№ 27 (316) 16–22 июля 2001

еженедельная газета Объединения педагогических изданий «ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

Основы программирования

В.М. Окулов

Окончание. Начало см. № 23

Занятие 21. Поиск данных

План занятия

- изучение различных схем поиска данных, написание и исследование программ по образцам процедур поиска;
- выполнение самостоятельной работы.

Линейный поиск

Основной задачей поиска является нахождение в заданном множестве элемента, обладающего заданным свойством. Большинство таких задач сводится к простейшей — поиску в массиве элемента с заданным значением.

Итак, пусть требуется найти в массиве *A* номер элемента, имеющего значение *X*. Предполагается, что никакой дополнительной информации ни об этом номере, ни о самом массиве нет.

Будем последовательно просматривать массив и сравнивать значения элементов массива *A* с *X*. Напишем соответствующий фрагмент: `For i:= 1 To N Do If A[i] = X Then k := i;`

При выполнении этого цикла мы найдем номер последнего элемента в *A*, равного *X*, при этом массив просматривается полностью. А если надо найти номер первого такого элемента?

```
For i := N DownTo 1 Do If A[i] = X Then k := i;
```

Но и в этом случае остаются лишние проверки. Совпадение найдено, зачем продолжать работу?

```
i := 1;
While (i <= N) and (A[i] <> X) Do Inc(i);
If i = N + 1 Then <X нет в A> Else <значение i указывает на место совпадения>;
```

С помощью этого алгоритма мы решаем задачу поиска первого элемента со значением *X*. Для нахождения последнего алгоритм нужно изменить.

```
i := N;
While (i > 0) and (A[i] <> X) Do Dec(i);
If i = 0 Then <X нет в A> Else <значение i указывает на место совпадения>;
```

Замечания для учителя

1. Необходимо обратить внимание на порядок проверки составного условия продолжения цикла. Предполагается, что второе условие не проверяется, если результат логического выражения ясен после проверки первого условия. В противном случае возникает ошибка из-за выхода индекса за границы массива.
2. При изучении данной темы в программах удобно использовать директиву `{$R+}`.
3. В качестве упражнения можно переписать приведенный фрагмент с использованием цикла **Repeat** … **Until**.

   Вспомним технику "барьерных" элементов, которая использовалась при изучении методов сортировки, и применим ее для поиска элемента в массиве. Нам потребуется изменить описание массива (предусмотреть в нем лишний элемент):

   (`Type MyArray = Array[0..NMax] Of Integer;` или
   `MyArray = Array[1..Nmax + 1] Of Integer; Var A : MyArray;`

   Значение `Nmax` определяется в разделе констант.

   ```
   i := 1; A[N + 1] := X;
   While A[i] <> X Do Inc(i);
   If i = N + 1 Then <X нет в A> Else <значение i указывает на место совпадения>;
   ```

   Вариант для поиска последнего совпадения:

   ```
   i := N; A[0] := X;
   While A[i] <> X Do Dec(i);
   If i = 0 Then <X нет в A> Else <значение i указывает на место совпадения>;
   ```

   Очевидно, что число сравнений зависит от местонахождения искомого элемента. В среднем количество сравнений равно $(N + 1)$ ***div*** 2. В худшем случае, когда элемента нет в массиве или он рассматривается последним, количество сравнений равно *N*. Стало быть, эффективность такого поиска — $O(N)$.

Бинарный поиск

Предположим, что нам известна некоторая дополнительная информация о массиве *A*. Скажем, известно, что он отсортирован. Тогда удается сократить время поиска, используя бинарный поиск, который еще называют двоичным, дихотомическим, поиском методом деления пополам...

Пусть для определенности массив A отсортирован в порядке неубывания.

Это позволяет по результату сравнения со средним элементом массива исключить из рассмотрения одну из половин. С оставшейся частью процедура повторяется. И так до тех пор, пока не будет найден искомый элемент или не будет просмотрен весь массив.

Пример

Пусть $X = 6$, а массив A состоит из 10 элементов:

3 5 6 8 12 15 17 18 20 25.

1-й шаг. Найдем номер среднего элемента: $m = \left[\frac{1+10}{2}\right] = 5$. Так как $6 < A[5]$, то далее рассматриваются только элементы, индексы которых меньше 5.

3 5 6 8 ~~12 15 17 18 20 25~~.

2-й шаг. Находим индекс среднего элемента оставшейся части массива: $m = \left[\frac{1+4}{2}\right] = 2$, $6 > A[2]$; следовательно, первый и второй элементы из рассмотрения исключаются:

~~3 5~~ 6 8 ~~12 15 17 18 20 25~~.

3-й шаг. Рассматриваем два элемента, значение $m = \left[\frac{3+4}{2}\right] = 3$:

~~3 5~~ 6 8 ~~12 15 17 18 20 25~~;

$A[3] = 6$. Элемент найден, его номер — 3.

Программная реализация бинарного поиска может иметь следующий вид:

```
Procedure Search;
 Var i, j, m: Integer; f: Boolean;
 Begin
  i := 1; j := N; {На первом шаге рассматривается весь массив.}
  f := False; {Признак того, что X не найден.}
  While (i <= j) and not f Do Begin
   m := (i + j) div 2;
  {Или m := i + (j - i) div 2, так как i + (j - i) div 2 = (2 * i + (j - i)) div 2 = (i + j) div 2.}
   If A[m] = X Then f := True {Элемент найден. Поиск прекращается.}
   Else If A[m] < X Then i := m + 1
    {Исключаем из рассмотрения левую часть A.} Else j := m - 1 {Правую часть.}
 End
End;
```

Рассмотрим еще одну, рекурсивную, реализацию предложенного алгоритма. Условимся, что ненулевое значение переменной t является индексом искомого элемента, а нулевое покажет, что элемент не найден.

```
Procedure Search(i, j: Integer; Var t: Integer);
{Массив A и переменная X — глобальные величины}
 Var m: Integer;
 Begin
  If i > j Then t := 0
   Else Begin m := (i + j) div 2;
         If A[m] < X Then Search(m + 1, j, t)
         Else If A[m] > X Then Search(i, m - 1, t)
         Else t := m
        End
 End;
```

Поскольку каждое сравнение уменьшает диапазон поиска приблизительно в два раза, то общее количество сравнений имеет порядок $O(\log_2 N)$.

Рассмотрим обратную задачу. Даны позиция элемента *Pos* в неупорядоченном массиве A из N элементов и количество допустимых сравнений (L). Определить все интервалы значений N в диапазоне от 1 до 10 000, при которых элемент A с номером *Pos* может быть найден за L сравнений с помощью алгоритма бинарного поиска.

Массив A как таковой в решении задачи не требуется. Необходимо проверять, совпадает ли средний элемент отрезка в схеме бинарного поиска со значением *Pos* и за сколько сравнений достигается это совпадение. Оформим модифицированную схему бинарного поиска как функцию *Can:*

```
Function Can(N: Integer): Boolean;
  Var i, p, q, ll: Integer;
  Begin
    p := 0; q := N - 1; ll := 0;
    While p <= q Do  Begin
       i := (p + q) div 2;
       Inc(ll);
      If i = Pos Then  Begin
         Can := (ll = L); Exit
      End
      Else If i > Pos Then q := i - 1 Else p := i + 1
    End;
    Can := False
  End;
```

Поясним работу функции последовательными значениями ее переменных при различных значениях N ($Pos = 10$ и $L = 3$).

p	Q	*i*	*ll*	*N*	*Can*
0	11	5	1	12	
6	11	8	2	—	
9	11	10	3	—	True
0	12	6	1	13	
7	12	9	2	—	
10	12	11	3	—	
10	10	10	4	—	False
0	13	6	1	14	
7	13	10	2	—	False

p	Q	*I*	*ll*	*N*	*Can*
0	14	7	1	15	
8	14	11	2	—	
8	10	9	3	—	
10	10	10	4	—	False
0	15	7	1	16	
8	15	11	2	—	
8	10	9	3	—	
10	10	10	4	—	False

Итак, только одно значение N (из рассмотренных в таблице) удовлетворяет условию задачи ($N = 12$). Осталось организовать цикл по количеству элементов в массиве и запомнить интервалы значений, удовлетворяющих условиям поиска (функция `Can` — внутренняя по отношению к процедуре `Solve`):

```
Procedure Solve;
Const MaxN = 10000;
MaxCnt = MaxN shr 1;
Var i, j: Integer;
Res: Array [1..MaxCnt, 1..2] Of Integer;
Pos, L, Cnt: Integer;
Begin
  ReadLn(Pos, L);
  Cnt := 0; i := Pos;
  Repeat
      j := i;
      While (j <= MaxN) and Can(j) Do Inc(j);
      If i < j Then Begin
       Inc(Cnt);
       Res[Cnt, 1] := i; Res[Cnt, 2] := j - 1 {Запоминаем интервал значений.}
     End;
     i := j + 1
  Until (i >= MaxN);
  WriteLn(Cnt);
  For i := 1 To Cnt Do WriteLn(Res[i, 1], ' ', Res[i, 2])
 End;
```

Ответом задачи при $Pos = 10$ и $L = 3$ являются четыре интервала: 12—12, 17—18, 29—30, 87—94, а при $Pos = 9000$ и $L = 2$ таких интервалов нет вообще.

Случайный поиск

Поиск в неупорядоченном массиве можно организовать так:

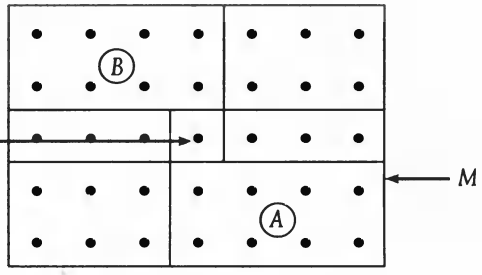
1. Выбирается случайным образом элемент с номером q.
2. Массив X разбивается на три части: элементы, меньшие $X[q]$, равные $X[q]$ и большие $X[q]$.
3. В зависимости от количества элементов в каждой части выбирается одна из них для дальнейшего поиска.

Теоретическая оценка числа сравнений этого метода имеет порядок $O(k \cdot N)$, т.е. для худшего случая $O(N^2)$, но на практике он работает значительно быстрее.

Соответствующая функция приведена ниже:

```
Function Search(N, k: Integer; X: MyArray): Integer;
{N — количество элементов в массиве X, k — номер по порядку разыскиваемого элемента.}
  Var i, cnl, cne, cnm, q: Integer;
      Sl, Se, Sm: MyArray;
  Begin
   If N = 1 Then Search := X[1]
    Else Begin
     FillChar(Sl, SizeOf(Sl), 0);
     FillChar(Se, SizeOf(Se), 0);
     FillChar(Sm, SizeOf(Sm), 0);
     cnl := 0; cne := 0; cnm := 0;
     q := Integer(Random(N)) + 1;
     For i := 1 To N Do{Разбиение на три части.}
      If X[i] < X[q] Then Begin Inc(cnl); Sl[cnl] := X[i] End
       Else If X[i] = X[q] Then Begin Inc(cne); Se[cne]:=X[i] End
            Else Begin Inc(cnm); Sm[cnm] := X[i] End;
     If cnl >= k Then Search := Search(cnl, k, Sl){Выбор части для дальнейшего поиска.}
      Else If cnl + cne >= k Then Search := X[q]
           Else Search := Search(cnm, k - cnl - cne, Sm)
   End
  End;
```

Поиск элемента произвольного массива за линейное время



Для реализации этой схемы поиска требуется уметь сортировать массив из пяти элементов за 7 сравнений. Идея поиска поясняется рисунком.

Вначале исходный массив разбивается на пятерки, и каждая из них сортируется за 7 сравнений.

