

Какие типы данных и алгоритмы необходимо знать

- Целые - LongInt, Int64
- Вещественные - Comp, Extended
- Массивы статические и динамические
- Записи
- Динамические структуры данных: стеки, очереди, списки, деревья
- Рекурсивные процедуры для вычисления: факториала числа, степени числа, наибольшего общего делителя; определения хода коня, задача коммивояжера, задача о рюкзаке, рекурсивный обход и волновой алгоритм поиска пути в лабиринте.
- Алгоритмы на графы: алгоритм Дейкстры, раскраска графа, поиск в ширину и глубину.
- Динамическое программирование: жадный алгоритм, метод ветвей и границ, метод «разделяй и властвуй».
- Эффективные формулы для работы с геометрическими объектами: площадь и объемы фигур, взаимное расположение фигур на плоскости, построение выпуклого многоугольника.

Определение слов в строке

- **Задача:** Требуется определить массив слов в заданном тексте. Будем считать, что слово состоит из русских и латинских букв и может быть цифр, разделены они между собой пробелом и может быть одним из знаков препинания.
- **Решение:** Будет считывать последовательно каждый символ и проверять на множество символов, не входящих в слово. Если такого символа нет, то накапливаем его в текущем слове. Как только такой символ появится, запоминаем слово в массиве и переходим к накоплению символов для следующего слова.

Определение слов в строке

```
Type mas=array[1..50] of string[25];
```

```
Var
```

```
  S: string;
```

```
  A: mas;
```

```
  C: char;
```

```
  I, k: integer;
```

Определение слов в строке

```
Begin
  I:=0; S:='';
  While not eoln do begin
    Read(c);
    If c in [',', '.', '(', '-', '!', '?', ':', ';'] then begin
      If s='' then continue;
      Inc(i); A[i]:=s;
      S:='';
    End else S:=s+c;
  End;
  If not (s='') then begin inc(i); a[i]:=s; end;
  For k:=1 to i do writeln(a[k]);
End.
```

Сумма больших чисел

Задача: Найти сумму больших чисел A и B , используя строковое представление этих чисел.

Решение: Будем складывать по правилам сложения двух чисел, выделяя каждую цифру числа и работая с ней отдельно.

Суммируя так каждую цифру числа A и B , будем сохранять полученное значение как новый символ строки C .

Вычислять сумму нужно с последних цифр строки, перед этим выравнив длины строк.

Сумма больших чисел

```
var
```

```
A,B,C,s:string;
```

```
p,ost,i,n1,n2,n,k1,k2,cod:integer;
```

```
Begin
```

```
  Readln(A);
```

```
    Readln(B);
```

```
  n1:=Length(A);
```

```
    n2:=Length(B);
```

Сумма больших чисел

```
if n2 >= n1 then n := n2 else n := n1;
```

```
for i := 1 to n1 - n2 do B := '0' + B;
```

```
for i := 1 to n2 - n1 do A := '0' + A;
```

```
ost := 0; C := "";
```

```
for i := n downto 1 do
```

```
begin
```

```
  Val(A[i], k1, cod);
```

```
  Val(B[i], k2, cod);
```

Сумма больших чисел

```
p:=k1+k2+ost;
if p>9 then
  begin
    p:=p mod 10;
    ost:=1;
  end else ost:=0;
str(p,s); C:=s+C;
end;
if ost=1 then C:='1'+C;
writeln(C);
end.
```


Алгоритм нахождения простых чисел

Задача: Найти все простые числа до заданного целого N .

Решение: Применим алгоритм «Решето Эратосфена». Каждое число является делителем любого целого числа.

Будем постепенно исключать такие числа, которые делятся на это число.

Просмотрев все числа до заданного N и удалив лишние, получим множество простых чисел.

Алгоритм нахождения простых чисел

Решето Эратосфена

```
var s: set of byte;  
    i, k, k0, m, N: byte;  
    first: boolean;  
Begin  
    readln(N);  
    {инициализация множества целых чисел от 2 до N}  
    s:=[]; for i:=2 to N do s:=s+[i];  
    {берем первое простое число - 2}  
    k0:=2; m:=1;  
    repeat  
        k:=k0; first:=true;
```

Алгоритм нахождения простых чисел

```
for i:=k+1 to N do { просматриваем все остальные числа }
if i in s then      {если оно из множества }
  if i mod k =0 then {проверяем на делимость }
    s:=s-[i] {если делится нацело, то исключаем из множества}
  else if first then begin
    inc(m); first:=false;
    k0:=i; {иначе фиксируем следующее простое число}
  end;
until first;
{вывод простых чисел}
for i:=2 to N do if i in s then write(i, ' ');
end.
```

Алгоритм нахождения простых чисел

Если $N > 255$, то множество использовать нельзя.

