№ 34 (323) 8–15 сентября 2001

ИНФОРМАТИКА

Еженедельная газета Объединения педагогических изданий «ПЕРВОЕ СЕНТЯБРЯ»
ПОДПИСКА: (095) 249-47-58

Читайте в номере

Пермская олимпиада по базовому курсу информатики

Л.А. Залогова, С.В. Русаков, И.Г. Семакин, Л.В. Шестакова,
г. Пермь

Продолжение. Начало в № 33/2001

9-й класс. Теоретический тур*

1. Квадрат, круг, ромб и треугольник вырезаны из белой, синей, красной и зеленой бумаги. Известно, что круг не белый и не зеленый, синяя фигура лежит между ромбом и красной фигурой, треугольник не синий и не зеленый, квадрат лежит между треугольником и белой фигурой. Какая фигура вырезана из зеленой бумаги?

а) Круг.
б) Ромб.
в) Квадрат.
г) Треугольник.
д) Круг или ромб.

2. Подозреваемые в ограблении банка дали следующие показания:

Петров: "Это сделал не я". Иванов: "Это сделал Сидоров".

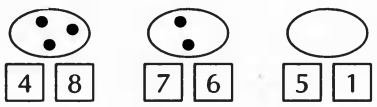
Сидоров: "Это сделал не я". Алексеев: "Это сделал Иванов".

Свидетель утверждает, что только один из них сказал правду. Кто совершил преступление?

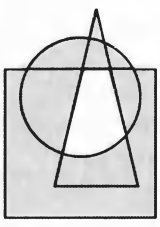
а) Алексеев.
б) Петров.
в) Иванов.
г) Сидоров.
д) Точного ответа дать нельзя.

3. Сколько вишен надо положить в пустую тарелку?

а) 1.
б) 9.
в) 4.
г) 5.
д) 2.



4. Высказывания X, Y, Z истинны для точек, принадлежащих прямоугольнику, треугольнику и кругу соответственно. Для всех точек, принадлежащих выделенной серым цветом области, истинно высказывание:



а) X **и** Y **и** Z **или не** Z.
б) X **и** Y **или не** Z.
в) (X **или** Y **или** Z) **или не** (X **и** Y **и** Z).
г) (X **или** Y **или** Z) **и не** (X **и** Y **и** Z).
д) (X **или** Y **или** Z) **и не** X **и не** Y **и не** Z.

5. Выбрать высказывание на обычном языке, соответствующее следующему логическому выражению:

($X + Y > 13$) **и** ($X + Z \le 13$) **и** ($Y + Z \le 13$)
или ($X + Y \le 13$) **и** ($X + Z > 13$) **и** ($Y + Z \le 13$)
или ($X + Y \le 13$) **и** ($X + Z \le 13$) **и** ($Y + Z > 13$).

а) Хотя бы одна пара чисел из X, Y, Z в сумме превышает 13.

б) Хотя бы одна пара чисел из X, Y, Z в сумме не превышает 13.

в) Только одна пара чисел из X, Y, Z в сумме не превышает 13.

г) Только одна пара чисел из X, Y, Z в сумме превышает 13.

д) Все пары чисел из X, Y, Z в сумме превышают 13.

6. Рядом со школой растут 6 деревьев: сосна, береза, липа, тополь, ель и клен. Известно, что береза ниже тополя, а липа выше клена, сосна ниже ели, липа ниже березы, сосна выше тополя. Перечислить деревья в порядке возрастания их высоты.

а) Сосна, береза, липа, тополь, ель, клен.
б) Ель, сосна, тополь, береза, липа, клен.
в) Липа, клен, береза, тополь, сосна, ель.
г) Клен, липа, береза, сосна, тополь, ель.
д) Клен, липа, береза, тополь, сосна, ель.

7. Вычислить значение выражения $ZXY + YXZ$, если известно, что ни одна из десятичных цифр X, Y и Z не равна нулю и $3YZ + ZYX = 603$.

а) 808.
б) 727.
в) 707.
г) 8208.
д) Значение выражения вычислить невозможно.

8. Сколько существует трехзначных чисел, в записи которых встречается только одна двойка?

а) 729.
б) 243.
в) 216.
г) 234.
д) 225.

9. В саду 100 фруктовых деревьев. 14 яблонь и 42 груши. В какой системе счисления посчитаны деревья?

а) 2.
б) 3.
в) Такой системы не существует.
г) 4.
д) 6.

* Несколько первых заданий являлись общими в теоретических турах 9-го и 8-го классов (см. № 33). Мы специально повторяем их для удобства чтения и практического использования материала.

10. Восстановите в примере двоичные цифры, на месте которых стоит знак "*": $1*01_2 + 1**_2 = 1*100_2$

а) $1001_2 + 101_2 = 11100_2$.
б) $1101_2 + 101_2 = 10100_2$.
в) $1101_2 + 111_2 = 10100_2$.
г) $1001_2 + 111_2 = 11100_2$.
д) $1101_2 + 101_2 = 11100_2$.

11. Следующий член последовательности 12; 101; 120; 202; 221 равен

а) 1010.
б) 310.
в) 301.
г) 230.
д) 300.

12. Программист написал программу на языке машинных команд. Каждая команда занимает 4 байта памяти и размещается в ней, начиная с нулевого адреса. Шестнадцатеричный адрес последней команды в программе равен 28. Сколько команд содержит эта программа?

а) 7.
б) 8.
в) 10.
г) 11.
д) 28.

13. Какая пара объектов находится в отношении "объект — модель"?

а) Компьютер — программа.
б) Программа — блок-схема.
в) Вода — химическая формула воды H_2O.
г) Телевизор — программа передач.
д) Учитель — расписание уроков.

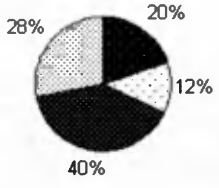
14. Упорядочить фрагменты текста "1Г6", "10Г6", "500М6", "5Г6" в порядке убывания:

а) "10Г6", "5Г6", "1Г6", "500М6".
б) "10Г6", "1Г6", "500М6", "5Г6".
в) "500М6", "5Г6", "10Г6", "1Г6".
г) "5Г6", "500М6", "1Г6", "10Г6".
д) "500М6", "10Г6", "5Г6", "1Г6".

15. Дан фрагмент таблицы:

	A	B	C	D
1	125	75		175

Восстановить числовое значение клетки C1 при условии, что ему соответствует сектор (40%) на круговой диаграмме.



а) 250.
б) 1000.
в) 1562,5.
г) 150.
д) 937,5.

16. Дан фрагмент таблицы:

	A	B
1	осадки	мм
2	дождь	13
3	дождь	17
4	снег	12
5	дождь	9
6	снег	18

Что будет высвечиваться в клетке B7, если туда занесена формула:
=СЧЕТЕСЛИ(A2:A6;"дождь")+СЧЕТЕСЛИ(B2:B6;">10")?

а) 3.
б) 4.
в) 7.
г) 0.
д) Дождь.

17. Дан фрагмент таблицы:

	A	B	C
1	12	=$A1+5	
2	15		

Содержимое клетки B1 скопировано в клетки C1 и B2. Какие формулы будут занесены в эти клетки?

а) C1=$A1+5 и B2=$A1+5.
б) C1=$A2+5 и B2=$A2+5.
в) C1=$B1+5 и B2=$A2+5.
г) C1=$B1+5 и B2=$B2+5.
д) C1=$A1+5 и B2=$A2+5.

18. Дан фрагмент таблицы:

	A	B	C	D	E
1	ФИО	пятерки	четверки	тройки	
2	Школьников	75	20	5	грамота

Ученик поощряется грамотой за хорошую учебу, если количество пятерок составляет более 70% от количества всех его оценок.

Выбрать формулу, позволяющую определять, получит ли ученик грамоту.

а) =ЕСЛИ(B2/СУММ(B2:D2)>70;"ГРАМОТА";"НЕТ").
б) =ЕСЛИ(СУММ(B2:D2)*100/B2>70;"ГРАМОТА";"НЕТ").
в) =ЕСЛИ(B2/СУММ(B2:D2)*100>70;"ГРАМОТА";"НЕТ").
г) =ЕСЛИ(B2/СУММ(B2:D2)*100 >70;"НЕТ";"ГРАМОТА").
д) =ЕСЛИ(B2/СУММ(B2:D2)>70; "НЕТ";"ГРАМОТА").

19. Дан фрагмент таблицы в режиме отображения формул:

	A	B
1	23	=A2+B3
2	47	=A1+A3
3	12	=A3+B1

Чему будут равны значения клеток B1, B2 и B3 в режиме отображения значений?

а) Вычислить невозможно.
б) B1=59; B2=35; B3=12.
в) B1=23; B2=35; B3=12.
г) B1=59; B2=35; B3=35.
д) B1=23; B2=35; B3=35.

20. Даны команды графического исполнителя, обитающего в координатной плоскости с целочисленными координатами:

(1) ПРЫЖОК в текущем направлении в соседнюю точку;

(2) ШАГ в текущем направлении в соседнюю точку;
(3) ПОВОРОТ НА 90° ПРОТИВ ЧАСОВОЙ СТРЕЛКИ;
(4) ПОВОРОТ НА 45° ПРОТИВ ЧАСОВОЙ СТРЕЛКИ;
(5) ЛИНИЯ(X_1, Y_1, X_2, Y_2) — рисование линии из точки с координатами (X_1, Y_1) в точку с координатами (X_2, Y_2);
(6) ПЕРЕМЕЩЕНИЕ(X_1, Y_1, X_2, Y_2) из точки с координатами (X_1, Y_1) в точку с координатами (X_2, Y_2). Выбрать минимальный набор команд, позволяющий нарисовать все цифры почтового индекса.

а) (1), (2), (3), (4).
б) (1), (2), (4).
в) (1), (2), (4), (6).
г) (4), (5).
д) (5).

Дана база знаний, содержащая сведения о времени начала и окончания работы магазина, продавцах: магазин(эльдорадо,8,20). магазин(сатурн,9,19). магазин(весна,10,19). продавец(юля,сатурн). продавец(сергей, эльдорадо). продавец(ваня,весна). продавец(катя,сатурн). день(*X*,*A*,*B*):—продавец(*X*,*Y*),магазин(*Y*,*A*,*B*).

21. Каким будет ответ на цель: ?день(*X*,*A*,_),*A*<10.
а) *X*=эльдорадо *A*=8 *X*=сатурн *A*=9.
б) *X*=юля *A*=9 *X*=сергей *A*=8 *X*=катя *A*=9.
в) Нет решения.
г) *X*=юля *X*=сергей *X*=ваня *X*=катя.
д) *X*=эльдорадо *X*=сатурн *X*=весна.

22. Каким будет ответ на цель:
?продавец(*X*,*A*),продавец(*Y*,*A*),*X*<>*Y*.
а) Нет решения.
б) *X*=юля *X*=сергей *X*=катя.
в) *X*=сатурн.
г) *X*=юля *A*=сатурн *X*=катя *A*=сатурн.
д) *X*=ваня *X*=сергей.

23. Каким будет ответ на цель: ?день(*X*,_,19).
а) *X*=сатурн *X*=весна.
б) *X*=эльдорадо.
в) Нет решения.
г) Нет.
д) *X*=юля *X*=ваня *X*=катя.

24. Сформулировать цель: "Какой магазин начинает работу раньше 10 часов и заканчивает позже 19?"
а) магазин(*X*,*A*,*B*),*A*<10;*B*>19.
б) день(*X*,*A*,*B*),*A*<10,*B*>19.
в) магазин(*X*,*A*,*B*),*A*<10,*B*>19.
г) день(*X*,*A*,*B*),*A*<10;*B*>19.
д) день(*X*,*A*,*B*),*A*<10,*B*>19; магазин(*X*,*A*,*B*),*A*<10,*B*>19.

25. Каким будет ответ на цель: ?день(юля,11,20).
а) Нет решения.
б) Нет.
в) сатурн.
г) катя.
д) юля.

Дана однотабличная база данных "Автомобилисты":

	Владелец	Модель	Номер	Дата регистрации
1	Левченко Н.	Волга	И537ИП-59	15.08.96
2	Сидоров А.	Жигули	Ф131ФП-59	14.02.95
3	Горохов И.	Форд	Б171БП-59	27.10.95
4	Федоров К.	Волга	И138ИП-59	20.05.96
5	Сидоров А.	Жигули	И321ИП-59	27.10.95

Записи пронумерованы.

26. Определить ключевое поле таблицы.
а) Владелец.
б) Модель.
в) Номер.
г) Дата регистрации.
д) Владелец+Модель.

27. Сформулировать условие отбора, позволяющее получить номера "волг" и "жигулей", зарегистрированных ранее 01.01.96.
а) ((Модель="Волга") OR (Модель="Жигули")) AND (Дата регистрации>01.01.96).
б) (Модель="Волга") OR (Модель="Жигули") OR (Дата регистрации>01.01.96).
в) (Модель="Волга") AND (Модель="Жигули") AND (Дата регистрации<01.01.96).
г) ((Модель="Волга") OR (Модель="Жигули")) AND (Дата регистрации<01.01.96).
д) (Модель="Волга") AND (Модель="Жигули") OR (Дата регистрации<01.01.96).

28. Укажите порядок строк таблицы после сортировки в порядке возрастания по двум полям: Модель+Номер.
а) 1; 4; 2; 5; 3.
б) 3; 4; 5; 1; 2.
в) 4; 1; 5; 2; 3.
г) 3; 5; 2; 4; 1.
д) 2; 1; 5; 4; 3.

29. Какие записи будут удовлетворять условию отбора: (Дата регистрации>13.02.95) AND (Дата регистрации<28.10.95)?
а) Таких записей нет.
б) 2; 3; 5.
в) 1; 4.
г) 1.
д) 4.

30. Какого типа должно быть поле Номер?
а) Числовое.
б) Текстовое или числовое.
в) Текстовое.
г) Дата или числовое.
д) Текстовое или дата.

Ключ к тесту

№ вопроса	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Правильный ответ	в	б	б	г	г	д	б	д	д	в	а	г	в	г	а
№ вопроса	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Правильный ответ	в	д	в	а	д	б	г	д	в	б	в	г	в	б	в

Принципы проверки учебных и олимпиадных задач по информатике

Е.В. Андреева,
Москва

Введение

При обучении программированию в курсе информатики наиболее сложным для учащихся оказывается самопроверка (отладка и тестирование собственных программ). Дальнейшая же проверка правильности выполнения учебных заданий преподавателем, в свою очередь, является наиболее трудоемкой частью процесса обучения и предполагает проработку методики будущего оценивания ожидаемых решений. Если при проверке решения теоретических задач основное внимание уделяется анализу хода решения и правильности полученного ответа, то при решении практических задач ситуация иная. Несмотря на важность работ в области доказательства правильности программ [1, 2], предлагаемые в них методы можно и нужно использовать в основном во время разработки алгоритма решения задачи. Завершением же этапа реализации (собственно программирования) является тестирование программы, то есть выявление особенностей представленного решения на наборе специально подобранных тестов.

Признавая очевидные преимущества проведения тестирования программ с целью оценки их работоспособности и эффективности, многие преподаватели искренне полагают, что результаты тестирования никогда не могут дать достоверной информации о качестве программы и уж тем более оценить степень обученности учащихся. Тем не менее тестирование уже много лет используется в мире как средство оценки решений различных задач, и это ведет к совершенствованию систем тестов, появлению в данной области деятельности новых компьютерных технологий, в большей степени отвечающих потребностям методологии современного программирования. Ниже будут сформулированы требования, которым должна отвечать качественная, профессионально разработанная полная система тестов, предназначенная для проверки учебной или олимпиадной задачи по программированию. Такая система тестов зачастую может оказаться для преподавателя гораздо более полезной, чем попытка найти все возможные ошибки или же, наоборот, доказать правильность уже существующей программы ученика с помощью анализа листинга предложенного решения. С другой стороны, никакая система тестирования не может быть самодостаточным инструментом для проверки решений задач в процессе обучения программированию. Так как, если программа уже успешно прошла все тестовые испытания, преподавателю имеет смысл просмотреть ее листинг в присутствии ученика, отметить недостатки (если таковые имеются) в эффективности программной реализации тех или иных элементов алгоритма и в стиле программирования. Если же в результате беседы выяснится, что решение было основано на имплицитном знании, то следует закрепить такое интуитивно верное решение в вербальной (эксплицитной, логической) форме [3]. Для этого необходимо в разбор заданий, проводимый для учащихся, включать и методику оценивания программ.

Одним из вариантов развивающего обучения в дидактике признано проблемное обучение. Его целью, в отличие от традиционного обучения, является усвоение не только результатов научного познания и системы знаний, но и поиск самого пути. В результате формируется познавательная самостоятельность ученика и развиваются его творческие способности. Таким образом, проблемное обучение — это оптимальное сочетание репродуктивной и творческой деятельности по передаче и усвоению системы научных понятий и приемов, способов логического мышления. Это дидактический подход, учитывающий психологические закономерности мыслительной деятельности субъектов [4]. С внедрением компьютера в процесс обучения впервые появилась возможность, в несколько раз повысив активность учащихся, обеспечить цикличность функционирования традиционного контура обратной связи "преподаватель—ученик" в реальном масштабе времени. В результате намного проще стало реализовывать ведущие принципы развивающего обучения: индивидуализацию и дифференциацию. Кроме того, компьютер, позволяя ошибаться, разрешая ошибаться, создавая возможность ошибаться, дает возможность познавать через противоречия. А, как показано Ж.Пиаже [5], ошибочные теории учеников являются существенной частью процесса формирования мышления.

