УДК 373

Э.М. Булатова,
И.Т. Халкечева

Методика обучения разработке эффективных алгоритмов решения задач в школьном курсе информатики

В статье рассмотрены учебные задачи школьного курса информатики, посвященные способам их решения и оптимизации алгоритмов. Акцентируется внимание на том, какие сложности возникают у школьников в процессе решения задач на программирование с учетом оптимизации алгоритмов по времени выполнения и по используемым машинным ресурсам.

Ключевые слова: обучение школьников информатике; эффективные алгоритмы решения задач по информатике; алгоритмизация; оптимизация.

Содержательная линия школьного курса информатики «Алгоритмизация и программирование» претерпевает существенные изменения с начала обучения информатике в школе. Достаточно широкое распространение в последние годы получает позиция тех специалистов, кто считает эту содержательную линию невостребованной в связи с тем, что лозунг «программирование — вторая грамотность» потерял свою актуальность и стало очевидно, что программирование — это не вторая грамотность, а достаточно специфический вид деятельности человека, для занятия которой необходимо владеть широким спектром знаний и умений, а также способностью к системно-логическому мышлению. Сегодня изучение программирования носит больше профориентационный характер. В то же время успешное овладение элементами алгоритмизации и программирования усиливает фундаментальный характер знаний содержания школьной информатики (см., например, [2; 5–11]). Таким образом, алгоритмизация и программирование остаются самой главной содержательной линией курса информатики. Обучение алгоритмизации и программированию позволяет учащимся испытать свои способности к профессиональной деятельности в области программирования (см., например, [1; 3–6; 8; 12]).

Что касается спецификации ЕГЭ по информатике и ИКТ, то здесь распределение по линиям изучения информатики имеет следующий вид:

- алгоритмизация и программирование — 65 б;
- компьютер (архитектура) — 3 б;
- информационные технологии — 9 б;
- информация, информационные процессы, представление информации — 17 б;
- формализация и моделирование — 6 б.

Таким образом, алгоритмизация и программирование остаются значимым предметом по объему содержания в школьном курсе информатики и требуют особого внимания и методически обоснованного подхода к выбору содержания обучения.

Наибольшие сложности у учащихся в процессе изучения алгоритмизации и программирования вызывают задачи на программирование с учетом оптимизации алгоритмов по времени выполнения и по используемым машинным ресурсам. Это, прежде всего, задачи, связанные с работой с массивами, записями и рекурсивными функциями.

Массивы — упорядоченная совокупность элементов одного типа, когда каждому элементу массива данных (чисел) ставится в соответствие порядковый номер, называемый индексом. Количество элементов массива фиксировано и в языке Pascal должно быть указано заранее в разделе описаний программы. Такой способ образования новых значений позволяет обозначать значения этих типов одним именем, а доступ к отдельным элементам массивов организовывать посредством указания этого группового имени и порядкового номера (индекса) необходимого элемента.

Если индекс состоит из одного числа, то массив называется одномерным, или вектором. Если индекс состоит из двух чисел, то массив называется двумерным, или матрицей. Приведем примеры.

1. A = (84; –25; 0; 2,6; –462; 7,4; 3) = (A1, A2, A3, A4, A5, A6, A7), где A — одномерный массив (вектор), размерностью 7, т. е. состоит из 7 элементов; A2 — элемент массива A, имеющий индекс (порядковый номер) 2.

Для этого массива A2 = –25; A4 = 2,6; A7 = 3.

2. $B = \begin{pmatrix} 3 & -5 & 7 \\ 9 & 0 & -7 \end{pmatrix}$, здесь B — двумерный массив размерностью 2 × 3,

т. е. количество строк равно 2, количество столбцов равно 3.

B_{12} — элемент матрицы B, стоящий на пересечении 1-й строки и 2-го столбца.

B_{21} — элемент матрицы B, стоящий на пересечении 2-й строки и 1-го столбца.

