Федеральное агентство по образованию

Государственное образовательное учреждение высшего профессионального образования
«МАТИ» - Российский государственный технологический университет им. К.Э. Циолковского

Кафедра «Высшая математика»

Примеры решения задач с использованием массива

Методические указания к лабораторной работе по курсу «Информатика»

Составитель: Сидоров Б.Н.

Москва 2015

Методические указания предназначены для студентов, изучающих курс «Информатика» Рассмотрен ряд методов сортировки одномерного массива и их применение при решении задач. Даны задачи для самостоятельного решения.

1. ОПРЕДЕЛЕНИЯ

Массив-это структура,объединяющая упорядоченные данные одного и того же типа,при этом:

1.Все элементы массива имеют одно имя.

2.Число элементов массива определяется при его описании и не может изменяться в программе.

3.К каждому элементу массива имеется прямой доступ, для чего указывается имя массива и после него в квадратных скобках порядковый номер этого элемента в массиве,называемый индексом (для чего и необходима упорядоченность).Количество индексов определяет размерность массива.

Описание массива осуществляется с использованием служебного слова array.

var имя : array [тип индекса] of тип компонент

где:

имя - произвольное имя,которое дается описываемому массиву в программе,

тип индекса : определяет границы изменения,индекс должен быть перечисляемым и ограниченным. Прежде всего это ограниченный целый тип

пр: 1...100; -700...100,; 200...300

a : array [1...100] of тип компонент;

так же возможно использования типа char

пр: var s : array [shar] of тип компонент

тогда доступ к отдельному элементу массива

s ['d']:= ...

Тип компонента- это любой тип данных;

При работе с массивами необходимо иметь в виду:

1. Действие с элементами массива определены так же,как и с переменными его типа

a[1] : = a[2] + a[4]

a[1] : = sgr (a[1] + a[3])...

2. Индекс массива может задаваться как прямо

a[1], a[2]...

так и косвенно,через переменную

a[i]

так и косвенно,через выражение

a[i+3]

3. При обработке массива наиболее удобно использовать оператор цикла

Обычно это оператор for,изменяющий индекс массива от начального до конечного значения,тем самым обеспечивая последовательный доступ к каждому элементу массива,с которым,в результате,производятся необходимые действия. Т.о. работа с массивами сводится к алгоритму.

```
For i:= нач. to   кон. do
  begin
    Обработка элемента имя[i]
  end;
```

Массивы занимают большой объем памяти,поэтому их надо использовать в ограниченных случаях:

а) когда требуется повторная обработка некоторой структуры данных, например: при нахождении дисперсии (Si-Scp) требуется

дважды обработать последовательность Si:

- для вычисления Scp;
- для вычисления суммы;

б) когда все данные должны быть представлены в программе одновременно, например: сортировка таблиц

в) когда обрабатываются соответствующие математические величины: векторы,матрицы.

2. ВВОД МАССИВА

Элементы массива вводятся в цикле, предварительно выводим поясняющий текст

```
const n=10;
 var a:array[1..n] of integer;
    i    :integer;
 begin
 writeln('введите ' , n, ' элементов массива ');
 for i:=1 to n do
  begin
    writeln ('введите [',i,'] -й элемент массива');
   readln (a[i]);
   end;
     .............
```

далее идет решение задачи обработки массива и вывод результата

3. ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ МАССИВОВ

Рассмотрим основные типы задач и их вариации.

Первый тип задач — это задачи на вычисления суммы, количества элементов, удовлетворяющем какому либо условию.

Все эти задачи решаются по схожему алгоритму

Для суммы:

```
S:=0;
 FOR I: = 1 TO N DO
  IF( условие ) THEN S:=S+A[I];
```

Для произведения:

```
P:=0;
 FOR I: = 1 TO N DO
  IF( условие ) THEN P:=P*A[I];
```

Для количества:

```
K:=0;
 FOR I: = 1 TO N DO
   IF( условие ) THEN K:=K+1;
```

Пример.

Дано: A(N)

Определить среднее арифметическое положительных элементов массива.

РЕШЕНИЕ.

