Решение олимпиадных задач

Требования к программам

- Допускаются следующие среды по языкам программирования:
  - Turbo Pascal (Borland Pascal)
  - QBasic (Turbo Basic)
  - Turbo C++ (Borland C++)
- Визуальные среды разрешаются к использованию в консольном режиме без использования форм.

Требования к программам

- Работать с входными и выходными файлами
- Выдерживать требования к размеру файла
- Выполнение программы за заданное время, выделенное на один тестовый входной файл
- Не разрешается использование
  - Дополнительных модулей и библиотек
  - Вставок ассемблеровских кодов
  - Дополнительных выводов на консоль экрана

Какие задачи можно считать олимпиадными

- Задачи на полный перебор вариантов решений
- Задачи с диапазоном данных, превышающим стандартные типы данных Integer, Real, String.
- Задачи на нестандартный подход к решению задачи
- Задачи с использованием графов

Какие типы данных используются

- Целые LongInt
- Вещественные Extended
- Массивы статические и динамические
- Записи
- Динамические структуры данных: стеки, очереди, списки, деревья

Какие алгоритмы необходимо знать

- Алгоритмы со строками: сумма больших чисел, выделение слов в строке, определение палиндрома
- Алгоритмы с целыми числами: определение простых чисел, делителей числа, суммы цифр числа
- Эффективные методы сортировки и бинарный поиск

Какие алгоритмы необходимо знать

- 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.

Что еще нужно уметь

- Знать операторы языка Паскаль
- Знать необходимые ключи компиляции
- Уметь работать с командной строки
- Уметь работать с текстовыми файлами
- Выполнять отладку программы
- Создавать самим тесты к программе
- Проверять работу программы на время
- Оценивать эффективность алгоритма
- Выявлять наилучший метод решения задачи

Разбор задач

1. Определение слов в строке
2. Сумма больших чисел
3. Алгоритмы нахождения простых чисел
4. Определение хода коня
5. Задача о рюкзаке
6. Алгоритм Дейкстры
7. Восстановление дерева графа
8. Волновой алгоритм
9. 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 появиться, 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, …, [sqrt(M)], то оно будет простым.