Все массивы, встречающиеся в программе, должны быть описаны в строке var. Одномерные массивы:

var <имя>: array [n1..n2] of <тип>,

где <имя> — имя массива; array — служебное слово, означает «массив»; n1, n2 — минимальное и максимальное значение индекса; of — служебное слово, означает «из»; <тип> — тип компонент массива, может быть любой тип.

n1, n2 — константы целого типа (в общем случае последовательные значения других скалярных типов, кроме вещественного).

Для двумерных массивов нужно указать минимальное и максимальное значение для каждого значения индекса:

```
var <имя>: array |n1..n2, m1..m2| of <тип>.
```

Пример:

```
var a, s: array [1..7] of real;
z, vect1, vect2: array [0..5] of integer;
p: array [-8..72] of char;
var в, x4, pg: array [1..2,1..3] of integer;
q, nu: array [-2..2,0..4] of real.
```

Имя элемента массива образуется из имени массива, за которым в квадратных скобках следует индекс элемента (для многомерных массивов — индексы через запятую). Индексы могут быть выражениями.

При выполнении программы проверяется, не выходит ли индекс массива за границы, объявленные при описании этого массива. Если выходит, то возникает аварийная ситуация в программе.

В языке Pascal существует единственная операция с именем массива, а именно: одному массиву можно присвоить другой массив, если при описании они эквивалентны.

```
var a, в: array [1.. 5] of integer;
к: integer;
begin
for к := 1 to 5 do a [k] := k*k;
в := a ;
for к := 1 to 5 do writeln (b[k]:3);
end.
```

В результате работы этой программы мы получим числа 1, 4, 9, 16, 25.

В процедурах ввода и вывода не допускается использование имен массивов, поэтому для организации ввода и вывода данных в массивы используются циклы.

Задачи с массивами можно подразделить на следующие группы:

– нахождение сумм, произведений, количества некоторых элементов массива;
– преобразование массива;
– выборка из массива, формирование нового массива;

– нахождение минимального и максимального элементов массива;

– сортировка массива, т. е. упорядочение массива по возрастанию или убыванию элементов.

Записи относятся к составному типу. Если массив — это фиксированное число элементов одного типа, то записи — это фиксированное количество элементов разных типов. Компоненты записи называются полями. Количество полей строго фиксировано. Тип поля может быть любой из стандартных типов: перечисляемый, ограниченный, строковый тип, массивы, записи.

Имя поля должно быть уникально (единственно) в пределах одной записи.

Описание записи:

```
var <имя>: record
<имя поля 1>: <тип 1>;
<имя поля 2>: <тип 2>;
...
<имя поля n>: <тип n>;
end;
```

где <имя> — имя записи, идентификатор; record — служебное слово, означает «запись»; <имя поля 1> — имя первого поля, идентификатор; <тип 1> — тип первого поля; end — служебное слово, означает «конец».

Например,

```
var klass: record
nom: 1..11;
buk: char;
end.
```

Переменная klass комбинированного типа (запись) имеет два поля: nom — ограниченного типа, базовый — целый; buk — символьного типа.

Тип элемента записи соответствует типу поля. Для обращения к элементу записи нужно указать:

<имя записи>.<имя поля>.

Например:

klass.nom — ограниченного типа (klass.nom:=3).

klass.buk — символьного типа (klass.buk:='a').

Над элементом записи можно совершать операции, соответствующие типу их поля. Над всей записью в целом можно делать только одну операцию — операцию присваивания.

Например:

```
type klass = record
nom: 1..11;
buk: char;
end;
```

```
var k1,k2: klass;
begin
k1.nom:=7;
k1.buk:='б';
k2 := k1;
writeln (k2.nom, k2.buk)
end.
```

Рекурсивной функцией называется такая функция, которая в процессе выполнения вызывает сама себя. Достоинством рекурсивных определений является то, что они позволяют с помощью конечных формул определять бесконечное множество объектов.