Для проверки каждого числа на простое используем свойство: если число M не делится на $2, 3, \dots, [\text{sqrt}(M)]$, то оно будет простым.

Для быстроты алгоритма исключаем четные числа и проверяем только делимость на нечетные.

Алгоритм нахождения простых чисел

Эффективный алгоритм проверки на простое число

```
var n,i,m,k: longint;  
    flag: boolean;  
Begin  
    readln(k);  
    m:=0;  
    if k>1 then begin Write(2,' '); m:=1; end;  
    For n:=3 to k do begin  
        if n mod 2 =0 then continue;  
        flag:=true;
```

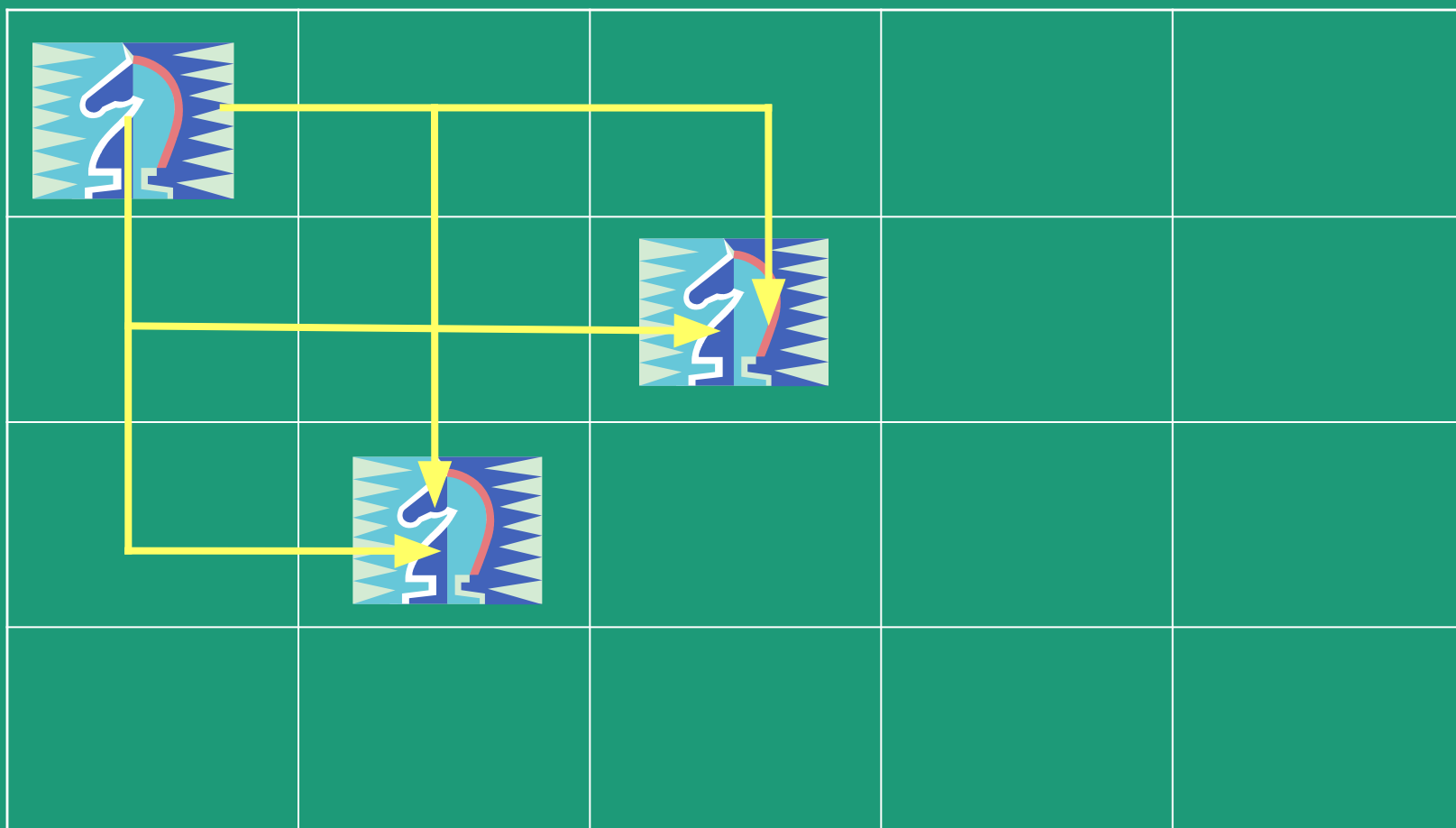
Алгоритм нахождения простых чисел

```
For I:=3 to round(sqrt(n)) do begin
  if n mod i = 0 then
    begin flag:=false; break; end;
end;
if flag then begin inc(m); write(n,' '); end;
inc(n,2);
end;
end.
```

