

Процедура *complint* приближенно вычисляет комплексный криволинейный интеграл с помощью конечной суммы Римана — Стильеса

$$\sum_{t=1}^n f(z_k) [z_t - z_{t-1}], \text{ где } a \leq t \leq b \text{ и } z_k \in (z_{t-1}, z_t).$$

Программист должен задать:

- 1) процедуру *gamma(t,z)* для вычисления  $z(t)$  на кривой  $\Gamma$  и процедуру *func(z,f)* для вычисления значений функции;
- 2) концевые точки  $a$  и  $b$  интервала изменения параметра и  $n$  — число подынтервалов, на которые должен быть разбит интервал  $[a,b]$ .

```

procedure complint(a,b,n)result: (sum1,sum2);
  value a,b,n; real a,b,n,sum1,sum2;
begin real t,d,ztl1,ztl2,delz1,delz2,part1,part2;
  integer i: array zt,zk,fz[1:2];
  sum1:=sum2:=0;
  d:=(b-a)/n; t:=a;
line:  gamma(t,zt);
  if t=a then go to next;
  delz1:=zt[1]-ztl1; delz2:=zt[2]-ztl2;
  zk[1]:=ztl1+delz1/2; zk[2]:=ztl2+delz2/2;
  func(zk,fz);
  part1:=fz[1]×delz1-fz[2]×delz2;
  part2:=fz[1]×delz2+fz[2]×delz1;
  sum1:=sum1+part1;
  sum2:=sum2+part2;
  if t≥b-0.25×d then go to final;
next: ztl1:=zt[1]; ztl2:=zt[2];
  t:=t+d;
  go to line;
final: end complint;
```

### Свидетельство к алгоритму 986

Алгоритм 986 является стереотипным переизданием алгоритма 98а.

Алгоритм 986 был транслирован на машине БЭСМ-6 в системе БЭСМ-АЛГОЛ при  $n=1, 10, 25, 50, 100, 250, 500, 1000, 2000$  для следующих задач из сборника Волковыского Л. И. и др. [35].

1. Задача 388 (1). Вычисление  $I_1 = \int x dz$  и  $I_2 = \int y dz$  по радиусу-вектору точки  $z=2+i$ .

Для решения этой задачи задавались параметры  $a=0$ ,  $b=1$  и  
**procedure** гамма(t,z);

**value** t; **real** t; **array** z;  
**begin** z[1]:=2×t; z[2]:=t **end**  
    для вычисления  $I_1$  задавалась  
**procedure** func(z,f);

**array** z,f;  
**begin** f[1]:=z[1]; f[2]:=0 **end**

а для вычисления  $I_2$  тело процедуры *func* имело вид

**begin** f[1]:=z[2]; f[2]:=0 **end**

Уже при  $n=1$  были получены точные результаты  $I_1=2+i$  и  $I_2=1+i/2$ .

2. **Задача 388 (2)**. Вычисление  $I_1 = \int x dz$  по полуокружности  $|z|=1$ ,  $y \geq 0$  (начало пути в точке  $z=1$ ).

Задавались параметры  $a=0$ ,  $b=3.141592654$ , тело процедуры *gamma*

**begin** z[1]:=cos(t); z[2]:=sin(t) **end**

и процедура *func* такая же, как и в задаче 1. Результаты приведены в табл. 22.

Таблица 22

<i>n</i>	<i>sum2</i>	<i>n</i>	<i>sum2</i>	<i>n</i>	<i>sum2</i>
10	1.5450849	50	1.5697629	250	1.5707549
25	1.5666654	100	1.5705379	500	1.5707859

Значения *sum1* были в пределах от  $0.9_{10}-12$  до  $0.23_{10}-10$ ; точное значение  $I_1=i \times \pi/2=i \times 1.57079633$ ; время вычисления интеграла для  $n=250$  было 11с и для  $n=500-19$  с.

3. **Задача 393(1)**. Вычисление  $\int dz/\sqrt{z} = -2+i2$  по полуокружности  $|z|=1$ ,  $y \geq 0$  и  $\sqrt{1}=1$ .

Задавались параметры  $a=0$ ,  $b=3.14159265$ , процедура *gamma*, как в задаче 2, и тело процедуры *func*

**begin** f[1]:=sqrt(0.5×(1+z[1])); f[2]:=-sqrt(0.5×(1-z[1])) **end**  
Выражения для f[1] и f[2] следуют из соотношений

$$\frac{1}{\sqrt{z}} = \frac{\sqrt{z_1 - iz_2}}{\sqrt{z_1^2 + z_2^2}} = \sqrt{z_1 - iz_2} = \sqrt{\cos t - i \sin t} = \cos\left(\frac{t}{2}\right) - i \sin\left(\frac{t}{2}\right) = \sqrt{\frac{1+z_1}{2}} - i \sqrt{\frac{1-z_1}{2}}.$$

Результаты приведены в табл. 23.

Таблица 23

<i>n</i>	<i>sum2</i>	<i>n</i>	<i>sum2</i>	<i>n</i>	<i>sum2</i>
10	1.9802187	50	1.9988246	250	1.9999372
25	1.9959690	100	1.9996638	500	1.9999825

Значения *sum1* отличались от значений *sum2* только знаком; время вычисления интеграла для  $n=250$  было 10с, а для  $n=500$ —20с.

4. *Задача 392 (1)*. Вычисление  $\int z^2 dz = -2/3$  по полуокружности  $|z|=1$ ,  $0 \leq \arg(z) \leq \pi$ , начало пути в точке  $z=1$ .

Задавались параметры  $a=0$ ,  $b=3.14159265$ , процедура *gamma*, как в задаче 2, тело процедуры *func*

**begin**  $f[1]:=1-2 \times z[2]^{\uparrow 2}$ ;  $f[2]:=2 \times z[1] \times z[2]$  **end**

Результаты приведены в табл. 24.

Т а б л и ц а 24

<i>n</i>	<i>sum1</i>	<i>n</i>	<i>sum1</i>	<i>n</i>	<i>sum1</i>
10	—0.72123174	50	—0.66885947	250	—0.66675439
25	—0.67543273	100	—0.66721495	500	—0.66668859

Значения *sum2* были в пределах от  $-0.13_{10}-8$  до  $-0.39_{10}-8$ ; время вычисления для  $n=250$  было 11 с и для  $n=500$ —21с.

5. *Задача 387 (3)*. Вычисление  $\int \bar{z} dz = i2\pi$  по окружности радиуса  $r=1$  с центром в начале координат.

Задавались параметры  $a=0$ ,  $b=2\pi=6.2831853$ , процедура *gamma*, как в задаче 2, тело процедуры *func*

**begin**  $f[1]:=z[1]$ ;  $f[2]:=-z[2]$  **end**

Результаты приведены в табл. 25.

Т а б л и ц а 25

<i>n</i>	<i>sum2</i>	<i>n</i>	<i>sum2</i>	<i>n</i>	<i>sum2</i>
50	6.2666616	250	6.2825238	1000	6.2831439
100	6.2790519	500	6.2830199	2000	6.2831749

Значения *sum1* были в пределах от  $0.4_{10}-11$  до  $0.2_{10}-10$ ; время вычисления интеграла для  $n=250$  было 11 с и для  $n=500$ —2 с.

Комплексные интегралы описываются, например, в работе М. А. Лаврентьева и Б. В. Шабата [34, с. 45].

### Свидетельство к алгоритму 98а

Алгоритм 98а получен в результате исправления, сокращения, оптимизации и ординарной переработки алгоритма 98 (Pfaltz J. L. «САСМ», 1962, № 6).

Исправление заключалось в локализации массива *fz* (в алгоритме 98 ему соответствует массив FZ). Оптимизация состояла в замене массивов RSSUM, ZTL, DELZ, PART [1:2] простыми переменными *sum1*, *sum2*, *ztl1*, *ztl2*, *delz1*, *delz2*, *part1* и *part2* для ускорения работы алгоритма.

## АЛГОРИТМ 996

### Вычисление символа Якоби [A1]

Процедура *symjac* вычисляет значение символа Якоби  $\left(\frac{n}{m}\right)$  по закону квадратичной взаимности. Если  $m$  — нечетное, то параметру  $r$  присваивается одно из трех значений:  $-1, 0, 1$ . Если  $m$  — четное, то символ Якоби не определен и параметру  $r$  присваивается значение 2. Для нечетных  $m$  программа предусматривает проверку, являются ли  $m$  и  $n$  взаимно простыми. Значение  $r=0$  в том и только в том случае, если  $m$  и  $n$  имеют нетривиальный общий множитель. В специальном случае, когда  $m$  простое,  $r=-1$ , если только  $n$  является квадратичным невычетом числа  $m$ .

```
procedure symjac(n,m)result:(r);
  value n,m; integer n,m,r;
begin integer s; Boolean p,q;
  Boolean procedure parity(x);
    value x; integer x;
    comment Значение функции parity будет true, если
    x нечетное, и false, если x — четное;
    parity:=x÷2×2≠x;
start: if ¬ parity(m) then
  begin r:=2; go to final end;
  p:= true;
cycle:n:=n—n÷m×m; q:= false;
  if n≤1 then go to done;
even: if ¬ parity(n) then
  begin q:= ¬ q; n:=n÷2;
    go to even
  end теперь n нечетное;
  if q ∧ parity((m↑2—1)÷8) then p:= ¬ p;
  if n=1 then go to done;
  if parity((m—1)×(n—1)÷4) then p:= ¬ p;
  s:=m; m:=n; n:=s;
  go to cycle;
done: r:= if n=0 then 0 else
  if p then 1 else —1;
final: end symjac;
```

### Свидетельство к алгоритму 996

Алгоритм 996 является стереотипным переизданием алгоритма 99а.

### Свидетельство к алгоритму 99а

Алгоритм 99а получен в результате исправления и ординарной переработки алгоритма 99 (Garland S. J., Knapp A. W. «САСМ», 1962, № 6).

Исправлена одна синтаксическая ошибка, указанная в «Замечании к алгоритму 99» (May R. W. «САСМ», 1962, № 11).

В результате трансляции алгоритма 99а получено:

$$\left(\frac{5}{19}\right) = 1 \quad (n=5 \text{ — квадратичный вычет}),$$

$$\left(\frac{3}{31}\right) = -1 \quad (n=3 \text{ — квадратичный невычет}),$$

$$\left(\frac{7}{12}\right) = 2 \quad (m=12 \text{ — четное}).$$

Символ Якоби описывается, например, в работе И. М. Виноградова [6, с. 78—81].

### **Свидетельство к алгоритму 100б [O2]**

Алгоритм 100б «Заполнение информацией матрицы связи» не публикуется здесь, потому что с помощью соответствующих алгоритмов 100 (K i v i a t P. J. «САСМ», 1962, № 6) и 100а [23] авторы выпуска не получили удовлетворительных результатов на машине.



## АЛГОРИТМ 50СJ

### Как программировать игру в шахматы [Z]

#### Введение \*

Миллионы людей интересуются шахматами. Большинство из них проявляет интерес и к реализации шахматной игры на машине. И этот интерес далеко не только спортивный. Исследователи проблем современной кибернетики видят (и не без оснований!) в программировании шахматной игры один из возможных путей раскрытия законов человеческого мышления и создания «мыслящих машин».

Привлечение широких кругов программистов к участию в решении этих проблем, в разработке шахматных программ могло бы не только значительно ускорить эти работы, но и содействовало бы распространению многих идей и технических приемов, накопленных в процессе таких разработок. К сожалению, в настоящее время создание шахматных программ ведется в большинстве случаев отдельными разрозненными группами, нередко дублирующими одна другую. Поскольку публикация шахматных программ, составленных в машинном коде или в автокоде, дело отнюдь не легкое и вряд ли рентабельное, то эти программы превратились фактически в частную собственность их разработчиков. Большинство других людей, интересующихся этими проблемами, не может не только принять участие в их разработке, но даже воспользоваться уже созданными программами для того, чтобы один на один сыграть с машиной в шахматы.

Использование для публикации шахматных программ универсальных алгоритмических языков сделало возможным для широкого круга программистов восприятие основных идей шахматного программирования и дало им возможность принять участие в разработке шахматных программ. Публикуемый здесь алгоритм 50СJ выполняет в первую очередь именно эти функции.

Разумеется, шахматная программа на алгоритмическом языке не может соревноваться по быстрдействию с шахматной программой в машинном коде или автокоде. Но она и не предназначена для этого. Ознакомившись с основными идеями этой программы, пользователь может сам усовершенствовать ее, и если наступит момент, когда его АЛГОЛ-программа станет потреблять слишком много машинного времени, то и сможет без особого труда перевести ее на один из низших языков программирования. При этом в его распоряжении будет не только готовая отлаженная АЛГОЛ-программа, но и готовый набор идентификаторов, обозначений и специальных терминов. К тому же приведенная здесь АЛГОЛ-программа для решения двухходовок требует так мало машинного времени (см. ниже «Свидетельство к алгоритму 50СJ»), что вряд ли на этом этапе имеет смысл стремиться к его экономии.

---

\* Введение к алгоритму 50СJ написано Агеевым М. И. Индекс СJ у номера алгоритма указывает на источник, в котором был опубликован исходный вариант этого алгоритма (журнал «The Computer Journal»). Алгоритм 50СJ публикуется здесь на правах обычного подтверждения исходного варианта [40]. (Прим. ред.)

Кроме алгоритма 50 А. Белла [40], послужившего прототипом алгоритма 50СJ, в том же журнале «The computer journal» позднее был опубликован и более общий алгоритм 68 Дж. Манинга «Белые начинают и дают мат в  $n$  ходов» [42]. Однако этот общий алгоритм основан на использовании рекурсивных процедур, а следовательно, имеет более узкую область применимости и, по признанию самого его автора, работает весьма медленно. Кроме того, алгоритм А. Белла имеет гораздо более обширную описательную часть (см. ниже перевод пояснительного текста к алгоритму 50) и легко может быть обобщен для решения задач на большее число ходов. Таким образом, публикация на русском языке переработанного варианта алгоритма 50 представляется первоочередной.

## Пояснительный текст к алгоритму 50

А. Белл (Bell A. G. «The Computer Journal», 1970, № 5) \*

Большинство программ, написанных для игры в шахматы [41], содержат ограничения. В данной работе излагается метод воспроизведения всех ходов в любой шахматной позиции, включая превращение пешки в фигуру, рокировку и взятие пешек на проходе. Для проверки и демонстрации изложенных методов написана АЛГОЛ-программа решения любой шахматной задачи на мат в два хода. Полностью эта программа и требующиеся для работы с нею таблицы вместе с некоторыми факультативными процедурами приводятся в дополнениях 1—4.

### Таблицы

Наиболее существенной особенностью данной программы является то, что для работы с нею требуется шесть таблиц (массивов). Чтобы программа была короче, лучше всего, если эти таблицы будут вводиться, а не вычисляться. Потому они полностью приводятся в дополнении 1. Кроме того, так будет проще понять и проконтролировать их работу.

Эти шесть таблиц подразделяются на три группы; таблицы коня и короля, таблицы слона и ладьи, таблицы белой и черной пешек.

### Таблицы коня и короля

Таблица коня (дополнение 1.1a) состоит из 64 строк [каждая строка соответствует одному полю (одной клетке) шахматной доски, как показано на рис. 2], числа

Черные							
57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8
Белые							

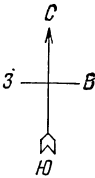


Рис. 2. Нумерация полей шахматной доски.

в самом левом столбце таблицы являются номерами строк и не входят в состав таблицы.

Рассмотрим строку 1. Она используется тогда, когда конь находится на поле 1. Последнее число в строке — это общее число полей, на которые может пойти данный

\* См. [40]. Перевод и отладка алгоритма выполнены Агеевым М. И.

конь (в рассматриваемом случае только два поля), а предыдущие два числа — номера этих полей (18 и 11). Нули в таблицах не используются.

Таблица ходов короля (дополнение 1.1б) построена точно так же. Таким образом, процедуре ХОДЫ КОНЯ И КОРОЛЯ для определения полей, на которые может ходить одна из этих фигур, требуется только номер поля, занятого конем или королем, вместе с соответствующей таблицей. Генерируемые (т. е. получаемые в процессе выполнения процедуры) ходы последовательно запоминаются в массиве с идентификатором СПИСОК. Следует заметить, что перед выбором хода делаются две пробы (эти пробы применяются ко всем фигурам при обзоре их ходов). Первая проба обусловлена тем, что никакая фигура не может ходить на поле, занятое фигурой того же цвета, вторая — тем, что, если фигура может пойти на поле, занятое королем противника, то обзор дальнейших ходов заканчивается (путем выхода на метку ПРЕРЫВАНИЕ). Так проверяется правильность предыдущего хода противника. Рокировка рассматривается в разделе «Рокировка».

#### *Таблицы ладьи и слона*

Таблица ладьи (дополнение 1.2а) служит для генерации ходов ладьи. Первая строка — это приращения, необходимые для передвижения ладьи с ее исходного поля в каждом из четырех возможных направлений, т. е. на восток (+1), запад (—1), север (+8) и юг (—8). Снова, как и в предыдущей таблице, рассмотрим строку 1, которая используется тогда, когда ладья находится на поле 1. Первое число (т. е. 8) — это последнее поле доски (см. рис. 2), если ладья передвигается на восток, второе число (1), если на запад, третье (57), если на север, и четвертое (1), если на юг. Попытки передвижения на запад и на юг с поля 1 будут исключены.

Максимальное число ходов, которое может сделать ладья, равно 14. Каждый из этих ходов может быть сделан только тогда, когда горизонталь и вертикаль этой ладьи свободны, или когда только последние поля заняты фигурами противника. Однако, как и в процедуре ХОДЫ КОНЯ И КОРОЛЯ, каждое генерированное поле должно быть проверено на присутствие одноцветной фигуры. Если такое присутствие обнаруживается, то ход на это поле исключается и выбирается новое приращение (т. е. новое направление) с помощью оператора.

**if СВОЯ ДОСКА [i]  $\neq$  0 then go to НОВОЕ НАПРАВЛЕНИЕ;**

Если же поле не занято своей фигурой, то данный ход добавляется к списку ходов (к массиву СПИСОК) и делается новая проверка (не занято ли поле фигурой противника) с помощью условия

**if ЧУЖАЯ ДОСКА [i]  $\neq$  0 then**

Если не занято, то в рассматриваемом направлении генерируется следующее поле. В противном случае либо выход из процедуры, если поле занято королем противника (подобно тому, как это делалось в процедуре ХОДЫ КОНЯ И КОРОЛЯ), либо выбор другого направления. Когда процедура исчерпает все поля по всем четырем направлениям, она заканчивается.

Таблица слона (дополнение 1.2б) используется точно так же. Четырьмя направлениями в этом случае являются северо-восток (+9), юго-запад (—9), северо-запад (+7) и юго-восток (—7).

#### *Таблицы белой и черной пешек*

В отличие от фигур цвет для пешек существен. В данной программе белые пешки передвигаются по доске вверх, а черные — вниз, отсюда две таблицы. Генерация ходов для пешки наиболее сложная, поскольку пешка имеет пять различных ходов.

Рассмотрим ходы белой пешки:

- 1) с ее начальной позиции она может продвинувшись вверх по доске на одно или два свободных поля;
- 2) после этого она продвигается за один ход на одно свободное поле;
- 3) берет фигуру противника на смежном северо-восточном или северо-западном поле;
- 4) превращается в ферзя, ладью, слона или коня при достижении восьмой горизонтали;
- 5) берется на проходе.

Черная пешка отличается от белой лишь тем, что она должна всегда передвигаться вниз по доске.

Таблицы отражают только ходы 1, 2 и 3. Превращение пешки в фигуру рассматривается в самой программе (см. раздел «Выполнение ходов»), а взятие пешки на проходе — в одноименном разделе.

Таблица белой пешки (дополнение 1.3а) используется для генерации возможных ходов белой пешки. Пешки могут находиться только на полях от 9 до 56-го, и строки таблицы пронумерованы соответственно этому. Первый столбец — это продвижение пешки вперед на одно поле, и если это поле свободно, ход фиксируется. Тогда и только тогда программа попытается продвинуть пешку через одно поле на второе (номер этого поля находится во втором столбце). Если это поле тоже свободно, то данный ход тоже фиксируется. Заметим, что как только пешка сделала ход, то двойной ее ход, определяемый таблицей, всегда запрещается, потому что поле, очевидно, занято (самой этой пешкой!).

Третий и четвертый столбцы определяют поля, на которые может быть сделан ход, если они заняты фигурой противника (т. е. путем взятия). Поле 65 никогда не бывает занято фигурой противника (безфигурная область) и предназначено для интерпретации краевых эффектов.

Таблица черной пешки (дополнение 1.3б) используется точно так же. Снова процедура ХОДЫ ПЕШЕК требует только задания поля, занятого пешкой, и таблицы, соответствующей цвету пешки. Делается обычная проверка на взятие короля противника.

### *Обзор ходов, возможных в данной позиции*

Процедура ОБЗОР ХОДОВ использует три процедуры, описанные в предыдущем разделе, и работает для любой фигуры, как белой, так и черной. Однако отдельные фигуры должны между собой различаться, поэтому каждой фигуре ставится в соответствие целое число. Так король (Кр) — это 6, ферзь (Ф) — 5, ладья (Л) — 4, слон (С) — 3, конь (К) — 2 и пешка (П) — 1.

Рассмотрим рис. 3,а, где изображена позиция \*, в которой белые дают мат в два хода. Рисунки 3,б и в являются требующимися для процедуры представлениями этой позиции, записанными в массивах ДОСКА БЕЛЫХ и ДОСКА ЧЕРНЫХ.

Процедура ОБЗОР ХОДОВ имеет пять параметров.

1. СВОИ ПОЛЯ — целый массив размерности [0:16], указывающий

а) сколько полей занято на доске белыми или черными фигурами (в СВОИ ПОЛЯ [0]),

б) какие поля они занимают (рис. 3,г и д).

2. СВОЯ ДОСКА — представление доски, содержащей фигуры той стороны, ходы которой мы хотим зафиксировать (рис. 3,б).

3. ЧУЖАЯ ДОСКА — представление доски, содержащей фигуры противника

---

\* Ради стремления, как можно меньше отклоняться от оригинала, в изображениях фигур на шахматной доске сохранены обозначения А. Белла для белых (буквой В) и черных (буквой Ч) фигур. Однако более наглядно было бы различать их знаком + и — соответственно (Прим. ред.)

(рис. 3, в). Этот параметр используется для определения взятия фигур и, следовательно, должен быть отделен от параметра СВОЯ ДОСКА.

4. РУБЕЖ — значение индекса переменных в массиве СПИСОК, начиная с которого записываются генерированные ходы.
5. ПРЕРЫВАНИЕ — метка, используемая тогда, когда может быть взят король противника, т. е. когда его последний ход был незаконный.