Постановка задачи с учетом ее дальнейшей автоматической проверки

Наиболее полно принципы тестирования промышленных программ были сформулированы в [6]. Однако они должны быть модифицированы и дополнены применительно к особенностям учебных программ. Если тестирование программы проводится интуитивно и без четкого плана испытаний, то этот процесс можно назвать искусством. Если же тестированию предшествует тщательный подбор данных для контрольных примеров, а само тестирование выполняется последовательно и аккуратно, то тестирование становится наукой. Тестовые испытания учебной программы направлены на то, чтобы проверить правильность работы программы во всех предполагаемых практических ситуациях, оговоренных в условии задачи, а также оценить ее эффективность. При этом программа учащегося, с точки зрения преподавателя, является "черным ящиком". Проверку такой программы без непосредственного участия ученика можно провести лишь в том случае, когда задача поставлена

четко и полностью. Особое внимание при этом следует уделять описанию форматов входных и выходных данных, типам входных величин и диапазонам их допустимых значений. Полезным является включение тестовых примеров входных и выходных данных в условие задачи. Эта мера позволяет ученику проверить, правильно ли он понимает задачу. В последнее время неотъемлемой частью формулировки условия учебной или олимпиадной задачи по программированию является явное указание на максимальное допустимое время ее работы на любых данных, не противоречащих условию. Данное указание как раз и позволяет в дальнейшем оценить эффективность алгоритма, используемого при написании программы. Приведем пример неудачной и удачной формулировки условия учебной задачи с точки зрения ее дальнейшего тестирования преподавателем.

Условие 1.

Написать программу сортировки числового массива по неубыванию.

Условие 2.

Написать программу сортировки массива, состоящего из N ($N \le 30\,000$) целых чисел, каждое из которых по модулю не превосходит 32 000, по неубыванию.

Входные данные вводятся с клавиатуры. В первой строке ввода находится число N — размер массива, в следующих строках N элементов массива, разделенных пробелами и/или символами перевода строки.

Результат работы программы — N отсортированных чисел, вывести в текстовый файл OUTPUT.TXT, разделяя числа пробелами и/или символами перевода строки.

Время работы программы на одном тесте не должно превышать 3 *с*.

Пример входных данных:

```
3
4 -1
2
```

Возможный файл OUTPUT.TXT *для приведенного примера входных данных*:

```
-1 2 4
```

Таким образом, постановка задачи (спецификация задания) должна обладать свойствами четкости и полноты.

Система тестов для проверки учебных и олимпиадных задач

Перейдем теперь к описанию тестовых данных, необходимых для оценки работоспособности и качества программ учащихся.

1. Как верно указано в [6], первый тест должен быть максимально прост, так как его цель — проверить, работает ли программа вообще. Вполне допустимо в качестве первого теста использовать тест из условия задачи. Именно с помощью такого теста преподаватель сможет определить — верно ли понято учащимися условие задачи, соответствует ли программа описанным в условии форматам входных и выходных данных и правильно ли в ней названы входной и выходной файлы (если в условии задачи предполагается, что ввод и/или вывод осуществляется через файл). Если это не так, то при автоматической проверке учащемуся будет указан следующий вид ошибки: *Presentation Error* — нарушение формата представления выходных данных.

2. Вторая группа тестов должна отслеживать так называемые *вырожденные случаи*. То есть случаи, когда решение задачи или не существует (тогда в условии задачи должно быть оговорено, что именно должна сообщать программа в такой ситуации), или коренным образом отличается от основного алгоритма решения. В первую очередь к данной группе относятся *нулевые примеры*. Для числовых входных данных это обычно нулевые значения (конечно, если по условию задачи ноль является допустимым значением для той или иной переменной), а для текстовых — это пустой входной файл, а также последовательность из пробелов и/или символов перевода строки. Другой пример вырожденности — нарушение общности входных данных, требующее специальной их обработки. Так, если в учебной задаче требуется решить уравнение $ax^2 + bx + c$ для любых действительных значений a, b и c, то при $a = 0$ квадратное в общем случае уравнение становится линейным, что программа учащегося, несомненно, должна учитывать. Причем только в этом случае существенным становится равенство нулю значения параметра b. Таким образом, для проверки данной задачи необходимы следующие тесты на "вырожденность":

$$a = 0,\ b = 0,\ c = 0;$$
$$a = 0,\ b = 0,\ c = 1;$$
$$a = 0,\ b = 1,\ c = 2.$$

Если же в задаче входными параметрами являются координаты N точек на плоскости, а требуется, например, найти такую точку на той же плоскости, расстояние от которой до наиболее удаленной из N точек минимально, то очевидно, что при решении задачи случаи $N = 1$ и $N = 2$ следует рассматривать отдельно. Если программа учащегося обрабатывает подобные входные данные некорректно, то при автоматической проверке возможны следующие типы диагностики ошибок: *Runtime Error* — ошибка выполнения программы (в основном в результате деления на ноль) или *Wrong Answer* — неверный ответ (программа не учитывает, что при вырожденных входных данных алгоритм их обработки может иметь специфические отличия от основного алгоритма).

3. Следующая группа тестов должна проверять *граничные случаи* (входные и выходные данные принимают граничные значения). Основное назначение подобных тестов — обнаружить возможные программистские ошибки, которые могли быть сделаны учащимися при реализации в том числе и правильного алгоритма. Прежде всего в данном случае следует проверить, что при написании программы для переменных были выбраны подходящие типы данных. Так, если программа вычисляет сумму целых чисел, каждое из которых по модулю не превосходит 32 767, то есть для представления таких чисел можно

использовать двухбайтовый знаковый тип данных, то для вычисления результата такого типа уже явно недостаточно. Причем может даже оказаться, что получение точного результата возможно только с помощью так называемой "длинной арифметики" [7] — организации выполнения арифметических действий с помощью массивов, что существенно меняет сложность решения.

В задачах, при решении которых используются численные методы, к граничным можно отнести тестовые данные, для обработки которых требуется максимальная точность вычислений, в частности, большие или очень маленькие, но отличные от нуля значения для входных параметров.

Часто критичным для программы является количество оперативной памяти, которое используется во время ее выполнения. Если все переменные располагаются в статической памяти, то наиболее распространенная ошибка при программировании — выход за границу массива данных во время выполнения программы, причем проявиться эта ошибка может зачастую лишь на входных данных максимального объема. Если же максимально допустимое количество данных по условию задачи таково, что программист должен использовать динамическую память, то обязательно должны присутствовать тесты, проверяющие программу на корректность работы с ними. Так, начинающий программист может неправильно выделять необходимую оперативную память во время работы программы и/или неаккуратно к ней обращаться. Также память в программе может использоваться неэффективно. В этом случае возможно подобрать тест, на котором программа учащегося работать не будет как раз в силу недостатка динамической памяти, хотя эталонная программа в подобных условиях функционирует нормально.

При работе со строками программа учащегося часто не может обрабатывать корректные с точки зрения условия задачи строки текста, длина которых превышает 255 символов. Это связано с особенностями используемого языка программирования или применением массива символов фиксированной длины вместо одной-двух символьных переменных.

Если по условию задачи требуется обработать файл, состоящий из заранее неизвестного количества чисел или строк, то для многих программ критичным оказывается наличие пробелов после последнего числа и пустых строк в конце файла. Иногда при тестировании подобная небрежность в программировании намеренно прощается, тогда при подготовке тестов приходится следить, чтобы признак конца файла располагался непосредственно за последним значащим символом входных данных, причем обязательно в той же самой строке.

4. В следующую группу можно объединить тесты, проверяющие правильность алгоритма решения задачи в целом. Они делятся на общие тесты и тесты специального вида. Последние должны проверить работоспособность программы в случае специальной организации входных данных. Например, входные данные могут быть отсортированы сначала в порядке возрастания, а затем в порядке убывания, хотя по условию определенный порядок над ними не предполагается, однако алгоритм решения может использовать сортировку входных величин. Или значения всех параметров можно сделать равными между собой. Помимо выявления специфики работы программ учащихся на таких тестах, они хороши тем, что зачастую позволяют проверяющему определить правильность выходных данных "вручную". Это уменьшает объем работы по проверке компьютерных результатов.

Общие тесты должны проверить все ветви логической схемы алгоритма, такая проверка называется *испытанием ветвей* [6]. Так, например, если задача решается с помощью алгоритма на графах, то программу следует проверить на данных, которые соответствуют связанному и несвязанному графу, графу без циклов и графу, все вершины которого принадлежат тому или иному циклу, графу с пустым множеством ребер (если такой граф допустим по условию задачи) и полному графу и т.д. Однако если алгоритм решения задачи никак не зависит от особенностей входных данных, то есть логическое ветвление как таковое в алгоритме отсутствует, то в таком случае можно ограничиться и одним-двумя общими тестами. Например, если для решения той же задачи на графах требуется лишь подсчитать степень каждой вершины (проверка наличия эйлерова пути в графе), то достаточно составить два общих теста, один из которых будет соответствовать эйлерову графу, а другой нет. Если же в программе или ее части подразумеваются два и более исходов, то каждый из возможных путей работы программы желательно проверить. К этой же группе тестов относятся невырожденные примеры входных данных, для которых программа должна определять, что решение в задаче отсутствует, если подобная возможность оговорена в условии.

5. Если решение задачи может привести к ошибочному применению эвристических алгоритмов, то необходимо подобрать такие наборы тестовых данных, на которых такие приближенные решения будут работать неверно. Иногда для этого приходится рассматривать или даже реализовывать несколько различных эвристических подходов к решению задачи, подбирая для опровержения каждого из них свой собственный тест. Например, при поиске гамильтонова цикла в графе уже на небольшом тестовом примере можно показать, что "жадный" алгоритм часто решает данную задачу неверно. Гораздо сложнее подобрать тестовый пример для случая, когда задача решается с помощью "перебора с предпочтением" и самостоятельно заканчивает свою работу, если время работы приближается к максимально допустимому. В этом случае говорят, что программа написана с отсечением по времени. Если перебор организован грамотно, то подобная программа зачастую быстро находит нужный вариант, а в дальнейшем незаконченным перебором лишь пытается доказать, что найденный вариант действительно лучший. К счастью, подобные проблемы возникают в основном при тестировании олимпиадных задач, для которых описанный прием решения считается приемлемым.

6. Если прогон программы на предыдущих группах тестов показал ее правильность в целом, то тогда следует приступить к проверке эффективности используемых алгоритмов.

Данное качество является наиболее уязвимым как в учебных программах, так и в решениях олимпиадных задач по информатике. Упрощая для себя решение задачи или его реализацию, учащиеся неоправданно увеличивают вычислительную сложность алгоритма, зачастую делая его непригодным для работы над большими, однако допустимыми по условию входными данными. Осознать данный факт самостоятельно они при этом не всегда могут. Соответственно возрастает значимость подобного рода тестирования. Если в результате выбора способа решения задачи алгоритм полиномиальной сложности был заменен на экспоненциальный, то проверить это несложно. Например, при вычислительной сложности $N!$, где N — количество входных данных, программа не сможет завершить свою работу за одну минуту даже при $N \le 15$ (точная верхняя граница работоспособности подобного алгоритма зависит от быстродействия компьютера, но при увеличении скорости процессора в 100 раз не может быть повышена более чем **на** 2). Наиболее часто подобный ошибочный выбор алгоритма происходит, когда задачу дискретной математики решают методом полного перебора вариантов вместо, например, динамического программирования, вычислительная сложность которого обычно не превышает N^3. В этом случае с помощью уже средних по объему тестовых входных данных можно определить, что работа программы не укладывается в отведенное в условии для ее работы время. При автоматической проверке это соответствует диагностике *Time limit exceeded.* Та же самая диагностика в предыдущих группах тестов скорее всего обозначает, что программа "зацикливается".

Если же применяемый алгоритм остается полиномиальным или имеет тот же порядок сложности, что и эталонный алгоритм решения той или иной задачи, отличаясь от него в фиксированное число раз, то проверить это не так просто. В этом случае тестовые данные должны быть подобраны так, чтобы эталонная программа на них заканчивала свою работу приблизительно за половину максимально допустимого по условию времени выполнения программы. То есть увеличение учащимся вычислительной сложности алгоритма в два раза можно считать допустимым. При составлении таких входных данных зачастую приходится их организовывать специальным образом, так как трудоемкость работы программы не всегда зависит лишь от количества входных данных. Подбор тестовых данных и дальнейшее тестирование программы учащегося при этом должны проводиться по крайней мере на идентичных компьютерах.

7. Последнюю группу составляют так называемые случайные тесты. Такие тесты должны отражать реальные в полном объеме входные данные. Генерация входных данных для такой группы тестов (часто достаточно одного-двух тестов) не представляет труда. Однако проверка правильности работы программы учащегося сложна именно на таких тестах. Объясняется это тем, что в данном случае трудно проверить "вручную" результаты работы программы. Зачастую для этого необходимо написать специальную программу, по сложности не уступающую решению исходной задачи. Особенно это характерно для "большого случайного теста", которым обычно завершается тестирование программы.

Тестовый драйвер

Очевидно, что без автоматизации процесса испытания программы провести ее тестирование в полном объеме практически невозможно. Прежде всего к средствам автоматического тестирования относится программа, которая пересылает данные очередного теста во входной файл или на вход проверяемой программы, записывает выходные данные, выдаваемые программой на печать, в файл и запускает программу проверки правильности выходного файла. Та же самая программа должна уметь прерывать выполнение программы учащегося по истечении отведенного на ее работу времени, а также генерировать протокол проверки программы после ее запуска на полном наборе тестов (иногда тестирование прерывается, когда на одном из тестов работа программы признана неудовлетворительной, в этом случае указывается номер первого не прошедшего теста). Такую программу называют *тестовым драйвером.*

Наиболее известный в нашей стране тестовый драйвер разработан в Санкт-Петербургском государственном институте точной механики и оптики А.Сухановым и модифицирован Р.Елизаровым. Именно он используется при проверке задач большинства студенческих и школьных командных соревнований по программированию и школьных олимпиад по информатике. Данный драйвер написан для операционной системы DOS и требует доработки в случае его использования для проверки учебных программ. На факультете ВМиК МГУ им. М.В. Ломоносова А.Черновым для московских студенческих соревнований по программированию была создана система, успешно функционирующая в операционных системах UNIX и Windows NT. Данная программа создает исчерпывающий протокол тестирования, что позволяет использовать ее и при проверке учебных задач. В СУНЦ МГУ специально для испытания учебных задач при активном участии студента мехмата МГУ И.Никокошева был создан свой тестовый драйвер. Любая из упомянутых программ может работать и сама по себе для проверки решения одним учащимся одной учебной задачи.

Рассмотрим на примере последнего из упомянутых тестовых драйверов условия успешного их применения. Для проверки той или иной программы с помощью системы тестов прежде всего необходимо создать набор тестовых входных данных согласно принципам, сформулированным выше, и записать их в файлы с идентичными именами. Обычно последние 1—2 символа в имени файла обозначают его порядковый числовой номер в системе тестов. В правильно составленном наборе тестов их сложность с увеличением номеров должна возрастать. Затем, для того чтобы "научить" тестовый драйвер проверять решение какой-либо задачи, необходимо создать определенный конфигурационный файл. Минимальная информация, которую должен содержать такой файл, следующая:

1) название задачи;

2) способ ввода и вывода данных проверяемой программой (консоль или файл, в последнем случае указывается еще и имя входного и выходного файлов согласно условию задачи);

3) количество тестов (при отсутствии данной информации осуществляется проверка программы на всех тестах, находящихся в текущем каталоге);

4) шаблон для имен тестовых входных файлов;

5) шаблон для имен файлов правильных ответов (иногда, например в случае неоднозначных правильных ответов, такие файлы могут отсутствовать);

6) программа для проверки правильности полученного ответа (возможно, как указание одной из стандартных программ, так и созданной специально для проверки данной задачи);

7) время работы проверяемой программы в секундах на одном тесте.

Указанная система тестирования содержит несколько встроенных алгоритмов для проверки правильности полученных ответов. Прежде всего это так называемые *компараторы* — программы, сравнивающие два файла между собой на совпадение. В нашем случае это файл, в котором записан вывод проверяемой программы для одного теста, и файл с правильным ответом на этот тест. Файлы с правильными ответами часто могут быть получены лишь при наличии эталонной программы с решением исходной задачи.

Компаратор **numbers** проверяет, что в обоих сравниваемых файлах содержится один и тот же набор чисел, причем в том же самом порядке. Вид разделителей между числами (пробелы и символы перевода строки) и их количество, в том числе и после последнего числа, при этом не существенны. Если числа не являются целыми, то сравнение производится с точностью, определяемой количеством цифр после точки в каждом из чисел правильного ответа. При сравнении текстовых файлов используются два компаратора: **exactly** — сравнивающий два файла на точное совпадение и **tokens** — игнорирующий при сравнении пробельные символы в конце каждой строки и символы перевода строки в конце файла. Кроме того, программа **sets** позволяет сопоставлять между собой множества чисел, расположенных в анализируемых файлах. То есть здесь не учитывается порядок расположения чисел в ответе.

К сожалению, заранее предусмотреть все возможные способы анализа полученных результатов невозможно. Прежде всего потому, что ряд задач допускает не единственный вариант правильного ответа. Поэтому в ряде случаев для проверки правильности результатов тестирования требуется создавать алгоритм, который по своей сложности не уступает решаемой задаче. Иногда он сводится к *реверсивной проверке*, то есть к подстановке ответа в исходную систему, например, систему уравнений или систему шифрования данных. Возможность такой проверки позволяет провести тестирование и без наличия эталонной программы. При анализе же найденного кратчайшего способа выхода из лабиринта, который может не совпадать с путем, найденным эталонной программой, требуется проверить, что предложенный путь, во-первых, для заданного лабиринта существует, а во-вторых, является кратчайшим. Для проверки последнего качества предложенного пути удобно сравнить его длину с длиной пути из имеющегося правильного ответа.

Если проблемы с проверкой ответов на корректность решены, то любая программа с помощью тестового драйвера может быть протестирована с помощью имеющейся для рассматриваемой задачи системы тестов. Сначала текст программы будет проанализирован на безопасность (обычно в учебных и олимпиадных задачах запрещается использовать расширенную память, защищенный режим процессора, читать и записывать вектора прерываний, использовать сетевые средства, создавать каталоги и вспомогательные файлы, отличные от упомянутых в условии задачи), а затем откомпилирован. Тип компилятора определяется по расширению файла с решением.