В задаче необходимо найти сумму S и число K положительных элементов массива (тогда среднее арифметическое SRARF=S/K), Для этого используется цикл FOR,позволяющий получить последовательно доступ к каждому элементу массива,который необходимо сравнить с нулем, и если он больше нуля,то добавить его в сумму S:=S=A[I] и увеличить счетчик числа элементов K:=K+1. (запись S:=S+A[i] означает,что к старому значению S добавлен элемент A[i] и это становится новым значением S)

Текст программы

```
const n=10;
var a: array [1...n] of real;
   s,SRARF : real;
   i,K: integer;
begin
   {ввод размерности массива A}
   WRITELN (' Введите размерность массива N<100');
   READLN (N);
   {ввод массива A}
   WRITELN ('Введите',N,'элементов массива');
   FOR I:=1 to N do
      begin
         WRITELN ('введите',I,'элемент массива');
         READLN (A[I]);
      end;
    {обработка}
  S:=0 ;K:=0;
  FOR I:=1 to N do
     IF (A(I)>0) then
                 begin
                    s:=s + A[I];
                    к:=к + 1
                   end;
  SRARF:= S/K ;
```

```
{вывод результата}
WRITELN ('SRARF = ',SRARF);
end.
```

Второй тип задач на циклический сдвиг элементов, здесь все просто поэтому перейдем сразу к примеру

а) влево

было : \0\7\4\5\2\9\8\
должно быть : \7\4\5\2\9\8\0\

Для решения задачи необходимо на место каждого элемента поставить последующий ,дополнительно необходимо позаботиться о сохранении первого элемента (переменная t),остальные элементы перед тем как на их место будут поставлены последующие, уже сохранены,так как передвинуты вперед.

```
const n=10;
 var a:array [1..n] of real;
     i : integer;
     t  : real;
begin
  .................
  t:=a[1];
  for i:=1 to n-1 do a[i]:=a[i+1];
  a[n]:=t;
 .................
end.
```

б) вправо

было : \0\7\4\5\2\9\8\
должно быть : \8\0\7\4\5\2\9\

Для решения этой задачи необходимо на место каждого элемента i поставить предыдущий: i-1 ,а для того чтобы i-элемент не затерся необходимо движение начинать с конца массива и дополнительно сохранить последний элемент в переменной t.

```
const n=10;
 var a: array [1...n] of real;
```

```
    i: integer;
    t: real;
 begin
    .....................
    t:= a[n];
    for i:= n dowto 2 do
          a[i]:= a[i-1];
    a[i]=t ;
    .....................
end.
```

Примеры задач, с использованием циклического сдвига

пример: Из массва A(N) удалить К-й элемент,K<N.

Решение.

Эта задача сводится к уже решенной задаче о циклическом сдвиге элементов массива,что очевидно,если ее сформулировать так: необходимо сдвинуть элементы массива влево,начиная с элемента К+1,а К-й элемент запоминать уже не нужно,он автоматически уничтожается (так как на его место встает элемент К+1).

```
const n=10;
 var a: array [1..n] of real;
 i,k: integer;
 begin
.............
 for i:=k to n-1 do
   a[i]:=a[i+1];
.............
end.
```

Пример

Дан массив A(N),элемент B и K<N.

Требуется за элементом A(K) вставить элемент B.

Решение.

Эта задача распадается на две:первая,уже решенная необходимо сдвинуть вправо все элементы массива A(N),начиная с элемента K+1.Вторая-на освободившееся место в A(N) с индексом K+1 поставить элемент B,эта задача решается одним оператором присваивания.

A(K+1):=B

```
const n=10;
  var a:array [1...n] of real;
  i,k: integer;
  в: real;
begin
  .....
  for i:=n+1 dowto k+2 do
     a[i]:= a[i-1];
  a[k+1]:=в;
  .....
end .
```

Замечание: при вводе переменной n,необходимо вместо условия n<100 задать условие n<99.

Третий тип задач — это задачи на вычисление минимального, максимального значения элементов массива, рассмотрим на примере вычисления минимума

Дано: A(N). Определить минимальный элемент массива.

