УДК 681.3

**В.П. Гладков**

Пермский государственный технический университет

# СИСТЕМА УПРАЖНЕНИЙ ДЛЯ ВЫРАБОТКИ НАВЫКОВ АЛГОРИТМИЗАЦИИ

*Рассматривается методика построения алгоритмов решения сложных задач.*

Композитор Людвиг Шпор, живший на рубеже XVIII–XIX вв., изобрел дирижерскую палочку, но в историю музыки вошел благодаря тому, что написал музыкально-педагогическое сочинение – «Школа игры на скрипке».

Материал «школы» состоит из сравнительно коротких нотных текстов, имеющих характер упражнений. Тексты располагаются в таком порядке, что освоение каждого из них создает основу для исполнения дальнейших текстов, многие из которых сами по себе, вне контекста, могут представляться трудными. Однако, будучи включенными в «школу», осваиваются сравнительно легко [1].

Подобный прием дает хорошие результаты в обучении любому предмету, где большую роль играет творческая составляющая мастерства. Несомненно, к таким предметам относится и алгоритмизация – наука и искусство создания алгоритмов.

Для построения подобной системы упражнений необходимо установить систему правил.

Рассмотрим пример 1. В квадратном массиве размерностью $n \times n$ требуется переставить местами строку с номером $p$ и столбец с номером $q$. Предполагается, что строки и столбцы нумеруются с единицы.

Для случая $p = q$ заданное преобразование трудности не представляет. Например, при $p = 3$, $q = 3$ для массива $\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$, поменяв

элементы заданных строки и столбца симметричных главной диаго-

$$\begin{matrix} 1 & 2 & 7 \\ 4 & 5 & 8 \\ 3 & 6 & 9 \end{matrix}$$

нали, получаем .

Однако если $p \neq q$, алгоритм преобразования становится неочевидным из-за проблем элемента, принадлежащего как строке $p$, так и столбцу $q$.

Для решения основной задачи сформулируем последовательность промежуточных задач, решение которых приводит к решению исходной. Рассмотрим решение для случая $p = 2$, $q = 3$.

1) Переставить местами строки с номерами $p$ и $q$.

$$\begin{matrix} 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \end{matrix}$$

Получим .

```
for i := 1 to n do
begin r := a[p, i]; a[p, i] := a[q, i];
a[q, i] := r; end;
```

2) Переставить местами строку с номером $p$ и столбец с номером $q$. Получим

$$\begin{matrix} 1 & 2 & 4 \\ 7 & 8 & 5 \\ 3 & 9 & 6 \end{matrix}$$

.

```
for i := 1 to n do
begin r := a[p, i]; a[p, i] := a[i, q];
a[i, q] := r; end;
```

3) Переставить местами строки с номерами $p$ и $q$. Получим нужный результат

$$\begin{matrix} 1 & 2 & 4 \\ 3 & 9 & 6 \\ 7 & 8 & 5 \end{matrix}$$

.

```
for i := 1 to n do
begin r := a[p, i]; a[p, i] := a[q, i]; a[q, i]
:= r; end;
```

Анализ полученного решения подсказывает еще один алгоритм, в котором осуществляется перескок через значения индексов, совпадающие с номерами меняемых строки и столбца. Этот алгоритм может быть реализован следующей программой на Паскале:

```
    i := 1; //индекс строки
    j := 1; //индекс столбца
    while (i <= n) and (j <= n) do
      if i = p then i := i + 1     //перескок
в строке
      else if j = q then j := j + 1    //перескок
в столбце
           else begin r := a[p, j];  a[p, j] := a[i, q];
a[i, q] := r;  i := i + 1;  j := j + 1 end.
```

Рассмотренный пример 1 позволяет сделать вывод о том, что для получения искомых упражнений нужно вначале научиться решать поставленную задачу в частных случаях или для конкретных значений исходных данных, затем полученный алгоритм следует обобщить для общего случая. Для получения частного случая следует либо добавить ограничивающее условие, либо задать конкретные (константные) значения исходным переменным, либо выполнить оба условия [2].

Рассмотрим пример 2. Пусть необходимо сформировать двумерный массив размером $n \times m$ так, чтобы любой его подмассив размером $p \times q$ ($p < n$, $q < m$) имел одинаковую сумму элементов, равную $s$.

Формируем подзадачу. Отбрасываем условие произвольности двумерного массива и переходим к квадратному: $n = m$ и $p = q$. Задаемся конкретными значениями переменных, подбирая их так, чтобы облегчить решение задачи и увидеть закономерности решения: $n = 3$, $p = 2$.