Если компиляция не была осуществлена успешно, то в протокол проверки в качестве типа ошибки будет занесено *Compilation Error*. Если же исполняемый файл для проверяемой программы был создан, то тестовый драйвер последовательно запускает его на каждом из тестов для указанной задачи. Если код завершения программы после ее работы на тесте равен нулю, то запускается программа проверки выходных данных проверяемой программы. Результаты тестирования (*Ok* — в случае успешной проверки или одна из описанных выше диагностик — в случае некорректной работы программы) выдаются на экран и заносятся в файл протокола, который можно посмотреть по окончании тестирования. В протокол могут быть помещены в том числе и входные данные теста, на котором работа программы признана ошибочной.

Так может быть завершено тестирование учебной или олимпиадной задачи в полуавтоматическом режиме. Большей автоматизации можно достигнуть, используя командные файлы для сборки программ учащихся в текущий каталог и запуск их по очереди на проверку. Однако наиболее естественное направление в автоматизации тестирования должно использовать современные сетевые средства передачи файлов и пересылки результатов их проверки непосредственно учащимся.

Описанная система имеет так называемую *открытую архитектуру*, то есть позволяет добавлять к ней произвольное число используемых компиляторов, а также задач, проверка которых в дальнейшем будет производиться с помощью этого же тестового драйвера.

Система "клиент — сервер" для организации компьютерного задачника

Современная визуальная технология конструирования программ, а также создание мощных процессоров баз данных, например BDE [8], сделали возможным написание программы-оболочки для компьютерного задачника без привлечения коллективов профессиональных программистов. В подобной программе, реализованной в сетевой архитектуре "клиент — сервер", обработку поступающих запросов клиентов и передачу им результата на низком уровне берет на себя стандартный процессор баз данных, который занимается установкой соединения между СУБД и приложением. Запросами со стороны клиента (ученика) в нашем случае являются заявка на получение условия той или иной назначенной учащемуся задачи и посылка на проверку файла с текстом ее решения на том или ином языке программирования.

В СУНЦ МГУ под руководством автора был создан сетевой программный комплекс для проверки учебных и олимпиадных задач в автоматическом режиме и тестирования знаний учащихся по теоретическим основам

информатики. Непосредственное участие в его реализации принимал студент мехмата МГУ Ю.Спорынин. Работа с сетью как в программе сервера, так и в программах клиентов осуществляется через протокол TCP/IP. Это позволяет использовать данную систему не только в локальной сети, но и в сети Интернет.

Система успешно применяется в учебном процессе с 1999/2000 уч. года. На нее возложены функции автоматической проверки заданий практикумов по программированию в СУНЦ МГУ, проведение олимпиад в режиме *on-line* (по принципам студенческих командных соревнований по программированию) и тестирование знаний учащихся по любым предметам. Система постоянно пополняется. Однако набор задач и тестов по информатике, проверка которых уже автоматизирована в настоящее время, позволяет считать ее компьютерным задачником, пригодным для широкого использования в преподавании информатики.

Естественно, что потребность в подобных системах возникает и в других центрах, в которых уделяется большое внимание обучению программированию. На взгляд автора, усилия специалистов в данной области должны быть объединены с целью автоматизации проверки широкого класса задач в области информатики. Причем привязка к фиксированной оболочке для компьютерного задачника здесь не является обязательной. Достаточно использования совместимых тестовых драйверов, конфигурация которых вместе с наборами тестовых данных и условиями задач и несет в себе львиную долю необходимой для автоматического тестирования информации.

Опыт использования компьютерного задачника в учебном процессе и при подготовке к олимпиадам позволяет утверждать, что необходимость подобных систем для рядовых преподавателей информатики и программирования невозможно переоценить. Они сделают труд преподавателя более квалифицированным, повысят эффективность и интенсивность занятий в несколько раз, позволят увеличить интерес учащихся к данному предмету.

Литература

1. *Грис Д.* Наука программирования. М.: Мир, 1984.
2. *Шень А.* Программирование: теоремы и задачи. М.: МЦНМО, 1995.
3. *Пинаев В.Н.* Олимпиады по информатике. Информатика, № 22/2001.
4. *Окулов С.М., Пестов А.А., Пестов О.А.* Информатика в задачах. Киров: Вятский государственный педагогический университет, 1998.
5. *Пейперт С.* Переворот в сознании: дети, компьютеры и плодотворные идеи. М.: Педагогика, 1989.
6. *Ван Тассел Д.* Стиль, разработка, эффективность, отладка и испытание программ. М.: Мир, 1985.
7. *Андреева Е., Фалина И.* Системы счисления и компьютерная арифметика. М.: Лаборатория базовых знаний, 2000.
8. *Дарахвелидзе П.Г., Марков Е.П.* DELPHI 4. С.-Петербург: БХВ, 1998.

Вопросы, задания и контрольные работы для начинающих программистов

В.П. Гладков, А.П. Шестаков,
г. Пермь

Продолжение. Начало в № 20, 33/2001

3. Арифметические выражения

3.1. Какие записи являются выражениями в Паскале?

а) `5;`
б) `y;`
в) `sin(x);`
г) `5 + ysin(x).`

3.2. Почему нельзя опускать знак умножения?

3.3. Почему аргументы функции должны быть заключены в круглые скобки?

3.4. Расставьте круглые скобки в следующих выражениях Паскаля, чтобы показать последовательность их вычисления:

а) `a * b * 2 + 3.456/y;`
б) `a + b * c - d/f;`
в) `a + b + c + d + e;`
г)
```
-a + sin(abs(3 * (b + c)) -
  (e - d)/(2 * d * c)) - 14;
```
д) `b mod a + c div b * s;`
е) `-3 + 7 div 2 mod 7/2 - trunc(cos(exp( - 1)));`
ж) `a + b - c;`
з) `a + b * c;`
и) `a - b * c - d;`
к) `a - b * -k.`

3.5. Какие круглые скобки в приведенных выражениях можно снять, не изменив порядка их вычислений?

а) `(a + b)/c;`
б) `a + (b/c);`
в) `a/(a * b);`
г) `a + (- 3) * (s * d/r) - (a + 25);`
д) `(a + b) + ((c + d) * 2 * e);`
е) `(x1/x2) * y;`
ж) `(sqrt(p) * q)/r;`
з) `(((a - b) - c) - d) - e;`
и) `a - (b - (c - (d - e)));`
к) `((a - b) - (c - d)) - e;`
л) `(a - (b - c)) - (d - e);`
м) `(a * b) div c;`
н) `b + (a - (c/3));`
о) `(a * (b/(c * (d/(e * f))))).`

3.6. Сколько операций выполняется при вычислении следующих выражений? Как сократить количество операций?

а) $(a + 1/2) \cdot (b + 7/10) - 3/4$;
б) $ax^4 + bx^3 + cx^2 + dx + e$.

3.7. Перечислите правила записи выражений.

3.8. Перечислите правила вычисления значения выражений.

3.9. Для чего нужны скобки в арифметическом выражении? Можно ли обойтись без них в выражении $(2 + a) \cdot (3 - b)$ и вообще?

3.10. Запишите следующие выражения на Паскале:

а) $\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-a)^2}{2\sigma^2}}$;

б) $\sqrt{\frac{\pi}{8}} \sqrt[5]{\frac{\sqrt{\alpha^2+\beta^2}-\beta}{\alpha^2+\beta^2}}$;

в) $x_{m_{n_k}} y^{m^{n^k}}$;

г) $2^{1+\sqrt{x}}$;

д) $\sqrt[5]{1+x^8}$;

е) $\sqrt[x]{x^7+7^x}$;

ж) $10^4\alpha - 3\frac{1}{5}\beta$;

з) $\frac{ab}{c} + \frac{d-c}{ab}$;

и) $\log_2 \frac{x}{6} + \cos^5 \frac{(x+y^7)^3}{2a^3b^5} - \sin^4 x^7$;

к) $\arcsin x$;

л) $\dfrac{\left(\frac{\sqrt{a}+\sqrt{b}}{2b\sqrt{a}}\right)^{-1} + b\left(\frac{\sqrt{a}+\sqrt{b}}{2a\sqrt{b}}\right)^{-1}}{\left(\frac{a+\sqrt{ab}}{2ab}\right)^{-1} + \left(\frac{b+\sqrt{ab}}{2ab}\right)^{-1}}$;

м) $\dfrac{\left(\frac{(a+x)^3}{ax} - 4\right) \cdot \left(\frac{(a-x)^{a+x}}{\frac{a}{x}+y}\right) \cdot (a^x - b^x)}{\left(a^{\sin^3 x^5} + y - ax^6\right) \cdot \left((a+x)^2 - ax\right) \cdot \left((a-x)^{x+b} + ax\right)}$.

3.11. Запишите в общепринятой форме следующие выражения Паскаля:

а) `sqrt(1 + sqr(x)/abs(a + b * x));`
б) `a + b/(c + d/(e + f/(g * h)));`
в) `(((a * x + b) * x + c) * x + d) * x + e;`
г) `(x + y)/a[1] * a[2]/(x - y);`
д)
```
ln(abs(y - sqrt(abs(x))) *
  (x - y/(z + sqr(x)/4)));
```
е)
```
sin(sqr(abs(cos(x)/exp(3) +
  sin(y)/cos(y - 1)) +
  abs(3 * (d + 2 * s) + 4))/2)/5;
```
ж) `exp(0.25 * ln(x/y)) + exp(4 * ln(r));`
з)
```
sqrt(sqrt(sqrt(sqrt(sqrt(7))))) +
  sqr(sqr(sqr(sqr(sqr(sqr(8))))));
```

и)
```
sqr (sqrt (sqr (sqrt (9)))) -
   sqrt (sqr (sqrt (16)))  +
   sqrt (sqr (sqrt (sqr (25)))) -
   sqr (sqrt (sqr(36)));
```
к) `1 * 2 * 3 * 4 * 5 * 6/sqr(1 * 2 * 3).`

3.12. Всегда ли правильно записанное на Паскале выражение имеет смысл?

3.13. Проверьте записи выражений, найдите выражения с семантическими ошибками:

а) `maxint - trunc(sqrt(16 + a));`
б) `12 + maxint;`
в) `random(10) + a - int(a/b) * b;`
г) `sin(pi/4) + sqr(cos(2 * pi/sqr(sqr(sqr(2)))));`
д) `trunc(round(8));`
е) `(2 + 3) * 25 div (9 mod 3 div 2 mod 10);`
ж) `pi/trunc(pi) + round(pi)/int(pi);`
з) `chr(ord('a')) + 'a'.`

3.14. Найдите выражения с синтаксическими ошибками:

а) `b + asin(x)/k;`
б) `maxint - sqrty * 2 * d;`
в) `a + b * (* c - d * e);`
г) `(a + b)/2/a + b/(4 - sqr(2));`
д)
```
sin(abs(sqr(x * (a + cos(3 + 7 * x) +
   5 * f)) - 2 * (sin(2) + 2 - 2/(a + b))).
```

3.15. Определите значения выражений:

а) `16 div 4 * 2;`
б) `16 + 4 * 2;`
в) `16 div (4 * 2);`
г) `1 + 19 mod 5;`
д) `(1 + 19) mod 5;`
е) `3 div 10 + 25 mod 5;`
ж) `1 + 25 div 5 mod 2;`
з) `(1 + 25 div 5) mod 2;`
и) `k mod 10 + k div 10 mod 10 +`
`k div 10 div 10 mod 10` (при $k = 123, 987, 54$).

3.16. Определите тип выражений:

а) `1 + 0.0;`
б) `120/6;`
в) `sqr(4);`
г) `sqr(4.0);`
д) `sqrt(16);`
е) `sqrt(4.0);`
ж) `sin(0);`
з) `succ(-1);`
и) `trunc(pi);`
к) `sqr(2) + trunc(12/7).`

3.17. Какие из следующих выражений синтаксически правильны (совместимы по типу)? Определите типы этих выражений, исходя из следующих описаний переменных:

```
var x, y, z: real; i, j, k: integer;
```

а) `x + y * i;`
б) `i mod (j + y);`
в) `i + j - k;`
г) `i div j + x;`
д) `(x + y)<(i + j);`
е) `k - trunc(x * i);`
ж) `i * x + j * y;`
з) `(x < y) and (y < z);`
и) `x = i.`

3.18. Определите, эквивалентны ли записи выражений а) и б) на Паскале и запишите их в общепринятой форме:

а) `a + c * d/b + e/(f * g);`
б) `a + c * d/b + e/f/g.`

3.19. Установите, что будет получено при выполнении каждого из следующих операторов присваивания при заданных описаниях и известном значении X:

```
var X, x1, x2, x3: word;
```

а) `x1 := X and 31;`
б) `X := X div 32;`
в) `x2 := X and 27;`
г) `X := X div 128;`
д) `x3 := X and 7.`

3.20. Приводятся выражения, записанные в общепринятой форме. Под ними даны выражения, записанные по правилам Паскаля. Установите, в каких случаях выражение, стоящее снизу, является эквивалентом своего верхнего соседа, или укажите ошибки в записи:

а) $\dfrac{b+\sqrt{b^2-4ac}}{2a}$
`(b + sqrt(sqr(b) - 4 * a * c))/2 * a;`

б) $\cos^2 x$
`sqrt(cos(x));`

в) $\dfrac{a}{c} \cdot \dfrac{b}{d}$
`ab/cd;`

г) $\dfrac{a}{c} \cdot \dfrac{b}{d}$
`((a/c) * b)/d;`

д) $\sin x + \cos\dfrac{y}{2}$
`sin(x) + cos(y/2);`

е) $\dfrac{x+1}{y+1}$
`x + 1/y + 1;`

ж) $\sin\dfrac{x+y}{2}$
`sin(x + y/2);`

з) $\dfrac{a+\sin(x)}{\sqrt{a^2+x^2+1}}$
`a + sin(x)/sqrt(sqr(a) + sqr(x) + 1);`

и) $\dfrac{2e^{-|x+y|}}{x^2+y^2}$
`2exp(- abs(x + y)/(sqr(x) + y * y);`

к) $\dfrac{\sin^2 x}{y_1+0{,}5}$
`sqr(sin(x))/(y1 + 0,5);`

л) $x \cdot 10^{-2}$
`xE - 2;`

м) $7{,}1 \cdot 10^{-z}$
`7.1E - z;`

н) $(7{,}3 \cdot 10)^{-4}$
`7.3E - 4;`

о) $3{,}6 \cdot 10^{-\sqrt{2}}$
`3.6E - sqrt(2).`

Продолжение следует

Роботландский сетевой университет глазами учителей и школьников

А.А. Дуванов,
руководитель Роботландского университета,
г. Переславль-Залесский

Роботландский университет — это сетевая школа при негосударственном образовательном учреждении дополнительного образования "Роботландия+" (г. Переславль-Залесский), в которой обучаются учителя и дети, как правило, в рамках одной "команды": учитель — руководитель команды плюс его подопечные — школьники. Роботландия плодотворно сотрудничает с "Информатикой", и постоянные читатели этой газеты, конечно, знакомы с материалами университета и его продуктами, которые распространяются среди подписчиков с 10%-ной скидкой (купон скидок вы найдете на с. 18).

С точки зрения учителя, сетевое обучение — это не совсем обычные по форме курсы повышения квалификации: учителя получают знания по современным предметным разделам, методические рекомендации, а в процессе обучения адаптируют полученный материал в среду своих учеников, которые вместе с ними обучаются на курсах. Так как университет работает совместно с Российской академией повышения квалификации и переподготовки работников образования, то, кроме удостоверения Роботландии, учителя получают также и удостоверение государственного образца о прохождении курсов повышения квалификации.

Для детей занятия в рамках сетевого сообщества — это не только глубокое освоение любимых тем школьных предметов, а также разделов, еще не успевших войти в школьные планы, но и спортивный азарт конкурсов, продуктивное общение со сверстниками, наглядные примеры использования сетевых технологий.

Пятилетний опыт проведения сетевых курсов Роботландского университета однозначно выявляет высокий интерес к такой форме обучения как у учителей, так и у школьников.

"В чем я вижу плюсы такого типа обучения?

Во-первых, обеспечено единство методического, программного и организационного обеспечения. Все вопросы тщательно "разжеваны", что значительно облегчает работу учителя. Однако остается значительное поле для творчества. После освоения методики можно гибко подстроить ее под специфику своей школы.

Во-вторых, решается проблема обеспечения учебными пособиями учащихся и учителей.

В-третьих, стиль обучения с постоянным (хотя и сетевым) общением участников Роботландского университета в ходе конкурсов и перекрестных проверок создает творческую атмосферу, заинтересованность учащихся и дополнительные положительные мотивации.

Немаловажным в этой форме является и то, что учитель тоже повышает свою квалификацию и получает об этом свидетельство государственного учреждения. А это — и экономия школьных средств, и рост тарифного разряда учителя".

Виктор Фомич Клятченко,
учитель информатики средней школы № 2, г. Люберцы

"Я — один-единственный учитель информатики в школе. Веду и уроки, и факультативы. Если с содержанием уроков более или менее все понятно и отработано, то с подбором содержания факультатива постоянно возникали проблемы. Что отобрать из огромного многообразия компьютерных технологий, как грамотно построить курс? А главное, где взять время и силы, чтобы постоянно быть в курсе всех новинок в компьютерном мире? К кому обратиться за советом? Где посмотреть чужие работы? Работы других детей? Многие из этих проблем решились после того, как я выбрала два курса в Роботландском университете".

Елена Викторовна Болгарина,
учитель информатики гимназии № 99, г. Екатеринбург

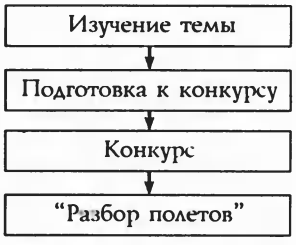
Как происходит обучение

От конкурса до конкурса. А в промежутках — обычные для факультатива занятия детей с учителем и не совсем обычные курсовые обсуждения в рамках постоянно действующей сетевой конференции.

"Мне понравилось дистанционное обучение. Но было очень трудно. Задачи были сложные, мы очень подробно их разбирали. Когда приходили конкурсные задачи, то забывали обо всем остальном. И решали, решали, решали... Дома, в школе, все вместе, по одному. Хотя и было трудно, но было интересно".

