ждении и введении в действие федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 040100 Социология (квалификация (степень) «бакалавр»)». URL: www.consultant.ru – Официальный сайт компании «Консультант Плюс».

3. *Сальникова Н.А.* Адаптивное тестирование как инструмент повышения качества учебного процесса // Известия Волгоградского государственного технического университета. 2011. Т. 8. № 10 (83). С. 126–129.

4. *Сальникова Н.А., Михнев И.П.* Проведение аттестации знаний студентов с помощью компьютерного тестирования // Известия Волгоградского государственного технического университета. 2007. Т. 4. № 7 (33). С. 182–185.

5. *Астафурова О.А., Сальникова Н.А., Кулагина И.И.* Интеграция научных разработок в обучении бакалавров экономического профиля // Известия Волгоградского государственного технического университета. 2014. Т. 11. № 14 (141). С. 12–14.

6. *Лопухов Н.В., Науменко И.И.* Применение системного моделирования в образовательном процессе на примере модели «расширение сферы деятельности» // Альманах современной науки и образования. 2011. № 8. С. 120–122.

Experience of development of the course “Statistical information in SPSS”

***Irina Petrovna Medintseva**, Ph.D., Associate Professor of Information Systems and Mathematical Modelling, Volgograd branch of the Russian Presidential Academy of National Economy and Public Administration*

In article the problem of creation of the course «Statistical Information in SPSS» is considered, its structure, the purpose and tasks are defined. The course role in formation of competences when training students sociologists is noted. Importance of the solution of the professional focused tasks is shown.

Keywords: training of students sociologists, competences, statistical information, the analysis of data in SPSS

УДК [004.421+004.438](076.5)

МЕТОДИКА БЫСТРОГО ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ НА ОСНОВЕ ИЗУЧЕНИЯ КЛАССОВ ЗАДАЧ (8-10)

***Юрий Александрович Аляев**, доцент, доцент кафедры программного обеспечения вычислительной техники и автоматизированных систем,*
E-mail: alyr1@yandex.ru,
Пермский военный институт внутренних войск МВД России,
http://pvivv.ru

Предлагается методика быстрого обучения программированию на основе изучения классов задач, разработанная и применяющаяся на практике в процессе обучения программированию студентов вузов.

Ключевые слова: алгоритм, программа, язык программирования Паскаль, массив

Введение

Разделы курса «Информатика» – алгоритмизация и программирование – остаются наиболее важными для формирования алгоритмического мышления. Поскольку в школах данные разделы преподаются в недостаточном объеме, в вузе возникает необходи-

мость начинать обучение с нуля и достичь хорошего уровня программирования при ограниченном количестве часов преподавания.

Этого удается добиться за счет применения рациональных методов обучения, прежде всего, последовательно проводя идеи обучения на основе выделения элементарных операций деятельности по построению алгоритмов и программ; выявления структуры алгоритма и форм ее записи на алгоритмическом языке; одинаковой формы алгоритма для решения задач с одинаковой структурой исходных данных [1–2]. Благодаря этим идеям, задачи по программированию удается разбить на ряд классов и типизировать методы решения задач каждого класса.

Ю.А. Аляев

Предлагаемая методика быстрого обучения программированию на основе изучения классов задач, появилась и применяется на протяжении многих лет в процессе обучения программированию студентов пермских вузов благодаря В.П. Гладкову [2]. В статье рассматриваются методики решения по трем (8–10) из девятнадцати выделенных классов задач (1–7 классы задач рассмотрены в [3–5]).

8 Задачи, решаемые методом перебора

Метод перебора состоит в том, что алгоритм генерирует все возможные комбинации исходных данных и отбирает те из них, которые удовлетворяют заданным условиям. В математике этот метод не получил широкого распространения, к тому же часто прямое применение метода перебора приводит к очень большому количеству вариантов, перебрать которые за обозримое время не может даже современный компьютер. Однако применение этого метода в программировании оправдано тем, что для многих задач получить решение с помощью перебора гораздо проще, чем с помощью других методов. В результате очень сложные задачи, решить которые мог только хорошо подготовленный математик, поддаются усилиям человека с довольно скромными математическими способностями. Кроме того, можно принять некоторые меры для уменьшения количества перебираемых вариантов за счет исключения заведомо неперспективных.

Задача 1. Для каждого четырехзначного числа составляется дробь: отношение суммы цифр числа к самому числу. Найдите четырехзначное число, для которого эта дробь наибольшая.

Решение. Числу 1234 соответствует дробь 0,0081, а числу 9876 – 0,003. Для перебора всех четырехзначных чисел организуем четыре вложенных цикла. В первом цикле будут перебираться цифры в разряде тысяч от 1 до 9 (нуль исключается, потому что число заведомо четырехзначное), во втором – сотен от 0 до 9, в третьем – десятков от 0 до 9, в четвертом цикле – единиц от 0 до 9. Исходным значением искомого отношения может быть нуль.

```
y:=0; {начальное значение отношения}
a:=0; {искомое число}
for i:=1 to 9 do
   for j:=0 to 9 do
     for k:=0 to 9 do
              for l:=0 to 9 do
              begin s:=i+j+k+l;
                    x:=((10*i+j)*10+k)*10+l;
                    if y<s/x then begin y:=s/x; a:=x; end;
                  end;.
```

Задача 2. Найдите все натуральные трехзначные числа, каждое из которых обладает двумя следующими свойствами:

1) первая цифра в три раза меньше последней его цифры;

2) сумма самого числа с числом, получающимся из него перестановкой второй и

третьей цифр, делится на 8 без остатка.