Предположим, что сумма элементов в массиве 2×2 должна быть равна $s = 10$. Поскольку такая сумма должна быть у любого подмассива, то, заполняя подмассив в левом верхнем углу, получим:

$$\begin{matrix} 1 & 2 & . \\ 3 & 4 & . \\ . & . & . \end{matrix}$$

Анализ первых двух строк показывает, что для получения недостающих элементов в правом верхнем углу нужно скопировать первый столбец:

$$\begin{matrix} 1 & 2 & 1 \\ 3 & 4 & 3 \\ . & . & . \end{matrix}$$

Аналогично для получения подмассива в левом нижнем углу нужно скопировать первую строку:

$$\begin{matrix} 1 & 2 & 1 \\ 3 & 4 & 3 \\ 1 & 2 & . \end{matrix}$$

Для получения недостающего элемента копируем первую строчку правого верхнего подмассива: $\begin{matrix}1 & 2 & 1\\ 3 & 4 & 3\\ 1 & 2 & 1\end{matrix}$.

Обобщая полученное решение, т.е. добавляя отброшенное условие и переходя к исходным переменным, задающим размерности массивов, получаем следующий алгоритм решения примера 2:

1) генерируем массив размерностью $p \times q$, содержащий элементы, сумма которых равна $s$.

```
readln(s);  //сумма элементов подмассива
k := 1;       //используется для формирования элементов
for i := 1 to p do  //перебор строк
for j := 1 to q do begin a[i, j] := k;  s := s - k ;  k := k + 1; end; //формирование элементов.
```

2) формируем верхний ряд подмассивов, переставляя столбики.

```
for k := q + 1 to n do
for i := 1 to p do a[i, k] := a[i, k - q].
```

3) формируем нижние строки.

```
for k := p + 1 to n do
for j := 1 to n do a[k, j] := a[k - p, j].
```

Другим приемом получения системы обучающих задач является использование разных структур данных для одной задачи.

Рассмотрим пример 3. Пусть необходимо вычислить значение многочлена $y = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_2 \cdot x^2 + a_1 \cdot x^1 + a_0$, где все числа целые. Коэффициенты $a_i$ являются первыми членами арифметической прогрессии, определяемой первым элементом ($q$) и разностью между соседними элементами ($d$). Например, при $q = 2$, $d = 3$, $n = 5$, $x = 4$ необходимо вычислить выражение $2 \cdot 4^5 + 5 \cdot 4^4 + 8 \cdot 4^3 + 11 \cdot 4^2 + 14 \cdot 4^1 + 17 \cdot 4^0$.

Если воспользоваться буквальным пониманием этой записи, то программа будет представлять сумму степеней $x$ и может быть записана на Паскале

```
y := 0;  q := 2;  d := 3;  n := 5;  x := 4;
for i := n downto 0 do
begin u := 1;
      for j := 1 to i do u := u*x;
      y := y + q*u;
```

```
        q := q + d;
    end;
    writeln(y).
```

Изменим структуру данных, перейдя в записи многочлена к схеме Горнера. Для этого будем последовательно выносить степень $x$ в каждой паре первых слагаемых. Получим:

$$(2\cdot4 + 5)\cdot4^4 + 8\cdot4^3 + 11\cdot4^2 + 14\cdot4^1 + 17\cdot4^0 =$$
$$= ((2\cdot4 + 5)\cdot4 + 8)\cdot4^3 + 11\cdot4^2 + 14\cdot4^1 + 17\cdot4^0 =$$
$$= (((2\cdot4 + 5)\cdot4 + 8)\cdot4 + 11)\cdot4^2 + 14\cdot4^1 + 17\cdot4^0 =$$
$$= ((((2\cdot4 + 5)\cdot4 + 8)\cdot4 + 11)\cdot4 + 14)\cdot4^1 + 17\cdot4^0 =$$
$$= (((((0\cdot4 + 2)\cdot4 + 5)\cdot4 + 8)\cdot4 + 11)\cdot4 + 14)\cdot4 + 17.$$

В этом случае программа на Паскале будет такой:

```
q := 2;  d := 3;  n := 5;  x := 4;
y := 0;
for i := 5 downto 0 do
begin y := y*x + q;
         q := q + d;
end;
writeln(y).
```

Легко заметить, что в этом случае исчез вложенный цикл, следовательно, результат будет получен быстрее.

