ОЛИМПИАДЫ ПО ИНФОРМАТИКЕ ЗАДАЧИ И РЕШЕНИЯ

ЧАСТЬ 1

Задача №1

У продавца и покупателя имеется неограниченное кол-во монет достоинством (1,2,5,10,20,50,100,200,500 к примеру). Покупатель купил товар на сумму n. Нужно найти минимальное кол-во монет, которые будут использованы при расплате. Деньги может давать как покупатель, так и продавец.

Рассмотрим кольцо многочленов над некоторым полем. Пусть I некоторый идеал в этом кольце, порожденный многочленами g_1, g_2, ..., g_n. При этом набор G={g_1, g_2, ..., g_n} называется базисом (или системой образующих) идеала I (обозначение I=<G>).

Пусть на мономах задан линейный порядок (term order), удовлетворяющий условиям: * для любого монома a, отличного от 1, a>1; * если a<b, то для любого монома c имеем ac<bc.

Примерами таких порядков являются: лексикографический (lex); общей степени, затем лексикографический (deglex); общей степени, затем обратный лексикографический (degrevlex).

У любого многочлена f однозначно определяется старший (в смысле заданного порядка) моном (leading term), который мы будем обозначать lt(f). Можно считать, что все рассматриваемые многочлены нормированы (коэффициент при старшем мономе равен 1). Если lt(f) делится на lt(g), то многочлен f можно редуцировать относительно многочлена g - результатом будет многочлен f-(lt(f)/lt(g))*g.

Многочлен f называется редуцированным относительно G, если lt(f) не делится на lt(g) ни для какого g из G. Любой многочлен f за конечное число редукций можно привести к редуцированному относительно G многочлену. Заметим, что результат такой редукции, вообще говоря, неоднозначен.

Определение Базис G идеала I называется _базисом Грёбнера_ (стандартным базисом) относительно порядка <, если в результате любой редукции элемента f идеала I к редуцированному относительно G многочлену всегда получается 0.

Каждый элемент g из G можно редуцировать относительно набора G-{g}. Результатом будет редуцированный базис, в котором старшие термы любых двух элементов не делятся один на другой.

Доказано, что

1) любой идеал обладает базисом Грёбнера относительно любого порядка;

2) редуцированный базис Грёбнера идеала I относительно порядка < определяется однозначно с точностью до порядка следования многочленов;

2') два идеала I и J равны между собой тогда и только тогда, когда их базисы Грёбнера совпадают.

Для построения базисов Грёбнера Бухбергер придумал алгоритм, который теперь носит его имя.

IK> и каким именно образом они используются ?

Пусть достоинства монет 1=c_0, c_1, ..., c_{m-1}. Любую сумму вида $(k_0-k_m)*c_0 + (k_1-k_{m+1})*c_1 + ... + (k_{m-1}-k_{2m-1})*c_{m-1}$, где все k_i, i=0..2m-1 неотрицательны, которые мы будем кодировать мономами $x_0^{k_0}x_1^{k_1}...x_{2m-1}^{k_{2m-1}}$.

Пусть I - идеал, порожденный мономами соответствующими нулевой сумме.

Утв 1. Базисом I является набор многочленов $x_0^{c_i}-x_i$, $x_1^{c_i}x_{m+i}-1$, i=0..m-1. Доказательство индукцией по m. [skipped]

Пусть G - базис Грёбнера идеала I относительно deglex. Пусть нам дана сумма s. Редуцируем многочлен x_0^s относительно G.

Утв 2. Результатом этой редукции будет многочлен $x_0^{k_0}x_1^{k_1}...x_{2m-1}^{k_{2m-1}}$, причем в каждой паре ($k_i$,$k_{m+i}$), i=0..m-1 не более одного числа отлично от нуля и $s = (k_0-k_m)*c_0 + (k_1-k_{m+1})*c_1 + ... + (k_{m-1}-k_{2m-1})*c_{m-1}$ и есть искомое минимальное по числу монет представление суммы s.

Доказательство от противного. [skipped]

IK> Каков собственно алгоритм?

Алгоритм простой:

1) построить базис Грёбнера указанного идеала относительно deglex;

2) редуцировать многочлен x_0^s;
3) интерпретировать результат.
Если нужно решить несколько задач для одного и того же набора номиналов, но разных сумм, то базис Грёбнера нужно построить только один раз (это самая трудоемкая операция), и затем очень быстро проводить редукцию для каждой из сумм.

```
{$B-}
const m=9;
      c:array[1..m] of integer=(1,2,5,10,20,50,100,200,500);
      m2=2*m;
type  TTerm=array[1..m2] of integer;
      PTermList=^TTermList;
      TTermList=record
        u,v:TTerm;
        next:PTermList;
      end;
var   Basis:TTermList;
      p,q,r,t:PTermList;
      i,j,k:integer;
function cmp(var x,y:TTerm):integer;
var t,i,dx,dy:integer;
begin
  dx:=0; dy:=0; t:=0;
  for i:=1 to m2 do
  begin
    inc(dx,x[i]); inc(dy,y[i]);
    if t=0 then
    begin
      if x[i]>y[i] then t:=1;
      if y[i]>x[i] then t:=-1;
    end;
  end;
  if dx=dy then cmp:=t else if dx>dy then cmp:=1 else cmp:=-1;
end;
function S(var a,b,c:PTermList):boolean;
var x,y:TTerm;
    i,t:integer;
begin
  S:=False;
  for i:=1 to m2 do
  begin
    if a^.u[i]>b^.u[i] then t:=a^.u[i] else t:=b^.u[i];
    x[i]:=t-a^.u[i]+a^.v[i]; y[i]:=t-b^.u[i]+b^.v[i];
    if x[i]>y[i] then t:=y[i] else t:=x[i];
    dec(x[i],t); dec(y[i],t);
  end;
  case cmp(x,y) of
   1: begin c^.u:=x; c^.v:=y; end;
  -1: begin c^.u:=y; c^.v:=x; end;
  else S:=True;
  end;
end;
function Reduce(var a:PTermList):boolean;
var p:PTermList;
    x:TTerm;
    i:integer;
begin
  p:=Basis.next;
  repeat
  if a<>p then
  begin
    i:=1; while (i<=m2) and (a^.u[i]>=p^.u[i]) do inc(i);
    if i>m2 then
    begin
      if S(a,p,a) then begin Reduce:=true; exit; end;
      p:=@Basis;
    end;
  end;
  p:=p^.next;
  until (p=nil);
  Reduce:=false;
end;
procedure ReduceBasis;
var p,q,t:PTermList;
```

```
begin
  p:=@Basis;
  repeat
    q:=p; p:=p^.next;
    while (p<>nil) and Reduce(p) do
    begin
      q^.next:=p^.next; dispose(p); p:=q^.next;
    end;
    if p=nil then break;
    t:=p;
    while (t^.next<>nil) and (cmp(t^.next^.u,p^.u)=1) do t:=t^.next;
    if t<>p then
    begin
      q^.next:=p^.next; p^.next:=t^.next; t^.next:=p;
    end;
    p:=q^.next;
  until p=nil;
end;
begin
  Basis.next:=nil;
  for i:=1 to m do
  begin
    if i>1 then
    begin
      new(p); with p^ do
      begin
        FillChar(u,SizeOf(u),0); u[1]:=c[i];
        FillChar(v,SizeOf(v),0); v[i]:=1;
        next:=Basis.next;
      end;
      Basis.next:=p;
    end;
    new(p); with p^ do
    begin
      FillChar(u,SizeOf(u),0); u[1]:=c[i]; u[m+i]:=1;
      FillChar(v,SizeOf(v),0);
      next:=Basis.next;
    end;
    Basis.next:=p;
  end;
  write('Construct Groebner basis');
  p:=@Basis; new(r);
  repeat
    q:=p^.next;
    while (q<>nil) do
    begin
      if (not S(p,q,r)) and (not Reduce(r)) then
      begin
        t:=@Basis;
        while (t^.next<>nil) and (cmp(t^.next^.u,r^.u)=1) do t:=t^.next;
        r^.next:=t^.next; t^.next:=r; new(r);
        write('.');
        ReduceBasis;
        p:=@Basis; break;
      end;
      q:=q^.next;
    end;
    p:=p^.next;
  until p=nil;
  writeln(' Done');
  with r^ do
  repeat
    FillChar(u,SizeOf(u),0); FillChar(v,SizeOf(v),0);
    write('Amount of money (0 - exit): '); readln(u[1]);
    if u[1]=0 then break;
    Reduce(r);
    write('Coins: ');
    j:=0;
    for i:=1 to m2 do
    begin
      for k:=1 to u[i] do
        if i<=m then write('+',c[i]) else write('-',c[i-m]);
      inc(j,u[i]);
    end;
```

```
    writeln; writeln('Total number: ',j);
  until false;

  dispose(r);
  p:=Basis.next;
  while p<>nil do
  begin
    q:=p; p:=p^.next; dispose(q);  end;
end.
```

Задача №2

Напечатать все перестановки чисел 1..N

First = (1,2,...,N)
Last = (N,N-1,...,1)
Всего таких перестановок будет N!=N*(N-1)*...*2*1. Для составления алгоритма Next зададимся вопросом: в каком случае i-ый член перестановки можно увеличить, не меняя предыдущих? Ответ: если он меньше какого-либо из следующих членов (членов с номерами больше i).
Мы должны найти наибольшее i, при котором это так, т.е. такое i, что X[i]<X[i+1]>...>X[N] (если такого i нет, то перестановка последняя). После этого X[i] нужно увеличить минимально возможным способом, т.е. найти среди X[i+1],...,X[N] наименьшее число, большее его. Поменяв X[i] с ним, остается расположить числа с номерами i+1,...,N так, чтобы перестановка была наименьшей, то есть в возрастающем порядке. Это облегчается тем, что они уже расположены в убывающем порядке:

```
procedure Next;
begin
  {найти i: X[i]<X[i+1]>X[i+2]>...>X[N]};
  {найти j: X[j]>X[i]>X[j+1]>...>X[N]};
  {обменять X[i] и X[j]};
  {X[i+1]>X[i+2]>...>X[N]};
  {перевернуть X[i+1],X[i+2],...,X[N]};
end;
```

Теперь можно написать программу:

```
program Perestanovki;
  type Pere=array [byte] of byte;
  var N,i,j:byte;
      X:Pere;
      Yes:boolean;
  procedure Next(var X:Pere;var Yes:boolean);
     var i:byte;
     procedure Swap(var a,b:byte);  {обмен переменных}
       var c:byte;
     begin c:=a;a:=b;b:=c end;
  begin
     i:=N-1;
     {поиск i}
     while (i>0)and(X[i]>X[i+1]) do dec(i);
     if i>0 then
       begin
         j:=i+1;
         {поиск j}
         while (j<N)and(X[j+1]>X[i]) do inc(j);
         Swap(X[i],X[j]);
         for j:=i+1 to (N+i) div 2 do Swap(X[j],X[N-j+i+1]);
         Yes:=true
       end
     else Yes:=false
  end;
begin
  write('N=');readln(N);
  for i:=1 to N do X[i]:=i;
  repeat
     for i:=1 to N do write(X[i]);writeln;
     Next(X,Yes)
  until not Yes
end.
```

Решение через рекурсию

Опишем рекурсивную процедуру Generate(k), предъявляющую все перестановки чисел 1,...,N, у которых фиксировано начало X[1],X[2],...,X[k]. После выхода из процедуры массив X будут иметь то же значение, что перед входом. Понятно, что при k=N мы снова имеем только тривиальное решение - саму перестановку. При k<N будем сводить задачу к k+1:

```
procedure Generate(k:byte);
  var i,j:byte;
  procedure Swap(var a,b:byte);
    var c:byte;
  begin c:=a;a:=b;b:=c end;
begin
  if k=N then
    begin for i:=1 to N do write(X[i]);writeln end
  else
    for j:=k+1 to N do
      begin
        Swap(X[k+1],X[j]);
        Generate(k+1);
        Swap(X[k+1],X[j])
      end
end;
```

Основная программа:

```
program PerestanovkiRecursion;
  type Pere=array [byte] of byte;
  var N,i,j:byte;
      X:Pere;
  procedure Generate(k:byte);
      ..............
begin
  write('N=');readln(N);
  for i:=1 to N do X[i]:=i;
  Generate(0)
end.
```

Чтобы до конца разобраться в этой непростой программе, советуем выполнить ее на бумаге при N=3. Обратите внимание, что порядок вывода перестановок не будет лексикографическим!

Задача №3

Напечатать все последовательности длины N из чисел 1,2..M

First = (1,1,...,1) Last = (M,M,...,M)
Всего таких последовательностей будет M^N (докажите!). Чтобы понять. как должна действовать процедура Next, начнем с примеров. Пусть N=4,M=3. Тогда:
Next(1,1,1,1) -> (1,1,1,2) Next(1,1,1,3) -> (1,1,2,1) Next(3,1,3,3) -> (3,2,1,1)
Теперь можно написать общую процедуру Next:

```
procedure Next;
  begin
    {найти i: X[i]<M,X[i+1]=M,...,X[N]=M};
    X[i]:=X[i]+1;
    X[i+1]:=...:=X[N]:=1
end;
```

Если такого i найти не удается, то следующей последовательности нет - мы добрались до последней (M,M,...,M). Заметим также, что если бы членами последовательности были числа не от 1 до M, а от 0 до M-1, то переход к следующей означал бы прибавление 1 в M-ичной системе счисления. Полная программа на Паскале выглядит так:

```
program Sequences;
  type Sequence=array [byte] of byte;
  var M,N,i:byte;
      X:Sequence;
      Yes:boolean;
  procedure Next(var X:Sequence;var Yes:boolean);
    var i:byte;
  begin
    i:=N;
    {поиск i}
    while (i>0)and(X[i]=M) do begin X[i]:=1;dec(i) end;
```

```
      if i>0 then begin inc(X[i]);Yes:=true end
      else Yes:=false
  end;
begin
  write('M,N=');readln(M,N);
  for i:=1 to N do X[i]:=1;
  repeat
    for i:=1 to N do write(X[i]);writeln;
    Next(X,Yes)
  until not Yes
end.
```

Решение через рекурисю.

Опишем рекурсивную процедуру Generate(k), предъявляющую все последовательности длины N из чисел 1,...,M, у которых фиксировано начало X[1],X[2],...,X[k]. Понятно, что при k=N мы имеем тривиальное решение: есть только одна такая последовательность - это она сама.
При k<N будем сводить задачу к k+1:

```
procedure Generate(k:byte);
  var i,j:byte;
begin
  if k=N then
    begin for i:=1 to N do write(X[i]);writeln end
  else
    for j:=1 to M do
      begin X[k+1]:=j; Generate(k+1) end
```

end;

Основная программа теперь выглядит очень просто:

```
program SequencesRecursion;
  type Sequence=array [byte] of byte;
  var M,N:byte;
      X:Sequence;
  procedure Generate(k:byte);
       ............
begin
  write('M,N=');readln(M,N);
  Generate(0)    end.
```

Задача №4

Перечислить все разбиения N на целые положительные слагаемые

Пример: N=4, разбиения: 1+1+1+1, 2+1+1, 2+2, 3+1, 4.
First = (1,1,...,1) - N единиц Last = (N)
Чтобы разбиения не повторялись, договоримся перечислять слагаемые в невозрастающем порядке. Сказать, сколько их будет всего, не так-то просто (см.следующий пункт). Для составления алгоритма Next зададимся тем же вопросом: в каком случае i-ый член разбиения можно увеличить, не меняя предыдущих?
Во-первых, должно быть X[i-1]>X[i] или i=1. Во-вторых, i должно быть не последним эле ментом (увеличение i надо компенсировать уменьшением следующих). Если такого i нет, то данное разбиение последнее. Увеличив i, все следующие элементы надо взять минимально возможными, т.е. равными единице:

```
procedure Next;
  begin
    {найти i: (i<L) and ( (X[i-1]>X[i]) or (i=1) )}
    X[i]:=X[i]+1;
    { L:= i + X[i+1]+...+X[L] - 1 }
    X[i+1]:=...:=X[L]:=1
  end;
```

Через L мы обозначили количество слагаемых в текущем разбиении (понятно, что 1<=L<=N).
Программа будет выглядеть так:

```
program Razbieniya;
  type Razb=array [byte] of byte;
  var N,i,L:byte;
      X:Razb;
  procedure Next(var X:Razb;var L:byte);
```

```
    var i,j:byte;
        s:word;
  begin
    i:=L-1;s:=X[L];
    {поиск i}
    while (i>1)and(X[i-1]<=X[i]) do begin s:=s+X[i];dec(i) end;
    inc(X[i]);
    L:=i+s-1;
    for j:=i+1 to L do X[j]:=1
  end;
begin
  write('N=');readln(N);
  L:=N; for i:=1 to L do X[i]:=1;
  for i:=1 to L do write(X[i]);writeln;
  repeat
    Next(X,L);
    for i:=1 to L do write(X[i]);writeln
  until L=1
end.
```

Задача №5

Перечислить все расстановки 8-ми ферзей на шахматной доске, при которых они не бьют друг друга

Классической задачей, которая решается методом перебора с отходом назад считается задача о восьми ферзях: требуется перечислить все способы расстановки 8-ми ферзей на шахматной доске 8 на 8, при которых они не бьют друг друга. Эту задачу решил больше 200 лет тому назад великий математик Леонард Эйлер. Заметьте, что у него не было компьютера, но тем не менее он абсолютно верно нашел все 92 таких расстановки!
Очевидно, на каждой из 8 вертикалей должно стоять по ферзю. Каждую такую расстановку можно закодировать одномерным массивом
X[1],...,X[8],
где X[i] - номер горизонтали для i-го ферзя. Поскольку никакие два ферзя не могут стоять на одной горизонтали (тогда они бьют друг друга), то все X[i] различны, т.е. образуют перестановку из чисел 1..8. Можно, конечно, перебрать все 8! таких перестановок и выбрать среди них те 92, которые нас интересуют. Но число 8!=40320 довольно большое.
Поэтому мы воспользуемся алгоритмом перебора с отходом назад, который позволит значительно сократить перебор и даст ответ намного быстрее:

```
program Queens;
  const N=8;
  type Index=1..N;
       Rasstanovka=array [Index] of 0..N;
  var X:Rasstanovka;
      Count:word;
  function P(var X:Rasstanovka;k,y:Index):boolean;
    var i:Index;
  begin
    i:=1;
    while (i<k)and(y<>X[i])and(abs(k-i)<>abs(y-X[i])) do inc(i);
    P:=i=k
  end;
  procedure Backtracking(k:Index);
    var i,y:Index;
  begin
    for y:=1 to N do
      if P(X,k,y) then
         begin
           X[k]:=y;
           if k=N then
             begin
               for i:=1 to N do write(X[i]);writeln;inc(Count)
             end;
           Backtracking(k+1)
         end
  end;
begin
```

```
   Count:=0;
   writeln('Расстановки ',N,' ферзей:');
   Backtracking(1);
   writeln('Всего ',Count,' расстановок')
 end.
```

Задача №6

На двумерной плоскости задано N точек с координатами (X_1,Y_1), (X_2,Y_2), ..., (X_n,Y_n). Написать программу, которая из этих точек выделяет вершины квадрата, содержащего максимальное число заданных точек.

ПРИМЕЧАНИЕ: предполагается, что точки, расположенные на сторонах квадрата, принадлежат ему.

Это переборная задача. Обратите внимание, что стороны квадрата могут и не быть параллельны осям координат! Каждую из N точек мы последовательно рассматриваем в качестве верхнего левого угла квадрата, каждую из оставшихся N-1 - как нижнюю правую вершины и смотрим, есть ли для них в этом множестве из N точек точки, соответствующие верхнему правому и нижнему левому углу. Если да, то подсчитываем, сколько точек лежат в данном квадрате.

Пусть координата левого верхнего угла (x_1,y_1), нижнего правого (x_2,y_2), тогда координата пересечения диагоналей четырехугольника ((x_1+x_2)/2,(y_1+y_2)/2); координата верхнего правого угла ((x_1+x_2)/2+[y_1-(y_1+y_2)/2],(y_1+y_2)/2+[x_1-(x_1+x_2)/2])= =(($x_1+x_2+y_1-y_2$)/2, ($x_1-x_2+y_1+y2$)/2), нижнего левого - (($x_1+x_2-y_1+y_2$)/2,($-x_1+x_2+y_1+y_2$)/2)

(Постройте чертеж и проверьте !).

Для (x_1,y_2) и (x_2,y_2) должны выполняться следующие неравенства: $x_1<=x_2$, $y_1>=y_2$ (иначе это будут уже не левый верхний и правый нижний углы квадрата).