Определение хода коня

- **Задача:** Известно местоположение белого и черного коня. Нужно узнать, за какое наименьшее число ходов белый конь срубит черного, если тот будет стоять на месте.
- **Решение:** Рекурсивно будем переходить на 8 известных перемещений коня, считая каждый раз количество проходов.
- Если во время прохода мы дойдем до местоположения черного коня, то это количество нужно сравнить по минимуму с уже найденным.
- Если такого пути найти не удастся, то вывести сообщение «No solution».

Определение хода коня



Определение хода коня

```
type mas=array[1..2,1..8]of integer;
```

```
    mat=array[1..10,1..10]of byte;
```

```
const v:mas=((1,1,-1,-1,-2,-2,2,2),
```

```
           (2,-2,2,-2,1,-1,1,-1)); { на сколько позиций по X и
```

```
по Y нужно отступить, чтобы перейти на новое место }
```

```
var p, t, i, n, m: byte;
```

```
    h, minh, x, y: integer;
```

```
    flag: boolean;
```

```
    a: mat;
```

Определение хода коня

```
procedure hod (x, y: byte);
var i: byte;
begin
  if a[x, y]=1 then begin { если дошли до места }
    if h<minh then minh:=h; flag:=true; exit; end;
  if a[x, y]=2 then exit else a[x, y]:=2; { если здесь уже были }
  for i:=1 to 8 do { просматриваем остальные ходы }
    if ((x+v[1,i]>=1)and(x+v[1,i]<=n))and
      ((y+v[2,i]>=1)and(y+v[2,i]<=m)) then begin
      inc(h); hod(x+ v[1,i], y+ v[2,i]); dec(h);
    end;
  end;
end;
```

Определение хода коня

Begin

```
assign(input,'input.txt'); reset (input);  
assign(output,'output.txt');rewrite(output);  
readln(n,m); { размеры шахматного поля }  
readln(x,y); { местоположение белого коня }  
readln(p,t); { местоположение черного коня }  
fillchar(a,sizeof(a),0); { инициализация данных }  
a[x,y]:=2; a[p,t]:=1;  
h:=0; minh:=n*m; flag:=false;
```

Определение хода коня

```
for i:=1 to 8 do
  if ((x+v[1,i]>=1)and(x+v[1,i]<=n)) and
     ((y+v[2,i]>=1)and(y+v[2,i]<=m)) then begin
    inc(h);hod(x+v[1,i],y+v[2,i]); dec(h);
  end;
if (not flag) and (minh=n*m) then
  writeln('No solution')
else writeln(minh);
Close(output);
end.
```

Задача о рюкзаке

Задача: Дано N предметов K_i с указанием веса W_i и стоимости каждого предмета V_i . Определить набор предметов T с максимальной стоимостью груза S , вес которого не больше P .

Решение: Для решения этой задачи используем рекурсивный перебор вариантов для массива весов, проверяя условия на достижение оптимального веса P и максимальной стоимости S выбранных предметов из массива T . Каждый раз начинаем перебор с текущего предмета в массиве W .

Для эффективного перебора необходимо отсортировать массив стоимостей по убыванию и затем для одинаковых значений стоимостей массив весов сортируем по возрастанию, например, по методу пузырька.