Из отсортированных пятиэлементных подмассивов формируется новый массив из средних элементов — медиан (M). В массиве M выбирается средний элемент m, и относительно него уже исходный массив разбивается на три части, как в предыдущей схеме случайного поиска. В области A на рисунке выделены элементы, заведомо большие m, в области B — меньшие m. Как минимум четвертая часть элементов исходного массива исключается из дальнейшего поиска за счет однократного применения этой схемы.

```
Function Search(N, k: Integer; X: MyArray): Integer;
{N — количество элементов в массиве X, k —  номер элемента (k ≤ N).}
  Var i, j, cnl, cne, cnm, q, r: Integer;
      M, Sl, Se, Sm, Y: MyArray;
  Begin
   If N <= 5 Then Begin Sort(N, X); Search := X[k] End
   {Если количество элементов в массиве меньше или равно 5,
    то сортируем элементы массива за 7 сравнений и выбираем k-й элемент.}
    Else Begin
     i := 1; r := 0;
     While i <= N Do Begin
      j := 0;
      While (j <= 5) and (i + j <= N) Do Begin
       Inc(j); Y[j] := X[i + j - 1] {Разбиваем на пятиэлементные массивы (не более).}
      End;
      Inc(i, 5);
      Sort(j, Y);
      Inc(r);
      M[r] := Y[j div 2 + 1] {Средний элемент отсортированного массива Y записываем в массив M.}
     End;
     q := Search(r, r div 2, M); {Находим средний по значению элемент в массиве медиан.}
     cnl := 0; cne:= 0; cnm := 0;
     FillChar(Sl, SizeOf(Sl), 0);
     FillChar(Se, SizeOf(Se), 0);
     FillChar(Sm, SizeOf(Sm), 0);
     For i := 1 To N Do{Разбиваем массив на три части.}
      If X[i] < q Then Begin Inc(cnl); Sl[cnl] := X[i] End
       Else If X[i] = q Then Begin Inc(cne); Se[cne]:=X[i] End
            Else Begin Inc(cnm); Sm[cnm]:= X[i] End;
     If cnl >= k Then Search := Search(cnl, k, Sl){Выбор части массива для дальнейшего поиска.}
      Else If cnl + cne >= k Then Search := X[q]
           Else Search := Search(cnm, k - cnl - cne, Sm)
   End
  End;
```

Оценим порядок функции $T(N)$, где T — время работы нашего алгоритма.

После поиска элемента m не менее половины найденных медиан будут больше или равны m. Следовательно, по крайней мере половина $\lceil N/5 \rceil$ групп даст по три числа, больших или равных m, кроме, быть может, последней группы и группы, содержащей m. Итак, $3 \cdot (\lceil 1/2 \cdot \lceil N/5 \rceil - 2 \rceil) \geq 3 \cdot N/10 - 6$ элементов исключаются из дальнейшей обработки.

Остается $7 \cdot N/10 + 6$ элементов. Поиск среднего элемента в массиве медиан (выделен в программе жирным шрифтом) требует $T(N/5)$ времени. Рекурсивный вызов процедуры поиска (выделен жирным шрифтом) при разбивке массива требует $T(7 \cdot N/10 + 6)$. Остальные шаги алгоритма, включая сортировку 5-элементных массивов, выполняются за время $O(N)$. Таким образом, $T(N) \leq T(N/5) + T(7 \cdot N/10 + 6) + O(N)$. Сумма коэффициентов в правой части при N: $1/5 + 7/10 = 9/10$ меньше 1. Существует теорема, согласно которой $T(N) \leq c \cdot N$ для некоторой константы c. Стало быть, порядок алгоритма — $O(N)$.

Продолжим тему *бинарного поиска*. Рассмотрим следующую задачу. Дан целочисленный массив *Mas* из N элементов (N — нечетно и $5 \leq N \leq 1499$). Значения элементов массива различны и принимают значения от 1 до N. Пример: $N = 5$. Массив: 2, 5, 4, 3, 1. Требуется найти номер элемента i такой, что количество элементов *Mas*, меньших *Mas*[i], равно количеству элементов, больших, чем *Mas*[i]. В нашем примере $i = 4$.

Единственной разрешенной операцией при поиске является вызов функции *Med3*. Ее аргументы — три различных номера элементов массива, а возвращает она номер элемента, имеющего среднее значение.

Последовательность применения функции *Med3*:

1. *Med3*(1, 2, 3) возвращает 3 (т.е. третий элемент лежит между первым и вторым).
2. *Med3*(3, 4, 1) возвращает 4 (четвертый элемент лежит между третьим и первым. Иными словами, "по одну сторону" от четвертого элемента оказались третий и второй).
3. И, наконец, *Med3*(4, 2, 5) возвращает 4. (Четвертый элемент к тому же лежит между вторым и пятым. Это означает, что первый и пятый элементы оказались "по другую сторону" от четвертого.) Таким образом, элемент № 4 — искомый.

Проверьте самостоятельно, решает ли поставленную задачу та же последовательность вызовов функции *Med3* для массива 2, 6, 8, 7, 1.

Но что делать с произвольным массивом? Рассмотрим вначале работу с еще одним конкретным массивом. Пусть есть массив *Mas:*

Номер	1	2	3	4	5	6	7	8	9	10	11	12	13
Значение	8	1	3	7	5	4	13	10	2	11	12	9	6

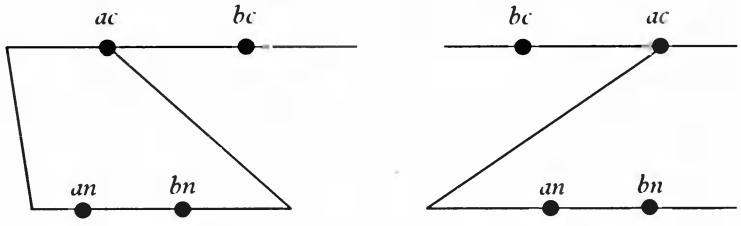
Предлагается следующая процедура использования функции *Med3*:

1. Выбираются случайным образом два не совпадающих номера (a и b) элементов. Пусть для определенности $a = 5$ и $b = 12$.
2. Просматриваем все номера элементов, помеченные признаком ***True***. (В начальный момент времени для всех элементов этот признак равен ***True***. В дальнейшем мы будем помечать этим признаком и части массива, требующие такого же просмотра.) Если операция *Med3*(a, b, i) дает в качестве результата:
   - значение a, то помечаем элемент с номером i как элемент, принадлежащий левой части (признак *_Left*);
   - значение i, то помечаем элемент с номером i как элемент, принадлежащий средней части (признак *_Middle*);
   - значение b, то помечаем элемент с номером i как элемент, принадлежащий правой части (признак *_Right*).

В таблице буквами *L*, *M*, *R* обозначены признаки *_Left*, *_Midlle*, *_Right*. Буквой *N* обозначен признак *_None*, так как два элемента с выбранными номерами не рассматриваются на данном шаге анализа.

Номер	1	2	3	4	5	6	7	8	9	10	11	12	13
Признак	*M*	*L*	*L*	*M*	*N*	*L*	*R*	*R*	*L*	*R*	*R*	*N*	*M*

Итак, разбили, а что дальше? Напрашивается единственно разумный ход — выбрать одну из трех частей и продолжить поиск. Вся сложность задачи — в выборе параметров поиска и в аккуратном переходе от одного этапа поиска к другому.



Какую часть выбрать для следующего этапа поиска? После первого этапа в левой части оказались 4 элемента, в правой — 4 и в средней (без учета a и b) — 3. Первоначально осуществлялся поиск элемента со значением 7 в массиве из 13 элементов. После этого этапа необходимо искать элемент со вторым значением среди элементов, помеченных признаком *_Middle*.

Часть параметров поиска определена: количество и признак элементов, среди которых ищется искомый; номер элемента. Еще один параметр — учет одного из предыдущих значений a или b — требует пояснений (это самый сложный момент). Поясним его выбор рисунком.

Оказалось, что поиск следует продолжать в левой (по отношению к a) части. Генерируем новые значения a и b (обозначены как an, bn, в отличие от старых an, bn). Если после этого средний элемент (*Med3*(an, bn, ac)) находится

не на месте *bn*, то структура последующего поиска нарушается. Стало быть, необходимо поменять значения у *an* и *bn*. Иными словами, если после первого этапа поиска мы выяснили, что его следует продолжать в левой части, то и следующее деление на три части должно осуществляться с учетом этого результата.

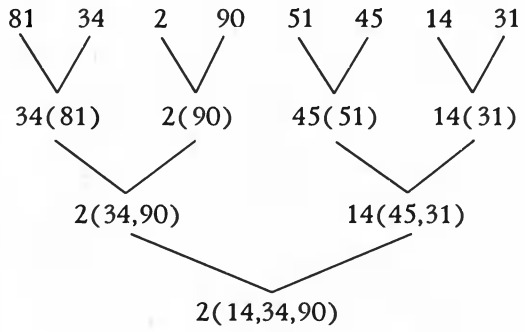
Примечания. 1. Прервите чтение и проделайте ручной просчет поиска по данным из условия задачи без смены значений. Результат уже на этом простом примере окажется отрицательным.

2. Сделайте еще два рисунка и выясните особенности использования функции *Med3* при продолжении поиска в средней и правой частях.

```
Program Median; {Без ввода элементов исходного массива M и функции Med3.}
  Const   MaxN = 1500;
  Type  TDir = (_None, _Middle, _Left, _Right);
  Var  N: Integer;
    Mas : Array [1..MaxN] Of TDir;
    Lt, Rt : Integer; (Количество элементов в подмножествах)
  Procedure MakeMiddle(Dir: TDir); {Помечаем элементы, среди которых осуществляется поиск.}
    Var i: Integer;
  Begin
    For i := 1 To N Do
     If Mas[i] = Dir Then Mas[i] := _Middle Else Mas[i] := _None
  End;
  Procedure Swap(Var a, b: Integer);
    Var c: Integer;
  Begin  c := a; a := b; b := c End;
  Procedure GetNext(Var a, b: Integer; len: Integer); {Генерируем номера a и b.}
    Var i, t, ca, cb: Integer;
  Begin
    t := 0;
    ca := Random(len) + 1;
    cb := Random(len) + 1; While ca=cb Do cb := Random(len) + 1;
   For i := 1 To N Do {Сложность в том, что номера оставшихся элементов для поиска идут не подряд.}
     If Mas[i] = _Middle Then  Begin
         Inc(t); If ca = t Then a := i; If cb = t Then b := i
       End
  End;
 Procedure Doit(Dir: TDir; Res, len, lb: Integer);
{Основная процедура, параметры поиска описаны ранее.}
    Var a, b, i, t: Integer;
  Begin
   MakeMiddle(Dir); {Выделяем элементы, среди которых осуществляется поиск.}
   If len = 1 Then Begin a := 1; While Mas[a] <> _Middle Do Inc(a);
   WriteLn(a)
  End
    Else  Begin
    GetNext(a, b, len);
    If (Dir = _Right) Then Begin If Med3(lb, a, b) <> a Then Swap(a, b) End
     Else If Med3(a, b, lb) <> b Then Swap(a, b);
   {Корректируем значения a и b с учетом предыдущего поиска. Пояснение приведено на рисунке в тексте.}
     If len = 2 Then Begin If Res = 1 Then WriteLn(a) Else WriteLn(b) End
      Else Begin
          lt := 0; rt := 0;
          Mas[a] := _None; Mas[b] := _None;
          For i := 1 To n Do
           If Mas[i] = _Middle Then Begin
             t := Med3(a, b, i);
             If t = b Then Begin Inc(rt); Mas[i] := _Right End{Пишем в правую часть.}
             Else If t = a Then Begin Inc(lt); Mas[i] := _Left End
           End;
          If Res = lt + 1 Then WriteLn(a) Else
          {Искомый элемент попал на границу интервала, поиск завершен.}
          If Res = len - rt Then WriteLn(b) Else
          If Res < lt + 1 Then Doit(_Left, Res, lt, a){Поиск с новыми параметрами.}
          Else If res < len-rt Then Doit(_Middle, Res-lt-1, len-lt-rt-2, b)
          Else Doit(_Right, Res-len+rt, rt, b)
        End
    End
  End;
  Begin
    ReadLn(N); {И ввод исходного массива…}
    Randomize; FillChar(Mas, SizeOf(Mas), _Middle);
    Doit(_Middle, N div 2 + 1, N, 1)
    {Первый вызов, N div 2 + 1 — среднее значение,
     1 - условно считаем, что ранее был первый номер (обеспечивает правильный поиск).}
  End.
```

Поиск второго минимального элемента в массиве

Известно, что второй минимальный элемент в массиве из N элементов можно найти за $N + \lceil \log_2 N \rceil - 2$ сравнения. Так, при $N = 8$ это 9 сравнений. Попробуйте свои силы, самостоятельно найдя алгоритм решения задачи без дальнейшего чтения.



Очевидно, что без поиска минимального элемента найти второй минимальный невозможно. Поиск минимального элемента требует $N - 1$ сравнения, и от этого никуда не уйти. Осталось $\lceil \log_2 N \rceil - 1$ сравнение. Повторное применение алгоритма поиска минимального элемента в оставшейся части массива потребует $N - 2$ сравнения. Однако при этом мы как бы забываем об операциях, которые делали на первом этапе. Значит, чтобы сократить общее количество сравнений, необходимо уже на первом этапе использовать их для поиска и второго минимального элемента. Но как это сделать?

Пусть $N = 8$ и элементы массива A имеют значения: 81, 34, 2, 90, 51, 45, 14, 31. На рисунке изображена схема поиска минимального элемента. Сравниваем соседние пары элементов, а затем сравниваем минимальные элементы пар и т.д.