Решение. Простое решение можно получить по аналогии с предыдущим:

```
for i:=1 to 9 do {цифра сотен}
  for j:=0 to 9 do {цифра десятков}
        for k:=0 to 9 do {цифра единиц}
if (i=3*k) and ((100*i+10*j+k+100*i+10*k+j) mod 8=0)
then write(i,j,k,' ');.
```

Посмотрим, каким образом можно уменьшить перебор. Если цифра сотен в три раза меньше цифры единиц, то k=3*i. Поскольку k – цифра, то k=3*i<=9 или i<=3. Рассмотрим второе условие, приведем в нем подобные члены: (100*i+10*j+3*i+100*i+10*3*i+j) mod 8=0 получим (200*i+33*i+11*j) mod 8=0. Поскольку 200*i делится на 8, то его можно отбросить. Таким образом, нужно проверять условие (33*i+11*j) mod 8=0 или 11*(3*i+j) mod 8=0. Получаем следующий фрагмент программы:

```
for i:=1 to 3 do
  for j:=0 to 9 do
        if (3*i+j) mod 8=0
        then write(i,j,3*i);.
```

Задача 3. Найдите все пятизначные числа вида 5m27n (m и n – цифры), которые делятся на 15.

Решение. Легко организовать полный перебор всех возможных вариантов.

```
for n:=0 to 9 do
  for m:=0 to 9 do
        if (50000+n*1000+270+m) mod 15=0
        then write(5,n*1000+270+m,' ');.
```

Уменьшим перебор. Число делится на 15, если оно делится на 5 и 3 (15=5·3). Число делится на 5, если оно оканчивается на 0 или 5. Следовательно, значениями m могут быть только цифры 0 или 5. Число делится на 3, если сумма его цифр делится на 3. Три цифры числа известны, их сумма равна 5+2+7=14. Если m=0, то сумма равна 14 и нужно подобрать такое n, чтобы сумма была кратна 3. Отсюда: значение n изменяется от 1 до 9 с шагом 3. Если m=5, то сумма четырех цифр равна 19, следовательно, n изменяется от 2 до 9 с шагом 3.

```
n:=1;
while n<=9 do
begin write(5,n,270,' ');
   n:=n+3;
end;
writeln;
n:=2;
while n<=9 do
begin write(5,n,275,' ');
   n:=n+3;
end;.
```

Задача 4. Последовательность перестановок на множестве {1,2,...,n} представлена в лексикографическом порядке, если она записана в порядке возрастания получающихся чисел. Например, лексикографическая последовательность перестановок трех элементов имеет вид 123,132,213,231,312,321. Сформировать все k-элементные перестановки множества {1,2,3,...,n} в лексикографическом порядке.

Решение. Для хранения перестановок будем использовать массив.

```
const nn=100;
```

```
type mas=array[1..nn] of integer;
var       a:mas;
          i,j,k,n,p:integer;
begin
  write('Введите n и k ');readln(n,k);
  for i:=1 to k do a[i]:=i;
  p:=k;
  while p>=1 do
  begin
          for j:=1 to k do write(a[j],' ');writeln;
          if a[k]=n then p:=p-1 else p:=k;
          if p>=1
          then for i:=k downto p do a[i]:=a[p]+i-p+1;
  end;
end.
```

Задача 5. Любую целочисленную сумму денег большую 7 руб. можно выплатить без сдачи купюрами по 3 руб. и 5 руб. По заданному целому положительному n определить пару целых неотрицательных чисел a и b, таких, что n=3*a+5*b.

Решение. Для решения задачи можно использовать ветвящиеся алгоритмы. Заметим, что если выплачивать деньги трешками, то в остатке могут получиться 0, 1 или 2. Если в остатке получился 0, то вся сумма выдается купюрами по 3 руб. (a=n div 3, b=0). Если в остатке получился 1 руб., нужно забрать обратно три купюры по 3 руб. и выдать две купюры по 5 руб. (3*3+1=10 div 5 =2, a=n div 3-3, b=2). Это решение можно применять, начиная с n=10. Если в остатке получилось 2 руб., то нужно забрать одну купюру 3 руб. и выдать одну купюру 5 руб. (3+2=5 div 5 =1, a=n div 3-1, b=1). Построим таблицу решений:

n mod 3=0	1		
n mod 3=1		1	
n mod 3=2			1
a=n div 3, b=0	x		
a=n div 3-3, b=2		x	
a=n div 3-1, b=1			x

От нее перейдем к программе:

```
ost:=n mod 3;
chas:=n div 3;
if ost=0
then write('3*',chas,'+5*0=',n)
else      if ost=1
          then write('3*',chas-3,'+5*2=',n)
          else write('3*',chas-1,'+5*1=',n);.
```

Возможен вариант решения, когда деньги выдаются купюрами по 5 руб. В этом случае возможны остатки 0, 1, 2, 3 и 4 рубля. Если в остатке получается 0, то (a=0, b=n div 5) все деньги можно выдать купюрами по 5 руб. Если в остатке получается 1, то можно забрать одну купюру 5 руб. и выдать две купюры по 3 руб. (5+1=6 div 3=2, a=2, b=n div 5-1). Если в остатке получается 2, то можно забрать две купюры по 5 руб. и выдать четыре купюры по 3 руб. (2*5+2=12 div 3 =4, a=4, b=n div 5-2). Если в остатке получается 3, то эту тройку можно выдать одной трешкой (a=1, b=n div 5). Если в остатке получается 4, то можно забрать одну купюру 5 руб. и выдать три купюры по 3 руб. (5+4=9 div 3=3, a=3, b=n div 5-1). Построим таблицу решений:

n mod 5=0	1				
n mod 5=1		1			
n mod 5=2			1		
n mod 5=3				1	
n mod 5=4					1
a=0, b=n din 5	x				
a=2, b=n div 5-1		x			
a=4, b=n div 5-2			x		
a=1, b=n div 5				x	
a=3, b=n div 5-1					x