Задача о рюкзаке

```
Var now, t, w, v:array[1..50] of integer;  
S, Smax, N, P, i:integer;
```

```
Procedure Swap(var A, B: integer);
```

```
Begin
```

```
  A := A xor B;
```

```
  B := A xor B;
```

```
  A := A xor B;
```

```
End;
```

Задача о рюкзаке

```
Procedure Solve (k, P, S:integer);
Var i: integer;
Begin
  If (P<0) then exit;
  If ((k>N) or (P=0)) and (S > Smax) then Begin
    T := now ; Smax := S;
  End
  Else if (k<=N) then begin
    For I := 1 to ( P div W [k] ) do
      Now [k] := I; Solve( k+1, P - I * W [k], S + i* V [k] )
    End;
  End;
End;
```

Задача о рюкзаке

```
Procedure Sort;  
Var I,j,k:integer;  
Begin  
Repeat k:=0;  
For i:=1 to N-1 do  
For j:=1 to n-i do  
If (v[i]>v[i+1]) or ((v[i]=v[i+1]) and (w[i]<w[i+1]))  
then begin Swap(v[i],v[i+1]); Swap(w[i],w[i+1]);  
K:=1; End;  
Until k=0;  
End;
```


Задача о рюкзаке

BEGIN

{ввод данных} Readln(N, P);

For i:=1 to N do Readln(W [i], V [i]);

{сортировка массивов} sort;

{обнуление переменных}

Smax:=0; Fillchar(now,sizeof(now),0);

{вызов рекурсивной процедуры} Solve (1, P, 0);

{вывод результатов} Writeln(Smax);

For i:=1 to N do

 if T [i] <> 0 then writeln(I, ' ', T [i], ' ', W [i], ' ', V [i]);

END.

Алгоритм Дейкстры

Задача: Известны, что все цены неотрицательны. Найти наименьшую стоимость проезда из 1 города в k-ый

Решение: В процессе работы алгоритма некоторые города будут выделенными :

- для каждого выделенного города i хранится наименьшая стоимость пути; при этом известно, что минимум достигается на пути, проходящем только через выделенные города;

- для каждого невыделенного города i хранится наименьшая стоимость пути, в котором в качестве промежуточных используются только выделенные города.

Алгоритм Дейкстры

Множество выделенных городов расширяется на основании следующего замечания: если среди всех невыделенных городов взять тот, для которого хранимое число минимально, то это число является истинной наименьшей стоимостью.

Добавив выбранный город к выделенным, мы должны скорректировать информацию, хранимую для невыделенных городов. При этом достаточно учесть лишь пути, в которых новый город является последним пунктом пересадки, а это легко сделать, так как минимальную стоимость проезда в новый город мы уже знаем.

Алгоритм Дейкстры

```
var lincs: array[1..100,1..100]of real;
```

```
    sum: array[1..100]of real;
```

```
N,m: integer;
```

```
i,j,k: integer;
```

```
r: real;
```

Алгоритм Дейкстры

```
procedure Go(m:integer);  
var i:integer;  
begin  
  for i:=1 to N do if(lincs[i,m]<>0) and  
    (sum[i]>sum[m]+lincs[i,m]) then begin  
    sum[i]:=sum[m]+lincs[i,m];  
    Go(i);  
  end;  
end;
```

Алгоритм Дейкстры

```
begin
  assign(input,'input.txt');reset(input);
  assign(output,'output.txt');rewrite(output)
;
  readln(n); readln(m);
  for i:=1 to N do begin
sum[i]:=2147483647;
    for j:=1 to n do lincs[i,j]:=0; end;
  while not eof do begin
    readln(i,j,r);
```

Алгоритм Дейкстры

```
if (lincs[i,j]>r)or (lincs[i,j]=0) then begin
lincs[i,j]:=r; lincs[j,i]:=r;
end; end;
sum[m]:=0;
go(m);
writeln(sum[1]:0:3);
j:=1;
while j<>m do begin
write(j,' ');
```

Алгоритм Дейкстры

```
k:=0;
for i:=1 to N do if (lincs[i,j]<>0) and (sum[j] = sum[i]+
lincs[i,j]) then begin
    k:=i;
end;
j:=k;
end;
Writeln ( j );
Close (output);
end.
```


Восстановление дерева графа

Задача: дана строка из чисел, полученных последовательным обходом дерева с корня до каждого листочка.

Решение: Сформируем два массива: массив корней и массив листьев. Будем читать каждое число и проверять на головной корень. Полученные пары чисел необходимо проверить на совпадение и исключить совпадающие.

Затем необходимо отсортировать массивы по возрастанию.

Вывод осуществить до тех пор пока есть повторения в первом массиве чисел.