БС				БК	
		БС			
				ЧП	
БЛ		ЧК	ЧФ	ЧКр	БЛ
		ЧП	ЧП		БЛ
	БК	ЧП	БЛ		
	БКр	БЛ	БФ		

а

3				2	
		3			
4					1
					1
	2			1	
	6	1	5		

б

				1	
		2	5	6	
		1		1	
		1			

в

11	10	12	14	18	22	32	33	39	51	57	62
----	----	----	----	----	----	----	----	----	----	----	----

г

7	20	28	30	36	37	38	46
---	----	----	----	----	----	----	----

д

Рис. 3. Представление белых и черных фигур на доске:

а — пример реальной позиции; б — ДОСКА БЕЛЫХ; в — ДОСКА ЧЕРНЫХ; г — число и позиции 11 белых фигур (ПОЛЯ БЕЛЫХ); д — число и позиции 7 черных фигур (ПОЛЯ ЧЕРНЫХ).

При обращении к процедуре ОБЗОР ХОДОВ генерируемая информация последовательно запоминается в массиве СПИСОК, начиная с элемента с индексом РУБЕЖ. Для примера, приведенного на рисунках 3, а—д, цикл устанавливается на рассмотрение 11 фигур

for p:=СВОИ ПОЛЯ [0] step —1 until 1 do

и переменная ПОЛЕ последовательно получает значение номера поля, которое занимает каждая из белых фигур. Если теперь это поле оказывается свободным, то фигура была взята. Если же на рассматриваемом поле имеется фигура, то перед генерацией ее ходов она сначала регистрируется путем запоминания в массиве СПИСОК следующих трех элементов информации:

- 1) p — указатель \* местонахождения в массиве СВОИ ПОЛЯ номера поля данной фигуры;
- 2) СВОЯ ДОСКА [ПОЛЕ] — числовое значение данной фигуры (пешка—1, конь—2 и т. д.);
- 3) ПОЛЕ — положение данной фигуры на доске.

Эта информация используется позже, когда ход делается фактически.

\* p — начальная буква слова pointer — указатель. (Прим. ред.)

Следующий оператор в процедуре ОБЗОР ХОДОВ является критическим. Идентификатор К ХОДАМ описан как название переключателя, и результатом оператора

**go to К ХОДАМ [СВОЯ ДОСКА [ПОЛЕ]]**

является обращение к соответствующей процедуре с соответствующим параметром. Заметим, что для решения вопроса о том, какую таблицу использовать для пешки, определяется цвет пешки, а ходы ферзя получаются в результате генерации сначала ходов ладьи, а затем ходов слона.

В данном примере сначала рассматривается одиннадцатая фигура (при такой записи цикл работает быстрее). Она занимает в массиве ДОСКА БЕЛЫХ поле 62, содержащее число 2, т. е. коня. Поэтому выполняются операторы

СПИСОК [с]:=11, СПИСОК [с+1]:=2 и СПИСОК [с+2]:=62.

Затем в процедуре делается переход к метке КОНЕМ, выполняется обращение к процедуре ХОДЫ КОНЯ И КОРОЛЯ, и генерируются ходы для коня, занимающего поле 62, т. е. последовательность 56, 47, 45 и 52. Далее делается переход к метке ДАЛЕЕ, где проверяется, был ли генерирован хотя бы какой-нибудь ход (если фигура не может сделать ни одного хода, то информация о ней затирается информацией о новой фигуре). Если ходы были генерированы, то процедура записывает в массив СПИСОК значение —ПОЛЕ (т. е. в данном примере —62), чтобы отметить окончание перечня ходов данной фигуры, и затем переходит к рассмотрению новой фигуры в массиве СВОИ ПОЛЯ. Так продолжается до исчерпания этого массива. Получающийся в результате список чисел в массиве СПИСОК приведен в табл. 26. Заметим, что две пешки ( $p=5$  и  $p=2$ ) не имеют ходов и поэтому в таблице пропущены.

Таблица 26

Название фигуры	Содержание массива СПИСОК				
	<i>p</i>	Фигура	Исходное поле	Поля возможных ходов	Признаки конца
Конь	11	2	62	56, 47, 45, 52	—62
Слон	10	3	57	50, 43, 36	—57
Слон	9	3	51	60, 42, 58, 44, 37	—51
Пешка	8	1	39	47, 46	—39
Ладья	7	4	33	34, 35, 36, 41, 49, 25, 17, 9, 1	—33
Пешка	6	1	32	40	—32
Конь	4	2	18	35, 28, 3, 1	—18
Ферзь	3	5	14	15, 16, 13, 6, 23, 5, 21, 28, 7	—14
Король	1	6	10	1, 2, 3, 9, 11, 17, 19	—10

После генерации ходов всех фигур индекс последней записи (в массиве СПИСОК — *Прим. ред.*) запоминается в переменной РУБЕЖ. Это значение используется тогда, когда ходы выполняются фактически. Незаконность некоторых ходов (король не может ходить на поля 11 или 19) обнаружится после того, как ходы будут фактически сделаны и окажется, что противник может взять короля (см. следующий раздел).

### Выполнение ходов

Процедура ВЫПОЛНЕНИЕ ХОДА оперирует со списком, генерируемым процедурой ОБЗОР ХОДОВ, и выполняет следующие функции.

1. Изменение содержимого четырех массивов: ДОСКА БЕЛЫХ, ДОСКА ЧЕРНЫХ, ПОЛЯ БЕЛЫХ и ПОЛЯ ЧЕРНЫХ в зависимости от того, какая из сторон должна ходить (если переменная УРОВЕНЬ нечетная, то ходят белые, в противном случае — черные). Для этого необходимо знать следующее:

а) поля, с которых и на которые может пойти фигура (эта информация отыскивается в массиве СПИСОК по значению индекса  $p$ ; в данном примере для первого хода конем  $p=3$ , поэтому  $OT=62$  и  $HA=56$ );

б) номер позиции рассматриваемой фигуры в массиве ПОЛЯ СВОИХ (в данном примере конь находится в позиции 11); эта информация хранится в переменной НОМЕР [УРОВЕНЬ];

в) начальное значение фигуры на случай превращения пешки в фигуру; эта информация запоминается в  $q$  [УРОВЕНЬ] \*. Для выполнения хода требуются шесть операций, обозначенных в программе метками от  $aa1$  до  $aa6$ .

$aa1$ : Значение данной фигуры (из  $q$  [УРОВЕНЬ]) присваивается переменной ДОСКА СВОИХ [НА] (кроме случая превращения пешки в фигуру, когда вместо пешки присваивается значение ферзя, ладьи, слона или коня соответственно).

$aa2$ : Поле ДОСКА СВОИХ [ОТ] очищается.

$aa3$ : Любая фигура, возможно взятая при опробовании предыдущего хода данной фигуры (когда проверялся ход на поле ОТ), возвращается на доску противника.

$aa4$ : Любая фигура, возможно взятая новым ходом данной фигуры, записывается в  $c$  [УРОВЕНЬ] \*\*. Чаще всего туда записывается нуль, но это не важно.

$aa5$ : Поле НА доски противника очищается от любой фигуры, которая возможно на нем находилась.

$aa6$ : Список позиций фигур на доске корректируется. В данном примере конь в позиции 11 занимал поле 62, теперь это поле заменяется на 56; тем самым подготавливается возможность генерации списка белых фигур на втором ходе.

2. После выполнения хода УРОВЕНЬ увеличивается и для нового расположения фигур на доске генерируются ответные ходы противника. При этом проверяется, является ли сделанный ход законным? Если нет, то из процедуры ОБЗОР ХОДОВ осуществляется выход к метке НЕЗАКОННЫЙ ХОД, что является, по существу, отклонением только что сделанного хода.

3. Если ход законный, то все готово к выполнению списка ответных ходов противника. Запоминается номер позиции \*\*\* первой фигуры противника и ее значение \*\*\*\*, и процедура заканчивается (назначение переменной ОБРАТНО К [УРОВЕНЬ], в которой запоминается исходное поле фигуры, только что сделавшей ход, разъясняется в следующем разделе).

Назначение процедуры ВЫПОЛНЕНИЕ ХОДА, оперирующей с массивом СПИСОК, состоит в том, чтобы просмотреть \*\*\*\*\* все поля, на которые может пойти данная фигура, пока не встретится поле с отрицательным значением НА. Это будет означать, что ходы данной фигуры кончились. Поэтому фигура и доска восстанавливаются в исходное состояние (пять операций с метками  $bb1$ — $bb5$ ; нам известно, что в данном случае ДОСКА ЧУЖИХ [—НА] имеет нулевое значение, в остальном эти операции идентичны операциям  $aa1$ — $aa6$ ). Подготавливаются ходы следующей фигуры, и делается выход из процедуры через метку ПРОДОЛЖЕНИЕ, за которой проверяется, не исчерпан ли список ходов, и обычно сразу же вызывается процедура ВЫПОЛНЕНИЕ ХОДА и начинает свои ходы новая фигура.

### *Решение задачи на мат в два хода*

Для проверки правильности и быстродействия процедур ОБЗОР ХОДОВ и ВЫПОЛНЕНИЕ ХОДА они используются здесь в законченной программе, решающей

---

\*  $q$  — начальная буква слова *queening* — превращение пешки в фигуру. (Прим. ред.)

\*\*  $c$  — начальная буква слова *capture* — взятие. (Прим. ред.)

\*\*\* В переменной НОМЕР [УРОВЕНЬ]. (Прим. ред.)

\*\*\*\* В переменной  $q$  [УРОВЕНЬ]. (Прим. ред.)

\*\*\*\*\* В результате последовательных к ней обращений. (Прим. ред.)

любую задачу на мат в два хода. Программа решает задачу так же как и человек, хотя и недалекий. Она делает один за другим начальные ходы за белых, пока не найдет такой начальный ход, который независимо от двух последующих ходов черных дает белым возможность объявить своим вторым ходом шах (во избежание пата) и всегда брать на третьем ходе короля черных.

Для ускорения работы программы необходимо иногда брать ход назад. Например, белые делают первый ход (Б1), а черные — ответный (Ч1) такой, что белые не могут шаховать черного короля на втором ходу. Тогда прежде чем пытаться сделать другой

				4С	4Л	4Кр	11	6	7	8	13	15	16	23	24	31	45	47
				4П		4П												
				5П	4П	6П												
				4П	4П	4П												
				4П		6П												
				4П		6П	6С											
				6П		6П	6Л											
				6С	6Л	6Кр	11	21	29	37	38	39	46	53	55	62	63	64

$\alpha$

$\beta$

Рис. 4 Позиция форсированного мата в два хода. Белые могут сделать только один ход, черные имеют тоже только один ответ, а белые еще только один ход:  
а — позиция Белла — простейшая из известных задач на мат в два хода;  
б — массив ПОЛЯ БЕЛЫХ; в — массив ПОЛЯ ЧЕРНЫХ.

ход белыми нужно взять назад ответный ход черных (Ч1). Это осуществляется процедурой ВОЗВРАТ ХОДА при условии задания поля, занимаемого черной фигурой в данный момент, и ее исходного поля. Исходное поле любой фигуры перед выполнением ее ходов всегда запоминается в переменной ОБРАТНО К [УРОВЕНЬ].

Для иллюстрации процесса решения задачи рассмотрим позицию, приведенную на рис. 4, а.

В массивах ПОЛЯ БЕЛЫХ и ПОЛЯ ЧЕРНЫХ (рис. 4, б и в) содержится 11 белых и 11 черных фигур соответственно.

Когда ходы черных на уровне 2 исчерпаны, то решение получено и позиция, возникшая после первого правильного хода, может быть выведена на печать.

Блок-схема программы приведена на рис. 5, а фактическое содержание массива СПИСОК, отображающее ходы, генерированные на пяти уровнях, в табл. 27.

Таблица 27

Ход	УРОВЕНЬ	Номер фигуры	Значение	Исходное поле	НА	Далее
Б1	1	9	1 П	31	38 П	—31
Ч1	2	11	6 Кр	64 —	56 (незаконный)	—64, 5, 1, 39, 31, —39 П
Б2	3	8	3 С	24	31 П	—24
Ч2	4	11	6 Кр	64 —	56	—64
	5	11	1 П	47	56 Кр	Перечисление ходов закончено



На машине АТЛАС данная программа решила задачу на мат в два хода за 45 с. После переписывания программы в безик-код \* это время сократилось до 5 с. Программа может также проверять единственность решения.

### Исключения

Приведенная здесь программа не даст решения в двух случаях, когда возможны:  
1) рокировка или 2) взятие пешки на проходе. Эти случаи не были включены в про-

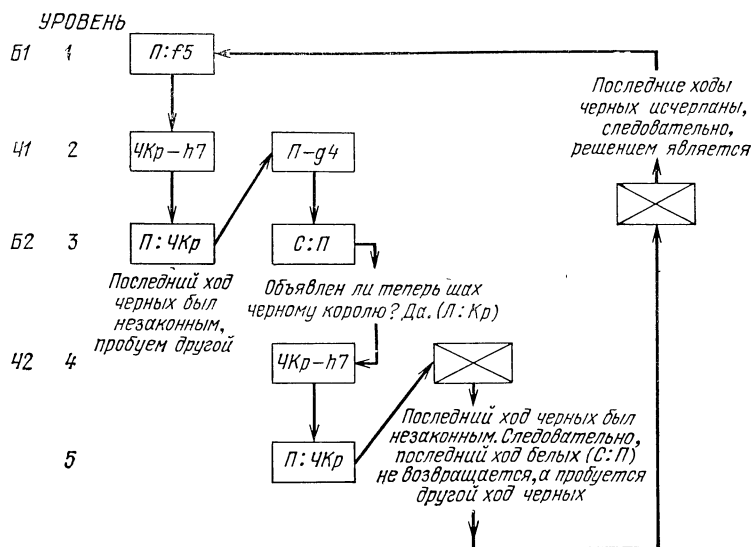


Рис. 5. Блок-схема решения задачи, приведенной на рис. 4.

грамму, потому что они замедляют ее работу, а встречаются в таких задачах редко. Однако, если требуется, то их можно включить в программу следующими способами.

### Рокировка

Для любой заданной позиции, в которой возможны четыре варианта рокировки, в программе должны быть предусмотрены четыре логические переменные. Оператор КОРОТКАЯ РОКИРОВКА БЕЛЫХ:=ДЛИННАЯ РОКИРОВКА БЕЛЫХ:=КОРОТКАЯ РОКИРОВКА ЧЕРНЫХ:=ДЛИННАЯ РОКИРОВКА ЧЕРНЫХ:=true; будет использоваться тогда, когда программа предназначается для позиции, в которой все еще возможны все рокировки (рис. 6).

Для упрощения задачи позиции короля, королевской ладьи и ферзевой ладьи в массиве СВОИ ПОЛЯ должны быть фиксированы для обеих сторон (например, 1, 2 и 3, как это дано на рис. 6,б и в) \*\*.

Для выполнения короткой рокировки белых нужно только дополнить массив

\* basic code. (Прим. ред.)

\*\* Этого можно достичь, если выписывать входные данные, подлежащие вводу в память машины, в соответствующем порядке: первым вводится король, второй — королевская ладья (если она имеется, или другая фигура в противном случае), третьей — ферзевая ладья и лишь после этого все остальные фигуры. Например, для получения начала массива ПОЛЯ БЕЛЫХ такого, как это показано на рис. 6,б (для позиции 6,а), нужно начинать входные данные следующими числами:  $n$ ; 6; 5; 4; 8; 4; 1; ..., где  $n$  — количество белых фигур, участвующих в задаче. (Прим. ред.)

СПИСОК следующей последовательностью чисел:

1 6 5—7    2 4 8 6    —8 1    6 7—5  
 Кр e1—g1    Л h1—f1    Возврат рокировки

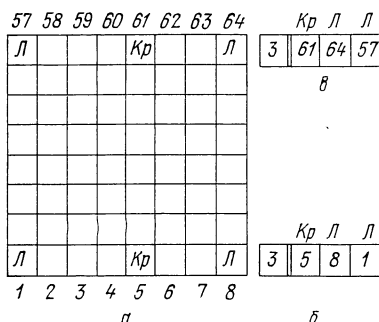
Тогда процедура ВЫПОЛНЕНИЕ ХОДА будет работать правильно.

Для длинной рокировки белых необходим следующий список:

1 6 5—3    3 4 1 4    —1    1 6 3—5  
 Кр e1—c1    Л a1—d1    Возврат рокировки

Рис. 6. Рокировка:

a — положение фигур, участвующих  
 в рокировке; б — ПОЛЯ БЕЛЫХ;  
 в — ПОЛЯ ЧЕРНЫХ.



Для короткой рокировки черных — список

1 6 61—63    2 4    64 62    —64    1 6 63—61  
 Кр e8—g8    Л h8—f8    Возврат рокировки

Для длинной рокировки черных — список

1 6 61—59    3 4 57 60    —57    1 6 59—61  
 Кр e8—c8    Л a8—d8    Возврат рокировки

Затруднение состоит в том, что необходимо выяснить, можно ли соответствующий список добавлять к массиву СПИСОК на законном основании. Необходимо поставить три вопроса.

1. Делались ли ходы королем или ладьями? (четыре логические переменные помогают ответить на этот вопрос).

2. Имеются ли фигуры между королем и ладьей?

3. Не находится ли под боем неприятельских фигур какое-нибудь из трех полей (на котором он находится, на которое или через которое он пойдет). Для того чтобы точно определить подлежащие проверке условия, этот отрывок программы выписан полностью в дополнении 2 к данному алгоритму. Процедура ХОДЫ РОКИРОВКИ подставляется в программу вместо пустой процедуры того же названия.

### Взятие пешки на проходе

Для реализации взятия пешки на проходе требуется использование массива НА ПРОХОДЕ, в котором запоминается позиция «фиктивной» пешки, когда пешка делает ход на два поля вперед. Необходимо проделать четыре следующие операции и вставить в программу указанные в них операторы с метками *ep1*, *ep2*, *ep3* и *ep4*\* (последний в двух местах) соответственно.

1. Проверить, делается ли данной пешкой ход на два поля вперед, и, если да, то запомнить позицию фиктивной пешки.

\* *ep* — сокращение от *en passant* — на проходе. (Прим. ред.)

ep1: НА ПРОХОДЕ [УРОВЕНЬ]:=

if q[УРОВЕНЬ]=1  $\wedge$  abs(ОБРАТНО К[УРОВЕНЬ]—НА)>10 then

(ОБРАТНО К[УРОВЕНЬ]+НА)÷2 else 0;

2. При составлении списка ходов проверить, может ли рассматриваемая пешка взять возможную фиктивную пешку, и, если да, то внести этот ход в СПИСОК.

ep2: if ЧУЖАЯ ДОСКА[ТАБЛИЦА ПЕШЕК[i]] $\neq$ 0 $\vee$

ТАБЛИЦА ПЕШЕК[i]=НА ПРОХОДЕ[УРОВЕНЬ—1] then

3. После выполнения хода пешкой проверить, не является ли он взятием на проходе, и, если да, то взять формально фигуру с отрицательным значением и убрать с доски фактическую пешку\*.

ep3: if q[УРОВЕНЬ]=1 $\wedge$  НА=НА ПРОХОДЕ[УРОВЕНЬ—1] then

begin ДОСКА ЧУЖИХ[НА]:=—1;

ДОСКА ЧУЖИХ[if НА<32 then НА+8 else НА—8]:=0

end;

4. При выполнении возврата хода проверить, не отрицательно ли значение c[УРОВЕНЬ], т. е. не был ли выполненный ход взятием на проходе. Если да, то восстановить фактическую пешку (это нужно сделать в двух местах программы).

ep4: if c[УРОВЕНЬ]=—1 then

ДОСКА ЧУЖИХ[if НА<32 then НА+8 else НА—8]:=0 else

ДОСКА ЧУЖИХ[ОТ]:=c[УРОВЕНЬ];

### *Ввод и вывод*

Рекомендуется таблицы и позиции вводить так, как это показано в дополнении 3. Процедура ПЕЧАТЬ ПОЗИЦИИ, приведенная в дополнении 4, полезна во время отладки программы. В обоих приложениях используются операторы ввода — вывода, принятые в системе АТЛАС (ICT, 1966) и являющиеся самоочевидными\*\*.

### *Резюме и приложения*

Данная работа посвящена главным образом описанию методов генерации списка всех законных ходов в любой шахматной позиции.

Здесь не находят разрешения следующие вопросы: сколько раз уже повторялась данная позиция, сколько ходов было сделано, начиная с последнего взятия фигуры или с последнего хода пешкой. Если бы эти вопросы были включены в рассмотрение, то задача значительно усложнилась бы, но нет ничего незаконного в том, чтобы их игнорировать. Кроме того, представляется невероятным, что такая ситуация может встретиться в реальной задаче.

Автор надеется, что данная работа будет полезна всем, кто захочет написать программу игры в шахматы. Разумеется, методы данной работы (особенно содержание таблиц) не догма и должны быть приспособлены к машине и используемому языку.

Главную ценность представляют следующие моменты.

1. Табличное управление программой.

2. Проверка законности хода при его фактическом выполнении и вызове для противника процедуры ОБЗОР ХОДОВ. Это исключает проблемы связки и хода под шах, на решение которых некоторые программы теряют время. Если все законно, то данная программа сразу же готова к выполнению ответного хода противника.

3. Превращение пешки в фигуру, рокировка и взятие пешки на проходе. Ни одна из ранее опубликованных программ не включала в себя рассмотрение этих операций.

---

\* Нижеследующие операторы с меткой «ep3:» (так же как и следующие далее операторы с меткой «ep4:») были опубликованы в алгоритме 50 неправильно. Правильные их варианты см. ниже в «Свидетельстве к алгоритму 50СJ». (Прим. ред.)

\*\* В алгоритме 50СJ операторы ввода — вывода системы АТЛАС заменены операторами ввода — вывода системы БЭСМ — АЛГОЛ [29, 45]. (Прим. ред.)

Они являются идиосинкразией шахматной игры, и их нелегко приспособить к таблично управляемой программе.

4. Тот факт, что предыдущие позиции как таковые не запоминаются, а вместо этого упаковывается информация (надеемся, что минимальная), из которой программа восстанавливает позицию, когда это требуется. Некоторые шахматные программы фактически каждый раз заново запоминают каждую новую позицию, которую может получить игрок. Следовательно, для решения задачи на мат в два хода потребовалось бы  $64$  (число полей)  $\times 30$  (среднее число ходов)  $\times 4$  (итерации)  $= 7680$  ячеек, не говоря уже о времени, затрачиваемом на их запоминание.