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

Алгоритм нахождения простых чисел

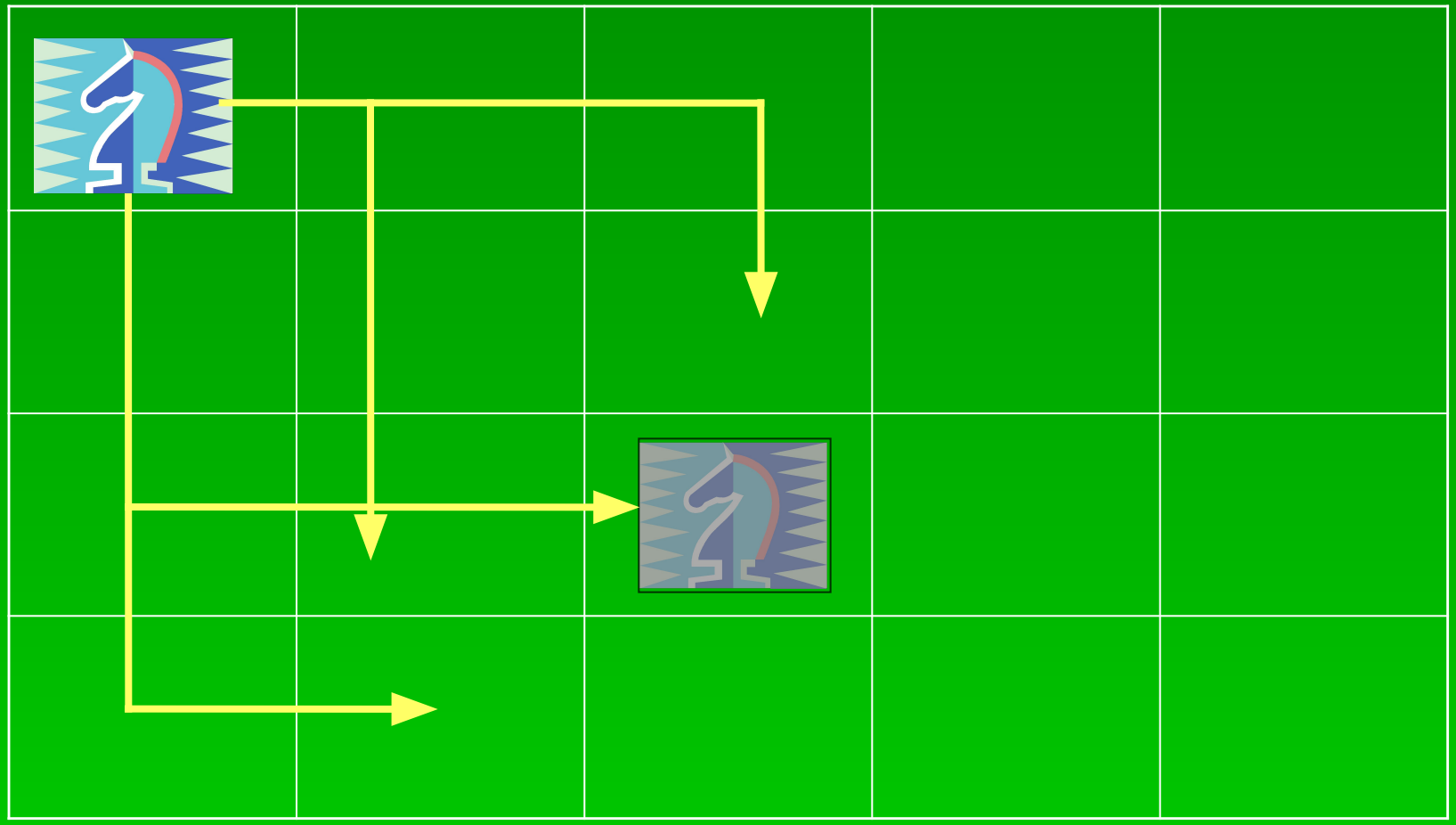
Эффективный алгоритм проверки на простое число

```
var n,i,m: longint;
    flag: boolean;
Begin
Write(2,' '); m:=1; n:=3;
while n<= maxlongint do begin
     flag:=true; i:=3;
     while i<= round(sqrt(n)) do begin
       if n mod i = 0 then begin  flag:=false;  break; end;
       inc(i,2);
     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 удается, 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 У
  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;
```

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 Ki с указанием веса Wi и стоимости каждого предмета Vi. 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;
```

Задача о рюкзаке

```
Procedure Sort;
Var i:integer;
Begin
For i:=1 to N-1 do
For j:=I to n-1 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]);
End;
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,filesize(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.

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]);
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

Input.txt:

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

Output.txt:

7 0

0

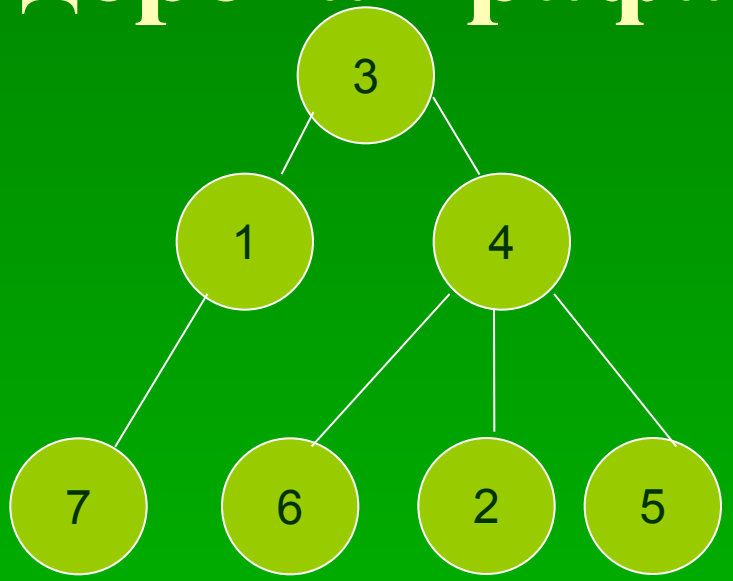
1 4 0

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

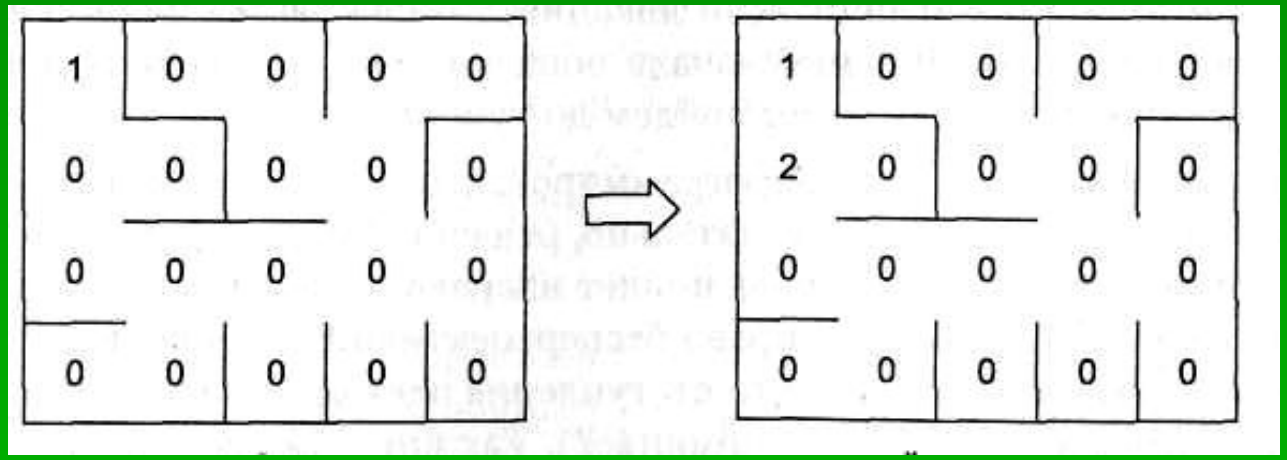
Задача: Дана схема лабиринта в виде матрицы чисел из 0 и 1. Найти наименьшую длину пути и направление движения выхода из лабиринта, если это возможно.

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

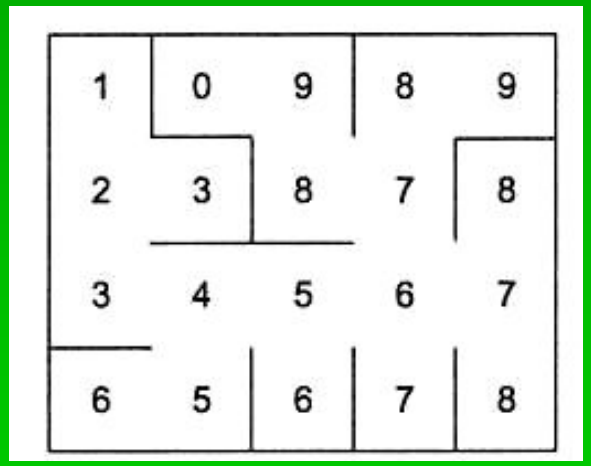
1) Найти в лабиринте свободное место, помеченные 1
2) Для каждой из четырех соседних с ней свободных мест проверяем два условия: помечена ли она нулем и есть ли стена между двумя свободными местами (выбранной и соседней).
3) Если оба условия выполнены, помечаем соседнее свободное место двойкой. И переходим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

Волновой алгоритм

Начало пути



Полученный путь



Волновой алгоритм

```
Program voln;
Uses crt;
Const Map:array[1..10, 1..10] of byte=(
(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));
```

Волновой алгоритм

```
var   xs, ys, xe, ye:byte;
      x, y, i: byte;
mapm: array [1..10, 1..10] of byte;
moves: byte;
movesx, movesy: [1..100] of byte;
```

Волновой алгоритм

```
procedure next (var x,y:byte);
begin
      if (x<10) 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<10) 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
{вывод массива лабиринта}
    for y:=1 to 10 do begin
     for x:=1 to 10 do do write(map[x,y],' ');
     writeln;
    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!!!'); readln; halt; end;
```