Восстановление дерева графа

Inp.txt:

3 1 7 3 4 2 3 4 5 3 4 6

Out.txt:

7 0

0

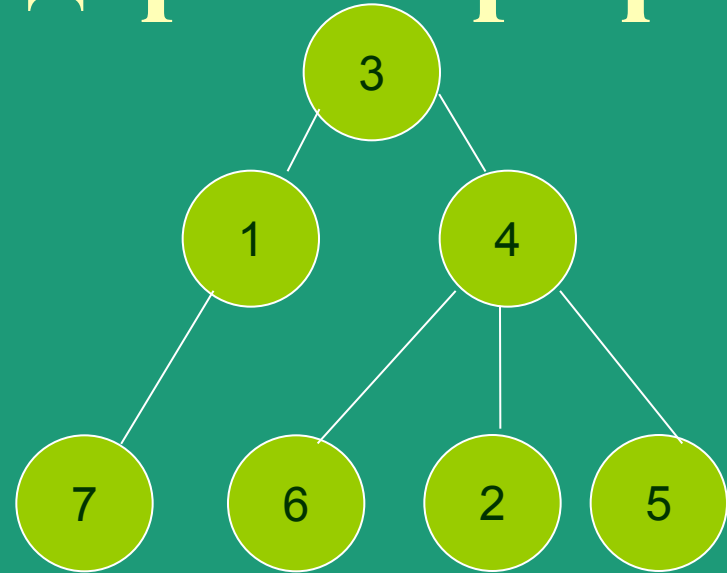
1 4 0

2 5 6 0

0

0

0



Восстановление дерева графа

```
Var flag:boolean;  
    maxn, temp, top, i, n, j: Integer;  
    a,b:array[1..10000] of integer;  
Begin  
    assign(input,'inp.txt');reset(input);  
    assign(output,'out.txt');rewrite(output);  
{ строим два массива }  
    read(top);  
    maxn:=top;{ формируем головной корень графа}  
    a[1]:=top; N:=1;  
    While not eof do Begin  
        Read (temp);
```

Восстановление дерева графа

```
    If temp=top then begin
        a[N]:=top; continue;
    End;
    b[N]:=temp;{формируем новую пару}
    If temp>maxn then maxn:=temp; {ищем количество
участников}
    Flag:=false; {проверяем на совпадение}
    for i:=1 to N-1 do
        if (a[i]=a[N]) and (b[i]=b[N]) then begin
            flag:=true; break; end;
    If not flag then N:=N+1;
    a[N]:=temp;{формируем новый корень графа}
End;
```

Восстановление дерева графа

$a[n]:=0$; {обнуляем лишний корень}

$N:=N-1$; {корректируем размер массива}

{проводим сортировку методом пузырька}

For $i:=1$ to $N-1$ do

For $j:=i+1$ to N do

IF ($a[j-1]>a[j]$) or ($(a[j-1]=a[j])$ and ($b[j-1]>b[j]$)) then

Begin { перестановка }

$a[j]:=a[j]$ xor $a[j-1]$; $a[j-1]:=a[j]$ xor $a[j-1]$; $a[j]:=a[j]$ xor $a[j-1]$;

$b[j]:=b[j]$ xor $b[j-1]$; $b[j-1]:=b[j]$ xor $b[j-1]$; $b[j]:=b[j]$ xor $b[j-1]$;

End;

Восстановление дерева графа

```
{выводим структуру дерева}  
  j:=1;  
  For i:=1 to maxN do Begin  
    While a[j]=i do Begin  
      write(b[j], ' ');  
      j:=j+1;  
    End;  
    Writeln(0);  
  end;  
End.
```

Волновой алгоритм

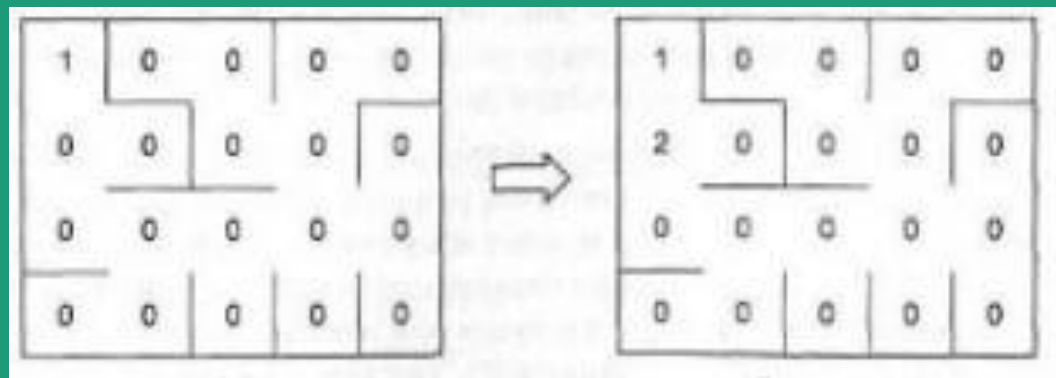
Задача: Дана схема лабиринта в виде матрицы чисел из 0 и 1. Найти наименьшую длину пути и направление движения выхода из лабиринта, если это возможно.