Марина Галушко,
9-й класс, г. Благовещенск

Процесс обучения можно изобразить схемой.



Из таких элементов строится цикл обучения: на фоне сетевой электронной конференции дети изучают тему, потом готовятся к конкурсу (решают задания, подобные конкурсным, или изучают готовые проекты), затем — конкурс. Конкурс — это всегда праздник и эмоциональный всплеск с турнирными таблицами, награждением команд-победителей.

Большое внимание уделяется перекрестной проверке детьми работ друг друга. Дети из Находки проверяют работы детей из Перми, Самары и Челябинска. Перекрестная проверка — важный элемент обучения. Проверяя "чужие" работы, дети по-другому начинают относиться к своим решениям, и в ито-

ге от такой "учительской" функции материал темы усваивается гораздо лучше.

Преподаватели университета

Преподаватели университета — авторы курса информатики для младших школьников, известного под названием "Роботландия": А.А. Дуванов, Ю.А. Первин, Я.Н. Зайдельман. Курс "Английский с компьютером" ведет Н.А. Прохорова, преподаватель английского языка, автор методической системы "Magic Land".

В последнее время Роботландия стала привлекать к педагогической работе университета талантливых школьных учителей, которые наиболее ярко проявили себя при многолетнем сотрудничестве в качестве студентов роботландской школы. Так, в прошедшем учебном году успешно провел курс 41 "Введение в Интернет" Анатолий Владимирович Бычков из Барнаула, а в новом учебном году ряды кураторов пополнит Алексей Владиславович Рудь из Снежинска, он будет вести курс 31 "Азы программирования".

"О кураторах особо. Такие разные и такие жутко интересные люди. Мы искренне рады, что нам буквально посчастливилось в жизни пообщаться с такими интеллигентными людьми. Остается только пожалеть, что знакомство наше и общение только электронное".

Марина Анатольевна Важенина,
директор Качканарского межшкольного компьютерного центра

"Приятно отметить чуткое отношение преподавателей к своим учащимся, высокую профессинальную компетентность, положительную эмоциональную атмосферу, созданную на курсе! Удивляюсь терпению куратора, который терпеливо, шаг за шагом, привел нас к первому конкурсу проектов (кроме групповых рассылок, мы получали множество сообщений с ответами на самые разные вопросы, и ни один не остался без внимания)! ".

Ольга Федоровна Брыксина,
учитель информатики педагогического колледжа, г. Чапаевск

Особенности Роботландской школы

Курсы через Интернет как форма заочного обучения распространены сейчас довольно широко. Однако студенты, настроенные на повышенную эффективность сетевой школы, нередко испытывают разочарование: обучение строится по старым канонам бумажной почты, повышается скорость доставки материала, а формы и методы учения остаются прежними — "варка" в собственном соку и эпизодическое общение с педагогом-куратором.

Роботландская школа устроена по-другому: учитель и курируемый им детский коллектив образуют в рамках курсов активный сетевой узел — команду, которая связана с куратором университета и другими командами в едином учебном информационном пространстве. Эта связь крепится активной совместной деятельностью.

"Роботландский сетевой университет — не просто громкое название. Это — обучение, отлаженные методики, полезное и, что очень важно, интеллигентное общение детей и взрослых.

Мы, школа № 124 г. Снежинска, 3 учителя и около 30 детей, учились на четырех курсах: "Азы", "Буки программирования", "Конструирование", "Интернет-конструирование". Что выгодно отличает все курсы Роботландии от других известных нам проектов (выражаю общее мнение всех троих руководителей и многих коллег—студентов университета):

— Постоянный и очень доброжелательный контакт с руководителями — авторитетами школьной информатики.

— Общение с коллегами.

— Полные методические материалы.

— Соревновательный момент (регулярные турниры). Азарт детей и взрослых.

— Перекрестные проверки. Оценивание работ соперников и возможность "увидеть себя со стороны", грамотно апеллировать.

— Изготовление полезных продуктов для школы.

— Фундаментальность и "классичность" материалов, идущих от руководителей. Строгая научность, такт, идеальная грамотность.

В общем, много положительных эмоций весь год — приятно работать!".

Алексей Владиславович Рудь,
учитель информатики гимназии № 99, г. Снежинск

Характерные черты Роботландской школы:

1. Совместное обучение учителя и школьника в рамках одной команды.

Дидактические, методические и организационные курсовые материалы дополняются активной живой работой учителей на местах, что, конечно, во много раз повышает эффективность сетевого обучения.

"Я, как директор образовательного учреждения, хотела бы отметить, что для самообразования учителя информатики занятия в Роботландском университете — просто подарок. Пятеро из наших учителей уже попробовали себя здесь. Кто-то более, кто-то менее успешно, но можно быть уверенным: ни для кого эти занятия не прошли даром — новые знания, другие требования и сами условия работы, безусловно, обеспечивают рост учителя и с методической стороны, и в плане получения новых научных знаний по предмету. Удостоверения о повышении квалификации, которые высылаются учителям по окончании обучения, — тоже не лишнее, особенно при аттестации".

Марина Анатольевна Важенина,
директор Качканарского межшкольного компьютерного центра

2. Турнирный цикл обучения.

Обучение строится как цепочка краткосрочных турнирных проектов. Вся теория, излагаемая в курсе, подчинена одной цели — применить ее на практике для создания работы, которая будет участвовать в конкурсе. Мотивировка обучения смещается из области абстракций на практику, что является мощным побуждающим началом активной деятельности ребенка.

3. Моделирование коллективной деятельности.

Работая над проектом, школьники объединяются в рабочие группы, в которые набираются соисполнители с определенными функциональными обязанностями (сценарист, редактор, художник, программист...). Таким образом, моделируется работа творческого коллектива, в котором каждый участник выполняет свою "профессиональную" миссию.

"Мне понравился курс "Азы программирования". Сначала мы думали, что знаем все о Кукараче, но оказалось, что знали совсем чуть-чуть. Но зато теперь мы можем решить задачу, используя рекурсивную пружину. А это совсем не просто. Но самое интересное, что нам было здорово вместе. Мы научились решать задачи сообща. Один предложит идею, а все вместе додумаем решение до конца".

Дима Устич, 9-й класс, г. Благовещенск

"Кажется, что эту задачу решить просто невозможно, а потом каждый говорит то, о чем думает, появляется интересная мысль и вырисовывается ход решения. Это очень здорово — решать трудные задачи вместе. А сколько у всех радости, когда задача уже решена и Кукарача делает именно то, что мы хотим".

Юля Бердышева, 9-й класс, г. Благовещенск

"Ребятам интересно, когда они вместе делают какое-то дело. Это то самое общее дело, вокруг которого формируется коллектив: есть дело — есть коллектив. И совсем не важно, что все они из разных классов. У них уже сформировался свой микромир.

Не секрет, что сегодня престиж учебы падает. И учителя всеми силами стараются его поднять. Поэтому особенно важно, когда общее дело связано с получением новых знаний. Именно такую возможность дает Роботландский университет. И не просто изучить коллективно какое-то направление в информатике, но иметь возможность показать свои знания, увидеть, чему научились другие, посоревноваться и получить в результате готовый продукт. И не только в рамках школы и города. Это тоже очень увлекает.

На городском методобъединении учителей я рассказывала о Роботландском университете, и ребята помогали мне подготовить карту, показывающую географию курсов Роботландского университета, на которых мы обучаемся. На ребят это произвело сильнейшее впечатление. Оказывается, Роботландский университет охватывает всю территорию России и даже выходит за ее пределы.

Роботландский университет дает нам не только широчайшие возможности в учебной деятельности, но и предоставляет широкий полигон для воспитательной работы".

Надежда Владимировна Евладова, учитель информатики, г. Благовещенск Амурской области

4. Реальная практическая польза детских проектов.

"Мы делали задания для первоклассников и семиклассников. Они потом на уроках их выполняли. Мы помогли ученикам и учителям. Мне очень понравилась эта работа".

Алена Жуковская, 3-й класс, г. Благовещенск

"Сегодня мы участвовали в научно-практической конференции по информатике и в номинации "Web-страница" заняли бесспорное 1-е место!

Выставили работу по проблеме макроэтики, к работе отнеслись как к конкурсной работе Роботландского университета, потому все было по правилам и обоснованно.

Если честно, то большая часть работ этой конференции была бы снята с конкурса Роботландского университета как несостоятельная: 1,3 мегабайта "весом", при этом полезности особой нет. Так жаль, что многие учителя считают, что плох сайт, в котором нет никаких движений (к нам было такое замечание), а разноцветные страницы одной работы — красиво! Но мы чувствовали себя почти профи и могли ответить на все вопросы и толково объяснить, почему движение на сайте — плохо и для чего проводится оптимизация графики. Ведь год назад я и мои дети ничего этого не знали, а сегодня так уже не скажешь".

Галина Николаевна Малько, учитель информатики, г. Барнаул

Проекты, предлагаемые на курсах, как правило, имеют прикладное значение. Школьники с полным правом гордятся тем, что труд их не ложится под сукно и не пылится на выставках, а реально используется на школьных уроках и не только в своей родной школе, но и в других школах страны, работающих на роботландских курсах.

Во-первых, обучение, с точки зрения ученика, уходит на второй план, являясь для него не мотивом, а как бы побочным эффектом деятельности.

Основной мотив — создание законченного компьютерного приложения, имеющего практическую ценность (компьютерная сказка, обучающая программа, справочник, тест, игра, сайт).

Во-вторых, ученик овладевает компьютерным инструментарием в комплексе, параллельно осваивая разные способы создания компьютерных объектов, имеет возможность проводить не отсроченные по времени аналогии, сравнивая типы создаваемых информационных блоков, интерфейсы программ, методы проектирования.

В-третьих, конечный продукт, созданный детьми в интегрированной среде конструирования, приближен по внешнему виду к профессиональным продуктам. Это достигается перекладыванием на систему основных операций по созданию интерфейса проектируемого приложения. Таким образом, ребенок получает внешне привлекательный продукт, не отвлекаясь от поставленной задачи на технические детали и затрачивая на работу минимальное время.

В-четвертых, ученик во время работы осваивает технологии проектирования сложных систем. За основу, например, может быть взята технология "сверху вниз" (метод постепенной детализации).

В-пятых, появляется возможность естественным образом моделировать работу детей в творческих коллективах, работающих над одним большим проектом.

"Многие ребята теперь мои помощники — создатели web-сайтов. Кроме того, часть ребят начала самостоятельное дальнейшее изучение HTML. Во всяком случае, результаты очень даже впечатляющие: учениками созданы проекты по краеведению, виртуальный музей братьев Каменских, они занимают призовые места в различных Интернет-конкурсах".

Людмила Николаевна Олейник, учитель информатики школы № 92, г. Пермь

"Хочется отметить исключительное внимание сотрудников университета к разработке учебных сред (таких красивых и нескучных). Приятны их педагогическая проработка, продуманность целевых функций. Включение в программы заданий творческого характера помогло нам, учителям,

вырастить победителей городских и областных олимпиад по информационной культуре и программированию".

Марина Анатольевна Важенина, директор Качканарского межшкольного компьютерного центра

"Материал курса 42 и знания, которые он дает, весомейшего значения. Уже применили их, работая над другими проектами, оформляли визитки команд, участвовали в проекте "My Sweet Valentine", оформляли школьные задания. А сколько нового узнала я сама! Обязательно продолжим обучение на следующий год на курсе "Web-программирование".

Ирина Ивановна Коробкова, учитель информатики средней школы № 50, г. Челябинск

"В прошлом году я обучалась на курсе "Английский с компьютером" в индивидуальном режиме. Мне понравилось, что в программе "Magic Land" можно не только использовать готовые уроки и корректировать их по желанию, но и создавать собственные задания. Причем учитель — новичок в компьютерном деле с этим успешно справляется".

Светлана Ивановна Колишева, учитель английского языка детской компьютерной школы, г. Сургут

5. Перекрестные проверки работ.

"Мне очень понравилось учиться в Роботландском университете, потому что мы не только научились решать задачи, но и увидели, как это делают другие. Помню, нам очень понравились решения ребят из Снежинска. Мы очень тщательно тестировали их решения, а ошибок все равно не было. Мы даже обнаружили ошибки в своих решениях, когда проверяли эту команду".

Дима Комлев, 9-й класс, г. Благовещенск

Перекрестные проверки работ не внутри одной школы, а благодаря Интернету среди однокурсников, распределенных по всему географическому пространству России и стран СНГ, — это дополнительный, очень важный этап обучения. Школьники знакомятся с работами друг друга очень подробно. Выполняя учительскую функцию, дети учатся искать ошибки, пользоваться формальными оценочными критериями.

Разработке оценочных критериев в университете уделяется повышенное внимание. При этом особое внимание учителей и школьников обращается на принципы формализации плохо формализуемых правил.

В качестве примера можно указать построение числовой характеристики для оценки лабиринта игры "Мудрый крот". Эта оценка представляет собой сумму взвешенных баллов за отдельные качества:

— наличие "ключевых" мест,
— сложность "ключевых" мест,
— новизна "ключевых" мест,
— разнообразие "ключевых" мест,
— количественная простота,
— красота лабиринта.

Такое "измерение" лабиринта, наряду с понятием "ключевое место" ("изюминки", разгадав которые можно считать задачу принципиально решенной), дает в руки проверяющему формализм, который позволяет осмысленно исследовать работу и объективно ее оценить.

Другим примером оценочного формализма может служить способ оценки дизайна сайта, когда проверяющий выставляет балл, ориентируясь не на субъективное мнение ("нравится — не нравится"), а на объективные свойства проверяемой работы с учетом четкого представления о том, почему обнаруженное свойство является погрешностью и как сильно оно влияет на потребительские качества продукта (см. "Назаметки Сидорова", № 16/2001).

"Меня сразу покорили фундаментальность в подготовке материалов, оптимальность структуры курса, стиль общения, форма контроля знаний и, в самом хорошем смысле, "дотошность" куратора при проверке и анализе работ учащихся. Насколько Юрий Абрамович внимательно изучал работы слушателей курсов! Все это завершалось проведением глубокого сравнительного анализа мнений учащихся-коллег. И каждый раз я с волнением отправляла на суд Юрия Абрамовича свои вариации на тему, и как приятно было получить признания мэтра!

Следует отметить как безупречность технологии самого процесса обучения, так и выбранной формы проверки достижений. Полное обеспечение учебно-методическими материалами, рекомендациями по выполнению конкурсного проекта и перекрестной проверки работ — все это тоже Роботландский университет! Организаторы университета прозорливо используют дух соперничества в лучших традициях отечественной педагогики, внеся в нее свое слово. Моим студентам, как будущим педагогам, крайне полезно было поучаствовать в перекрестной проверке, почувствовать свою значимость, весомость собственного мнения".

Ольга Федоровна Брыксина, учитель информатики педагогического колледжа, г. Чапаевск

6. Развитые горизонтальные связи.

Помимо связи с куратором, команды очень активно общаются друг с другом. Это происходит в следующих формах:

— курсовая электронная конференция;
— перекрестные проверки работ;
— коллективные сетевые обучающие игры;
— непосредственная электронная переписка.

"Все возникающие трудности очень оперативно решаются куратором. При этом всегда можно получить консультацию и по вопросам, не касающимся напрямую курса. По электронной почте задать вопрос очень просто: это можно сделать и на лист рассылки, и непосредственно куратору. А куратор очень оперативно и в самой тактичной форме не только ответит автору, но и похвалит за умный вопрос и предложит всем студентам попытаться найти на него ответ.

Очень нравится, что учителя делятся опытом работы.

Я не устаю удивляться и восхищаться, как много разных, интересных и подчас неожиданных подходов к решению одной и той же проблемы могут предложить коллеги-учителя".

Людмила Николаевна Олейник, учитель информатики школы № 92, г. Пермь

7. Повышенное внимание культуре и этике общения, а также способам изложения своих идей в виде технических заданий, описаний и компьютерных приложений.

"Учащиеся через проект учатся этике электронной переписки, правилам работы в сети.

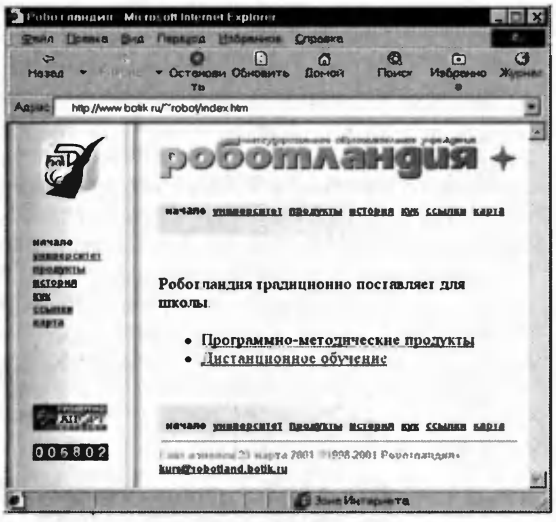
Общение происходит со сверстниками, а не с абстрактными пользователями, причем общение тематическое, что несет познавательный аспект (например, проект "Малая Родина")".

Олег Викторович Карпов,
учитель информатики муниципального лицея,
г. Орехово-Зуево

"На всех курсах университета царит атмосфера доброжелательности. Здесь вы найдете друзей и единомышленников. Ведь университет несет не только технические знания, но и воспитывает. Воспитывает по-настоящему гуманистическое отношение друг к другу, к окружающим людям, учит общению".

Анатолий Владимирович Бычков,
учитель информатики школы-лицея № 73,
г. Барнаул

8. Основной канал связи — электронная почта.



Несмотря на существующую онлайновую поддержку дистанционных курсов (www.botik.ru/~robot), основным каналом связи остается электронная почта. Электронная почта — это одно из самых экономных средств сетевой связи. Небольшие финансовые затраты позволяют считать этот сервис по-настоящему народным, что особенно актуально для небогатых сельских школ. Электронная почта, недорогая сама по себе, не предъявляет в отличие от WWW-сервиса практически никаких требований к мощности компьютера, на котором она установлена. И, что самое удивительное, электронной почтой в сети можно сделать практически все, даже читать WWW-страницы и перемещаться по их ссылкам, если в этом есть необходимость.