Программа:

```
{В исходном множестве поочередно перебираются все пары точек.}
{Предполагая, что отрезок, соединяющий эти точки, является ребром}
{квадрата строим квадрат и смотрим, все ли его вершины имеются в}
{исходном множестве. Если все, то определяем, сколько точек из}
{исходного множества лежит внутри этого квадрата. Если это число}
{превосходит старый рекорд то запоминаем найденный квадрат.}
{ }
{$A-,B-,D-,E+,F-,I+,L-,N-,O-,R-,S-,V-}
{$M 65520,0,655360}
uses crt;
const
maxn = 100;{ Максимальное число точек }
type
 xy = record x,y : real end; { Тип для записи координат точек }
var
 m : array[1..maxn] of xy; { Координаты точек множества }
 i,j,g,k,n,p : word; { вспомогательные переменные  }
 num : word; { для записи числа точек в текущем квадрате }
 rec : word; { для записи числа точек в лучшем квадрате }
 a1,b1,c1 : real; { вспомогательные переменные  }
 r,c : array[1..5] of xy;{ для записи вершин квадратов }
 f1,f2 : boolean;
 o : array[1..4] of shortint;
Function sign(a : real) : shortint;{ Функция signum }
begin
 if a<0 then sign:=-1
 else if a>0 then sign:=1
 else sign:=0
end;
{ нахождение коэффициентов прямой,
проходящей через точки x1,y1 и x2,y2 }
procedure getabc(x1,y1,x2,y2:real; var a,b,c:real);
begin
a:=y2-y1; b:=x1-x2; c:=-(a*x1+b*y1)
end;
begin
 write('Введите число точек...'); readln(n);
 for i:=1 to n do
 begin
 write('Введите координаты ',i,'-ой точки...');
 readln(m[i].x,m[i].y); end;
 rec:=0;{ Обнуление рекорда }
```

```
for i:=1 to n do
 { Перебор всех квадратов, для которых отрезок m[i]-m[j] }
 for j:=1 to n do { является ребром }
 if i<>j then
 begin
c[1]:=m[i]; c[2]:=m[j];
 { Определение вершин квадрата }
 c[3].x:=c[2].x+(c[1].y-c[2].y);
 c[3].y:=c[2].y+(c[2].x-c[1].x);
 c[4].x:=c[1].x+(c[1].y-c[2].y);
 c[4].y:=c[1].y+(c[2].x-c[1].x);
 c[5]:=c[1];
 num:=0;
{ Проверка на наличие всех вершин квадрата
в исходном множестве точек }
f1:=false; f2:=false;
for g:=1 to n do
if (m[g].x=c[3].x) and (m[g].y=c[3].y) then f1:=true;
for g:=1 to n do
if (m[g].x=c[4].x) and (m[g].y=c[4].y) then f2:=true;
 if (c[1].x=c[2].x) and (c[1].y=c[2].y) then f1:=false;
if f1 and f2 then
{Если все вершины квадрата есть в исходном множестве}
for k:=1 to n do { то определяем число точек в квадрате}
 begin
 for g:=1 to 4 do
 begin
getabc(c[g].x,c[g].y,c[g+1].x,c[g+1].y,a1,b1,c1);
 o[g]:=sign(a1*m[k].x+b1*m[k].y+c1);
 end;
 if ((o[1]=o[2]) and (o[2]=o[3]) and (o[3]=o[4])) or
((o[1]=o[2]) and (o[2]=o[3]) and (o[4]=0)) or
((o[1]=o[2]) and (o[2]=o[4]) and (o[3]=0)) or
((o[1]=o[3]) and (o[3]=o[4]) and (o[2]=0)) or
((o[2]=o[3]) and (o[3]=o[4]) and (o[1]=0)) or
((m[k].x=c[1].x) and (m[k].y=c[1].y)) or
((m[k].x=c[2].x) and (m[k].y=c[2].y)) or ((m[k].x=c[3].x)
 and (m[k].y=c[3].y)) or ((m[k].x=c[4].x)
 and (m[k].y=c[4].y)) then inc(num);
 end;
 if rec<num then begin r:=c; rec:=num end;
 end;
 if rec=0 then { Не найдено ни одного квадрата }
 begin
 writeln('Не найдено ни одного квадрата.'); halt
 end;
 { Вывод результатов }
 write('Лучший квадрат : ');
for i:=1 to 3 do write('(',r[i].x:2:2,
```

Задача №7

Задан набор неповторяющихся пар (A_i,A_j), A_i, A_j принадлежат множеству A={A_1, A_2, ..., A_n}. Необходимо составить цепочку максимальной длины по правилу (A_i,A_j)+(A_j,Ak)=(A_i,A_j,A_k). При образовании этой цепочки любая пара может быть использована не более одного раза.

Для более удобного хранения информации заведем матрицу C[1...n,1..n] (так называемую матрицу смежности) в которой C[i,j]=1, если в наборе есть пара (A_i,A_j) и C[i,j]=0 иначе. Будем строить все возможные цепочки (по правилу, данному в условии) и искать среди них ту, которая имеет максимальную длину. В качестве начального символа цепочки можно взять любой символ из A. Пусть это символ Ai. Ищем, просматривая строку i матрицы C слева направо элемент C[i,j]=1 (другими словами, ищем пару с первым элементом Ai). Если такого элемента не существует, то берем в качестве начала строки другой элемент множества A. Если элемент C[i,j]=1 найден, то ему соответствует пара (A_i,A_j). Помечаем ее как уже использованную полагая, например, C[i,j]=-1. Далее просматриваем слева направо строку j матрицы C в поисках еще не использованной пары (A_j,A_k) (C[j,k]=1). Присоединяем элемент Ak к имеющейся цепочке, полагаем C[j,k]=-1, ищем единичный элемент в строке k и т.д. Предположим, на некотором шаге мы получили цепочку A_i A_j A_k ... A_s A_l A_p и в строке p матрицы больше нет ни одного единичного

элемента. Это означает, что при таком подборе предыдущих элементов мы нашли максимальную по длине строку. Если ее длина больше длин всех найденных ранее строк, запоминаем эту строку как рекорд. После этого "отщепляем" от строки последний элемент Ар и смотрим, есть ли еще в строке l единичный элемент с индексом, большим р. Если да, то приписываем уже этот элемент к строке и пытаемся затем снова увеличить длину полученной строки, если же нет, то "отщепляем" от строки элемент A_1, в строке S ищем единичный элемент с индексом, большим l и т.д. Останов осуществляется тогда, когда мы должны "отщепить" от строки A_i. Перебираем цепочки, начинающиеся со всех возможных элементов множества А. Находим строку максимальной длины:

```
const M=10; {максимально число элементов в A}
{будем считать, что A состоит из чисел от 1 до N}
var c:array[1..M,1..M] of integer;
curstr, maxstr: array[0..M] of integer;
{в этих переменных хранятся текущая цепочка и}
{цепочка максимальной длины.}
{В нулевом элементе хранится длина цепочки}
N,E : integer; {N - число элементов в A}
i,j,k : integer; {E - число пар в наборе}
procedure find;
var l,j : integer;
begin
l:=curstr[curstr[0]]; {l = последний элемент цепочки}
for j:=1 to N do {просмотр строки l}
if C[l,j]=1
then begin
curstr[0]:=curstr[0]+1;
curstr[curstr[0]]:=j; {j -> в цепочку}
c[l,j]:=-1; {пара использована}
find;
c[l,j]:=1; {пару снова разрешено использовать}
curstr[0]:=curstr[0]-1;
end;
if curstr[0]>maxstr[0] {если нашли более}
then maxstr:=curstr {длинную строку}
end;
begin
readln(N); readln(E);
for i:=1 to N do
for j:=1 to N do
C[i,j]:=0;
for k:=1 to E do begin
write('очередная пара: ',i,j);
c[i,j]:=1
end;
for i:=1 to N do begin
curr[0]:=1; {поиск цепочки}
curr[1]:=i; {начинающейся элементом i}
find;
end;
for i:=1 to maxstr[0] do
write(maxstr[i]); {печать максимальной строки}
end.
```

Задача № 8

Имеется N городов. Для каждой пары городов (I,J) можно построить дорогу, соединяющую эти два города и не заходящие в другие города. Стоимость такой дороги A(I,J). Вне городов дороги не пересекаются.Написать алгоритм для нахождения самой дешевой системы дорог, позволяющей попасть из любого города в любой другой. Результаты задавать таблицей B[1:N,1:N], где B[I,J]=1 тогда и только тогда, когда дорогу, соединяющую города I и J, следует строить.

Легко понять, что сеть дорог будет реализовывать некоторый связный (так как можно проехать из любого города в любой) граф без циклов (так как одно ребро из цикла можно выбросить, а связный граф останется связным). Поэтому алгоритм построения сети дорог минимальной суммарной стоимости очень прост. На каждой итерации необходимо находить дорогу минимальной стоимости, которая не образует цикла с уже выбранными дорогами на предыдущих итерациях. Основную трудность такого решения составляет проверка условия, образуют ли ребра цикл. Однако решение существенно упрощается, если рассматривать только минимальные ребра

только между двумя множествами: множеством помеченных вершин и множеством непомеченных вершин. Понятно, что эти множества должно соединять хотя бы одно ребро, чтобы граф был связным. Ясно, что оно должно быть минимальным по длине. В описываемом ниже алгоритме это делается следующим образом. Для каждой вершины к из множества непомеченных вершин (а на начальном этапе это все вершины, кроме первой) определяется ближайшая вершина из множества помеченных вершин БЛИЖ[к]. На каждой итерации определяется кратчайшее ребро (i,j) между множеством помеченных вершин и множеством непомеченных вершин, используя массив БЛИЖ. Найденное ребро выбирается для системы дорог, а соответствующая вершина j считается помеченной. После этого пересчитывается массив БЛИЖ. При этом учитывается, что к изменение некоторой величины БЛИЖ[k] может произойти только тогда, когда расстояние от k до j меньше, чем от k до БЛИЖ[k].

Алгоритм

```
для i от 1 до N выполнять
нц
флаг[i]:=0;
БЛИЖ[i]:=1
кц
флаг[1]:=1;
для k от 1 до N-1 выполнять
нц
минрас:=бесконечность;
для i от 2 до N выполнять
если флаг[i]=0 и минрас > C[БЛИЖ[i],i]
то минрас:=C[БЛИЖ[i],i];
j:=i;
все
Вывод ребра (БЛИЖ[j],j)
флаг[j]:=1;
для i от 2 до N выполнять
если флаг[i]=0 и C[БЛИЖ[i],i]>C[i,j]
то БЛИЖ[i]:=j;
все
кц
```

Задача № 9

Элементами массива a[1..n] являются неубывающие массивы [1..m] целых чисел (a: array [1..n] of array [1..m] of integer; a[1][1] <= ... <= a[1][m], ..., a[n][1] <= ... <= a[n][m]). Известно, что существует число, входящее во все массивы a[i] (существует такое x, что для всякого i из [1..n] найдётся j из [1..m], для которого a[i][j]=x). Найти одно из таких чисел x.

Введем массив b[1]..b[n], отмечающий начало "остающейся части" массивов a[1]..a[n].

```
  for k:=1 to n do begin
  |  b[k]:=1;
  end;
  eq := true;
  for k := 2 to n do begin
  | eq := eq and (a[1][b[1]] = a[k][b[k]]);
  end;
  {инвариант: оставшиеся части  пересекаются,  т.е.  существует
   такое  x,  что для всякого i из [1..n] найдётся j из [1..m],
   не меньшее b[i], для которого a[i][j] =  x;  eq  <=>  первые
   элементы оставшихся частей равны}
  while not eq do begin
  | s := 1; k := 1;
  | {a[s][b[s]] - минимальное среди a[1][b[1]]..a[k][b[k]]}
  | while k <> n do begin
  | | k := k + 1;
  | | if a[k][b[k]] < a[s][b[s]] then begin
  | | | s := k;
  | | end;
  | end;
  | {a[s][b[s]] - минимальное среди a[1][b[1]]..a[n][b[n]]}
  | b [s] := b [s] + 1;
  | for k := 2 to n do begin
  | | eq := eq and (a[1][b[1]] = a[k][b[k]]);
  | end;
  end;
```

```
  writeln (a[1][b[1]]);
```

Задача №10

На прямой своими концами заданы N отрезков и точка X. Определить, принадлежит ли точка межотрезочному интервалу. Если да, то указать концевые точки этого интервала. Если нет, то найти,
а. Какому количеству отрезков принадлежит точка.
б. Каким именно отрезкам принадлежит точка.

Отсортируем концевые точки отрезков в порядке неубывания. Если несколько концевых точек отрезков имеют одинаковую координату, то сначала выпишем те из них, которые являются начальными точками отрезков, а за ними - конечные точки. Для такой сортировки необходимо для каждой точки сохранить информацию, является ли она правым или левым концом отрезка.
Рассмотрим одну из возможных реализаций:
Заведем массив A[1..2,1..2*N]; сначала в ячейки A[1,2i-1] и A[1,2i] заносим координаты начала и конца i-го отрезка, а в ячейки A[2,2i-1] и A[2,2i] - числа +i и -i -- признаки того, что соответствующие координаты являются началом и концом отрезка i. Сортируем столбцы массива A по неубыванию элементов первой строки, и если при этом несколько элементов первой строки равны (то есть соответствующие точки имеют одинаковые координаты), то сначала выписываем начальные точки отрезков (A[2,i]>0), а затем - конечные (A[2,2i-1]<0).
Заведем еще массив Pr[1..N], в котором будет храниться информация об отрезках, содержащих точку A[1,i]. После окончания работы алгоритма Pr[j]=1, если отрезок j содержит точку A[1,i], и Pr[j]=0 иначе. Сначала массив Pr нулевой.
Пусть в переменной C хранится количество отрезков, пересекающихся в точке A[1,i]. Сначала C=0.
Делаем проверку, размещается ли точка X левее A[1,1] или правее A[1,2*N]. Если да, то выводим сообщение о принадлежности точки бесконечному интервалу, если же нет, то будем росматривать массив A слева направо в порядке неубывания элементов. Пока выполняется следующее условие:

```
 массив A не закончился и (или текущая точка A[1,i] < X
или текущая начальная точка отрезка A[1,i] = X)
 повторять
 Если A[1,i] - начальная точка отрезка,
 то
увеличить C на 1 (начался еще один отрезок с номером A[2,i]),
и присвоить Pr[A[2,i]] значение 1.
 Если A[1,i] - конечная точка отрезка,
 то
уменьшить C на 1 (отрезок с номером -A[2,i] закончился),
 и присвоить Pr[-A[2,i]] значение 0..
 конец_пока.
 Когда мы выйдем из цикла, то проверим:
 Если C=0, то X лежит на межотрезочном
  интервале (A[1,i-1], A[1,i]), иначе X принадлежит C отрезкам.
   Номера отрезков есть индексы единичных элементов массива Pr.
Набросок этого фрагмента программы:
 i:=1;
 пока (i<=2*N) и
 ((A[1,i]<X) или
 (A[1,i]=X) и (A[2,i]>0))
 повторять
 если A[2,i]>0
 то C:=C+1;
 Pr[A[2,i]:=1
 конец_то
 иначе C:=C-1;
 Pr[-A[2,i]:=0
 конец_иначе
 i:=i+1;
 конец_пока
 если C=0,
то X лежит на межотрезочном интервале (A[1,i-1],A[1,i]),
```

```
иначе X принадлежит C отрезкам. Напечатаем номера этих отрезков:
 для i:=1 до N повторять
 если Pr[i]=1
 то печатать i
```

Задача №11

Задана матрица натуральных чисел A(n,m). За каждый проход через клетку (i,j) взымается штраф A(i,j). Необходимо минимизировать штраф и
а) Пройти из какой-либо клетки 1-ой строки в n-ую строчку, при этом из текущей клетки можно перейти
1) в любую из 3-х соседних, стоящих в строке с номером на 1-цу большем;
2) в любую из 8 соседних клеток;
б) Реализовать пункт а) для перехода из клетки (1,1) в (n,m).

Для решения пункта 1 задачи достаточно воспользоваться
тем фактом, что для определения минимальной величины штрафа, взимаемого за проход в клетку i-той строки достаточно знать минимальные величины штрафа, взимаемого за проход в клетки (i-1)-той строки, которые являются соседними рассматриваемой клетке. Поэтому алгоритм решения пункта 1 следующий:

```
для i от 1 до n
Штраф[i,1]:=A[i,1] для i от 2 до n
для j от 1 до m
нц
Штраф[i,j]:=Штраф[i-1,j]+A[i,j];
если j>1 и Штраф[i,j] < Штраф[i-1,j-1]+A[i,j];
 то Штраф[i,j]:=Штраф[i-1,j-1]+A[i,j];
если j<m и Штраф[i,j] < Штраф[i-1,j+1]+A[i,j];
 то Штраф[i,j]:=Штраф[i-1,j+1]+A[i,j];
кц
```

Задача №12

Даны две строки x и y. Строка x состоит из нулей и единиц, строка y из символов A и B. Можно ли строку x преобразовать в строку y по следующему правилу: цифра 0 преобразуется в непустую последовательность букв A, а цифра 1 - либо в непустую последовательность букв A, либо в непустую последовательность букв B?

Пусть строка S_1 состоит из цифр 0 и 1 и имеет длину N, а строка S_2 (из символов A и B) - длину M. Заведем матрицу A размера N на M, при этом строки матрицы помечаются i-ой цифрой строки S_1, а j-й столбец - j-м символом строки S_2.
Возьмем в качестве примера S_1='00110', S_2='AAAABBAA'.
Первая цифра строки S_1 (цифра 0) может быть преобразована в одну из последовательностей букв 'A','AA','AAA','AAAA', являющиеся префиксами строки S_2. Заносим символ 'x' в те столбцы первой строки, буквы-пометки которых соответствуют последним буквам возможных последовательностей. Таким образом, помечаются элементы A[1,1], A[1,2], A[1,3] и A[1,4].
Шаг алгоритма: будем преобразовывать очередную цифру S_1[i], которой соответствует строка i. Находим 'x' в предыдущей строке при просмотре ее слева направо (этому столбцу соответствует последняя буква в какой-то из непустых последовательностей букв, порожденных на предыдущем шаге). Если текущую цифру можно преобразовать в последовательность букв, помечающих следующие за данным столбцы, то в этих столбцах в данной строке ставим 'x'.
Далее от места над последней помеченной ячейкой ищем в предыдущей строке 'x' и, когда находим, повторяем указанные выше операции.
Эти действия проводим далее для i=2,...,N.
Вид матрицы после N шагов:

Если после N шагов в позиции (N,M) стоит x, то строки можно преобразовать друг в друга.

Замечание: Можно обойтись и одномерным массивом. В самом деле, при заполнении следующей строки мы обращаемся только к элементам предыдущей строки, к каждому - по одному разу.
Алгоритм (без учета замечания) может быть следующим:

```
for i:=1 to N do
for j:=1 to M do
A[i,j]:=' '; {инициализация}
if S1[1]=0
then element:='A' {0 преобразуется в A}
else element:=S2[1]; {1 - в A или в B}
i:=1;
while (i<=M) and (S2[i]=element) do begin {первый шаг}
A[1,i]:='x';
i:=i+1
end;
for i:=2 to N do begin
j:=2;
while j<M do begin {просмотр строки i}
if A[i-1,j-1]='x'
then begin
if S1[i]=0
then element:='A'
else element:=S2[j];
if S2[j]=element
then
while (j<=M) and (S2[j]=element) do begin
A[i,j]:='x';
j:=j+1;
end {end for while}
else j:=j+1
end {end for then}
else j:=j+1
end {end for while}
end; {end for for}
if A[N,M]='x'
```

then writeln('Можно преобразовать') else writeln('Нельзя преобразовать');</P></DIR>
Напишите программу, использующую одномерный массив.

Задача №13.

Пусть известно, что для перемножения матрицы размера n*m на матрицу размера m*k требуется n*m*k операций. Необходимо определить, какое минимальное число операций потребуется для перемножения n матриц $A_1,...A_n$, заданных своими размерами n(i)*m(i).

При этом можно перемножать любые две рядом стоящие матрицы, в результате чего получается матрица нужного размера.
Замечание:
n(i) - число строк в матрице A_i
m(i) - число столбцов в матрице A_i
n(i)=m(i)+1.
Определим через F[i,j] минимальное число операций, которое требуется для перемножения группы матриц с номерами от i до j включительно. Ясно, что F[i,i]=0. Перемножение матриц в такой группе может производиться различными способами, а именно, производить сначала перемножение наилучшим способом группы от i до k, затем от k+1 до j, наконец перемножить получившиеся матрицы. Понятно, что k может быть величиной от i до j-1. Учитывая требование получить наилучший результат, величина F[i,j] определяется как
F[i,j]=max(F[i,k]+F[k+1,j]+n[i]*n[k+1]*m[j]), где k может быть величиной от i до j-1, n[i], n[k+1], m[j] определяют размеры матриц, получившихся при перемножении в группах.

```
для i от 1 до N выполнять
F[i,i]:=0;
```

```
длял от 1 до N-1 выполнять
для i от 1 до N-l выполнять
нц
Kol:=бесконечность;
j:=i+l;
для k от i до j-1 выполнять
если Kol > F[i,k]+F[k+1,j]+n[i]*n[k+1]*m[j]
 то Kol:=F[i,k]+F[k+1,j]+n[i]*n[k+1]*m[j];
все
F[i,j]:=Kol;
кц
```

Задача №14

а) Из последовательности, состоящей из N чисел, вычеркнуть минимальное количество элементов так, чтобы оставшиеся образовали строго возрастающую последовательность.
б) Из заданной числовой последовательности A[1..N] вычеркнуть минимальное число элементов так, чтобы в оставшейся подпоследовательности каждый последующий элемент был больше предыдущего кроме, быть может, одной пары соседних элементов (одного "разрыва" возрастающей подпоследовательности).
Например: A=(1,2,3,2,4,3,4,6);
Искомая подпоследовательность (1,2,3,2,3,4,6)
Разрыв подчеркнут. ---
б) Из заданной числовой последовательности A[1..N] вычеркнуть минимальное число элементов так, чтобы в оставшейся подпоследовательности каждый последующий элемент был больше предыдущего кроме, быть может, m пар соседних элементов (возрастающая подпоследовательность с m "разрывами").

а). Пусть K(L,i) обозначает множество возрастающих подпоследовательностей длины L, которые составлены из элементов с номерами от 1 до i-1. Понятно, что их может быть очень много. Однако их число можно существенно сократить, воспользовавшись следующим фактом: из двух цепочек длины L более перспективной будет цепочка, у которой величина последнегоэлемента меньше, так как ее может продолжить большее числоэлементов. Пусть S(L,i) - это именно такая самая перспективная цепочка длины Lc минимальным по величине последним элементом, а S(i) – это множество всех цепочек S(L,i) со всевозможными L. Понятно, что в S(i) содержится не более i-1 цепочек (с длинами 1,...,i-1). Пусть мы знаем S(i). Для того, чтобы определить, какие цепочки может продолжать i-й элемент,достаточно знать последние элементы перспективных цепочек. Для этого удобно завести массив адресов АДР последних элементов перспективных цепочек длины 1,2,...,n. При этом легко понять, что последний элемент перспективной цепочки длины s не превышает последнего элемента перспективной цепочки длины s+1. Следовательно, для очередного i-го элемента достаточно найти только максимальную длину перспективной цепочки, для которой он будет последним. Понятно, что это будет цепочка максимальной длины, последний элемент которой меньше текущего элемента. Учитывая упорядоченность последних элементов перспективных цепочек, поиск можно сделать дихотомией. При присоединении i-го элемента к какой-нибудь цепочке длины p ее длина увеличивается на 1, а ее последним элементом становится элбмент i. При этом множество S(i+1) совпадает с S(i) за исключением одной цепочки S(p+1,i+1) = S(p,i) сцепленная с i-ым элементом. Для хранения цепочек удобно хранить для каждого элемента номер элемента, который ему предшествует в цепочке.
Другое решение этой задачи:
Заведем 3 массива длины N: в массиве A[1..N] помещаются самичисла последовательности. Элемент B[i] имеет значение длины максимальной возрастающей подпоследовательности, завершающейся элементом A[i], а величина C[i] есть индекс предшествующего элементу A[i] элемента вэтой максимальной последовательности (A[i]=0 есть такого элемента нет).
Если N=1, то A[1] и есть искомая подпоследовательность. При этом B[1]=1, C[1]=0. Предположим, что мы заполнили массивы A,B и C от начала до элемента M-1. Попытаемся обработать элемент A[M]. Просматривая массив A от 1 до M-1-го элемента мы ищем такое A[k], что одновременно
1) A[k]<A[M],
2) B[k] максимально.