Если убрать записи в круглых скобках на рисунке, то это обычный поиск минимального элемента в массиве за $N - 1$ сравнение (напишите процедуру такого поиска). А сейчас обратимся к записям в круглых скобках. Рассмотрим, например, 2(34, 90). Почему мы не включили 81? Да потому, что в процессе выполнения алгоритма оставляется цепочка от меньшего элемента пары, которая постоянно растет за счет большего элемента из пары, взятого уже без своей цепочки. Последнее, 7-е, сравнение оставляет в этой цепочке 2(14, 34, 90). Осталось в конечной цепочке из трех элементов найти второй минимальный элемент. Это легко делается за два сравнения. Общее же количество сравнений равно 9.

Постройте точно такую же схему для массива из 16 элементов. Через 15 сравнений в последней цепочке окажутся 4 элемента. За 3 сравнения мы найдем второй минимальный элемент. Итого 18 сравнений.

Перейдем к реализации данного алгоритма. Вначале опишем данные:

```
Program SearchTwoMin; {Без ввода исходных данных, вывода результата и основной программы.}
 Const MaxN=10000;
 Type MyArray=Array[1..MaxN] Of Integer;
 Var A, IndNext, IndFirst: MyArray;
 {Массив элементов. Номер следующего элемента цепочки. Номера первых элементов цепочек.}
    N: Integer; {Количество элементов в массиве A.}
    CntIf: LongInt; {Счетчик числа сравнений.}
    res: Integer; {Второй минимальный элемент.}
```

Предполагается, что после поиска минимального элемента в дополнительном массиве IndNext записаны номера элементов последней цепочки, например, так: *, 4, 7, 0, *, *, 2, *, — а номер минимального элемента массива (а это 3 для рассматриваемого примера) является значением переменной IndFirst[1]. Символом "*" обозначено не интересующее нас значение, оно может быть любым, а 0 — признак конца цепочки. Задача решена. Просмотр элементов последней цепочки по этим данным реализуется с помощью следующей процедуры:

```
Procedure Search;
  Var t: Integer;
   Begin
     t := IndNext[IndFirst[1]];
     {Для примера значением t является 7. Определяем номер первого элемента "цепочки".}
     res := A[t]; {Начальное присвоение.}
     t := IndNext[t]; {Переход ко второму элементу цепочки.}
     While t <> 0 Do Begin
      Inc(CntIf); {Изменяем значение счетчика числа сравнений.}
      If A[t] < res Then res := A[t]; {Изменяем значение минимального элемента.}
      t := IndNext[t] {Переадресация к следующему элементу цепочки.}
     End
   End;
```

Осталось получить значения элементов массивов IndNext и IndFirst в процессе поиска первого минимального элемента массива A (напоминаем, что мы работаем с конкретным примером). В начальный момент времени имеем 8 цепочек по одному элементу. После первого шага работы получаются 4 цепочки по два элемента. Номер первого элемента первой цепочки — 2 (IndFirst[1]), номер следующего элемента — 1 (IndNext[2]). Номер первого элемента второй цепочки — 3 (IndFirst[2]), номер следующего элемента — 4 (IndNext[3]). Номер первого элемента третьей цепочки — 6 (IndFirst[3]), номер следующего элемента — 5 (IndNext[6]). Номер первого элемента четвертой цепочки — 7 (IndFirst[4]), номер следующего элемента — 8 (IndNext[7]). Результаты действий на первом шаге обработки приведены в таблице.

Начальные значения	IndFirst								IndNext							
	1	2	3	4	5	6	7	8	0	0	0	0	0	0	0	0
A[IndFirst[1]] и A[IndFirst[2]]	2	2	3	4	5	6	7	8	0	1	0	0	0	0	0	0
A[IndFirst[3]] и A[IndFirst[4]]	2	3	3	4	5	6	7	8	0	1	4	0	0	0	0	0
A[IndFirst[5]] и A[IndFirst[6]]	2	3	6	4	5	6	7	8	0	1	4	0	0	5	0	0
A[IndFirst[7]] и A[IndFirst[8]]	2	3	6	7	5	6	7	8	0	1	4	0	0	5	8	0

В результате осталось 4 элемента, точнее, 4 цепочки элементов, номера их первых элементов являются значениями IndFirst[1] — IndFirst[4]. После второго шага остаются две цепочки элементов.

Начальные значения	IndFirst				IndNext							
	2	3	6	7	0	1	4	0	0	5	8	0
A[IndFirst[1]] и A[IndFirst[2]]	3	3	6	7	0	4	2	0	0	5	8	0
A[IndFirst[3]] и A[IndFirst[4]]	3	7	6	7	0	4	2	0	0	8	6	0

Номер первого элемента первой цепочки — 3 (IndFirst[1]), затем 2 (IndNext[3]) и 4 (IndNext[2]). Номер первого элемента второй цепочки — 7 (IndFirst[2]), затем 6 (IndNext[7]) и 8 (IndNext[6]). Осталось последнее сравнение элементов A[IndFirst[1]] и A[IndFirst[2]]. Элементы массива IndNext после корректировки ссылок примут значения: 0, 4, 7, 0, 0, 8, 2, 0, а IndFirst[1] останется равным 3. Значение A[7] больше A[3] — значит, про цепочку седьмого элемента забываем, а сам тот элемент включаем первым элементом в цепочку третьего элемента, старое же значение IndNext[3] становится значением IndNext[7].

```
Procedure Create;
  Var i, lf, rg, t: Integer;
   Begin
    CntIf := 0; {Счетчик числа сравнений.}
    FillChar(IndNext, SizeOf(IndNext), 0); {Начальные значения элементов массива IndNext.}
    For i := 1 To N Do IndFirst[i] := i; {Начальные значения элементов массива IndFirst.}
    t := N;
    While t > 1 Do Begin
    {Номер шага, t div 2 — количество сравнений на данном шаге. Для примера: 8, 4, 2.}
      i := 2;
      While i <= t Do Begin
        Inc(CntIf); {Изменяем значение счетчика числа сравнений.}
        lf := i - 1; rg := i - 1;
        If A[IndFirst[i-1]] < A[IndFirst[i]] Then rg := i
         Else lf := i;
        {В сравниваемой паре элементов массива A rg — номер большего,
         a lf — номер меньшего элементов, определяемых по значению из IndFirst.}
          {Следующие три строки кода — самый сложный момент для понимания логики работы.
           Связываем цепочки сравниваемых элементов массива A.
           Изменяем номер первого элемента новой цепочки.}
        IndNext[IndFirst[rg]] := IndNext[IndFirst[lf]];
        IndNext[IndFirst[lf]] := IndFirst[rg];
        IndFirst[i div 2] := IndFirst[lf];
        Inc(i, 2)
      End;
      If t mod 2 = 1 Then IndFirst[(t + 1) div 2] := IndFirst[t];
      {Учитываем нечетность значения N.}
      t := (t + 1) div 2
   End
  End;
```

Основная программа, как обычно, состоит из вызова процедур:

```
Begin
 Init; {Ввод исходных данных.} Create; Search; Done {Вывод результата.}
End.
```

Хеширование

Выше рассматривался поиск, основанный на сравнении значений элементов массива с некоторым заданным значением X (ключом). В конечном итоге определялся номер элемента массива, имеющего это значение.

Теперь рассмотрим следующую внешне простую задачу. Пусть в результате анализа какого-то текста мы записали в массив все встреченные буквосочетания из четырех символов. После этого на вход системы анализа текста поступает некоторое буквосочетание. И эта система должна определить, имеется оно в массиве или нет.

Очевидно, что каждому буквосочетанию соответствует некоторый цифровой код (конечно, можно хранить и сами буквосочетания). Поэтому задача сводится к поиску в массиве некоторого цифрового значения — ключа. Эта задача уже была рассмотрена ранее.

Поскольку количество различных буквосочетаний длиной не более 4 символов из 32 букв русского алфавита чуть больше 10^6, то при линейном поиске в худшем случае для ответа на вопрос "есть или нет?" потребуется 10^6 сравнений, при бинарном — порядка 50. Можно ли уменьшить количество сравнений и ослабить требования по оперативной памяти, используя массив размером не 10^6, а, например, 10^3?

Обозначим цифровой аналог буквосочетания символом R, а его адрес в массиве через A. Если будет найдено преобразование $f(R)$, дающее значение A, такое, что

- f выполняется достаточно быстро;
- случаев, когда $f(R_1) = f(R_2)$, будет минимальное количество,

то задачу поиска можно было бы решать совсем иначе. Способ поиска с использованием такой функции f называется ***хешированием*** (расстановкой ключей, рандомизацией и т.д.). Суть метода состоит в том, что ключ преобразуется в адрес. Первое требование быстрого вычисления функции очевидно. Второе требование уже не столь понятно.

Предположим, что анализируется текст из специфической страны "Д...", в которой все слова начинаются с буквы "О", а в качестве f выбран цифровой аналог первой буквы. В этом случае получаем одно значение A для всех слов (от чего уходили, к тому и пришли, но в стране "Д..." все возможно). Следовательно, необходимо выбирать такое преобразование f, чтобы количество совпадений адресов (A) для различных значений ключей (R) (это называют коллизиями) было минимально.

Как правило, полностью избежать коллизий не удается. Для любого преобразования f (хеш-функции) можно подобрать такие исходные данные, что рассматриваемый метод поиска превращается в линейный поиск. Однако цель построения хеш-функции ясна — как можно более равномерно "разбрасывать" значения ключей по таблице адресов (ее размер обозначим через M) так, чтобы уменьшить количество коллизий. Итак, реализация метода логически разбивается на две составляющие:

- построение f;
- схему разрешения коллизий.

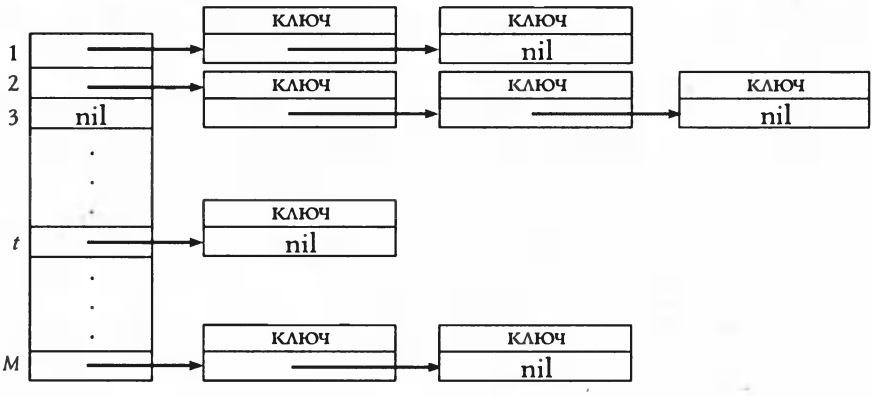
Коротко рассмотрим их.

Итак, обозначим за t цифровой аналог ключа R.

- *В первом способе* в качестве M выбирается простое число и находится остаток от деления t на M (t **mod** M).
- *Во втором способе* находится значение t^2. Из этого значения "вырезаются" q каких-то разрядов (лучше средних). Значение q выбирается таким образом, чтобы 2^q было равно M.
- *В третьем способе* w разрядов для представления ключа t разбиваются на части длиной q разрядов. Выполняется, например, логическое сложение этих частей, и его результатом является номер элемента в таблице адресов.

Считается (если не заниматься подбором специальных входных данных), что все эти способы дают достаточно хорошие (в смысле количества возможных коллизий) хеш-функции.

Из известных методов разрешения коллизий рассмотрим один — "метод цепочек". Его суть в том, что элементом таблицы является ссылка на список элементов, имеющих одно значение хеш-функции. Проиллюстрировать этот метод можно следующим рисунком.