Освоив эти моменты, достаточные для решения шахматных задач, нужно дополнить их другими аспектами шахматной игры. Например, при разыгрывании шахматной партии процедура ОБЗОР ХОДОВ должна выдавать дополнительную информацию, такую как число контролируемых полей и их относительную важность (контроль центральных полей лучше, чем краевых).

Поскольку АЛГОЛ-программы обычно работают в 5—10 раз медленнее, чем эквивалентные им программы в автокоде, то данную программу нужно закодировать на одном из низших языков конкретной машины. Это не так трудно, как кажется, поскольку таблицы вообще не зависят от машины и языка. Перевод данной программы на язык Бэзик Атлас (Atlas Basic) занял около 10 дней.

Данная программа была также успешно проверена с помощью АЛГОЛ-транслятора машины CDC 6600, потребовав один день на перфорацию и изменение операторов ввода — вывода.

Наконец, главным образом для демонстрации, данная АЛГОЛ-программа была расширена в лаборатории SRC Atlas для фактической игры в шахматы на одной стороне доски. Больше всего труда было затрачено на разработку соответствующей системы ввода — вывода.

Стратегия была простой. Программа просматривала позицию на три хода вперед и была согласна меняться на равные или более ценные по рангу фигуры, отдавая предпочтение обмену более мощными фигурами (т. е. она всегда меняла ферзя на ферзя); если же не было никакого обмена или никакой возможности взятия фигура, то она пыталась взять под контроль как можно больше полей. Хотя эта программа и была написана на языке АЛГОЛ, она всегда почти мгновенно отвечала на ходы противника, выступавшего на одной стороне доски.

Данная программа сыграла две партии против программы Джона Скотта на машине ICL 1900 [8i], результаты приведены в табл. 28. (Поскольку ни одна из этих

Т а б л и ц а 28

Номер хода	Партия 1		Партия 2	
	Скотт (белые)	Атлас (черные)	Атлас (белые)	Скотт (черные)
1	d2 — d4	Kb8 — c6	e2 — e4	d7 — d5
2	e2 — e4	f7 — e6	e4:d5	Фd8:d5
3	d4 — d5	Cf8 — b4+	d2 — d4	Kg8 — f6
4	c2 — c3	Cb4 — c5	Cc1 — f4	Фd5 — e4+
5	b2 — b4	Kc6:b4	Cf4 — e3	Cc8 — d7
6	c3:b4	Фd8 — f6	d4 — d5	Фe4:d5
7	b4:c5	Фf6:a1	Фd1:d5	; Kf6:d5
8	Cf1 — d3	Фa1:a2	Ce3 — c5	e7 — e5
9	Kg1 — f3	e6:d5	Cc5:f8	Kpe8:f8
10	e4:d5	Фa2:d5	Cf1 — d3	Kd5 — f4
11	Kb2 — c3	Фd5:c5	Cd3 — c4	Kf4:g7+
12	Cc1 — d2		Kpe1 — d2	h7 — h5
13			Kb1 — c3	Lh8 — h6

программ не самообучающаяся, то возможны только две партии.) На четвертом ходе первой из этих партий стратегия была видоизменена с целью получения более выгодной позиции, т. е. я смошенничал.

В двух вышеприведенных случаях стратегия обмена фигур на равные или более ценные дает серенькие, но примерно равные партии. Программа Скотта выдает ответные ходы примерно через 1 мин, а составлена она в безик-коде 1900. Отсюда ясно, что поскольку шахматные программы тратят значительную часть времени на расчет вариантов, то описанные в данной работе технические приемы довольно эффективны.

Проблеме организации хорошей игры в шахматы на машине посвящены следующие работы: шахматная программа Гринблатта [5], методы игры, описанные Артуром Самюэлем, особенно «Alpha-Beta pruning» [7], и опыт анализа эндшпиля Барбары Хуберман [6].

### *Благодарности*

Автор хотел бы выразить благодарность личному составу вычислительной лаборатории «Атлас» за их помощь в подготовке к изданию данной работы.

### **Программа к алгоритму 50СJ для решения задач на мат в два хода \***

```
begin integer i,j,k,n,УРОВЕНЬ,Б1,Ч1,Б2,Ч2;
  Boolean ВОЗМОЖЕН ПАТ;
  integer array ТАБЛИЦА КОНЯ, ТАБЛИЦА КОРОЛЯ [1:576],
    ТАБЛИЦА СЛОНА, ТАБЛИЦА ЛАДЬИ [0:259], ТАБЛИЦА БП,
    ТАБЛИЦА ЧП [36:227], ПОЛЯ БЕЛЫХ, ПОЛЯ ЧЕРНЫХ [0:16],
    ДОСКА БЕЛЫХ, ДОСКА ЧЕРНЫХ [1:65], c,q,НОМЕР,
    РУБЕЖ ХОДОВ, ОБРАТНО К, НА ПРОХОДЕ [1:5], СПИСОК [1:500];
  procedure ПЕЧАТЬ ПОЗИЦИИ;
СМ ДОПОЛНЕНИЕ 4.;
  procedure ОБЗОР ХОДОВ (СВОИ ПОЛЯ, СВОЯ ДОСКА,
    ЧУЖАЯ ДОСКА, РУБЕЖ, ПРЕРЫВАНИЕ);
  integer РУБЕЖ; label ПРЕРЫВАНИЕ;
  integer array СВОИ ПОЛЯ, СВОЯ ДОСКА, ЧУЖАЯ ДОСКА;
  begin integer c,i,j,k,m,p, ПОЛЕ;
    switch К ХОДАМ: = ПЕШКОЙ, КОНЕМ, СЛОНОМ, ЛАДЬЕЙ,
      ФЕРЗЕМ, КОРОЛЕМ;
  procedure ХОДЫ РОКИРОВКИ;
    СМ ДОПОЛНЕНИЕ 2.;
  procedure ХОДЫ КОНЯ И КОРОЛЯ (ТАБЛИЦА КОНЬКОР);
  integer array ТАБЛИЦА КОНЬКОР;
  begin j: = 9 × ПОЛЕ;
    for i: = j — ТАБЛИЦА КОНЬКОР [j] step 1 until j — 1 do
      if СВОЯ ДОСКА [ТАБЛИЦА КОНЬКОР [i]] = 0 then
        begin
          if ЧУЖАЯ ДОСКА [ТАБЛИЦА КОНЬКОР [i]] = 6 then
            go to ПРЕРЫВАНИЕ;
          c: = c + 1; СПИСОК [c]: = ТАБЛИЦА КОНЬКОР [i]
        end
      end
    end ХОДЫ КОНЯ И КОРОЛЯ;
```

---

\* Границы массивов в начале нижеследующей программы являются значениями следующих выражений:  $576 = 9 \times 64$ ;  $259 = 4 \times 64 + 3$ ;  $36 = 4 \times 9$ ;  $227 = 4 \times 56 + 3$ ; 5 — число уровней, а 500 — максимальное (предположительно) количество ходов, возможных для фигур данной задачи на пяти уровнях. (Прим. ред.)

```

procedure ХОДЫ СЛОНА И ЛАДЬИ(ТАБЛИЦА СЛОНЛАД);
integer array ТАБЛИЦА СЛОНЛАД;
begin
  for j:=0 step 1 until 3 do
    begin k:=ТАБЛИЦА СЛОНЛАД[j];
      m:=ТАБЛИЦА СЛОНЛАД[4×ПОЛЕ+j];
      for i:=ПОЛЕ+k step k until m do
        begin
          if СВОЯ ДОСКА [i]≠0 then
            go to НОВОЕ НАПРАВЛЕНИЕ;
          c:=c+1; СПИСОК[c]:=i;
          if ЧУЖАЯ ДОСКА[i]≠0 then
            go to if ЧУЖАЯ ДОСКА[i]=6 then
              ПРЕРЫВАНИЕ else НОВОЕ НАПРАВЛЕНИЕ
          end i;
        end i;
    end j
  НОВОЕ НАПРАВЛЕНИЕ: end j
  end ХОДЫ СЛОНА И ЛАДЬИ;
procedure ХОДЫ ПЕШЕК(ТАБЛИЦА ПЕШЕК);
integer array ТАБЛИЦА ПЕШЕК;
begin
  for i:=4×ПОЛЕ,i+1 do
    begin j:=ТАБЛИЦА ПЕШЕК[i];
      if СВОЯ ДОСКА[j]≠0 ∨ ЧУЖАЯ ДОСКА[j]≠0 then
        go to ВЗЯТИЕ ПЕШКОЙ;
      c:=c+1;СПИСОК[c]:=j
    end i;
  ВЗЯТИЕ ПЕШКОЙ:for i:=4×ПОЛЕ+2,i+1 do
    ep2: if ЧУЖАЯ ДОСКА[ТАБЛИЦА ПЕШЕК[i]]≠0 then
      begin
        if ЧУЖАЯ ДОСКА[ТАБЛИЦА ПЕШЕК[i]]=6 then
          go to ПРЕРЫВАНИЕ;
        c:=c+1; СПИСОК[c]:=ТАБЛИЦА ПЕШЕК[i]
      end i
    end ХОДЫ ПЕШЕК;
    c:=РУБЕЖ;
    for p:=СВОИ ПОЛЯ[0] step —1 until 1 do
      begin ПОЛЕ:=СВОИ ПОЛЯ[p];
        if СВОЯ ДОСКА[ПОЛЕ]=0 thengo to ПРОДОЛЖЕНИЕ;
        СПИСОК[c]:=p; СПИСОК[c+1]:=СВОЯ ДОСКА[ПОЛЕ];
        СПИСОК[c+2]:=ПОЛЕ; c:=c+2;
        go to К ХОДАМ[СВОЯ ДОСКА[ПОЛЕ]];
      ПЕШКОЙ: if ДОСКА БЕЛЫХ[ПОЛЕ]=1 then ХОДЫ ПЕШЕК(ТАБЛИЦА БП) else
        ХОДЫ ПЕШЕК(ТАБЛИЦА ЧП);
        go to ДАЛЕЕ;
      КОНЕМ: ХОДЫ КОНЯ И КОРОЛЯ(ТАБЛИЦА КОНЯ); go to ДАЛЕЕ;
      КОРОЛЕМ: ХОДЫ КОНЯ И КОРОЛЯ(ТАБЛИЦА КОРОЛЯ); go to ДАЛЕЕ;
      ЛАДЬЕЙ: ХОДЫ СЛОНА И ЛАДЬИ(ТАБЛИЦА ЛАДЬИ); go to ДАЛЕЕ;
      ФЕРЗЕМ: ХОДЫ СЛОНА И ЛАДЬИ(ТАБЛИЦА ЛАДЬИ);
      СЛОНОМ: ХОДЫ СЛОНА И ЛАДЬИ(ТАБЛИЦА СЛОНА);
      ДАЛЕЕ: if СПИСОК[c]≠ПОЛЕ then
        begin СПИСОК[c+1]:=—ПОЛЕ; c:=c+2 end else
          c:=c—2;
        ПРОДОЛЖЕНИЕ: end p;
    end p;
  end

```

```

РОКИРОВКА: СМ ДОПОЛНЕНИЕ 2;;
    РУБЕЖ:=с
    end ОБЗОР ХОДОВ;
procedure ВЫПОЛНЕНИЕ ХОДА (р,ДОСКА СВОИХ,ДОСКА ЧУЖИХ,
    ПОЛЯ СВОИХ,ПОЛЯ ЧУЖИХ,ПРОДОЛЖЕНИЕ);
    integer p; label ПРОДОЛЖЕНИЕ;
    integer array ДОСКА СВОИХ,ДОСКА ЧУЖИХ,ПОЛЯ СВОИХ,
    ПОЛЯ ЧУЖИХ;
    begin integer ОТ,НА;
    go to start;
НЕЗАКОННЫЙ ХОД: УРОВЕНЬ:=УРОВЕНЬ-1; p:=p+1;
start:  ОТ:=СПИСОК[p];НА:=СПИСОК[p+1];
        if НА>0 then
            begin
ep1:          if q[УРОВЕНЬ]=-1 then go to aal;
aa2:          ДОСКА СВОИХ[ОТ]:=0;
aa3:          ДОСКА ЧУЖИХ[ОТ]:=с[УРОВЕНЬ];
aa4:          с[УРОВЕНЬ]:=ДОСКА ЧУЖИХ[НА];
aa5:          ДОСКА ЧУЖИХ[НА]:=0;
aa6:          ПОЛЯ СВОИХ[НОМЕР[УРОВЕНЬ]]:=НА;
aal:          if abs(q[УРОВЕНЬ])=1  $\wedge$  (НА>56  $\vee$  НА<9) then
                begin p:=p-1; q[УРОВЕНЬ]:=-1;
                    ДОСКА СВОИХ[НА]:= if ДОСКА СВОИХ[НА]=0 then
                        5 else ДОСКА СВОИХ[НА]-1;
                    if ДОСКА СВОИХ[НА]=2 then
                        begin p:=p+1; q[УРОВЕНЬ]:=1 end
                    end else
                    ДОСКА СВОИХ[НА]:=abs(q[УРОВЕНЬ]);
ep3:          УРОВЕНЬ:=УРОВЕНЬ+1;
                НА:=РУБЕЖ ХОДОВ[УРОВЕНЬ]:=
                    РУБЕЖ ХОДОВ[УРОВЕНЬ-1];
ЗАКОННЫЙ ЛИ ХОД: ОБЗОР ХОДОВ (ПОЛЯ ЧУЖИХ,ДОСКА ЧУЖИХ,
                ДОСКА СВОИХ,РУБЕЖ ХОДОВ[УРОВЕНЬ],
                НЕЗАКОННЫЙ ХОД);
                НОМЕР[УРОВЕНЬ]:=СПИСОК[НА];
                q[УРОВЕНЬ]:=СПИСОК[НА+1];
                ОБРАТНО К[УРОВЕНЬ]:=СПИСОК[НА+2]
            end else
            begin
bb1:          ДОСКА СВОИХ[-НА]:=abs(q[УРОВЕНЬ]);
bb2:          ДОСКА СВОИХ[ОТ]:=0;
ep4:          ДОСКА ЧУЖИХ[ОТ]:=с[УРОВЕНЬ];
bb3:          с[УРОВЕНЬ]:=0;
bb4:          ПОЛЯ СВОИХ[НОМЕР[УРОВЕНЬ]]:=-НА;
bb5:          НОМЕР[УРОВЕНЬ]:=СПИСОК[p+2];
                q[УРОВЕНЬ]:=СПИСОК[p+3];
                ОБРАТНО К[УРОВЕНЬ]:=СПИСОК[p+4]; p:=p+3;
                go to ПРОДОЛЖЕНИЕ
            end
        end

```

```

    end ВЫПОЛНЕНИЕ ХОДА;
procedure ВОЗВРАТ ХОДА(ОТ,ДОСКА СВОИХ,ДОСКА ЧУЖИХ,
    ПОЛЯ СВОИХ);
    integer ОТ;
    integer array ДОСКА СВОИХ,ДОСКА ЧУЖИХ,ПОЛЯ СВОИХ;
    begin ДОСКА СВОИХ[ОБРАТНО К[УРОВЕНЬ]]:=abs(q[УРОВЕНЬ]);
    ДОСКА СВОИХ[ОТ]:=0;
ep4:    ДОСКА ЧУЖИХ[ОТ]:=c[УРОВЕНЬ];
        c[УРОВЕНЬ]:=0;
        ПОЛЯ СВОИХ[НОМЕР[УРОВЕНЬ]]:=ОБРАТНО К[УРОВЕНЬ]
    end ВОЗВРАТ ХОДА;
ВВОД ТАБЛИЦ:
    input(ТАБЛИЦА КОНЯ,ТАБЛИЦА КОРОЛЯ, ТАБЛИЦА ЛАДЬИ,
        ТАБЛИЦА СЛОНА,ТАБЛИЦА БП,ТАБЛИЦА ЧП);
ОЧИСТКА ДОСOK:
    for i:=1 step 1 until 65 do
        ДОСКА БЕЛЫХ[i]:=ДОСКА ЧЕРНЫХ[i]:=0;
    for i:= 1 step 1 until 5 do c[i]:=0;
ВВОД БЕЛЫХ ФИГУР:
    input(n); ПОЛЯ БЕЛЫХ[0]:=n;
    for i:=1 step 1 until n do
        begin input(j,k);
            ДОСКА БЕЛЫХ[k]:=j; ПОЛЯ БЕЛЫХ[i]:=k
        end i;
ВВОД ЧЕРНЫХ ФИГУР:
    input(n); ПОЛЯ ЧЕРНЫХ[0]:=n;
    for i:=1 step 1 until n do
        begin input(j,k);
            ДОСКА ЧЕРНЫХ[k]:=j; ПОЛЯ ЧЕРНЫХ[i]:=k
        end i;
    output('Т', 'ИСХОДНАЯ ПОЗИЦИЯ', '/'); ПЕЧАТЬ ПОЗИЦИИ;
НАЧАЛО: c[1]:=0; УРОВЕНЬ:=РУБЕЖ ХОДОВ[1]:=1;
ОБЗОР ХОДОВ(ПОЛЯ БЕЛЫХ,ДОСКА БЕЛЫХ,ДОСКА ЧЕРНЫХ,
    РУБЕЖ ХОДОВ[1],ОШИБКА);
НОМЕР[УРОВЕНЬ]:=СПИСОК[1]; q[УРОВЕНЬ]:=СПИСОК[2];
for Б1:=3 step 1 until РУБЕЖ ХОДОВ[1] do
    begin ВОЗМОЖЕН ПАТ:=true;
        ВЫПОЛНЕНИЕ ХОДА(Б1,ДОСКА БЕЛЫХ,ДОСКА ЧЕРНЫХ,
            ПОЛЯ БЕЛЫХ,ПОЛЯ ЧЕРНЫХ,Б1 ПРОДОЛЖЕНИЕ);
        for Ч1:=РУБЕЖ ХОДОВ[1]+2 step 1 until РУБЕЖ ХОДОВ[2] do
            begin
                ВЫПОЛНЕНИЕ ХОДА(Ч1,ДОСКА ЧЕРНЫХ,ДОСКА БЕЛЫХ,
                    ПОЛЯ ЧЕРНЫХ,ПОЛЯ БЕЛЫХ,Ч1 ПРОДОЛЖЕНИЕ);
                for Б2:=РУБЕЖ ХОДОВ[2]+2 step 1 until
                    РУБЕЖ ХОДОВ[3] do
                    begin ВОЗМОЖЕН ПАТ:=false;
                        ВЫПОЛНЕНИЕ ХОДА(Б2,ДОСКА БЕЛЫХ,
                            ДОСКА ЧЕРНЫХ,ПОЛЯ БЕЛЫХ,ПОЛЯ ЧЕРНЫХ,
                            Б2 ПРОДОЛЖЕНИЕ);
                    end
                end
            end
        end
    end
ЕСТЬ ЛИ ШАХ ЧЕРНЫМ:n:=РУБЕЖ ХОДОВ[4];
    ОБЗОР ХОДОВ(ПОЛЯ БЕЛЫХ,ДОСКА БЕЛЫХ,ДОСКА
        ЧЕРНЫХ,n,ШАХ ЧЕРНЫМ);

```



```

    go to Б2 ПРОДОЛЖЕНИЕ;
ШАХ ЧЕРНЫМ: for Ч2:=РУБЕЖ ХОДОВ[3]+2 step 1 until
    РУБЕЖ ХОДОВ[4] do
    begin
        ВЫПОЛНЕНИЕ ХОДА (Ч2,ДОСКА ЧЕРНЫХ,
            ДОСКА БЕЛЫХ,ПОЛЯ ЧЕРНЫХ,ПОЛЯ БЕЛЫХ,
            Ч2 ПРОДОЛЖЕНИЕ);
УШЛИ ОТ ШАХА: УРОВЕНЬ:=4;
        ВОЗВРАТ ХОДА (СПИСОК[if q[УРОВЕНЬ]=-1 then
            Ч2+2 else Ч2+1],ДОСКА ЧЕРНЫХ,
            ДОСКА БЕЛЫХ,ПОЛЯ ЧЕРНЫХ);
        go to Б2 ПРОДОЛЖЕНИЕ;
Ч2 ПРОДОЛЖЕНИЕ: УРОВЕНЬ:=4
    end Ч2;
ЧЕРНЫЕ БЕРУТ НАЗАД СВОЙ ПЕРВЫЙ ХОД:УРОВЕНЬ:=3;
        ВОЗВРАТ ХОДА (СПИСОК[if q[УРОВЕНЬ]=-1 then
            Б2+2 else Б2+1],ДОСКА БЕЛЫХ,ДОСКА
            ЧЕРНЫХ,ПОЛЯ БЕЛЫХ);
ВОЗВРАТ КОРОЛЯ БЕЛЫХ:
СМ СВИДЕТЕЛЬСТВО К АЛГОРИТМУ 50СJ.;
        go to Ч1 ПРОДОЛЖЕНИЕ;
Б2 ПРОДОЛЖЕНИЕ: УРОВЕНЬ:=3
    end Б2;
БЕЛЫЕ БЕРУТ НАЗАД СВОЙ ПЕРВЫЙ ХОД:УРОВЕНЬ:=2;
        ВОЗВРАТ ХОДА (СПИСОК[if q[УРОВЕНЬ]=-1 then
            Ч1+2 else Ч1+1], ДОСКА ЧЕРНЫХ,ДОСКА БЕЛЫХ,
            ПОЛЯ ЧЕРНЫХ);
ВОЗВРАТ КОРОЛЯ ЧЕРНЫХ:
СМ СВИДЕТЕЛЬСТВО К АЛГОРИТМУ 50СJ.;
        go to Б1 ПРОДОЛЖЕНИЕ;
Ч1 ПРОДОЛЖЕНИЕ: УРОВЕНЬ:=2
    end Ч1;
    n:=РУБЕЖ ХОДОВ[2];
    if ВОЗМОЖЕН ПАТ then
НЕ МАТ ЛИ ЭТО:
        begin ОБЗОР ХОДОВ (ПОЛЯ БЕЛЫХ,ДОСКА БЕЛЫХ,
            ДОСКА ЧЕРНЫХ,n,МАТ1);
ЭТО ПАТ НА ПЕРВОМ ХОДЕ:
        go to Б1 ПРОДОЛЖЕНИЕ
    end;
        output('/',Т,'РЕШЕНИЕ','/'); goto ОТВЕТ;
МАТ1: output('/',Т,'МАТ НА ПЕРВОМ ХОДЕ','/');
ОТВЕТ: ПЕЧАТЬ ПОЗИЦИИ;
ОДНО РЕШЕНИЕ НАЙДЕНО: go to КОНЕЦ;
Б1 ПРОДОЛЖЕНИЕ: УРОВЕНЬ:=1
    end Б1;
        output('/',Т,'ХОДЫ БЕЛЫХ ИСЧЕРПАНЫ','/');
        go to КОНЕЦ;
ОШИБКА: output(Т,'ПЕРВЫМ ХОДОМ БЕРЕТСЯ КОРОЛЬ ЧЕРНЫХ','/');
        КОНЕЦ: end