От нее перейдем к программе:

```
ost:=n mod 5;
chas:=n div 5;
if ost=0
then write('3*0+5*',chas,'=',n)
else if ost=1
        then write('3*2+5*',chas,'=',n)
        else    if ost=2
                then write('3*4+5*',chas,'=',n)
                else if ost=3
                then write('3*1+5*',chas,'=',n)
                else write('3*3+5*',chas,'=',n);.
```

Получим с помощью перебора все решения этой задачи. Значение количества выданных купюр по 3 руб. не может превысить n div 3, а значение количества выданных купюр по 5 руб не может превысить n div 5. Исходя из этого, организуем циклы перебора:

```
repeat
  write('Введите значение n>7 ');
  readln(n);
until n>7;
for a:=0 to n div 3 do
  for b:=0 to n div 5 do
        if n=3*a+5*b
        then writeln('3*',a,'+5*',b,'=',n);.
```

Для n=128 получаем следующие решения, причем только два из них могли быть получены ранее с помощью построенных ветвящихся алгоритмов:

```
3* 1+5*25=128; {решение получено ветвящимся алгоритмом}
3* 6+5*22=128;
3*11+5*19=128;
3*16+5*16=128;
3*21+5*13=128;
3*26+5*10=128;
3*31+5* 7=128;
3*36+5* 4=128;
3*41+5* 1=128. {решение получено ветвящимся алгоритмом}
```

Задача 6. Требуется расставить n ферзей на шахматной доске размером n × n так, чтобы они не угрожали друг другу.

Решение. Для решения задачи организуем перебор с возвратом. Будем исходить из следующего: в каждом столбце может стоять только один ферзь, поэтому представим шахматную доску одномерным массивом, в котором номер элемента будет соответствовать столбцу, а содержимое – строке, в которой стоит ферзь. Два ферзя нельзя

поставить в одну строку, поэтому массив должен содержать сведения о перестановке элементов от 1 до n. Условие того, что два ферзя не находятся в одной строке, запишем в виде неравенства всех элементов массива друг другу. Два ферзя не могут находиться на одной диагонали. Это условие запишем в виде неравенства модулей разности содержимого любых двух элементов массива и модуля разности их индексов.

В программе последовательно перебираем клетки шахматной доски, и пытаемся поставить ферзя на очередную клетку. Если этого не удается сделать, то переходим к следующей клетке. Если клетки столбца закончились, а нам не удалось поставить ферзя, то возвращаемся к предыдущему столбцу, и продолжаем перебор его клеток. Перебор выполняется до тех пор, пока не будет получена нужная расстановка или закончились все клетки.

Представленная ниже программа решает задачу, используя перебор с возвратом.

```
uses crt;
const nn=10;
type mas=array[1..nn]of byte;
var a:mas; {представление шахматной доски}
   n:byte;  {размер доски}
   i,j:byte;  {индексы}
          f:boolean;
      function proverka(i:byte):boolean;
      {функция проверяет можно ли поставить ферзя в строку a[i], столбец i}
var f1:boolean;j:byte;
begin j:=1;f1:=true;
     while (j<i) and f1 do
    if not((a[i]=a[j]) or (abs(a[i]-a[j])=abs(i-j)))
           then j:=j+1
           else f1:=false;
           proverka:=f1;
      end;
      begin textbackground(7);
           textcolor(1);
           clrscr;
           write('Введите n ');
           readln(n);
           for i:=1 to n do a[i]:=0;
           i:=0;
           f:=true;
           f1:=true;
           while f and (i<=n)  do
           begin if f1
                 {делаем ход по столбцу}
          then begin i:=i+1;a[i]:=1;
                f1:=proverka(i);
              end
          else {возврат}
               begin while (a[i]=n) and (i>=1) do i:=i-1;
                 if i>=1 then a[i]:=a[i]+1;
                    if i=0 then f:=false;
                        f1:=proverka(i);
               end;
           end;
           if f
```

```
        then begin writeln('Решение');
                for i:=1 to n do write(a[i],' ');
             end
        else write('Решений нет');
    readln;
    end.
```

9 Одномерные массивы

Задача 1. Вычислить сумму нескольких произвольных элементов массива, индексы которых вводятся с клавиатуры. Признаком окончания ввода является нуль.

В этом и следующих примерах будем считать, что описан одномерный массив с нижней границей индекса 1 и верхней – nn. В массиве используются только n элементов.

```
Const nn=50;
type mas=array [1..nn] of real;
var     a:mas; {исходный массив}
  n:integer; { количество используемых элементов <=nn}
  i,j,k:integer; { индексы массива }
```

Решение. Пусть n=5 и массив a содержит следующие значения: a[1]=-3,a[2]=17,a[3]=4,a[4]=-5,a[5]=81. Последовательность ввода имеет вид: 3,1,10,3,5,0. В этом случае искомая сумма вычисляется в следующем порядке:

1) s:=0;
2) s:=s+a[3]=0+4=4;
3) s:=s+a[1]=4+(-3)=1;
4) сообщение пользователю: «В массиве нет элемента с индексом 10»;
5) s:=s+a[3]=1+4=5;
6) s:=s+a[5]=5+81=86;
7) 0 – конец ввода.

В результате получена сумма s=86.

Организуем цикл ввода элементов до нуля. В теле цикла будем проверять допустимость индекса, и находить сумму или выдавать сообщение о неправильном индексе. Получим фрагмент алгоритма на языке Паскаль:

```
{Ввод массива}
write('Введите используемое количество элементов ');readln(n);
write('Введите ',n,' элементов массива ');
for i:=1 to n do read(a[i]);
{Вычисление суммы}
s:=0;
write('Введите индекс ');read(i);
while i<>o do
begin if (1<=i) and (i<=n)
   then s:=s+a[i]
   else writeln('В массиве нет элемента с индексом ',i);
write('Введите индекс ');read(i);
end;
{Вывод ответа} writeln('s=',s);.
```

Задача 2. Для каждого элемента массива просмотреть все последующие элементы. Здесь возможно несколько вариантов.