Если не принять специальных мер, рекурсия становится бесконечной. Чтобы процесс рекурсии когда-нибудь завершился, необходимо рекурсивный вызов поместить внутри условного оператора, когда одна ветвь этого оператора содержит рекурсивный вызов, а другая — нет. Переход на ветвь условного оператора, не содержащую рекурсивный вызов, но обеспечивающую завершение работы рекурсивной подпрограммы, должен произойти через конечный промежуток времени.

Пример. Вывести первые 15 чисел Фибоначчи и вычислить их сумму. Числа Фибоначчи: 1, 1, 2, 3, 5, 8, 13, 21, … ( ).

```
program fibon;
const n=15;
label m1;
var
f1, f2, f3, S, i: longint;
begin
f1:=1; f2:=1; i:=2; S:=2;
m1: f3:=f1+f2;
write(f3:8);
f1:=f2; f2:=f3; i:=i+1; S:=S+f3;
if i<n then goto m1;
writeln ('сумма=', S);
end.
```

Решение этих задач требует от учащихся умения писать эффективные программы. Под эффективной программой подразумевается, что она использует минимально необходимую память, и алгоритм ее решения имеет минимальную сложность. Поэтому очень важно, чтобы учащиеся были знакомы с понятием сложности алгоритма и на примерах могли подсчитать сложность предложенного алгоритма. Кроме того, необходимо, чтобы они были знакомы с базовыми алгоритмами обработки данных.

Во время решения задач учащихся желательно знакомить с разными способами решения одной и той же задачи, вместе с ними вычислять сложность алгоритма, при необходимости находить пути уменьшения сложности

алгоритма. Но прежде чем учащиеся научатся писать эффективные алгоритмы, они должны в принципе научиться решать задачи.

При решении задач на обработку массивов желательно поступать следующим образом:

1. Решать специально подобранные базовые технические задачи, которые позволяют оттачивать технику работы с массивами.

2. Решать специально подобранные базовые задачи, алгоритмы которых являются составляющими многих задач на обработку массивов данных.

3. Учиться читать чужие программы. В этой ситуации уместно провести аналогию с изучением иностранного языка: вначале человек, изучающий иностранный язык, учится читать текст со словарем. Так и в программировании. Умение уровня «читать текст со словарем» достигается за счет выполнения заданий сначала на листе бумаги, а затем и с использованием отладчика программ. Учащийся должен понимать, как выполняются основные алгоритмические конструкции, как организованы одномерные и двумерные массивы, т. е. знать правила работы с каждым типом данных.

4. Знакомить учащихся с основными базовыми положениями теории алгоритмов. В частности, рассказать, что алгоритмы решения задач можно классифицировать, например, следующим образом:

– решение задачи «в лоб»;
– метод введения дополнительных данных;
– метод преобразования входных данных;
– метод уменьшения размерности задачи.

Необходимо решать специально подобранные задачи по каждому методу.

5. Учиться анализировать программу на предмет вычислительной сложности. Если мы знаем алгоритм меньшей сложности, то предлагать учащимся попытаться найти его (или знакомить учащихся с таким алгоритмом).

Литература

1. *Гербеков Х.А., Башкаева О.П.* Объектно-ориентированное программирование в школьном курсе информатики // Вестник Российского университета дружбы народов. Серия «Информатизация образования». 2017. Т. 14. № 2. С. 156–160.

2. *Зубов А.А.* Программирование на Delphi. Трюки и эффекты. СПб.: Питер, 2005. 396 с.

3. *Зыков С.В.* Введение в теорию программирования: учеб. пособие. М.: Интуит.ру, 2004. 400 с.

4. *Иванова Г.И.* Основы программирования: учебник. М.: МГУ, 2004. 416 с.

5. *Кудинов Ю.И.* Основы современной информатики. СПб: Лань, 2009. 256 с.

6. *Лапчик М.П., Семакин И.Г., Хеннер Е.К.* Теория и методика обучения информатике. М.: Академия, 2008. 592 с.