```

# Дополнение 1 к алгоритму 50С1

## Дополнение 1.1а

Таблица коня

Исходное поле	Поля возможных ходов								Число ходов
1	0	0	0	0	0	0	18	11	2
2	0	0	0	0	0	17	19	12	3
3	0	0	0	0	9	18	20	13	4
4	0	0	0	0	10	19	21	14	4
5	0	0	0	0	11	20	22	15	4
6	0	0	0	0	12	21	23	16	4
7	0	0	0	0	0	13	22	24	3
8	0	0	0	0	0	0	14	23	2
9	0	0	0	0	0	26	19	3	3
10	0	0	0	0	25	27	20	4	4
11	0	0	17	26	28	21	5	1	6
12	0	0	18	27	29	22	6	2	6
13	0	0	19	28	30	23	7	3	6
14	0	0	20	29	31	24	8	4	6
15	0	0	0	0	21	30	32	5	4
16	0	0	0	0	0	22	31	6	3
17	0	0	0	0	34	27	11	2	4
18	0	0	33	35	28	12	3	1	6
19	25	34	36	29	13	4	2	9	8
20	26	35	37	30	14	5	3	10	8
21	27	36	38	31	15	6	4	11	8
22	28	37	39	32	16	7	5	12	8
23	0	0	29	38	40	8	6	13	6
24	0	0	0	0	30	39	7	14	4
25	0	0	0	0	42	35	19	10	4
26	0	0	41	43	36	20	11	9	6
27	33	42	44	37	21	12	10	17	8
28	34	43	45	38	22	13	11	18	8
29	35	44	46	39	23	14	12	19	8
30	36	45	47	40	24	15	13	20	8
31	0	0	37	46	48	16	14	21	6
32	0	0	0	0	38	47	15	22	4
33	0	0	0	0	50	43	27	18	4
34	0	0	49	51	44	28	19	17	6
35	41	50	52	45	29	20	18	25	8
36	42	51	53	46	30	21	19	26	8
37	43	52	54	47	31	22	20	27	8
38	44	53	55	48	32	23	21	28	8
39	0	0	45	54	56	24	22	29	6
40	0	0	0	0	46	55	23	30	4
41	0	0	0	0	58	51	35	26	4
42	0	0	57	59	52	36	27	25	6
43	49	58	60	53	37	28	26	33	8
44	50	59	61	54	38	29	27	34	8
45	51	60	62	55	39	30	28	35	8
46	52	61	63	56	40	31	29	36	8
47	0	0	53	62	64	32	30	37	8
48	0	0	0	0	54	63	31	38	6

Продолжение таблицы коня

Исходное поле	Поля возможных ходов								Число ходов
49	0	0	0	0	0	59	43	34	3
50	0	0	0	0	60	44	35	33	4
51	0	0	57	61	45	36	34	41	6
52	0	0	58	62	46	37	35	42	6
53	0	0	59	63	47	38	36	43	6
54	0	0	60	64	48	39	37	44	6
55	0	0	0	0	61	40	38	45	4
56	0	0	0	0	0	62	39	46	3
57	0	0	0	0	0	0	51	42	2
58	0	0	0	0	0	52	43	41	3
59	0	0	0	0	53	44	42	49	4
60	0	0	0	0	54	45	43	50	4
61	0	0	0	0	55	46	44	51	4
62	0	0	0	0	56	47	45	52	4
63	0	0	0	0	0	48	46	53	3
64	0	0	0	0	0	0	47	54	2

Таблица короля

Дополнение 1.16

Исходное поле	Поля возможных ходов								Число ходов
1	0	0	0	0	0	2	9	10	3
2	0	0	0	1	3	9	10	11	5
3	0	0	0	2	4	10	11	12	5
4	0	0	0	3	5	11	12	13	5
5	0	0	0	4	6	12	13	14	5
6	0	0	0	5	7	13	14	15	5
7	0	0	0	6	8	14	15	16	5
8	0	0	0	0	0	7	15	16	3
9	0	0	0	1	2	10	17	18	5
10	1	2	3	9	11	17	18	19	8
11	2	3	4	10	12	18	19	20	8
12	3	4	5	11	13	19	20	21	8
13	4	5	6	12	14	20	21	22	8
14	5	6	7	13	15	21	22	23	8
15	6	7	8	14	16	22	23	24	8
16	0	0	0	7	8	15	23	24	5
17	0	0	0	9	10	18	25	26	5
18	9	10	11	17	19	25	26	27	8
19	10	11	12	18	20	26	27	28	8
20	11	12	13	19	21	27	28	29	8
21	12	13	14	20	22	28	29	30	8
22	13	14	15	21	23	29	30	31	8
23	14	15	16	22	24	30	31	32	8
24	0	0	0	15	16	23	31	32	5
25	0	0	0	17	18	26	33	34	5
26	17	18	19	25	27	33	34	35	8
27	18	19	20	26	28	34	35	36	8
28	19	20	21	27	29	35	36	37	8
29	20	21	22	28	30	36	37	38	8
30	21	22	23	29	31	37	38	39	8
31	22	23	24	30	32	38	39	40	8
32	0	0	0	23	24	31	39	40	5

Продолжение таблицы короля

Исходное поле	Поля возможных ходов									Число ходов
33	0	0	0	25	26	34	41	42		5
34	25	26	27	33	35	41	42	43		8
35	26	27	28	34	36	42	43	44		8
36	27	28	29	35	37	43	44	45		8
37	28	29	30	36	38	44	45	46		8
38	29	30	31	37	39	45	46	47		8
39	30	31	32	38	40	46	47	48		8
40	0	0	0	31	32	39	47	48		5
41	0	0	0	33	34	42	49	50		5
42	33	34	35	41	43	49	50	51		8
43	34	35	36	42	44	50	51	52		8
44	35	36	37	43	45	51	52	53		8
45	36	37	38	44	46	52	53	54		8
46	37	38	39	45	47	53	54	55		8
47	38	39	40	46	48	54	55	56		8
48	0	0	0	39	40	47	55	56		5
49	0	0	0	41	42	50	57	58		5
50	41	42	43	49	51	57	58	59		8
51	42	43	44	50	52	58	59	60		8
52	43	44	45	51	53	59	60	61		8
53	44	45	46	52	54	60	61	62		8
54	45	46	47	53	55	61	62	63		8
55	46	47	48	54	56	62	63	64		8
56	0	0	0	47	48	55	63	64		5
57	0	0	0	0	0	49	50	58		3
58	0	0	0	49	50	51	57	59		5
59	0	0	0	50	51	52	58	60		5
60	0	0	0	51	52	53	59	61		5
61	0	0	0	52	53	54	60	62		5
62	0	0	0	53	54	55	61	63		5
63	0	0	0	54	55	56	62	64		5
64	0	0	0	0	0	55	56	63		3

Дополнение 1.2а

Дополнение 1.2б

Дополнение 1.2а

Дополнение 1.2б

Таблица ладьи

Таблица слона

Таблица ладьи

Таблица слона

Исходное поле	Предельные поля возможных ходов								Исходное поле	Предельные поля возможных ходов							
	В    З    С    Ю				СВ   ЮЗ   СЗ   ЮВ					В    З    С    Ю				СВ   ЮЗ   СЗ   ЮВ			
	1   —1    8   —8				9   —9    7   —7					1   —1    8   —8				9   —9    7   —7			
1	8	1	57	1	64	1	1	1	9	16	9	57	1	63	9	9	2
2	8	1	58	2	56	2	9	2	10	16	9	58	2	64	1	17	3
3	8	1	59	3	48	3	17	3	11	16	9	59	3	56	2	25	4
4	8	1	60	4	40	4	25	4	12	16	9	60	4	48	3	33	5
5	8	1	61	5	32	5	33	5	13	16	9	61	5	40	4	41	6
6	8	1	62	6	24	6	41	6	14	16	9	62	6	32	5	49	7
7	8	1	63	7	16	7	49	7	15	16	9	63	7	24	6	57	8
8	8	1	64	8	8	8	57	8	16	16	9	64	8	16	7	58	16

**Продолжение таблицы ладьи и слона**

Исходное поле	Предельные поля возможных х ходов								Исходное поле	Предельные поля возможных ходов							
	В З С Ю				СВ ЮЗ СЗ ЮВ					В З С Ю				СВ ЮЗ СЗ ЮВ			
	1 —1	8 —8			9 —9	7 —7				1 —1	8 —8			9 —9	7 —7		
17	24	17	57	1	62	17	17	3	41	48	41	57	1	59	41	41	6
18	24	17	58	2	63	9	25	4	42	48	41	58	2	60	33	49	7
19	24	17	59	3	64	1	33	5	43	48	41	59	3	61	25	57	8
20	24	17	60	4	56	2	41	6	44	48	41	60	4	62	17	58	16
21	24	17	61	5	48	3	49	7	45	48	41	61	5	63	9	59	24
22	24	17	62	6	40	4	57	8	46	48	41	62	6	64	1	60	32
23	24	17	63	7	32	5	58	16	47	48	41	63	7	56	2	61	40
24	24	17	64	8	24	6	59	24	48	48	41	64	8	48	3	62	48
25	32	25	57	1	61	25	25	4	49	56	49	57	1	58	49	49	7
26	32	25	58	2	62	17	33	5	50	56	49	58	2	59	41	57	8
27	32	25	59	3	63	9	41	6	51	56	49	59	3	60	33	58	16
28	32	25	60	4	64	1	49	7	52	56	49	60	4	61	25	59	24
29	32	25	61	5	56	2	57	8	53	56	49	61	5	62	17	60	32
30	32	25	62	6	48	3	58	16	54	56	49	62	6	63	9	61	40
31	32	25	63	7	40	4	59	24	55	56	49	63	7	64	1	62	48
32	32	25	64	8	32	5	60	32	56	56	49	64	8	56	2	63	56
33	40	33	57	1	60	33	33	5	57	64	57	57	1	57	57	57	8
34	40	33	58	2	61	25	41	6	58	64	57	58	2	58	49	58	16
35	40	33	59	3	62	17	49	7	59	64	57	59	3	59	41	59	24
36	40	33	60	4	63	9	57	8	60	64	57	60	4	60	33	60	32
37	40	33	61	5	64	1	58	16	61	64	57	61	5	61	25	61	40
38	40	33	62	6	56	2	59	24	62	64	57	62	6	62	17	62	48
39	40	33	63	7	48	3	60	32	63	64	57	63	7	63	9	63	56
40	40	33	64	8	40	4	61	40	64	64	57	64	8	64	1	64	64

**Дополнение 1.3а**

*Таблица белой пешки*

**Дополнение 1.3б**

*Таблица черной пешки*

**Дополнение 1.3а**

*Таблица белой пешки*

**Дополнение 1.3б**

*Таблица черной пешки*

Исходное поле	Поля возможных ходов								Исходное поле	Поля возможных ходов							
	+8	+16	+7	+9	—8	—16	—7	—9		+8	+16	+7	+9	—8	—16	—7	—9
9	17	25	65	18	1	9	2	65	25	33	25	65	34	17	25	18	65
10	18	26	17	19	2	10	3	1	26	34	26	33	35	18	26	19	17
11	19	27	18	20	3	11	4	2	27	35	27	34	36	19	27	20	18
12	20	28	19	21	4	12	5	3	28	36	28	35	37	20	28	21	19
13	21	29	20	22	5	13	6	4	29	37	29	36	38	21	29	22	20
14	22	30	21	23	6	14	7	5	30	38	30	37	39	22	30	23	21
15	23	31	22	24	7	15	8	6	31	39	31	38	40	23	31	24	22
16	24	32	23	65	8	16	65	7	32	40	32	39	65	24	32	65	23
17	25	17	65	26	9	17	10	65	33	41	33	65	42	25	33	26	65
18	26	18	25	27	10	18	11	9	34	42	34	41	43	26	34	27	25
19	27	19	26	28	11	19	12	10	35	43	35	42	44	27	35	28	26
20	28	20	27	29	12	20	13	11	36	44	36	43	45	28	36	29	27
21	29	21	28	30	13	21	14	12	37	45	37	44	46	29	37	30	28
22	30	22	29	31	14	22	15	13	38	46	38	45	47	30	38	31	29
23	31	23	30	32	15	23	16	14	39	47	39	46	48	31	39	32	30
24	32	24	31	65	16	24	65	15	40	48	40	47	65	32	40	65	31

**Продолжение таблицы белой и черной пешек**

Исходное поле	Поля возможных ходов								Исходное поле	Поля возможных ходов							
	+8	+16	+7	+9	—8	—16	—7	—9		+8	+16	+7	+9	—8	—16	—7	—9
41	49	41	65	50	33	41	34	65	49	57	49	65	58	41	33	42	65
42	50	42	49	51	34	42	35	33	50	58	50	57	59	42	34	43	41
43	51	43	50	52	35	43	36	34	51	59	51	58	60	43	35	44	42
44	52	44	51	53	36	44	37	35	52	60	52	59	61	44	36	45	43
45	53	45	52	54	37	45	38	36	53	61	53	60	62	45	37	46	44
46	54	46	53	55	38	46	39	37	54	62	54	61	63	46	38	47	45
47	55	47	54	56	39	47	40	38	55	63	55	62	64	47	39	48	46
48	56	48	55	65	40	48	65	39	56	64	56	63	65	48	40	65	47

**Дополнение 2 к алгоритму 50СJ**

Следующие операторы вставляются в программу, начиная с метки РОКИРОВКА\*.

РОКИРОВКА:

if СВОИ ПОЛЯ[1]=5 then go to РОКИРОВКА БЕЛЫХ else  
if СВОИ ПОЛЯ[1]=61 then go to РОКИРОВКА ЧЕРНЫХ else  
go to КОНЕЦ РОКИРОВКИ;

РОКИРОВКА БЕЛЫХ: if  $\neg$  КОРОТКАЯ РОКИРОВКА БЕЛЫХ  $\vee$   
ДОСКА БЕЛЫХ[6] $\neq$ 0  $\vee$   
ДОСКА БЕЛЫХ[7] $\neq$ 0  $\vee$  ДОСКА ЧЕРНЫХ[6] $\neq$ 0  $\vee$   
ДОСКА ЧЕРНЫХ[7] $\neq$ 0 then go to ДЛИННАЯ БЕЛЫХ;

for n:=УРОВЕНЬ—2 step —2 until 1 do

if НОМЕР[n]=1  $\vee$  НОМЕР[n]=2 then  
go to ДЛИННАЯ БЕЛЫХ;

ДОСКА БЕЛЫХ[6]:=ДОСКА БЕЛЫХ[7]:=6;n:=c;

ОБЗОР ХОДОВ (ПОЛЯ ЧЕРНЫХ, ДОСКА ЧЕРНЫХ, ДОСКА БЕЛЫХ, n,  
НЕ ВОЗМОЖНА КОРОТКАЯ БЕЛЫХ);

ВОЗМОЖНА КОРОТКАЯ БЕЛЫХ: ХОДЫ РОКИРОВКИ (5,7,2,8,6);

НЕ ВОЗМОЖНА КОРОТКАЯ БЕЛЫХ:

ДОСКА БЕЛЫХ[6]:=ДОСКА БЕЛЫХ[7]:=0;

ДЛИННАЯ БЕЛЫХ: if  $\neg$  ДЛИННАЯ РОКИРОВКА БЕЛЫХ  $\vee$

ДОСКА БЕЛЫХ[2] $\neq$ 0  $\vee$

ДОСКА БЕЛЫХ[3] $\neq$ 0  $\vee$  ДОСКА БЕЛЫХ[4] $\neq$ 0  $\vee$

ДОСКА ЧЕРНЫХ[2] $\neq$ 0  $\vee$  ДОСКА ЧЕРНЫХ[3] $\neq$ 0  $\vee$

ДОСКА ЧЕРНЫХ[4] $\neq$ 0 then go to КОНЕЦ РОКИРОВКИ;

for n:=УРОВЕНЬ—2 step —2 until 1 do

if НОМЕР[n]=1  $\vee$  НОМЕР[n]=3 then

go to КОНЕЦ РОКИРОВКИ;

---

\* В нижеследующих операторах редактором выпуска сделаны некоторые тождественные сокращения и внесены поправки, указанные в нижеследующем свидетельстве к алгоритму 50СJ. (Прим. ред.)

ДОСКА БЕЛЫХ[3]:=ДОСКА БЕЛЫХ[4]:=6; n:=c;  
 ОБЗОР ХОДОВ(ПОЛЯ ЧЕРНЫХ, ДОСКА ЧЕРНЫХ,ДОСКА БЕЛЫХ,n,  
 НЕ ВОЗМОЖНА ДЛИННАЯ БЕЛЫХ);  
 ВОЗМОЖНА ДЛИННАЯ БЕЛЫХ: ХОДЫ РОКИРОВКИ(5,3,3,1,4);  
 НЕ ВОЗМОЖНА ДЛИННАЯ БЕЛЫХ:  
 ДОСКА БЕЛЫХ[3]:=ДОСКА БЕЛЫХ[4]:=0;  
 go to КОНЕЦ РОКИРОВКИ;  
 РОКИРОВКА ЧЕРНЫХ: if  $\neg$  КОРОТКАЯ РОКИРОВКА ЧЕРНЫХ  $\vee$   
 ДОСКА ЧЕРНЫХ[62] $\neq$ 0  $\vee$  ДОСКА ЧЕРНЫХ[63] $\neq$ 0  $\vee$   
 ДОСКА БЕЛЫХ[62] $\neq$ 0  $\vee$  ДОСКА БЕЛЫХ[63] $\neq$ 0 then  
 go to ДЛИННАЯ ЧЕРНЫХ;  
 for n:=УРОВЕНЬ—2 step —2 until 2 do  
 if НОМЕР[n]=1  $\vee$  НОМЕР[n]=2 then go to ДЛИННАЯ ЧЕРНЫХ;  
 ДОСКА ЧЕРНЫХ[62]:=ДОСКА ЧЕРНЫХ[63]:=6; n:=c;  
 ОБЗОР ХОДОВ(ПОЛЯ БЕЛЫХ,ДОСКА БЕЛЫХ,ДОСКА ЧЕРНЫХ,n,  
 НЕ ВОЗМОЖНА КОРОТКАЯ ЧЕРНЫХ);  
 ВОЗМОЖНА КОРОТКАЯ ЧЕРНЫХ: ХОДЫ РОКИРОВКИ(61,63,2,64,62);  
 НЕ ВОЗМОЖНА КОРОТКАЯ ЧЕРНЫХ:  
 ДОСКА ЧЕРНЫХ[62]:=ДОСКА ЧЕРНЫХ[63]:=0;  
 ДЛИННАЯ ЧЕРНЫХ: if  $\neg$ ДЛИННАЯ РОКИРОВКА ЧЕРНЫХ  $\vee$   
 ДОСКА ЧЕРНЫХ[58] $\neq$ 0  $\vee$  ДОСКА ЧЕРНЫХ[59] $\neq$ 0  $\vee$   
 ДОСКА ЧЕРНЫХ[60] $\neq$ 0  $\vee$  ДОСКА БЕЛЫХ[58] $\neq$ 0  $\vee$   
 ДОСКА БЕЛЫХ[59] $\neq$ 0  $\vee$  ДОСКА БЕЛЫХ[60] $\neq$ 0 then  
 go to КОНЕЦ РОКИРОВКИ;  
 for n:=УРОВЕНЬ—2 step —2 until 2 do  
 if НОМЕР[n]=1  $\vee$  НОМЕР[n]=3 then  
 go to КОНЕЦ РОКИРОВКИ;  
 ДОСКА ЧЕРНЫХ[59]:=ДОСКА ЧЕРНЫХ[60]:=6; n:=c;  
 ОБЗОР ХОДОВ(ПОЛЯ БЕЛЫХ,ДОСКА БЕЛЫХ,ДОСКА ЧЕРНЫХ,n,  
 НЕ ВОЗМОЖНА ДЛИННАЯ ЧЕРНЫХ);  
 ВОЗМОЖНА ДЛИННАЯ ЧЕРНЫХ:ХОДЫ РОКИРОВКИ(61,59,3,57,60);  
 НЕ ВОЗМОЖНА ДЛИННАЯ ЧЕРНЫХ:  
 ДОСКА ЧЕРНЫХ[59]:=ДОСКА ЧЕРНЫХ[60]:=0;  
 КОНЕЦ РОКИРОВКИ:

Нижеследующее описание процедуры помещается в программу на место пустой  
 процедуры ХОДЫ РОКИРОВКИ.

```

procedure ХОДЫ РОКИРОВКИ(КР1,КР2,НОМЕР ЛАДЬИ,Л1,Л2);
  integer КР1,КР2,НОМЕР ЛАДЬИ,Л1,Л2;
begin СПИСОК[c]:=1; СПИСОК[c+1]:=6;
  СПИСОК[c+2]:=КР1; СПИСОК[c+3]:=-КР2;
  СПИСОК[c+4]:=НОМЕР ЛАДЬИ; СПИСОК[c+5]:=4;
  СПИСОК[c+6]:=Л1; СПИСОК[c+7]:=Л2;
  СПИСОК[c+8]:=-Л1; СПИСОК[c+9]:=1;

```

```

СПИСОК[c+10]:=6; СПИСОК[c+11]:=КР2;
СПИСОК[c+12]:=-КР1; c:=c+13
end ХОДЫ РОКИРОВКИ;

```

Наконец, во внешнем блоке программы должно быть описание логических переменных, подобное следующему:

```

Boolean КОРОТКАЯ РОКИРОВКА БЕЛЫХ, ДЛИННАЯ РОКИРОВКА БЕЛЫХ,
      КОРОТКАЯ РОКИРОВКА ЧЕРНЫХ, ДЛИННАЯ РОКИРОВКА ЧЕРНЫХ;
а после метки НАЧАЛО должно быть присваивание этим переменным начальных значений в зависимости от того, какая из рокировок еще допускается *.

```

### Дополнение 3 к алгоритму 50СJ

Ниже приведены операторы, обеспечивающие ввод таблиц дополнения 1 и исходной позиции задачи на мат в два хода... \*\*

Для примера, приведенного на рис. 3, исходные данные должны поступать в читающее устройство в следующем порядке:

```

11
6 10 1 12 5 14 2 18 1 22 1 32 4 33 1 39 3 51 3 57 2 62
7
1 20 1 28 1 30 2 36 5 37 6 38 1 46.