Вариант 1. Массив просматриваем по одному элементу слева направо. Подмассив просматриваем по одному элементу тоже слева направо. Обозначим i – индекс массива, а j – индекс подмассива, тогда схема примет вид:

```
for i:=1 to n-1 do {у последнего элемента нет последующего}
for j:=i+1 to n do {обработка a[j]}
```

Вариант 2. Подмассив просматриваем справа налево.

```
for i:=1 to n-1 do
begin
  j:=n;
  while j>i do
  begin
          {обработка a[j]}
          j:=j-1;
  end;
end;.
```

Могут быть использованы и схемы перебора парами, соседями и т.д.

Задача 3. Для каждого элемента просмотреть все предшествующие ему элементы. Пусть i – индекс массива, а j – индекс подмассива.

Случай 1. Основной массив просматривается слева направо, подмассив просматривается так же.

```
for i:=2 to n do {у первого элемента нет предыдущего}
          for j:=1 to i-1 do
                  {обработка a[j]}
```

Случай 2. Основной массив просматривается слева направо, а подмассив – справа налево.

```
for i:=2 to n do
begin
  j:=i-1;
  while j>0 do
  begin
          {обработка a[j]}
          j:=j-1;
  end;
end;.
```

Случай 3. Основной массив просматривается справа налево, а подмассив – слева направо.

```
i:=n;
while i>1 do
begin
  for j:=1 to i-1 do {обработка a[j]}
  i:=i-1;
end;.
```

Случай 4. Основной массив просматривается справа налево, а подмассив – справа налево.

```
i:=1;
while i>1 do
begin
  j:=i-1;
  while j>0 do
  begin
          {обработка a[j]}
          j:=j-1;
  end;
  i:=i-1;
end;.
```

Задача 4. Найти сумму (произведение) элементов одномерного массива.

Решение. Каждый элемент массива прибавляется к сумме s:=s+a[i] или умножается на произведение предыдущих элементов p:=p*a[i]. Поскольку требуется перебрать все элементы массива, выберем схему перебора элементов по одному, двигаясь с начала или с конца массива.

```
{сумма элементов массива}  {произведение элементов массива}
 s:=0;                      p:=1;
 i:=1;                      i:=n;
 while i<=n do              while i>0 do
 begin                      begin
        s:=s+a[i];                  p:=p*a[i];
        i:=i+1;                     i:=i-1;
 end;.                      end;.
```

Задача 5. Задан одномерный массив. Нужно переставить элементы в обратном порядке. Другой массив использовать не разрешается. Для массива a={1,2,3,4,5,6,7,8,9} получим результат a={9,8,7,6,5,4,3,2,1}.

Решение. При преобразовании меняются местами два элемента: первый и последний, второй и предпоследний и так далее. В случае нечетного количества элементов имеется центральный элемент, который должен остаться на месте. Меняются элементы, симметричные относительно центрального. Индексы каждой пары таких элементов определим по формулам: i и n+1-i. Всего в массиве должно произойти n div 2 обменов. Последняя фраза легко переводится на язык Паскаль: организовать арифметический цикл со счетчиком до n div 2, в теле которого меняются местами элементы a[i] и a[n+1-i].

```
for i:=1 to n div 2 do
begin   r:=a[i];
        a[i]:=a[n+1-i];
        a[n+1-i]:=r;
end;.
```

Второе решение. Введем обозначения для пары меняемых элементов: i – элемент, входящий в пару и расположенный в начале массива, j – соответствующий элемент пары, расположенный в конце массива. При таком соглашении i должно быть меньше j. Если i=j, то они указывают на один и тот же элемент, который можно не менять. Если i>j, то пар больше нет. Таким образом, имеем итерационный цикл, который выполняется до тех пор, пока есть пары элементов. В теле цикла a[i] и a[j] меняются местами.

```
i:=1; j:=n;
while i<j do
begin r:=a[i];
   a[i]:=a[j];
   a[j]:=r;
   i:=i+1; j:=j-1;
end;.
```

Задача 6. Из массива a получить такой массив b, в котором элементы расположены в таком порядке: $a_n a_1 a_{n-1} a_2 a_{n-2} a_3$....

Решение. Элементы в массиве b чередуются следующим образом. На первом месте стоит последний элемент из массива a, на втором – первый, на третьем – предпоследний, т.е. на нечетных местах располагаются элементы с конца массива a в порядке убывания индексов. На четных местах в массиве b располагаются элементы с начала массива a в порядке возрастания индексов. Выбираем схему перебора по элементам массива b (цикл по выходному массиву). Элементы массива a перебираем по схеме «от обоих концов к середине». Запишем это решение на языке Паскаль:

```
i:=1; {индексы массива a}
j:=n;
```

```
for k:=1 to n do
  if odd(k)
  then begin b[k]:=a[j]; j:=j-1; end
  else begin b[k]:=a[i]; i:=i+1; end;.
```

Задача 7. В заданном одномерном массиве все нулевые элементы переписать в конец массива. Для массива a={1,0,2,3,0,4,5,0,0,0,6} получим:

a={1,2,3,4,5,6,0,0,0,0,0}.