Очевидно, что качество хеш-функции вполне можно оценивать средней длиной списков. Обработку списков и, стало быть, методы разрешения коллизий мы рассмотрим позднее, а сейчас рассмотрим следующую простую задачу:

Даны N ($1 \le N \le 15\,000$) чисел типа `LongInt` и таблица не очень большого размера. Используя случайные тесты, оценим качество работы трех различных хеш-функций по среднему и максимальному значениям количества различных чисел, для которых в результате преобразования получается один и тот же адрес в таблице. Приведем полный текст программы, тестирующей первую функцию:

```
{$R+}
Program My21_1;
 Const InputFile = 'Input.Txt';
       MaxN = 15000; {Максимальное значение количества чисел.}
       MaxTbl = 547; {Размер таблицы.}
```

```
 Var Tbl: Array[0..MaxTbl] Of LongInt;
     Count, Dif: Array[0..MaxTbl] Of Integer;
{Подсчитываем количество совпадающих ключей и количество различных, но дающих один адрес в таблице.}
N: Integer;
 Procedure RandFile; {Создание файла из N чисел.}
  Var i: Integer;
  Begin
   Randomize;
   Assign(Output, InputFile); Rewrite(Output);
   N := Integer(Random(MaxN)) + 1;
   WriteLn(N);
   For i := 1 To N Do WriteLn(LongInt(Random(65000))*29999);
   Close(Output)
  End;
 Procedure Solve;
  Var i, Ind: Integer;
      nm: LongInt;
  Begin
   FillChar(Dif, SizeOf(Dif), 0);
   FillChar(Tbl, SizeOf(Tbl), 0);
   Assign(Input, InputFile); Reset(Input);
   ReadLn(N);
   For i := 1 To N Do Begin
    ReadLn(nm); {Читаем число.}
    Ind := nm mod MaxTbl; {Вычисляем адрес в таблице.}
    If Tbl[Ind] = 0  Then Begin Tbl[Ind] := nm; Count[Ind] := 1 End
    {Первая запись в таблицу по этому адресу.}
     Else If nm = Tbl[Ind] Then Inc(Count[Ind])
           Else Inc(Dif[Ind]) {Коллизия — различные числа дают один адрес в таблице.}
   End;
   Close(Input)
  End;
 Procedure Issl;
  Var s, i, Max, r: Integer;
  Begin
   s := 0; Max := 0;
   For i := 1 To MaxTbl Do Begin
    Inc(s, Dif[i]); {Подсчитываем количество коллизий.}
    If Dif[i] > Max Then Max := Dif[i] {Определяем максимальное значение количества коллизий.}
   End;
   r := Round(s/MaxTbl);
   Assign(Output, 'Con'); ReWrite(Output);
   WriteLn(N, ' ', r, ' ', Max)
  End;
 Begin
  RandFile; {Формирование файла из случайных чисел.}
  Solve; {Анализ файла с помощью различных хеш-функций.}
  Issl {Подсчет среднего и максимального значений коллизий.}
 End.
```

Для тестирования второй и третьей функций требуется изменить одну строку в процедуре Solve — Ind := GetNum(nm), где GetNum вычисляет адрес элемента таблицы. Вторая хеш-функция GetNum имеет вид:

```
Function GetNum(nw: LongInt): Integer;
  Var w, q: LongInt;
      t: 0..MaxTbl;
  Begin
   q := (nw and $FFFF0000 ) shr 16; {Выделяем старшие разряды числа и сдвигаем на 16 разрядов.}
   w := (nw and $FFFF); {Выделяем младшие разряды числа.}
   t := (Integer(q * w) and $7FC0) shr 6; {Выделяем 9 бит произведения чисел.}
   GetNum := t
  End;
```

Третья хеш-функция кодируется следующим образом:

```
 Function GetNum(nw: LongInt): Integer;
  Var r, t: Integer;
  Begin
   r := Integer(nw and $1FF); {Выделение младших 9 бит числа.}
   t := Integer(nw and $3FE00 shr 9); {Выделение следующих 9 бит числа.}
```

```
  r := r xor t; {Первый шаг формирования адреса элемента таблицы.}
  t := Integer(nw and $7FC0000 shr 18); {Выделение следующей группы разрядов числа.}
  r := r xor t;
  t := Integer(nw and $F8000000 shr 27); {Выделение оставшихся разрядов.}
  r := r xor t;
  GetNum1 := r
End;
```

В таблице приведены результаты подсчета для всех трех рассматриваемых хеш-функций.

1			2			3		
N	*r*	Max	*N*	*r*	Max	*N*	*r*	Max
6149	10	21	11 792	20	38	7736	13	29
14 015	24	25	9320	16	32	6029	10	20
2606	4	12	6596	11	21	12 403	22	40
6886	11	22	5465	9	21	12 497	22	39
14 710	26	41	1535	2	8	1286	1	8
2219	3	11	1157	1	9	3662	6	15
2658	4	11	12 650	22	39	10 078	17	33
3693	6	14	2830	4	14	13 689	24	42
1756	2	9	8345	14	28	14 076	25	42

Данные в таблице дают лишь общую картину работы хеш-функций, и не более того. Для сравнительного анализа необходимо по крайней мере исследовать их работу на одном и том же файле из случайных чисел.

Для каждой из хеш-функций сформируйте оценки $N_{среднее}$, $r_{среднее}$, $Max_{среднее}$. Сравните работу хеш-функций по этим оценкам, например, для 100 различных файлов с исходными данными.

Задания для самостоятельной работы

1. Во входном файле даны массив *A* и массив из элементов, которые необходимо найти в массиве *A*. Изменить массив *A* таким образом, чтобы суммарное количество сравнений при поиске элементов было минимальным.

Подсказка. По данным из второго массива формируются частоты поиска каждого элемента и элементы массива *A* сортируются в порядке убывания значений этих частот ("метод перемещения в начало").

На базе этой задачи можно провести небольшое исследование. Необходимо найти значения *N*, при которых массивы не помещаются в оперативную память, отводимую под программу поиска.

2. Даны массив *A* и число *X*.

а) Переставить элементы массива таким образом, чтобы вначале были записаны элементы, меньшие или равные *X*, а затем — большие или равные *X*.

б) Переставить элементы массива таким образом, чтобы вначале были записаны элементы, меньшие *X*, затем равные *X* и, наконец, большие *X*.

Примечание. Указанные перестановки следует сделать за один просмотр *A* и без использования дополнительных массивов.

3. Найти реализацию (рекурсивную и нерекурсивную) бинарного поиска с использованием "барьерной" техники.

4. Ниже по тексту приведены три версии процедуры бинарного поиска. Какая из них верная? Исправьте ошибки. Какая из них более эффективная?

```
Procedure Search;
Var i, j, k: Integer;
Begin   i := 1; j := N;
 Repeat  k := (i + j) div 2;
  If X <= A[k] Then j := k - 1;
  If A[k] <= X Then i := k + 1
 Until i > j
End;
```

```
Procedure Search;
Var i, j, k: Integer;
Begin   i := 1; j := N;
 Repeat k := (i + j) div 2;
  If X < A[k] then j := k
  Else i := k + 1
 Until i >= j
End;
```

```
Procedure Search;
Var i, j, k: Integer;
Begin   i := 1; j := N;
 Repeat k := (i + j) div 2
  If A[k] < X Then i := k
  Else j := k
 Until (A[k] = X) or (i >= j)
End;
```

5. Дан массив из *N* различных целых чисел. Разрешенными операциями являются сложение двух элементов массива и сравнение сумм. Найти наибольший элемент массива за минимальное количество сравнений.

6. Пусть *X*[1..*N*] и *Y*[1..*N*] — два возрастающих массива из различных элементов. Найти за время $O(\log_2 N)$ средний элемент множества, полученного объединением этих массивов.

Занятие 22. Двухмерные массивы. Работа с элементами

План занятия

- структура двухмерного массива и его описание;
- шаблон для решения задач на двухмерные массивы;
- экспериментальный раздел занятия;
- выполнение самостоятельной работы.

Массивы, положение элементов в которых описывается двумя индексами, называют двухмерными. Структура такого массива может быть представлена прямоугольной матрицей. Каждый элемент матрицы однозначно определяется указанием номера строки и номера столбца. Но память компьютера — это аналог одномерного массива ячеек. Как же происходит размещение двухмерного массива в физической памяти? Известны два способа: построчное размещение и размещение по столбцам.

В конкретной системе программирования используется один из этих способов. Рассмотрим первый из них на примере. Пусть есть двухмерный массив A из целых чисел (тип Integer — 2 байта). Количество строк равно 5, количество столбцов — 4. Пусть первый элемент $A[1, 1]$ записан в ячейке с номером 1000. Вычислим адрес $A[4, 3]$.

$$\mathrm{Addr}(A[4, 3]) = 1000 + 2 \cdot (4 \cdot (4 - 1) + (3 - 1)) = 1028.$$

В каждой строке записано по 4 элемента. В трех строках — 12 элементов, в 4-й строке до 3-го элемента записано 2 элемента. Складываем и умножаем на 2, поскольку каждый элемент занимает 2 байта. В общем случае для массива $A[N, M]$ из элементов, занимающих V байт памяти, формула имеет вид:

$$\mathrm{Addr}(A[i, j]) = \mathrm{Addr}(A[1, 1]) + V \cdot (M \cdot (i - 1) + (j - 1)).$$

Двухмерный массив на языке Турбо Паскаль можно описать по-разному.

```
Const MaxN = …; MaxM = …; {Максимальные значения количества строк и столбцов двухмерного массива.}
```

Первый способ:

```
Type OMyArray = Array[1..MaxM] Of Integer; {Одномерный массив из целых чисел.}
TMyArray = Array[1..MaxN] Of OMyArray; {Одномерный массив, элементами которого являются одномерные массивы.}
```

Второй способ:

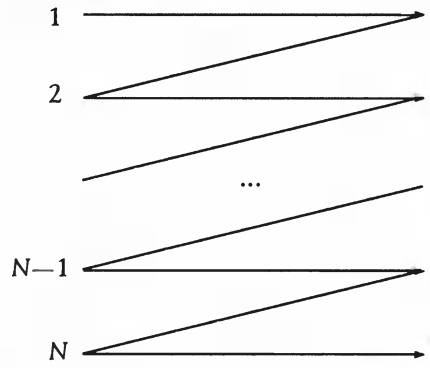
```
Type TMyArray = Array[1..MaxN, 1..MaxM] Of Integer; {Двухмерный массив из целых чисел.}
```

Мы будем отдавать предпочтение второму способу описания двухмерных массивов.

Для работы с двухмерными массивами изготовим программу — шаблон. Она осуществляет ввод данных из файла и вывод элементов двухмерного массива в файл. Размеры двухмерного массива также вводятся из файла. В дальнейшем мы не будем приводить процедуры ввода и вывода двухмерного массива, имея в виду этот шаблон:

```
{$R+}
Program My22_1;
 Const MaxN = 10; MaxM = 8;
 Type TMyArray = Array[1..MaxN, 1..MaxM] Of Integer;
 Var A: TMyArray;
     N, M: Integer;
 Procedure TInit(Var v, w: Integer; Var X: TMyArray);
  Var i, j: Integer;
  Begin
   Assign(Input, 'Input.Txt');
   Reset(Input);
   ReadLn(v, w); {В первой строке файла записаны значения N и M.}
   For i := 1 To v Do
    For j := 1 To w Do Read(X[i, j]);
   Close(Input)
  End;
 Procedure TPrint(v, w: Integer; X: TMyArray);
  Var i, j: Integer;
  Begin
   Assign(Output, 'Output.Txt');
   ReWrite(Output);
   For i := 1 To v Do Begin
    For j := 1 To w Do Write(X[i, j]: 4); WriteLn
   End;
  Close(Output)
  End;
Begin
 TInit(N, M, A);
 TPrint(N, M, A)
End.
```

Массив просматривается так, как это показано на рисунке: сначала элементы первой строки, затем — второй и так — до N-й строки.



Напомним, что для удобства работы экран должен быть разбит на три окна, которые одновременно обозримы. Изменение значений во входном файле (не забывайте его записывать) и запуск программы приводят к изменению выходного файла. Эти изменения должны быть видны без дополнительных операций по переходу от одного окна к другому. Текущим, очевидно, является каталог с файлами программы INPUT.TXT и OUTPUT.TXT. Названия входного и выходного файлов могут быть и другими.

Возможно формирование массива с использованием датчика случайных чисел и путем ввода данных с клавиатуры, но мы не будем останавливаться на этом.

Экспериментальный раздел занятия

1. Найти значение первого максимального элемента массива и его индексы. Результат вывести в файл OUTPUT.TXT.

Будем считать, что задачу решает программа, из которой и вызывается процедура поиска первого максимального элемента.

Причем процедура эта вызывается оператором

```
Search(n, m, A, Max, iMax, jMax);
```

А вот и она сама:

```
Procedure Search(v, w: Integer; X: TMyArray;
Var smax, simax, sjmax: Integer);
  Var i, j: Integer;
  Begin
   smax := X[1, 1];
   simax := 1; sjmax := 1;
   For i := 1 To v Do
    For j := 1 To w Do
     If X[i, j] > smax Then Begin
      smax := X[i, j];
      simax := i;
      sjmax := j
     End
  End;
```

Если сама процедура `Search` вряд ли вызывает вопросы, то вот для вывода и исходного файла, и результата в файл OUTPUT.TXT наших знаний уже не хватает. В принципе мы можем переопределить вывод и значение максимального элемента с его индексами выводить на экран. Это делается при помощи операторов

```
Assign(Output, 'Con');
Rewrite(Output);
```

поскольку `'Con'` как раз и означает, что в качестве устройства вывода выбран монитор.