```

### Дополнение 4 к алгоритму 50СJ

Нижеследующая процедура печатает позицию с обозначением фигур, подобным изображенному на рис. 3,а...\*\*\*

```

procedure ПЕЧАТЬ ПОЗИЦИИ;
begin switch ФИГУРА:=П,К,С,Л,Ф,КР;
  for i:=56 step -8 until 0 do
    begin output('/');
      for j:=1 step 1 until 8 do
        begin output('ВВ');
          if ДОСКА БЕЛЫХ[i+j]=0 then
            go to ЧЕРНАЯ else output('Т','+');
          go to ФИГУРА [ДОСКА БЕЛЫХ[i+j]];
ЧЕРНАЯ: if ДОСКА ЧЕРНЫХ[i+j]=0 then
            go to ПОЛЕ СВОБОДНО else output('Т','—');
          go to ФИГУРА[ДОСКА ЧЕРНЫХ[i+j]];
П:      output('Т','П □'); go to НОВОЕ ПОЛЕ;
К:      output('Т','К □'); go to НОВОЕ ПОЛЕ;
С:      output('Т','С □'); go to НОВОЕ ПОЛЕ;
Л:      output('Т','Л □'); go to НОВОЕ ПОЛЕ;
Ф:      output('Т','Ф □'); go to НОВОЕ ПОЛЕ;

```

---

\* Кроме указанных здесь А. Беллом операторов для выполнения рокировки нужны еще и другие изменения в программе. См. ниже «Свидетельство к алгоритму 50СJ». (Прим. ред.)

\*\* Здесь эти операторы опущены, поскольку в алгоритме 50СJ они вставлены (в переработанном для системы БЭСМ — АЛГОЛ [29] виде) в основную программу, начиная с метки «ВВОД ТАБЛИЦ»: и до метки «НАЧАЛО»: (Прим. ред.)

\*\*\* Процедура ПЕЧАТЬ ПОЗИЦИИ приведена здесь в переработанном для системы БЭСМ — АЛГОЛ [29] и несколько модифицированном виде. (Прим. ред.)



```

КР:      output('T','КР'); go to НОВОЕ ПОЛЕ;
ПОЛЕ СВОБОДНО: output('ZBDB',0);
НОВОЕ ПОЛЕ: end j
      end i;
      output('/')
end ПЕЧАТЬ ПОЗИЦИИ;

```

### Таблица замены в алгоритме 50СJ английских идентификаторов русскими \*

abishop — СЛОНОМ  
 aking — КОРОЛЕМ  
 aknight — КОНЕМ  
 apawn — ПЕШКОЙ  
 aqueen — ФЕРЗЕМ  
 arook — ЛАДЬЕЙ  
 back to — ОБРАТНО К  
 bishop — ТАБЛИЦА СЛОНА  
 black king side — РОКИРОВКА ЧЕРНЫХ  
 black king side castle — КОРОТКАЯ РОКИРОВКА ЧЕРНЫХ  
 blackmen — ДОСКА ЧЕРНЫХ  
 blackpiece — ПОЛЯ ЧЕРНЫХ  
 black queen side — ДЛИННАЯ ЧЕРНЫХ  
 black queen side castle — ДЛИННАЯ РОКИРОВКА ЧЕРНЫХ  
 brawn — ТАБЛИЦА ЧП  
 b1 — Ч1  
 b2 — Ч2  
 can castle bk side — ВОЗМОЖНА КОРОТКАЯ ЧЕРНЫХ  
 can castle bq side — ВОЗМОЖНА ДЛИННАЯ ЧЕРНЫХ  
 can castle wk side — ВОЗМОЖНА КОРОТКАЯ БЕЛЫХ  
 can castle wq side — ВОЗМОЖНА ДЛИННАЯ БЕЛЫХ  
 cant castle bk side — НЕ ВОЗМОЖНА КОРОТКАЯ ЧЕРНЫХ  
 cant castle bq side — НЕ ВОЗМОЖНА ДЛИННАЯ ЧЕРНЫХ  
 cant castle wk side — НЕ ВОЗМОЖНА КОРОТКАЯ БЕЛЫХ  
 cant castle wq side — НЕ ВОЗМОЖНА ДЛИННАЯ БЕЛЫХ  
 castle — ХОДЫ РОКИРОВКИ  
 castling — РОКИРОВКА  
 continue — ПРОДОЛЖЕНИЕ  
 cutoff — ПРЕРЫВАНИЕ  
 depth — УРОВЕНЬ  
 empty — ПОЛЕ СВОБОДНО  
 en passant — НА ПРОХОДЕ  
 entry — НАЧАЛО  
 from — ОТ  
 hismen — ДОСКА ЧУЖИХ  
 hispiece — ПОЛЯ ЧУЖИХ  
 illegal move — НЕЗАКОННЫЙ ХОД  
 kfrom — КР1  
 king — ТАБЛИЦА КОРОЛЯ

---

\* Русские идентификаторы в большинстве случаев не являются здесь результатом буквального перевода заменяемых ими английских идентификаторов, а подобраны редактором выпуска по существу выполняемых ими функций в программе. (Прим. ред.)

knight — ТАБЛИЦА КОНЯ  
 knightorking — ТАБЛИЦА КОНЬКОР  
 knightorkingmove — ХОДЫ КОНЯ И КОРОЛЯ  
 kto — КР2  
 list — ФИГУРА  
 listmoves — ОБЗОР ХОДОВ  
 locate — НОМЕР  
 makemove — ВЫПОЛНЕНИЕ ХОДА  
 moveof — К ХОДАМ  
 moves — СПИСОК  
 mymanin — СВОЯ ДОСКА  
 mymen — ДОСКА СВОИХ  
 mypiece — ПОЛЯ СВОИХ  
 new direction — НОВОЕ НАПРАВЛЕНИЕ  
 nextman — ДАЛЕЕ  
 notstalemat — ШАХ ЧЕРНЫМ  
 numberofmoves — РУБЕЖ ХОДОВ  
 opponents — ЧУЖАЯ ДОСКА  
 pawn — ТАБЛИЦА ПЕШЕК  
 pawncapture — ВЗЯТИЕ ПЕШКОЙ  
 piece — СВОИ ПОЛЯ  
 pointer — p  
 printblack — ЧЕРНАЯ  
 printposition — ПЕЧАТЬ ПОЗИЦИИ  
 reversemove — ВОЗВРАТ ХОДА  
 rfrom — Л1  
 rook — ТАБЛИЦА ЛАДЬИ  
 rook — НОМЕР ЛАДЬИ \*  
 rookorbishop — ТАБЛИЦА СЛОНЛАД  
 rookorbishopmove — ХОДЫ СЛОНА И ЛАДЬИ  
 rto — Л2  
 See Appendix 2 — СМ ДОПОЛНЕНИЕ 2  
 See Appendix 4 — СМ ДОПОЛНЕНИЕ 4  
 See Appendix 4 in order to — СМ ДОПОЛНЕНИЕ 4 ДЛЯ ВЫПОЛНЕНИЯ  
 sp — НОВОЕ ПОЛЕ  
 square — ПОЛЕ  
 startat — РУБЕЖ  
 terminate — КОНЕЦ РОКИРОВКИ  
 THE ANSWER — ОТВЕТ  
 theend — КОНЕЦ  
 to — НА  
 white king side — РОКИРОВКА БЕЛЫХ  
 white king side castle — КОРОТКАЯ РОКИРОВКА БЕЛЫХ  
 whiteorblackpawnmmove — ХОДЫ ПЕШЕК  
 wpawn — ТАБЛИЦА БП  
 white queen side — ДЛИННАЯ БЕЛЫХ  
 white queen side castle — ДЛИННАЯ РОКИРОВКА БЕЛЫХ  
 white men — ДОСКА БЕЛЫХ  
 whitepiece — ПОЛЯ БЕЛЫХ  
 w1 — Б1

---

\* В алгоритме 50 идентификатор *rook* употребляется в разных блоках в двух разных смыслах. (Прим. ред.)

## Свидетельство к алгоритму 50СJ

Алгоритм 50СJ получен из алгоритма 50 А. Белла (Bell A. G. «The Computer Journal», 1970, № 2) в результате следующих изменений:

- 1) внесения нижеперечисленных поправок \*
- 2) замены английских идентификаторов русскими, подобранными по смыслу выполняемых ими функций в программе, и снабжения ряда узловых операторов смысловыми метками, разъясняющими работу программы;
- 3) замены процедур ввода — вывода системы Elliot Algol процедурами ввода — вывода системы БЭСМ — АЛГОЛ [29] {варианта процедур ввода — вывода языка АЛГАМС [41, вып. 3], получившего широкое распространение как в Советском Союзе (например, на машинах БЭСМ-6, М-220, БЭСМ-4, Минск-22 и др.), так и за рубежом}.

Поправки, внесенные редактором данного выпуска в программу алгоритма 50 А. Белла, целесообразно разбить на следующие группы.

- I. Ликвидация пата на первом ходе.
- II. Обеспечение выдачи информации об ошибках в исходной позиции.
- III. Исправление операторов превращения пешки.
- IV. Исправление операторов взятия на проходе.
- V. Исправление операторов рокировки.

### *I. Ликвидация пата на первом ходе*

В программе алгоритма 50 А. Белла не была предусмотрена опасность возникновения пата после первого хода белых. В тех случаях, когда программа алгоритма 50 рассматривала первый ход, приводящий к пату, раньше ключевого хода, приводящего к мату, она выдавала в качестве решения этот патовый ход. Так, при попытке решения нижеследующих десяти задач из газеты «Вечерняя Москва» в первых же двух задачах программа А. Белла выдала в качестве ответа ход, в результате которого на доске возникал пат.

В частности, при попытке решения задачи № 2 из нижеприведенных десяти задач, программа алгоритма 50 выдала в качестве ответа ход Лс7 (пат) вместо правильного ответа Крf5.

Для исправления этой ошибки в программе были сделаны следующие изменения

1. В начале программы добавлено описание

**Boolean ВОЗМОЖЕН ПАТ;**

2. Ниже метки «НАЧАЛО:» в оператор цикла с заголовком

**for Б1:=3 step 1 until РУБЕЖ ХОДОВ[1] do**

вставлен оператор

**ВОЗМОЖЕН ПАТ:= true;**

3. Ниже предыдущего оператора в оператор цикла с заголовком

**for Б2:=РУБЕЖ ХОДОВ[2]+2 step 1 until РУБЕЖ ХОДОВ[3] do**

вставлен оператор

**ВОЗМОЖЕН ПАТ:= false;**

4. Еще ниже вместо строк

**ПЕЧАТЬ ПОЗИЦИИ; ОТВЕТ:**

**goto КОНЕЦ;**

вставлены операторы

**n:=РУБЕЖ ХОДОВ[2];**

**if ВОЗМОЖЕН ПАТ then**

---

\* Позднее была внесена еще одна поправка: после оператора с меткой ОЧИСТКА ДОСОК был добавлен оператор цикла для очистки массива c[1 : 5].

НЕ МАТ ЛИ ЭТО: **begin** ОБЗОР ХОДОВ (ПОЛЯ БЕЛЫХ, ДОСКА БЕЛЫХ,  
 ДОСКА ЧЕРНЫХ, *n*, МАТ1);  
 ЭТО ПАТ НА ПЕРВОМ ХОДЕ: **goto** Б1 ПРОДОЛЖЕНИЕ  
     **end**;  
     output('/', 'Т', 'РЕШЕНИЕ', '/'); **goto** ОТВЕТ;  
 МАТ1: output('/', 'Т', 'МАТ НА ПЕРВОМ ХОДЕ', '/');  
 ОТВЕТ: ПЕЧАТЬ ПОЗИЦИИ;  
 ОДНО РЕШЕНИЕ НАЙДЕНО: **goto** КОНЕЦ;

## II. Выдача информации об ошибках в исходной позиции

Программа А. Белла во всех случаях неправильного задания исходной позиции заканчивала свою работу без выдачи какой-либо информации о характере ошибки в исходной позиции.

Поэтому ради облегчения работы с программой в нее, кроме вышеуказанной строки

МАТ1: output('/', 'Т', 'МАТ НА ПЕРВОМ ХОДЕ', '/');

были внесены следующие операторы.

1. Перед меткой «НАЧАЛО:» были вставлены строки  
 output('Т', 'ИСХОДНАЯ ПОЗИЦИЯ', '/');  
 ПЕЧАТЬ ПОЗИЦИИ;
2. Перед меткой «КОНЕЦ:» были вставлены строки  
 output('/', 'Т', 'ХОДЫ БЕЛЫХ ИСЧЕРПАНЫ', '/');  
**goto** КОНЕЦ;

ОШИБКА: output('Т', 'ПЕРВЫМ ХОДОМ БЕРЕТСЯ КОРОЛЬ ЧЕРНЫХ', '/');

Наличие в программе этих операторов вывода может облегчить использование программы для составления шахматных задач и уже облегчило отладку самой программы. Так, при попытке решить десятую из нижеприведенных задач программа алгоритма напечатала текст

### ХОДЫ БЕЛЫХ ИСЧЕРПАНЫ

Это послужило сигналом для обнаружения еще одной ошибки, описываемой ниже в п. III.

Для облегчения использования программы и ее совершенствования многие узловые операторы были снабжены в алгоритме 50СJ дополнительными смысловыми метками, такими как ЕСТЬ ЛИ ШАХ ЧЕРНЫМ, УШЛИ ОТ ШАХА, ЧЕРНЫЕ БЕРУТ НАЗАД СВОЙ ПЕРВЫЙ ХОД и т. п.

## III. Исправление операторов превращения пешки

Программа алгоритма 50 неправильно работала в задачах, содержащих превращение пешки в фигуру.

Первая ошибка заключалась в следующем. Оператор с меткой «aa1:» (в процедуре ВЫПОЛНЕНИЕ ХОДА) при превращении пешки в фигуру (кроме коня) уменьшает на единицу формальный параметр *p*, являющийся индексом переменных массива СПИСОК. В результате после выполнения обращений к процедуре ВЫПОЛНЕНИЕ ХОДА оказываются уменьшенными на единицу и фактические параметры Ч1, Б2 и Ч2. Поэтому при возвращении хода, сопровождавшегося превращением пешки, фактические значения параметра ОТ процедуры ВОЗВРАТ ХОДА должны быть не СПИСОК[Ч1+1], СПИСОК[Б2+1] или СПИСОК[Ч2+1], как это было в программе А. Белла, а СПИСОК[Ч1+2], СПИСОК[Б2+2] или СПИСОК[Ч2+2] соответственно.

Для исправления этой ошибки были внесены следующие изменения.

1. В процедуре ВЫПОЛНЕНИЕ ХОДА после метки «aa1:» строки  
 if q[УРОВЕНЬ]=1  $\wedge$  (НА>56  $\vee$  НА<9) then

```

begin p:=p-1;
были заменены строками
if abs(q[УРОВЕНЬ])=1  $\wedge$  (НА>56  $\vee$  НА<9) then
begin p:=p-1; q[УРОВЕНЬ]:=-1;
и оператор
if ДОСКА СВОИХ[НА]=2 then p:=p+1
был заменен оператором
if ДОСКА СВОИХ[НА]=2 then
begin p:=p+1; q[УРОВЕНЬ]:=1 end
2. Перед меткой «aa2:» оператор
ДОСКА СВОИХ[НА]:=q[УРОВЕНЬ];
был заменен оператором
ДОСКА СВОИХ[НА]:=abs(q[УРОВЕНЬ]);
3. Оператор
bb1: ДОСКА СВОИХ[-НА]:=q[УРОВЕНЬ];
был заменен оператором
bb1: ДОСКА СВОИХ[-НА]:=abs(q[УРОВЕНЬ]);
4. В процедуре ВОЗВРАТ ХОДА строка
begin ДОСКА СВОИХ[ОБРАТНО К[УРОВЕНЬ]]:=q[УРОВЕНЬ];
была заменена строкой
begin ДОСКА СВОИХ[ОБРАТНО К[УРОВЕНЬ]]:=abs(q[УРОВЕНЬ]);
5. В конце программы перед меткой «Ч2 ПРОДОЛЖЕНИЕ:»
оператор
ВОЗВРАТ ХОДА(СПИСОК[Ч2+1], ДОСКА ЧЕРНЫХ,ДОСКА БЕЛЫХ,
ПОЛЯ ЧЕРНЫХ);
был заменен оператором
ВОЗВРАТ ХОДА(СПИСОК[if q[УРОВЕНЬ]=-1 then Ч2+2 else Ч2+1],
ДОСКА ЧЕРНЫХ,ДОСКА БЕЛЫХ,ПОЛЯ ЧЕРНЫХ);
6. Ниже перед меткой «Б2 ПРОДОЛЖЕНИЕ:» оператор
ВОЗВРАТ ХОДА(СПИСОК[Б2+1], ДОСКА БЕЛЫХ,ДОСКА ЧЕРНЫХ,
ПОЛЯ БЕЛЫХ);
был заменен оператором
ВОЗВРАТ ХОДА(СПИСОК[if q[УРОВЕНЬ]=-1 then Б2+2 else Б2+1],
ДОСКА БЕЛЫХ,ДОСКА ЧЕРНЫХ,ПОЛЯ БЕЛЫХ);
7. Перед меткой «Ч1 ПРОДОЛЖЕНИЕ:» оператор
ВОЗВРАТ ХОДА(СПИСОК[Ч1+1],ДОСКА ЧЕРНЫХ,ДОСКА БЕЛЫХ,
ПОЛЯ ЧЕРНЫХ);
был заменен оператором
ВОЗВРАТ ХОДА(СПИСОК[if q[УРОВЕНЬ]=-1 then Ч1+2 else Ч1+1],
ДОСКА ЧЕРНЫХ,ДОСКА БЕЛЫХ,ПОЛЯ ЧЕРНЫХ);

```

*Вторая ошибка* в операторах превращения пешки в фигуру состояла в том, что в программе А. Белла операторы с метками aa2—aab\* помещались после оператора с меткой aa1 и выполнялись на каждом превращении пешки, в то время, как эти операторы должны выполняться только по одному разу на каждом новом ходе. Четырехкратное выполнение на одном и том же ходе операторов с метками aa3 и aa4 приводило к неверным результатам в тех случаях, когда превращение пешки в фигуру сопровождалось взятием фигуры противника. Так, при попытке решения указанной ниже задачи № 3 типа Валодао-таск из газеты «64» (когда превращение черной пешки h3 в фигуру сопровождалось взятием ладьи) вместо правильного ответа d4 программа напечатала текст.

---

\* В программе алгоритма 50 это метки A2—A6. (Прим. ред.)

## ХОДЫ БЕЛЫХ ИСЧЕРПАНЫ

Это происходило потому, что оператор с меткой *aa4* запоминал в *c[УРОВЕНЬ]* значение взятой ладьи только при первом превращении пешки (в ферзя). При последующих же трех превращениях пешки (в ладью, слона и коня) этот оператор записывал в *c[УРОВЕНЬ]* каждый раз ноль. В результате ладья белых исчезала с доски навсегда и не восстанавливалась уже перед следующей пробой хода черными.

Для исправления этой ошибки в процедуре **ВЫПОЛНЕНИЕ ХОДА** операторы с метками *aa2—aab* были помещены перед оператором с меткой *aal*, а перед ними был вставлен условный оператор

```
if q[УРОВЕНЬ]=—1 then goto aal;
```

### IV. Исправление операторов взятия на проходе

При попытке решения задач со взятием пешки на проходе с помощью программы А. Белла (после подстановки в нее операторов с метками *ep1*, *ep2*, *ep3* и *ep4*, описанных в разделе «Взятие пешки на проходе») было обнаружено, что операторы с метками *ep3* и *ep4* опубликованы в алгоритме 50 неправильно. Эти операторы должны в действительности иметь следующий вид.

```
ep3: if q[УРОВЕНЬ]=1 ∧ НА= НА ПРОХОДЕ[УРОВЕНЬ—1] then
  begin c[УРОВЕНЬ]:=—1;
    ДОСКА ЧУЖИХ[if НА<32 then НА+8 else НА—8]:=0
  end;
ep4: if c[УРОВЕНЬ]=—1 then
  begin ДОСКА ЧУЖИХ[ОТ]:=0;
    ДОСКА ЧУЖИХ[if ОТ<32 then ОТ+8 else ОТ—8]:=1
  end else
  ДОСКА ЧУЖИХ[ОТ]:=c[УРОВЕНЬ];
```

Полезно также заметить, что операторы раздела «Взятие пешки на проходе» вставляются в программу следующим образом:

- 1) операторы с меткой «*ep1*:» — на место метки «*ep1*:»;
- 2) операторы с меткой «*ep2*:» — на место строки

```
ep2: if ЧУЖАЯ ДОСКА[ТАБЛИЦА ПЕШЕК[i]] = 0 then
```

- 3) операторы с меткой «*ep3*:» — на место метки «*ep3*:»;
- 4) операторы с меткой «*ep4*:» в процедуру **ВЫПОЛНЕНИЕ**

**ХОДА** на место строк

```
ep4:
bb3: ДОСКА ЧУЖИХ[ОТ]:=c[УРОВЕНЬ];
а в процедуру ВОЗВРАТ ХОДА — на место строки
ep4: ДОСКА ЧУЖИХ[ОТ]:=c[УРОВЕНЬ];
```

### V. Исправление операторов рокировки

Первая ошибка в операторах рокировки состояла в том, что в процедуре **ВОЗВРАТ ХОДА** алгоритма 50 А. Белла не были предусмотрены операторы возврата короля в исходное положение, если возвращаемый ход был рокировкой. В результате программа алгоритма 50 работала правильно тогда, когда рокировка выполнялась на первом ходе белых или тогда, когда рокировка не сопровождалась матом королю противника. Если же рокировка выполнялась на втором ходе белых (Б2) и сопровождалась матом, то черные должны были взять назад свой первый ход (Ч1), чтобы испытать другую защиту от первого хода белых. Но для этого должна была выполняться процедура **ВОЗВРАТ ХОДА** для возврата второго хода белых (Б2). Поскольку при этом возвращаемый ход белых был рокировкой, то процедура **ВОЗВРАТ ХОДА** не мог-

ла восстановить исходное положение на доске и оставляла рокированного короля на новом его месте, возвращая в исходное положение только ладью.

Так, например, программа алгоритма 50 не дала правильного решения в указанной ниже задаче № 6 типа Валодао-таск из еженедельника «64».