7. *Малыхина М.П.* Программирование на языке высокого уровня Turbo Pascal. СПб.: BHV, 2006. 544 с.

8. *Незнанов А.А.* Программирование и алгоритмизация. М.: Академия, 2010. 304 с.

9. *Рагимханова Г.С.* Программирование на Turbo Pascal. Махачкала: ДГПУ, 2013. 100 с.

10. *Сурхаев М.А.* Развитие системы подготовки будущих учителей информатики для работы в условиях новой информационно-коммуникационной образовательной среды: автореф. дис. … д-ра пед. наук. М., 2010. 46 с.

11. *Сурхаев М.А.* Умения, необходимые учителю для работы в образовательной среде, основанной на средствах ИКТ // Стандарты и мониторинг в образовании. 2008. № 6. С. 50–51.

12. *Уваров В.М., Силакова Л.А., Красникова Н.Е.* Практикум по основам информатики и вычислительной техники. М.: Академия, 2012. 240 с.

13. *Фаронов В.В.* Turbo Pascal: учеб. пособие. М.: Нолидж, 2009. 367 с.

Literatura

1. *Gerbekov H.A., Bashkaeva O.P.* Ob''ektno-orientirovannoe programmirovanie v shkol'nom kurse informatiki // Vestnik Rossijskogo universiteta druzhby' narodov. Seriya «Informatizaciya obrazovaniya». 2017. T. 14. № 2. S. 156–160.

2. *Zubov A.A.* Programmirovanie na Delphi. Tryuki i e'ffekty'. SPb.: Piter, 2005. 396 s.

3. *Zy'kov S.V.* Vvedenie v teoriyu programmirovaniya: ucheb. posobie. M.: Intuit.ru, 2004. 400 s.

4. *Ivanova G.I.* Osnovy' programmirovaniya: uchebnik. M.: MGU, 2004. 416 s.

5. *Kudinov Yu.I.* Osnovy' sovremennoj informatiki. SPb: Lan', 2009. 256 s.

6. *Lapchik M.P., Semakin I.G., Xenner E.K.* Teoriya i metodika obucheniya informatike. M.: Akademiya, 2008. 592 s.

7. *Maly'xina M.P.* Programmirovanie na yazy'ke vy'sokogo urovnya Turbo Pascal. SPb.: BHV, 2006. 544 s.

8. *Neznanov A.A.* Programmirovanie i algoritmizaciya. M.: Akademiya, 2010. 304 s.

9. *Ragimxanova G.S.* Programmirovanie na Turbo Pascal. Maxachkala: DGPU, 2013. 100 s.

10. *Surxaev M.A.* Razvitie sistemy' podgotovki budushhix uchitelej informatiki dlya raboty' v usloviyax novoj informacionno-kommunikacionnoj obrazovatel'noj sredy': avtoref. dis. … d-ra ped. nauk. M., 2010. 46 s.

11. *Surxaev M.A.* Umeniya, neobxodimy'e uchitelyu dlya raboty' v obrazovatel'noj srede, osnovannoj na sredstvax IKT // Standarty' i monitoring v obrazovanii. 2008. № 6. S. 50–51.

12. *Uvarov V.M., Silakova L.A., Krasnikova N.E.* Praktikum po osnovam informatiki i vy'chislitel'noj texniki. M.: Akademiya, 2012. 240 s.

13. *Faronov V.V.* Turbo Pascal: ucheb. posobie. M.: Nolidzh, 2009. 367 s.

E.M. Bulatova,
I.T. Khalkecheva

Methods of Teaching Development of Effective Algorithms of Solving Tasks in the School Course of Computer Science

In the article the educational tasks of the school course of computer science devoted to the optimization of algorithms and methods for their solution are considered. Attention is focused on the difficulties that students face in the process of solving programming problems, taking into account the optimization of algorithms in terms of execution time and the machine resources used.

Keywords: teaching computer science to students; effective algorithms for solving problems in computer science; algorithmization; optimization.