Решение: Пометим сначала все свободные пути лабиринта нулями. Стартовую точку входа пометим единицей. Далее на каждой итерации выполняем действия:

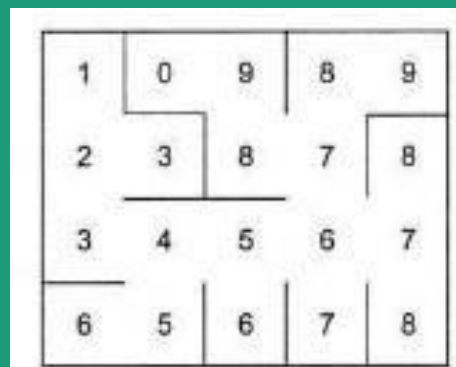
- 1) Найти в лабиринте свободное место, помеченные 1
- 2) Для каждой из четырех соседних с ней свободных мест проверяем два условия: помечена ли она нулем и есть ли стена между двумя свободными местами (выбранной и соседней).
- 3) Если оба условия выполнены, помечаем соседнее свободное место двойкой. И переходим к следующей итерации, т.е. начинаем поиск с 2 и т.д.

Волновой алгоритм

Начало пути



Полученный путь



Волновой алгоритм

Input.txt:

10

0 0 1 0 0 0 0 0 0 0

1 0 0 0 0 1 0 0 1 0

0 0 0 1 1 1 0 0 1 1

0 1 0 0 0 1 0 0 1 0

0 0 0 0 1 1 1 0 1 0

0 0 1 1 1 0 1 0 0 0

0 0 0 1 0 0 1 0 0 0

1 1 0 1 0 0 1 1 1 0

0 1 0 0 0 0 1 0 0 0

0 1 0 0 0 0 1 0 0 0

1 1

10 10

Волновой алгоритм

```
Program voln;
```

```
var Map:array[1..100, 1..100] of integer;
```

```
xs, ys, xe, ye,x, y, i,n: integer;
```

```
mapm: array [1..100, 1..100] of integer;
```

```
moves: integer;
```

```
movesx, movesy: array [1..10000] of  
integer;
```

Волновой алгоритм

```
procedure next (var x,y:integer);  
begin  
  if (x<n) and (mapm[x,y]-mapm[x+1,y]=1) then  
begin x:=x+1; exit; end;  
  if (x>1) and (mapm[x,y]-mapm[x-1,y]=1) then  
begin x:=x-1; exit; end;  
  if (y<n) and (mapm[x,y]-mapm[x,y+1]=1) then  
begin y:=y+1; exit; end;  
  if (y>1) and (mapm[x,y]-mapm[x,y-1]=1) then  
begin y:=y-1; exit; end;  
end;
```

Волновой алгоритм

BEGIN

assign(input, 'input.txt'); reset(input);

assign(output, 'output.txt'); rewrite(output);

{ввод массива лабиринта}

readln(n);

for y:=1 to n do begin

for x:=1 to n do read(map[x,y]);

readln;

end;

readln(xs, ys); {стартовая координата}

readln(xe, ye); {финишная координата}

{проверка корректного ввода}

if (map[xs, ys]=1) or (map[xe, ye]=1) then begin

writeln('Error'); Close(output); exit; end;

Волновой алгоритм

{волновой алгоритм поиска минимального пути}

mapm [xs, ys]:=1; i:=1;

Repeat i:=i+1;

for x:=1 to n do

for y:=1 to n do

if mapm[x,y]=i-1 then begin

if (y<n) and (mapm[x,y+1]=0) and (map[x,y+1]=0) then
mapm[x,y+1]:=i;

if (y>1) and (mapm[x,y-1]=0) and (map[x,y-1]=0) then
mapm[x,y-1]:=i;

if (x<n) and (mapm[x+1,y]=0) and (map[x+1,y]=0) then
mapm[x+1,y]:=i;

if (x>1) and (mapm[x-1,y]=0) and (map[x-1,y]=0) then
mapm[x-1,y]:=i; end;