Снова изменим структуру данных и перейдем к польской инверсной записи Я. Лукашевича, в которой операнды предшествуют знаку операции. В этом случае вычисляемый многочлен примет вид

$$4\ 2\cdot5 + 4\cdot8 + 4\cdot11 + 4\cdot14 + 4\cdot17 +.$$

Программа на Паскале, вычисляющая многочлен, представленный в таком виде, может быть следующей:

```
    //Процедура вычисляет либо сумму, либо произ-
ведение
    procedure part(p, q : longint; t : Boolean;
var z : longint);
    begin if t then z := p + q else z := p * q;
end;

    //Фрагмент программы, вычисляющий многочлен.
    q := 2;  d := 3;  n := 5;  x := 4;
    y := q;
```

```
for i := 1 to n do
begin part(y, x, false, y);
          part(q, d, true, q);
          part(y, q, true, y);
end;
writeln(y).
```

Хорошим упражнением является продолжение работы над готовой программой, повышение ее эффективности.

Рассмотрим пример 4. Требуется написать программу для решения старинной задачи: «Сколько можно купить быков, коров и телят, если плата за быка – 10 рублей, за корову – 5 рублей, за теленка – 0,5 рубля. Необходимо на 100 рублей купить 100 голов скота».

Введем обозначения: $b$ – количество купленных быков; $k$ – количество купленных коров; $t$ – количество купленных телят.

Тогда решение задачи сведется к решению системы уравнений:

$$\begin{cases} 10 \cdot b + 5 \cdot k + 0{,}5 \cdot t = 100, \\ b + k + t = 100. \end{cases}$$

Чтобы вычисления вести в целых числах, перепишем систему:

$$\begin{cases} 20 \cdot b + 10 \cdot k + t = 200, \\ b + k + t = 100. \end{cases}$$

Установим ограничения: на 100 рублей можно купить:

– не более 100/10 = 10 быков, т.е. $0 \le b \le 10$,

– не более 100/5 = 20 коров, т.е. $0 \le k \le 20$,

– не более 100/0,5 = 200 телят, т.е. $0 \le t \le 200$.

Полученная модель позволяет написать программу:

```
var b, k, t : integer;
begin
 for b := 0 to 10 do
 for k := 0 to 20 do
 for t := 0 to 200 do
 if (20*b + 10*k + t = 200) and (b + k + t = 100)
 then writeln('быков ', b, 'коров ', k, 'телят ', t);
end.
```

Программа позволяет получить ответ, но подсчитаем, сколько раз выполняется условие в теле циклов.

Цикл по $b$ выполняется 11 раз, цикл по $k$ – 21 раз, цикл по $t$ – 201 раз, поскольку циклы вложенные, то окончательный результат получаем перемножением: $11 \cdot 21 \cdot 201 = 46\,431$ раз.

Для ускорения вычислений можно $t$ вычислять непосредственно из уравнения. Это позволит отбросить цикл и сократить проверку. Получаем программу:

```
var b, k, t : integer;
begin
 for b := 0 to 10 do
 for k := 0 to 20 do
 begin t := 100 - (b + k);
    if 20*b + 10*k + t = 200
    then writeln('быков ', b, 'коров ', k,
'телят ', t);
end; end.
```

В этой программе цикл по $b$ выполняется 11 раз, цикл по $k$ – 21 раз, поскольку циклы вложенные, то окончательный результат получаем перемножением: $11 \cdot 21 = 231$ раз.

Можно еще сократить количество проверок, если ограничение по коровам пересчитывать в зависимости от выбранного количества быков. Это приводит к программе:

```
var b, k, t : integer;
begin
 for b := 0 to 10 do
 for k := 0 to (100 - 10*b) div 5+1 do
 begin t := 100 - (b + k);
      if 20*b + 10*k + t = 200
      then writeln('быков ', b, 'коров ', k,
'телят ', t);
end; end.
```

В этой программе, если количество быков равно нулю, вложенный цикл выполнится максимальное количество раз – 21. Если количество быков максимально (10), то цикл выполнится только 1 раз. Нетрудно заметить, что при увеличении количества купленных быков количество выполненных проверок будет уменьшаться на 2. Для подсчета количества проверок нужно найти сумму: $21 + 19 + 17 + \ldots + 1 = 121$.

Таким образом, преобразования исходной программы позволили уменьшить количество проверок с 46 431 до 121. Это говорит

о том, что анализ готовой программы может быть не менее интересен, чем построение алгоритма.

Описанные подходы к построению системы упражнений позволяют сформулировать достаточное количество задач различной сложности, что обеспечивает выработку навыков алгоритмизации и программирования даже при самостоятельных занятиях студента.

**Библиографический список**

1. Глазман И.М., Любич Ю.И. Конечномерный анализ в задачах. – М.: Наука, 1969.

2. Гладков В.П. Лекции по программированию для начинающих. – Пермь: Изд-во Перм. гос. техн. ун-та, 1998.

Получено 04.10.2010