Для исправления этой ошибки в ведущую часть (после метки «НАЧАЛО:») программы алгоритма 50СJ перед оператором **goto Ч1 ПРОДОЛЖЕНИЕ**; был вставлен пустой оператор **ВОЗВРАТ КОРОЛЯ БЕЛЫХ: СМ СВИДЕТЕЛЬСТВО К АЛГОРИТМУ 50СJ;** а перед оператором **goto Б1 ПРОДОЛЖЕНИЕ**; вставлен оператор **ВОЗВРАТ КОРОЛЯ ЧЕРНЫХ: СМ СВИДЕТЕЛЬСТВО К АЛГОРИТМУ 50СJ;**

В случае использования данной программы для решения задач с рокировкой в начало программы (например, после описания процедуры **ВОЗВРАТ ХОДА**) нужно вставить описание

```
procedure ВОЗВРАТ КОРОЛЯ (p,НА,ПОЛЯ,ДОСКА);  
integer p,НА; integer array ПОЛЯ,ДОСКА;  
if СПИСОК[p+2]<0  $\wedge$  СПИСОК[p+6]=—НА then  
  begin ПОЛЯ[1]:=НА; ДОСКА[НА]:=6;  
    ДОСКА[if СПИСОК[p+2]=—(НА+3) then НА+2 else НА—2]:=0  
  end ВОЗВРАТ КОРОЛЯ;
```

Первый из вышеуказанных пустых операторов нужно заменить обращением

**ВОЗВРАТ КОРОЛЯ (Б2,5,ПОЛЯ БЕЛЫХ,ДОСКА БЕЛЫХ);**

а второй — обращением

**ВОЗВРАТ КОРОЛЯ (Ч1,61,ПОЛЯ ЧЕРНЫХ,ДОСКА ЧЕРНЫХ);**

*Вторая ошибка* в операторах рокировки алгоритма 50 состояла в том, что в отрывке программы, приведенном в дополнении 2 к алгоритму 50СJ, запрет рокировки не был предусмотрен для тех случаев, когда между королем и ладьей имеются фигуры противника. Для исправления этой ошибки оператор

**РОКИРОВКА БЕЛЫХ: if  $\neg$  КОРОТКАЯ РОКИРОВКА БЕЛЫХ  $\vee$**

**ДОСКА БЕЛЫХ[6] $\neq$ 0  $\vee$  ДОСКА БЕЛЫХ[7] $\neq$ 0 then**

**goto ДЛИННАЯ БЕЛЫХ;**

был заменен в алгоритме 50СJ оператором

**РОКИРОВКА БЕЛЫХ: if  $\neg$  КОРОТКАЯ РОКИРОВКА БЕЛЫХ  $\vee$**

**ДОСКА БЕЛЫХ[6] $\neq$ 0  $\vee$  ДОСКА БЕЛЫХ[7] $\neq$ 0  $\vee$**

**ДОСКА ЧЕРНЫХ[6] $\neq$ 0  $\vee$  ДОСКА ЧЕРНЫХ[7] $\neq$ 0 then**

**goto ДЛИННАЯ БЕЛЫХ;**

Аналогичные исправления были сделаны для операторов с метками **ДЛИННАЯ БЕЛЫХ**, **РОКИРОВКА ЧЕРНЫХ** и **ДЛИННАЯ ЧЕРНЫХ**.

*Третья ошибка* в операторах рокировки алгоритма 50 состояла в том, что в вышеуказанном отрывке программы перед обращением

**ОБЗОР ХОДОВ (ПОЛЯ БЕЛЫХ,ДОСКА БЕЛЫХ,ДОСКА ЧЕРНЫХ,p,**

**НЕ ВОЗМОЖНА КОРОТКАЯ ЧЕРНЫХ);**

был пропущен оператор

**ДОСКА ЧЕРНЫХ[62]:=ДОСКА ЧЕРНЫХ[63]:=6;**

## *VI. Результаты отладки алгоритма 50СJ*

После внесения вышеуказанных исправлений программа алгоритма 50СJ была транслирована на машине БЭСМ-6 в системе БЭСМ—АЛГОЛ в трех вариантах:

А) без операторов взятия пешки на проходе и без операторов рокировки;

Б) с операторами взятия на проходе, но без операторов рокировки;

В) с операторами взятия на проходе и с операторами рокировки

и с ее помощью, кроме двух задач, приведенных А. Беллом, были решены еще и все 10 задач на мат в два хода, публиковавшиеся в газете «Вечерняя Москва» с декабря 1972 г. по февраль 1973 г. под рубрикой «Конкурс — 26». Эти задачи следующие.

1. Белые: Крд6, Фf3, Лс2.  
Черные: Крд4.
2. Белые: Кре4, Фg4, Лс8, Кd5.  
Черные: Крд6, g5.
3. Белые: Крг2, Фf1, Лd8, Ch2, a4.  
Черные: Кrb7, Ca7.
4. Белые: Кре3, Фе4, Ле7.  
Черные: Кре1, Лb8, Ca8, c3.
5. Белые: Кpf2, Фf6, Ch7.  
Черные: Крг4, g5.
6. Белые: Крд2, Фс6, Лb5.  
Черные: Крд4, Лd8, Кf7.
7. Белые: Кrb1, Фd3, Ла8, Лс6, e6.  
Черные: Кrb7, Кb8.
8. Белые: Кpf8, Фb4, Ла5, e5.  
Черные: Крд7, Сс8, с7.
9. Белые: Кра2, Фb2, Лb5, Ca7.  
Черные: Кра6, Лh7, g6.
10. Белые: Кра8, Фе5, Сс8, d7, h5.  
Черные: Кpf7, h6.

Время (в секундах), затраченное на решение этих десяти задач (всеми тремя вариантами программы) и на исследование их однозначности, приведено в табл. 29.

Для проведения исследования задачи на однозначность решения достаточно удалить из программы строку

ОДНО РЕШЕНИЕ НАЙДЕНО: goto КОНЕЦ;

после чего программа печатает все решения, возможные в данной задаче, и затем текст ХОДЫ БЕЛЫХ ИСЧЕРПАНЫ.

Т а б л и ц а 29

Номер задачи	Ключевой ход	Время решения [с]			Время исследования [с]
		Вариант А	Вариант Б	Вариант В	
1	Фh3	7.10	7.12	7.54	38.38
2	Кpf5	40.32	41.06	44.04	44.56
3	Ла8	31.14	31.82	34.00	140.54
4	Лb7	38.84	39.56	42.56	209.24
5	Фh8	20.02	20.48	22.00	34.70
6	Лf5	115.00	118.02	126.00	201.68
7	Лс8	30.00	30.62	32.74	108.98
8	Фb8	39.90	41.02	43.84	56.06
9	Лb5	102.36	104.42	111.28	261.26
10	Фс7	24.76	25.50	27.44	30.92
Средние значения		44.94	45.96	49.14	112.63

Поскольку время решения задачи данной программой существенно зависит от того, в какой последовательности вводятся в память машины значения фигур и занимаемых ими полей, то при решении вышеуказанных задач строго выдерживался один и тот же порядок: фигуры вводились в порядке возрастания номеров, занимаемых ими полей. В общем случае время, затрачиваемое на решение задач, не может служить характеристикой их сложности. Такой характеристикой скорее может служить машинное время, затрачиваемое на исследование однозначности решения.



Таблица 29 показывает, что после вставления в программу операторов, осуществляющих взятие пешки на проходе, быстродействие программы снижается несущественно (примерно на 2%). Более существенно (еще примерно на 7%) возрастает время работы программы после вставления в нее операторов, осуществляющих рокировку. Следует заметить, что с операторами рокировки процедура ОБЗОР ХОДОВ превращается в рекурсивную, а следовательно, может использоваться далеко не на каждом из современных трансляторов.

Для проверки работы программы в специальных случаях были решены все шесть задач на мат в два хода, опубликованные в еженедельнике «64» № 21 (256) от 25—31 мая 1973 г. в статье «Таких три разных хода». Эти задачи, содержащие одновременно превращение пешки, взятие на проходе и рокировку (так называемые задачи типа Валодуа-таск), следующие.

1. Белые: Kpe1, Ла1, Cd6, Ке2, Ке3, а7, d3, g2, h7.  
Черные: Kph1, Cb1, c3, g5, h4.
2. Белые: Kpe1, Lh1, Cc4, Ch8, Kd2, Kd3, а7, b2, e5.  
Черные: Kpa1, Kg3, а4, b7, d7, f5, h5.
3. Белые: Kpe1, Фе8, Lh1, Cc1, Kg2, Kh6.  
Черные: Kpf3, Ла4, Ca6, e4, g3, h3.
4. Белые: Kpe1, Ла1, Ла7, Ch1, Ch2, Kc3, Kg6, b4, d5, d7, f5.  
Черные: Kpd6, Lf4, Lh8, Ca5, а6, b6, e7, g5.
5. Белые: Kpe1, Лг2, Lh1, Ch7, Ch8, f5.  
Черные: Kpb1, Cc1, а2, e7.
6. Белые: Kpe1, Фс2, Lh1, Lh8, а5, d2, f2, h7.  
Черные: Kpa1, Ch2, b7, f3.

Время, затраченное на решение этих задач и на исследование их однозначности, приведено в табл. 30.

Таблица 30

Номер задачи	Ключевой ход	Время (с)	
		решения	исследования
1	g4	46.72	73.34
2	b4	70.22	83.64
3	d4	159.02	180.02
4	0—0—0	178.64	179.72
5	0—0	99.56	100.34
6	Ла8	35.40	77.50
Средние значения		98.26	115.76

В том виде, как она здесь приведена, программа алгоритма 50CJ в качестве результата решения печатает расположение всех фигур на доске после выполнения ключевого хода. Например, для вышеуказанной задачи № 4 из газеты «64» программа напечатала следующее:

РЕШЕНИЕ

0	0	0	0	0	0	0	—Л
+Л	0	0	+П	—П	0	0	0
—П	—П	0	—КР	0	0	+К	0
—С	0	0	+П	0	+П	—П	0
0	+П	0	0	0	—Л	0	0
0	0	+К	0	0	0	0	0
0	0	0	0	0	0	0	+С
0	0	+КР	+Л	0	0	0	+С

Во всех вышеуказанных случаях использования алгоритма 50СJ указывается время его работы, которое выдавалось на печать вариантом транслятора БЭСМ — АЛГОЛ от 10.11.71 с диспетчером версии 17.9.71. Сюда включается и время подготовки к печати позиции, которое для решения одиночной задачи равно примерно 1 с.

### Замечание к алгоритму 50

Дж. Бери (Berry J. L. «The Computer Journal», 1971, № 1) \*

В отношении алгоритма 50 (данный журнал, т. 13, с. 208—219) мне хотелось бы указать на то, что авторская «Позиция Белла» (с. 211), представленная в качестве простейшей из известных позиций на мат в два хода, не является фактически законной шахматной позицией, т. е. она не может встретиться в игре за шахматной доской.

Я построил более простую позицию, которая дает форсированный мат в два хода тем же способом, что и позиция Белла, и мой пример может встретиться как результат обычных шахматных ходов. Эта позиция показана на рис. 7, а ниже (табл. 31) приведена примерная партия, приводящая к этой позиции (нет сомнения, что возможны и более короткие партии такого рода). Эта партия дает позицию, в которой у белых нет другого выбора, как только сделать мат черному королю серией трех форсированных полуходов, т. е. использует ту же идею, что и позиция Белла (см. рис. 7).

Черные							
					ЧС	ЧЛ	ЧКр
				ЧП		ЧП	
				БП		БП	
					ЧП		
					ЧП	ЧП	
				ЧП		БП	
				БП		БП	БС
					БС	БКр	БЛ
Белые							

Рис. 7.

Таблица 31

Номер хода	Белые	Черные	Номер хода	Белые	Черные
1	b3	Kf6	19	fg	Ca6
2	Cb2	Ke4	20	Kpf2	Ke5
3	Ce5	f5	21	Kpg1	Cd3
4	h4	Kg5	22	cd	Фа8
5	hg	Kpf7	23	d4	Фс6
6	Ch2	a5	24	de	ab
7	Kf3	Ла6	25	Ла2	ba
8	Kc3	h5	26	Л: a3	Фb7
9	a3	Kpg6	27	d4	d6
10	Ke5+	Kph7	28	d5	de
11	Kg4	hg	29	Kb5	Фb8
12	b4	Ле6	30	Лf3	c5
13	Фb1	Kc6	31	Лf4	ef
14	Фb2	b6	32	Kd4	cd
15	Фb1	Ле3	33	Ф: b6	Фа8
16	Фb2	Лg8	34	Фb3	Фс8
17	g6+	Kph8	35	Фe3	Фe6
18	Фb1	Лg3	36	de	de

\* Это замечание было опубликовано в форме письма редактору журнала «The Computer Journal». (Прим. ред.)

## Ответ автора алгоритма 50 на замечание Дж. Бери

Данный пример был предназначен для того, чтобы показать некоторые фактические генерируемые числа и пути, выбираемые программой. Позиция была незаконной из-за расположения пешек, которое Бери исправил, удалив две (лишние) средние пешки на королевской вертикали е.

Я охарактеризовал эту позицию как «простейшую». Под этим словом я понимал позицию, для которой разветвления ходов минимальное, т. е. ограничивается одним и только одним генерируемым (законным или незаконным) ходом на каждом уровне. Позиция Бери допускает незаконные ходы как белого, так и черного короля, и в случае задания этой позиции программа потребовала бы несколько больше времени для доказательства единственного (П:П) решения.

В связи с этим, если уж игнорировать незаконные ходы короля и допускать превращение пешки в фигуру, то позиция

ЧКр				
ЧП				
БКр	ЧП		БП	
	БП			

дает форсированную последовательность ходов

- 1) d7 Kpb8
- 2) d8(Ф)×

## ПРИЛОЖЕНИЕ 2

### Подтверждения и замечания к алгоритмам, опубликованным в предыдущих выпусках

#### Подтверждения к алгоритмам 157а, 160а, 170а и 190а

З. А. Шиншинова, Москва, июль 1973

Вышеперечисленные алгоритмы были вновь транслированы на машине БЭСМ-6 в системе БЭСМ-АЛГОЛ с примерами, приведенными в соответствующих свидетельствах, и дали верные результаты. В алгоритм 170а предварительно были внесены поправки, предложенные В. Д. Сычевым и Ю. И. Марковым в их «Замечании к алгоритму 170а» [38].

#### Подтверждение к алгоритму 160а

Б. Л. Шмульян, Москва, июнь 1972

Алгоритм 160а был модифицирован для получения функции

$$f(m, n) = \begin{cases} C^m_n & \text{при } n \leq m, \\ 0 & \text{при } n > m. \end{cases}$$

Для этого был помечен последний оператор процедуры, который принял вид  
m1: comb:=r

Первым оператором в теле процедуры был поставлен следующий:

if n>m then begin r:=0; go to m1 end;

Алгоритм был транслирован с языка АЛГОЛ в системе 4—70 и дал правильные результаты при  $m=3, 4, \dots, 15$  и  $n=1, 2, \dots, 5$ .

## Замечание к алгоритму 176

Э. Шуграф (Schuegraf E. «САСМ», 1972, № 12)

Алгоритм 176 содержит одну опечатку \*. Строка, имеющая вид

**begin**  $c[i]=y[i]$ ;

должна быть

**begin**  $c[i]:=z[i]$ ;

## Замечание к алгоритму 195

Э. Шуграф (Schuegraf E. «САСМ», 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. 32.

Т а б л и ц а 32

$n$	$m$			
	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  $((m-1) \div \div 2) \times n$  (обратите внимание на определение  $m$ ). При проверке результатов было обнаружено, что для вырожденных и близких к вырожденным матриц алгоритм дает неверные решения. Это 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** (с,n,m,eps,signal) dataresult:(v);

**value** n,m; **real** eps; **integer** n,m; **label** signal; **array** c,v;

После оператора

**if**  $abs(c[r,l]) > abs(c[piv,l])$  **then**  $piv:=r$ ;

вставить оператор

**if**  $abs(c[piv,l]) < eps$  **then go to** signal;

## Подтверждение к алгоритму 210a

Б. Л. Шмульян, Москва, июнь 1972.

Алгоритм 210a был переведен на язык ФОРТРАН-IV (с соответствующими изменениями), транслирован в системе 4—70 и использован при  $n=4$  для построения графиков функций по дискретным значениям (с помощью машинного графопостроителя). Процедура на ФОРТРАНЕ-IV имела вид

SUBROUTINE LAGR(N,U,A,X,Y)

C ALGORITHM 210A

DIMENSION X(1),Y(1)

---

\* Эта опечатка была уже замечена авторами выпуска «Алгоритмы (151—200)» и исправлена в алгоритме 176a. Следует заметить, что в данном замечании Э. Шуграфа ничего не говорится о первой поправке к алгоритму 176, указанной в «Свидетельстве к алгоритму 176a» [25]. (Прим. ред.)

\*\* Здесь используются обозначения алгоритма 195a. В алгоритме 195 эти буквы были прописными. (Прим. ред.)

```

A=0.0
N1=N+1
DO 20 J=1,N1
R=1.0
DO 10 I=1,N1
10 IF (I.NE.J) R=R*(U-X(I))/(X(J)-X(I))
20 A=A+R*Y(J)
RETURN
END

```

### Подтверждение к алгоритму 245

#### «Доказательство алгоритмов — новый вид подтверждения»

Р. Лондон (London R. L. «CACM», 1970, № 6)

#### Аннотация

Подтверждение алгоритма может принимать форму доказательства правильности алгоритма. В качестве иллюстративного, но имеющего практическое значение примера этого утверждения, приведено доказательство того, что алгоритм 245 является правильным.

#### Подтверждение алгоритмов путем доказательства

Поскольку теперь существуют соответствующие технические приемы доказательства правильности многих алгоритмов (см., например, [18i—22i]), то можно и уместно подтверждать алгоритмы с помощью таких доказательств. Подтверждение путем апробирования по-прежнему остается полезным, так как оно проще и дает также, например, и сведения о затратах времени. Однако наличие доказательства могло бы быть желательным дополнительным подтверждением алгоритма. Доказательство убедительно показывает, что алгоритм отлажен, поскольку в нем нет никаких ошибок.

Не важно, захотят и смогут ли все пользователи алгоритма проверять его доказательство, иногда довольно громоздкое. Ни от кого не требуется, чтобы перед использованием алгоритмом он согласился с доказательством, точно так же как ни от кого не ожидают, что он станет вновь проверять на машине тесты подтверждения. В обоих случаях можно полагаться, по крайней мере частично, на автора и рецензента.

В качестве примера подтверждения алгоритма здесь доказываем, что алгоритм TREESORT 3\* правильно выполняет свою задачу сортировки массива  $M[1:n]$  в порядке возрастания. Данный алгоритм был уже раньше подтвержден («CACM», 1965, № 7), но в этом подтверждении, например, не была сделана проверка для массивов нечетной длины. Поскольку TREESORT 3 является быстрым алгоритмом сортировки массива на месте его записи, широко применяемым на практике и достаточно сложным, так что его правильность не самоочевидна, то использование TREESORT 3 в качестве примера представляет нечто большее, чем абстрактное упражнение. Это пример большой практической важности.

#### Краткое описание алгоритма 245 и метод доказательства

Работу алгоритма легче всего проследить, если его рассматривать как бинарное дерево. Элемент массива  $M[k \div 2]$  будет считаться родителем элемента  $M[k]$  для  $2 \leq k \leq n$ . Другими словами, дети элемента  $M[j]$  — это  $M[2j]$  и  $M[2j+1]$  при условии, что по крайней мере один из детей существует.

---

\* Т. е. процедура алгоритма 245. Соответствующая процедура алгоритма 245а имеет идентификатор *treesort*. (Прим. ред.)

Первая часть алгоритма переставляет элементы массива  $M$ , так чтобы для некоторого сегмента этого массива любой родитель стал больше любого из своих детей. Каждое выполнение вспомогательной процедуры *siftup* расширяет этот сегмент, вызывая появление еще одного родителя, доминирующего над своими детьми. Вторая часть алгоритма использует процедуру *siftup* для того, чтобы сделать родителей превосходящими своих детей по всему массиву, меняет значениями  $M[1]$  с последним элементом и повторяет указанные действия над массивом, укороченным на один элемент. Все вышесказанное является мотивировкой, а не частью формального доказательства.

Доказательство правильности алгоритма 245 состоит из трех частей. Сначала показывается, что процедура *siftup* выполняется так, как это формально определено ниже. Затем показывается, что тело процедуры TREESORT 3, использующее процедуру *siftup* двумя способами, сортирует массив в порядке возрастания (доказательство процедуры *exchange* опущено). Доказательство проводится по методу, описанному в работах [18i, 19i, 22i]: между строками программ вставляются утверждения, касающиеся хода вычисления, и доказательство заключается в демонстрации того, что каждое утверждение истинно в любой момент, когда выполнение процедуры достигает этого утверждения, в предположении, что ранее встречавшиеся утверждения истинны. В заключение отдельно показывается, что процедура всегда заканчивается.

Нумеруются строки доказуемого алгоритма и соответственно утверждения, записанные в форме комментариев программы. Нумерация используется только для ссылок и никакого другого значения не имеет. Дополнительная пара скобок **begin** — **end** была вставлена в тело процедуры TREESORT 3 для того, чтобы можно было различать моменты выполнения двух утверждений (3.1 и 4.1). В процедуре *siftup* правильный результат выражают утверждения 10.1 и 10.2; в теле процедуры TREESORT 3 то же выражают утверждения 9.3 и 9.4.

### Определение процедуры *siftup* и система обозначений

Определим формально процедуру *siftup*( $i, n$ ) так, что в ней  $n$  будет означать просто некоторый формальный параметр, а не длину массива  $M^*$ . Пусть  $A(s)$  означает последовательность неравенств  $M[k \div 2] \geq M[k]$  для  $2s \leq k \leq n$  (если  $s > n \div 2$ , то  $A(s)$  — пустая последовательность). Если перед обращением к процедуре *siftup*( $i, n$ ) выполняется  $A(i+1)$  и если  $1 \leq i \leq n \leq$  размер массива, то после выполнения обращения *siftup*( $i, n$ )

- 1) будет выполняться  $A(i)$ ,
- 2) элементы сегмента массива от  $M[i]$  до  $M[n]$  будут переставлены,
- 3) сегмент массива вне интервала от  $M[i]$  до  $M[n]$  останется неизменным.

Для доказательства этих свойств процедуры *siftup* необходимо ввести некоторые обозначения. Формальный параметр  $i$  внутри процедуры *siftup* будет меняться. Поскольку  $i$  вызывается по значению, то вне *siftup* такого изменения наблюдаться не будет. Тем не менее в доказательстве процедуры *siftup* необходимо наравне с текущим значением  $i$  пользоваться и его начальным значением.

Пусть  $i_0$  будет значением  $i$  перед входом в процедуру *siftup*. Аналогично пусть  $M_0$  будет значением массива  $M$  перед входом в процедуру *siftup*. Утверждение « $M = p(M_0)$  при  $M := \text{copy}$ » означает «Если выполнено  $M[i] := \text{copy}$ , то  $M$  является перестановкой  $M_0$ , как это описано в п. 2) и 3) определения процедуры *siftup*». Утверждение « $M = p(M_0)$ » означает то же самое, только без ссылки на выполнение  $M[i] := \text{copy}$ .