Волновой алгоритм

```
{волновой алгоритм поиска минимального пути}
    mapm [xs, ys]:=1;
    i:=1;
    repeat
     i:=i+1;
     for x:=1 to 10 do
     for y:=1 to 10 do
         if mapm[x,y]=i-1 then begin
    if (y<10) and (mapm[x,y+1]=0) and (map[x,y+1]=0)
    then mapm[x,y+1]:=i;
```

Волновой алгоритм

```
if (y>1) and (mapm[x,y-1]=0) and (map[x,y-1]=0) then
    mapm[x,y-1]:=i;
if (x<10) 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;
{ вывод результата для ненайденного пути }
if i=100 then begin
    writeln('No solution!'); readln; halt; end;
until mapm[xe,ye]>0;
```

Волновой алгоритм

```
{ 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=', moves[i], ',y=',
    movesy[i]);
writeln('total: ', moves, ' moves');
{вывод карты лабиринта с полученным путем}
    for y:=1 to 10 do begin
     for x:=1 to 10 do do write(map[x,y],' ');
     writeln;
    end;
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.

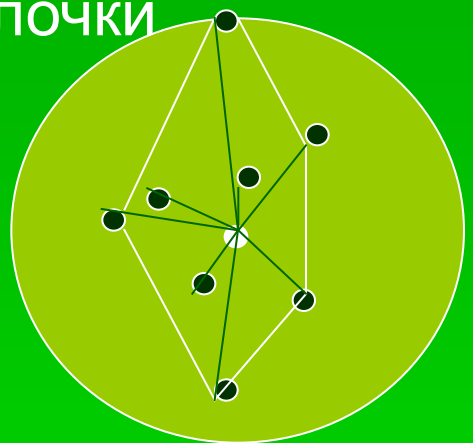
Просматривая все тройки точек по очереди, вычисляем ориентированную площадь треугольника по трем точкам.

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 координат).

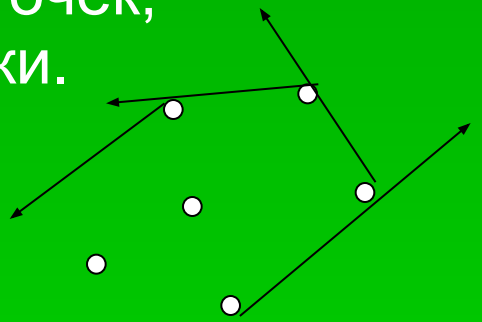
Так как внутренние точки принадлежат некоторому треугольнику, то будем последовательно просматривать отсортированный массив и удалять внутренние вершин. Оставшиеся точки будут являться вершинами выпуклой оболочки



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 лежать 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

```
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
   точек}
  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

```
{Поиск следующей точки выпуклой оболочки}
Function GetNext(Const 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 RealLess(mx,nw) Then Begin
    fn:=i; mx:=nw: rsx:=Rast(gn,A[i]); End
```

Построение выпуклого многоугольника

```
    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