Очевидно, что для того, чтобы получить максимальную по длине подпоследовательность, заканчивающуюся A[M], этот элемент надо приписать к подпоследовательности с последним элементом A[K]. Получаем, что B[M]=B[K]+1, C[M]=K.
Пусть мы обработали все N элементов массива A. Находим максимальный элемент массива B, пусть это будет B[K]. По построению это длина максимальной подпоследовательности. Выписываем эту подпоследовательность: полагаем j:=k; печатаем элемент A[j]; так как предшествующий в последовательности элемент имеет индекс C[j], то делаем присваивание j:=C[j], распечатываем этот элемент и т.д., до тех пор, пока j не станет 0 (т.е. мы дошли до начала последовательности).

Запись алгоритма:

```
for i:=2 to N do B[i]:=0;
C[1]:=0; B[1]:=1; Max:=1; IndexMax:=1;
for i:=2 to N do
for j:=1 to i-1 do
if (A[j]<A[i]) AND (B[i]<B[j]+1)
then begin
C[i]:=j;
B[i]:=B[j]+1;
if B[i]>Max
then begin Max:=B[i]
IndexMax:=i
end;
while IndexMax<>0 do begin
writeln(A[Index-Max]);
IndexMax:=C[IndexMax]
```

end;

В программе в переменной Max хранится максимальная найденная до сих пор длина подпоследовательности, в IndexMax находится индекс последнего элемента этой подпоследовательности.
Структура данных, каждый элемент которой (в данном случае A[i]) содержит ссылку (в этой задаче C[i]) на предыдущий элемент (или, если требуется, на последующий элемент) называется однонаправленным списком. Если у элемента есть ссылка как на предыдущий, так и на последующий за ним элемент, то список двунаправленный (его можно реализовать, используя не один массив для индексов, а два).
б). Частный случай пункта в).
в). Заведем матрицу C[0..m+1,1..n]. В ней i-тая строка будет хранить информацию о последовательностях с i-1 разрывами; j-ый элемент в этой строке есть длина самой длинной подпоследовательности элементов "хвоста" массива A (от j-го элемента до n-го), начинающейся в j-ой позиции и с не более чем i-1 разрывом.
Правило заполнения матрицы:
Заполним нулевую строку нулями (чтобы можно было заполнить первую строку по общему алгоритму). Для каждого номера строки i от 1 до m+1 выполнить следующие действия:
Для j-го элемента массива A (j изменяется от n до 1) находим максимальную по длине подпоследовательность, которую можно присоединить к этому элементу так, чтобы получить подпоследовательность максимальной длины с не более чем i-1 разрывом. Для этого мы:
1) ищем элемент A[k] последовательности A, больший A[j], стоящий в массиве A правее j-го элемента и с максимальным C[i,k];
2) затем просматриваем элементы i-1 -ой строки матрицы C, начиная с j+1 -го и до конца, и находим C[i-1,s] - максимальный из них;
3) сравниваем C[i-1,s] с C[i,k]. Больший из них (обозначим его C[row,col]), увеличенный на 1, запоминаем в C[i,j]. Это и будет длина максимальной подпоследовательности, начинающейся в позиции j, с не более чем i-1 разрывом.
4) Запоминаем индексы row и col элемента массива C, предшествующего C[i,j], в ячейках X[i,j] и Y[i,j] соответственно.
После окончания цикла максимальный элемент m+1 -ой строки матрицы C и есть максимальная длина возрастающей подпоследовательности с m разрывами. Выписать всю подпоследовательность можно следующим образом: для каждого элемента подпоследовательности в массивах X и Y хранится информация о предшественнике. Мы, начиная

с максимального элемента m+1 -ой строки матрицы C, восстанавливаем всю подпоследовательность.
Обоснование алгоритма:
Пусть мы знаем C[i-1,j] для всех j от 1 до n и для некоторого i, а также C[i,k] для k от j+1 до n. Мы хотим вычислить C[i,j].
Для j-го элемента массива A существует максимальная по длине подпоследовательность с не более чем i-1 -им разрывом, начинающаяся с A[j]. Второй элемент (обозначим его A[k]) этой максимальной подпоследовательности (если он, конечно, есть) может быть
1) больше, чем A[j]. Тогда мы находим его среди элементов, обладающих следующими свойствами:
а) k>j,
б) C[i,k] максимальный (т.е. мы присоединяем к A[j] максимальную по длине подпоследовательность с не более чем i-1 разрывом, формируя подпоследовательность опять не более чем с i-1 разрывом);
2) меньше или равный, чем A[j]. Тогда мы ищем его среди элементов, обладающих следующими свойствами:
а) k>j;
б) C[i-1,k] максимальный (т.е. мы присоединяем максимальную подпоследовательность с не более чем i-2 -мя разрывами, формируя подпоследовательность с не более чем i-1 разрывом).
Полученная подпоследовательность получается максимальной длины, так как длина подпоследовательности, начиная с A[k], максимальная.
Упоминавшиеся выше индексы row и col, которые запоминаются в X[i,j] и Y[i,j] соответственно, обозначают
col - индекс следующего за A[j] элемента в максимальной по длине подпоследовательности, начинающейся в позиции j и имеющей не более i-1 разрыва;
row-1 - максимальное количество разрывов в подпоследовательности, начинающейся в A[col].

```
const max=100; {максимальная длина массива A}
var
sw,l,i,j,k,n,m,maxind,swind:integer;
c:array [0..max,1..max] of integer;
x,y:array [1..max,1..max] of integer;
a,b:array [1..max] of integer;
begin
Write('Введите N:');
readln(n);
Writeln('Введите последовательность');
for i:=1 to n do
read(a[i]);
readln;
Write('Введите количество разрывов подпоследовательности:');
readln(m);
fillchar(c,sizeof(c),0); {зануление c, x и y} fillchar(x,sizeof(x),0);
fillchar(y,sizeof(y),0);
for i:=1 to m+1 do
for j:=n downto 1 do
begin
c[i,j]:=1;
for k:=j+1 to n do
if (a[k]>a[j]) and (c[i,k]>=c[i,j]) then
begin
c[i,j]:=c[i,k]+1;
x[i,j]:=k;
y[i,j]:=i;
end;
for k:=j+1 to n do
if c[i-1,k]>=c[i,j] then
begin
c[i,j]:=c[i-1,k]+1;
x[i,j]:=k;
y[i,j]:=i-1;
end;
end;
maxind:=1;
for i:=1 to n do
if c[m+1,i]>c[m+1,maxind] then
```

```
maxind:=i;
l:=c[m+1,maxind];
j:=maxind;i:=m+1;k:=1;
while (c[i,j]<>1) do
begin
b[k]:=j;  {в массив b выписываем индексы элементов} inc(k);
 {максимальной по длине подпоследователь}
sw:=x[i,j]; {ности в прямом порядке}
i:=y[i,j];
j:=sw;
end;
b[k]:=j;
for i:=1 to k do write(a[b[i]],' ');
writeln;
writeln('Длина=',l);
end.
```

Задача №15

Возвести число A в натуральную степень n за как можно меньшее количество умножений. Можно, конечно, число A умножить само на себя n-1 раз, но для этого надо выполнить n-1 операцию умножения. Рассмотрим метод, требующий меньшего числа умножений (он, однако, не всегда дает минимальное число умножений).

Если n - четное, n=2m то будем вычислять A^n, используя тождество $A^{am}=(A^m)^2$;
если же n=2m+1, то
$A^{2m+1}=(A^{2m})*A=((A^m)^2)*A$.
Так(м образом, возведение A в 13 степень будет выглядеть следующим образом:
$(A^{13})=((A^6)^2)*A=(((A^3)^2)^2)*A=(((A*A*A)^2)^2)*A$
и вычисление требует 5 операций умножения.
Используя данный метод, для возведения в степень n потребуется порядка $\log_2(n)$ операций умножения.
Программа на Паскале может выглядеть так:

```
var A,N: integer;
function power(N: integer): integer;
begin
if N>1
then if odd(N) {N нечетно?}
then power:=SQR(power(N div 2))*A else power:=SQR(power(N div 2))
else power:=A
end;
begin
read(A,N);
writeln(power(N));
```

end;
Можно ту же самую идею реализовать и по другому (далее мы приводим выдержку из книги Д.Кнута "Искусство программирования для ЭВМ", т.2, с.482):
"Запишем n в двоичной системе счисления и заменим в этой записи каждую цифру "1" парой букв SX, а каждую цифру "0" - буквой S, после чего вычеркнем крайнюю левую пару букв SX. Результат, читаемый слева направо, превращается в правило вычисления x^n, если букву "S" интерпретировать как операцию возведения в квадрат, а букву "X" - как операцию умножения на x. Например, если n=23, то его двоичным представлением будет 10111; строим последовательность SX S SX SX SX, удаляем из нее начальную пару SX и в итоге получаем следующее правило вычисления: S SX SX SX. Согласно этому правилу, мы должны "возвести x в квадрат, затем снова возвести в квадрат, затем умножить на x, возвести в квадрат, умножить на x, возвести в квадрат и, наконец, умножить на x"; при этом мы последовательно вычисляем x^2, x^4, x^5, x^{10}, x^{11}, x^{22}, x^{23}.
Этот "бинарный метод" легко обосновать, рассмотрев последовательность получаемых в ходе вычисления показателей: если "S" интерпретировать как операцию умножения на 2, а "X" - как операцию прибавления 1 и если начать с 1, а не с x, то наше правило дает нам в соответствии со свойствами двоичной системы счисления число n".

Приведенный метод не дает минимального числа операций умножения. Для вычисления x^{23} нам, по изложенному выше методу, потребуется 7 операций умножения. В действительности их необходимо только 6:
$x \to x^2 \to x^3 \to x^5 \to x^{10} \to x^{20} \to x^{23}$.

Задача №16

Найти максимальную по длине последовательность z, полученную вычеркиванием элементов как из x, так и из y.

Пусть $x=x_1,x_2, \dots ,x_m$, $y=y_1,y_2, \dots ,y_n$.
Заведем матрицу A[0..m,0..n]. Элемент A[i,j] будет длиной максимальной общей подпоследовательности у $x_1, \dots ,x_i$ и у $y_1, \dots, y_j$. Сначала A[i,0]=A[0,j]=0, i=0, ... ,m, j=0, ... ,n. Пусть $x_i=y_j$, тогда требуется увеличить длину максимальной общей подпоследовательности $x_1, \dots ,x_{i-1}$ и $y_1, \dots ,y_{j-1}$ на 1: A[i,j]=A[i-1,j-1]+1, если $x_i=y_j$. В случае, если $x_i<>y_j$, то, очевидно, A[i,j]=max{A[i-1,j],A[i,j-1],A[i-1,j-1]}, но так как всегда A[i-1,j-1]<=A[i,j-1], то A[i,j]=max{A[i-1,j],A[i,j-1]}. Величина A[m,n] и дает длину максимальной общей подпоследовательности. Найдем саму подпоследовательность. Пусть A[m,n]=d. Двигаясь по последней строке справа налево ищем самый левый элемент в этой строке со значением d. Двигаемся от него вверх по столбцу в поиске элемента столбца с минимальным первым индексом и значением d. Пусть это A[i,j]. Элемент A[i-1,j-1] обязан быть равен d-1, а x_i и y_i - это последние общие совпадающие элементы в x и y. Начиная от элемента A[i-1,j-1] повторяем, как было описано выше, движение влево и вверх по матрице, находим предпоследний совпадающий элемент в x и y, и т.д.

Программа:

```
for i:=0 to m do A[i,0]:=0;
for j:=0 to n do A[0,j]:=0; for i:=1 to m do
for j:=1 to n do
if x[i]=y[i]
then A[i,j]:=A[i-1,j-1]+1
else A[i,j]:=max(A[i-1,j],A[i,j-1]);
writeln('Длина последовательности =',A[m,n]); d:=A[m,n]; i:=m; j:=n;
while (d<>0) do begin
while A[i,j-1]=d do j:=j-1;
while A[i-1,j]=d do i:=i-1;
write('Элемент последовательности номер', d,'есть', x[i]);
i:=i-1; j:=j-1; d:=d-1;
{переход к поиску предшествующего элемента в последовательности}
end;
```

Задача №17

Пусть x и y - две бинарных последовательности (т.е. элементы последовательностей - нули и единицы); x и y можно рассматривать как запись в двоичной форме некоторых двух натуральных чисел.
Найти максимальное число z, двоичную запись которого можно получить вычеркиванием цифр как из x, так и из y. Ответ выдать в виде бинарной последовательности.
В последовательностях x и y избавляемся от лидирующих нулей. Если хоть одна из последовательностей стала пустой, то z=0. Иначе крайние левые цифры и в x и в y есть 1. Запускаем на полученных последовательностях алгоритм задачи 24 (для получения максимального z необходимо, чтобы старшая цифра z была 1 и двоичная запись z имела максимальную длину), но при этом для каждого A[i,j] - длины последовательности, необходимо хранить добавочно и саму последовательность; при присвоении значения A[i,j] одновременно будем запоминать и последовательность максимальной длины. Если таких несколько, то берем из них последовательность с максимальным значением. Поэтому алгоритм задачи 23 запишется так:

Пусть S[0..m,0..n] - массив строк. В S[i,j] будет храниться подпоследовательность, длина которой A[i,j].

```
for i:=0 to m do A[i,0]:=0;
for j:=0 to n do A[j,0]:=0;
for i:=0 to m do
for j:=0 to n do S[i,j]:='';
for i:=1 to m do
for j:=1 to n do begin
if x[i]=y[j]
then begin
A[i,j]:=A[i-1,j-1]+1; S[i,j]:=S[i-1,j-1]+x[i];
end;
A[i,j]:=max(A[i,j],A[i-1,j],A[i,j-1]);
S[i,j]:=max(S[i,j],S[i-1,j],S[i,j-1]);
end;
```

write(A[m,n],'- длина',S[m,n]);

Попробуйте не заводить массив S[0..m,0..n], а обойтись одномерным массивом S[0..n].

Задача №18

Вокруг считающего стоит N человек, из которых выделен первый, а остальные занумерованы по часовой стрелке числами от 2 до N. Считающий, начиная с кого-то, ведет счет до M. Человек на котором остановился счет, выходит из круга. Счет продолжается со следующего человека и так до тех пор, пока не останется один человек.
Определить
a) номер оставшегося человека, если известно M и то, что счет начинался с первого человека;
b) номер человека с которого начинался счет, если известно M и номер оставшегося человека L.

Для решения этой задачи используем структуру данных 'список'. Каждый элемент списка указывает на следующий за ним элемент (другими словами - хранит информацию о расположении следующего элемента).
a). Заведем массив A из N ячеек - по числу людей в круге. В каждую из ячеек A[i] занесем номер стоящего следующим за i-ым человека. Первоначально A[i]=i+1, i=1,...,.N-1, A[N]=1. Начиная счет от текущего человека (человека с номером IndTek, с самого сначала IndTek=1) будем делать ход - отсчитывать M ячеек, двигаясь по кругу в поисках человека, который должен выйти из круга. С учетом определения массива A это будет выглядеть следующим образом:
{IndTek - это номер человека, с которого начинается счет} for i:=1 to M-1 do {Отсчитываем M человек, начиная с IndTek} begin IndPred:=IndTek; {в IndPred сохраняем номер текущего человека в круге} IndTek:=A[IndTek]; {и вычисляем номер следующего за ним} end;
После выполнения этого цикла в переменной IndTek будет номер человека, на котором остановился счет (т.е. номер человека, который покинул круг), а в переменной IndPred - номер предшествующего ему человека в круге.
Удалим человека с номером IndTek. Для этого мы просто изменим ссылку A[IndPred] у предшествующего ему человека так, чтобы он указывал не на IndTek, а на следующего за IndTek, т.е. на A[IndTek], и новый отсчет M-того человека начнем со следующего за удаленным:
A[IndPred]:=A[IndTek]; IndTek:=A[IndTek];{Новый номер начального человека}
Все вышеописанные операции мы будем повторять до тех пор, пока в круге не останется одного единственного человека, т.е. до тех пор, пока A[IndTek] не станет равным IndTek, что означает, что следующим за человеком с номером IndTek является он сам.
Полностью весь фрагмент программы будет выглядеть так:

```
{Инициализация массива A}
<P>for i:=1 to N-1 do
A[i]:=i+1;
A[N]:=1;
{IndTek - это номер человека, с которого начинается счет}
IndTek:=1;
while A[IndTek] <> IndTek do
begin
for i:=1 to M-1 do {Отсчитываем M человек, начиная с IndTek}
begin
```

```
IndPred:=IndTek; {в IndPred сохраняем номер текущего человека в круге}
IndTek:=A[IndTek]; {и вычисляем номер следующего за ним}
end;
A[IndPred]:=A[IndTek];
IndTek:=A[IndTek]; {Новый номер начального человека}
end;
```
writeln('Номер последнего оставшегося человека ',IndTek);

Решения пункта b).

Будем считать, что человек с номером N+i - это то же самое, что и человек с номером i. Предположим, что как и в пункте a), что мы начали счет с первого человека, выбрасывали из круга M-того, и последний оставшийся в круге человек имеет номер K. Очевидно, что если бы мы начали счет со второго человека, то последний оставшийся в круге человек имел бы номер K+1, ..., если с j-го, то K+j-1.

Если номер оставшегося человека L, то из равенства L=K+j-1 определяем номер j первого человека (если j<=0, то заменим j на j+N, считая как и раньше, что человек с номером N+j - это то же самое, что и человек с номером j).

Задача №19

Задана полоска длиной 2^k клеток и шириной в одну клетку. Полоску сгибают пополам так, чтобы правая половинка оказалась под левой. Сгибание продолжают до тех пор, пока сверху находится больше одной клетки. Необходимо пронумеровать клетки таким образом, чтобы после окончания сгибания полосы номера клеток в получившейся колонке были расположены в порядке 1,2,3,4,...,2^k.

Будем моделировать сложение полоски, затем пронумеруем получившуюся колонку клеток числами от 1 до 2^n, после чего распечатаем числа, приписанные первой, второй, ..., 2^n-ой клетке исходной полоски.

Сначала создаем двусвязный список, состоящий из 2^k элементов. Поле next будет указывать на элемент находящийся под данным, а поле last - на элемент находящийся над данным. Для верхнего элемента last=0, а для нижнего next=n+1, где n-общее число элементов. Вначале длина верхней полоски равняется n элементов, после первого сгибания их она станет n/2, после второго - n/4, и т.д. Пусть в данный момент длина верхней полоски есть cn элементов. Значит нам необходимо cn/2 правых элементов опустить под cn/2 левых. Для этого запустим цикл, который будет менять i от 1 до cn/2 и на каждом шаге помещать (cn-i+1)-ю колонку элементов под i-ю, при этом порядок элементов в (cn-i+1)-ой колонке меняется на противоположный. После каждого сгибания cn уменьшается вдвое. Так продолжаем до тех пор пока cn>1.

Программа

```
{$A-,B-,D-,E+,F-,I-,L-,N-,O-,R-,S-,V-}
{$M 65520,0,655360}
uses crt;
const
maxk = 13; {Максимальное значение для k}
type
input = record
last,next,new : word;
end;
var
k,i,j,n,cn,half : word;
m : array[1..1 shl maxk] of input;
Procedure concat(a,b : word);
var i,j,nj : word;
begin
i:=a;while m[i].next<>n+1 do i:=m[i].next;
j:=b; while m[j].next<>n+1 do j:=m[j].next;
while j<>0 do
begin
nj:=m[j].last; m[i].next:=j; m[j].last:=i; i:=j; j:=nj;
end;
m[i].next:=n+1;
end;
begin
Write('Enter k...');readln(k);
n:=1 shl k; {Определение длины полоски}
```

```
for i:=1 to n do{Начальные значения}
with m[i] do
begin
last:=0;
next:=n+1;
new:=0;
end;
cn:=n;
while cn>1 do {Сгибание полоски}
begin
half:=cn div 2;
for i:=1 to half do concat(i,cn-i+1);
cn:=half;
end;
j:=1;
for i:=1 to n do {Нумерация клеток}
begin
m[j].new:=i; j:=m[j].next;
end;
for i:=1 to n do write(m[i].new:5);
writeln;
```
end.

Попробуйте найти формулу и написать программу, которая без моделирования складывания полоски по номеру клетки выдавала бы ее номер.

Задача №20

Квадрат разбит на 4^k равновеликих квадратных клеток. Квадрат перегибается поочередно относительно вертикальной (правая половина подкладывается под левую) и горизонтальной (нижняя половина подкладывается под верхнюю) оси симметрии до тех пор, пока все клетки не будут расположены друг под другом. Требуется занумеровать клетки исходного квадрата таким образом, чтобы в результате выполнения операций перегиба номера клеток, расположенных друг под другом, образовали числовую последовательность 1,2,3,...,4^k, начиная с верхней клетки.