\* В алгоритме 245а некоторые обозначения изменены, а именно  $M$  заменено на  $m$ ,  $loop$  — на  $iter$ . (Прим. ред.)

\*\* По-видимому, здесь Р. Лондон имел в виду фактический параметр, соответствующий формальному параметру  $i$ . (Прим. ред.)

*Код процедуры siftup с утверждениями*

```

0  procedure siftup(i,n); value i,n; integer i,n;
1  begin real copy; integer j;
    comment
        1.1:  $1 \leq i_0 = i \leq n \leq \text{размер массива}$ 
        1.2:  $A(i_0+1)$ 
        1.3:  $M = p(M_0)$ ;
2      copy := M[i];
3  loop: j := 2 × i;
    comment
        3.1:  $i \leq n$ 
        3.2:  $2i = j$ 
        3.3:  $i = i_0$  или  $i \geq 2i_0$ 
        3.4:  $M = p(M_0)$  при  $M[i] = \text{copy}$ 
        3.5:  $A(i_0)$  или  $(i = i_0 \text{ и } A(i_0+1))$ 
        3.6:  $M[i \div 2] > \text{copy}$  или  $i = i_0$ 
        3.7:  $M[i \div 2] \geq M[i]$  или  $i = i_0$ ;
4  if j ≤ n then
5      begin if j < n then
6a      begin if M[j+1] > M[j] then
6b      j := j+1 end;
        comment
            6.1:  $i = j \div 2$ 
            6.2:  $2i \leq j \leq n$ 
            6.3:  $i = i_0$  или  $i \geq 2i_0$ 
            6.4:  $M = p(M_0)$  при  $M[i] = \text{copy}$ 
            6.5:  $A(i_0)$  или  $(i = i_0 \text{ и } A(i_0+1))$ 
            6.6:  $M[i \div 2] > \text{copy}$  или  $i = i_0$ 
            6.7:  $M[i \div 2] \geq M[i]$  или  $i = i_0$ 
            6.8:  $(2i < n \text{ и } M[j] = \max(M[2i], M[2i+1]))$  или
                 $(2i = n \text{ и } M[j] = M[n])$ 
            6.9:  $M[i] \geq M[j]$  или  $i = i_0$ ;
7      if M[j] > copy then
8a      begin M[i] := M[j];
        comment
            8.1:  $i = i_0$  или  $i \geq 2i_0$ 
            8.2:  $2i \leq j \leq n$ 
            8.3:  $M[j \div 2] = M[i] = M[j] > \text{copy}$ 
            8.4:  $M[i \div 2] \geq M[j]$  или  $i = i_0$ 
            8.5:  $M = p(M_0)$  при  $M[j] = \text{copy}$ 
            8.6:  $A(i_0)$ ;
8b      i := j;
        comment
            8.7:  $i \geq 2i_0$ 
            8.8:  $i = j \leq n$ 
            8.9:  $M[i \div 2] > \text{copy}$ 
            8.10:  $M[i \div 2] \geq M[i]$ 
            8.11:  $M = p(M_0)$  при  $M[i] = \text{copy}$ 
            8.12:  $A(i_0)$ ;
8c      go to loop end
9      end;

```

```

      comment
      9.1:  $M[j] \leq \text{сору}$ , если пришли от строки 7, или
            $2i = j > n$ , если пришли от 4;
10       $M[i] := \text{сору}$ ;
      comment
      10.1:  $M = p(M_0)$ 
      10.2:  $A(i_0)$ ;
11      end siftup;

```

### Обоснование утверждений процедуры *siftup*

Справедливость вышеуказанных утверждений основывается на следующих соображениях.

1.1—1.2: Утверждения, необходимые для выполнения процедуры *siftup*.

1.3: здесь  $p$  — это тождественная перестановка.

3.1—3.7: Если пришли от строки 2, то

3.1: 1.1.

3.2: 3.

3.3, 3.5—3.7:  $i = i_0$  по 1.1. 3.5 требует и 1.2.

3.4: 1.3 и 2.

Если же пришли от строки 8, то соответственно 8.8, 3, 8.7, 8.11, 8.12, 8.9 и 8.10.

6.1: В 3.2  $j = 2i$ , а по 6b  $j$  может быть равно  $2i+1$ . В обоих случаях  $i = j \div 2$ .

6.2: После 4  $j \leq n$ . На участке от 3.1 до 6.2  $j$  меняется только в 6b.

Перед 6b  $j < n$  согласно 5. Следовательно в 6.2  $j \leq n$ .  $2i \leq j$  по 6.1.

6.3—6.7: 3.3—3.7 соответственно.

6.8: Если 4 есть **true**, а 5 — **false**, то  $j = 2i = n$  (согласно 3.2), так что второй пункт утверждения 6.8 выполняется.

Если 4 — **true** и 5 — **true**, то в 6a  $2i = j < n$  (согласно 3.2), так что равенство  $M[j+1] = M[2i+1]$  установлено. Теперь в 6.8  $j = 2i$  или  $j = 2i+1$ . В любом случае по 6a и 6b первый пункт утверждения 6.8 выполняется.

6.9: По 6.5  $i \neq i_0$  дает  $A(i_0)$ .  $2i_0 \leq 2i \leq j \leq n$  по 6.3 и 6.2.

Следовательно,  $A(i_0)$  и 6.1 дают  $M[i] = M[j \div 2] \geq M[j]$ .

8.1: 6.3.

8.2: 6.2.

8.3:  $i = j \div 2$  по 6.1,  $M[i] = M[j]$  по 8a и  $M[j] > \text{сору}$  по 7.

8.4: 6.7 и 6.9.

8.5: 6.4 требует, чтобы значение  $M[i]$  было заменено значением  $\text{сору}$ . Поскольку (согласно 8a)  $M[i] = M[j]$ , то  $M[j]$  может с таким же успехом быть заменено на  $\text{сору}$ .

8.1 и 8.2 дают  $i_0 \leq i \leq n$ , так что изменение массива  $M$  в 8a производится внутри сегмента от  $M[i_0]$  до  $M[n]$ .

8.6: Согласно 8a и если выполняется 6.8 (первый пункт), то  $M[i] \geq M[2i]$  и  $M[i] \geq M[2i+1]$ . Согласно 8a и если выполняется 6.8 (второй пункт), то  $M[i] = M[j] = M[n] = M[2i]$ , и для данного вызова процедуры *siftup*  $M[2i+1]$  не существует. В 6.5  $A(i_0+1)$  выполняется, поскольку  $A(i_0)$  включает в себя и  $A(i_0+1)$ . Если  $i = i_0$ , то выполнение  $A(i_0+1)$  и вышеуказанные соотношения для  $M[i]$  дают выполнение  $A(i_0)$ . Если  $i \neq i_0$ , то 8a, 8.4, выполнение  $A(i_0)$  в 6.5 и вышеуказанные соотношения для  $M[i]$  дают выполнение  $A(i_0)$  в 8.6.

8.7: 8b, 8.1 и 8.2.

8.8: 8b и 8.2.



8.9: 8b и 8.3.  
8.10: В 8.6  $2i_0 \leq j \leq n$  по 8.1 и 8.2. Следовательно, по 8.6  
 $M[j \div 2] \geq M[j]$ . Для соотношения  $M[j \div 2] \geq M[j]$  используется 8b.  
8.11: 8b и 8.5.  
8.12: 8.6.  
9.1: В 9.1 приходим только, если 7 есть false или если 4 есть false.  $2i = j$  по 3.2.  
10.1—10.2: Если пришли от 7, то  
10.1: 6.4 и 10. (6.2 и 6.3 дают  $i_0 \leq i \leq n$ , обеспечивая то, что изменение массива  $M$  в 10 будет происходить внутри сегмента от  $M[i_0]$  до  $M[n]$ .)  
10.2: Согласно 10, 9.1, 6.2 и 6.8  $M[i] = \text{сору} \geq M[j] \geq M[2i]$   
и если  $M[2i+1]$  существует, то  $M[j] \geq M[2i+1]$ . Если  $i = i_0$ , то выполнение 10.2 следует так же, как и в утверждении 8.6. Если же  $i \neq i_0$ , то 6.6 и 10 дают  $M[i \div 2] > \text{сору} = M[i]$ . Теперь выполнение  $A(i_0)$  в 6.5 дает выполнение  $A(i_0)$  и в 10.2.  
Если пришли от 4, то  
10.1: 3.4 и 10. (3.1 и 3.3 дают  $i_0 \leq i \leq n$ )  
10.2:  $2i > n$  означает, что в  $A(i_0)$  нет соотношений, имеющих форму  $M[i] \geq \dots$ . Если  $i = i_0$ , то 3.5 дает 10.2. Если же  $i \neq i_0$ , то 3.6 и 10 дают  $M[i \div 2] > \text{сору} = M[i]$ . Теперь  $A(i_0)$  в 3.5 дает 10.2.

### *Код тела процедуры TREESORT 3 с утверждениями*

```

0  integer i;
   comment
     0.1:  $A(n \div 2 + 1)$ ;
1  for i := n ÷ 2 step -1 until 2 do
2    begin
       comment
         2.1:  $A(i+1)$ 
         2.2: Условия для процедуры siftup выполняются;
3    siftup(i,n);
       comment
         3.1:  $A(i)$ ;
4    end;
   comment
     4.1:  $M[p] \leq M[p+1]$  для  $n+1 \leq p \leq n-1$ 
     4.2:  $A(2)$ , т. е.  $M[k \div 2] \geq M[k]$  для  $4 \leq k \leq n$ ;
5  for i := n step -1 until 2 do
6    begin
       comment
         6.1:  $M[p] \leq M[p+1]$  для  $i+1 \leq p \leq n-1$ 
         6.2.:  $M[k \div 2] \geq M[k]$  для  $4 \leq k \leq i$ 
         6.3:  $M[i+1] \geq M[r]$  для  $1 \leq r \leq i$ 
         6.4: Условия для процедуры siftup выполняются;
7    siftup(1,i);
       comment
         7.1:  $M[p] \leq M[p+1]$  для  $i+1 \leq p \leq n-1$ 
         7.2:  $M[k \div 2] \geq M[k]$  для  $2 \leq k \leq i$ 
         7.3:  $M[1] \geq M[r]$  для  $2 \leq r \leq i$ 
         7.4:  $M[i+1] \geq M[1]$ ;
8    exchange(M[1],M[i]);
       comment

```

8.1:  $M[i] \geq M[r]$  для  $1 \leq r \leq i-1$   
 8.2:  $M[p] \leq M[p+1]$  для  $i \leq p \leq n-1$   
 8.3:  $M[k \div 2] \geq M[k]$  для  $4 \leq k \leq i-1$ ;  
 end;  
 comment  
 9.1:  $M[p] \leq M[p+1]$  для  $2 \leq p \leq n-1$   
 9.2:  $M[2] \geq M[1]$   
 9.3:  $M[p] \leq M[p+1]$  для  $1 \leq p \leq n-1$ , т. е. массив  $M$   
 полностью упорядочен  
 9.4:  $M$  является перестановкой элементов  $M_0$ ;

### Обоснование утверждений тела процедуры TREESORT 3

Справедливость вышеуказанных утверждений основывается на следующих соображениях.

- 0.1: Пустое утверждение, поскольку  $2(n \div 2 + 1) > n$ .  
 2.1: Если пришли от 0.1, то согласно 1 подставляем  $i = n \div 2$  в 0.1. Если пришли от 3.1, то согласно 1 подставляем  $i = i + 1$  в 3.1, для объяснения изменения  $i$  от 3.1 к 2.1.  
 2.2: 2.1, границы для  $i$  определяются строкой 1, а  $n$ —размер массива.  
 3.1: 2.1 и определение процедуры *siftup*( $i, n$ ).  
 4.1: Пустое утверждение.  
 4.2: Если  $n \geq 4$ , то выполняется 3; следовательно, при  $i = 2$  имеет место 3.1. Если  $n \leq 3$ , то данное утверждение пустое.  
 6.1—6.3: Если пришли от 4.1, то  
 6.1—6.2: Согласно 5 подставляем  $i = n$  в 4.1 и 4.2.  
 6.3: Для  $i = n$  это пустое утверждение.  
 Если пришли от 8.1, то согласно 5 подставляем  $i = i + 1$  в 8.2, 8.3 и 8.1 соответственно.  
 6.4: 5 и 6.2, т. е.  $A(2)$  для подмножества  $M[1:i]$ .  
 7.1: 6.1 и п. 3 определения процедуры *siftup*.  
 7.2: 6.2 и п. 1 определения процедуры *siftup*.  
 7.3: 7.2, заметив, что  $M[1] = M[k \div 2]$  при  $k = 2$  и что операция  $\geq$  рекуррентна.  
 7.4: Для  $i = n$  это пустое утверждение. В других случаях 6.3 для соответствующего  $r$ , так как согласно п.2 определения *siftup*  $M[1]$  в 7.3—это один из элементов  $M[r]$  в 6.3 для  $1 \leq r \leq i$ .  
 8.1: 7.3 с учетом изменений, вызванных строкой 8 (этой строкой изменились значения только  $M[1]$  и  $M[i]$ ).  
 8.2: Согласно 8 заменяем в 7.4  $M[1]$  на  $M[i]$ ; тогда 7.1 будет выполняться и для  $r = i$ .  
 8.3: 7.2, исключая только одно или два соотношения вида  $M[1] \geq \dots$  и одно соотношение вида  $\dots \geq M[i]$ .  
 9.1—9.3: Если  $n \geq 2$ , то выполняется 8;  
 9.1: 8.2 при  $i = 2$ .  
 9.2: 8.1 при  $i = 2$ .  
 9.3: 9.1 и 9.2.  
 Если  $n \leq 1$ , то 9.1—9.3—пустые утверждения.  
 9.4: Над массивом  $M$  выполнялись только операции *siftup* и *exchange*, после каждой из которых массив  $M$  является перестановкой элементов массива  $M_0$ .

### Доказательство конечности процедуры TREESORT 3

Если процедуры *siftup* и *exchange* заканчиваются, то, очевидно, заканчивается и процедура TREESORT 3. Заметим, что оба параметра процедуры *siftup* вызываются по значению, так что в телах операторов цикла значение  $i$  не меняется.

Процедура *exchange*, безусловно, заканчивается. В процедуре *siftup* бесконечный цикл может возникнуть только при переходе от строки 3 к строке 8b и обратно к строке 3. Заметим, что все изменения параметра  $i$  (только в 8b) и переменной  $j$  (только в строках 3 и 6b) встречаются только в этом цикле, и что на каждом витке этого цикла меняются как  $i$  так и  $j$ . Согласно условию в строке 4 достаточно показать, что  $j$  строго возрастает по величине. Неравенство  $i \geq 1$  означает, что  $2i > i$ . В 8b  $j = i < 2i$ , в то время как в 3  $j = 2i$ , т. е.  $j$  (в 3)  $= 2i > i = j$  (в 8b). Следовательно, каждое присваивание переменной  $j$  в строке 3 строго увеличивает значение  $j$ . Другое присваивание переменной  $j$  (в 6b), если оно делается, тоже приводит к увеличению значения  $j$ .

### Замечание к «Подтверждению алгоритма 245» Р. Лондона

К. Редиш (Redish K. A. «The Computer Journal», 1971, № 1)

В своем «Подтверждении к алгоритму 245» («CASM», 1970, № 6) Ралф Л. Лондон продемонстрировал широко распространенную путаницу между понятиями *алгоритм*, его *представление* и результат его трансляции — *код*. При современном состоянии этого вопроса мы можем, вообще говоря, пытаться *доказывать* алгоритм и *апробировать* код. Возьмем, например, утверждение Р. Лондона «... алгоритм TREESORT 3 правильно выполняет свою задачу сортировки массива  $M[1:n]$  в порядке возрастания». В то время как это верно для *алгоритма*, это не будет верно для *кода*, пока мы не наложим ограничения на элементы массива. Ошибка в этом примере возникает от того, что процессоры обладают конечной точностью; логическое выражение  $A \geq B$  (*real*  $A, B$ ) обычно будет транслироваться как  $A - B \geq 0$ , что может дать неверный результат из-за переполнения или возникновения машинного нуля.

Остается вопрос, верно ли указанное утверждение для представления алгоритма в форме данной АЛГОЛ-программы. Поскольку в TREESORT 3 явно не используются никакие арифметические операции, то мы можем говорить, что для данного *представления* на языке АЛГОЛ данного алгоритма доказательство справедливо. Однако это более или менее случайно: оно не было бы справедливым, если бы автор оригинала написал что-нибудь вроде  $\text{if } M[j+1] - M[j] > 0$  даже в одном месте. Кроме того, это налагает весьма большую ответственность на апробирование *кода* и значительно снижает полезность доказательств. (Существует очень много примеров математически правильных алгоритмов, которые дают совершенно неверные результаты, если используется арифметика с конечной точностью.)

По-видимому, доказательство *представления алгоритма* на языке АЛГОЛ должно учитывать раздел 3.3.6 сообщения об этом языке [21]: «Числа и переменные типа *real* должны интерпретироваться в смысле численного анализа, т. е. как объекты, определенные с присущей им конечной точностью.»

Для доказательства TREESORT 3 мы могли бы добавить ограничения

1)  $\min\{\text{abs}(M[i])\} \geq \text{tol}$

2)  $\max\{\text{abs}(M[i])\} \leq 1/2 \times (\text{максимальное представимое число типа real})$ ,

где *tol* — наименьшее представимое число типа *real*, разделенное на *macheps*, где *macheps* — это наименьшее число, такое что  $1 + \text{macheps} \neq 1$ . [Условие 2) могло бы быть заменено условием, что все  $M[i]$  одного знака.]

Теперь мы могли бы с успехом начинать апробирование *кода*; значения величин, используемых в условиях 1) и 2), должны быть хорошо известны для конкретного процессора, и главная задача состоит в том, чтобы проверить, точно ли *код* соответствует *алгоритму*. Несмотря на то, что апробирование выполняется проще, оно должно быть исчерпывающим; опечатка может привести к коварным последствиям.

### Ответ Р. Лондона на вышеуказанное замечание К. Редиша.

Замечание д-ра К. Редиша представляет собой стандартное возражение, часто высказываемое против такого рода доказательства. Мое утверждение состояло и состо-

ит в том, что подтверждение алгоритма может принимать форму доказательства правильности алгоритма. В тех пределах, в которых доказательство учитывает особенности конкретного процессора и конкретной машины, хорошо известные различия между алгоритмом, его представлением и кодом можно игнорировать. Существование «математически правильных алгоритмов, которые дают совершенно неверные результаты, если используется арифметика с конечной точностью» означает только то, что доказательство математической правильности не применимо полностью к соответствующему коду. Но коды доказывались — доказательства [26i, 27i] правильности (исходный код) машинного представления в интервальной арифметике охватывают такие детали, как величина машинных операндов, машинная арифметика, переполнение, машинный нуль и операции с двойной точностью.

Кроме того, мое подтверждение путем доказательства следует совету Хоара [24i, с.579] «...по-видимому, лучше доказывать «условную» правильность программы и полагаться на представление, чтобы предупреждать, если нужно, отказ от выполнения программы, возникающий в результате нарушения границы представления».

Эта точка зрения дает ответ на беспокойство Редиша относительно переполнения и машинного нуля, но она не может быть применима к вопросу об ошибках округления и к вопросу о точности. Разумеется, я не претендую на то, что смогу дать анализ ошибок такой, какой дается в численном анализе. Если доказанный алгоритм вызывает затруднения и если верят доказательству, то причину этих затруднений будут искать в специфических особенностях машины (не учтенных в доказательстве), а не в так называемых «логических ошибках». Доказательство исключает их надежно.

Редиш утверждает, что необходимо накладывать ограничения на элементы массива. Однако существуют реализации языка АЛГОЛ (такие как на машине Wiggoughs B 5500), в которых неравенство  $A \geq B$  не приводит к переполнению или машинному нулю. (Отношение проверяется непосредственно, без вычитания.) Эти ограничения не являются необходимыми даже для кода TREESORT 3, хотя ограничения 1) и 2) представляются достаточными, чтобы исключить переполнение или машинный нуль. Однако ограничение 1) нужно исправить. Слово «наименьшее» в обоих случаях его применения должно быть заменено словами «наименьшее положительное».

Рассмотрим еще строку 3 процедуры *siftup*

loop: j:=2×i;

и представим себе, что сортируется массив  $M[1:N]$ , где  $N$  — наибольшее целое число, представимое в данном процессоре. Существуют простые примеры, которые требуют вычисления  $j=2 \times N$ , т. е. зависящей от представления величины, которая необходима в тесте строки 4. Поскольку массив такого размера, по существу, невозможно даже описать, то можно избегать применения массивов с границей индекса  $N \div 2$ . Так нужно ли и об этом заявлять в явной форме?

Разумеется, в других алгоритмах ограничения могут быть очень нужны, но это зависит от типа доказательства (или от различий между алгоритмом, его представлением и кодом). В идеальном случае особенности арифметики конечной точности (см., например, [28i, 29i]) в доказательстве подтверждения должны учитываться. Но когда это неуместно, невыполнимо или просто не делалось, то допустимо «качество на риск покупателя», точно так же как оно допустимо сейчас, когда доверяют алгоритму, подтвержденному стандартным способом на своей собственной машине. Представляется маловероятным, что стандартное подтверждение или подтверждение доказательством когда-нибудь смогут или будут должны охватывать все особенности всех машин.

Хотя опечатка, на которую ссылается Редиш, действительно может привести к каверзным последствиям, но это могут сделать и многие другие подобные факторы, такие как изменения в трансляторе или операционной системе, не объявленные со времени последнего выхода программиста на машину, непонимание программистом структуры языка или ошибка перфорации. Все эти важные вещи находятся вне моего

обзора проблемы доказательства и подтверждения. Они могут ускользнуть от обнаружения и при «исчерпывающем» апробировании. Заметим, что как Парсонс [29i], так и Кобб [30i], подтверждая отдельные алгоритмы, нашли новые нехитрые ошибки в ранее подтвержденных алгоритмах.

Тот факт, что подтверждение путем доказательства имеет свои границы, не является достаточным аргументом для недооценки и, следовательно, для отказа от его достоинств. Короче, доказательства полезны, но нужно понимать, что они не панацея [31i, 32i].

### **Подтверждение к алгоритмам 251—330**

М. И. Агеев, Москва, октябрь 1975

В течение 1971—1972 годов авторами выпуска были переведены на русский язык и переработаны алгоритмы 251—330 журнала «САСМ». Эти алгоритмы (кроме алгоритмов 266—268, 290, 292, 295, 300, 305, 307, 308, 313, 321, 323, 325, 327—330) были отлажены на машинах М-220 и БЭСМ-6 в системах ТА-1М и БЭСМ-АЛГОЛ. Многие из этих алгоритмов широко используются в настоящее время в различных задачах.