Решение. Для решения этой задачи необходимо ввести переменную Amin, смысл которой в том,что она хранит минимальное значение из просмотренных элементов массива: первоначально AMIN:=A(1) тогда,при просмотре массива (оператором FOR),будет сравниваться очередной элемент A(I) с минимальным из уже просмотренных Amin (а не с предыдущим !). Если A(I)<Amin ,то он и становится новым нимальным;когда будет просмотрен весь массив,то Amin будет содержать минимальный элемент массива

Текст программы:

```
const n=10;
  var a:array[1...n] of real;
      AMIN :real;
      i,к: integer;
    begin
    .....................
      AMIN:=A[1];
      K:= 1;

      for i:= 2 to n do
        begin
          if A[i]<AMIN then
            begin
              AMIN:=A[i];
              K:=I;
            end
        end;
      writeln('мин.элемент=',AMIN,'его порядковый номер',K);
     ...
    end.
```

Примеры задач, с использованием писка минимального элемента.

Пример 1.

Поиск количества минимальных элементов.
Задачу можно решить в два прохода, первый — поиск минимального элемента, второй их количества.

Текст программы:

```
const n=10;
 var a:array[1...n] of real;
    AMIN :real;
    i,к: integer;
 begin
 .....................
    AMIN:=A[1];
    for i:= 2 to n do
        if A[i]<AMIN then  AMIN:=A[i];
    k:=0;
  for i:= 1 to n do
        if A[i]=AMIN then  k:=k+1;;
      end;
    writeln('мин.элемент=',AMIN,' встречается ',k , ' раз');
```

```
  ...
end.
```

Пример 2.
Поиск минимального элемента, среди положительных элементов.
Здесь необходимы два изменения, первое - переменной AMIN, присвоить значение первого положительного элемента, это делаем в цикле FOR

```
 im:=0;
 For i:= 1 to n do
   if (A[i]>0)  and ( im=0) then im:=i;
```

и второе в сравнение if A[i]<AMIN then добавить условие положительности элемента массива. if (A[i]>0)and (A[i]<AMIN) then

Текст программы:

```
 const n=10;
var a:array[1...n] of real;
            AMIN :real;
            i,к, im: integer;
          begin
          .....

    im:=0;
     For i:= 1 to n do
        if (A[i]>0)  and ( im=0) then im:=i;
        AMIN:=A[im];
            for i:= im+1 to n do
                if (A[i]>0)and (A[i]<AMIN)  AMIN:=A[i];

            writeln('мин.элемент среди положительных =',AMIN);
          ...
        end.
```

4. Методы сортировки массива.

Сортировка — это расположение элементов по возрастанию или убыванию, из многочисленных методов сортировки рассмотрим два самых простых

4.1 Сортировка посредствам перестановки по индексам.

Суть метода: рассматриваем массив состоящий как бы из двух частей отсортированной и не отсортированной.

- Выбираем наименьший элемент не отсортированной части массива
- Меняем этот элемент с первым элементом не отсортированной части
- Уменьшаем не отсортированную часть на этот элемент

Данные действия повторяются до тех пор, пока в не отсортированной части останется один элемент.

Пример: исходный массив 7, 9, 5, 3, 8.
Примечание: # - обозначается наименьший элемент не отсортированной части.

	1 просмотр	2 просмотр	3 просмотр	4 просмотр	5 просмотр
Отсортированные		3	3 5	3 5 7	3 5 7 8
не отсортированные	7 9 5 3 # 8	9 5 # 7 8	9 7 # 8	9 8 #	9

МАССИВ ОТСОРТИРОВАН!

Первоначально отсортированная часть пуста, не отсортированная весь массив.

Алгоритм

I – указывает на начало не отсортированной части массива
M – индекс наименьшего элемента не отсортированной части
O – используется для просмотра не отсортированной части

FOR I = 1 **TO** N-1 **DO**
BEGIN
Поиск наименьшего элемента с индексом M
в части массива: I+1, I+2, . . . , N
A[I]⇔ A[M]
END

Поиск минимального элемента осуществляется в цикле по J

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

Задача обмена элементов A[I] и A[M] решается элементарно вводом вспомогательной переменной буфера P, в котором запоминается значение A[I] ; P:=A[I]; A[I]:= A[M]; A[M]:=P.