"Мне понравилось отправлять письма по электронной почте. Мы переписывались пословицами. Письма приходят очень быстро. У Толи дома компьютер, и он мог отправлять письма вечером. Мы отправили друзьям много своих пословиц".

Витя Поминов, 3-й класс, г. Благовещенск

"Многие учащиеся и учителя в сетевых проектах используют электронную почту. Но многие ли хорошо освоили работу с почтовыми программами? Как настроить кодировку почтовых программ, чтобы ваши письма смогли прочитать? Как сделать запрос на файловый сервер? На эти и другие вопросы дает ответ курс "Введение в Интернет". Он полезен не только учащимся, но и учителям".

Анатолий Владимирович Бычков,
учитель информатики школы-лицея № 73, г. Барнаул

9. Гипертекстовые учебники университета.

В современных гипертекстовых разработках можно выделить два направления:

— сайты,
— локальные приложения.

Сайты "живут" в Интернете, локальные приложения — на компьютере индивидуального или в локальной сети коллективного пользователя (школьного класса, например).

"Когда я впервые пришел на занятия, то вообще не имел представления о том, что такое сайтостроение. Сейчас я могу создать сайт. Может, он не всем будет нравиться. Но я постараюсь, чтобы он быстро загружался, не утомлял глаз мельканием и цветовой палитрой, не оглушал музыкой, просматривался во всех популярных браузерах и был интересен по содержанию".

Виталий Пшеничников, 10-й класс,
г. Благовещенск

Жанровые особенности в основном определяются разницей в скорости передачи информации. Интернет работает очень медленно (средняя скорость передачи информации — несколько килобайт в секунду). На стадии становления глобальной сети скорость передачи росла каждый год, внушая оптимизм разработчикам и пользователям. Но сейчас, к сожалению, этот процесс прекратился, и новый виток разгона сети возможен только после кардинального пересмотра существующих способов передачи. Мир замер в ожидании Эйнштейна связи. Замер и принял на вооружение пессимистические правила, продиктованные суровой действительностью: сейчас хороший сайт — это маленький, простой сайт.

Он должен:

— иметь посредственную малоразмерную графику,
— не использовать звук и видео,
— иметь минимум интерактивности и динамики.

Помимо проклятия скоростью, сайтостроение переживает тяжелое время разгула демократии в среде разработчиков средств просмотра: каждый браузер требует особого программирования (даже на уровне разных версий, не говоря уже об уровне разных компаний) — никто не желает соблюдать стандарты.

Со временем, конечно, технические и политические трудности Интернета будут преодолены. Очень вероятно также, что Интернет станет существенно дешевле.

Сейчас же разработчики могут, не сдерживая себя, направить усилия в гипертекстовый жанр, отличный от сайтостроения, в котором можно использовать любые средства интерактивности и мультимедиа, грузить толстые звуковые треки, полноразмерную графику, видео — без риска усыпить пользователя, ждущего окончания загрузки у монитора и сохраняя при этом содержимое его кошелька. Речь идет о жанре гипертекстового приложения, примерами которого служат гипертекстовые учебники Роботландского университета. Такие гипертекстовые изделия живут на локальном компьютере или в локальной сети и, следовательно, не имеют "родовых" пятен ужасно медленного Интернета. Они дешевы, доступны без подключения к Интернету, кроме того, могут позволить

себе работать только на одном браузере, игнорируя все остальные.

Роботландские гипертекстовые учебники — это мультимедийные интерактивные лаборатории, в которых познавательное чтение сочетается с работой на многочисленных тренажерах, исполнителях, испытателях и конструкторах; сопровождается экзаменовкой и тестированием в зачетных классах, и все это — в рамках одного гипертекстового продукта, работающего в браузере.

Таким образом, новые роботландские учебники предлагают школьнику и педагогу удобные средства для реализации поставленной педагогической задачи, делают обучение более эффективным, увлекательным и контролируемым.

"Эти занятия нас очень многому научили. Конечно, выучить язык HTML можно было и по обычным учебникам, но не думаю, что в них говорилось хотя бы о том, что записывать команды желательно лесенкой (а ведь это так удобно и наглядно). Это только маленький пример, ведь мы получили столько полезной информации, которую вряд ли нашли бы в литературе, да и времени это заняло бы много. А "Назаметкам Сидорова", на наш взгляд, цены нет".

Оксана Соломаха, 10-й класс, г. Благовещенск

"На занятия по HTML-конструированию я пришел, потому что уже умел создавать сайты в визуальном редакторе, и рассчитывал, что меня научат чему-то еще. И меня научили создавать сайты вручную. В редакторе, конечно, быстро. Но руками — "чище" и удовлетворения больше".

Роман Батюченко, 10-й класс, г. Благовещенск

"Чем мне нравятся роботландские курсы 42 и 43 — они построены на интерактивных учебниках. Работать по ним очень удобно".

Людмила Николаевна Олейник, учитель информатики школы № 92, г. Пермь

"Особую благодарность хочется выразить Дуванову Александру Александровичу за его электронные учебники! Приобретенная нами другая литература практически осталась невостребованной! И так как я за многие годы преподавания информатики привыкла осваивать программные продукты в основном самостоятельно, то для меня это был как подарок судьбы".

Ольга Федоровна Брыксина, учитель информатики педагогического колледжа, г. Чапаевск

"Хочется отметить чрезвычайно квалифицированное методическое руководство кураторов курсов, которые стараются поддерживать программы на уровне современных требований информатического образования. При всем своем желании традиционные подходы к обучению такое гибкое реагирование на новации обеспечить не могут. Будущее — за дистанционными формами обучения, и информатика как нельзя лучше подходит для прокладывания этого пути для других дисциплин".

Виктор Фомич Клятченко, учитель информатики средней школы № 2, г. Люберцы

НАБОР 2000/2001 ГОДА

В новом учебном году будут работать следующие курсы:

10. Введение в информатику
11. Зимние вечера
12. Азы информатики
21. Конструирование
31. Азы программирования
32. Буки программирования
41. Введение в Интернет
42. Web-конструирование
43. Web-программирование
50. Английский с компьютером

В университет принимаются "индивидуальные" и "коллективные" ученики.

"Коллективный студент" — это группа детей, работающая под руководством одного или нескольких наставников (наставник, как правило, школьный учитель).

"Индивидуальный студент" — это учитель, желающий пройти обучение индивидуально, без группы детей.

В конце годичного курса при успешном обучении школьный учитель получает удостоверение от негосударственного образовательного учреждения "Роботландия+" и удостоверение государственного образца от Российской академии повышения квалификации и переподготовки работников образования.

Занятия в университете платные (цены на сайте www.botik.ru/~robot).

Они начинаются 15 октября 2001 года и продолжаются в течение двух семестров до мая 2002 года. Для подписчиков "Информатики" предусмотрена 10% -ная скидка за обучение.

Заявку желательно прислать как можно раньше, чтобы увеличить запас времени на предварительное знакомство с учебными материалами и подготовку к занятиям.

ОБРАЗЕЦ ЗАЯВКИ

Заявка на обучение в Роботландском университете 2001/2002 года

Предполагаемый курс (отделение): 43. Web-программирование
Форма обучения: "Коллективный ученик"
Руководитель (обучаемый): Иванов Петр Николаевич
Профессия: Учитель информатики средней шк.
Стаж работы в школе: 3 года
Директор школы: Петров Семен Сергеевич
Электронный адрес: ivanov@sch62.nsk.ru
Название учреждения, почтовый адрес: 630090, Новосибирск, ул. Жемчужная, 6, средняя школа № 62
Почтовый адрес для посылок: тот же

Предполагаемое число детей в группе и их возраст (для "коллективного ученика"): 20 школьников, 10—11-е классы
Откуда получена информация об университете: Газета "Информатика"

Заявки принимаются по адресу: kurs@robotland.botik.ru

Для получения более подробной информации можно написать по адресу kurs@robotland.botik.ru (администрация университета) или заглянуть на сайт www.botik.ru/~robot

ОПИСАНИЕ КУРСОВ

10. Введение в информатику

Курс только для "индивидуальных студентов" — учителей, преподающих или собирающихся преподавать информатику в младших классах на базе программно-методических систем "Роботландия" и "Хиты Роботландии".

11. Зимние вечера

Курс для младших школьников (3—4-е классы). Знакомство с алгоритмами и исполнителями на базе программно-методической системы "Роботландия"; электронная почта; детские конкурсы.

12. Азы информатики

Курс для младших школьников (3—4-е классы). Информация, исполнители, алгоритмы, "черные ящики". Курс базируется на фрагменте нового интерактивного гипертекстового учебника-лаборатории "Роботландия.RU".

21. Конструирование

Курс для детей 4—7-х классов, изучивших основы информатики. Создание компьютерных проектов, имеющих практическое применение на школьных уроках, на базе роботландских программ-конструкторов.

31. Азы программирования

Для детей 6—8-х классов, изучивших основы информатики (исполнитель, алгоритм, программа, первичные инструментальные навыки работы с компьютером). Этот курс для детей, которые нашли себя в программировании и желают повысить уровень своего мастерства, составляя головоломные алгоритмы для исполнителей Кукарача и Корректор.

32. Буки программирования

Курс наиболее эффективен для старшеклассников. Классические задачи и методы их решения. Составление и анализ алгоритмов не "по наитию", а "по науке". Олимпиадные задачи по программированию.

41. Введение в Интернет

Для детей 5—8-х классов. Введение в сетевые коммуникации на базе электронной почты. Коммуникативные проекты и конкурсы, работа с сетевыми исполнителями.

42. Web-конструирование

Для старшеклассников. Создание HTML-документов, в том числе с использованием специально созданной для курса библиотеки на JavaScript и CSS. Основы web-дизайна и проектирования. Создание интернетовских страниц и приложений, пригодных для использования на школьных уроках.

Особое внимание будет уделено проблеме разработки школьного сайта.

43. Web-программирование

Для старшеклассников, имеющих опыт программирования и интерес к этой деятельности. Изучение JavaScript, CSS. Построение динамических гипертекстовых приложений.

50. Английский с компьютером

Методические вопросы преподавания языка с использованием компьютера рассматриваются автором апробированного многолетней учительской практикой курса английского языка — Magic Land. Проектирование и реализация компьютерных упражнений к урокам.

* * *

Посетите сайт www.botik.ru/~robot!

Здесь вы можете послать электронную заявку, найти ответы на интересующие вас вопросы, узнать мнения коллег-учителей, полистать (а при желании и заказать) гипертекстовые учебники университета, заглянуть на "Кухню Сидорова" (учебная страница о том, как правильно делать сайты).

Как увеличить скорость "Жизни",

или Эффективная организация данных для повышения скорости поиска клеток и разрешения отношений соседства при реализации клеточного автомата Джона Хортона Конвея "Жизнь"

Лев Наумов,
Санкт-Петербург, Lnaumov@chat.ru

Продолжение. Начало в № 33/2001

2. Стандартные принципы хеширования и их применение к "Жизни"

Хеширование (от англ. *hash* — размешивать) служит для оптимизации задач обработки информации, а конкретнее, преимущественно для поиска. Этот метод при правильной организации данных позволит нам эффективно искать соседей некоторого объекта по всему набору имеющейся у нас информации о клетках.

Принцип хеширования заключается в разбиении всего набора данных на маленькие группы, с которыми, очевидно, быстрее работать. При этом информационным полям конкретной структуры сопоставляется значение так называемой хеш-функции (оно еще называется ключом). Разбиение на группы, "размешивание" информации, происходит по признаку равенства этого значения. Нужно стремиться к тому, чтобы группы были приблизительно одинаковыми по величине. Когда мы знаем содержимое полей структуры, которую хотим найти, и можем произвести подсчет ключа для нее, то искать точное совпадение требуемых полей нужно будет уже не по всему набору данных, а лишь в рамках соответствующей группы. Подробности об этом методе можно найти в уже упоминавшейся в этой статье книге [4].

В худшем случае, хеширование приведет к обычному поиску полным перебором. Это произойдет, когда значения ключа для всех объектов-структур получатся одинаковыми. Но для этого нужна поразительно неудачная хеш-функция. Справедливости ради надо отметить, что это верно лишь с точки зрения трудоемкости, на практике получится еще медленнее из-за "холостого прокручивания" механизма хеширования (разнообразных проверок условий и тому подобного).

Достаточно эффективный способ применения предложенного метода — организация массива указателей на списки. Каждый из этих списков содержит структуры, значения хеш-функции для которых одинаковы (это — вышеописанные группы), а индекс ячейки массива, содержащей указатель на данный список, равен ее значению (этот способ будет широко использоваться в следующих параграфах).

Также вполне благоприятен вариант, когда индекс равен некоей функции от ключа. Ведь, например, в языках C/C++ массивы индексируются с нуля, и если наша хеш-функция принимает целые значения от 7, то функция преобразования индекса должна будет вычитать из получающегося ключа семерку. Правда, в таком случае это преобразование имеет смысл включить в процедуру подсчета хеш-функции.

Итак, возвращаясь к нашему примеру, продолжим рассмотрение данных, хранящихся в объектах вышеописанного типа *LifeCell*. Организуем, как было написано выше, массив указателей на списки экземпляров этой структуры. Сделаем это так, чтобы значения хеш-функции для всех элементов одного списка было одинаково и к тому же равнялось индексу указателя на этот список в массиве. Теперь, когда нам понадобится посмотреть, является ли клеткой объект ($x_0 + 1$; $y_0 - 1$), мы сосчитаем хеш-функцию от данных значений полей (поле x равно $x_0 + 1$, а $y = y_0 - 1$), а потом поищем информацию о ней не среди всех клеток, а лишь среди элементов соответствующего списка.

Хеширование особенно эффективно для больших, протяженных колоний, когда выигрыш от его применения с лихвой окупает накладные расходы на его реализацию, а число клеток на порядок больше числа возможных значений ключа.

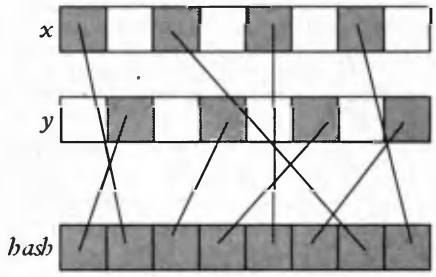


Рис. 3

Очень часто в хеш-функциях применяются битовые операции. Пусть x, y и ключ занимают по одному байту каждый. Тогда ключ можно по битам составить так, как показано на *рис.* 3. На этом рисунке x, y и ключ (*hash*) изображены как прямоугольники, поделенные на 8 квадратиков (бит). Младший бит — слева. Линиями показано, из каких битов x и y и как именно можно составить ключ.

Если целого байта на ключ жалко, то можно использовать четыре бита, как предложено на *рис.* 4.

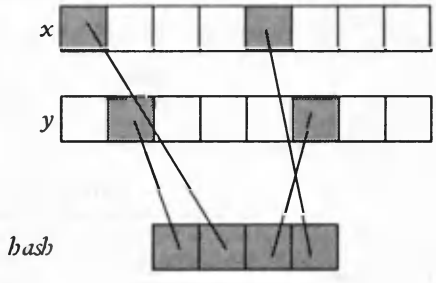


Рис. 4

Конечно, эффективность этого метода зависит от набора данных, на котором он используется. Скажем, точки (255; 0), (150; 45), (166; 37), (200; 139) размешаются хеш-функцией, изображенной на *рис.* 3, хорошо, все — в разные списки, а точки (85; 239), (117; 238), (87; 171), (213; 170) — все в один.

На самом деле нет никаких принципиальных ограничений на используемые хеш-функции, кроме здравого смысла. Помимо побитовых преобразований, это могут быть просто математические функции или композиции тех и других. Но не надо слишком мудрствовать. Надеемся, например, что синус суммы x и y вряд ли кто-нибудь будет использовать в качестве хеш-функции, по крайней мере применительно к "Жизни".

Ключ должен быть достаточно короток, чтобы расходы на него были позволительны, и в то же время достаточно длинен, чтобы хорошо "размешивать" данные. Чем больше мощность множества значений ключа, тем больше разных групп можно составить, тем меньше в них будет элементов (но хотелось бы, чтобы они все же были в каждой группе и чтобы их количество для всех групп было почти одинаковым или хотя бы одного порядка) и тем быстрее по ним будет производиться поиск.

Изображенные на *рис.* 3 и 4 модели получения ключа по полям *LifeCell* используют некие общие соображения теории хеш-функций, абстрагируясь от специфики задачи. Как было указано выше, есть наборы клеток, которые эти функции размешают крайне плохо. Давайте все-таки вспомним, что мы моделируем именно "Жизнь".

3. Более "жизненное" хеширование

Зададимся двумя целыми параметрами a и b, а также матрицей указателей на списки *mat* размера a на b. Далее, говоря "хеш-таблица", будем иметь в виду именно ее. В качестве ключей хеширования (здесь их будет даже два) предлагается использовать остатки от деления x на a в первом случае и y на b во втором. Вот примеры реализации таких хеш-функций для x и для y соответственно:

```
int HashX(int x)
{
return x%a;
}
```

```
int HashY(int y)
{
return y%b;
}
```

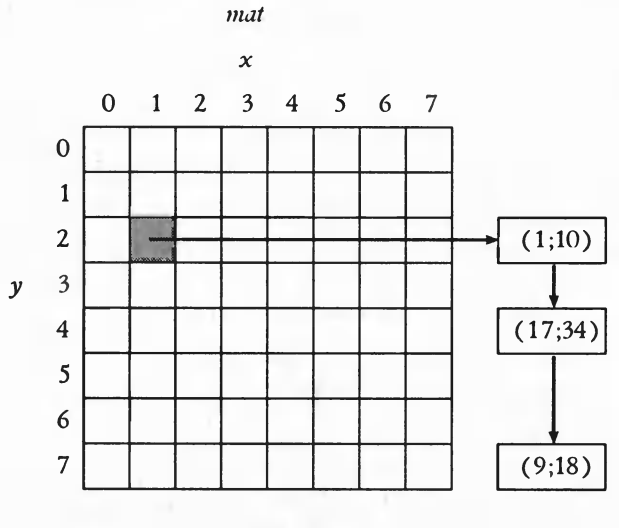


Рис. 5

Очевидно, что результатом первой функции будет целое число из отрезка от 0 до $a - 1$, а второй — от 0 до $b - 1$. Пусть после одного такого вычисления мы получили f — как значения ключа для x, а g — для y некоторой точки (x_0, y_0). Тогда информацию о том, является ли место (x_0, y_0) клеткой, нужно будет искать в списке, на который указывает $mat[f, g]$. На *рис.* 5 приведен пример такого размещения, в котором положено $a = 8$ и $b = 8$, хотя, вообще говоря, a и b вполне могут быть (а скорее даже должны быть) не равны, что мы будем наблюдать в дальнейшем.