Решение этой задачи аналогично решению предыдущей задачи со сгибанием полоски с той лишь разницей, что тут будет поочередно производится два сгибания: сначала правая половина под левую, а затем нижнюю под верхнюю

```
{$A-,B-,D-,E+,F-,G-,I+,L-,N-,O-,R-,S-,V-,X-}
{$M 16384,0,655360}
uses crt;
const
maxk = 6;
type
input = record
last1,last2,next1,next2,new : word;
end;
var
k,i,j,i1,i2,j1,j2,nj1,nj2,n,n1,cn,half : word;
m : array[1..1 shl maxk,1..1 shl maxk] of input;
Procedure concat(a,b,c,d : word);
var i1,i2,j1,j2,nj1,nj2 : word;
begin
i1:=a; i2:=b;
while (m[i1,i2].next1<>n+1) and (m[i1,i2].next2<>n+1) do
begin
i1:=m[i1,i2].next1; i2:=m[i1,i2].next2;
end;
j1:=c; j2:=d;
while (m[j1,j2].next1<>n+1) and (m[j1,j2].next2<>n+1) do
begin
j1:=m[j1,j2].next1; j2:=m[j1,j2].next2;
end;
while j1<>0 do
begin
nj2:=m[j1,j2].last2; nj1:=m[j1,j2].last1;
m[i1,i2].next1:=j1; m[i1,i2].next2:=j2;
m[j1,j2].last1:=i1; m[j1,j2].last2:=i2;
i1:=j1; i2:=j2; j1:=nj1; j2:=nj2;
```

```
end;
m[i1,i2].next1:=n+1; m[i1,i2].next2:=n+1;
end;
begin
Write('Введите k...');readln(k);
n:=1 shl k; {Определение числа клеток в одной строке или столбце}
n1:=n*n;  {Определение числа клеток в матрице}
for i:=1 to n do
for j:=1 to n do with m[i,j] do
begin
last1:=0; next1:=n+1;
last2:=0; next2:=n+1;
new:=0;
end;
cn:=n;
while cn>1 do {сгибание матрицы}
begin
half:=cn div 2;
for i:=1 to half do {сгиб по вертикали}
for j:=1 to cn do concat(j,i,j,cn-i+1);
for i:=1 to half do {сгиб по горизонтали}
for j:=1 to half do concat(i,j,cn-i+1,j);
cn:=half;
end;
j1:=1;j2:=1;
for i:=1 to n1 do {Назначение клеткам новые номера}
begin
m[j1,j2].new:=i;
nj1:=m[j1,j2].next1; nj2:=m[j1,j2].next2;
j1:=nj1; j2:=nj2;
end;
for i:=1 to n do {Вывод результатов}
begin
for j:=1 to n do write(m[i,j].new:8);
writeln;
end;
end.
```

Задача №21

Имеется n черных и белых карточек, сложенных в стопку. Карточки раскладываются на стол в одну линию следующим образом: первая кладется на стол, вторая под низ стопки, третья- на стол, четвертая - под низ стопки и т.д., пока все карточки не будут выложены на стол. Каким должно быть исходное расположение карточек в стопке, чтобы разложенные на столе карточки чередовались по цвету: белая, черная, белая, черная и т.д.

Проще всего взять и промоделировать раскладку карточек: берем стопку из n черных и белых карточек и начинаем расклад как говорится в задаче - первая на стол, вторая под низ, третья на стол, и т.д. при этом первой выложенной карточке приписывается белый цвет, а каждой последующей - цвет, не совпадающий с цветом предыдущей карточки. Повторяет так, пока не найдем раскраску всех n карточек.

	1	2	3	4
A	2	3	4	0

Для этой задачи возьмем такую структуру данных как список. У списка есть начало и конец, и каждый элемент его указывает на следующий за ним элемент. Будем считать , что у последнего элемента списка указатель на следующий за ним равен 0. Список можно представлять массивом. Например, если в списке 4 элемента, то массив будет выглядеть так:
т.е. за первым элементом находится A[1]=2 элемент, ..., за 4-ым - A[4]=0 (т.к. 4-ый элемент списка последний).
Пусть у нас есть указатель на начальный элемент списка и на последний (BEG и FIN соответственно). В списке n элементов.
Рассмотрим процедуру удаления элемента из начала списка (это соответствует переносу карточки на стол):
BEG:=A[BEG]; {теперь новый первый элемент списка - второй элемент старого списка}

Рассмотрим операторы перестановки элемента из начала списка в конец (это соответствует перемещению карточки сверху стопки под низ ее):

```
A[FIN]:=BEG; {следующей за последним элементом - бывший первый}
FIN:=BEG; {меняем ссылку на последний элемент}
BEG:=A[BEG] {новый первый элемент}
A[FIN]:=0 {корректировка ссылки у последнего элемента}
Фрагмент программы будет выглядеть так:
for i:=1 to N-1 do A[i]:=i+1;
A[N]:=0; {установка ссылок в списке}
BEG:=1; FIN:=N;
COLOR:=1; {белый цвет = 1, черный = 0}
while A[BEG]<>0 do
{пока первый элемент не является} {одновременно и последним}
begin
BEFORE:=BEG; {сохраняем индекс начала списка}
BEG:=A[BEG]; {удаляем первый элемент из списка}
A[BEFORE]:=COLOR; {раскрашиваем удаленный элемент}
{в нужный цвет}
COLOR:=1-COLOR; {меняем цвет}
A[FIN]:=BEG;  {переставляем элемент из}
FIN:=BEG;  {начала списка в конец}
BEG:=A[BEG];
A[FIN]:=0
end;
A[BEG]:=COLOR;  {раскрашиваем последний элемент}
{списка}
for i:=1 to N do  {распечатка цветов}
if A[i]=0
then writeln('элемент',i,' - черный')
else writeln('элемент',i,' - белый');
```

Задача №22

Форма тела задана матрицей A размерности M x N. Элементы матрицы - натуральные числа. Элемент A (i,j) соответствует высоте горизонтальной квадратной площадки размера 1 x 1 относительно нижнего основания.Нижнее основание формы горизонтально.
Определить объем невытекшей воды, если
a) Тело опускается полностью в воду, затем поднимается;
После подъема вода останется только во внутреннем углублении, над элементом A(2,2)=1. Объем воды равен 1.
b) В позицию (i_0,j_0) выливается объем воды V.

Для реализации данного алгоритма нам понадобится структура данных "очередь". Очередь в программировании, как и очередь в магазине, имеет начало и конец. Если приходит новый элемент, то он становится в конец очереди, если необходимо взять элемент из очереди, то он берется из ее начала. Очередь будем представлять в виде массива. Пусть у нас есть индекс первого - BEG и последнего FIN элементов очереди (если очередь пуста, то BEG=FIN+1; сначала же BEG=1, FIN=0).
Очередь опишем так: var Queue=array[1..MaxQ] of element;
Тут MaxQ - максимальное число элементов в очереди, element какой-то тип данных. В задаче ниже в качестве element можно взять такой type element = record x: byte; y: byte; end; где element - это запись, состоящая из x-овой и y-овой координаты элемента матрицы A=array[0..M+1,0..N+1] of integer. Индексы матрицы принимают такое значение из-за того, что мы будем окаймлять матрицу нулевыми элементами (так будет проще проверять выливание за край фигуры).
С очередью можно проводить операции:
вставка в очередь InQueue,
удаление из очереди OutQueue.

```
Procedure InQueue (x : element);
begin
FIN:=FIN+1; {на первое свободное место}
Queue[FIN]:=x; {ставим элемент x}
end;
Procedure OutQueue (var x : element);
begin
x:=Queue[BEG]; {берем первый элемент}
```

```
BEG:=BEG+1; {и изменяем указатель}
{на следующий за ним}
```
end;
Находим максимальный элемент D в матрице A. Пока все элементы в матрице A не станут равны D, повторяем следующую последовательность действий:
а) Находим в матрице минимальный ненулевой элемент z (если их несколько, то берем любой из них), и заносим его в очередь (очередь сначала пустая). Если этот минимальный ненулевой элемент z=D, то СТОП.
Сначала считаем, что p=D - это минимальный граничный элемент области, заполненной элементами со значением z.
б) Для каждого еще не просмотренного элемента очереди (пусть в матрице этот элемент находится в позиции (i,j)) повторяем следующее:
Для каждого соседа (сверху, снизу, справа, слева) данного элемента (i,j) проводим проверку-просмотр:
ЕСЛИ (сосед <> z) ТО P=min{P,сосед} ИНАЧЕ ЕСЛИ сосед еще не просмотрен ТО координата соседа - в очередь (в очереди появился еще один непросмотренный элемент) т.е. мы ищем минимальный окаймляющий элемент области, заполненной элементами со значением z.
Фрагмент программы поиска:
```
var Delta = array [1..4,1..2] of
 integer = ((0,1),(1,0),(-1,0),(0,-1));
{Delta - возможные смещения соседних клеток от текущей клетки}
Current, neighbor : element;
z : integer;
....
{Будем обозначать то, что элемент в матрице уже просмотрен}
{умножением его на -1}
{минимальный ненулевой элемент матрицы имеет значение z}
while BEG<>FIN+1 do begin
OutQueue(Current);
for i:=1 to 4 do begin
neighbor.x:=Current.x+Delta[i,1], neighbor.y:=Current.y+Delta[i,2],
if A[neighbor.x,neighbor.y]=z
then InQueue(neighbor)
else p:=min(A[neighbor.x,neighbor.y],p); end;
```
end;
Если в очереди нет больше непросмотренных элементов, то, если p<z (случай, когда данная область имеет "слив" за границы матрицы) в матрице во все просмотренные элементы из очереди заносим значение D (делаем их равными накопленному значению, чтобы больше их не рассматривать).
Если же p>z, то высчитываем, сколько воды поместится в "бассейн" с глубиной дна z и с высотой ограждающего бордюра p:
Объем = (p-z)* количество просмотренных элементов в очереди.
Добавляем полученный объем к ранее найденному объему воды,заполняющему матрицу до высоты x. Заполняем в матрице элементы, соответствующие просмотренным элементам из очереди, значением p ("Доливаем" воды в "бассейн" до уровня p). Переход на пункт а).
Суммарный полученный объем воды и является искомым.
Окаймление матрицы, упомянутое выше, может выглядеть следующим образом: матрица [1..N, 1..M] окаймляется нулями так: вводятся дополнительные нулевые 0-ая и (N+1)-ая строки и 0-ой и (M+1)-ый столбцы
```
var A: array[0..N+1,0..M+1] of byte;
{ввод и окаймление нулями}
for i:=1 to N do begin
A[i,0]:=0; A[i,M+1]:=0;
for j:=1 to M do read(A[i,j]);
end;
for j:=0 to M+1 do begin
A[0,j]:=0; A[N+1,j]:=0;
end;
```

Задача №23

Ханойские башни

Есть три стержня A, B, и C. На стержень A надето N дисков, наверху самый маленький, каждый следующий диск больше предыдущего, а внизу самый большой. На другие стержни дисков не надето. Необходимо перенести диски со стержня A на стержень C, пользуясь стержнем B, как вспомогательным, так, чтобы диски на стержне C располагались в том же порядке, в каком они располагаются на диске A перед перемещением.

При перемещении никогда нельзя класть больший диск на меньший.

Рекурсивный метод

Для того, чтобы переложить всю пирамиду, надо сначала переложить все, что выше самого большого диска, с первого на вспомогательный стержень, потом переложить это самое большой диск с первого на третий стержень, а потом переложить оставшуюся пирамиду со второго на третий стержень, пользуясь первым стержнем, как вспомогательным.

```
    /*
      данная процедура переносит N дисков со стержня A на стержень C
      пользуясь B как вспомогательным, соблюдая правила
     */
    процедура "Перенести" (A, B, C, N)
    начало
      если N=1
      // Если диск всего один, то надо его перенести напрямую
      то
         снять диск со стержня A и положить на стержень C;
         возврат из процедуры;
      иначе
         // Переносим все диски, кроме самого большога со стежня
         // A на стержень B
         Перенести (A,C,B,N-1);
         // Переносим самый большой диск со стержня A на стержень C
         снять диск со стержня A и положить на стержень C;
         // Переносим все диски со стержня B на стержень C поверх
         // самого большого диска
         Перенести (B,A,C,N-1);
         возврат из процедуры;
      конец если;
    конец процедуры "Перенести";
```

```
procedure Solve(n: integer; a,b,c: Char);
begin
 if n > 0 then
 begin
  Solve(n-1, a, c, b);
  Writeln('Переместить диск со стержня ', a, ' на стержень ',b);
  Solve(n-1, c, b, a);
 end;
end;
begin
 Solve(4, '1','2','3');
end.
```

Нерекурсивный метод

Стержню, на котором диски находятся в начале, дадим номер 0; стержню, на который их надо перенести - номер 2; и, соответственно, оставшемуся стержню - номер 1. Пусть всего дисков N. Занумеруем диски в порядке увеличения радиуса числами 0,1,2,...,N-1. Как известно, задача решается за 2^{N-1} ходов. Занумеруем ходы числами 1,2,...,2^{N-1}. Любое натуральное число i единственным образом представимо в виде $i=(2t+1)*2^k$, где t и k - целые (т.е. как произведение

нечетного числа на некоторую степень двойки). Так вот, на i-ом ходе переносится диск номер k со стержня номер $((-1)^{N-k}*t \bmod 3)$ на стержень номер $((-1)^{N-k}*(t+1) \bmod 3)$.
Пример для N=4.

Ход	k(диск)	t	Со_стержня	На_стержень	Стержни
					|)!'
1	0	0	0	1	|)! '
2	1	0	0	2	|) ' !
3	0	1	1	2	|) !'
4	2	0	0	1	|) !'
5	0	2	2	0	|') !
6	1	1	2	1	|')!
7	0	3	0	1	|)!'
8	3	0	0	1	)!' |
9	0	4	1	2	)! |'
10	1	2	1	0	!) |'
11	0	5	2	0	!') |
12	2	1	1	2	!' |)
13	0	6	0	1	! ' |)
14	1	3	0	2	' |)!
15	0	7	1	2	|)!'

если представить что стержни, на которые одеваются диски, расположены в углах равностороннего треугольника, то самый маленький диск каждым нечетным ходом движется по (или против, это от первоначального кол-ва дисков зависит) часовой стрелки.
Все четные ходы определяются однозначно. Какой диск меньше - тот и перекладывать (иначе противоречит условию). Т.к. трогать диск 0 нельзя и класть больший на меньший тоже нельзя.
Отметим две закономерности:
На каждом нечетном ходу происходит перенос наименьшего диска.
Наименьший диск всегда переносится циклически: либо A-B-C-A-B-C-... (в случае четного количества дисков), либо A-C-B-A-C-B-... (в случае нечетного).
А посему получаем алгоритм:
1. Определяем число дисков, откуда находим как будет перемещаться наименьший диск (данный шаг делается в начале, притом один раз).
2. Смотрим номер хода: если нечетный - переносим наименьший диск в направлении, определенном в п.1. если четный - то возможный ход один единственный - берем наименьший из двух верхних дисков и переносим его.
Можно написать немного по другому:
Для N от 1 до 2^{k-1} выполнять
1. В двоичном представлении N найти самый правый ненулевой разряд. Обозначим номер этого разряда t.

2. Обозначим номер стержня, на котором находится диск t через i. Переместить диск t со стержня i на стержень $(i+(-1)^t) \bmod 3$.
Кстати, по номеру хода легко можно восстановить положение дисков на стержнях: после i-ого хода диск номер j находится на стержне номер $(-1)^{n-j}*((i \text{ div } 2^j)-(i \text{ div } 2^{j+1})) \bmod 3$.

Задача №24

Ребус. Написать программу, в которой требуется заменить буквы цифрами так, чтобы получилось верное равенство. Найти количество всех возможных вариантов решений и максимальное значение из них.

МУХА
+
МУХА
СЛОН

Формат выходного файла(Ребус.OUT):

4865
+
4865
9730

```
program muxa;
uses Crt;
var
i,a,b,c,d: integer;
m,s,max,kol: Integer;
f: text;
ss,as,bs,cs,ds: string;
label 1;
begin
ClrScr;
max := 0;
kol := 0;
for a:=1 to 4 do
begin
for b:=0 to 9 do
begin
for c:=0 to 9 do
begin
for d:=0 to 9 do
begin
if (a=b) or (a=c) or (a=d) or (b=c) or (b=d) or (c=d) then goto 1;
s:=2*(d+c*10+b*100+a*1000);
str(s,ss);
str(a,as);
str(b,bs);
str(c,cs);
str(d,ds);
if (ss[1]<>ss[2]) and (ss[1]<>ss[3]) and (ss[1]<>ss[4])
and (ss[2]<>ss[3]) and (ss[2]<>ss[4]) and(ss[3]<>ss[4])
   then begin
for i:=1 to 4 do
begin
   if (ss[i]=as) or (ss[i]=bs) or (ss[i]=cs) or
```

```
    (ss[i]=ds) then goto 1;
end;
if s>max then begin max:=s; end;
kol:=kol+1;
end;
1:
end;
end;
end;
end;
m:=max div 2;
Assign(f,'Ребус.out');
Rewrite(f);
WriteLn(' ',m);
WriteLn('+');
WriteLn(' ',m);
WriteLn('-------');
WriteLn(' ',max);
Writeln('Общее количество решений: ',kol);
WriteLn(f,' ',m);
WriteLn(f,'+');
WriteLn(f,' ',m);
WriteLn(f,'-------');
WriteLn(f,' ',max);
Writeln(f,'Общее количество решений: ',kol);
Close(f);
readln;
end.
```