Решение. Воспользуемся услугами вспомогательного массива, в который вначале перепишем ненулевые элементы, оставшуюся часть вспомогательного массива заполним нулями. Затем перепишем содержимое вспомогательного массива в исходный массив. Соответствующий вариант решения приведен ниже. Здесь a – исходный массив, i – индекс массива a, b – вспомогательный массив, j – его индекс. Размерность обоих массивов – n.

```
j:=1; {указывает номер первой свободной позиции в массиве b}
  {переписываем все ненулевые элементы из b в a}
for i:=1 to n do
   if a[i]<>0 then begin b[j]:=a[i]; j:=j+1; end;
  {заполняем остаток массива b нулями}
  for i:=j to n do b[i]:=0;
  {переписываем массив b в a}
  a:=b;.
```

Главным недостатком полученного решения является использование вспомогательного массива – нерациональное использование оперативной памяти компьютера. Чтобы обойтись без него, запишем ненулевые элементы в начало исходного массива. Место записи элемента указывает переменная j, которая передвигается по массиву только после записи ненулевого элемента. При найденном нуле значение переменной j не изменяется. Переменная i будет указывать на проверяемый элемент массива, т.е. для одного массива используются два индекса: индекс i перебирает элементы массива a как входного (исходного), а индекс j – как выходного. Результат получается сразу в массиве a. Соответствующий вариант программы приведен ниже:

```
j:=1;
for i:=1 to n do
   if a[i]<>0 then begin a[j]:=a[i]; j:=j+1; end;
for i:=j to n do a[i]:=0;.
```

Задача 8. Провести поиск среди элементов одномерного массива элемента, равного заданному «аргументу поиска» a. Существует несколько вариантов решения задачи.

Вариант 1 (линейный поиск). Если нет дополнительной информации о разыскиваемых данных, то остается последовательный просмотр массива с увеличением шаг за шагом той его части, где искомого элемента не обнаружено. Такой метод называется линейным поиском. Исходными данными для решения задачи является массив чисел. В результате поиска получаем либо номер элемента, совпадающего с A, либо ответ: «Такого элемента нет». Существует 2 варианта окончания поиска: 1) закончились элементы массива, а нужный элемент не найден; 2) найден нужный элемент.

Первый вариант окончания поиска определяется по истинности условия: i>n, где i – текущий элемент массива, n – количество элементов в массиве.

Второй вариант окончания поиска определяется по истинности логической переменной f, где true соответствует найдено, a false – не найдено.

Таким образом, условие окончания поиска записывается в виде (i>n) or f, что соответствует словестному описанию: «Просмотрены все элементы или не найдено». Просмотр элементов массива в цикле будет выполняться в случае ложности приведенного условия. Найдем его отрицание согласно закону де Моргана: отрицание дизъюнк-

ции равно конъюнкции отрицаний.

not((i>n) or f)=not(i>n) and not f=(i<=n) and not f.

Программа на языке Паскаль:

```
program POISK;
{линейный поиск элемента в одномерном массиве}
const nn=12; {константа задает максимальный размер массива}
type mas1=array[1..nn] of integer; {тип массива}
var       b:mas1; {исходный массив}
  n:integer; {размер массива}
  a:integer; {число для поиска}
  i:integer; {индекс массива}
  f:boolean; {истина, если искомый элемент найден}
begin
 write('Введите размер массива от 1 до ',nn);
  repeat {вводим n до тех пор,}
          readln(n); {пока оно не попадет}
  until (1<=n) and (n<=nn); {на отрезок от 1 до nn}
  {Вводим элементы массива}
  for i:=1 to n do
  begin   write('Введите элемент массива B[',i:2,']=');
          readln(b[i])
  end;
  write('Введите искомый элемент ');
  readln(a);
  i:=1; {начинаем просмотр с первого элемента}
  f:=false; {пока не найдено}
  while (i<=n) and not f do
  {просматриваем массив, пока есть элементы и не найден искомый}
          if b[i]=a {если совпали,}
          then f:=true {то нашли,}
          else i:=i+1; {иначе перейти к следующему элементу}
  if f {вывод результатов поиска}
  then write('Номер найденного элемента ',i)
  else write('Не нашли');
end.
```

Тестирование.

Тест 1. n=8, B[1]=2, B[2]=8, B[3]=3, B[4]=1, B[5]=9, B[6]=2, B[7]=2, B[8]=2, a=5. Результат поиска: «Не нашли».

Тест 2. n=7, B[1]=2, B[2]=8, B[3]=3, B[4]=1, B[5]=9, B[6]=8, B[7]=2, a=8. Результат поиска: «Номер найденного элемента 2».

Вариант 2 (линейный поиск с барьером). Применяется широко распространенный прием фиктивного элемента, или барьера, расположенного в конце массива. Это позволяет упростить условие окончания цикла, так как заранее ясно, что хотя бы один элемент, равный A, в массиве есть.

```
program POISK1;
{поиск с барьером в одномерном массиве}
const nn=12; {константа задает максимальный размер массива}
type mas1=array[1..nn] of integer; {тип массива}
var       b:mas1; {исходный массив}
  n:integer; { размер массива }
  a:integer; { число для поиска }
```

```
  i:integer; { индекс массива }
begin
 write('Введите размер массива от 1 до ',nn-1);
  repeat {вводим n до тех пор,}
          readln(n); { пока оно не попадет }
  until (1<=n) and (n<=nn); {на отрезок от 1 до nn-1}
  {Вводим элементы массива}
  for i:=1 to n do
  begin
          write('Введите элемент массива B[',i:2,']=');
                readln(b[i]);
  end;
  write('Введите искомый элемент ');
  readln(a);
  b[n+1]:=a; {добавили барьер, равный a, в конец массива}
  i:=1; {начинаем просмотр с первого элемента}
  while b[i]<>a do i:=i+1;
  {просматриваем массив, пока не найдем}
  if i<=n {вывод результатов поиска}
  then write('Номер найденного элемента ',i)
  else write('Не нашли');
end.
```

Тестирование.

Тест 1. n=8, B[1]=2, B[2]=8, B[3]=3, B[4]=1, B[5]=9, B[6]=2, B[7]=2, B[8]=2, B[9]=5, a=5. Результат поиска: «Не нашли».

Тест 2. n=7, B[1]=2, B[2]=8, B[3]=3, B[4]=1, B[5]=9, B[6]=8, B[7]=2, B[8]=8, a=8. Результат поиска: «Номер найденного элемента 2».