Так в чем же обещанное преимущество хеш-метода, описанного в этом параграфе, над общим, описанном в предыдущем? Дело в том, что этот метод — адаптивный, то есть он может повышать свою эффективность в зависимости от конкретных данных. "Переразмешивать" структуры вследствие анализа своей собственной работы. Как это осуществляется? На этот вопрос мы ответим в следующем параграфе.

4. Адаптивное хеширование

Как вы уже, наверное, догадались, механизм "переразмешивания" основан на изменении a и b. Когда списки, "прицепленные" к разным ячейкам матрицы, слишком сильно отличаются по размеру, следует изменить параметры хеширования (a и b) и, как следствие, размеры нашей таблицы. Вернемся снова к случаю, когда оба параметра равны восьми, рассмотренному нами в предыдущем параграфе. Клетки с координатами 1, 9, 17, ... и вообще вида $1 + 8k$ (где k — некое целое число) в таком случае попадут в один список (еще раз взгляните на *рис.* 5). Как нужно поменять a и b, чтобы исправить ситуацию? Посмотрите на таблицу на с. 22.

Здесь приведены остатки от деления чисел указанного вида на целые числа от одного до двадцати четырех. Серые строки не представляют для нас никакого интереса, так как изменение параметра хеширования на соответствующие им значения делителя не приведут к "переразмешиванию", а сохранят текущую группировку информации. То есть все, что было в одном списке, в одном же и останется. Только количество списков уменьшится вследствие их объединения прямо целыми группами (например, это произойдет, если поменять значение a с 8 на 4). Остальные строки разделены на белые и черные зоны. Эти зоны представляют собой периоды значений хеш-функции (два соседних периода выделены разными цветами для наглядности), если параметр размешивания принять равным соответствующему делителю. В рамках рассматриваемой нами задачи ясно, что все значения хеш-функции внутри одного периода не повторяются, тогда длина периода и есть количество списков, на которые разобьется группа, имеющая непомерно большое в нашем понимании число элементов. Если мы рассматриваем в данный момент хеширование по x-координате, а d — делитель, претендующий на то, чтобы стать параметром размешивания (для определенности — a), то эта длина (обозначим ее, как *len*), очевидно, равна $max(a, d)/nod(a, d)$. Здесь функция *max*

Делитель	Делимое																		
	1	9	17	25	33	41	49	57	65	73	81	89	97	105	113	121	129	137	145
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	0	2	1	0	2	1	0	2	1	0	2	1	0	2	1	0	2	1
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1	4	2	0	3	1	4	2	0	3	1	4	2	0	3	1	4	2	0
6	1	3	5	1	3	5	1	3	5	1	3	5	1	3	5	1	3	5	1
7	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5
8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	0	8	7	6	5	4	3	2	1	0	8	7	6	5	4	3	2	1
10	1	9	7	5	3	1	9	7	5	3	1	9	7	5	3	1	9	7	5
11	1	9	6	3	0	8	5	2	10	7	4	1	9	6	3	0	8	5	2
12	1	9	5	1	9	5	1	9	5	1	9	5	1	9	5	1	9	5	1
13	1	9	4	12	7	2	10	5	0	8	3	11	6	1	9	4	12	7	2
14	1	9	3	11	5	13	7	1	9	3	11	5	13	7	1	9	3	11	5
15	1	9	2	10	3	11	4	12	5	13	6	14	7	0	8	1	9	2	10
16	1	9	1	9	1	9	1	9	1	9	1	9	1	9	1	9	1	9	1
17	1	9	0	8	16	7	15	6	14	5	13	4	12	3	11	2	10	1	9
18	1	9	17	7	15	5	13	3	11	1	9	17	7	15	5	13	3	11	1
19	1	9	17	6	14	3	11	0	8	16	5	13	2	10	18	7	15	4	12
20	1	9	17	5	13	1	9	17	5	13	1	9	17	5	13	1	9	17	5
21	1	9	17	4	12	20	7	15	2	10	18	5	13	0	8	16	3	11	19
22	1	9	17	3	11	19	5	13	21	7	15	1	9	17	3	11	19	5	13
23	1	9	17	2	10	18	3	11	19	4	12	20	5	13	21	6	14	22	7
24	1	9	17	1	9	17	1	9	17	1	9	17	1	9	17	1	9	17	1

возвращает максимум двух чисел, а *nod* — их наибольший общий делитель. Теперь мы можем просто сформулировать, что для серых строк $len = 1$.

Какое же d принять за новое значение a? Конечно, лучше — то, которому соответствует наибольшая *len*. Тогда одним из критериев оптимального решения является то, чтобы $nod(a, d) = 1$ или, проще говоря, чтобы a и d были взаимно простыми. Ясно, что можно найти сколь угодно большое d, удовлетворяющее этому условию. Вообще предел *len*, при d стремящемся к бесконечности, равен бесконечности. Но с практической точки зрения нам бы не очень хотелось непомерно расширять *mat*. Поэтому "лучше, чтобы $nod(a, d)$ равнялся единице", так как тогда меньшим d будет соответствовать большая *len*.

Так все-таки в каких пределах лучше искать d? Наверное, самым правильным будет завести некий базовый уровень t, в окрестности которого найти d_1 и d_2, такие, что $d_1 < a$, а $d_2 > a$ и оба "достаточно хорошо" переразмешают клетки. Потом выбрать из них то, которое ближе к t. Что значит "достаточно хорошо перемешивает"? Как уже не раз говорилось, это значит, что все группы должны быть приблизительно равных размеров.

Нужно отметить, что t может меняться от итерации к итерации. Можно использовать в качестве t кубический корень из числа клеток на данном шаге (округленный, естественно). Тем более что число клеток (размер популяции) является важным параметром работы клеточного автомата "Жизнь" и почти всегда вычисляется и отображается на экране. Однако при использовании этого метода бывает эффективным пересчитывать t не каждый раз, а через один или два раза, в зависимости от периодичности колонии, развивающейся в конкретном мире.

Что такое эти пресловутые группы "приблизительно равных размеров"? Как понять, что построенная хеш-таблица требует "перетряхивания"? Предлагается следующий метод: при каждом ее изменении, обусловленном событиями, происходящими с колонией, подсчитывать среднее арифметическое числа элементов в списке. Здесь можно воспользоваться тривиальным свойством среднего арифметического: если нам известно его значение A_n для набора из n чисел, то если добавить к набору чисел еще одно число q, то для $n + 1$ числа $A_{n+1} = (n \cdot A_n + q)/(n + 1)$. Например, можно написать это как-то так:

```
void AddValue(int q, float& A, int& n)
// Функция, добавляющая число q к набору
// из n чисел, по которому нужно считать
// среднее арифметическое A
{
   A = (A * n + q) / (++n);
}
```

```
void Iteration()
{
 float A=0;
 // Текущее значение среднего арифметического
 int n=0;
 // Текущее количество значений.
 //  Для данного n A представляет собой An
 int q=0; // Добавляемое значение
<Код программы, в котором, в частности,
 как-то задается q>
AddValue(q, A, n);
// Непосредственно добавление значения.
// Как правило, оно происходит в некотором
// цикле
<Код программы>
}
```

Пусть результат подсчета среднего помещен в *A*. Так, если изначально задать некое целое положительное число *r*, то перестраивать хеш-таблицу следует, если число клеток в некотором списке в *r* раз больше, чем *A*. Для быстрейшей проверки этого соотношения имеет смысл представить *mat* не как матрицу-указатель, а, например, как матрицу записей с двумя полями, одно из которых — указатель, а второе — длина списка, на который он указывает. Подсчитав сумму этих длин, можно заодно выяснить и число клеток на данном шаге, что может быть полезно (смотрите несколько абзацев выше). Также это позволит намного более простым способом подсчитать *A*. Зачем же тогда нужен описанный выше способ определения среднего? Дело в том, что есть еще один вариант, экономящий память, но теряющий в эффективности. Можно сравнивать *r*, умноженное на *A*, со значениями длин списков, полученными на следующей итерации (их все равно нужно считать для следующего значения среднего арифметического). Таким образом, *A* "отстает" на один шаг. Почему это вообще имеет смысл? Ответ лежит только в узкоспецифических особенностях "Жизни" по правилам Конвея (2—3 соседа, чтобы клетка не умерла, 3 — чтобы родилась). Это может быть верно и для правил, близких к ним, но никак не чего-то вроде 1, 6 и 7, чтобы клетка умерла, и 2, 3 и 5, чтобы родилась, "Жизнь" по которым очень сильно отличается от конвеевской. Так почему же все-таки это так? Во-первых, это обусловлено периодичностью стационарных состояний, а во-вторых, тем, что "сфера влияния" клетки — ее ближайшие соседи и за один шаг она не может "переместиться" дальше, чем на одно место. Таким образом, стоящая в списке, на который указывал $mat[u, v]$, клетка не попадет никуда, кроме как в один из восьми списков:

$mat[u, v+1]$, $mat[u, v-1]$,
$mat[u+1, v+1]$, $mat[u+1, v]$,
$mat[u+1, v-1]$, $mat[u-1, v+1]$,
$mat[u-1, v]$ или $mat[u-1, v-1]$.

Думаете, игра не стоит свеч? Напрасно! Попробуйте. Конечно, намного эффективнее (или по крайней мере логичнее) рассматривать *A* с текущими значениями длин списков, но как промежуточный вариант это очень неплохо. А если вы программируете "Жизнь" под DOS, то это — просто способ сэкономить память еще под несколько сотен клеток. Кроме того, это можно рассматривать просто как расширение описываемого метода хеширования, вносящее дополнительную неопределенность в результат.

Константу *r* выбирайте также осторожно, чтобы слишком частые "перестройки" не затормаживали работу машины из-за большой трудоемкости ("перетряхнуть" *mat* — операция относительно непростая), а слишком редкие — из-за неэффективного поиска.

Не стоит применять методы определения необходимости перестройки *mat*, основанные на разности минимальной и максимальной длин списков. Почти всегда минимум будет равен нулю. Вообще лучше избегать абсолютных отклонений при хешировании, намного практичнее использовать относительные.

Наверное, у вас возникает абсолютно справедливый вопрос: а какие a и b использовать при первом шаге программы? Да хоть уже не раз упоминавшиеся значения 8! А лучше — $a = 8$, а $b = 7$, чтобы первая же итерация с диагонально растянутыми колониями не потребовала переразмешивания. Однако метод адаптивный, и *mat* все равно будет перестраиваться под конкретную ситуацию на поле. Лучше выбирайте их не очень большими, помните, раз в итерацию вам придется пробегать матрицу размером a на b, делая, прямо скажем, нетривиальные операции. Поэтому и было упомянуто, что можно использовать t как плавающую границу. Пример такого метода: задаться некоторым параметром w и считать t не как было предложено выше, как кубический корень из числа клеток, а как корень степени w, чтобы, варьируя w от 3 до "бесконечности", удерживать t в разумных для используемого компьютера пределах. То есть так, чтобы пересчет занимал время, не обременяющее пользователя. Однако нужно иметь в виду, что чем больше a и b, тем быстрее для хорошо размешанных данных осуществляется поиск клетки.

Здесь приведен далеко не полный список возможных методов эффективной организации данных при программировании такой интереснейшей игры, как "Жизнь". Кроме того, они немного адаптированы под статью. Ведь, например, на практике вместо массива структур с двумя полями, очевидно, лучше применять просто два массива.

В конце обычно нужно делать выводы. Например, после статьи такого рода хорошо бы сравнить показатели работы предложенных алгоритмов. Однако в данном случае это, пожалуй, невозможно. Эффективность настолько зависит от реализации, что обсуждать это довольно тяжело. Для любого из описываемых методов, кроме сортировки по x и по y, сразу можно придумать популяцию, на которой он будет работать сколь угодно плохо. Возникает вопрос лишь в степени реальности возникновения таких популяций по ходу игры. Почему не остановиться на двойной сортировке? Она неприлично медленная, $O((N \cdot \ln N)^2)$. Однократная сортировка дает тоже очень хорошие результаты (неожиданно для такого простого способа), но не для очень больших популяций.

При достаточно профессиональной реализации про адаптивное хеширование можно сказать следующее:

1. Оно работает по эффективности сопоставимо, но все же, как правило, чуть хуже сортировки, хотя у него трудоемкость N в худшем случае (чего не бывает, если следовать материалу данной статьи), а у *QuickSort* — $O(N \cdot \ln N)$. То есть, казалось бы, должно быть наоборот, но здесь нельзя учесть накладные расходы на реализацию хеширования. Тут имеют значение и способности программиста. Поэтому выше и написано: "при достаточно профессиональной реализации". Может, ваша реализация сразу превзойдет сортировку.

Чтобы адаптивное хеширование заработало быстрее, нужна очень тонкая настройка w и r, а может, и t не как функции w. Это очень трудная и интересная задача. Но, опять-таки, преимущество сортировки очень зыбко. Часто в процессе игры появляются популяции, на которых хеширование сработает лучше.

Еще обращаю ваше внимание, что это для небольших популяций. Для больших же, при подходящих w и r, перевес уже в пользу хеширования.

2. В отличие от сортировки, хеширование не критично к размеру популяции и открывает реальные перспективы почти неограниченного роста (но это — отдельный разговор).

3. Хеширование позволяет проводить интересный статистический анализ процесса, так как задействует нетривиальные параметры автомата.

4. Вообще хеширование ближе к природе "Жизни", как таковой. Оно позволяет именно "играть" в нее, а не просто "быстро разрешать отношения соседства". Оно дает некую степень свободы, позволяющую концептуально менять правила жизни. Вообще этот метод имеет большое значение для прикладного использования "Жизни" (той же архивации данных, например).

Похоже, осталось больше вопросов, чем ответов. Но кому на них отвечать, как не вам? Информация, представленная в этой статье, пожалуй, может послужить хорошей отправной точкой или по крайней мере поводом для размышлений.

Литература

1. *Тоффоли Т., Марголус Н.* Машины клеточных автоматов. М.: Мир, 1991, с. 5—15.

2. *Гарднер М.* Математические досуги. М.: Мир, 1972, с. 458—488.

3. *Ахромеева Т.С., Дородницын В.А., Еленин Г.Г., Курдюмов С.П., Малинецкий Г.Г., Потапов А.Б., Самарский А.А.* Компьютеры и нелинейные явления: Информатика и современное естествознание / Под ред. Самарского А.А. М.: Наука, 1988, с. 5—43.

4. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построения и анализ. М.: МЦНМО, 2000, с. 135—180, 213—236.

5. *Романовский И.В.* Дискретный анализ, часть I. СПб.: Центр издательских систем Санкт-Петербургского государственного института точной механики и оптики, 1998, с. 50—54, 94—95.

Книжный шкаф

Как работают компьютеры?

Код.
М.: Издательско-торговый дом "Русская редакция", 2001.
Ориентировочная стоимость книги в книжных магазинах Москвы — 150 руб.
Книгу можно также приобрести в электронном магазине www.ITbooks.ru.

В издательстве "Русская редакция", известном выпуском компьютерной литературы в сотрудничестве с Microsoft Press, вышла замечательная книга Чарльза Петцольда "Код". Чарльз Петцольд (Charles Petzold) — один из известных специалистов в области информационных технологий, один из разработчиков идеологии операционной системы Windows. Но к этой операционной системе, равно как и к другим операционным системам и конкретным прикладным вопросам, книга не имеет прямого отношения. Если бы автор захотел подать свою книгу в Экспертный совет Министерства образования (мы, разумеется, не делаем никаких предположений о перспективах ее рассмотрения в Совете), то он бы мог назвать ее "книгой для чтения по основам информатики и вычислительной техники". Адресована эта книга, на наш взгляд, прежде всего школьникам (причем некоторые главы с увлечением читают — проверено — и учащиеся младшей школы). (Отметим, что в редакционной аннотации сказано, что издание "адресовано в первую очередь студентам вузов (как гуманитарных, так и технических)", но такое узкое позиционирование этой книги нам не кажется удачным.)

Книга состоит из 25 глав и содержит множество иллюстраций. В предисловии Чарльз Петцольд говорит, что написал книгу, устав отвечать на вопрос "Как работают компьютеры?". Таким образом, каждая глава посвящена тому или иному аспекту этого центрального вопроса. Автор не обошел своим вниманием практически ни одной "мелочи". Он честно прошел весь путь — от физических основ электроники до вопросов современного программирования. Математическая логика и конструирование логических схем, системы счисления и измерение количества информации, устройство микропроцессора, памяти и внутренняя организация персонального компьютера, функции операционной системы — это лишь беглое и крайне неполное перечисление вопросов, которые затрагиваются в книге.

Собрать правильный шестиугольник

Д.М. Златопольский,
Москва

Перед вами (см. *рис.* 1) — 6 фишек, каждая в виде равностороннего треугольника. На каждой фишке записаны 3 числа (некоторые в десятичной системе счисления, некоторые — в двоичной). Условимся числа, характеризующие одно и то же количество, но записанные в разных системах счисления, называть близнецами. Например, близнецами являются двоичное число 101 и десятичное 3.