Сортировка

```
{ сортировка пузырьковым методом}
      procedure Bubble(var item: DataArray; count:integer);
      var
        i,j: integer;
        x: DataItem;
      begin
        for i := 2 to count do
        begin
          for j := count downto i do
            if item[j-1]>item[j] then
            begin
              x := item[j-1];
              item[j-1] := item[j];
              item[j] := x;
            end;
         end;
      end; {конец сортировки пузырьковым методом}
    program SortDriver;
-------------------------------------------------------------------
    {эта  программа  будет  считывать 80 или меньше символов с
     дискового файла "test.dat",  отсортирует их и выдаст
     результат на экран дисплея }
     type
      DataItem = char;
      DataArray = array [1..80] of char;
```

```
  var
    test: DataArray;
    t, t2: integer;
    testfile: file of char;
  { сортировка пузырьковым методом}
   procedure Bubble(var item: DataArray; count:integer);
   var
    i,j: integer;
    x: DataItem;
   begin
    for i := 2 to count do
    begin
      for j := count downto i do
        if item[j-1]>item[j] then
        begin
          x := item[j-1];
          item[j-1] := item[j];
          item[j] := x;
        end;
    end;
   end;
   begin
    Assign(testfile, 'test.dat');
    Reset(testfile);
    t := 1;
   { считывание символов,которые будут сортироваться.}
    while not Eof(testfile) do begin
      read(testfile, test[t]);
      t := t+1;
    end;
   t := t-2; {скорректировать число считанных элементов }
   Bubble(test, t); { сортировать массив }
   { выдать отсортированный массив символов }
   for t2 := 1 to t do write(test[t2]);
   WriteLn;
  end.
-----------------------------------------------------------------
  { челночная сортировка является улучшенной версией сортиров-
              ки пузырьковым методом }
    procedure Shaker(var item: DataArray; count:integer);
    var
      j, k, l, r: integer;
      x: DataItem;
    begin
      l := 2; r := count; k := count;
      repeat
        for j := r downto l do
          if item[j-1] then
          begin    { обмен }
            x := item[j-1];
            item[j-1] := item[j];
            item[j] := x;
            k := j;
          end;
        l := k+1;
```

```
        for j := l to r do
          if item[j-1]>item[j] then
          begin   { обмен }
            x := item[j-1];
            item[j-1] := item[j];
            item[j] := x;
            k := j;
          end;
        r := k-1;
      until l>r
    end; { конец челночной сортировки  }
-------------------------------------------------------------------
      { сортировка выбором }
      procedure Selekt(var item: DataArray; count:integer);
      var
        i, j, k: integer;
        x: DataItem;
      begin
        for i := i to count-1 do
        begin
          k := i;
          x := item[i];
          for j := i+1 to count do { найти элемент с наименьшим
                значением }
          if item[j]<x then
          begin
             k := j;
             x := item[j];
          end;
          item[k] := item[i];  { обмен }
          item[i] := x;
        end;
    end; { конец сортировки выбором  }
-------------------------------------------------------------------
      { сортировка вставкой }
      procedure Inser(var item: DataArray; count:integer);
      var
        i, l: integer;
        x: DataItem;
      begin
        for i := 2 to count do
        begin
          x := item[i];
          j := i-1;
          while (x<item[j]) and (j>0) do
          begin
            item[j+1] := item[j];
            j := j-1;
          end;
          item[j+1] := x;
        end;
    end;  { конец сортировки вставкой }
-------------------------------------------------------------------
      { сортировка Шелла }
      procedure Shell(var item: DataArray; count:integer);
```

```
  const
    t = 5;
  var
    i, j, k, s, m: integer;
    h: array[1..t] of integer;
    x: DataItem;
  begin
    h[1]:=9; h[2]:=5; h[3]:=3; h[4]:=2; h[5]:=1;
    for m := 1 to t do
      begin
    k:=h[m];
    s:=-k;
    for i := k+1 to count do
    begin
      x := item[i];
      j := i-k;
      if s=0 then
      begin
        s := -k;
        s := s+1;
        item[s] := x;
      end;
      while (x<item[j]) and (j<count) do
        begin
          item[j+k] := item[j];
          j := j-k;
        end;
        item[j+k] := x;
      end;
    end;
  end; { конец сортировки Шелла }
-----------------------------------------------------------------
  { быстрая сортировка }
  procedure QuickSort(var item: DataArray; count:integer);
    procedure qs(l, r: integer; var it: DataArray);
      var
      i, j: integer;
      x, y: DataItem;
    begin
      i:=l; j:=r;
      x:=it[(l+r) div 2];
      repeat
        while it[i]<x do i := i+1;
        while x<it[j] do j := j-1;
        if y<=j then
        begin
          y := it[i];
          it[i] := it[j];
          it[j] := y;
          i := i+1; j := j-1;
        end;
      until i>j;
      if l<j then qs(l, j, it);
      if l<r then qs(i, r, it)
    end;
```

```
    begin
      qs(1, count, item);
    end; { конец быстрой сортировки }
-----------------------------------------------------------------
    type
      DataItem = string[80];
      DataArray = array [1..80] of DataItem;
    { алгоритм быстрой сортировки для символьных строк }
    procedure QsString(var item: DataArray; count:integer);
      procedure qs(l, r: integer; var it:DataArray);
        var
        i, l: integer;
        x, y: DataItem;
      begin
        i := l; j := r;
        x := it[(l+r) div 2];
        repeat
          while it[i] < x do i := i+1;
          while x < it[j] do j := j-1;
          if i<=j then
          begin
            y := it[i];
            it[i] := it[j];
            it[j] := y;
            i := i+1; j := j-1;
          end;
        until i>j;
        if l<j then qs(l, j, it);
        if l<r then qs(i, r, it);
      end;
    begin
      qs(1, count, item);
    end; { конец быстрой сортировки }
-----------------------------------------------------------------
      { быстрая сортировка записей с почтовым адресом }
      procedure QsRecord(var item: DataArray; count:integer);
        procedure qs(l, r:integer; var it:DataArray);
          var
            i, j: integer;
            x, y: DataItem;
          begin
            i := l; j := r;
            x := it[(l+r) div 2];
            repeat
              while it[i].name < x.name do i := i+1;
              while x.name < it[j].name do j := j-1;
            if i<=j then
            begin
              y := it[i];
              it[i] := it[j];
              it[j] := y;
              i := i+1; j := j-1;
            end;
          until i>j;
          if l<j then qs(l, j, it);
```

```
      if l<r then qs(i, r, it)
    end;
  begin
    qs(1, count, item);
  end; {конец быстрой сортировки записей}
-----------------------------------------------------------------
  { пример программы сортировки списка почтовых адресов }
    programm MlistSort;
    type
      address = record
        name: string[30];
        street: string[40];
        sity: string[20];
        state: string[2];
        zip: string[9];
      end;
      str80 = string[80];
      DataItem = addres;
      DataArray = array [1..80] of DataItem
      recfil = file of DataItem
      var
        test: DataItem;
        t, t2:integer;
        testfile: recfil;
              { найти запись в файле }
    function Find(var fp:recfil; i:integer): str80
    var
      t:address;
    begin
      i := i-1;
      Seek(fp, i)
      Read(fp, t)
      Find := t.name;
    end;
    procedure QsRand(var var fp:recfil; count:integer)
      procedure Qs(l, r:integer)
        var
          i, j, s:integer ;
          x, y, z:DataItem;
          begin
            i := l; j := r;
            s := (l+r) div 2;
            Seek(fp,s-1); { получить запись }
            Reed(fp,x);
            repeat
              while Find(fp, i) < x.name do i := i+1;
              while x.name < Find(fp, j) do j := j-1;
              if i<=j then
              begin
                Seek(fp,i-1);  Reed(fp,y);
                Seek(fp,j-1);  Reed(fp,z);
                Seek(fp,j-1);  Write(fp,y);
                Seek(fp,i-1);  Write(fp,z);
                i := i+1; j := j-1;
              end;
```

```
            until i>y;
            if l<j then qs(l, j)
            if l<r then qs(i, r)
          end;
        begin
          qs(1,count);
        end; {конец быстрой сортировки файла произвольного доступа}
        begin
          Assign(testfile, 'rectest.dat');
          Reset(testfile);
          t := 1;
          while not EOF(testfile) do begin
            Read(testfile,test); { подсчет числа записей в файле}
          t := t+1;
          end;
          t := t-1;
          QsRand(testfile,t)
        end.
-----------------------------------------------------------------
    { функция  "Find" используется в сортировке методом
     слияния для считывания из файла конкретной записи.}
    function Find(var fp:filtype; i:integer):DataItem;
    var
      t:DataItem;
    begin
      Seek(fp, i-1);
      Read(fp, t);
      Find := t;
    end;
    procedure Mergesort(var fp: filetype; count:integer);
       var
         i, j, k, l, t, h, m, p, q, r: integer;
        ch1, ch2:DataItem
        up: Boolean;
      begin
        up := TRUE;
        p := 1;
        repeat
          h := 1; m := count;
          if up then
          begin
            i := 1; j := count; k := count+1; l := 2*count;
          end else
          begin
            k := 1; l := count; i := count+1; j := 2*count;
          end;
          repeat
            if m>=p then q := p else q := m;
            m := m-q;
            if m>=p then r := p else r := m;
            m := m-r;
            while (q<>0) and (r<>0) do
            begin
              if Find(fp,i) < Find(fp,j) then
              begin
```

```
              Seek(fp, i-1); Read(fp,ch2);
              Seek(fp, k-1); Write(fp,ch2);
              k := k+h; i := i+1; q := q-1;
            end else
            begin
              Seek(fp, j-1); Read(fp,ch2);
              Seek(fp, k-1); Write(fp,ch2);
              k := k+h; j := j-1; r := r-1;
            end;
          end;
          while r<>0 do
          begin
             Seek(fp, j-1); Read(fp,ch2);
             Seek(fp, k-1); Write(fp,ch2);
             k := k+h; j := j-1; r := r-1;
          end;
          while q<>0 do
          begin
             Seek(fp, i-1); Read(fp,ch2);
             Seek(fp, k-1); Write(fp,ch2);
             k := k+h; i := i+1; q := q-1;
          end;
            h := -1; t := k;
            k := l;
            l := t;
          until m = 0:
          up := not up;
          p := p*2;
        until  p >= count;
        if not up then
          for i := 1 to count do
          begin
            Seek(fp, i-1+count); Read(fp,ch2);
            Seek(fp, i-1); Write(fp,ch2);
          end;
        end; { конец сортировки методом слияния }
-------------------------------------------------------------------
  function SeqSearch(item: DataArray; count:integer;
                     key:DataItem):integer;
  var
    t:integer;
  begin
    t:=1;
    while (key<>item[t]) and (t<=count) t:=t+1;
    if t>count then SeqSearch:=0
    else SeqSearch:=t;
  end; { конец последовательного поиска }
-------------------------------------------------------------------
  function BSearch (item: DataArray; count:integer;
                    key:DataItem):integer;
  var
    low, high, mid: integer;
    found:boolean;
  begin
    low:=1; high:=count;
```

```
  found:=false;        { не найден }
  while (low<=high) and (not found) do
  begin
    mid:=(low+high) div 2;
    if key<item[mid] then high:=mid-1
    else if key>item[mid] then low:=mid+1
    else found:=true;  { найден }
  end;
  if found then BSearch:=mid
  else BSearch:=0;  { не найден }
end; { конец поиска }
```

Методы программрования: переборные алгоритмы

Данная статья открывает цикл работ, посвященных не особенностям какого-то отдельного языка программирования (например Паскаля), а общим ИДЕЯМ и МЕТОДАМ разработки алгоритмов. Тем не менее, опираться мы все равно будем на теория, числа, оптимальные алгоритмы, вычислительная, определение, который вы уже знаете. Первоначальный вариант любого алгоритма мы будем записывать на псевдокоде - языке, который занимает промежуточное положение между нашим обычным языком и языками программирования. Он не имеет каких-то жестких правил и требований, т.к. предназначен прежде всего для человека, а не компьютера. Это позволит нам избавиться от излишней детализации алгоритма на раннем этапе разработки и сразу выразить его основную идею. Превратить этот псевдокод в программу на Паскале задача совсем несложная - как это делать вы быстро поймете.
Основные идеи первого задания - ПЕРЕБОР, РЕКУРСИЯ, ПЕРЕБОР С ОТХОДОМ НАЗАД. Этими понятиями должен хорошо владеть каждый программист. Кроме того, переборные задачи составляют значительную долю всех школьных олимпиад по информатике.

1. Порождение и перебор комбинаторных объектов

Во многих прикладных задачах требуется найти оптимальное решение среди очень большого (но конечного!) числа вариантов. Иногда удается построить это решение сразу, но в большинстве случаев единственный способ его отыскать состоит в переборе ВСЕХ возможных вариантов и сравнении их между собой. Поэтому так важно для нас научиться строить алгоритмы ПЕРЕБОРА различных комбинаторных объектов - последовательностей, перестановок, подмножеств и т.д.
Схема перебора всегда будет одинакова:
- во-первых, надо установить ПОРЯДОК на элементах, подлежащих перечислению (в частности, определить, какой из них будет первым, а какой последним);

- во-вторых, научиться переходить от произвольного элемента к НЕПОСРЕДСТВЕННО СЛЕДУЮЩЕМУ за ним (т.е. для заданного элемента x1 строить такой элемент x2, что x1<x2 и между x1 и x2 нет других элементов).
Наиболее естественным способом упорядочения составных объектов является ЛЕКСИКОГРАФИЧЕСКИЙ порядок, принятый в любом словаре (сначала сравниваются первые буквы слов, потом вторые и т.д.) - именно его мы и будем чаще всего использовать. А вот процедуру получения следующего элемента придется каждый раз изобретать за- ново. Пока запишем схему перебора в таком виде:

```
X:=First;
    while X<>Last do Next(X);
```

где First - первый элемент; Last - последний элемент; Next - процедура получения следующего элемента.

1.1. Последовательности

Напечатать все последовательности длины N из чисел 1,2,...,M.

First = (1,1,...,1) Last = (M,M,...,M)
Всего таких последовательностей будет M^N (докажите!). Чтобы понять. как должна действовать процедура Next, начнем с примеров. Пусть N=4,M=3. Тогда:
Next(1,1,1,1) -> (1,1,1,2) Next(1,1,1,3) -> (1,1,2,1) Next(3,1,3,3) -> (3,2,1,1)
Теперь можно написать общую процедуру Next:

```
procedure Next;
  begin
    {найти i: X[i]<M,X[i+1]=M,...,X[N]=M};
    X[i]:=X[i]+1;
    X[i+1]:=...:=X[N]:=1
end;
```

Если такого i найти не удается, то следующей последовательности нет - мы добрались до последней (M,M,...,M). Заметим также, что если бы членами последовательности были числа не от 1 до M, а от 0 до M-1, то переход к следующей означал бы прибавление 1 в M-ичной системе счисления. Полная программа на Паскале выглядит так:

```
program Sequences;
  type Sequence=array [byte] of byte;
  var M,N,i:byte;
      X:Sequence;
      Yes:boolean;
  procedure Next(var X:Sequence;var Yes:boolean);
    var i:byte;
  begin
    i:=N;
    {поиск i}
    while (i>0)and(X[i]=M) do begin X[i]:=1;dec(i) end;
    if i>0 then begin inc(X[i]);Yes:=true end
    else Yes:=false
  end;
begin
  write('M,N=');readln(M,N);
  for i:=1 to N do X[i]:=1;
  repeat
    for i:=1 to N do write(X[i]);writeln;
    Next(X,Yes)
  until not Yes
end.
```

1.2. Перестановки

Напечатать все перестановки чисел 1..N (то есть последовательности длины N, в которые каждое из чисел 1..N входит ровно по одному разу).

First = (1,2,...,N) Last = (N,N-1,...,1)
Всего таких перестановок будет N!=N*(N-1)*...*2*1 (докажите!). Для составления алгоритма Next зададимся вопросом: в каком случае i-ый член перестановки можно увеличить, не меняя предыдущих? Ответ: если он меньше какого-либо из следующих членов (членов с номерами больше i).
Мы должны найти наибольшее i, при котором это так, т.е. такое i, что X[i]<X[i+1]>...>X[N] (если такого i нет, то перестановка последняя). После этого X[i] нужно увеличить минимально возможным способом, т.е. найти среди X[i+1],...,X[N] наименьшее число, большее его. Поменяв X[i] с ним, остается расположить числа с номерами i+1,...,N так, чтобы перестановка была наименьшей, то есть в возрастающем порядке. Это облегчается тем, что они уже расположены в убывающем порядке:

```
procedure Next;
begin
  {найти i: X[i]<X[i+1]>X[i+2]>...>X[N]};
  {найти j: X[j]>X[i]>X[j+1]>...>X[N]};
  {обменять X[i] и X[j]};
  {X[i+1]>X[i+2]>...>X[N]};
```

```
            {перевернуть X[i+1],X[i+2],...,X[N]};
        end;
```

Теперь можно написать программу:

```
program Perestanovki;
  type Pere=array [byte] of byte;
  var N,i,j:byte;
      X:Pere;
      Yes:boolean;
  procedure Next(var X:Pere;var Yes:boolean);
     var i:byte;
     procedure Swap(var a,b:byte);  {обмен переменных}
       var c:byte;
     begin c:=a;a:=b;b:=c end;
  begin
     i:=N-1;
     {поиск i}
     while (i>0)and(X[i]>X[i+1]) do dec(i);
     if i>0 then
       begin
         j:=i+1;
         {поиск j}
         while (j<N)and(X[j+1]>X[i]) do inc(j);
         Swap(X[i],X[j]);
         for j:=i+1 to (N+i) div 2 do Swap(X[j],X[N-j+i+1]);
         Yes:=true
       end
     else Yes:=false
  end;
begin
  write('N=');readln(N);
  for i:=1 to N do X[i]:=i;
  repeat
     for i:=1 to N do write(X[i]);writeln;
     Next(X,Yes)
  until not Yes
end.
```

1.3. Разбиения

Перечислить все разбиения целого положительного числа N на целые положительные слагаемые (разбиения, отличающиеся лишь порядком слагаемых, считаются за одно).
Пример: N=4, разбиения: 1+1+1+1, 2+1+1, 2+2, 3+1, 4.
First = (1,1,...,1) - N единиц Last = (N)

Чтобы разбиения не повторялись, договоримся перечислять слагаемые в невозрастающем порядке. Сказать, сколько их будет всего, не так-то просто (см.следующий пункт). Для составления алгоритма Next зададимся тем же вопросом: в каком случае i-ый член разбиения можно увеличить, не меняя предыдущих?
Во-первых, должно быть X[i-1]>X[i] или i=1. Во-вторых, i должно быть не последним эле ментом (увеличение i надо компенсировать уменьшением следующих). Если такого i нет, то данное разбиение последнее. Увеличив i, все следующие элементы надо взять минимально возможными, т.е. равными единице:

```
procedure Next;
  begin
     {найти i: (i<L) and ( (X[i-1]>X[i]) or (i=1) )}
     X[i]:=X[i]+1;
     { L:= i + X[i+1]+...+X[L] - 1 }
     X[i+1]:=...:=X[L]:=1
end;
```

Через L мы обозначили количество слагаемых в текущем разбиении (понятно, что 1<=L<=N).
Программа будет выглядеть так:

```
program Razbieniya;
  type Razb=array [byte] of byte;
  var N,i,L:byte;
      X:Razb;
  procedure Next(var X:Razb;var L:byte);
```

```
    var i,j:byte;
        s:word;
  begin
    i:=L-1;s:=X[L];
    {поиск i}
    while (i>1)and(X[i-1]<=X[i]) do begin s:=s+X[i];dec(i) end;
    inc(X[i]);
    L:=i+s-1;
    for j:=i+1 to L do X[j]:=1
  end;
begin
  write('N=');readln(N);
  L:=N; for i:=1 to L do X[i]:=1;
  for i:=1 to L do write(X[i]);writeln;
  repeat
    Next(X,L);
    for i:=1 to L do write(X[i]);writeln
  until L=1
end.
```

1.4. Подсчет количеств

Иногда можно найти количество объектов с тем или иным свойством, не перечисляя их. Классический пример: C(n,k) - число всех k-элементных подмножеств n-элементного множества - можно найти, заполняя таблицу значений функции C по формулам:
C(n,0) = C(n,n) = 1 (n>=1) C(n,k) = C(n-1,k-1) + C(n-1,k) (n>1, 0<k<n);
или по формуле n!/(k!*(n-k)!) (первый способ эффективнее, если надо вычислить много значений C(n,k)).
Попробуем посчитать таким способом количество разбиений из пункта 1.3. Обозначим через R(N,k) (при N>=0, k>=0) число разбие- ний N на целые положительные слагаемые, не превосходящие k (при этом R(0,k) считаем равным 1 для всех k>=0). Очевидно, что число R(N,N) и будет искомым. Все разбиения N на слагаемые, не превосходящие k, разобьем на группы в зависимости от максимального слагаемого (обозначим его i).
Число R(N,k) равно сумме (по всем i от 1 до k) количеств разбиений со слагаемыми не больше k и максимальным слагаемым, равным i. А разбиения N на слагаемые не более k с первым слагаемым, равным i, по существу представляют собой разбиения n-i на слагаемые, не превосходящие i (при i<=k). Так что
R(N,k) = R(N-1,1)+R(N-2,2)+...+R(N-k,k).

2. Рекурсия

Вы уже знаете, что рекурсивными называются процедуры и функции, которые вызывают сами себя. Рекурсия позволяет очень просто (без использования циклов) программировать вычисление функций, заданных рекуррентно, например факториала f(n)=n!:
f(0)=1 f(n)=n*f(n-1).
Оказывается, рекурсивные процедуры являются удобным способом порождения многих комбинаторных объектов. Мы заново решим здесь несколько задач предыдущей главы и вы убедитесь, что запись многих алгоритмов значительно упростится благодаря использованию рекурсии.
Примечание: рекурсия, безусловно, весьма удобна, однако часто требует громадное количество ресурсов. В частности, программа для факториала на 7-мом десятке завесит самый быстрый Pentium - III+. Из-за времени выполнения. О том, как этого избежать, можно прочитать в статье Динамическое программирование.

2.1. Факториал

Еще раз напомним рекурсивный алгоритм вычисления факториала:

```
program Factorial;
  var N:word;
  function F(n:word):longint;
```

```
  begin
    if n=0 then F:=1 else F:=n*F(n-1)
  end;
begin
  write('N=');readln(N);
  writeln('N!=',F(N))
end.
```

2.2. Ханойская башня

Игра "Ханойская башня" состоит в следующем. Есть три стержня. На первый из них надета пирамидка из N колец (большие кольца снизу, меньшие сверху). Требуется переместить кольца на другой стержень. Разрешается перекладывать кольца со стержня на стержень, но класть большее кольцо поверх меньшего нельзя. Составить программу, указывающую требуемые действия.

Напишем рекурсивную процедуру перемещения M верхних колец с A-го стержня на B-ый в предположении, что остальные кольца больше по размеру и лежат на стержнях без движения:

```
procedure Move(M,A,B:integer);
  var C:integer;
begin
  if M=1 then begin writeln ('сделать ход ',A,'->',B) end
  else
    begin
      C:=6-A-B; {C - третий стержень: сумма номеров равна 6}
      Move(M-1,A,C);
      Move(1,A,B);
      Move(M-1,C,B)
    end
end;
```

Сначала переносится пирамидка из M-1 колец на третий стержень C. После этого M-ое кольцо освобождается, и его можно перенести на B. Остается перенести пирамиду из N-1 кольца с C на B. Чем это проще первоначальной задачи? Тем, что количество колец стало на единицу меньше. Теперь основную программу можно записать в несколько строк:

```
program Hanoi;
  var N:integer;
  procedure Move(M,A,B:integer);
         .............
begin
  write('N=');readln(N);
  Move(N,1,2)
end.
```

Если вы владеете основами компьютерной графики, можете попробовать "нарисовать" каждый ход на экране.

Таким образом, ОСНОВНАЯ ИДЕЯ любого рекурсивного решения - свести задачу к точно такой же, но с меньшим значением параметра. При этом какое-то минимальное значение параметра (например, 1 или 0) должно давать решение без рекурсивного вызова - иначе программа "зациклится" (последовательность рекурсивных вызовов будет бесконечной). Это напоминает метод математической индукции в математике. В некоторых задачах удобно наоборот, увеличивать значение параметра при рекурсивном вызове. Тогда, естественно, "безрекурсивное" решение должно предусматриваться для некоторого максимального значения параметра. Попробуем использовать эту идею для перебора комбинаторных объектов.

2.3. Последовательности (рекурсивный алгоритм)

Задача та же, что в пункте 1.1. Опишем рекурсивную процедуру Generate(k), предъявляющую все последовательности длины N из чисел 1,...,M, у которых фиксировано начало X[1],X[2],...,X[k]. Понятно, что при k=N мы имеем тривиальное решение: есть только одна такая последовательность - это она сама. При k<N будем сводить задачу к k+1:

```
procedure Generate(k:byte);
  var i,j:byte;
begin
```

```
    if k=N then
      begin for i:=1 to N do write(X[i]);writeln end
    else
      for j:=1 to M do
        begin X[k+1]:=j; Generate(k+1) end
end;
```

Основная программа теперь выглядит очень просто:

```
program SequencesRecursion;
  type Sequence=array [byte] of byte;
  var M,N:byte;
      X:Sequence;
  procedure Generate(k:byte);
       ............
begin
  write('M,N=');readln(M,N);
  Generate(0)
end.
```

2.4. Перестановки (рекурсивный алгоритм)

Задача та же, что в пункте 1.2. Опишем рекурсивную процедуру Generate(k), предъявляющую все перестановки чисел 1,...,N, у которых фиксировано начало X[1],X[2],...,X[k]. После выхода из процедуры массив X будут иметь то же значение, что перед входом (это существенно!). Понятно, что при k=N мы снова имеем только тривиальное решение - саму перестановку. При k<N будем сводить задачу к k+1:

```
  procedure Generate(k:byte);
    var i,j:byte;
    procedure Swap(var a,b:byte);
      var c:byte;
    begin c:=a;a:=b;b:=c end;
  begin
    if k=N then
      begin for i:=1 to N do write(X[i]);writeln end
    else
      for j:=k+1 to N do
        begin
          Swap(X[k+1],X[j]);
          Generate(k+1);
          Swap(X[k+1],X[j])
        end
end;
```

Основная программа:

```
program PerestanovkiRecursion;
  type Pere=array [byte] of byte;
  var N,i,j:byte;
      X:Pere;
  procedure Generate(k:byte);
      ..............
begin
  write('N=');readln(N);
  for i:=1 to N do X[i]:=i;
  Generate(0)
end.
```

Чтобы до конца разобраться в этой непростой программе, советуем выполнить ее на бумаге при N=3. Обратите внимание, что порядок вывода перестановок не будет лексикографическим!