Вариант 3 (поиск методом деления пополам, или двоичный поиск). Предварительно упорядочим (отсортируем) массив: делим его пополам и для сравнения выбираем средний элемент. Если он совпадает с искомым, то поиск заканчивается. Если средний элемент меньше искомого, то все элементы, расположенные левее, также будут меньше искомого. Следовательно, их можно исключить из дальнейшего поиска, оставив только правую часть массива. Если средний элемент больше искомого, то отбрасывается правая часть. Процедура деления массива пополам и отбор части массива для дальнейшего поиска продолжается до тех пор, пока искомый элемент не будет найден или длина части массива не станет равной нулю. Каждый раз величина той части массива, где ведется поиск, уменьшается вдвое. Поэтому максимальное число требующихся сравнений равно $]\log_2 N[$, где N – количество элементов в массиве.

Рассмотрим пример двоичного поиска.

Искомый элемент a=7. Исходный массив n=16.

Первый шаг поиска:

номер элемента: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16;

массив B={1,3,5,6,7,10,12,14,15,16,20,21,23,25,26,30};

начальная граница поиска L=1, конечная граница поиска R=16;

середина части поиска i=(L+R) div 2 = 8;

поскольку B[8]=14>a=7, то отбрасываем правую часть: L=1, R=i-1=7.

Второй шаг поиска:

номер элемента: 1 2 3 4 5 6 7;

массив B={1,3,5,6,7,10,12};

начальная граница поиска L=1, конечная граница поиска R=7;

середина части поиска i=(L+R) div 2 = 4;

поскольку B[4]=6<a=7, то отбрасываем левую часть: L=i+1=5, R=7.

Третий шаг поиска:

номер элемента: 5 6 7;
массив B={7,10,12};
начальная граница поиска L=5, конечная граница поиска R=7;
середина части поиска i=(L+R) div 2 = 6;
поскольку B[6]=10>a=7, то отбрасываем правую часть: L=5, R=i-1=5.

Четвертый шаг поиска:

номер элемента: 5;
массив B={7};
начальная граница поиска L=5, конечная граница поиска R=5;
середина части поиска i=(L+R) div 2 = 5;
поскольку B[5]=7=a=7, то искомый элемент найден в позиции 5.

Фрагмент программы на языке Паскаль, реализующий двоичный поиск, приведен ниже:

```
l:=1; r:=n;
f:=false; {не найдено}
while (l<=r) and not f do
begin i:=(l+r) div 2;
  if b[i]=a then f:=true {нашли}
  else if b[i]<a then l:=i+1 else r:=i-1;
end.
```

Тестирование.

Тест 1: a=7, n=15, b={1,3,5,7,9,10,12,14,16,18,21,23,25,27,29}.
Ответ: искомый элемент найден в позиции 4.
Тест 2: a=6, n=15, b={1,3,5,7,9,10,12,14,16,18,21,23,25,27,29}.
Ответ: искомый элемент не найден.

10 Двухмерные массивы

Рассмотрим двухмерный массив размерностью n × n для n=6. Здесь количества строк и столбцов совпадают, поэтому такие массивы называются квадратными.

a_{11} a_{12} a_{13} a_{14} a_{15} a_{16}
a_{21} a_{22} a_{23} a_{24} a_{25} a_{26}
a_{31} a_{32} a_{33} a_{34} a_{35} a_{36}
a_{41} a_{42} a_{43} a_{44} a_{45} a_{46}
a_{51} a_{52} a_{53} a_{54} a_{55} a_{56}
a_{61} a_{62} a_{63} a_{64} a_{65} a_{66}

Главной диагональю двухмерного массива называют прямую, мысленно проведенную из левого верхнего угла в правый нижний, элементы которой, в нашем случае, имеют следующий вид: a_{11},a_{22},a_{33},a_{44},a_{55},a_{66}. Если индекс строк обозначить i, а индекс столбцов – j, то в общем случае факт принадлежности элемента главной диагонали записывается так: i=j, или так: i-j=0. Для элементов, расположенных ниже главной диагонали, справедливо соотношение индексов i>j. Для элементов, расположенных выше главной диагонали, справедливо соотношение индексов i<j.

Индексы элементов, расположенных на линиях, параллельных главной диагонали, удовлетворяют соотношениям |i-j|=k, где k – номер линии по отношению к главной диагонали. Например, для индексов элементов a_{31},a_{42},a_{53},a_{64}, расположенных на 2 линии ниже главной диагонали, справедливо соотношение i-j=2, а для индексов элементов a_{13},a_{24},a_{35},a_{46}, расположенных на две линии выше главной диагонали, справедливо соотношение i-j=-2.

Линию, соединяющую правый верхний угол массива с левым нижним, называют побочной диагональю. Для индексов элементов побочной диагонали: a_{16},a_{25},a_{34},a_{43},a_{52},a_{61}, справедливо соотношение i+j=n+1, где n – количество строк и

столбцов массива (в примере n=6). Для индексов элементов, расположенных выше побочной диагонали, справедливо соотношение i+j<n+1, а для индексов элементов, расположенных ниже побочной диагонали, справедливо соотношение i+j>n+1.

Индексы элементов, расположенных на линиях параллельных побочной диагонали, удовлетворяют соотношению 2<=i+j<=2n. Например, для индексов элементов: a_{14},a_{23},a_{32},a_{41}, расположенных на 2 линии выше побочной диагонали, справедливо соотношение i+j=5, а для индексов элементов a_{36},a_{45},a_{54},a_{63}, расположенных на 2 линии ниже побочной диагонали, справедливо соотношение i+j=9.