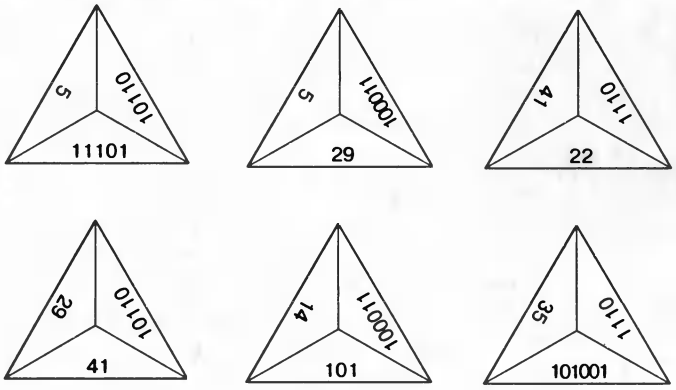


Рис. 1

Необходимо собрать из фишек правильный шестиугольник (*рис.* 2) таким образом, чтобы в смежных частях фишек оказались числа-близнецы.

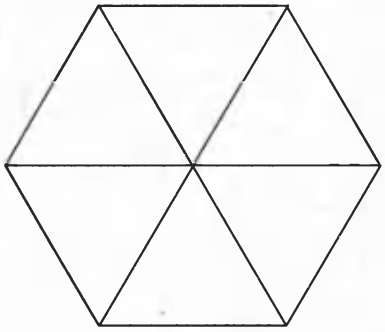


Рис. 2

Правильный ответ — см. *рис.* 3.

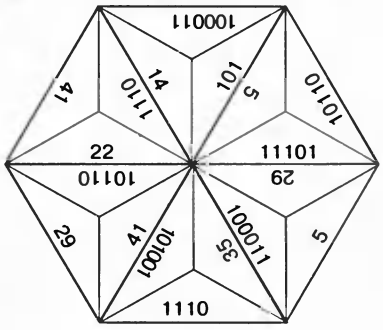


Рис. 3

Варианты заданий — вместо чисел, приведенных на фишках, которые представлены в левой колонке таблицы, использовать числа в других колонках.

Числа на фишках	Вариант 2	Вариант 3
5	6	4
14	13	15
22	23	21
29	28	27
35	36	34
41	43	39
101	110	100
1110	1101	1111
10110	10111	10101
11101	11100	11011
100011	100100	100010
101001	101011	100111

Калейдоскоп

Компьютеры играют на бирже лучше людей

Компьютерный алгоритм может заработать больше денег, чем человек. Такой вывод сделали специалисты на основе эксперимента, проведенного в исследовательском центре компании IBM.

В процессе исследования соревновались две команды из шести участников: в первой — люди, во второй — исключительно компьютеры. В конце сессии выяснилось, что электронные трейдеры заработали на семь процентов больше прибыли, чем живые.

По материалам газеты "Финансовая Россия"

Грибы, но несъедобные

Записи на компьютерных CD-дисках считаются едва ли не самым надежным способом хранения информации. Но оказывается, что диски подвержены воздействию грибка. Испанский ученый Виктор Карденас, находясь в командировке в Белизе, вдруг обнаружил, что поверхность его компакт-дисков с научными материалами разрушилась и диски стали непригодными. Проведя химико-биологические анализы, ученый пришел к выводу, что виной всему грибок, который питается углеродом и азотом, входящими в состав пластиковых покрытий компакт-дисков, и таким образом разрушает дорожки с записями.

По материалам журнала "Paradox"

Материалы рубрики подготовлены К.Кузнецовым

Первый инженер России

Владимира Григорьевича Шухова (1853—1939) называют великим русским инженером, гениальным русским инженером, а в начале XX века называли первым инженером России [1]. Шухов никогда не работал в научных учреждениях, но Академия наук СССР избрала его как крупнейшего ученого-новатора своим почетным членом. Никогда не раздавался его голос с кафедры учебного заведения, но целые поколения инженеров считают себя его учениками и последователями [2]. Представляя свои работы на международные выставки, Шухов открывал для мира достижения русской инженерной мысли и рано получил международное признание — уже в 1897 году он стал членом "Союза французских гражданских инженеров".

В 1876 году Владимир Григорьевич окончил Московское высшее техническое училище и ему предложили остаться там работать. К теоретической работе склоняли Шухова также его учитель — создатель русской авиации Н.Е. Жуковский и выдающийся русский математик П.Л. Чебышев. Но молодой выпускник имел другие планы. Его не устраивало, что сделанные им открытия или полученные математические формулы будут просто кем-то когда-нибудь использованы. Шухов хотел сам видеть плоды своего труда в виде машин и конструкций. И он вежливо, но упорно отказывался: "Я человек жизни..."

Великолепно владея высшей математикой и начертательной геометрией, Шухов, безусловно, мог бы стать выдающимся теоретиком и делать открытия за письменным столом. И он действительно совершал открытия, но для того, чтобы тотчас же воплотить их в жизнь. Выбору этого нелегкого для себя, но полезного для общества пути способствовала годичная командировка в США.

В Америке Шухову особенно понравилось, с какой скоростью осуществляется внедрение технических идей и как заботливо опекает "состоятельная общественность" талантливых изобретателей. Однако многое в жизни этой страны неприятно удивляло двадцатитрехлетнего русского инженера. "Подлинные инженерные находки тонут здесь в раздутых рекламой находках мнимых. Пренебрежение инженерной мыслью, ковбойски-высокомерное отношение к математике и науке в целом, глубокое убеждение в продажности всего на свете, конкуренция политических партий, перепалки кандидатов — все это производило отталкивающее впечатление" [1].

Шухов вернулся в Россию с твердым намерением внедрять прогрессивные технические идеи сразу, смело и дерзко, как это делают американцы, но, продумав стратегию претворения, как это делают англичане, скрупулезно рассчитав каждый шаг и его последствия, как это делают немцы, приняв во внимание все возможные мелочи, как это делают французы. Указанные качества, соединенные с русской сметливостью и самоотверженным служением делу, породили тот феномен шуховского гения, который был назван "русским синтезом".

Шухова называли "человеком-фабрикой", потому что он один с несколькими помощниками смог создать столько, сколько по силам десятку научно-исследовательских коллективов. Удалось даже составить своеобразную "азбуку Шухова" — почти для каждой буквы алфавита нашлись сделанные им открытия и изобретения [1, 2, 3]:

А — ангары;

Б — батопорты (плавучие гидротехнические затворы); баржи нефтеналивные;

В — воздушно-канатные дороги; первые в мире висячие металлические перекрытия; водопроводы в Москве, Тамбове, Воронеже, Киеве, Харькове; водонапорные башни;

Г — газгольдеры (газохранилища);

Д — доменные печи; дымовые трубы из кирпича и металла;

З — землечерпалки;

К — котлы паровые; кузнечные цехи; кессоны (специальные камеры для работы, например, под водой); крекинг-процесс (процесс переработки нефти и ее фракций);

М — мартеновские печи; мачты электропередач; маяки; медно-литейные цехи; мостовые краны;

Н — насосы, позволившие добывать нефть с глубины 2—3 км; нефтеперегонные установки; нефтепроводы, в том числе первый в мире: Балаханы — Черный Город;

П — пакгаузы (закрытые складские помещения); порты;

Р — радиобашни, в том числе Шаболовская; резервуары, в том числе первые в мире цилиндрические;

Т — танкеры; трубопроводы;

Ш — шпалопрокатные заводы;

Э — элеваторы, в том числе "миллионники" в Козлове и Саратове.

Под руководством Шухова спроектировано и построено около 500 мостов (в том числе через Оку, Волгу, Енисей); сооружена вращающаяся сцена МХАТА; с его именем связаны многие крупнейшие и сложнейшие индустриальные стройки — Магнитогорский металлургический комбинат, Челябинский тракторный, Белорецкий, Выксунский, Ижевский, Нижнетагильский заводы, "Азовсталь"; он руководил работами при подъеме наклонившегося минарета[1] и медресе[2] Улугбека в Самарканде. Шухов принимал участие в разработке и производстве нескольких типов мин, минных взрывателей, платформ для тяжелых орудий [3].

Если оценивать пользу, которую Шухов принес своей стране и всему миру, то становится очевидным, что в этом смысле он опережает одного из самых плодовитых изобретателей в истории человечества Т.Эдисона [1].

Свою знаменитую башню-антенну на Шаболовке в Москве (как и многое другое) Шухов проектировал, опираясь на опыт народа. На мысль о сетчатой, как бы плетеной из металла башне его натолкнули плетеные корзины. Башня была смонтирована без лесов, решетчатые секции "склепывались" и подтягивались стальными тросами, и казалось, в небо выдвигается гигантская подзорная труба. "Прекрасная и прозрачная, она заслонила собой от потомков другие создания Шухова..."

Литература

1. От махин до роботов: очерки о знаменитых изобретателях, отрывки из документов, научных статей, воспоминаний, тексты патентов / Сост. М.Н. Ишков. (В 2 кн. Кн. 2.) М.: Современник, 1990.

2. Детская энциклопедия. М.: Изд-во Академии педагогических наук РСФСР, 1960. Т. 5.

3. *Новиков В.В.* Шухов // Большая советская энциклопедия. 3-е изд. М.: Советская энциклопедия, 1978. Т. 29.

[1] Минарет — высокая башня при мечети, с которой служитель мечети (муэдзин) сзывает мусульман на молитву.

[2] Медресе — мусульманская школа.

Пятые Соловейчиковские чтения

28–29 сентября 2001 года, Московский городской Дом учителя

ПРОГРАММА ЧТЕНИЙ:

1. **Учение с увлечением: делать не только интересное, а все, что нужно, делать с интересом! Особая культура отношения к жизни. Как ее воспитать?**
   - По страницам книги Симона Соловейчика "Учение с увлечением"
   - Учение с увлечением (педагогические технологии)
   - Как научить учиться? Где баланс между "хочу" и "надо"? Возможно ли, чтобы учение было интересным для всех? Как увлечь, когда интереса нет? Мастер-классы В.Шаталова, Е.Ильина, С.Лысенковой, А.Лобка, М.Балабана, Ш.Амонашвили, И.Демаковой

2. **Главные направления современной образовательной политики в контексте педагогики:**
   - 12-летнее обучение
   - Изменение содержания образования (стандарты)
   - Единый государственный экзамен (тестирование)
   - Реструктуризация сельской школы

   Одним из главных критериев эффективности, грамотности и профессионализма любого управленческого решения в образовании является проработанность его педагогической составляющей. Насколько сегодняшние изменения в школе отвечают этому критерию?

3. **"Круглые столы" по теме "Педагогика и мировоззрение":**
   - Патриотическое воспитание
   - Религия и школа
   - Современная школа: другая идеология

4. **Педагогика сотрудничества:**
   - Отчет о трехмесячном курсе "Педагогика сотрудничества. Весна-2001"
   - Рассказ о II Международном фестивале "Школа для ребенка. Лето-2001"
   - Новые проекты Объединения педагогических изданий "Первое сентября"

ДЕВИЗ ЧТЕНИЙ:
"Учение с увлечением!"

В одном замечательном письме в редакцию были такие слова: "Если мы научимся учить детей так, чтобы они учились увлеченно, то через 10 лет наше общество придет к совершенно новому, более высокому качеству жизни". К этому стоит добавить, что сегодня для многих детей альтернативы ***учению с увлечением*** *нет. Либо учиться с увлечением, либо не учиться вообще…*

Чтения пройдут в Московском городском доме учителя. Открытие чтений 28 сентября в 11.00.

Впервые для участия в чтениях потребуются входные билеты, но не приобретенные, а самодельные, оформленные на ваше усмотрение.

Действительным входным билетом будут являться маленькие сочинения (одна или полстранички) на тему «Учение с увлечением». Что означают эти слова сегодня для вас и ваших коллег? Особенно здорово было бы услышать короткие истории про то, как иногда интерес к предмету возникает на уроке совершенно неожиданным образом, делая его запоминающимся навсегда.

Как и в прошлом году, гостиница "Дон" готова принять вас у себя. Номер вы можете зарезервировать уже с сегодняшнего дня. Место в двухместном номере стоит 320 руб., одноместный номер стоит 520 руб.

Телефон гостиницы "Дон": (095) 217-67-86

Thinking machines?

Computers in storybooks and on TV have exciting personalities and can do everything humans can do and more. But in real life things aren't quite like that[1].

Some computers appear to have human powers of reasoning and logic. But machines can't really think in the way that people can. When a computer seems clever, it just means that the programmer was intelligent.

Things computers do that make them look intelligent:

Build cars.

Beat all but the very best players at chess.

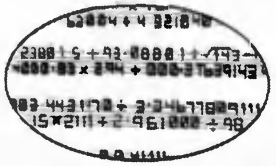
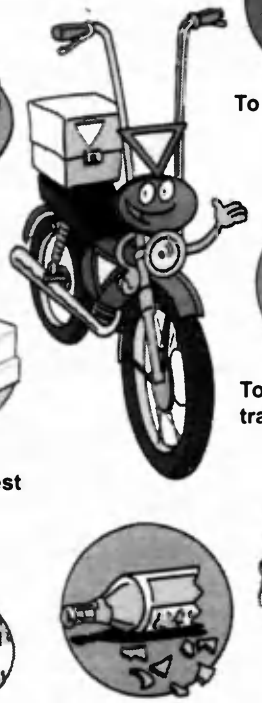


Figure out complicated problems involving hundreds of figures.

Dummies

Computers can only follow instructions. Most of the things that people do every day, even the ones that we think are really simple, are so complex[2] that it would be impossible to explain them as a set of instructions. This means that computers cannot do them. This robot has been programmed to carry packages across a busy town. But the program just isn't good enough.

Robobike is programmed to do all these things:



To read maps.

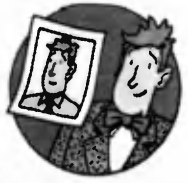
To find the fastest alternative route if a road is closed.

To stop and go at traffic lights.

To use cycle lanes.

To swerve around obstacles and broken glass.

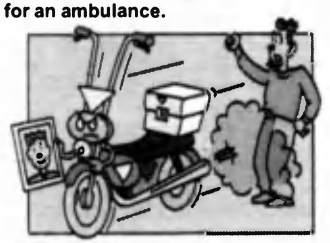
To phone an ambulance if there's an accident.



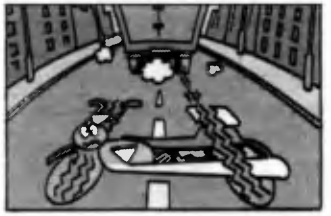
To recognize the receiver of the package by matching his face with a prescanned photograph.

But, despite all these instructions, all sorts of things can, and do, go wrong.

On his way, robobike passes the filming of a car crash for a TV drama. The robot, not recognizing cameras, calls for an ambulance.



On arrival, the receiver tries to collect his parcel. Robobike won't release it[3]. The receiver has grown a beard over the weekend[4] and so he no longer matches the robot's photographic image.



On the way back, a traffic light gets stuck[5] on red. The robot waits there for hours and eventually is run over by a truck.

Robobike has failed. A person could have coped with these new situations as they arose[6]. But when it comes to making decisions in unexpected circumstances, computers are no good.

Working by the rules

Computers are only good at tasks which never vary and are identical every single time they are done. But when they are good, they are very, very good. They can be much quicker and much more reliable than human beings.

Mike works in a factory. His job is to put a cherry on the top of sweet rolls.

Some days, if he's watched too much TV, his eyes get a bit blurry and the cherry doesn't end up exactly in the middle.

On other days, his sweet tooth gets the better of him and some cherries never quite make it to the rolls.

And a lot of the time he daydreams about his summer vacation and forgets the cherries altogether.

The cherry-placing process is a repetitive task which can be done by following a set of instructions. This makes it ideal for a robot, which will never be tempted to eat the cherries or doze off. So it is because they can't think that machines are better at these jobs than people, not because they can.

All the jobs computers do well are ones that can be broken down into a limited number of rules and commands. Even chess, which we think of as a game "for the brainy, is actually a game of rules, which a computer can be "taught".

The future

Will scientists ever be able to put enough rules into a computer so it can make decisions like a person? Some scientists think not. Others believe that, as more is understood about how the human brain works, scientists will be able to use this knowledge to make the first truly intelligent computers.

Мыслящие машины?

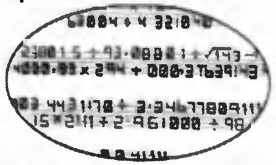
В рассказах и телевизионных передачах компьютеры представлены как личности, которые могут делать то же, что и человек, и даже больше него. Однако в действительности дела обстоят не совсем так.

Может показаться, что некоторые компьютеры имеют человеческие способности в области рассуждений и логики. Но на самом деле машины не способны мыслить так, как это делают люди. Если компьютер кажется умным, это просто значит, что таким был готовивший программу программист.

Компьютер может делать следующие вещи (благодаря чему он кажется умным):

Строить автомобили.

Выигрывать в шахматы у всех, кроме самых лучших игроков.

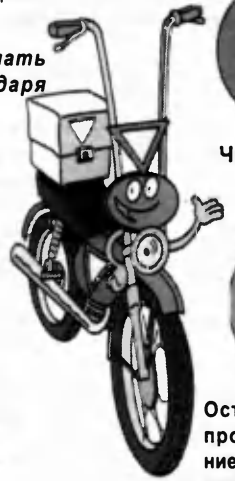


Выполнять вычисления для сложных задач с сотнями чисел.

Марионетки

Компьютер способен только выполнять команды. Большинство вещей, которые люди делают каждый день (причем даже те вещи, которые кажутся нам простыми), являются настолько сложными, что их было бы невозможно представить в виде набора команд для компьютера. А это значит, что компьютер не способен их выполнить. Изображенный здесь робот запрограммирован возить посылки через шумный город. Однако используемая им программа не очень хороша.

Велосипед-робот способен выполнять следующее:



Читать карты.

Находить самый быстрый альтернативный путь, если дорога закрыта.

Останавливаться и продолжать движение при соответствующих сигналах светофора.

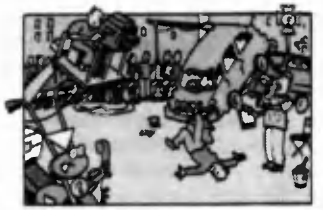
Использовать велосипедные дорожки.