Текст программы:

```
CONST N=20;
```

```
VAR a : array[1..N] of integer;
        i,j: integer;
        m, p : integer;
BEGIN
   {   задание массива случайных чисел }
   Randomize;
   For i:=1 to n do
      a[i]:=random(41) – 20;
    writeln(‘ Исходный массив ‘);
    For i:=1 to n do
    write( a[i]:6);
   writeln;
  {  сортировка }
For i:=1 to n-1 do
begin
  m:=i;
  for j:=i+1 to n do
      if a[j] < a[m] then begin m:=j; end;
  p:=a[i];
  a[i]:=a[m];
  a[m]:=p;
end;
{ вывод }
    writeln(‘ Отсортированный массив ‘);
    For i:=1 to n do
    write( a[i]:6);
   writeln;
   readln;
END.
```

4.2 .<u>Метод перестановки соседних элементов (пузырька)</u>

Суть метода, последовательно просматривая массив, упорядочивать соседние элементы, после такого просмотра получается более упорядоченный массив, поэтому очевидно, что после нескольких просмотров массив будет упорядочен полностью.

Пример: Исходный массив 7, 9, 5, 3, 8.

	?/7	7	7	7	7
	\9	?/9 ⌝	5	5	5
1 просмотр	5	\5 ⌟	?/9 ⌝	3	3
	3	3	\3 ⌟	?/9	8
	8	8	8	\8	9
	?/7	5	5	5	5
	\5	?/7 ⌝	3	3	3
2 просмотр	3	\3 ⌟	?/7 ⌝	7	7
	8	8	\8 ⌟	?/8	8
	9	9	9	\9	9
	?/5 ⌝	3	3	3	3
3 просмотр	\3 ⌟	?/5	5	5	5
	7	\7	?/7 ⌝	7	7
	8	8	\8 ⌟	?/8	8
	9	9	9	\9	9

Массив упорядочен!

<u>Алгоритм</u>

J=0 {J – номер просмотра}
WHILE (Массив не упорядочен) **Do**
BEGIN
J=J+1
FOR I=1 TO N **DO**
Упорядочивание соседних пар
END {while}

Теперь в алгоритме необходимо уточнить два блока: первый упорядочивание соседних пар это есть сравнение двух элементов A[I] и A[I+1] с их последующей перестановкой.

IF (A[I] > A[I+1]) **THEN** A⇔] A[I+1]

И, второе, необходимо уточнить признак упорядоченности массива. Для этого еще раз посмотрим на пример и заметим, что в результате одного просмотра:

- Самый большой элемент оказывается в конце массива
- Самый меньший элемент сдвигается на одну позицию к началу

Число необходимых просмотров для полной сортировки определяется наихудшим
случаем, а в худшем случае, когда наименьший элемент находится в конце массива, для его перемещения в начало массива длиной n, требуется n-1 просмотр, тогда условие окончания можно записать в виде J<N.

Программа в этом случае выглядит так:

```
{программа сортирует массив методом пузырька}
Var a: array[1..100] of integer;
 i, j, n, p: integer;
begin
 …
  j:=1;
 while j<n do
 begin
       j:=j+1;
      for i:=1 to n do
           if a[i]>a[i+1] then
    begin
             p:=a[i];
              a[i]:=a[i+1];
                a[i+1]:=p;
           end;
     end;

 …
 end.
```

В программе введены:

- Параметр P – использование как буфер для сохранения значения A[I] при обмене.
- Параметр N-J в цикле for? Используется для уменьшения времени работы программы, так как после просмотра самый большой элемент оказывается на своем месте и его из дальнейшего рассмотрения можно исключить.

Программу можно усовершенствовать введя иной признак упорядоченности массива, а именно если не было ни одной перестановки, то, очевидно, массив упорядочен. Для проверки этого условия введем переменную T, перед циклом for дадим ей начальное значение TRUE, а в цикле, если происходит перестановка, то изменим значение на FALSE, тогда, если на выходе из цикла FOR значение T осталось равным TRUE, то перестановок не было

```
      j:=1; T:=true
         while T do
     begin
            j: =j+1;
            T: = true
      For j: = 1 to n-j do
        If a[i] > a[j+1] then
     Begin
        T:=false
      P:=a[i];
      a[i]:=a[j+1];
           a[j+1]:=p;
      end;
end.
```

Данное усовершенствование дает особо значимый эффект, когда исходный массив почти упорядочен, тогда сортировке достаточно всего лишь несколько проходов.

Задача:

Если в массиве отрицательных элементов больше, чем положительных, то отсортировать массив по возрастанию, иначе по убыванию.