```
{Функция вычисления угла по трем точкам}
Function GetAngl(Const 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 GetAngl :=-11 Else
    GetAngl:=aa/(bb*cc);
End;
```

Задачи для самостоятельного решения

1. Поход в музей
2. Упорядоченные числа
3. Бильярд
4. Лягушка-царевна
5. Расписание процессора
6. Привал в лесу

Задачи для самостоятельного решения

1. **Поход в музей.** В одном детском саду воспитатели решили сводить детей на экскурсию в музей. Как обычно бывает в таких случаях, для сохранения порядка, детей решили строить парами. Но дело в том, что некоторые дети не хотят идти друг с другом. Это привело к тому, что воспитатели не смогли сводить детей в музей. Для решения этой проблемы директор детсада опросил всех детей и узнал, кто с кем не хочет идти. Итак, у директора есть список детей, и для каждого ребёнка - список тех, с кем он не хочет быть в паре. Необходимо посчитать максимальное количество пар, которые можно создать.

Задачи для самостоятельного решения

2. **Упорядоченные числа.** Натуральное число назовем упорядоченным, если его цифры оказываются упорядоченными по неубыванию при просмотре разрядов в порядке от старших к младшим. Таковыми, например, являются числа 111, 123, 15, 1123. Вам требуется посчитать количество упорядоченных чисел в диапазоне [10, N], где N - заданное натуральное число.

Задачи для самостоятельного решения

3. **Бильярд.** Дядя Вася решил поиграть в бильярд. Бильярдное поле задается системой неравенств $0 \le x \le X$, $0 \le y \le Y$ (X, Y - размеры поля) и имеет четыре лузы по углам поля с координатами (0,0), (X,0), (0,Y), (X,Y). На поле остался всего один шар с координатами (x,y). Дядя Вася хочет ударить по шару в направлении (u,v). Помогите ему выяснить, попадет ли шар в лузу, и если попадет, то сколько ударов об стенку сделает шар перед тем, как попасть в лузу.

Считать, что шар и лузы точечные, и шар попадает в лузу, когда его координаты совпадают с координатами лузы. После удара шар начинает двигаться в направлении (u,v) и в процессе движения не теряет скорость (то есть не останавливается). Удар о стенку считать абсолютно упругим (то есть касательная к стенке скорость сохраняется, а ортогональная стенке скорость меняет знак, сохраняя при этом абсолютную величину).

Попадание в лузу не является ударом о стенку. Касание шаром стенки, как и удар шара, лежащего у стенки, также не является ударом о стенку.

Задачи для самостоятельного решения

4. **Лягушка-Царевна.** Лягушка (ну, почти царевна) увидела, куда попала стрела, пущенная Иваном-Царевичем. А попала она на островок, и добраться до этого островка лягушка может только лишь, прыгая по кочкам. Кочки расположены на прямой и имеют целочисленные координаты xi (i = 0, ... , N). Лягушка делает один прыжок в секунду. Она прыгает только вперед. Длина ее прыжка по сравнению с предыдущим может увеличиваться не более чем на Amax.

В начальный момент времени (t = 0) она находится на кочке с координатой x0 = 0 и имеет нулевую скорость. За конечное время претендентка на царский трон должна попасть точно на островок с координатой xN (конечная скорость значения не имеет). При этом лягушка должна успеть подобрать стрелу до появления Ивана-Царевича (ну, чтобы он ее в жены взял). Время его прихода ей известно.

Задачи для самостоятельного решения

Для того, чтобы иметь ускорение Amax = 1, лягушке необходимо проглотить одного волшебного комара, Amax = 2 - соответственно, двух комаров и т.д. Ее задача состоит в том, чтобы, потратив как можно меньше волшебных комаров (их запас у нее весьма скромный), добраться до стрелы за время, не большее заданной величины T.

Задачи для самостоятельного решения

5. **Расписание процессора.** При создании новой операционной системы SOS программисты столкнулись со следующей проблемой. Многие программы при своем исполнении занимают процессор не все время, а иногда исполнение прерывают и освобождают процессор на время работы с иными устройствами компьютера. Пока уже исполняемые программы освобождают процессор на время работы с иными устройствами компьютера, возможно параллельное исполнение других программ. Порядок запуска программ не имеет значения. Но программы не должны ждать освобождения процессора, когда он им станет опять необходим.

Задачи для самостоятельного решения

- Разумным казалось просто выполнять задачи последовательно. Однако в этом случае работа системы оказывается совсем не эффективной, так как очень часто возможно параллельное исполнение программ. Для простоты Вам предлагается разобраться с ситуацией, когда необходимо выполнить всего две такие задачи, причем затратив на их выполнение наименьшее время. Временем выполнения набора задач называется количество тактов процессора, прошедших с момента начала исполнения первой запущенной задачи и до момента конца исполнения всего множества задач.

Задачи для самостоятельного решения

6. **Привал в лесу.** Туристы разбивали лагерь и решили натянуть навес между деревьями. Лесок в данном месте по площади небольшой и довольно редкий, так что деревьев немного: не более 20, но не менее 3. В любом случае есть три дерева, не лежащие на одной прямой.

Туристы оказались хорошо экипированными, они имели при себе ноутбук и терминал GPS, и поэтому решили определить координаты всех деревьев на интересующем их участке леса при помощи GPS и написать программу, которая нашла бы оптимальное место для навеса.

Оптимальное место для навеса - это участок, имеющий максимальную площадь. Участок должен иметь форму выпуклого многоугольника, вершины которого совпадают с координатами деревьев. Участок не должен содержать внутри ни одного дерева, однако деревья могут находиться на его сторонах.