Волновой алгоритм

```
{ вывод результата для ненайденного пути }  
if i=n*n then begin  
writeln('No solution!');Close(output); exit; end;  
until map[xe,ye]>0;  
{формирование массива координат для прохода по лабиринту }  
moves:=i-1; x:=xe; y:=ye; i:=moves;  
repeat  
movesx[i]:=x;  
movesy[i]:=y;  
next(x,y);  
map[x,y]:=2;  
i:=i-1;  
until (x=xs) and (y=ys);
```

Волновой алгоритм

```
map[xs, ys]:=2;
{вывод найденного пути}
for i:=1 to moves do writeln('x=', movesx[i], ',y=', movesy[i]);
writeln('total: ', moves, ' moves');
{вывод карты лабиринта с полученным путем}
writeln('map:');
for y:=1 to n do begin
for x:=1 to n do write(map[x,y], ' '); writeln; end;
writeln('mapm:');
for y:=1 to n do begin
for x:=1 to n do write(mapm[x,y], ' '); writeln; end;
Close(output);
end.
```

Построение выпуклого многоугольника

Задача: Даны точки на плоскости. Построить по этим точкам выпуклый многоугольник.

Решение: Есть несколько алгоритмов решения данной задачи:

1) Все треугольники, образованные тройками соседних вершин в порядке их обхода, имеют одну ориентацию.

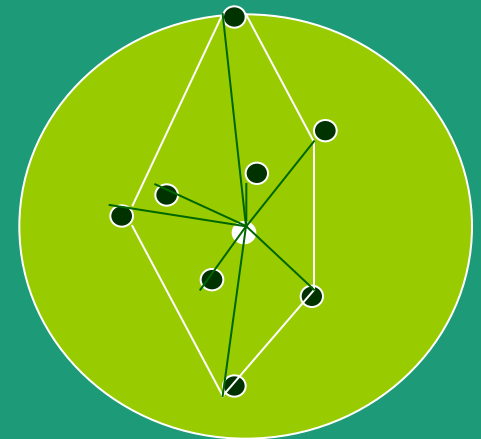
Просматривая все тройки точек по очереди, вычисляем ориентированную площадь треугольника по трем точкам.

Если ориентация точек не совпадает с заданной (по часовой стрелке), то точка лежит вне многоугольника.

Построение выпуклого многоугольника

2) (Алгоритм Грэхема) Пусть найден центр тяжести всех координат. Упорядочим точки относительно полярного угла (можно сравнивать сумму абсолютных значений значений координат).

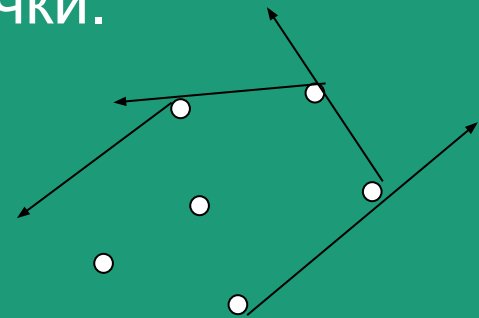
Так как внутренние точки принадлежат некоторому треугольнику, то будем последовательно просматривать отсортированный массив и удалять внутренние вершин. Оставшиеся точки будут являться вершинами выпуклой оболочки.



Построение выпуклого многоугольника

3) *Алгоритм Джарвиса.* Отрезок, определяемый двумя точками, является ребром выпуклой оболочки тогда и только тогда, когда все другие точки множества лежат на отрезке или с одной стороны от него.

Для каждого из этих отрезков можно определить положение остальных $N-2$ точек относительно него, так чтобы угол, образованный лучами имел минимальное значение. Если таких точек несколько, то выбирается точка, находящаяся на максимальном расстоянии от текущей. Таким образом, можно найти все пары точек, определяющих ребра выпуклой оболочки.



Построение выпуклого многоугольника

4) Алгоритм «разделяй и властвуй». Исходное множество из N точек разбивается на два подмножества, каждое из которых будет содержать одну из двух ломаных, которые при соединении образуют выпуклую оболочку.

Для начала нужно определить две точки, которые будут являться соседними вершинами выпуклой оболочки. Проведем прямую через эти две точки.