Сортировка - по убыванию, отличается от сортировки по возрастанию, тем, что при сравнении вместо знака < надо использовать знак >, что бы сортировку проводить по единому алгоритму будем при сравнивать не сами элементы, а элементы умноженные на коэффициент К=1, если сортировка по возрастанию и коэффициент К=-1,если сортировка по убыванию.

Текст программы:

```
Const n=10
Var a; array [1..n]of integer;
        Var k, m, am, i, j, kolot, kolpol : integer;
Begin
{элем. Массива случайные числа из диапазона -20 до 20}
        Randomize
          For j:=1 to n do a[i]:=random(41)-20;
             Writeln (‘исходн. Массив’);
          For j:=1 to n do write(a[i] : 6);
        Writeln;
{проверка условия}
kolot:=0;       kolpol:=0;
For i:= 1 to n do
   Begin
        If a[i] < 0 then kolot:= kolot + 1;
        If a[i] > 0 then kolpol:= kolpol + 1;
     End;
If kolot >kolpol then k:=1 else k:=-1
{ сортировка }
        For i:=1 to n-1 do
Begin
         m:= i ;
         am:= a[i];
              For j:=i+1 to n do
                   If k*a[j] < k*a[m] then
        Begin
                 am:=a[j]        m:=j        end;
a[m]:=a[i];
```

```
a[i]:=am;
end;
writeln(‘отсортированный массив’);
        For i:= 1 to n do
          Write(a[i] : 6);
Readln;
End.
```

5. Задачи для самостоятельного решения

Задача 1.

1. В одномерном массиве, найти количество нечетных отрицательных элементов
2. В одномерном массиве, найти сумму нечетных отрицательных элементов
3. В одномерном массиве, найти произведение нечетных отрицательных элементов
4. В одномерном массиве, найти количество пар сумма элементов которых, четна и положительна. (пара — это два рядом стоящих элемента)
5. В одномерном массиве, найти количество пар сумма элементов которых, не четна и отрицательна (пара — это два рядом стоящих элемента)
6. В одномерном массиве, найти количество пар модуль разности элементов которых, больше 10. (пара — это два рядом стоящих элемента)
7. В одномерном массиве, найти количество пар элементов разного знака . (пара — это два рядом стоящих элемента)
8. Найти индекс первого положительного элемента массива, если таких элементов нет, то сообщить об этом
9. Найти индекс первого отрицательного элемента массива, если таких элементов нет, то сообщить об этом
10. Найти индекс первого четного элемента массива, если таких элементов нет, то сообщить об этом
11. Найти индекс первого не четного элемента массива, если таких элементов нет, то сообщить об этом
12. Найти индекс первого положительного, четного элемента массива, если таких элементов нет, то сообщить об этом

Задача 2.

1. Если сумма элементов в массиве, стоящих на четных местах больше суммы элементов, стоящих на нечетных местах, то отсортировать массив по возрастанию, иначе – по убыванию.
2. Если в массиве есть отрицательные элементы, то отсортировать массив по возрастанию, иначе - по убыванию.
3. Если количество четных элементов в массиве превышает количество нечетных, то отсортировать массив по возрастанию, иначе – по убыванию.
4. Если в массиве есть нулевые элементы, то отсортировать массив по возрастанию, иначе – по убыванию.

5. Если сумма четных элементов в массиве превышает сумму нечетных, то отсортировать массив по возрастанию, иначе – по убыванию.
6. Если в массиве есть четные элементы кратные 11, то отсортировать массив по возрастанию, иначе – по убыванию.
7. Если сумма отрицательных элементов по модулю превышает сумму положительных, то отсортировать массив по возрастанию, иначе – по убыванию.
8. Если в массиве есть отрицательные элементы, кратные 71, то отсортировать массив по возрастанию, иначе по убыванию.
9. Если количество четных элементов, стоящих на нечетных местах, превышает количество нечетных элементов, стоящих на четных местах, то отсортировать массив по возрастанию, иначе по убыванию.
10. Если в массиве есть более одного элемента кратного 13, то отсортировать массив по возрастанию, иначе – по убыванию.
11. Если среднеарифметическое элементов массива больше 5, то отсортировать массив по возрастанию, иначе – по убыванию.
12. Если в массиве есть повторяющиеся элементы, то отсортировать массив по возрастанию, иначе по убыванию.