Однако если бы требовалось вывести исходный массив в файл OUTPUT.TXT, то получилось бы, что массив выводится в файл, а результаты работы — на экран. Где логика?

Воспользуемся стандартной процедурой `Append`. Добавим в основную программу такой фрагмент:

Процедура `Append(Var f: Text)` открывает существующий файл для добавления в него информации. Переменная `f` является файловой переменной типа **`Text`**, которая должна быть связана с внешним файлом с помощью процедуры `Assign`. Процедура `Append` открывает существующий внешний файл, имя которого поставлено в соответствие переменной `f`. Если файл уже открыт, то сначала он закрывается, а затем открывается повторно. Указатель текущей позиции при этом устанавливается на конец данного файла, т.е. очередная запись в файл окажется самой последней.

```
Search(n, m, A, Max, iMax, jMax);
Append(Output);
WriteLn(Max: 5, iMax: 5, jMax: 5);
```

Теперь после исходного файла, предварительно выведенного в массив, туда же будут дописаны результаты работы нашей процедуры.

Измените процедуру `Search` для поиска последнего максимального элемента, для поиска всех максимальных элементов и их индексов.

2. Сформируйте одномерный массив, каждый элемент которого равен сумме отрицательных элементов соответствующей строки заданной целочисленной матрицы.

Подсказка. Задача требует введения одномерного массива, поэтому приведем описание данных, процедуры подсчета суммы отрицательных элементов в строке и добавления данных в выходной файл, а также основную программу.

```
{$R+}
Program My22_2;
 Const MaxN = 10; MaxM = 8;
 Type TMyArray = Array[1..MaxN, 1..MaxM] Of Integer;
      OMyArray = Array[1..MaxN] Of Integer;
 Var A: TMyArray;
     Sum: OMyArray;
     n, m: Integer;
Procedure SumNeg(v, w: Integer; X: TMyArray; Var Ssum: OMyArray);
  Var i, j: Integer;
  Begin
   For i := 1 To v Do Begin
    Ssum[i] := 0;
    For j := 1 To w Do
     If X[i, j] < 0 Then Inc(Ssum[i], X[i, j])
   End
  End;
 Procedure OPrint(v: Integer; OSum: OMyArray);
  Var i: Integer;
  Begin
   Append(Output);
   For i := 1 To v Do Write(OSum[i]: 4);
   WriteLn
  End;
 Begin{Основная программа.}
  TInit(n, m, A);
  TPrint(n, m, A);
  SumNeg(n, m, A, Sum);
  OPrint(n, Sum)
End.
```

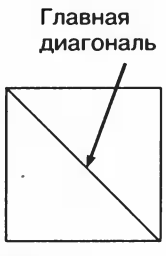
3. Рассмотрим функцию, которая определяет, есть ли в массиве элемент, равный 0:

```
Function SNeg(v, w: Integer; X: TMyArray): Boolean;
  Var i, j: Integer;
      pp: Boolean;
  Begin
   pp := False;
   For i := 1 To v Do
     For j := 1 To w Do
     If X[i, j] = 0 Then pp := True;
   Sneg := pp
  End;
```

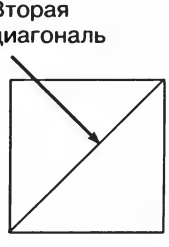
Фрагмент из основной программы:

```
...
  Append(Output);
  If SNeg(n, m, A) Then WriteLn('Yes') Else WriteLn('No');
...
```

Очевидно, что у функции SNeg есть серьезный недостаток. А именно: пусть первый элемент массива $X[1, 1]$ равен нулю. Просмотр массива продолжается, несмотря на его бессмысленность. Как известно, "задача определяет тип используемых конструкций повторения", поэтому используйте операторы **While** или **Repeat...Until** в функции SNeg и модернизируйте ее, устранив ненужный просмотр массива.



4. Определите, является ли данный квадратный массив симметричным относительно своей главной диагонали. Элементы на главной диагонали характеризуются совпадением значений индексов — $i = j$, на второй диагонали — $i = N - j + 1$, где N — размерность массива.




```
Function Sim(v: Integer; X: TMyArray): Boolean;
  Var i, j: Integer;
      pp: Boolean;
  Begin
   pp := True;
   For i := 1 To v - 1 Do
     For j := i + 1 To v Do
     If X[i, j] <> X[j, i] Then pp := False;
   Sim := pp
  End;
```

Измените функцию так, чтобы симметричность массива проверялась относительно второй диагонали. В предыдущем примере был рассмотрен недостаток функции SNeg. Он присущ и этой функции. Устраните его.

5. В массиве A размерностью $n \times m$ к элементам четных столбцов прибавьте элемент первого столбца соответствующей строки.

Решение:

```
Procedure Change(v, w: Integer; Var X: TMyArray);
Var i, j: Integer;
Begin
 For i := 1 To v Do
 For j := 1 To w div 2 Do Inc(X[i, 2 * j], X[i, 1])
End;
```

Особенностью решения задач этого типа является возможность изменения элемента, который должен использоваться в обработке. Например, изменяем только элементы нечетных столбцов. Модификация $X[i, 2 \cdot j - 1]$ приведет к тому, что элементы 3-го, 5-го и т.д. столбцов увеличатся на удвоенное значение элементов первого столбца. Проведите корректное изменение процедуры Change.

6. Заполните массив A размером $n \times m$ следующим образом (например, $n = 6$ и $m = 8$):

1	2	3	4	5	6	7	8
16	15	14	13	12	11	10	9
17	18	19	20	21	22	23	24
32	31	30	29	28	27	26	25
33	34	35	36	37	38	39	40
48	47	46	45	44	43	42	41

То есть заполнение производится "змейкой".

Правило заполнения.

Если номер строки — нечетное число,
то $A[i, j] = (i - 1) \cdot m + j$,
иначе — $A[i, j] = i \cdot m - j + 1$. Соответствующая процедура выглядит так:

```
Procedure Fill(v, w: Integer; Var X: TMyArray);
 Var i, j: Integer;
 Begin
  For i := 1 To v Do
   For j := 1 To w Do
    If i mod 2 = 1 Then X[i, j] := (i - 1) * w + j Else X[i, j] := i * w - j + 1
 End;
```

А теперь заполните массив по следующим правилам.

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

1	0	2	0	3	0	4
0	5	0	6	0	7	0
8	0	9	0	10	0	11
0	12	0	13	0	14	0

1	3	4	10	11	21
2	5	9	12	20	22
6	8	13	19	23	30
7	14	18	24	29	31
15	17	25	28	32	35
16	26	27	33	34	36

1	12	13	24	25	36
2	11	14	23	26	35
3	10	15	22	27	34
4	9	16	21	28	33
5	8	17	20	29	32
6	7	18	19	30	31

7. Составьте программу, запрашивающую координаты ферзя на шахматной доске и сообщающую, какие поля доски находятся под боем этой фигуры.

Заметим, что шахматную доску удобно представить в виде двухмерного массива размером 8 × 8. Координаты ферзя определяются двумя числами (номер строки и номер столбца), но в шахматах принято вводить букву и число. Буквой обозначим номер строки, а числом — номер столбца. Поэтому не будем отступать от традиций и введем координаты именно таким образом. В программе сделаем проверку корректности ввода, и если все правильно, то переведем букву в соответствующее ей число ('a' — 1, 'b' — 2, 'c' — 3, 'd' — 4, 'e' — 5, 'f' — 6, 'g' — 7, 'h' — 8).

Для решения задачи используем ряд свойств шахматной доски. Все ее диагонали делятся на восходящие и нисходящие.

При этом на каждой диагонали выполняется свойство:

- для клеток любой восходящей диагонали сумма номера строки и номера столбца постоянна ($i + j = Const$);
- для клеток любой нисходящей диагонали разность номера строки и номера столбца постоянна ($i - j = Const$).

Проверьте, что для восходящих диагоналей сумма индексов изменяется от 2 до 16, а для нисходящих — разность от −7 до 7.

А теперь приведем решение, основанное на этом факте:

```
Program My22_3;
Const MaxN = 8; MaxM = 8;
Type TMyArray = Array[1.MaxN, 1..MaxM] Of Integer;
Procedure TInit(v, k, l: Integer; Var X: TMyArray);
 Var i, j: Integer;
 Begin
  For i := 1 To v Do
   For j := 1 To v Do
    If (i = k) or (j = l) or (i + j = k + l) or (i - j = k - l) Then X[i, j] := 1
     Else X[i, j] := 0; {1 — клетка под боем, 0 — нет.}
  X[k, l] := 2 {В этой клетке находится ферзь.}
End;
Procedure Solve;
 Var  A: TMyArray;
c: Char; str, stl: Integer;
 Begin
  Assign(Input, 'Input.Txt'); Reset(Input);
  Assign(Output, 'Output.Txt'); ReWrite(Output);
  ReadLn(c, stl); {Координаты ферзя записаны в файле Input.Txt.}
  If (c < 'a') or (c > 'h') or (stl < 1) or (stl > MaxN)
   Then WriteLn ('Некорректный ввод.')
   Else Begin
    str := Ord(ch) — Ord('a') + 1; {Преобразуем символ в номер строки.}
    TInit(MaxN, str, stl, A);
    TPrint(MaxN, MaxM, A) {Процедура совпадает с ранее рассмотренной.}
   End;
  Close(Input); Close(Output)
End;
```

Основная программа состоит из одной строки — вызова процедуры Solve. Измените решение так, чтобы определялись координаты клеток доски, находящиеся под боем хотя бы одной из фигур — ферзя или/и ладьи.

Задания для самостоятельной работы

1. Дан двухмерный массив. Найти сумму и количество элементов в каждом столбце:

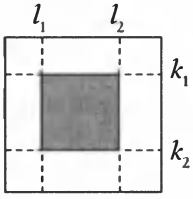
- кратных k_1 или k_2;
- попадающих в интервал от A до B;
- являющихся простыми числами;
- положительных и лежащих выше главной диагонали.

2. Дан двухмерный массив. Найти:

- сумму элементов в строках с k_1-й по k_2-ю;
- номера всех максимальных элементов;
- номера первых отрицательных элементов каждой строки (столбца);
- номера последних отрицательных элементов каждой строки (столбца);
- количество элементов в каждой строке, больших (меньших) среднего арифметического элементов данной строки;
- номера первой пары неравных элементов в каждой строке.

3. Даны два квадратных массива — A и B. Вывести тот из них, у которого след меньше (следом называется сумма элементов главной диагонали).

4. Дан двухмерный массив. Определить:
- есть ли в данном массиве отрицательный элемент;
- есть ли два одинаковых элемента;
- есть ли данное число A среди элементов массива;
- есть ли в заштрихованной области массива элемент, равный A (массив имеет размерность $n \times n$):



5. Дан двухмерный массив. Определить, есть ли в данном массиве строка (столбец):
- состоящая только из положительных элементов;
- состоящая только из положительных или нулевых элементов;
- состоящая только из элементов, больших числа A;
- состоящая только из элементов, принадлежащих промежутку от A до B.

6. Дан двухмерный массив. Выполнить с ним следующие преобразования:
- в каждой строке сменить знак максимального по модулю элемента на противоположный;
- последний отрицательный элемент каждого столбца заменить нулем;
- положительные элементы умножить на первый элемент соответствующей строки, а отрицательные — на последний;
- заменить все элементы строки с номером k и столбца с номером l на противоположные по знаку;
- к элементам столбца с номером k_1 прибавить элементы столбца k_2.

7. Написать программу, определяющую координаты полей шахматной доски, находящиеся под боем коня.

8. Ввести координаты ферзя и коня и определить:
- бьют ли фигуры друг друга;
- если конь ходит первым, то бьет ли он ферзя;
- бьет ли ферзь коня, если первый ход за ферзем.

9. Составить программу заполнения и вывода в файл таблиц сложения и умножения цифр в заданной (произвольной) системе счисления. Для десятичной системы счисления они имеют вид:

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

*	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	0	9	18	27	36	45	54	63	72	81

10. Даны два двухмерных массива одинаковой размерности. Создать третий массив той же размерности, каждый элемент которого равен:
- сумме соответствующих элементов первых двух;
- 1 (***True***), если соответствующие элементы массивов имеют одинаковый знак, иначе элемент равен 0 (***False***).

11. Дан двухмерный массив. Сформировать одномерный массив, каждый элемент которого равен:
- произведению четных, положительных элементов соответствующего столбца;
- количеству элементов соответствующей строки, больших данного числа;
- наибольшему по модулю элементу соответствующего столбца;
- количеству отрицательных элементов, кратных 3 или 5, соответствующей строки;
- первому четному элементу соответствующего столбца, а если такого нет, то нулю.