Задача 1. Сформировать двухмерный массив n × n указанного вида для произвольного n. Для n=4 формируемый массив имеет вид

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

Решение. Каждая строка начинается с элемента, значение которого совпадает с номером строки. Затем оно уменьшается с ростом номера столбца. Все элементы, лежащие выше главной диагонали, равны нулю. Легко догадаться, что в теле циклов перебора повторяется оператор a[i,j]:=i-j+1. Для перебора строк и столбцов используем вложенные циклы.

```
for i:=1 to n do
  for j:=1 to n do
        if i<=j
        then a[i,j]:=i-j+1
        else a[i,j]:=0;.
```

Задача 2. Сформировать двухмерный массив n × n указанного вида для произвольного n. Для n=4 формируемый массив имеет вид

0 1 0 2
3 0 4 0
0 5 0 6
7 0 8 0.

Решение. Ненулевое значение имеют элементы, сумма индексов которых нечетна. Для формирования значения можно использовать дополнительную переменную. Для перебора элементов будем использовать два вложенных цикла. Приведем в качестве примера схему перебора от последних элементов к первым.

```
k:=2*n; {значение последнего элемента}
for i:=n downto 1 do
  for j:=n downto 1 do
        if (i+j) mod 2 =1
        then begin a[i,j]:=k; k:=k-1; end
        else a[i,j]:=0;.
```

Задача 3. Найти индексы максимального элемента двухмерного массива.

Решение. Решение этой задачи совпадает с решением задачи для одномерного массива. Отличие заключается в необходимости для двухмерного массива вложенных циклов перебора. Фрагмент программы приведен ниже:

```
imax:=1; jmax:=1; {предполагаем максимальным первый элемент}
for i:=1 to n do
  for j:=1 to n do
        if a[imax,jmax]<a[i,j]
        then begin imax:=i; jmax:=j; end;.
```

Задача 4. Переставить строки и столбцы двухмерного массива так, чтобы его максимальный элемент оказался в левом верхнем углу.

Решение.

1) поиск индексов максимального элемента (пример 8.1.3);

2) обмен местами первой строки и строки imax;

3) обмен местами столбца 1 и столбца jmax. Шаги 2 и 3 можно поменять местами без изменения результата.

Фрагмент программы приведен ниже:

```
{поиск индексов максимального элемента}
imax:=1; jmax:=1; {предполагаем максимальным первый элемент}
for i:=1 to n do
  for j:=1 to n do
          if a[imax,jmax]<a[i,j]
          then begin imax:=i; jmax:=j; end.
{обмен местами первой и imax строк}
for i:=1 to n do
begin r:=a[1,i]; a[1,i]:=a[imax,i]; a[imax,i]:=r; end;
{обмен местами первого и jmax столбцов}
for i:=1 to n do
begin r:=a[i,1]; a[i,1]:=a[i,jmax]; a[i,jmax]:=r; end;.
```

Задача 5. Задан двухмерный массив n × n. Поменять местами строку с номером k и столбец с номером p этого массива.

Решение 1. Задача просто решается при k=p. Например, при n=3, k=p=2 и исходном массиве:

1 2 3
4 5 6
7 8 9.

Для того чтобы получить ответ меняем элементы 2 и 4, 8 и 6. Элемент 5 остается на своем месте. Получаем ответ:

1 4 3
2 5 8
7 6 9.

Однако при k≠p возникают трудности с элементом, стоящим на пересечении строки k и столбца p. Если применить тот же алгоритм для n=3, k=2, p=3 к тому же исходному массиву, то меняются местами элементы 4 и 3, 5 и 6, 5 и 9, что приводит к неверному ответу: в строке появился элемент 9, которого не было в исходной строке:

1 2 4
3 6 9
7 8 9.

Для преодоления этой трудности применим такой прием:

1) переставить строку k на место строки p:

1 2 3
7 8 9
4 5 6.

2) поменять местами строку p и столбец p:

1 2 4
7 8 5
3 9 6.

3) переставить строки k и p:

1 2 4
3 9 6
7 8 5.

Соответствующий фрагмент на языке Паскаль приводится ниже:

```
{обмен местами строк с номерами k и p}
for i:=1 to n do
begin r:=a[k,i]; a[k,i]:=a[p,i]; a[p,i]:=r; end;
{обмен местами строки и столбца с номером p}
for i:=1 to n do
begin r:=a[p,i]; a[p,i]:=a[i,p]; a[i,p]:=r; end;
{обмен местами строк с номерами k и p}
for i:=1 to n do
begin r:=a[k,i]; a[k,i]:=a[p,i]; a[p,i]:=r; end;.
```

Решение 2. Недостатком предыдущего решения является большое количество циклов. Решим задачу по-другому, сократив количество используемых циклов. Переставляемых элементов несколько в строке k и столбце p, поэтому переставляемые элементы строки k обозначим переменной i, а столбца p – j. Однако элемент, стоящий на пересечении строки k и столбца p, переставляться не должен. Поэтому переставляемые элементы k-й строки имеют индексы 1<=i<=n, кроме i=p; переставляемые элементы p-го столбца имеют индексы 1<=j<=n, кроме j=k. В соответствии с этим можно организовать цикл перебора, пока есть переставляемые элементы. В теле цикла элементы меняются местами или пропускаются, если они не являются переставляемыми. Фрагмент решения, реализующий описанную идею, приведен ниже.

```
i:=1; j:=1;
while (i<=n) and (j<=n) do
  if (i<>p) and (j<>k)
  then    begin r:=a[j,p]; a[j,p]:=a[k,i]; a[k,i]:=r;
                  i:=i+1; j:=j+1;
          end
  else    begin if i=p then i:=i+1;
                  if j=k then j:=j+1;
          end;.
```

Задача 6. Найдите максимальную сумму среди сумм тех столбцов целочисленного двухмерного массива n × n, диагональный элемент которых положительный.