3. Перебор с отходом назад

Как вы уже поняли, перебор комбинаторных объектов - задача весьма трудоемкая даже для компьютера. Например, перестановок из восьми чисел будет 8! = 40320 - число немаленькое. Поэтому в любой переборной задаче главная цель состоит в СОКРАЩЕНИИ ПЕРЕБОРА, т.е. в исключении тех объектов, которые заведомо не могут стать решением задачи. Предположим, что нам требуется рассмотреть только те перестановки, для которых сумма |X[i]-i| равна 8. Понятно, что их будет гораздо меньше: например, все перестановки, начинающиеся на 8,7,... рассматривать

не нужно! Как можно модифицировать наш переборный алгоритм в этом случае? Если на каком-то этапе сумма

|X[1]-1| + |X[2]-2| + ... + |X[k]-k|

уже больше 8, то рассматривать все перестановки, начинающиеся на X[1],...,X[k] уже не нужно - следует вернуться к X[k] и изменить его значение ("отойти назад" - отсюда название метода).
Для такой ситуации мы рассмотрим один общий метод, который почти всегда позволяет значительно сократить перебор. Пусть искомое решение находится среди последовательностей вида
X[1],...,X[N],
где каждое X[i] выбирается из некоторого множества вариантов A[i]. Предположим мы уже построили начало этой последовательности X[1],...,X[k] (k<N) и хотим продолжить его до решения.
Предположим также, что у нас есть некоторый простой метод P(X[1],...,X[k]), который позволяет получить ответ на вопрос: можно продолжить X[1],...,X[k] до решения (true) или нет (false). Заметим, что значение true еще НЕ ГАРАНТИРУЕТ существование такого продолжения, но зато значение false ГАРАНТИРУЕТ непродолжаемость ("не стоит дальше и пробовать"). Получаем простую рекурсивную процедуру ПЕРЕБОРА С ОТХОДОМ НАЗАД:

```
procedure Backtracking(k);
begin
   for (y in A[k]) do
     if P(X[1],...,X[k-1],y) then
       begin
         X[k]:=y;
         if k=N then {X[1],...,X[N] -решение}
         Backtracking(k+1)
       end
end;
```

ОЛИМПИАДЫ ПО ИНФОРМАТИКЕ ЗАДАЧИ И РЕШЕНИЯ

ЧАСТЬ 2

1. Простые задачи.

Каждая из проводимых олимпиад обычно включает задачи, алгоритм решения которых несложный (линейный или разветвляющийся), но от участника требуется грамотно сформулировать условия или определить расчетную формулу.

Задача 1.1

Проверить, поместится ли на диске компьютера музыкальная композиция, которая длится **m** минут и **n** секунд, если свободное дисковое пространство 6 мегабайт, а для записи одной секунды звука необходимо 16 килобайт.

Задача 1.2

Координаты двух полей шахматной доски заданы в виде двух пар чисел x_1 , y_1 и x_2 , y_2. На первом поле стоит ферзь, на втором - конь. Определить, бьет ферзь коня, конь - ферзя, или фигуры не угрожают друг другу.

Задача 1.3

Для нормального разведения золотых рыбок необходимо, чтобы на каждую рыбку в аквариуме приходилось не менее 3-х литров воды. По известным объему аквариума и количеству рыбок, в нем содержащихся, определить, является ли аквариум "перенаселенным" или нет, и указать количество рыбок, которых в случае перенаселенности необходимо поместить в другой аквариум.

Задача 1.4

По данным статистического исследования, производительность птицефермы такова, что каждые полторы курицы за полтора дня сносят полтора яйца. Написать программу, которая позволяет рассчитать, сколько яиц (без десятых долей) снесут N кур за d дней (N и d - целые числа).

Задача 1.5

Показать, что любую сумму, большую 7 копеек, можно выплатить, используя только 3-х и 5-ти копеечные монеты. (То есть, для любого целого $N>7$ найти такие целые числа x и y, что $3x+5y=N$).

Задача 1.6

Написать программу, которая переводит величину, заданную в метрах и сантиметрах, в футы и дюймы. 1 фут = 30,48 см; 1 дюйм = 2,54 см. Если величина не переводится нацело, округлить число дюймов до ближайшего целого. Учесть, что 1 фут равен 12 дюймам.

Задача 1.7

Диаметр колеса автомобиля 80 см. Колесо потребует замены через 200 000 оборотов. Определить, доедет ли колесо от города Саратова до города N-ска, если расстояние между ними x километров.

Задача 1.8

Количество цветов, которые может воспроизводить видеоадаптер, определяется количеством бит, отводимых в видеопамяти ПК для описания одной точки. Например, 2 бита позволяют воспроизводить 4 цвета, 4 бита - 16 цветов и т.д. Видеопамять содержит информацию о цвете каждой точки экрана. Определить, может ли картинка на экране монитора с разрешающей способностью видеоадаптера 800x600 содержать 2048 цвета, если видеопамять ПК - 4 мегабайта.

Задача 1.9

Известна цена монитора Samsung SyncMaster в январе 2000 г. и январе 2001 г. Ответьте на вопрос: "Произошло ли удешевление или нет? На сколько процентов изменилась цена изделия?"

Задача 1.10

В совпадающих по типу переменных a и b хранятся некоторые числовые значения. Поменять местами значения этих переменных, не используя третьей дополнительной переменной.

2. Задачи на использование циклов и стандартных алгоритмов.

К данному классу задач можно отнести те задачи, в которых используются стандартные алгоритмы для вычисления суммы, произведения, максимального или минимального элемента в какой-либо числовой последовательности и т. д.

Задача 2.1

Для любого целого числа $N>7$ найти все такие пары целых чисел x и y, что $3x+5y=N$.

Задача 2.2

Для заданного числа x распечатать числовую последовательность:
sin(x), sin(sin(x)), sin(sin(sin(x))), …
Вычисления прекратить, когда очередной элемент последовательности станет по модулю меньше, чем 10^{-2}.

Задача 2.3

Подсчитать количество сочетаний из N элементов по M (N>M). Для подсчета количества сочетаний используется формула:

$$C_N^M = \frac{N!}{M!(N-M)!}, \qquad N! = \begin{cases} 1, & N = 0 \\ 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N, & N - \textit{натуральное число} \end{cases}$$

Массивы.

Задача 2.4

Задан числовой массив A(50). Определить, каких элементов больше в этом массиве: положительных или отрицательных.

Задача 2.5

Информация о температуре воздуха и о количестве осадков в течение месяца задана в виде двух одномерных массивов.
Определить, сколько выпало осадков в виде снега и сколько - в виде дождя. (Для определенности предполагается, что при 0 градусов идет дождь).

Задача 2.6

В расписании движения поездов указано время отправления 12 пригородных поездов со станции г. Урюпинска.
Определить количество поездов, отправляющихся со станции в период времени с 16.00 до 19.30. (Время отправления поездов задается одномерным массивом.)

Задача 2.7

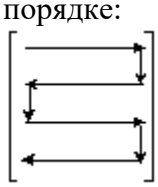
Для некоторой группы учащихся (всего в группе 25 чел.) известны данные о скорости ввода текстовой информации с клавиатуры (количество введенных символов за 10 минут).
Требуется составить отчет в следующем виде: напечатать фамилию и скорость ввода самого результативного учащегося; среднюю скорость ввода в данной группе; фамилии тех учащихся, скорость ввода которых ниже средней.

Задача 2.8

Из одного порта в другой необходимо перевезти 15 различных грузов. Грузоподъемность судна, на котором будет проходить перевозка, 50 тонн. Грузы пронумерованы, и информация о массах грузов хранится в массиве M(15).
Определить, сколько рейсов необходимо сделать судну, если грузы неделимы и могут перевозиться только подряд в порядке их нумерации. (Предполагается, что масса отдельного груза не превышает 50 тонн).

Задача 2.9

Заполнить квадратную матрицу размера n на n натуральными числами от 1 до n^2 в указанном порядке:



Например: $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 8 & 7 & 6 & 5 \\ 9 & 10 & 11 & 12 \\ 16 & 15 & 14 & 13 \end{bmatrix}$

Вычисление непрерывных дробей и радикалов.

Задача 2.10

Вычислить значение арифметического выражения:

$$R = \sqrt{98 + \sqrt{95 + \sqrt{92 + \ldots \sqrt{5 + \sqrt{2}}}}}$$

Задача 2.11

Вычислить значение дроби:

$$Q = \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{3 + \cfrac{1}{\ldots\ldots\ldots + \cfrac{1}{98 + \cfrac{1}{99 + \cfrac{1}{100}}}}}}}$$

Числа и числовые последовательности.

Задача 2.12

Распечатать числовую последовательность, которая задается по следующим правилам:
- первое число последовательности - произвольное нечетное число от 3 до 99;
- каждый следующий элемент последовательности определяется через предыдущий элемент **p**, и равен

3p+1, если **p** нечётное число,
p/2, если **p** чётное число.

Например:
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
Вычисления прекратить, когда очередной элемент последовательности станет равен 1.
(Известно, что в любой такой последовательности рано или поздно встречается 1.)

Задача 2.13

Распечатать числовую последовательность, которая задается по следующим правилам:
- первое число последовательности - натуральное число, кратное 3 (входной параметр задачи);
- каждый последующий элемент равен сумме кубов цифр предыдущего.

Например:
33
$3^3+3^3=54$
$5^3+4^3=189$
$1^3+8^3+9^3=1242$
$1^3+2^3+4^3+2^3=81$
$8^3+1^3=513$
$53+13+33=153$
Вычисления прекратить, когда очередной элемент последовательности станет равен 153.
(Известно, что любая такая последовательность рано или поздно приводит к 153).

Задача 2.14

Для заданного натурального числа определить, образуют ли его цифры арифметическую прогрессию. Предполагается, что в числе не менее трёх цифр.
Например: 1357, 963.

Задача 2.15

В трехзначном числе зачеркнули старшую цифру. Когда полученное число умножили на 7, то получили исходное число. Найти это число.

Задача 2.16

Получить все числа, не превышающие заданного числа *N*, которые делятся без остатка на все свои цифры.

Задача 2.17

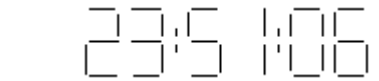
Задано натуральное число. Записать его в обратном порядке. Например, 12345 должно превратиться в 54321.

Задача 2.18

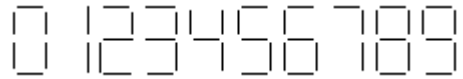
Напечатать все натуральные четырехзначные числа, в десятичной записи которых нет одинаковых цифр, и разность двух натуральных двузначных чисел, составленных из двух последовательных первых цифр и двух последовательных последних цифр числа, равна сумме всех цифр числа.

Задача 2.19

В электронных часах время показывается на табло в виде последовательности цифр, указывающих часы (от 0 до 23), минуты и секунды.
Например:



Каждая отдельная цифра на табло отображается в виде светящихся сегментов (отрезков) следующим образом:



Часы потребляют тем больше энергии, чем больше сегментов используется в записи времени. Написать программу, которая определяет время (чч.мм.сс) наибольшего и наименьшего потребления энергии часами.

Геометрические задачи.

Задача 2.20

Выпуклый многоугольник задан последовательностью координат своих вершин в порядке обхода: (x_1, y_1), (x_2, y_2), (x_3, y_3), . . . , (x_n, y_n). Вычислить площадь многоугольника.

Задача 2.21

Задана точка с координатами (x, y) и треугольник с координатами вершин (x_1, y_1), (x_2, y_2), (x_3, y_3). Определить, лежит ли точка внутри или вне треугольника.

Календарь.

Задача 2.22

Написать программу, которая по заданной дате (числу *d* и месяцу *m*) определяет число дней, прошедших от начала года, если известно, что год - не високосный.

Задача 2.23

Написать программу, которая по заданному числу дней, прошедших от начала года, определяет дату: число *d* и месяц *m*, если известно, что год - не високосный.

Задача 2.24

В журнале метеостанции записаны ежедневные температуры воздуха в г. Смоленске. Определить самый холодный будний день октября 1997 г., если известно, что 1 октября 1997 г. - среда. (Будними считаются все дни недели, за исключением субботы и воскресенья.)

Задача 2.25

Компьютерный вирус "Пятница, 13" может повредить информацию только в те дни, когда 13 число попадает на пятницу. Определить все месяцы 2000 года, в которых 13 число было пятницей. Учесть, что 2000 год - високосный и 1 января 2000 года - суббота. В качестве ответа распечатать номера месяцев.

Решения задач

Задача 1.1

Для решения этой задачи необходимо знать, что 1 мегабайт=1024 килобайт, поэтому 6 мегабайт=6x1024=6144 килобайт. Обозначим t - время звучания композиции в секундах, v - объём файла композиции в килобайтах, тогда:
t=60*m+n, v=16*t.
Программа на Паскале будет иметь вид:

```
var m,n,t,v:integer;
begin
    writeln('Введите m и n');
    readln(m,n);
    t:=60*m+n;
    v:=16*t;
    if v<=6144 then writeln('Композиция поместится')
    else writeln('Не хватает ',v-6144,' килобайт');
end.
```

Задача 1.2

Рассмотрим, как ходят фигуры: ферзь бьёт те поля (с координатами *x, y*), которые находятся с ним на одной вертикали ($x=x_1$), на одной горизонтали ($y=y_1$), или на любой из диагоналей ($|x - x_1| = (|y - y_1|$). Конь за один ход переходит на два поля по одной координате и на одно поле по другой координате, то есть поля, которые он бьёт, определяются по правилу: либо $|x - x_2| = 2$ и $|y - y_2|=1$, либо $|x - x_2| = 1$ и $|y - y_2| = 2$. При решении нужно учитывать, что фигуры не могут угрожать друг другу одновременно, и может быть ситуация, когда фигуры вообще не угрожают друг другу.
Основная часть программы для данной задачи будет иметь следующий вид:

```
if (x1=x2)or(y1=y2)or(abs(x1-x2)=abs(y1-y2)) then
writeln('Ферзь бьёт коня')
else if (abs(x1-x2)=2)and(abs(y1-y2)=1)or
     (abs(x1-x2)=1)and(abs(y1-y2)=2) then
     writeln('Конь бьёт ферзя')
     else writeln('Фигуры не угрожают друг другу');
```

Задача 1.3

При решении учтите, что число рыбок должно быть целым числом. Например, в аквариуме объёмом 20,5 литров может жить 6 рыбок (а не 6,83333...). Функция выделения целой части числа *x* в Паскале - **trunc(x)**.

Задача 1.4

При решении учтите, что если полторы курицы за полтора дня сносят полтора яйца, то одна курица за тот же срок (полтора дня) снесет одно яйцо. Например: 6 кур за 6 дней снесут 24 яйца.

Задача 1.5

Для решения этой задачи можно разделить число нацело *N* на 3 и рассмотреть остаток от деления. Существует три варианта: если остаток 0, то сумма выплачивается трехкопеечными монетами; если остаток 1 (наименьшее такое число 10), то необходимо убрать 3 монеты по 3 копейки и добавить 2 монеты по 5 копеек; если остаток от деления 2, то необходимо убрать 1 трёхкопеечную монету и добавить 1 монету достоинством 5 копеек. В Паскале операция деления нацело - **div**, операция вычисления остатка при делении целых чисел - **mod**.

Задача 1.10

При решении этой задачи необходимо воспользоваться тем условием, что a и b - числовые переменные, тогда поменять их местами можно, например, следующим образом:

```
a:=a+b;
b:=a-b;
a:=a-b;
```

Задача 2.1

Разделим *N* нацело на 5 и получим *k* - максимальное значение для *y* (т.е. *0<=y<=k*). Организуем цикл по переменной *y* , и будем рассматривать значения разности *N-5y*. Если это число делится нацело на 3, то полученное частное и есть соответствующее значение *x*.
Соответствующая программа будет иметь вид:

```
var x,y,n,k:integer;
begin
  writeln('Введите N');
  readln(n);
  k:=n div 5;
  for y:=0 to k do
  if (N-5*y) mod 3=0 then
  begin
    x:=(N-5*y) div 3;
    writeln('x=',x,'  y=',y);
  end;
end.
```

Задача 2.3

Для решения этой задачи необходимо вычислять функцию n! (читается n - факториал), которая представляет собой произведение натуральных чисел от 1 до n. Программа вычисления n! будет иметь вид:

```
var n,i:integer;
    p: real;
begin
 readln(n);
 p:=1;
 for i:=1 to n do p:=p*i;
 writeln(n,'!=',p:1:0);
```

end.

Значение факториала накапливается в этой программе в переменной **p**. Особенность оператора цикла **for i:=1 to n do** ... в том, что если **n** меньше начального значения i (в данном случае 1), то тело цикла не выполнится ни разу. Поэтому проверять условие, что **n>0** не имеет смысла, так как значение **p** в этом случае останется равным 1. Для переменной **p** выбран вещественный тип **real**, так как функция факториал очень быстро растет (формат печати :1:0 означает, что будет печататься только целая часть числа). На основе этой программы легко написать программу, вычисляющую C_N^M. Вычисление факториала удобно при этом офрмить в виде подпрограммы.

Задача 2.4

При решении учтите, что число 0 не относится ни к отрицательным, ни к положительным числам.

Задача 2.8

Обозначим: k - номер рейса судна, i - номер очередного груза, s - масса груза на судне в k-том рейсе. Решать задачу будем так: если на судно в k-том рейсе можно поместить ещё один груз, то мы грузим его и берём следующий, если груз не может быть размещен, то перевозим его следующим рейсом (увеличиваем k).

Основная часть соответствующей программы будет иметь вид:

```
k:=1; i:=1; s:=0;
repeat
    if s+m[i]<=50 then
    begin
      s:=s+m[i];
      i:=i+1;
    end
    else
    begin
      k:=k+1;
      s:=0;
    end;
until i>15;
writeln('Всего потребовалось', k,' рейсов');
```

Задача 2.10

Вычисление непрерывных радикалов производится в цикле, начиная от внутреннего радикала. В данной задаче начальное значение $R = \sqrt{2}$. Каждое следующее значение радикала будет вычисляться через предыдущее значение радикала по формуле $R = \sqrt{a + R}$, число изменяется от начального значения 5 до конечного значения 98 с шагом 3.

Программа для вычисления *R* будет иметь вид:

```
var r,a:real;
begin
  r:=sqrt(2); a:=5;
  while a<=98 do
  begin
     r:=sqrt(a+r);
     a:=a+3;
  end;
  writeln('R=',r);
end.
```

Вычисленное по данной программе значение $R \approx 10.4044$.

Задача 2.11

Вычисление непрерывных дробей производится снизу вверх, начиная от последней. Значение Q≈ 0.69777.

Задача 2.13

На каждом шаге данного алгоритма приходится разбивать целое число на отдельные цифры (причем количество цифр в числе неизвестно). Это можно выполнить, используя операции целочисленной арифметики (деления нацело - **div** и остатка от деления - **mod**). Процесс вычисления очередного члена последовательности **p** через предыдущий в рассматриваемой задаче будет иметь следующий вид (**s** и **p1** - рабочие переменные, **t** - очередная цифра числа):

```
s:=0; p1:=p;
while p1<>0 do
begin
   t:=p1 mod 10;
   p1:=p1 div 10;
   s:=s+t*t*t;
end;
p:=s;
```

Задача 2.18

Любое целое четырехзначное число можно представить в виде:
$N = a \cdot 1000 + b \cdot 100 + c \cdot 10 + d$ (*a, b, c, d* - цифры числа, причем $a \neq 0$).

Например: 1742=1*1000+7*100+4*10+2.

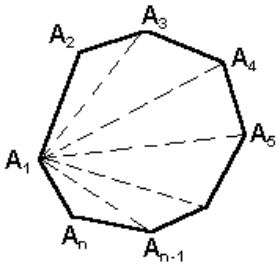
То, что цифры числа не должны совпадать, можно записать на Паскале в виде условия:
(a<>b)and(a<>c)and(a<>d)and(b<>c)and(b<>d)and(c<>d).

Условие на разность чисел, составленных из цифр числа:

*a*10+b-(c*10+d)=a+b+c+d.*
Тогда выполняемая часть программы будет иметь вид:

```
for a:=1 to 9 do
for b:=0 to 9 do
for c:=0 to 9 do
for d:=0 to 9 do
if (a<>b)and(a<>c)and(a<>d)and(b<>c)and(b<>d)and(c<>d)and
(a*10+b-(c*10+d)=a+b+c+d)
then writeln(a*1000+b*100+c*10+d);
```

Задача 2.20



Стандартный способ вычисления площади выпуклого многоугольника - разбиение исходного многоугольника на отдельные треугольники (рис.) с последующим вычислением площадей полученных треугольников и их суммированием. Площадь отдельного треугольника можно вычислить, например, по формуле Герона, но в данном случае более удобной будет формула расчета площади треугольника по координатам его вершин:

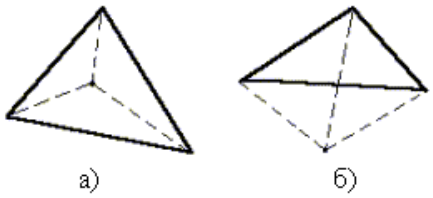
$$S = \frac{1}{2}|(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)|$$

Пусть **n** - число вершин, **X(n)**, **Y(n)** - массивы, содержащие координаты вершин, тогда основная часть программы для вычисления площади многоугольника будет иметь вид:

```
s:=0;
for i:=3 to n do
s:=s+0.5*abs((x[i-1]-x[1])*(y[i]-y[1])-
             (x[i]-x[1])*(y[i-1]-y[1]));
writeln('Площадь многоугольника s=',s);
```

Задача 2.21

Чтобы определить, лежит ли точка внутри треугольника, можно соединить эту точку отрезками с его вершинами и рассчитать площади получившихся треугольников (как в предыдущей задаче). Если сумма вычисленных площадей равна площади исходной фигуры (рис. а), то точка лежит внутри, если нет (рис. б) - снаружи.



Задача 2.22

Для решения задачи можно создать массив D(12), каждый элемент которого - число дней в соответствующем месяце.

Задача 2.24

Пусть *i* - номер дня в октябре месяце. Так как 1 октября среда, то 4 и 5 октября будут соответственно суббота и воскресенье. Соответственно, субботами и воскресеньями будут все те дни, которые отличаются от 4 и 5 на целое число недель. Субботами будут дни с такими номерами *i*, что остаток от деления на 7 равен 4 (*i mod 7 = 4*), воскресеньями - дни с номерами *i*, для которых *i mod 7 =5*.

3. Задачи с использованием строкового типа данных.

В этом классе задач приходится оперировать таким типом данных, как строки, проводить преобразования строковых данных в числовые и наоборот.

Задача 3.1

Для заданной строки символов проверить, является ли она симметричной или нет. (Симметричной считается строка, которая одинаково читается слева направо и справа налево).