12. Дан двухмерный массив. Определить:
- есть ли в данном массиве строка, в которой ровно два отрицательных элемента;
- есть ли в данном массиве столбец, в котором имеются одинаковые элементы;

- есть ли в данном массиве строка, в которой имеются два максимальных элемента всего массива;
- есть ли в данном массиве столбец, в котором равное количество положительных и отрицательных элементов;
- есть ли в данном массиве строка, содержащая больше положительных элементов, чем отрицательных.

13. Написать программы, получающие следующие двухмерные массивы:

1	0	0	0	0	0	1
0	1	0	0	0	1	0
0	0	1	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
1	0	0	0	0	0	1

1	1	1	1	1	1	1
0	1	1	1	1	1	0
0	0	1	1	1	0	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
1	1	1	1	1	1	1

1	0	0	1	0	0	1
0	1	0	1	0	1	0
0	0	1	1	1	0	0
1	1	1	1	1	1	1
0	0	1	1	1	0	0
0	1	0	1	0	1	0
1	0	0	1	0	0	1

1	1	1	1	1	1
1	2	3	4	5	6
1	3	6	10	15	21
1	4	10	20	35	56
1	5	15	35	70	126
1	6	21	56	126	252

1	2	3	4	5	6
2	3	4	5	6	1
3	4	5	6	1	2
4	5	6	1	2	3
5	6	1	2	3	4
6	1	2	3	4	5

14. Определить, является ли массив ($N \times N$) магическим квадратом (значения элементов массива принимают все значения из интервала от 1 до $N \times N$), то есть суммы по всем горизонталям, вертикалям и двум диагоналям должны совпадать.

Примеры магических квадратов

1	15	24	8	17
9	18	2	11	25
12	21	10	19	3
20	4	13	22	6
23	7	16	5	14

38	14	32	1	26	44	20
5	23	48	17	42	11	29
21	39	8	33	2	27	45
30	6	24	49	18	36	12
46	15	40	9	34	3	28
13	31	7	25	43	19	37
22	47	16	41	10	35	4

Поскольку сумма элементов магического квадрата равна $1 + 2 + 3 + ... + N^2 = N^2 \cdot (N^2 + 1) / 2$, то соответствующие суммы равны $N \cdot (N^2 + 1) / 2$. Так, при $N = 5$ в каждой строке, столбце и на главных диагоналях сумма элементов равна $5 \cdot (5^2 + 1) / 2 = 65$.

15. Дан двухмерный массив ($N \times N$). Зеркально отобразить его элементы относительно:

- горизонтальной оси симметрии;
- вертикальной оси симметрии;
- главной диагонали;
- побочной диагонали.

Занятие 23. Двухмерные массивы. Вставка и удаление строк и столбцов

План занятия

- изучение различных методов вставки и удаления элементов двухмерного массива;
- выполнение самостоятельной работы.

Экспериментальный раздел занятия

Рассмотрим следующие задачи:

1. Требуется в двухмерный массив вставить строку из нулей после строки с номером *t*. Для решения этой задачи необходимо:

- первые *t* строк оставить без изменения;
- все строки после строки с номером *t* сдвинуть на одну;
- элементам строки $t + 1$ присвоить заданное значение (ноль).

В процедуре ввода данных, кроме ввода размеров массива, следует предусмотреть ввод номера строки — `ReadLn(n, m, t)`. Ключевая процедура вставки строки имеет вид:

```
Procedure InsertStr(v, w, r: Integer; Var X: TMyArray);
   Var i, j: Integer;
  Begin
   For i := v DownTo r + 1 Do For j := 1 To w Do X[i + 1, j] := X[i, j];
   For j := 1 To w Do X[r + 1, j] := 0
  End;
```

Измените процедуру так, чтобы новая строка из нулей вставлялась перед строкой с заданным номером.

2. В двухмерный массив необходимо вставить новые строки из нулей после всех строк, содержащих максимальный элемент массива. На этот раз, чтобы напомнить о стандартно используемой схеме решения задачи, приведем полный текст решения:

```
{$R+}
Program My23_1;
 Const MaxN = 8; MaxM = 9;
 Type TMyArray=Array[1..2 * MaxN, 1..MaxM] Of Integer;
{Резервируем 2*MaxN строк на тот случай, если максимальный элемент массива есть в каждой строке.}
 Var A: TMyArray;
     n, m: Integer;
 Procedure TInit(Var v, w: Integer; Var X: TMyArray);
  Var i, j: Integer;
  Begin
   Assign(Input, 'Input.Txt'); Reset(Input);
   ReadLn(v, w);
   For i := 1 To v Do
    For j := 1 To w Do Read(X[i, j]);
   Close(Input)
  End;
 Procedure TPrint(v, w: Integer; X: TMyArray);
  Var i, j: Integer;
  Begin
   Assign(Output, 'Output.Txt'); ReWrite(Output);
   For i := 1 To v Do Begin
    For j := 1 To w Do Write(X[i, j]: 4); WriteLn
   End;
   Close(Output)
  End;
 Procedure InsertStr(v, w, r: Integer; Var X: TMyArray);
   Var i, j: Integer;
  Begin
   For i := v DownTo r + 1 Do
    For j := 1 To w Do X[i + 1, j] := X[i, j];
   For j := 1 To w Do X[r + 1, j] := 0
  End;
 Function TMax(v, w: Integer; X: TMyArray): Integer; {Поиск максимального элемента.}
  Var i, j, max: Integer;
  Begin
   max := X[1, 1];
```

```
  For i := 1 To v Do
   For j := 1 To w Do If X[i, j] > max Then max := X[i, j];
  TMax:=max
End;
Procedure Solve(Var v: Integer; w: Integer; Var X: TMyArray);
 Var i, j, smax: Integer;
 Begin
  smax := TMax(v, w, X); {Находим максимальный элемент.}
  i := 1;
  While i <= v Do Begin {Кстати, а почему используется этот тип цикла?}
   j := 1;
   While (j <= w) and (X[i, j] <> smax) Do Inc(j);
   If j <> w + 1 Then Begin{В строке есть максимальный элемент.}
    InsertStr(v, w, i, X); {Вставляем строку.}
    Inc(v); {Увеличиваем количество строк.}
    Inc(i, 2){Пропускаем вставленную строку.}
   End
   Else Inc(i) {Обычное изменение индекса.}
  End
 End;
Begin {В основной программе только вызовы процедур — крупных "кирпичиков".}
 TInit(n, m, A);
 Solve(n, m, A);
 TPrint(n, m, A)
End.
```

Измените решение для вставки нулевых строк перед строками, содержащими максимальный элемент массива.

3. Удалите строку с номером t из данного массива.

Для того чтобы удалить строку с номером t, необходимо сдвинуть все строки, начиная с данной, на одну вверх. Последнюю строку можно при желании "обнулить" и в дальнейшем уже не рассматривать:

```
Procedure DelStr(v, w, r: Integer; Var X: TMyArray);
  Var i, j: Integer;
 Begin
  For i := r To v - 1 Do
   For j := 1 To w Do X[i, j] := X[i + 1, j];
  For j := 1 To w Do X[v, j] := 0
 End;
```

4. Удалите строки, содержащие максимальный элемент массива. Задача решается следующей процедурой:

```
Procedure Solve(Var v: Integer; w: Integer; Var X: TMyArray);
 Var i, j, smax: Integer;
 Begin
  smax := TMax(v, w, X);
  i := 1;
  While i <= v Do Begin
   j := 1;
   While (j <= w) and (X[i, j] <> smax) Do Inc(j);
   If j <> w + 1 Then Begin
    DelStr(v, w, i, X);
    Dec(v)
   End
   Else Inc(i)
  End
 End;
```

Сравните эту процедуру с процедурой Solve, рассмотренной ранее при вставке строк, и объясните различия.

5. Поменяйте местами элементы столбцов с номерами l_1 и l_2.

С процедурой Swap мы уже встречались. Напомним ее.

```
Procedure Swap(Var x, y: Integer);
 Var z: Integer;
 Begin
  z := x; x := y; y := z
 End;
```

Написание процедуры, решающей задачу, не вызывает сложностей.

```
Procedure Change(v, st1, st2: Integer; Var X: TmyArray);
 Var i: Integer;
 Begin
  For i := 1 To v Do Swap(X[i, st1], X[i, st2])
 End;
```

6. Удалите все строки и столбцы, содержащие максимальный элемент массива.

2	7	9	3	4
5	6	9	0	1
2	1	9	5	8
7	6	9	4	3
5	6	9	2	1

Вначале рассмотрим пример. Пусть максимальный элемент равен 9. Если просматривать этот массив сначала по строкам, а затем по столбцам, по ходу дела удаляя те из них, которые содержат максимальный элемент, то результат окажется верным. При просмотре же вначале по столбцам, а затем по строкам исключается только третий столбец. Значит, надо вначале запомнить номера строк и столбцов, содержащих максимальный элемент массива, а лишь затем их удалять.

Пусть в процессе обхода начального массива мы получили номера строк и столбцов, содержащих максимальный элемент. Они могут располагаться, например, в массивах NumStr и NumStl. Пусть для примера они имеют вид: (1, 2, 3, 4, 5) и (3, 3, 3, 3, 3). Очевидно, что пять раз удалять третий столбец явно не имеет смысла, поэтому необходимо из полученных одномерных массивов предварительно удалить повторяющиеся элементы, т.е. во втором массиве оставить одну 3.

Однако этим не ограничиваются возникающие сложности. После удаления первой строки надо удалять вторую, но она в нашей матрице уже стала первой. Значит, необходимо после каждого удаления изменять номера строк (столбцов) в массиве NumStr, уменьшая на единицу те номера, которые больше номера удаленной строки. Ниже по тексту приведена только процедура Solve, остальные части программы совпадают с ранее рассмотренными.

```
Procedure Solve(Var v, w: Integer; Var X: TMyArray);
  Type OMyArray = Array[1..MaxN] Of Integer;
  {Определяем одномерный массив для хранения номеров удаляемых элементов. Константа MaxN берется
   при условии, что количество строк в матрице всегда больше количества столбцов.}
  Var i, smax: Integer;
      NumStr, NumStl: OMyArray; {Номера удаляемых строк и столбцов.}
      ykStr, ykStl: Integer; {Количество элементов в массивах NumStr и NumStl.}
  Procedure FNum(Var q, r: Integer; Var Y, Z: OMyArray);
 {Формирование массивов с номерами удаляемых строк и столбцов.}
   Var i, j: Integer;
   Begin
    q := 0; r := 0;
    For i := 1 To v Do
     For j := 1 To w Do
      If X[i, j] = smax Then Begin
       Inc(q); Y[q] := i;
       Inc(r); Z[r] := j
      End
   End;
  Procedure Sz(Var t: Integer; Var X: OMyArray); {Удаление из массива повторяющихся элементов.}
   Var i, j, l: Integer;
   Begin
    i := 1;
    While i < t Do Begin
     j := i + 1;
     While j <= t Do
      If X[i] = X[j] Then Begin
        For l := j To t - 1 Do X[l] := X[l + 1];
        X[t] := 0;
        Dec(t)
       End
       Else Inc(j);
     Inc(i)
    End
   End;
  Procedure Change(t, a: Integer; Var X: OMyArray);
 {Уменьшение на единицу значений элементов массива, превышающих заданную величину.}
  Var i: Integer;
```

```
 Begin
  For i := 1 To t Do
   If X[i] > a Then Dec(X[i])
 End;
Begin {Текст основной процедуры.}
 FillChar(NumStr, SizeOf(NumStr), 0); {Инициализация массивов.}
 FillChar(NumStl, SizeOf(NumStl), 0);
 smax := TMax(v, w, X); {Находим максимальное значение — текст функции не приводится.}
 FNum(ykStr, ykStl, NumStr, NumStl); {Определяем номера удаляемых строк и столбцов.}
 Sz(ykStr, NumStr); {Удаляем повторяющиеся элементы.}
 Sz(ykStl, NumStl);
 i := 1;
 While i <= ykStr  Do Begin
  DelStr(v, w, NumStr[i], X); {Удаляем строку.}
  Change(ykStr, NumStr[i], NumStr); {Корректируем массив номеров строк.}
  Dec(v);
  Inc(i)
 End;
  i := 1;
  While i <= ykStl  Do Begin
    DelStl(v, w, NumStl[i], X); {Удаляем столбец.}
    Change(ykStl, NumStl[i], NumStl); {Корректируем массив номеров столбцов.}
    Dec(w);
    Inc(i)
   End
End;
```

О технологии написания. Напоминаем, что процесс разработки процедуры `Solve` заключается в написании ее "тела" с заголовками составляющих частей. Программа компилируется, сохраняется, и только затем дописываются составляющие части. В процедуре `Solve` есть два практически совпадающих фрагмента (циклы по i). Исключите эти повторения.