Объезжать препятствия и разбитое стекло.

Вызывать машину "скорой помощи" при несчастных случаях.

Узнавать получателя (по его фотографии).

Однако, несмотря на это, все идет не так, как надо.



Велосипед-робот проезжает место, где ведется съемка автомобильной аварии для телевизионного фильма. Не распознав съемочные камеры, робот вызывает машину "скорой помощи".

Когда велосипед-робот прибывает на место, получатель не может забрать свою посылку. Робот не отдает ее. Получатель отрастил бороду за выходные дни и потому больше "не соответствует" имеющейся у робота фотографии.

Когда велосипед-робот едет назад, оказывается неисправным светофор — все время горит красный свет. Робот стоит и ждет много часов подряд, и в конце концов его переезжает грузовик.

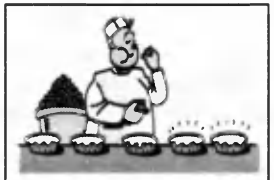
Велосипед-робот не выполнил задания и к тому же был выведен из строя. Человек мог бы справиться с этими новыми проблемами, когда они возникали. Но компьютеры не способны принимать решения в непредвиденных ситуациях.

Работа по правилам

Компьютеры хорошо справляются лишь с задачами, условия которых не меняются в процессе решения и являются одинаковыми каждый раз, когда эти задачи выполняются. Но уж если компьютеры способны что-то сделать, то делают они это великолепно. Компьютеры могут работать значительно быстрее и лучше человека.

Майк работает на кондитерской фабрике. Его работа заключается в том, что он кладет вишенки на сладкие булочки.

Когда Майк накануне долго смотрит телевизор, его взор "затуманен", и вишенки не попадают точно в середину.



В другие дни любовь к сладкому берет верх, и некоторые вишенки никогда не попадают на булочки.

При этом Майк много времени посвящает мечтам о летнем отдыхе, и здесь он совершенно забывает о вишенках.

Процесс укладывания вишен представляет собой задачу, связанную с повторением одной и той же операции. Такая задача может быть решена путем выполнения набора команд, что делает ее идеальной для робота, который никогда не захочет съесть вишню и никогда не задремлет. Таким образом, машины выполняют работу подобного рода лучше людей именно потому, что они не умеют мыслить (а вот умение мыслить могло бы здесь помешать).

Компьютеры способны хорошо выполнять такие задания, которые удается представить в виде ограниченного ряда правил и команд. Даже шахматы, считающиеся игрой для умных, являются фактически игрой с правилами, которым можно "научить" компьютер.

Будущее

Сумеют ли когда-нибудь ученые заложить в компьютер столько правил, что он сможет принимать решения, как человек? Некоторые ученые считают, что нет. Другие же верят, что, когда будет больше знаний о работе человеческого мозга, людям удастся создать первые действительно мыслящие компьютеры.

Thinking machines?

Общий словарик к тексту "Thinking machines?"

storybook — сборник рассказов, сказок
exciting — возбуждающий, волнующий; захватывающий, увлекательный (о рассказе и т.п.)
power — способность; возможность; могущество; власть, влияние; мощь; полномочие; сила; мощность, энергия; производительность
dummy — кукла, чучело; манекен; модель; макет; орудие в чужих руках; марионетка; подставное лицо
busy town — шумный город
thing — вещь, предмет; нечто, кое-что; качество, свойство, особенность; создание, существо; образец
the very best players — самые лучшие игроки
to swerve — отклоняться от прямого пути; сворачивать в сторону
despite — несмотря на
to go wrong — выйти из строя
on arrival — по прибытии
to fail — потерпеть неудачу; не иметь успеха; провалить(ся) на экзаменах; изменить; покинуть; не сбываться, обманывать ожидания; не исполнить, не сделать; ослабевать, терять силы; перестать действовать, выйти из строя; недоставать, не хватать; иметь недостаток (в чем-либо)
to cope — справиться; совладать
circumstances — обстоятельства, условия; материальное положение
task — задача; задание; урок; урочная работа
never — никогда; ни разу
identical — одинаковый, идентичный, тождественный; тот же самый (об одном предмете)
single — один; единственный; единый, целый; предназначенный для одного; отдельный, обособленный; одинокий; годный в один конец (о билете)
reliable — надежный; прочный; заслуживающий доверия, достоверный
blurry — неясный, туманный; расплывчатый, смазанный; запачканный
in the middle (of) — в середине (чего-либо)
a sweet tooth — любовь к сладкому
to get the better of somebody (or something) — взять верх; преодолеть; быть хозяином положения
altogether — вполне, всецело, совершенно; в общем, в целом; всего
to tempt — искушать, соблазнять; испытывать, проверять
a limited number of — ограниченный ряд
brainy — умный; способный; остроумный

Грамматический комментарий

[1] in real life things aren't quite like that — в действительности дела обстоят не совсем так [в реальной жизни дела обстоят не совсем так]

Простое настоящее время, мн. число. Употребляется краткая отрицательная форма глагола — aren't (полная форма: are not). Для ед. числа краткая отрицательная форма — isn't, полная — is not.

[2] most of the things that people do every day, even the ones that we think are really simple, are complex — большинство тех вещей, которые люди делают каждый день, причем даже те вещи, которые кажутся нам простыми, являются сложными

Простое настоящее время. Здесь местоимение one (во мн. числе) выступает заменителем ранее упомянутого существительного thing (во мн. числе).

[3] robobike won't release it — робот не отдает ее

Краткая отрицательная форма глагола (полная форма — will not). Здесь он употребляется с существительным-подлежащим, обозначающим неодушевленный предмет, и усиливает отрицание.

[4] the receiver has grown a beard over the weekend — получатель отрастил бороду за выходные дни [уик-энд]

Настоящее совершенное время (ед. число) — в данном случае употребляется для обозначения действия, происходившего в прошлом, но актуального в настоящем. Неправильный глагол: to grow — grew — grown.

[5] a traffic light gets stuck — оказывается неисправным светофор [светофор "заклинивает"]

Простое настоящее время. В выражении употребляется неправильный глагол: to stick — stuck — stuck.

[6] a person could have coped with these new situations as they arose — человек мог бы справиться с этими новыми задачами, когда они возникали

Настоящее совершенное время (ед. число). Глагол could указывает здесь на сослагательное наклонение.

"Компьютерный" словарик

tool — средство; инструмент | орудие труда; станок
computer-aided design (CAD) — система автоматизированного проектирования (САПР)
CAD software — программное обеспечение САПР
version — версия; вариант | перевод; текст (перевода или оригинала)
pattern — образец, шаблон; образ, изображение (pattern recognition — распознавание образов)
painting — закрашивание
painting software — программы для подготовки иллюстраций
fractal — фрактал
computer graphics — компьютерная графика, машинная графика
desktop publishing (DTP) — настольное издательство, настольная издательская система
desktop publishing system (DTP system) — настольная издательская система
package — пакет; комплект | тюк; кипа; посылка; место (багажа); сверток; пачка (сигарет); упаковка
font — шрифт; гарнитура шрифта, гарнитура | церковная купель; резервуар керосиновой или масляной лампы
word processor — система обработки текстов, система подготовки текстов, текстовый процессор
intelligent — интеллектуальный | умный, разумный, понимающий; понятливый, смышленый
to make decisions — принимать решения
set of instructions — набор команд

Информатика

«Турбо-Паскаль в примерах»
автора А.Б. Николаева

Информационная революция в значительной степени увеличила объем информации, которая обрушивается на человека и которую надо уметь правильно обрабатывать. Практически каждый второй ребенок в наше время знает такие слова, как «компьютер», «Интернет» и т.п., но только каждый пятый умеет правильно и грамотно работать с информацией на компьютере. Научить этому школьников призван предмет «Информатика», преподаваемый в российских школах. Издательство «Просвещение» выпускает книгу «Турбо-Паскаль в примерах», в которой для школьников раскрываются грамматические правила Паскаля (это – язык, а следовательно, есть и алфавит, и правила построения из него всех кирпичиков, составляющих программу).

Информатика, которую изучают школьники в старших классах, имеет несколько аспектов. Разделы «Понятие и свойства алгоритмов» и «Типы и структуры данных» лучше всего объяснять, используя алгоритмический язык Паскаль, а не Бейсик или Си. Бейсик не имеет такого разнообразия структур данных и структур управления, а изучение Си, в силу использования замысловатых, но быстро и эффективно работающих конструкций, несколько затрудняет восприятие алгоритмов. Паскаль довольно просто изучить, так же как и Бейсик, но он имеет конструкции, аналогичные существующим в ПЛ/1, Алгол68, Си, только более лаконичные.

Строгое и компактное описание синтаксиса Паскаля приводится с помощью металингвистических формул Бэкуса во второй главе книги. Если с какого-то момента изучение Паскаля становится скучным и не совсем понятным, следует обратиться к следующей главе, а эту рассматривать как справочник по синтаксису конструкций языка (использование скобок, точек, запятых и пр.).

Разделы третьей главы построены по принципу усложнения алгоритмов (линейные, разветвляющиеся, циклические участки в вычислениях) и рассмотрения различных методов программирования (операторное, процедурное, функциональное, модульное, структурное, объектно-ориентированное). В каждом разделе главы присутствует теоретический материал по теме, ссылки на материал предыдущих разделов, необходимый для этих примеров, и многочисленные примеры отлаженных программ.

Язык хорошо усваивается на своих ошибках и чужих примерах. В книге рассмотрена не только работа с простыми числами или объединенными в одномерные и многомерные массивы, но и обработка таких структур данных, как множество, запись, запись с вариантным полем, строка, указатель и его использование в динамических массивах, списках. Показано использование внешних текстовых файлов для ввода-вывода данных на дискеты и диск, причем в теоретической части этого раздела даются четкие правила работы и представления данных различных типов.

Двоичные (типизированные) файлы рассмотрены в большом примере, демонстрирующем еще и метод структурного программирования.

Методы процедурного и функционального программирования показаны на пяти примерах. Рассматриваются подпрограммы, аргументами которых являются имена процедур и функций, а также процедуры и функции с нетипизированными параметрами. Приводятся примеры рекурсивных алгоритмов.

Примеры использования стандартных модулей GRAPH и CRT с подробным систематизированным описанием процедур и функций и разработка своего модуля познакомят читателя с организацией библиотек (модулей) из типов, констант, переменных, процедур, функций.

В примерах по ООП используется модуль GRAPH и демонстрируется возможность перемещения графического объекта с помощью управляющих клавиш. Также в книге рассмотрены базовые алгоритмы, такие, как:

- сортировка, двоичный поиск;
- формирование чисел Фибоначчи;
- формирование массива счетчиков без IF и CASE;
- простые числа (структура данных – множество);
- метод Ньютона для нахождения корня уравнения;
- построение графика функции в текстовом и графическом режимах, перемещение графического объекта по экрану с помощью управляющих клавиш.

Книга вполне может служить хорошим подспорьем студентам вузов непрофильных специальностей, а также всем желающим постичь основы языка Турбо-Паскаль самостоятельно.

Юбилей

9 сентября 2001 года исполнилось 60 лет Деннису Ритчи – одному из создателей языка программирования С

1972 год. Д.Ритчи (слева) и К.Томпсон (справа) за PDP-11

Язык программирования С был создан как инструмент написания операционной системы UNIX в научно-исследовательской фирме Bell Telephone Laboratories американской корпорации AT&T (*American Telephone and Telegraph*). В 1969 году Кен Томпсон начал создавать эту систему для мини-компьютера PDP-7 фирмы DEC. Вскоре Деннис Ритчи, активно помогавший разрабатывать UNIX, дополнил созданный Томпсоном "для внутреннего пользования" язык программирования B ("Би") и в 1972 году представил язык С, в котором сочетались лучшие свойства ассемблера и языков высокого уровня. От ассемблера были взяты гибкие и эффективные средства работы с памятью, а от языков высокого уровня — широкий набор управляющих конструкций, возможность использования сложных структур данных, средства ввода и вывода [1, 2, 3].

Для повышения скорости работы операционных систем при их создании обычно применялся язык низкого уровня — ассемблер, однако язык С настолько хорошо себя зарекомендовал, что на нем была написана почти вся операционная система UNIX [4, 5, 6].

Первое описание языка было приведено в книге [4]. Долгое время данное описание считалось стандартом, однако некоторые его места допускали неоднозначное толкование, что породило множество трактовок языка С. Для исправления ситуации при Американском национальном институте стандартов (*American National Standards Institute* — ANSI) был образован комитет по стандартизации языка, и в 1983 году был утвержден стандарт, получивший название ANSI C [7]. "Прямыми потомками" языка С являются языки [8]: C++ (Б.Страуструп, 1984 г.); Concurrent C (Н.Джехани, 1986 г.); Objective C (Б.Кокс, 1986 г.); C* (Thinking Machines, 1987 г.); C+@, ранее Calico (Bell Labs, 1991). Самым известным здесь стал, безусловно, C++.

После реорганизации AT&T, уже в рамках Lucent Labs, Ритчи возглавлял группу, разрабатывавшую операционную систему Plan 9 (1995 г.). Она явилась основой следующего проекта его группы — операционной системы Inferno, о создании которой было объявлено в апреле 1996 года. Для этой системы Ритчи подготовил язык Limbo [8, 9].

Сейчас Ритчи, имеющий степень бакалавра по физике и магистра по прикладной математике, возглавляет отдел исследований в области системного программного обеспечения в фирме Bell Labs.

В 1983 году Деннису Ритчи и Кену Томпсону за разработку и реализацию языка программирования С и операционной системы UNIX была вручена премия Тьюринга. В 1988 году Ритчи был избран в Американскую национальную инженерную академию (National Academy of Engineering).

Почему же язык С покорил сердца миллионов программистов? Вот ответ самого Денниса Ритчи [8]: "Язык С — это невероятный, оглушительный и огромный успех. Хотя повороты судьбы ему помогали, он безусловно удовлетворил потребности в таком языке реализации систем, который был достаточно эффективным, чтобы заменить ассемблер, и в то же время достаточно абстрактным и гибким, чтобы описывать алгоритмы и взаимодействия в широком спектре программных сред". При этом "существует масса прекрасных языков (прекраснее, чем С), которые не привились, — отмечает Ритчи, — но кто-то все же выигрывает..."

Литература

1. *Березин Б.И., Березин С.Б.* Начальный курс С и С++. М.: Диалог-МИФИ, 1999.
2. *Дукаревич Г.Б.* Введение в программирование на языке Си // Вычислительная техника и ее применение. № 4/91.
3. Эй, Би и, наконец, Си // Информатика, № 5/2000.
4. *Kernighan B.W. and Ritchie D.M.* The C Programming Language. Prentice-Hall, 1978. [*Керниган Б., Ритчи Д.* Язык программирования Си: Пер. с англ. М.: Финансы и статистика, 1985.]
5. Язык компьютера: Пер. с англ. М.: Мир, 1989.
6. *Болски М.И.* Язык программирования Си. Справочник: Пер. с англ. М.: Радио и связь, 1988.
7. *Стариков Ю.А.* Мобильность программ и особенности реализации языка Си. Изд. 2-е, доп. М.: МП "Память", 1992.
8. *Богатырев Р.* Летопись языков. Си // Мир ПК, № 8/2001.
9. *Мосени П.* Inferno: ОС для сетевых компьютеров // Мир ПК, № 10/1997.

Гл. редактор
С.Л. Островский
Зам. гл. редактора
А.И. Сенокосов
Редакция:
Е.В. Андреева
Н.Л. Беленькая
Л.Н. Картвелишвили
Н.П. Медведева
Дизайн и верстка:
Н.И. Пронская
Корректоры:
Е.Л. Володина,
С.М. Подберезина

выходит четыре раза в месяц
При перепечатке ссылка
на ИНФОРМАТИКУ обязательна,
рукописи не возвращаются

Адрес редакции и издателя:
121165, Киевская, 24
тел. 249-48-96
Отдел рекламы
тел. 249-98-70

Учредитель: ООО "Чистые пруды"

Зарегистрировано в Министерстве РФ по делам печати. ПИ № 77-7230 от 12.04.2001.
Отпечатано в ОИД "Медиа-Пресса",
125993, ГСП-3, Москва, А-40, ул. "Правды", 24.
Тираж 6000 экз.
Срок подписания в печать по графику 05.09.2001.
Номер подписан 05.09.2001.
Заказ № 15648
Цена свободная

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
комплекта изданий 32744

Тел.: (095)249-31-38, 249-33-86. Факс (095)249-31-84

Internet: inf@1september.ru
WWW: http://www.1september.ru

Главный редактор комплекта изданий – А.С. Соловейчик

Английский язык (Е.В. Громушкина), индекс подписки — 32025; **Библиотека в школе** (О.К. Громова), индекс подписки — 33376; **Биология** (Н.Г. Иванова), индекс подписки — 32026; **Воскресная школа** (монах Киприан (Ященко), индекс подписки — 32742; **География** (О.Н. Коротова), индекс подписки — 32027; **Здоровье детей** (А.У. Лекманов), индекс подписки — 32033; **Информатика** (С.Л. Островский), индекс подписки — 32291; **Искусство** (Н.Х. Исмаилова), индекс подписки — 32584; **История** (А.Ю. Головатенко), индекс подписки — 32028; **Литература** (Г.Г. Красухин), индекс подписки — 32029; **Математика** (И.Л. Соловейчик), индекс подписки — 32030; **Начальная школа** (М.В. Соловейчик), индекс подписки — 32031; **Немецкий язык** (М.Д. Бузоева), индекс подписки — 32292; **Русский язык** (Л.А. Гончар), индекс подписки — 32383; **Спорт в школе** (Н.В. Школьникова), индекс подписки — 32384; **Управление школой** (А.И. Адамский), индекс подписки — 32652; **Физика** (Н.Д. Козлова), индекс подписки — 32032; **Французский язык** (Г.А. Чесновицкая), индекс подписки — 33371; **Химия** (О.Г. Блохина), индекс подписки — 32034; **Школьный психолог** (М.Н. Сартан), индекс подписки — 32898.