Задача 3.2

Для заданной строки символов проверить, является ли она палиндромом (симметричной с точностью до пробелов) или нет.
Например, палиндромами являются цепочки:
АРГЕНТИНА МАНИТ НЕГРА
А РОЗА УПАЛА НА ЛАПУ АЗОРА
(Предполагается, что все буквы строки - прописные).

Задача 3.3
Для заданной строки символов определить сумму всех входящих в неё цифр.

Задача 3.4
Для заданной строки символов вычислить произведение входящих в эту строку целых чисел (без учета их знаков). Например, для строки "kjjjkkj**2**.**5**jkjn,,,hfd**45**jgfvjlkfdii**10**,**2**hfhg" произведение **2*5*45*10*2=9000**.

Задача 3.5
Для заданной строки символов вычислить сумму входящих в неё цифр, причем знак очередного слагаемого должен быть противоположным знаку предыдущего слагаемого.
Например:
Для строки "asdd**1**vnb**24**vnf**63**vbn,-**5**h-**2**kk"
Сумма S=1-2+4-6+3-5+2= -3.

Задача 3.6
Для заданной строки символов найти наибольшее записанное в этой строке целое число (без учета знака числа). Например, для строки "sdfvgsd1.9fdmjgvb-15.2dnj05" наибольшее целое число 15.

Задача 3.7
Для заданной строки определить все входящие в неё символы. Например: строка "abccbbbabba" состоит из символов "a","b" и "c".

Задача 3.8
Задана строка символов. Среди литер этого текста особую роль играет знак #, появление которого означает отмену предыдущей литеры текста; k знаков # отменяют k предыдущих литер (если такие есть) Напечатать строку с учетом роли знака #. Например, строка "VR#Y##HELO#LO" должна быть напечатана в виде: "HELLO".

Задача 3.9
Задана строка символов. Определить, какой символ встречается в этой строке подряд наибольшее число раз. В ответе указать символ, образующий самую длинную последовательность, длину последовательности и номер символа, с которого она начинается.
Например, в строке "asadddbbbbababaaaaaahhgg" символ a образует последовательность длиной в 6 символов, начиная с символа с номером 15.

Задача 3.10
Для заданной строки символов, состоящей из строчных букв и пробелов, определить слово наибольшей длины, которое начинается и заканчивается на одну и ту же букву. Например: строка - "револьвер системы наган", слово - "револьвер".

Задача 3.11
В символьной строке имеются круглые скобки. Проверить, правильно ли они расставлены.

Задача 3.12
Заданы две строки. Определить, являются ли они анаграммами, то есть одна строка получена из другой перестановкой букв.
Например:
строки "БУК" и "КУБ" или "СОЛЬ" и "ЛОСЬ" являются анаграммами.

Задача 3.13
Вспомним игру: "Придумай слово", в которой из букв слова-донора составляют другие слова. Например, из слова МОНИТОР можно получить МОТОР, РОТ, ТИР и др. Вхождение каждой буквы в новое слово допускается не более того числа раз, с каким она входит в слово-донор. Пусть дана последовательность слов, разделенных пробелами в виде строки символов. Известно,

что первое слово в этой строке является донором. Выбрать из оставшихся слов последовательности те, которые можно получить из заданного слова-донора.

Задача 3.14

Задана строка символов, содержащая два или более слов, разделенных пробелами. Написать программу, меняющую местами все четные и нечетные слова в строке, предполагая, что за один раз можно менять местами не более двух символов.

Решения задач

Задача 3.1

Проще всего в этой задаче определить длину строки n, организовать цикл по номеру символа в строке и сравнивать попарно первый символ с последним, второй - с предпоследним и т.д. Если хотя бы одна пара различна, то строка не симметричная. Так как просматривается сразу пара символов, то в цикле будет m = n div 2 повторений. Для запоминания результата просмотра введем переменную k (k будет равна 0, если строка симметрична и 1 иначе).
Программа, решающая эту задачу, будет иметь вид:

```
var   s:string;
i,k,n,m:integer;
begin
  readln(s);
  n:=length(s);
  k:=0;
  m:=n div 2;
  for i:=1 to m do
  if s[i]<>s[n-i+1] then k:=1;
  if k=0 then writeln('Строка симметрична')
  else writeln('Строка несимметрична');
end.
```

Задача 3.2

Если удалить из заданной строки все пробелы, то задача сведется к предыдущей.

Задача 3.3

Так как все цифры от 0 до 9 располагаются в таблице кодировки компьютера подряд, то проще всего проверить, является ли символ s[i] цифрой, можно с помощью неравенства '0'<=s[i]<='9' (на Паскале **(s[i]>='0')and(s[i]<='9')**). Для преобразования строки в число в Паскале используется процедура val(s,v,k), где s - строка (или символ), v - переменная числового типа, куда будет занесён результат преобразования, k - переменная целочисленного типа, которая принимает значение 0, если преобразование строки в число прошло успешно.

Задача 3.4

Пусть s - строка. Обозначим длину строки - l. Организуем цикл, в котором будем проверять, является ли очередной символ цифрой, если да, то организуем новый цикл, в котором будем формировать строку sn, состоящую из цифр (очередное целое число). Потом преобразуем sn в число и вычислим произведение. Программа на Паскале, реализующая данный алгоритм, будет иметь следующий вид (переменная p в этой программе используется для накапливания значения произведения, переменная kod - для хранения кода результата преобразования строки в число):

```
var sn,s:string;
    l,k,kod:integer;
    v,p:real;
begin
 writeln('Введите строку');
 readln(s);
 l:=length(s);
 p:=1; k:=1;
 repeat
  sn:='';
  while (s[k]>='0')and(s[k]<='9')and(k<=l) do
   begin
    sn:=sn+s[k];
    k:=k+1;
   end;
   if sn<>'' then
   begin
```

```
    val(sn,v,kod);
    p:=p*v;
   end;
   k:=k+1;
   until k>l;
   writeln(' p=',p);
end.
```

Задача 3.8

Обозначим s - исходную строку, l - длину этой строки. Для решения создадим ещё одну строку s1(вначале пустую). Далее организуем цикл по номеру символа в строке s. Если очередной символ не #, то добавим его к строке s1, если это знак # и строка s1 не пустая, то удалим из неё последний символ.

Программа, реализующая данный алгоритм, будет иметь следующий вид:

```
var s,s1:string;
  dl,i,k:integer;
begin
  writeln('Введите строку');
  readln(s);
  dl:=length(s);
  s1:='';
  k:=0;
  for i:=1 to dl do
  if (s[i]='#')and(k<>0) then
   begin
    delete(s1,k,1);
    k:=k-1;
   end
  else
    begin
     k:=k+1;
     s1:=s1+s[i];
    end;
 writeln(s1);
end.
```

Задача 3.10

Пусть s - заданная строка. Для решения данной задачи определим длину строки l и организуем цикл по номеру символа i. Символ строки является первым символом некоторого слова в том случае, когда он сам не является пробелом, и либо он - первый символ строки, либо слева от него стоит пробел. Если мы нашли первый символ некоторого слова, то запомним его номер и организуем цикл, в котором найдем номер последнего символа этого слова (символ будет последним в слове либо тогда, когда после него стоит пробел, либо когда он последний символ строки). Если последний символ слова совпадает с первым символом этого слова, и длина слова наибольшая из всех найденных, запомним эту длину и номер первого символа этого слова.

В приведенной программе введены следующие обозначения:

max - переменная, в которой запоминается текущая максимальная длина найденного слова; k - переменная, в которой поочередно запоминается номер первого символа каждого слова; koord - переменная, в которой хранится номер первого символа слова с максимальной длиной.

```
var s:string;
    koord,k,i,n,max:integer;
    fst:char;
begin
   writeln('Введите строку');
   readln(s);
   n:=length(s);
   max:=0; i:=1;
   while i<=n do
   if (s[i]<>' ')and((i=1)or(s[i-1]=' ')) then
   begin
       k:=i;
       while (i<n)and(s[i+1]<>' ') do  i:=i+1;
       if (s[i]=s[k])and(i-k+1>max) then
       begin
         koord:=k;
         max:=i-k+1;
         i:=i+1;
       end;
   end
```

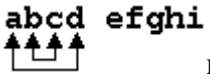
```
   else i:=i+1;
  if max<>0 then
  begin
     writeln(' max=',max);
     for i:=0 to max-1 do write(s[koord+i]);
  end
  else writeln('Такого слова нет')
end.
```

Задача 3.11

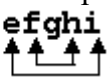
Обозначим K - число левых скобок, M - число правых скобок, тогда, на каждом шаге подсчета скобок, должно выполняться условие: K>=M. После подсчета всех скобок должно выполниться условие K=M.

Задача 3.14

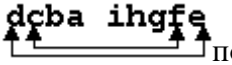
Имеется несколько путей решения этой задачи. Для упрощения предположим, что строка не начинается и не заканчивается пробелом и что между словами в строке стоит ровно по одному пробелу. Пусть известна пара слов, которую необходимо переставить, и - номера первой и последней букв в первом слове, и - номера первой и последней букв во втором слове. Рассмотрим способ, в котором для перестановки слов будем использовать следующий алгоритм:
Запишем буквы первого слова в обратном порядке (поменяв первую с последней, вторую с предпоследней и т.д.).

Например, из строки 

получим **dcba efghi**.
Потом аналогичным образом переставим буквы второго слова:

из строки 

получим **dcba ihgfe**.
Для получения окончательного результата необходимо записать буквы полученного словосочетания в обратном порядке:

Из строки 

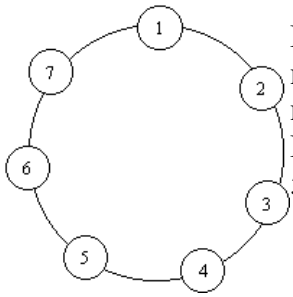
получим **efghi abcd** (что и требовалось получить).
Таким образом, для перестановки двух слов достаточно написать подпрограмму, которая меняет в заданной строке порядок букв на противоположный (инвертирует строку), и вызвать эту подпрограмму для первого слова, второго слова и всего словосочетания.
Обозначим **invert(k,l)** - процедуру, которая записывает в заданной строке s символы с k-того по l-й в обратном порядке, тогда программа, решающая задачу, будет иметь вид:

```
var s:string;
    i,n,m1,m2,l1,l2:byte;
procedure invert(k,l:byte);
var i:byte;
    ch:char;
begin
   for i:=k to ((l+k) div 2) do
   begin
      ch:=s[i];
      s[i]:=s[l+k-i];
      s[l+k-i]:=ch;
   end;
end;
begin
writeln('Введите строку'); readln(s);
i:=0; n:=0;
m1:=1;m2:=1;l1:=1;l2:=1;
while i<length(s) do
begin
  i:=i+1;
  if (s[i]=' ')or(i=length(s)) then
  begin
    n:=n+1;
    if n=1 then
    begin
      m2:=i-1;
      l1:=i+1
    end
    else
    begin
```

```
      n:=0;
      if i=length(s) then l2:=i else l2:=i-1;
      invert(m1,m2);invert(l1,l2);invert(m1,l2);
      m1:=i+1
    end
  end
end;
writeln(s)
end.
```

4. Задачи повышенной сложности

Задача 4.1



В круге стоят N человек (рис.). Они пронумерованы от 1 до N. Поочередно из круга начинает выходить каждый третий человек. Это продолжается до тех пор, пока в круге не останется последний человек. Определить его номер. Например, если в круге стояло 7 человек, то его поочерёдно покинут 3, 6, 2, 7, 5, 1. Оставшимся будет человек, стоявший на 4 месте.

Задача 4.2

Вывести на экран цифры числа 3^{1000}. Если попытаться получить число непосредственно умножением, компьютер выдаст сообщение об ошибке.

Задача 4.3

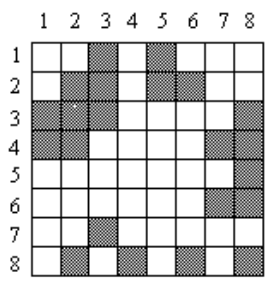
Написать программу для умножения двух чисел, количество цифр в каждом из которых может достигать 100. Например, для умножения вида:
92789693458975698723 65*5705782370079678659.

Задача 4.4

Сообщество роботов живет по следующим законам: один раз в год они объединяются в полностью укомплектованные группы по 3 или 5 роботов (причем число групп из 3 роботов - максимально возможное). За год группа из 3 роботов собирает 5, а группа из 5 - 9 новых собратьев. Каждый робот живет 3 года после сборки. Известно начальное количество роботов (K>7), все они только что собраны. Определить сколько роботов будет через N лет.

Задача 4.5

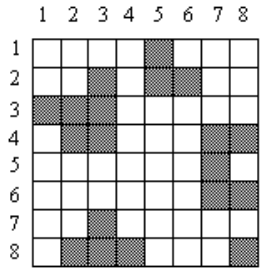
На квадратном клетчатом листе бумаги 8x8 клеток заштрихована часть клеток (пример на рисунке). Определить вписанный в решётку прямоугольник максимальной площади, не содержащий заштрихованных клеток. В качестве ответа вывести площадь прямоугольника и координаты его двух противоположных вершин. (Предполагается, что прямоугольник с максимальной площадью один.)



Для приведенного примера координаты вершин (3,4) и (7,6), площадь 15 клеток.

Задача 4.6

На квадратном клетчатом листе бумаги n на m клеток нарисовано несколько фигур, каждая из которых состоит только из целых клеток. Различные фигуры не накладываются и не соприкасаются (пример на рисунке). Определить фигуру максимальной площади. (В качестве ответа вывести площадь фигуры и координаты одной из её точек. Предполагается, что фигура с максимальной площадью одна.)
Например, на листе 8 на 8 клеток:



Площадь фигуры равна 6. Координаты одной из её клеток (2,3).

Задача 4.7

Подсчитайте количество одно-, двух-, трёх- и четырехпалубных кораблей, расположенных на поле для игры "Морской бой". Корабли не могут быть изогнутыми и друг с другом не соприкасаются. Поле для игры задается в виде таблицы 10x10, каждый элемент которой равен либо 0, если клетка свободна, либо 1, если занята.

Задача 4.8

Лабиринт задан в виде матрицы размером n на m. Стенам лабиринта соответствуют единицы, проходам - нули. Определить, можно ли из точки с координатами (i_1, j_1) попасть в точку с координатами (i_2, j_2). Для усложнения задачи можно предложить указать самый короткий путь из заданной точки, причем из всех путей одинаковой длины выбирать путь с наименьшим числом поворотов.

Задача 4.9

На рисунке изображен треугольник из чисел. Вычислите наибольшую сумму чисел, расположенных на пути, начинающемся в верхней точке треугольника и заканчивающемся на его основании. Каждый шаг на пути может идти вниз по диагонали влево или вниз по диагонали вправо. Число строк в треугольнике больше1 и не больше100. Треугольник составлен из целых чисел от 0 до 99.

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

Задача 4.10

Имеется n населенных пунктов, пронумерованных от 1 до n. Некоторые пары пунктов соединены дорогами (в том числе дорогами с односторонним движением). Определить, можно ли попасть по этим дорогам из одного заданного пункта в другой. (Для усложнения задачи можно предложить указать все возможные пути без петель и тупиков из одного пункта в другой).

Задача 4.11

Заданы две фразы. Определить наибольшую последовательность отличных от пробелов символов, входящую в обе фразы в одном и том же порядке. Например, в предложениях:
ПРИШЛА ВЕСНА и РАСТАЯЛ СНЕГ такая последовательность - Р,А,С,Н.

Задача 4.12

В морском порту города Владивостока хранятся N контейнеров (N - чётное число). Для погрузки контейнеров на судно, чтобы обеспечить равномерную загрузку, их необходимо разделить на две половины так, чтобы их массы были максимально близки. Решить эту задачу, предполагая, что информация о массах контейнеров (в тоннах) хранится в массиве M(N). В качестве ответа указать номера контейнеров одной половины и получаемые массы для каждой из половин.
Например:
Если M(6)=(10, 15, 18, 20, 16, 14), то одну половину составят 1, 4, 5 контейнеры (другую 2, 3, 6).
Масса первой группы m1=10+20+16=46 т., масса второй группы m2=15+18+14=47 т.

Задача 4.13

На шахматной доске необходимо расставить 8 ферзей так, чтобы они не угрожали друг другу.

Задача 4.14

На шахматной доске размером 4x4 клетки расставить 4 ладьи так, чтобы они не угрожали друг другу. Определить все такие расстановки (всего их будет 24).

Решения задач

Задача 4.2

Для записи больших чисел удобно использовать массивы, записывая цифры числа как элементы массива. Оценим количество цифр, необходимых для записи 3^{1000}:
$3^2 = 9$; $3^{1000} = (3^2)^{500} = 9^{500} \approx 10^{500}$.
Таким образом, в записи этого числа будет менее 500 цифр.
Запишем вначале в массив только число 3 и далее будем умножать его поэлементно на 3 нужное число раз, учитывая перенос из разряда в разряд, возникающий при умножении.
Программа, решающая эту задачу, может быть записана, например, так:

```
const stp=1000;
var i,j,k,prn,x:integer;
    a:array [1..500] of integer;
begin
     a[500]:=3; prn:=0;
     for i:=2 to stp do
     for j:=500 downto 1 do
     begin
          x:=a[j]*3;
          a[j]:=(x+prn) mod 10;
          prn:=(x+prn) div 10;
     end;
     k:=1; while (a[k]=0) do k:=k+1;
     for i:=k to 500 do write(a[i]:1);
     writeln
```

end.

Здесь **stp** - степень, в которую возводится число 3, **a[500]** - массив, в котором хранятся цифры полученного числа, **prn** - перенос из разряда в разряд. Объём вычислений в данной программе можно значительно сократить, если каждый раз умножать на 3 не весь массив a, а только его занятую часть.

Задача 4.3

Принцип решения этой задачи такой же, как и у предыдущей. Результат умножения двух стозначных чисел будет не превышать по размеру 200 знаков (т.е. для хранения результата понадобится массив из 200 элементов). Для ввода исходных чисел удобно использовать строковый тип данных.

Задача 4.4

Рассмотрим следующий вариант решения. Создадим массив *R(3)*, где R_1, R_2, R_3 - количество роботов соответствующего возраста. Тогда общее количество роботов $S = R_1 + R_2 + R_3$. Обозначим *x* - количество троек, *y* - количество пятерок, которое можно сформировать из общего числа роботов (идея разбиения на тройки и пятерки рассмотрена в задаче 1.5). Каждый год роботы стареют, общее количество роботов увеличивается на $5x+9y$ и уменьшается на R_3 (число роботов, проживших 3 года). Программа решения может быть записана так:

```
var k,i,n,p:integer;
    s,x,y:longint;
    r:array [1..3] of longint;
begin
  write('Начальное количество роботов k='); readln(k);
  write('Число лет n='); readln(n);
  r[1]:=k; r[2]:=0; r[3]:=0; s:=k;
  for i:=1 to n do
  begin
      x:=s div 3;
      p:=s mod 3;
      if p=0 then y:=0
      else if p=1 then begin x:=x-3; y:=2 end
      else begin x:=x-1; y:=1 end;
      r[3]:=r[2]; r[2]:=r[1]; r[1]:=5*x+9*y;
```

```
      s:=r[1]+r[2]+r[3];
  end;
  writeln('s=',s)
end.
```
В этой программе использовался тип **longint**, предназначенный для хранения больших целых чисел (до 2147483647). Это связано с тем, что общее число роботов увеличивается очень быстро. Так, например, если начальное число роботов - 10, то через 10 лет их будет 143702.

Задача 4.5

Представим лист бумаги в виде числовой матрицы A(8x8). Обозначим заштрихованные клетки единицами, а чистые - нулями.
Данную программу удобно реализовать, используя подпрограммы. Напишем подпрограмму, которая проверяет, есть ли в прямоугольнике с координатами левой верхней вершины **(i1,j1)** и координатами правой нижней вершины **(i2,j2)** заштрихованные клетки (т.е. элементы, равные 1).
```
function prov(a:matr;i1,j1,i2,j2:integer):boolean;
var i,j:integer;
begin
  prov:=true;
  for i:=i1 to i2 do
  for j:=j1 to j2 do
  if a[i,j]=1 then prov:=false;
end;
```
Эта функция будет возвращать значение "истина" (true), если заштрихованных клеток в рассматриваемом прямоугольнике нет, и "ложно" (false) - в противном случае.
В основной программе организуем два попарно вложенных цикла: в первом цикле будут изменяться координаты верхнего левого угла рассматриваемого прямоугольника, во втором - координаты нижнего правого угла. Если в прямоугольнике нет заштрихованных точек, то вычислим его площадь, и если она является максимальной, запомним её и координаты противоположных вершин этого прямоугольника. Часть основной программы, реализующая данный алгоритм, будет иметь следующий вид:
```
maxs:=0;
 for i1:=1 to n do
 for j1:=1 to n do
   for i2:=i1 to n do
   for j2:=j1 to n do
   begin
     s:=(i2-i1+1)*(j2-j1+1);
     if prov(a,i1,j1,i2,j2)and(maxs<s) then
     begin
       maxs:=s;
       maxi1:=i1;
       maxj1:=j1;
       maxi2:=i2;
       maxj2:=j2;
     end;
   end;
writeln(' s=',maxs);
writeln('(',maxi1,',',maxj1,')',    '(',maxi2,',',maxj2,')');
```
Здесь maxs - площадь полученного прямоугольника, (maxi1, maxj1) - координаты его левой верхней вершины, (maxi2, maxj2) - координаты его правой нижней вершины.