Задания для самостоятельной работы

1. Вставить:
- первую строку после строки, в которой находится первый встреченный минимальный элемент;
- второй столбец после первого столбца, в котором все элементы положительны;
- нулевую строку и нулевой столбец перед строкой и столбцом, в которых находится первый минимальный элемент;
- последнюю строку после всех строк, в которых есть заданное число A;
- второй столбец перед всеми столбцами, в которых нет отрицательных элементов;
- первую строку перед всеми строками, в которых есть 0, и первый столбец после всех столбцов, в которых есть отрицательные элементы;
- столбец из нулей после столбцов с минимальными элементами;
- первую строку между средними строками;
- строку из нулей перед всеми строками, первый элемент которых делится на 3.

2. Удалить:
- столбцы, в которых есть минимальный элемент;
- строку с номером k и столбец с номером l;
- все столбцы, в которых нет нулевого элемента;
- все строки и столбцы, на пересечении которых стоят отрицательные элементы;
- среднюю строку (строки);
- все столбцы, в которых первый элемент больше последнего;
- столбец, в котором находится первый четный отрицательный элемент;
- все столбцы, в которых первый элемент больше заданного числа A.

3. Заменить:
- минимальный элемент в каждой строке на противоположный по знаку;
- все элементы первых трех столбцов на их квадраты;
- все симметричные элементы квадратной матрицы на нули.

4. Поменять местами:
- средние столбцы;
- средние строки с первой и последней;
- средние столбцы со вторым и предпоследним;

- средние строки;
- первый максимальный и последний минимальный элементы;
- в каждой строке первый элемент и максимальный по модулю;
- в каждой строке первый отрицательный и последний положительный;
- первую строку и строку, в которой находится первый нулевой элемент;
- вторую и предпоследнюю строки;
- первую строку с последней строкой, вторую — с предпоследней и так далее;
- столбцы так, чтобы в конечном итоге они имели следующий порядок:
- последний, предпоследний, ..., второй, первый;
- первый, последний, второй, предпоследний, третий, ...

5. Найти максимальный элемент массива, встречающийся более одного раза.

6. Расстоянием между строками назовем сумму произведений соответствующих элементов строк. Найти две строки с максимальным расстоянием.

7. Дан массив размерностью $N \times N$, N — нечетное число. Вывести элементы массива при обходе его по спирали начиная с центра.

8. Для заданного целочисленного массива $(N \times N)$ найти максимум среди сумм элементов диагоналей, параллельных главной диагонали.

9. Для заданного целочисленного массива $(N \times N)$ найти минимум среди сумм элементов диагоналей, параллельных побочной диагонали матрицы.

10. Значения элементов массива находятся в интервале от A до B. Две строки матрицы назовем похожими, если они отличаются только порядком элементов. Найти пары похожих строк.

11. Элемент массива $A[i, j]$ называют седловой точкой, если он является минимальным в строке с номером i и максимальным в столбце с номером j. Найти седловые точки.

12. Для данного массива найти такие значения k, что k-я строка совпадает с k-м столбцом.

13. Среди столбцов заданного целочисленного массива найти столбцы, состоящие только из нечетных элементов.

14. Элемент $A[i, j]$ назовем локальным минимумом, если он строго меньше всех своих соседей. Соседями являются элементы $A[k, l]$ с $(i - 1 \le k \le i + 1)$, $(j - 1 \le l \le j + 1)$, $(k, l) \ne (i, j)$. Найти количество локальных минимумов для заданного массива.

Найти максимум среди локальных минимумов.

15. Решить задачу поиска числа X в двухмерном массиве $A[1..N, 1..M]$ (элементы в строках и столбцах упорядочены) за время, пропорциональное $O(N + M)$.

Занятие 24. Несколько задач на технику работы с двухмерными массивами

План занятия

- разбор и модификация программ: умножения матриц; решения системы линейных уравнений; приведения матрицы к "блочному" виду; сортировки элементов строк матрицы и поиска суммы минимальных элементов в строках и столбцах матрицы;
- выполнение самостоятельной работы.

Экспериментальный раздел занятия

1. Требуется написать программу вычисления произведения двух целочисленных матриц $A[n, m]$ и $B[m, t]$. Элементы результирующей, также целочисленной, матрицы $C[n, t]$ определяются по следующей формуле:

$$C[i, j] = A[i, 1] \cdot B[1, j] + A[i, 2] \cdot B[2, j] + \ldots + A[i, m] \cdot B[m, j],$$

где (n, m), (m, t), (n, t) — размерности матриц A, B, C соответственно. При выполнении умножения количество столбцов в первой матрице равно количеству строк во второй.

Пример

A			B		C	
1	4	2	3	4	$1\cdot3+4\cdot2+2\cdot1=13$	$1\cdot4+4\cdot0+2\cdot2=8$
2	1	5	2	0	$2\cdot3+1\cdot2+5\cdot1=13$	$2\cdot4+1\cdot0+5\cdot2=18$
			1	2		

Приведем полный текст решения. В файл OUTPUT.TXT записываются массивы A, B и результат — массив C. Напоминаем, что процедуры TInit, TPrint используются для ввода и вывода двухмерных массивов любой размерности.

```
{$R+}
Program My24_1;
 Const MaxN = 8; MaxM = 8;
 Type TMyArray = Array[1..MaxN, 1..MaxM] Of Integer;
 Var A, B, C: TMyArray;
     n, m, t: Integer;
 Procedure Init;
  Begin
   Assign(Input, 'Input.Txt'); Reset(Input);
   Assign(Output, 'Output.Txt'); ReWrite(Output)
  End;
 Procedure TInit(Var v, w: Integer; Var X: TMyArray);
  Var i, j: Integer;
  Begin
   ReadLn(v, w);
   For i := 1 To v Do
    For j := 1 To w Do Read(X[i, j])
  End;
 Procedure TPrint(v, w: Integer; X: TMyArray);
  Var i, j: Integer;
  Begin
   Append(Output);
   For i := 1 To v Do Begin
    For j := 1 To w Do Write(X[i, j]: 4);
    WriteLn
   End;
   WriteLn
  End;
 Procedure Solve(v, w, r: Integer; X, Y: TMyArray; Var Z: TMyArray);
   Var i, j, t: Integer;
  Begin
   For i := 1 To v Do
    For t := 1 To r Do Begin Z[i, t] := 0;
     For j := 1 To w Do
     Z[i, t] := Z[i, t] + X[i, j] * Y[j, t]
    End
```

```
 End;
Procedure Done;
 Begin  Close(Input); Close(Output) End;
Begin
 Init;
 TInit(n, m, A); TPrint(n, m, A);
 TInit(m, t, B); TPrint(m, t, B);
 Solve(n, m, t, A, B, C);
 TPrint(n, t, C);
 Done
End.
```

Пусть A — квадратная матрица. Измените программу так, чтобы вычислялось выражение $A + A \cdot A + A \cdot A \cdot A$.

2. Решение системы линейных уравнений методом Гаусса. Система из n линейных алгебраических уравнений с n неизвестными имеет вид:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_{1n} &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_{2n} &= b_2 \\ \ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_{nn} &= b_n \end{aligned}$$

Решением системы называются такие числа $x_1 = c_1$, $x_2 = c_2$, ..., $x_n = c_n$, которые, будучи подставленными в любое уравнение системы, превращают его в верное числовое равенство. Не вдаваясь в теорию, мы будем предполагать, что наша система имеет единственное решение. В этом случае можно использовать схему Гаусса, сутью которой является последовательное исключение неизвестных из уравнений:

— сначала из первого уравнения выражается x_1 через другие неизвестные и полученное значение подставляется в оставшиеся уравнения;

— затем из второго уравнения выражается x_2...

— и так далее, пока последнее уравнение не превратится в уравнение с одним неизвестным.

Данная процедура носит название *прямого хода*. После ее применения исходная система уравнений преобразуется в равносильную (имеющую то же решение) треугольную:

$$\begin{aligned} x_1 + \alpha_{12}x_2 + \alpha_{13}x_3 + \ldots + \alpha_{1n}x_n &= \beta_1 \\ x_2 + \alpha_{23}x_3 + \ldots + \alpha_{2n}x_n &= \beta_2 \\ \ldots & \\ x_n &= \beta_n \end{aligned}$$

Из этой системы последовательно находятся значения всех неизвестных x_n, x_{n-1}, ..., x_1 (*обратный ход* метода): значение x_n подставляется во все уравнения, затем полученное значение x_{n-1} — и т.д. до первого уравнения, из которого находится значение x_1.

Прямой ход заключается в следующей процедуре:

1. Коэффициенты первого уравнения делятся на a_{11}, получим:

$$x_1 + a_{12} / a_{11}x_2 + \ldots + a_{1n} / a_{11}x_n = b_1 / a_{11} \text{ или (обозначив частные по-другому)}$$
$$x_1 + \alpha_{12}x_2 + \alpha_{13}x_3 + \ldots + \alpha_{1n}x_n = \beta_1.$$

   Если $a_{11} = 0$, то в первом столбце ищется ненулевой коэффициент, а затем переставляются соответствующие строки. Существование такого коэффициента следует из единственности решения системы.
2. Пользуясь полученным результатом, неизвестное x_1 исключается из других уравнений системы. Для этого из каждого уравнения вычитается первое, предварительно умноженное на соответствующий коэффициент при x_1.
3. После этого первое уравнение оставляют в покое, а из оставшихся аналогично исключают x_2.

Точно так же исключаются и все оставшиеся неизвестные, кроме последнего.

Описанный алгоритм реализован в следующей программе:

```
{$R+}
Program My24_2;
 Const MaxN = 10; MaxM = MaxN+1;
Type TMyArray = Array[1..MaxN, 1..MaxM] Of Real;
     OMyArray = Array[1..MaxN] Of Real;
 Var A: TMyArray;
     X: OMyArray;
     n: Integer;
 Procedure TInit(Var v: Integer; Var X: TMyArray); {Ввод коэффициентов системы уравнений.}
  Var i, j: Integer;
  Begin
   Assign(Input, 'Input.Txt'); Reset(Input);
   ReadLn(v);
   For i := 1 To v Do
    For j := 1 To v + 1 Do Read(X[i, j]);
```

```
  Close(Input)
 End;
Procedure TPrint(v: Integer; X: OMyArray); {Вывод решения.}
 Var i: Integer;
 Begin
  Assign(Output, 'Output.Txt'); ReWrite(Output);
  For i := 1 To v Do Write(X[i]: 6);
   WriteLn;
  Close(Output)
 End;
Function Solve(v: Integer; A: TMyArray; Var X: OMyArray): Boolean;
{Находим решение, если оно есть, — массив X.}
 Var i, j, k, l: Integer;
     w: Real;
     Pp: Boolean;
 Procedure Swap(k, l: Integer); {Переставляем строки с номерами k, l.}
  Var j: Integer; w: Real;
  Begin
   For j := 1 To v Do Begin w := A[k, j]; A[k, j] := A[l, j]; A[l, j] := w End
  End;
 Function Search(j: Integer): Integer; {Поиск ненулевого коэффициента в столбце с номером j.
                                        Поиск осуществляется в строках с номерами (j + 1) .. (v).}
  Var i: Integer;
  Begin
   i := j + 1;
   While (i <= v) and (A[i, j] = 0) Do Inc(i);
   Search := i
  End;
 Begin {Основная часть решения.}
  j := 1; Pp := False;
  While (j <= v) and not(Pp) Do Begin
   If A[j, j] = 0 Then Begin
     k := Search(j);
     If k > v Then Pp := True Else Swap(j, k)
    End;
   If not Pp Then Begin{Если есть решение.}
    w := A[j, j]; {Прямой ход. Преобразуем уравнение с номером j.}
    For k := j To v + 1 Do A[j, k] := A[j, k] / w;
    For k := j + 1 To v Do Begin {Исключаем xj из остальных уравнений.}
     w := A[k, j];
     For l := j To v + 1 Do A[k, l] := A[k, l] - A[j, l] * w
    End
   End;
   Inc(j) {Переходим к другой строке.}
  End;
  If not Pp Then Begin {Обратный ход.}
   For j := v DownTo 1 Do Begin
    X[j] := A[j, v + 1];
    For i := j - 1 DownTo 1 Do {Исключаем найденное x  из всех уравнений.}
     A[i, v + 1] := A[i, v + 1] - A[i, j] * X[j]
   End
  End;
  Solve := not Pp
 End;
Begin{Основная программа.}
 TInit(n, A); {Ввод системы уравнений.}
 If Solve(n, A, X) Then TPrint(n, X) {Если есть решение, то его поиск и вывод результата.}
End.
```

3. Дана матрица, элементы которой принимают лишь два значения — 0 и 1.