**Замечания к алгоритмам 3б, 4б, 5б, 9б, 22б, 23б, 27б, 32б, 38б, 39б, 40 б, 43б, 48б, 147а, 178**

М. И. Агеев, Москва, май 1975

В материалы выпуска „Библиотека алгоритмов 1б—50б“ [4б] нужно внести следующие поправки.

Номер алгоритма	Страница	Строка	Напечатано	Должно быть
3б	12	11 св.	end	end;
4б	17	9 св.	if abs (y) ≤ eps then go to fin	if abs (y) ≤ eps then go to fin
5б	20	6 св.	—2. 0	2.0
9б	28	33 св.	entier (0.4342944819 × ln (abs (x)) + 1);	entier (0.4342944819 × ln (abs (x)) + 1);
22б	50	14 св.	x ≠ 0 и k=0, 1, ..., n)	x ≠ 0 и k = 0, 1, ..., n
23б*	52	2 св.	x, y [1:k]	x, y [1 : n]
„	53	7 св.	оператор	операторы
„	„	18 св.	begin array x, y [1:k]	begin array x, y [1 : n]
25б*	57	21 св.	fin : c [k] : = r	fin : c [k] : = r
27б	63	1 св.	Ньюхауза	Ньюхауса
27б*	70	4 св.	27 18 3 55 75	0 27 18 3 55 75
32б	82	9 св.	„Замечания к алгоритму 32“	„Замечания к алгоритму 32
„	86	1 св.	чанию к алгоритму 32“ и	чанию к алгоритму 32 и к
38б	95	9 св.	$\sum_{k=1}^n c_k x^k$	$\sum_{k=0}^n c_k x^k$
39б	100	9 св.	r <sub>x<sub>i</sub></sub> , x <sub>j</sub>	$\tilde{r}_{x_i, x_j}$
40б**	101	12 св.	i[k] < i [k + 1]	i[k] ≤ i [k + 1]
43б	116	5 св.	Д. Доминго,	К. Доминго,
48б	125	16 св.	3. 14159264	3.14159265
147а	135	4 св.	Г. Роландом	Р. Парсонсом
„	„	8 св.	Г. Роланда;	Р. Парсонса;
„	136	15 св.	Г. Роландом.	Р. Парсонсом.
178	138	26 св.	нужно заменить на строку	нужно заменить на строки

\* Эти поправки были предложены А. Е. Колесниковым (г. Кишинев, АН МССР).  
Эта поправка была предложена Д. И. Воловым (г. Горький, ГИИВТ).

### Список литературы, которой пользовались авторы выпуска

1. Бронштейн И. Н., Семендяев К. А. Справочник по математике для инженеров и учащихся ВТУЗов. М., Физматгиз, 1962.
2. Агеев М. И., Аликов В. П., Галис Р. М. Алгоритмы (1—50). М., ВЦ АН СССР, 1966.
3. Ригордан Дж. Введение в комбинаторный анализ. М., ИЛ, 1963.
4. Фаддеев Д. К., Фаддеева В. Н. Вычислительные методы линейной алгебры. М., Физматгиз, 1963.
5. Лебедев А. В., Федорова Р. М. Справочник по математическим таблицам. М., АН СССР, 1956.
6. Виноградов И. М. Основы теории чисел. М., Гостехиздат, 1953.
7. Knuth D. E. A Proposal For Input-Output Conventions In ALGOL 60. — «Comm. ACM», 1964, № 5. (Русский перевод: Кнут Д. Е. Проект соглашений по вводу—выводу в языке АЛГОЛ-60. — В кн.: Современное программирование. Вып. 1. М., «Сов. радио», 1966).
8. Диткин В. А. (отв. ред.) Таблицы интегралов Френеля. М., АН СССР, 1953.
9. Носова Л. Н. Таблицы функций Томсона и их первых производных. М., АН СССР, 1960.
10. Программирующая программа. — «Ж. вычисл. матем. и матем. физ.», 1964, № 1. Авт.: Попов В. Н., Степанов В. А., Стишева А. Г., Травникова Н. А.
11. Наур П. (ред.) Сообщение об алгоритмическом языке АЛГОЛ-60. — «Ж. вычисл. матем. и матем. физ.», 1961, № 2.
12. Athman R. E. Subset ALGOL 60 (IFIP). — «Comm. ACM», 6, 1963, № 1 (Русский перевод: Сообщение о сокращенном АЛГОЛе-60 (IFIP). — «Ж. вычисл. матем. и матем. физ.», 1965, № 3).
13. Лавров С. С. Универсальный язык программирования. М., «Наука», 1972.
14. Агеев М. И. Основы алгоритмического языка АЛГОЛ-60. М., ВЦ АН СССР, 1965.
15. Боттенбрух Г. Структура АЛГОЛ-60 и его использование. М., ИЛ, 1963.
16. Мак-Кракен Дж. Программирование на АЛГОЛе. М., «Мир», 1964.
17. Communications of the ACM. Algorithms. USA, 1960—1972.
18. Шура-Бура М. Р., Любимский Э. З. Транслятор АЛГОЛ-60. — «Ж. вычисл. матем. и матем. физ.», 1964, № 1.
19. Хренов Л. С. Пятизначные таблицы тригонометрических функций. М., Физматгиз, 1962.
20. Athman R. E. (Ed.) Suggetions On ALGOL 60 (ROME) Issues. «Comm. ACM», 6, 1963, № 1. (Русский перевод: Атман Р. Е. Предложения по спорным вопросам АЛГОЛ-60 (Рим). — «Ж. вычисл. матем. и матем. физ.», 1964, № 1).
21. Naug P. (Ed.) Revised Report on the Algorithmic Language ALGOL-60. — «Comm. ACM», 1963, № 1. (Русский перевод: Алгоритмический язык АЛГОЛ-60. М., «Мир», 1965).
22. Агеев М. И. Единая форма наглядной записи алгоритмов. — В кн.: Алгоритмы и алгоритмические языки. Вып. 3. М., ВЦ АН СССР, 1968.
23. Алгоритмы (51—100). М., ВЦ АН СССР, 1966. Авт.: Агеев М. И., Аликов В. П., Малюк Л. В., Марков Ю. И.
24. Агеев М. И., Кривонос Л. С., Марков Ю. И. Алгоритмы (101—150). М., ВЦ АН СССР, 1967.
25. Алгоритмы (151—200). М., ИПУ — ВЦ АН СССР, 1970. Авт.: Агеев М. И., Грюнберг М. Г., Марков Ю. И., Швакова Г. М.
26. Агеев М. И., Марков Ю. И., Швакова Г. М. Алгоритмы (201—250). М., ИПУ — ВЦ АН СССР, 1971.
27. Белявский Е. И., Деген А. Б., Этин Ю. Б. АЛГОЛ-60. Л., Гидромет. Изд-во, 1966.
28. Брудно А. Л., АЛГОЛ. М., «Наука», 1971.
29. Система БЭСМ — АЛГОЛ. В кн.: Труды 2-й Всесоюзной конференции по программированию. Новосибирск, Ин-т геологии и геофизики СО АН СССР, 1965. Авт.: Курочкин В. М., Подшивалов Д. Б. и др.

30. Ершов А. П. (ред.) Альфа-система автоматизации программирования. Новосибирск, Ин-т математики СО АН СССР, 1965.
31. Сообщение о процедурах ввода — вывода в языке АЛГОЛ-60. — «Ж. вычисл. матем. и матем. физ.», 1964, № 5.
32. Tables of associated Legendre functions. National Bureau of Standards. Columbia Univ. Press, New York, 1945.
33. Мелентьев П. В. Приближенные вычисления. М., Физматгиз, 1962.
34. Лаврентьев М. А., Шабат Б. В. Методы теории функций комплексного переменного. М., Физматгиз, 1965.
35. Волковский Л. И., Лунц Г. Л., Араманович И. Г. Сборник задач по теории функций комплексного переменного. М., «Наука», 1970.
36. Янке Е., Эмде Ф., Леш Ф. Специальные функции. М., «Наука», 1968.
37. Сегал Б. И., Семендяев К. А. Пятизначные математические таблицы. М., Физматгиз, 1962.
38. Библиотека алгоритмов 16—506. М., «Сов. радио», 1975. Авт.: Агеев М. И., Алик В. П., Галис Р. М., Марков Ю. И.
39. Ветчинкин В. П. Новые формулы и таблицы эллиптических интегралов и функций. М., ВВА РККА, 1935.
40. Bell A. G. Algorithm 50. How to program a computer to play legal chess. — «The Computer Journal», 1970, v. 13, № 5.
41. Алгоритмы и алгоритмические языки. Вып. 3, 4 и 5. М., ВЦ АН СССР, 1968—1971.
42. Mapping J. R. Algorithm 68. White to move and mate in n moves. — «The Computer Journal», 1971, v. 14, № 2.
43. Алексеева С. М. и Алексеев О. Г. Об алгоритме определения кратчайшего пути в сетевом графе. — «Ж. вычисл. матем. и матем. физ.», 1971, № 4, с. 1078—1086.
44. Ляшенко В. Ф. Программирование для цифровых вычислительных машин М-20, БЭСМ-3М, БЭСМ-4, М-220. М., «Сов. радио», 1967.
45. Савинков В. М., Цальп В. Д., Программирование на АЛГОЛе. М., «Высшая школа», 1975.

### Список литературы, на которую ссылаются авторы исходных алгоритмов

- 1i. Dwyer P. S. Linear Computations. Wiley, 1951;
- 2i. DiDonato A. R., Hershey A. V. New Formulae for Computing Incomplete Elliptic Integrals of the First and Second Kind. — «J. ACM», Oct. 1959, 6, 4.
- 3i. Warshall S. A theorem on Boolean matrices. — «J. ACM», 1962, 9, 11—12.
- 4i. Good I. J. A Five-year Plan for Automatic Chess. Edinburgh, Machine Intelligence 2, Oliver and Boyed, 1967.
- 5i. Greenblatt R. D., Eastlake D. E., Crocker S. D. The Greenblatt Chess Program. Thomson, AFIPS Conference Proceedings, 1967.
- 6i. Huberman B. J. A Program to Play Chess End Games. Stanford University, Computer Science Department, Technical Report № CS106, 1968.
- 7i. Samuel A. L. Some Studies in Machine Learning using the Game of Checkers. II—Recent Progress, IBM Journal, November, 1967.
- 8i. Scott J. J. A chess-playing program. Machine Intelligence 4, Edinburgh University Press, 1969.
- 9i. Lipton M., Matthews R. C. O., Rice J. M. Chess problem: Introduction to an art. Faber & Faber. 1963.
- 10i. Peck J. E. L. Polynomial curve fitting with constraint. — «Soc. Indust. Appl. Math. Rev.», 1961.



- 11i. Goerzel G. Mathematical Methods for Digital Computers.
- 12i. Legendre A. M. Tafeln der Elliptischen Normalintegrale. Stuttgart, 1931.
- 13i. Romberg W. Vereinfachte numerische Integration. Det. Konglinge Norske Videnskaber Selskab Forhandling, 28, 1955, 30—36.
- 14i. Stiefel E., Rutishauser H. Remarques concernant l'integration numerique.— «Comptes Rendus Acad. Seil» (Paris), 252, 1961, 1899—1900.
- 15i. Bauer F. L., Rutishauser H., Stiefel E. New aspects in numerical quadrature. — «Proc. Symp. Appl. Math.», 15, 1963, (199—218).
- 16i. Rutishauser H. Ausdehnung des Rombergchen Prinzips. — «Numer. Math.», 5, 1963 (48—54).
- 17i. Greenstadt J. The determination of the characteristic roots of a matrix by the Jacobi method. In: «Mathematical Methods for Digital Computers». A. Ralston and H. S. Wilf, eds.
- 18i. Floyd R. W. Assigning meanings to programs. Proc. of a Symposium in Applied Mathematics, Vol. 19—Mathematical Aspects of Computer Science, J. T. Schwartz (Ed.), American Math. Society, Providence, R. I., 1967, pp. 19—32.
- 19i. Knuth D. E. The Art of Computer Programming. Vol. 1—Fundamental Algorithms. Addison—Wesley, Reading, Mass., 1968, Sec. 1.2.1.
- 20i. McCarthy J. A basis for a mathematical theory if computation. In Computer Programming and Formal Systems, P. Braffort and D. Hirschberg (Eds.). North Holland, Amsterdam, 1963, pp. 33—70.
- 21i. McCarthy J. and Painter J. A. Correctness of a compiler for arithmetic expressions. Proc. of a Symposium in Applied Mathematics. Vol. 19—Mathematical Aspects of Computer Science, J. T. Schwartz (Ed.), American Math. Society, Providence, R. I., 1967, pp. 33—41.
- 22i. Naur P. Proof of algorithms by general snapshots. BIT6, 1966, pp. 310—316.
- 23i. Naur P. (Ed.) Revised report on the algorithmic language, ALGOL-60. — «Comp., J.», 5 (Jan. 1963), 349—367.
- 24i. Hoare C. A. R. An axiomatic basis for computer programming. — «Comm. ACM», 12 (Oct. 1970), 576—583.
- 25i. Burstall R. M. Proving properties of programs by structural induction. — «Comp. J.», 1969, № 9, 41—48.
- 26i. Good D. I. and London R. L. Interval arithmetic for the Burroughs B5500: Four Algol procedures and proofs of their correctness. Computer Science Tech. Rep. № 26, U. of Wisconsin, Madison, Wis., June 1968.
- 27i. Good D. I. and London R. L. Computer interval arithmetic: Definition and proof of correct implementation. — «J. ACM», 17 (Oct. 1970), 603—612.
- 28i. Van Wijngaarden A. Numerical analysis as an independent science. — «BIT», 6 (1966), 66—81.
- 29i. Parsons R. G. Certification of Algorithm 147. — «Comm. ACM», 12 (Dec. 1969), 691—692. (Русский перевод: Р. Парсонс «Подтверждение к алгоритму 147» [38]).
- 30i. Cobb S. M. Certification of Algorithm 47. — «Comm. ACM», 12 (Nov. 1969), 635—636. (Русский перевод: С. Кобб «Подтверждение к алгоритму 47» [38]).
- 31i. London R. L. Proving programs correct: some techniques and examples. — «BIT», 10 (1970), 168—182.
- 32i. London R. L. Computer programs can be proved correct. In Theoretical Approaches to Non—Numerical Problem Solving. R. B. Banerji and M. D. Mesarovic (Eds.), Lecture Notes in Operations Research and Mathematical Systems 28, Springer Verlag, New York, 1970, pp. 281—302.
- 33i. Moore R. Automatic Error Analysis in Digital Computation. KMSD Report 48421, 28 Jan. 1959.
- 34i. Fettis H. E. Algorithm 163. Modified Hankel funktion. — «CACM», 1963, № 9.

# СОДЕРЖАНИЕ

Предисловие	3
Алгоритм 516. Корректировка обратной матрицы после изменения одного элемента в прямой матрице [F1]	6
Алгоритм 526. Генератор тест-матриц [F1]	7
Алгоритм 536. Извлечение корней $n$ -й степени из комплексного числа [B4]	9
Свидетельство к алгоритму 546 [S14]	11
Алгоритм 556. Полный эллиптический интеграл первого рода [S21]	11
Алгоритм 566. Полный эллиптический интеграл второго рода [S21]	12
Алгоритм 576. Функций Томсона $ber$ и $bei$ [S19]	14
Алгоритм 586. Обращение матрицы методом Гаусса-Жордана [F1]	15
Свидетельство к алгоритму 596 [C2]	17
Алгоритм 606. Вычисление интеграла по Ромбергу [D1]	17
Алгоритм 616. Процедуры интервальной арифметики [A1]	21
Алгоритм 626. Последовательность присоединенных функций Лежандра второго рода [S16]	26
Алгоритм 636. Разделение элементов массива [M1]	29
Алгоритм 646. Быстрая сортировка (рекурсивная процедура) [M1]	31
Алгоритм 656. Поиск элемента в сортируемом массиве (рекурсивная процедура) [M1]	32
Алгоритм 666. Обращение симметричной матрицы методом квадратных корней [F1]	33
Алгоритм 676. Умножение уплотненной симметричной матрицы на прямоугольную [F1]	36
Свидетельство к алгоритму 686 [A1]	37
Свидетельство к алгоритму 696 [H]	38
Алгоритм 706. Интерполяция по Эйткуэну [E1]	38
Свидетельство к алгоритму 716 [G6]	39
Алгоритм 726. Генератор композиций [A1]	39
Алгоритм 736. Неполные эллиптические интегралы [S21]	40
Алгоритм 746. Аппроксимация с помощью полиномиальной кривой данной степени, проходящей через данные точки (метод наименьших квадратов) [E2]	44
Алгоритм 756. Разложение многочлена на множители [C2]	49
Свидетельство к алгоритму 766 [M1]	51
Алгоритм 776. Интерполяция, дифференцирование и интегрирование функций [D1, D4, E1]	52
Алгоритм 786. Корни полиномов с целыми коэффициентами, получаемые в форме простых дробей [C2]	56
Алгоритм 796. Коэффициенты полиномиальной аппроксимации производной любого порядка от табличной функции [D4]	58
Алгоритм 806. Вычисление обратной гамма-функции с точностью до 10 цифр [S14]	61
Свидетельство к алгоритмам 816, 826, 836 [G7]	62
Алгоритм 846. Вычисление интеграла по Симпсону от таблично заданной функции [D1]	63
Алгоритм 856. Вычисление собственных значений и собственных векторов симметричной матрицы методом Якоби [F2]	64
Свидетельство к алгоритмам 866 и 876 [G6]	71
Свидетельство к алгоритмам 886, 896 и 906 [S20]	72
Свидетельство к алгоритму 916 [E2]	72
Алгоритм 926. Решение системы линейных алгебраических уравнений и обращение матрицы [F4]	72
Свидетельство к алгоритму 936 [A1]	74
Алгоритм 946. Генератор сочетаний [G6]	75
Свидетельство к алгоритму 956 [A1]	76
Алгоритм 966. Матрица причинно-следственных отношений [H]	76
Алгоритм 976. Кратчайший путь [H]	77
Алгоритм 986. Комплексный криволинейный интеграл [D1]	79
Алгоритм 996. Вычисление символа Якоби [A1]	82
Свидетельство к алгоритму 1006 [O2]	83
Приложение 1. Алгоритм 50 CJ. Как программировать игру в шахматы [Z]	84
Приложение 2. Подтверждения и замечания к алгоритмам, опубликованным в предыдущих выпусках	118
Список литературы, которой пользовались авторы выпуска	130
Список литературы, на которую ссылаются авторы исходных алгоритмов	131

МИХАИЛ ИЛЛАРИОНОВИЧ АГЕЕВ

ВЛАДИМИР ПАВЛОВИЧ АЛИК

ЮРИЙ ИВАНОВИЧ МАРКОВ

**БИБЛИОТЕКА АЛГОРИТМОВ**

516 — 1006

Справочное пособие

Выпуск 2

Редактор Н. Г. Д а в ы д о в а

Художественный редактор З. Е. В е н д р о в а

Технический редактор А. А. Б е л о у с

Корректор Л. А. М а к с и м о в а

Сдано в набор 3/III 1975 г.    Подписано к печати 23/XII 1975 г. §    Т-20234

Формат 70×100/16

Бумага типографская № 1

Объем 11,05 усл. п. л.,

10,034 уч.-изд. л.

Тираж 42 000 экз.

Зак. 311

Цена 53 коп.

Издательство «Советское радио», Москва, Главпочтамт, а/я 693

Московская типография № 10 Союзполиграфпрома  
при Государственном Комитете Совета Министров СССР  
по делам издательств, полиграфии и книжной торговли.  
Москва, М-114, Шлюзовая наб., 10.

**Агеев М. И. и др.**

**A23 Библиотека алгоритмов 516—1006. (Справочное пособие). Вып. 2. М., «Сов. радио», 1976.**

136 с. с ил. (Серия «Библиотека технической кибернетики»).

Перед заглав. авт.: М. И. Агеев, В. П. Алик, Ю. И. Марков

Приведены переводы на русский язык алгоритмов по вопросам прикладной математики и программирования на языке АЛГОЛ-60, опубликованных в журнале «САСМ» (США) под номерами 51—100, исправленных и отраженных на ЭВМ, а также снабженных подтверждениями и свидетельствами. Как приложение приводится алгоритм «Как программировать игру в шахматы» и статья Р. Лондона «Доказательство алгоритмов — новый вид подтверждения».

Книга предназначена для специалистов, связанных с работами на ЭВМ.

А  $\frac{30502-021}{046(01)-76}$  80-75

**6Ф7.3**

## ГОТОВЯТСЯ К ВЫХОДУ В СВЕТ

В ИЗДАТЕЛЬСТВЕ

«СОВЕТСКОЕ РАДИО»

**Параев Ю. И. Введение в статистическую динамику процессов управления и фильтрации.**

Систематизированно излагается обширный (в основном журнальный) материал по применению теории марковских процессов в задачах автоматического управления и обработки информации. Исследуется поведение, вычисляются оптимальные оценки текущих значений координат автоматических систем, оптимальное управление такими системами, в том числе и при стохастической обратной связи. Подробно рассматриваются автоматические системы, описываемые линейными дифференциальными уравнениями и линейными уравнениями со случайными коэффициентами.

Книга написана на доступном математическом уровне и предназначена для инженеров, студентов, аспирантов и научных работников, занимающихся применением вероятностных методов к задачам автоматического управления и обработки информации. Она может также служить учебным пособием для студентов и аспирантов по специальностям автоматика и телемеханика, статистическая радиофизика, техническая кибернетика и др.

**Пацюков В. П. Дифференциальные игры при различной информированности игроков.**

Излагаются точные и приближенные методы решения антагонистических дифференциальных игр, оценки оптимальности управляемых систем, методы решения антагонистической дискретной игры и игр с правилами, описываемыми уравнениями в частных производных. Излагаются методы решения некоторых безапелляционных дифференциальных игр  $n$  лиц. Книга является монографией по дифференциальным играм при различной информированности игроков. Иллюстрирована многочисленными примерами.

Книга удачно дополняет изданные ранее книги аналогичной тематики (кн. Флейшмана Б. С., Волгина Л. Н. и др.).

Книга рассчитана на широкий круг читателей—инженеров, научных работников и экономистов, интересующихся применением математики к обоснованию оптимальных решений в условиях конфликта.

*Указанные книги можно заказать и купить в магазинах, распространяющих научно-техническую литературу.*

Цена 53 к.