Задача 4.6

Подобные задачи достаточно просто реализуются с использованием рекурсии. Решение построим следующим образом: представим лист в виде числовой матрицы A(nxm). Обозначим заштрихованные клетки единицами, а чистые - нулями. Напишем рекурсивную процедуру, которая определяет площадь заштрихованной фигуры и заменяет клетки уже просмотренной фигуры двойками (чтобы не просматривать эту фигуру ещё раз). В основной программе организуем цикл перебора всех элементов матрицы. Если очередной элемент равен 1 (ещё не просмотренная фигура), то будем вызывать процедуру определения площади этой фигуры).
Процедура рекурсивного определения площади фигуры будет иметь вид:
```
procedure zaliv(i,j:byte; var s:integer);
begin
    a[i,j]:=2;
    s:=s+1;
    if (i+1<=n)and(a[i+1,j]=1) then  zaliv(i+1,j,s);
    if (j+1<=m)and(a[i,j+1]=1) then  zaliv(i,j+1,s);
```

```
    if (j-1>0)and(a[i,j-1]=1)  then  zaliv(i,j-1,s);
    if (i-1>0)and(a[i-1,j]=1)  then  zaliv(i-1,j,s);
end;
```
Здесь n и m - число столбцов и строк в рассматриваемой матрице, i, j - координаты найденной клетки фигуры, s - переменная, в которой накапливается площадь фигуры. Процедура учитывает найденную клетку в площади, потом проверяет, заштрихована ли клетка, которая расположена ниже просматриваемой (если это не последняя строка), и если да, то рекурсивно вызывается с этой клетки, потом тот же процесс повторяется для клеток расположенных справа, слева и сверху от рассматриваемой. Рабочая часть основной программы будет иметь вид:
```
max:=0; im:=0; jm:=0;
for i:=1 to n do
for j:=1 to m do
   if a[i,j]=1 then
   begin
     s:=0;
     zaliv(i,j,s);
     if s>max then
     begin
        max:=s;
        im:=i;
        jm:=j;
     end;
   end;
writeln('Smax=',max,'  im=',im,'  jm=',jm);
```
Здесь max - переменная, в которой хранится максимальная площадь фигуры, im, jm - координаты первой найденной точки этой фигуры.

Задача 4.8

Наиболее простым способом решения задач по поиску пути является рекурсивный поиск. Его алгоритм во многом аналогичен рассмотренному в задаче 4.6.

Пусть *A(nxm)* - матрица, задающая лабиринт (1 - стена, 0 - проход). Напишем процедуру рекурсивного перемещения по лабиринту. Путь прохода будем отмечать цифрами, начиная с 2 (2, 3, 4 и т.д.). Пусть на *k*-том шаге мы попали на клетку с координатами *(i, j)*. Если это та клетка, путь до которой необходимо было отыскать (с координатами (i_2, j_2)), то задача решена. Необходимо распечатать матрицу с отмеченным путем и прекратить выполнение программы. Если нет, то необходимо проверить, можно ли перейти на соседнюю клетку (справа, слева, сверху или снизу), и если возможно, то вызвать процедуру перемещения уже с этой клетки. Если перехода на соседнюю клетку нет, то необходимо очистить исходную клетку, чтобы её можно было использовать в других вариантах при поиске пути. Обозначим **move(i,j,k)** - процедуру рекурсивного перемещения (**i, j** - номер клетки, **k** - номер шага), **writematr** - процедуру, распечатывающую матрицу A, тогда программа, реализующая предложенный алгоритм, может быть записана следующим образом:
```
const n=4; m=5;
type matr=array [1..n,1..m] of integer;
{Матрица, задающая варианты прохода. 1-стена; 0-проход.}
const labir:matr=((1,0,0,0,1),
                  (0,0,1,0,1),
                  (1,0,0,0,0),
                  (0,0,1,0,0));
var a:matr;
    i1,j1,i2,j2,f:byte;
    k:integer;
procedure writematr;
var i,j:byte;
begin
for i:=1 to n do
      begin
         for j:=1 to m do write(a[i,j]:3,' ');
         writeln;
      end
end;
procedure move(i,j:byte; k:integer);
begin
   a[i,j]:=k; k:=k+1;
   if (i=i2)and(j=j2) then begin writematr; f:=1; halt end
   else
   begin
```

```
     if (i+1<=n)and(a[i+1,j]=0) then move(i+1,j,k);
     if (j+1<=m)and(a[i,j+1]=0) then move(i,j+1,k);
     if (j-1>0)and(a[i,j-1]=0) then move(i,j-1,k);
     if (i-1>0)and(a[i-1,j]=0) then move(i-1,j,k);
   end;
   a[i,j]:=0; k:=k-1;
end;
begin
  a:=labir;
  writeln('Координаты i1, j1'); readln(i1,j1);
  writeln('Координаты i2, j2'); readln(i2,j2);
  f:=0; k:=2;
  if(a[i1,j1]=1)or(a[i2,j2]=1) then
  writeln ('Неверные координаты')
  else move(i1,j1,k);
  if f=0 then writeln('Прохода нет');
end.
```

Здесь - **labir** - матрица-константа 4x5, задающая пример лабиринта, **f** - переменная, через которую отслеживается, есть ли проход из начальной точки в конечную или нет, процедура **halt** - стандартная процедура, которая прекращает выполнение программы.

Если взять в качестве начальной точки точку с координатами (4, 1), в качестве конечной - точку с координатами (1, 4), то рассмотренная программа выдаст следующий вариант прохода:

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

Задача 4.11

Для решения этой задачи рассмотрим стандартную задачу перебора всех возможных сочетаний элементов некоторого массива. Пусть X(n) - массив с элементами 1, 2, … n (массив номеров). Напишем программу, которая на основе этого массива генерирует все подмножества номеров, состоящие из m элементов.

Например, при n=5, m=3. X(5)=(1,2,3,4,5)

Подмножества из 3-х номеров: (1,2,3), (1,2,4), (1,2,5), (1,3,4), (1,3,5), (1,4,5), (2,3,4), (2,3,5), (2,4,5), (3,4,5). Порядок перебора, рассмотренный в примере, называется лексикографическим. Такой порядок подобен расположению слов в словаре: сначала сравниваются первые буквы, потом вторые и т.д. Всего сочетаний из n элементов по m будет C_n^m (см. задачу 2.3), то есть $C_5^3 = 10$.

Для упрощения структуры программы напишем процедуру, которая печатает первые m элементов массива X(n) (предположим, что массив описан глобально):

```
procedure printm(m:integer);
var i:integer;
begin
  for i:=1 to m do write(x[i],' ');
  writeln;
end;
```

Далее рассмотрим функцию, которая на основе очередного сочетания получает следующее по порядку сочетание:

```
function posl(m:integer):boolean;
var j,f:integer;
label 10,20;
begin
    f:=0;
    posl:=true;
    for j:=m downto 1 do
      if x[j]=n+j-m then f:=j
      else
      begin
        inc(x[j]);
        goto 10
      end;
10: if f<>0 then if f=1 then
    begin
      posl:=false;
      goto 20
    end
    else for j:=f to m do x[j]:=x[f-1]+j-f+1;
```

```
20:
end;
```
Эта функция возвращает значение "истина" (true), если переданное сочетание не последнее (для рассмотренного примера: (1,3,4) или (2,4,5)), и значение "ложь" (false), если переданное в функцию сочетание последнее (для рассмотренного примера - (3,4,5)). Функция использует только первые m элементов массива X(n). В первом цикле проверяется, можно ли увеличить какой-либо из элементов массива (начиная с последнего). Если можно увеличить последний (m-й) элемент, то он увеличивается, служебной переменной f присваивается значение 0, и происходит выход из функции (например, из последовательности (1,2,4) получается последовательность (1,2,5)). Если последний элемент уже нельзя увеличить (например, последовательность (1,4,5)), то находится элемент, который еще можно увеличить (в данном случае 1), этот элемент увеличивается, а во втором цикле вслед за ним выстраиваются остальные (таким образом, из (1,4,5) получим (2,3,4)). В этом случае в f сохраняется номер элемента, следующего за увеличенным.
Программа, печатающая все сочетания из n элементов по m (при n=5, m=3), будет иметь вид:
```
const n=5;
var x:array[1..n] of integer;
    m,j:integer;
…
{Описание процедур printm  и posl}
…
begin
m:=3;
for j:=1 to m do x[j]:=j;
repeat
   printm(m);
until not posl(m);
end.
```
Удобство написанной функции в том, что, для получения всех возможных сочетаний из n номеров достаточно добавить в основную часть программы цикл по m:
```
begin
  for m:=n downto 1 do
  begin
    for j:=1 to m do x[j]:=j;
    repeat
      printm(m);
    until not posl(m);
  end;
end.
```
Для решения основной задачи необходимо удалить из обеих строк пробелы, далее перебирать в порядке убывания длины сочетания из букв одной строки и проверять, входят ли они в другую (если да, то решение найдено). Программа, решающая задачу, будет иметь вид:
```
const k=255;
var x: array [1..k] of integer;
    m,n,j: integer;
    s1,s2,s: string;
label 1;
…
{Описание процедуры posl}
…
procedure prints(m:integer);
var i:integer;
begin
  write(m,': ');
  for i:=1 to m do write(s1[x[i]]);
  writeln;
  readln;
end;
function spc(s:string):string;
var str:string;
    i:integer;
begin
     str:='';
     for i:=1 to length(s) do
     if s[i]<>' ' then str:=str+s[i];
     spc:=str;
end;
function equal(m:integer):boolean;
var i,j:integer;
```

```
begin
  j:=1;
  for i:=1 to length(s2) do
  if (s1[x[j]]=s2[i])and(j<=m) then j:=j+1;
  if j>m then equal:=true else equal:=false;
end;
begin
   writeln('Введите первую строку'); readln(s1);
   writeln('Введите вторую строку'); readln(s2);
   s1:=spc(s1); s2:=spc(s2);
   if (length(s2)<length(s1)) then begin s:=s1; s1:=s2; s2:=s end;
   n:=length(s1);
   for m:=n downto 1 do
   begin
        for j:=1 to m do x[j]:=j;
        repeat
              if equal(m) then
              begin
               prints(m);
               goto 1;
              end;
        until not posl(m);
   end;
1:
end.
```

В этой программе процедура **prints** печатает найденную последовательность символов и её длину, функция **equal** проверяет, входит ли очередная последовательность, составленная из символов одной строки в другую, функция **spc** удаляет из переданной в неё строки пробелы.
Работу полученной программы можно значительно ускорить, если добавить в неё часть, в которой перед началом перебора из обеих строк удалялись бы те символы, которые встречаются только в одной из них. (Например, заданные в условии строки после такого преобразования приняли бы вид: РЛАЕСНА и РАСАЛСНЕ).

Задача 4.13

Эта задача решена во многих книгах по программированию (например, в [4,5]). Так как ферзей 8, то очевидно, что на каждой вертикали и горизонтали будет стоять по одному ферзю, поэтому можно считать, что ферзь с номером k стоит на k-той вертикали и необходимо найти его координату по горизонтали.

Задача 4.14

Как и в предыдущей задаче, будем считать, что на каждой вертикали стоит по ладье, и для каждой из них необходимо найти координату по горизонтали (причем не совпадающую с соответствующими координатами остальных ладей). Таким образом, исходная задача сводится к нахождению всех возможных перестановок из 4 элементов. Известно, что число перестановок из n элементов равно n!=1*2*3*…*n. Например, из 3 элементов можно получить 3!= 1*2*3=6 перестановок:

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

Так как алгоритмы перестановок часто используются в олимпиадных задачах, рассмотрим их подробнее. Наиболее коротким и простым для запоминания является рекурсивный алгоритм получения перестановок. Пусть *X[n]* - массив, элементы которого числа от 1 до n. Для упрощения программы будем использовать процедуру **printm**, печатающую массив *X*, и процедуру s**wap(a,b)**, меняющую местами значения переменных **a** и **b**. Тогда программа рекурсивного получения перестановок (при n=4) будет иметь вид:

```
const n=4;
var x:array [1..n] of integer;
    i:integer;

procedure printm;
var i:integer;
begin
```

```
  for i:=1 to n do write(x[i],' ');
  writeln;
end;
procedure swap(var a,b:integer);
var v:integer;
begin
 v:=a; a:=b; b:=v
end;
procedure perest(k:integer);
var i:integer;
begin
  if k=n-1 then printm
  else
  for i:=k+1 to n do
  begin
    swap(x[k+1],x[i]);
    perest(k+1);
    swap(x[k+1],x[i]);
  end;
end;
begin
  for i:=1 to n do x[i]:=i;
  perest(0);
end.
```

Эта программа работает по следующему принципу: первоначально процедура perest будет рекурсивно вызываться до тех пор, пока не будет распечатан исходный массив *X* (без перестановки элементов):

```
1  2  3  4
```

Потом произойдет отход на шаг назад, перестановка двух последних элементов, и при очередном вызове печать получившегося массива:

```
1  2  4  3
```

после чего массив вернется в первоначальное состояние.

Потом произойдет еще один отход назад и перестановка последнего и третьего с конца элементов, после чего процедура будет снова рекурсивно вызываться, пока не будет распечатан массив *X*:

```
1  4  3  2
```

Далее опять будут переставлены два последних элемента:

```
1  4  2  3
```

и т.д. Особенность данного алгоритма в том, что после окончания он оставляет исходный массив X без изменений. Из программ, которые не используют рекурсию, рассмотрим следующую:

```
const n=4;
label 10,20,30;
var x,c:array[1..n] of integer;
    i,j:integer;
…
{описание процедур swap и printm}
…
begin
     for i:=1 to n do x[i]:=i;
     printm;
     for i:=2 to n do c[i]:=1;
 20: for i:=2 to n do
     begin
       if c[i]<>i then goto 10;
       for j:=2 to i do c[j]:=1;
     end;
     goto 30;
 10: for j:=1 to trunc(i/2) do swap(x[j],x[i+1-j]);
     printm;
     c[i]:=c[i]+1; goto 20;
 30:
end.
```

Массив *X(n)* в этой программе - массив номеров, массив *C(n)* - служебный массив, который используется для отслеживания числа сделанных перестановок.

ЗАДАЧИ НА ДЛИННУЮ АРИФМЕТИКУ

Рассмотрим достаточно популярную в программировании задачу на работу с "длинными" числами. Реально с "астрономическими" или "микроскопическими" числами приходится сталкиваться не так уж и часто. Тем не менее, упражнения, рассматриваемые в этой публикации, могут послужить хорошей тренировкой в области программирования и занять достойное место в классах с углубленным изучением информатики или на кружках по программированию. Алгоритмы, представленные ниже, записаны на Turbo Pascal, версия 7.0. При желании или необходимости они могут легко быть адаптированы к любой другой программной среде.

Диапазон представления целых чисел (**Integer, Word, LongInt**) ограничен, о чем не раз уже говорилось (впрочем, для действительных величин это замечание тоже актуально). Поэтому при решении задач всегда приходится действовать с оглядкой, — как бы не допустить возникновения ошибки выхода за диапазон или переполнения. Например, вычисляя факториал ($n! = 1 * 2 * 3 * \ldots * n$), в диапазоне представления величин типа `Integer` удастся правильно получить только 7! = 5040, а в диапазоне представления типа `LongInt` — 12! = 479001600. Для больших значений, конечно, можно использовать действительные типы данных, но это уже не гарантирует точного результата. Поэтому полезно для получения точных значений при действиях с многозначными числами разработать другие способы представления таких чисел, алгоритмы выполнения арифметических и других операций, процедуры ввода и вывода результатов и т.д.

Покажем реализацию решения такого рода задач на примере умножения одного многозначного числа на другое. Именно эта арифметическая операция наиболее часто используется при решении других задач.

Наиболее естественным способом представления многозначного числа является запись каждого его разряда в виде отдельного элемента линейного массива (или списка, где память под цифру будет отводиться по мере надобности, в то время как в массиве приходится заранее задавать максимальное количество элементов в нем). Пусть (для удобства дальнейших действий) разряды "длинного" числа при записи в массив нумеруются с единицы, начиная с разряда единиц, т.е. цифра из разряда единиц — элемент массива с номером один, цифра из разряда десятков — элемент массива с номером два и т.д. Определим для работы с "длинными" неотрицательными числами тип данных:

```
Const MNax = 2000;
Type Digit = 0..9;
  DlChislo = Array[1..Nmax] Of Digit;
```

Для решения поставленной задачи необходимо уметь выполнять следующие действия:

1) ввод "длинного" числа;

2) собственно умножение двух "длинных" чисел;

3) вывод "длинного" числа;

4) определение количества цифр в записи числа.

Каждую из подзадач реализуем в виде отдельной подпрограммы. Начнем с ввода. Ввести большое число целесообразно в виде строки, а в дальнейшем преобразовать в массив цифр. В процедуре учтен указанный выше способ размещения "длинного" числа в массиве, т.е. с точки зрения пользователя число записывается как бы в обратном порядке.

```
{Процедура преобразования длинного числа, записанного
 в виде строки, в массив цифр; переменная OK принимает значение True,
 если в записи числа нет посторонних символов, отличных от десятичных
 цифр, иначе — false}
Procedure Translate(S : String; Var A : DlChislo; Var OK : Boolean);
Var I : Word;
Begin
  Zero(A); I := Length(S); OK := True;
  While (I >= 1) And OK Do
   Begin
     If S[I] In ['0'..'9']
     Then A[Length(S)- I + 1]:= Ord(S[I]) - 48
     Else OK := False; I := I - 1
   End
End;
```

В процедуре вызывается подпрограмма `Zero(A)`, назначение которой — запись нуля в каждый разряд длинного числа. Вот текст этой процедуры:

```
{Процедура обнуления длинного числа}
Procedure Zero(Var A : DlChislo);
Var I : Integer;
Begin
  For I := 1 To NMax Do A[I] := 0;
End;
```

Таким образом, длинное число записано в массив, где впереди (в качестве элементов с большими номерами) стоят незначащие нули. При выполнении действий и выводе ответа они не учитываются.

Сейчас разработаем функцию определения количества значащих цифр в записи числа, поскольку она потребуется при реализации подпрограммы умножения.

```
{Функция определения количества цифр в записи длинного числа}
  Function Dlina(C : DlChislo) : Integer;
  Var I : Integer;
  Begin
    I := NMax;
    While (I > 1) And (C[I] = 0) Do I := I - 1;
    Dlina := I
  End;
```

При ее разработке было использовано следующее соображение: если число не равно нулю, то количество цифр в его записи равно номеру первой цифры, отличной от нуля, если просмотр числа осуществляется от старшего разряда к младшему. Если же длинное число равно нулю, то получается, что количество цифр в его записи равно одной, что и требовалось.

Ну и, наконец, главная процедура, ради которой и была проделана вся предшествующая работа. При составлении алгоритма используется идея умножения "столбиком", хотя в нашем варианте сложение выполняется не по окончанию умножения, а по ходу его, т.е. перемножив очередные цифры, сразу же добавляем результирующую цифру в нужный разряд и формируем перенос в следующий разряд.

```
{Процедура умножения длинных чисел.
 A, B — множители, C — произведение}
 Procedure Multiplication(A, B : DlChislo; Var C : DlChislo);
 Var I, J : Integer; P : Digit; VspRez : 0..99;
  Begin
    Zero(C);
    For I := 1 To Dlina(A) Do {Цикл по количеству цифр
                          в первом числе}
     Begin
       P := 0; {Первоначально перенос равен нулю}
       For J := 1 To Dlina(B) Do {Цикл по количеству цифр
                             во втором числе}
        Begin
          VspRez := A[I] * B[J] + P + C[I + J - 1];
          C[I + J - 1] := VspRez Mod 10; {Очередное значение цифры в
                                    разряде I + J - 1}
          P := VspRez Div 10 {Перенос в следующий разряд}
        End;
      C[I + J] := P {последний перенос может быть отличен от нуля,
                запишем его в пока ещё свободный разряд}
    End
  End;
```

Сейчас приведем листинг программы целиком.

```
Program DlUmn;
Const NMax = 2000;
Type Digit = 0..9; DlChislo = Array[1..Nmax] Of Digit;
Var S : String;
    M, N, R, F : DlChislo;
    I, MaxF : Word;
    Logic : Boolean;
{Процедура обнуления длинного числа}
Procedure Zero(Var A : DlChislo);
Var I : Integer;
  Begin
    For I := 1 To NMax Do A[I] := 0;
  End;
{Функция определения количества цифр в записи длинного числа}
Function Dlina(C : DlChislo) : Integer;
Var I : Integer;
 Begin
   I := NMax;
   While (I > 1) And (C[I] = 0) Do I := I - 1;
   Dlina := I
 End;
{Процедура печати длинного числа}
Procedure Print(A : DlChislo);
Var I : Integer;
 Begin
```

```
    For I := Dlina(A) DownTo 1 Do Write(A[I] : 1);
    WriteLn
  End;
{Процедура преобразования длинного числа в массив цифр}
Procedure Translate(S : String; Var A : DlChislo;
                    Var OK : Boolean);
Var I : Word;
 Begin
   Zero(A); I := Length(S); OK := True;
   While (I >= 1) And OK Do
   Begin
     If S[I] In ['0'..'9']
     Then A[Length(S) - I+ 1] := Ord(S[I]) - 48
     Else OK := False;
     I := I - 1
   End
 End;
Procedure Multiplication(A, B : DlChislo; Var C : DlChislo);
Var I, J : Integer; P : Digit; VspRez : 0..99;
 Begin
  Zero(C);
  For I := 1 To Dlina(A) Do
  Begin P := 0;
       For J := 1 To Dlina(B) Do
       Begin
         VspRez := A[I] * B[J] + P + C[I + J - 1];
                   C[I + J - 1] := VspRez Mod 10;
                   P := VspRez Div 10
                  End;
       C[I + J] := P
   End
 End;
{Основная программа}
Begin
   Repeat {повторяем ввод, пока число не будет введено правильно}
     Write('Введите первый множитель: ');
     ReadLn(S); Translate(S, M, Logic)
   Until Logic;
   Repeat
     Write('Введите второй множитель: ');
     ReadLn(S); Translate(S, N, Logic)
   Until Logic;
   Multiplication(M, N, R); Print(R)
End.
```

В приведенном листинге **Print** — процедура вывода длинного числа. Предоставим читателю самостоятельно разобраться в алгоритме ее работы.

Вернемся к вычислению факториала. Используя разработанные подпрограммы, определим, значение факториала какого максимального числа можно разместить в памяти при таком представлении длинных чисел.

Вот измененный фрагмент основной программы, решающий поставленную задачу.

```
  Begin
    MaxF := 810;
    Zero(F);
    F[1] := 1;
    For I := 1 To MaxF Do
     Begin
       Str(I, S); {преобразование числа I к строковому типу S}
       Translate(S, M, Logic);
       Multiplication(F, M, F);
       Print(F);
       WriteLn('Факториал числа ', I : 4, ' содержит ', Dlina(F), ' цифр.')
     End
   End.
```