1	2	3	4	5	6
0	1	0	0	0	1
0	0	1	0	1	0
1	0	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	1
0	1	0	1	0	0

2	6	3	5	1	4
1	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
1	0	1	0	0	1
0	1	0	0	1	1
0	0	0	0	0	1

Требуется привести ее к "блочному" по столбцам виду, т.е. переставить столбцы так, чтобы вначале шли столбцы с единицей в первой строке, затем столбцы, у которых в первой строке нули, а во второй — единицы (если такие столбцы есть) и т.д. Пример приведен на рисунке (в первой строке указаны исходные номера столбцов).

Суть решения заключается в просмотре очередной строки одновременно слева направо и справа налево. Если при просмотре слева направо найден столбец с 0, а справа налево — с 1, то эти столбцы переставляются. При переходе к следующей строке просмотр слева направо следует продолжить уже со столбца, на котором закончился просмотр при обработке предыдущей строки.

А вот и соответствующая программа:

```
Procedure Solve(v: Integer; Var X: TMyArray);
  Var i, w: Integer;
  Procedure Swap(q, r: Integer);
   Var i, w: Integer;
   Begin
    For i := 1 To v Do Begin w := X[i, q]; X[i, q] := X[i, r]; X[i, r] := w End
   End;
Procedure Change_St(i: Integer; Var k: Integer);
{Обработка строки с номером i. Параметр k – номер столбца, на котором заканчиваются действия в строке i.}
   Var j: Integer;
   Begin
    j := v;
    While k < j Do Begin
     If X[i, k] = 1 Then Inc(k){Поиск единицы.}
      Else If X[i, j] = 0 Then Dec(j){Поиск нуля.}
           Else Begin Swap(k, j); Inc(k); Dec(j) End
    End;
    If X[i, k] = 1 Then Inc(k) {Подумайте: зачем нужен этот оператор?}
   End;
  Begin
   w := 1;
   i := 1;
   While (i <= v) and (w<v) Do Begin
   {Пока не просмотрены все строки и не исчерпаны все столбцы, выполняем действия из тела цикла.}
    Change_St(i, w); {Обрабатываем строку с номером i.}
    Inc(i) {Изменяем номер строки.}
   End
  End;
```

Проверьте, соответствует ли эта программа приведенному выше алгоритму.

В некоторых задачах необходимо проверить, является ли матрица "блочной", и по столбцам, и по строкам. Так, следующий рисунок иллюстрирует приведение матрицы к "двойному" блочному виду (сверху и слева указаны номера исходных столбцов и строк):

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	1
3	1	0	1	0	0	0
4	0	0	0	0	1	1
5	0	1	0	1	0	0
6	1	0	1	0	0	0

	4	2	6	5	1	3
1	1	1	0	0	0	0
2	0	0	1	1	0	0
3	0	0	0	0	1	1
4	0	0	1	1	0	0
5	1	1	0	0	0	0
6	0	0	0	0	1	1

	4	2	6	5	1	3
1	1	1	0	0	0	0
5	1	1	0	0	0	0
2	0	0	1	1	0	0
4	0	0	1	1	0	0
3	0	0	0	0	1	1
6	0	0	0	0	1	1

Напишите программу, проверяющую, имеет ли матрица "блочный" вид (по строкам, столбцам или одновременно и по тем, и по другим).

4. Рассмотрим следующую задачу.

A

21	12	10	43	37	25
14	52	74	12	42	13
13	10	20	15	25	11
40	21	30	10	20	31
32	22	12	41	25	40
10	20	30	40	50	60

B

3	2	1	6	5	4
4	6	1	5	2	3
2	6	1	4	3	5
4	5	2	3	6	1
3	2	5	1	6	4
1	2	3	4	5	6

Пусть дана целочисленная матрица *A*. Сформировать матрицу *B* таким образом, чтобы в каждой строке были записаны номера столбцов матрицы *A* в порядке, соответствующем неубыванию значений элементов в той же строке матрицы *A*. То есть в первой строке матрицы *B* записано, что минимальным элементом в первой строке матрицы *A* является элемент третьего столбца, далее по возрастанию идет элемент второго столбца и т.д.

Для простоты решения изменим в программе типы данных. Двухмерный массив определим через одномерные массивы (найдите место в тексте процедуры, где этот факт используется).

```
Type OMyArray = Array[1..MaxN] Of Integer;
     TMyArray = Array[1..MaxN] Of OMyArray;
```

Элементы каждой строки переписываются в дополнительный массив, а затем сортируются одним из известных методов с одновременной перестановкой соответствующих элементов в строке матрицы *B*. Начальный вид *B* приведен ниже.

B

1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6

```
Procedure Solve(v: Integer; X: TMyArray; Var Y: TMyArray);
  Var i, t, j: Integer;
      Z: OMyArray;
  Procedure Swap(Var q, r: Integer);
   Var w: Integer;
   Begin
    w := q; q := r; r := w
   End;
  Begin
   For i := 1 To v Do Begin
    Z := X[i];
    For t := v - 1 DownTo 1 Do {Сортировка.}
     For j := 1 To t Do
      If Z[j] > Z[j + 1] Then Begin
       Swap(Z[j], Z[j + 1]); Swap(Y[i, j], Y[i, j + 1])
      End
   End
  End;
```

5. Дана целочисленная матрица *A*. Требуется выполнить следующее преобразование: найти в каждой строке минимальный элемент и вычесть его из всех элементов данной строки; затем аналогичное преобразование выполнить и со всеми столбцами; при выполнении действий подсчитывать сумму минимальных элементов. На рисунках изображены исходная матрица, матрица после указанного преобразования по строкам (минимальные элементы расположены в отдельном столбце) и, наконец, матрица после второго преобразования по столбцам (минимальные элементы расположены в отдельной нижней строке).

10	6	4	8	7	14	
6	6	7	11	7	10	
4	7	5	4	3	11	
8	11	5	6	5	11	
6	7	3	5	6	7	
14	10	10	11	8	9	

6	2	0	4	3	10	**4**
0	0	1	5	2	4	**6**
1	4	2	1	0	8	**3**
3	6	0	1	0	6	**5**
3	4	0	2	3	4	**3**
6	2	2	3	0	1	**8**

Сумма минимальных элементов для приведенного примера равна 31.

6	2	0	3	3	9	**4**
0	0	1	4	2	3	**6**
1	4	2	0	0	7	**3**
3	6	0	0	0	5	**5**
3	4	0	1	3	3	**3**
6	2	2	2	0	0	**8**
0	**0**	**0**	**1**	**0**	**1**	

```
Function Solve(v: Integer; Var X: TMyArray): Integer;
{Функция возвращает сумму минимальных элементов.}
  Var s, i, w, j: Integer;
  Function MinStr(t: Integer): Integer; {Поиск минимального элемента в строке.}
  Var j, min: Integer;
  Begin
   min := X[t, 1];
   For j := 1 To v Do If X[t, j] < min Then min := X[t, j];
   MinStr := min
  End;
  Function MinStl(t: Integer): Integer; {Поиск минимального элемента в столбце.}
  Var i, min: Integer;
  Begin
   min := X[1, t];
   For i := 2 To v Do If X[i, t] < min Then min := X[i, t];
   MinStl := min
  End;
  Procedure SubStr(i, k: Integer); {Вычитание из элементов строки с номером i значения k.}
   Var j: Integer;
   Begin
    For j := 1 To v Do X[i, j] := X[i, j] - k
   End;
  Procedure SubStl(j, k: Integer); {Вычитание из элементов столбца с номером j значения k.}
   Var i: Integer;
   Begin
    For i := 1 To v Do X[i, j] := X[i, j] - k
   End;
  Begin
   s := 0; {Искомое значение.}
   For i := 1 To v Do Begin {Преобразование по строкам.}
    w := MinStr(i);
    Inc(s, w);
    SubStr(i, w)
   End;
   For j := 1 To v Do Begin {Преобразование по столбцам.}
    w := MinStl(j);
    Inc(s, w);
    SubStl(j, w)
   End;
   Solve := s {Результат.}
  End;
```

Модифицируйте решение задачи следующим образом. Из полученной в результате двойного преобразования матрицы исключите произвольный столбец и строку, на пересечении которых получен ноль. С новой матрицей снова выполните то же самое двойное преобразование. Продолжайте этот процесс до тех пор, пока это возможно. Найдите общее значение суммы минимальных элементов, получаемой в результате этих действий.

Задания для самостоятельной работы

1. Решить системы линейных уравнений:

$$\begin{aligned} 4{,}4x_1 - 2{,}5x_2 + 19{,}2x_3 - 10{,}8x_4 &= 4{,}3 \\ 5{,}5x_1 - 9{,}3x_2 - 14{,}2x_3 + 13{,}2x_4 &= 6{,}8 \\ 7{,}1x_1 - 11{,}5x_2 + 5{,}3x_3 - 6{,}7x_4 &= -1{,}8 \\ 14{,}2x_1 + 23{,}4x_2 - 8{,}8x_3 + 5{,}3x_4 &= 7{,}2 \end{aligned}$$

$$\begin{aligned} 15{,}7x_1 + 6{,}6x_2 - 5{,}7x_3 + 11{,}5x_4 &= 12{,}4 \\ 8{,}8x_1 - 6{,}7x_2 + 5{,}5x_3 - 4{,}5x_4 &= 5{,}6 \\ 6{,}3x_1 - 5{,}7x_2 - 23{,}4x_3 + 6{,}6x_4 &= 7{,}7 \\ 5{,}6x_1 + 8{,}8x_2 - 6{,}7x_3 - 23{,}8x_4 &= 7{,}3 \end{aligned}$$

2. Напомним, что следом квадратной матрицы называется сумма элементов, расположенных на главной диагонали. Даны квадратная матрица A порядка n и натуральное число m. Вычислить следы матриц A, A^2, A^3, ..., A^m.

3. Дана квадратная матрица A порядка n. Переставить строки так, чтобы элементы в первом столбце были упорядочены по неубыванию.

4. В матрице A порядка $n \times m$ заменить нулями элементы, стоящие в строках и столбцах, где имеются нули.

5. Дана квадратная матрица A порядка n, каждый элемент которой равен 0, 1, 5 или 11. Подсчитать в ней количество четверок $A[i, j]$, $A[i+1, j]$, $A[i, j+1]$, $A[i+1, j+1]$, в каждой из которых все элементы различны.

Указание. Заметьте, что если все числа различны, то их сумма равна 17. Верно и обратное утверждение.

6. Дана квадратная матрица A порядка n, состоящая только из чисел 0 и 1. Прямоугольником назовем часть матрицы, заполненной 1. Известно, что прямоугольники не соприкасаются друг с другом. Найти количество прямоугольников. На рисунке их 4.

0	1	0	0	1	1
0	0	0	0	1	1
1	1	0	0	0	0
1	1	0	1	1	0
1	1	0	1	1	0

7. Дана квадратная матрица A порядка n, состоящая из натуральных чисел. Находится минимальный элемент A и вычитается из всех элементов матрицы. Затем строки и столбцы, содержащие нулевые элементы после вычитания, "вычеркиваются" из матрицы. Процесс продолжается до тех пор, пока не будет получено одно число. Найти суммы минимальных элементов.

8. Дана квадратная матрица A порядка n, состоящая только из чисел 0 и 1. Считаем, что столбец "покрывается" другим столбцом, если множество строк, представленных 1 в этом столбце, содержится в множестве строк, представленных 1 в другом столбце. Исключить такие столбцы из матрицы.

9. Дана квадратная матрица A порядка n, состоящая только из чисел 0 и 1. Исключить из матрицы столбцы и строки, содержащие одну-единственную 1. Если после исключения появились новые строки и столбцы, удовлетворяющие этому условию, то продолжить процесс исключения.

10. Квадратную матрицу A порядка n, состоящую только из чисел 0 и 1, назовем правильной, если в ней нет квадратов 2×2 и более, составленных только из чисел 0 или 1. Проверить, является ли A правильной. На рисунке матрица A является правильной.

1	0	1	0	1
0	1	0	1	0
1	0	1	0	1
0	1	1	0	1
1	0	0	0	1

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 типографии ОАО ПО "Пресса-1".
125865, ГСП, Москва, ул. "Правды", 24.
Тираж 6600 экз.
Срок подписания в печать по графику 27.06.2001.
Номер подписан 27.06.2001.
Заказ № 14254
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.