Нужно найти точку максимально удаленную от прямой. Все точки, лежащие в образованном треугольнике исключаются из дальнейшего рассмотрения. Остальные точки будут делиться на два подмножества: точки, которые лежат левее, и точки, которые лежат правее новых прямых.

С каждым из подмножеств проделываем то же самое. Каждое из них содержит ломаные, которые и дают выпуклую оболочку.

Это реализуется рекурсивной процедурой, которая для данного ей множества возвращает соответствующую часть выпуклой оболочки.

Построение выпуклого многоугольника

Алгоритм Джарвиса

```
Const Maxn=100;
```

```
Type Real=Extended;
```

```
  Tpoint=Record
```

```
    x,y:Real; End;
```

```
  Var A:Array[1..MaxN] Of TPoint; {Массив с  
координатами точек плоскости}
```

```
    N:Integer; {Количество точек}
```

```
    rs:Array[1..MaxN]Of Integer; {Номера точек,  
принадлежащих выпуклой оболочке}
```

```
    M:Integer; {Количество точек в выпуклой  
оболочке}
```

Построение выпуклого многоугольника

{ Поиск номера самой левой нижней точки,
принадлежащей выпуклой оболочке }

```
Function GetLeft:Integer;
```

```
  Var i, Lf: Integer;
```

```
  Begin
```

```
    Lf:=1;
```

```
    For I:=2 To N Do
```

```
      If (A[i].x< A[Lf].x) Or ((A[i].x=0) And (A[Lf].y<A[i].y))
```

```
        Then Lf:=i;
```

```
    GetLeft:=Lf;
```

```
  End;
```

Построение выпуклого многоугольника

{Функция вычисления расстояния между двумя
точками}

```
Function rast( A,B:Tpoint):Real;
```

```
begin
```

```
    rast:=sqrt(sqr(a.x-b.x)+sqr(a.y-b.y));
```

```
end;
```

Построение выпуклого многоугольника

{Функция вычисления угла по трем точкам}

```
Function GetAnгл(A,B,C:Tpoint):Real;
```

```
Var aa,bb,cc :Real;
```

```
Begin
```

```
aa:=(B.x-A.x)*(C.x-B.x)+(B.y-A.y)*(C.y-B.y);
```

```
bb:=Rast(A,B); cc:=Rast(C,B);
```

```
If (cc=0) Then GetAnгл := -11
```

```
Else GetAnгл:=aa/(bb*cc);
```

```
End;
```

Построение выпуклого многоугольника

{Поиск следующей точки выпуклой оболочки}

```
Function GetNext(pr,gn:Tpoint):Integer;
```

```
Var i, fn: Integer; mx, rsx, nw:Real;
```

```
Begin mx:= -10;
```

```
For i:=1 To N Do Begin
```

```
  nw:=GetAngl(pr,gn,A[i]); {найдем угол}
```

```
  If mx-nw<=1.e-6 Then Begin
```

```
    fn:=i; mx:=nw; rsx:=Rast(gn,A[i]); End
```

```
  Else If (nw=mx) Then nw:= Rast(gn,a[i]);
```

```
  If (rsx<nw) Then Begin fn:=i; rsx:=nw; End;
```

```
End; GetNext:=fn; End;
```


Построение выпуклого многоугольника

```
Procedure Solve;  
  Var nx: Integer; ls: TPoint;  
Begin  M:=0; {Количество точек в выпуклой оболочке}  
      nx:=GetLeft; {Находим самую левую и нижнюю точку}  
      While (M=0) Or (nx<>rs[1])Do Begin {Пока не вернулись к первой          Inc(M); rs[M]:=nx; {Очередная точка выпуклой оболочки}  
          If M>1 Then ls:=A[rs[M-1]] {Предыдущая точка выпуклой оболочки}  
              Else Begin ls:=A[rs[1]]; ls.y:=ls.y-1; End;  
          nx:=GetNext(ls,a[rs[M]]) {Поиск следующей точки оболочки}  End;  
      End;
```

Построение выпуклого многоугольника

```
var i:integer;  
BEGIN  
assign(input,'input1.txt');reset(input);  
assign(output,'output1.txt');rewrite(output);  
{ввод массива точек}  
readln(n);  
for i:=1 to n do read(a[i].x,a[i].y);  
{решение} Solve;  
{вывод точек выпуклой оболочки}  
for i:=1 to m do writeln(a[rs[i]].x:0:2,' ',a[rs[i]].y:0:2);  
Close(output);  
END.
```