Решение. Перебираем столбцы (i). Для каждого столбца (j), проверяем, положителен ли его диагональный элемент (i=j). Если диагональный элемент положителен, то находим сумму элементов этого столбца и проверяем ее на максимум. При поиске максимальной суммы возникает проблема присваивания начального значения, которое заранее неизвестно, так как заранее неизвестен столбец с положительным элементом на диагонали. Поэтому приходится заводить указатель, указывающий, в первый ли раз вычислено нужное значение.

```
p:=0; {нет начального значения для поиска максимума}
for i:=1 to n do {перебор столбцов}
if a[i,i]>0
then      begin s:=0;
           for j:=1 to n do s:=s+a[j,i];
           if p=0
           then begin max:=s; p:=1; end { установлено начальное значение для максимума }
           else if max<s then max:=s;
  end;.
```

Задача 7. Отсортировать элементы двухмерного массива по элементам второй строки.

Исходный массив Результат

1 2 3 4 5 5 2 4 3 1
9 3 7 3 1 1 3 3 7 9
6 7 8 9 1 1 7 9 8 6.

Решение. От сортировки одномерного массива этот случай отличается только тем, что переставлять нужно не два сравниваемых элемента, а два столбца:

```
for i:=1 to n-1 do
for j:=i+1 to n do
  if a[2,i]>a[2,j]
  then for k:=1 to n do
         begin r:=a[k,i]; a[k,i]:=a[k,j]; a[k,j]:=r;
  end;.
```

Задача 8. Найти произведение матрицы n × k на матрицу k × m.

Решение. Матрицы представляем в виде двухмерных массивов. В результате получается матрица – двухмерный массив n × m, где элемент $c_{ij} = \sum_{p=1}^{k} a_{ip} \cdot b_{pj}$, т.е. каждый элемент ответа равен сумме произведений элементов i-й строки массива a на соответствующие элементы j-го столбца массива b.

```
for i:=1 to n do
for j:=1 to m do
begin c[i,j]:=0;
  for p:=1 to k do c[i,j]:=c[i,j]+a[i,k]*b[k,j];
end;
```

Таким образом, представлены методики решения по трем из девятнадцати выделенных классов задач:

8 Задачи, решаемые методом перебора.

9 Одномерные массивы.

10 Двухмерные массивы.

В следующей статье мы продолжим знакомство с методикой быстрого обучения программированию на основе изучения классов задач. Будут рассмотрены методики по следующим пяти из девятнадцати выделенных классов задач:

11 Данные типа string.

12 Процедуры и функции.

13 Рекурсия.

14 Тип данных множество.

15 Тип данных запись.

Литература

1. *Аляев Ю.А.* Алгоритмизация и языки программирования Pascal, C++, Visual Basic / Ю.А. Аляев, О.А. Козлов. М.: Финансы и статистика, 2002, 2004, 2007. 320 с.

2. *Аляев Ю.А.* Практикум по алгоритмизации и программированию на языке Паскаль / Ю.А. Аляев, В.П. Гладков, О.А. Козлов. М.: Финансы и статистика, 2004, 2007. 528 с.

3. *Аляев Ю.А.* Методика быстрого обучения программированию на основе изучения классов задач (1–3) // Образовательные ресурсы и технологии. 2015'1(9). С. 3–14. URL: http://www.muiv.ru/vestnik/pdf/pp/ot_2015_1_3-14.pdf.

4. *Аляев Ю.А.* Методика быстрого обучения программированию на основе изучения классов задач (4–5) // Образовательные ресурсы и технологии. 2015'2(10). С. 3–16. URL: http://www.muiv.ru/vestnik/pdf/pp/ot_2015_2_3-16.pdf.

5. *Аляев Ю.А.* Методика быстрого обучения программированию на основе изучения классов задач (6–7) // Образовательные ресурсы и технологии. 2015'3(11). С. 3–20. URL: http://www.muiv.ru/vestnik/pdf/pp/ot_2015_003_020.pdf.

Methods of the quick education to programming on base of the study of the classes of the problems (8-10)

***Yuri Alexandrovich Alyaev**, assistant professor, assistant professor of the pulpit of software of the computing machinery and automated systems, Perm military institute of internal troops of the MIA*

Is offered methods of the quick education to programming on base of the study of the classes of the problems, designed and using in practice in process of the education to programming student high school.

Keywords: algorithm, program, programming language Pascal, array

УДК 519.1(075.8)+510.6(075.8)

ПРАКТИЧЕСКАЯ ДИСКРЕТНАЯ МАТЕМАТИКА И МАТЕМАТИЧЕСКАЯ ЛОГИКА (практические занятия 1–3)

***Сергей Феофентович Тюрин**, профессор, профессор кафедры автоматики и телемеханики,*
E-mail: tyurinsergfeo@yandex.ru,
Пермский национальный исследовательский политехнический университет,
http://pstu.ru,
***Юрий Александрович Аляев**, доцент, доцент кафедры программного обеспечения вычислительной техники и автоматизированных систем,*
E-mail: alyr1@yandex.ru,
Пермский военный институт внутренних войск МВД России,
http://pvivv.ru

Предлагается методика решения задач на практических занятиях по дисциплине «Дискретная математика и математическая логика», разработанная и применяющаяся на практике в вузах Пермского края.

Ключевые слова: дискретная математика, математическая логика, множество, алгебра логики

«…Всё те же мы – в углах дизъюнкций,
В узлах завязанных конъюнкций,
И вновь – инверсии черта!..»
С.Ф. Тюрин

Предисловие

Издавая в 2006 г. учебник «Дискретная математика и математическая логика» [1], авторы планировали вслед за ним издать и задачник. Переосмыслив имеющийся материал в последующие годы, они пришли к выводу о необходимости подготовки не совсем учебника, но советчика и подсказчика. Кроме того, был накоплен новый, интересный материал. Акцент сделан на практику, поскольку известно, что именно умение решать задачи является мерилом математического знания.


С.Ф. Тюрин

В предлагаемой серии статей нашёл отражение опыт многолетнего преподавания авторами дисциплин «Дискретная математика» и «Математическая логика и теория алгоритмов» в вузах Пермского края.