算法初步(made by Dark);

图论算法

一．最短路

1． Spfa 链表实现：

```
program spfa;
type pointer=^rec;
rec=record
next:pointer;
spot:longint;
len:longint;
end;
var now,head:array[0..10000] of pointer;
p1:pointer;
open,closed,temp,x,y,z,i,n,m:longint;
que:array[0..10000] of longint;
visit:array[0..10000] of boolean;
dist,d:array[0..10000] of longint;
procedure spfa(start:longint);
var t:longint;
begin
fillchar(dist,sizeof(dist),0);
fillchar(visit,sizeof(visit),false);
fillchar(que,sizeof(que),0);
open:=1;
closed:=0;
que[1]:=start;dist[start]:=0;visit[start]:=true;
while closed<=open do
begin
inc(closed);
temp:=que[closed];visit[temp]:=false;p1:=head[temp];
while (p1<>nil)and(p1^.spot<>0) do
begin
if dist[temp]+p1^.len<dist[p1^.spot] then
begin
dist[p1^.spot]:=dist[temp]+p1^.len;
inc(d[temp]);
if d[temp]>n then exit;
if visit[p1^.spot]=false then
```

```
begin
inc(open);
que[open]:=p1^.spot;
visit[p1^.spot]:=true;
end;
p1:=p1^.next;
end
else p1:=p1^.next;
end;
end;
end;
begin
readln(n,m);
for i:=1 to n do
begin
new(p1);
head[i]:=p1;
now[i]:=p1;
end;
for i:=1 to m do
begin
readln(x,y,z);
now[x]^.len:=z;
now[x]^.spot:=y;
new(p1);
now[x]^.next:=p1;
now[x]:=p1;
now[y]^.len:=z;
now[y]^.spot:=x;
new(p1);
now[y]^.next:=p1;
now[y]:=p1;
end;
for i:=1 to n do
now[i]:=nil;
spfa(1);
end.
```

2.dijkstra 算法

用于稠密图，时间 O(n^2);

```
dvar map:array[0..1000,0..1000] of longint;
d:array[0..1000] of longint;
v:array[0..1000] of boolean;
i,j,n,m,k,t,min,ans,start,mb:longint;
begin
```

```
fillchar(d,sizeof(d),$2f);
readln(n,m);
start:=1;
mb:=n;
for i:=1 to n do
begin
for j:=1 to n do
begin
read(map[i,j]);
end;
end;
for i:=1 to n do
begin
d[i]:=map[start,i];
v[i]:=false;
end;
v[start]:=true;
for i:=2 to n do
begin
min:=maxlongint;
k:=0;
for j:=1 to n do
  if(not v[j])and(d[j]<min) then begin
  min:=d[j];
  k:=j;end;
  if k=mb then begin writeln(d[k]);exit; end;
  if (k=0) or(min=maxlongint) then exit;
  v[k]:=true;
  for j:=1 to    n do
  if (not v[j])and(d[k]+map[k,j]<d[j]) then d[j]:=d[k]+map[k,j];
  end;
  writeln(d[mb]);
  end.
```

3.Bellman-ford 算法

用于稀疏图和判断负环，时间 O（NM）；

见老书。

4．FLOYD 算法

优化版

二．最小生成树：

1．　PRIM 算法

用于稠密图，时间 O（N^2）；

```
var map:array[0..1000,0..1000] of longint;
```

```
dist:array[0..1000] of longint;
d:array[0..1000] of 0..1;
i,j,n,m,min,ans,k:longint;
procedure prim;
begin
dist[1]:=0;
for i:=1 to n do
dist[i]:=map[1,i];
for i:=1 to n-1 do
begin
min:=maxlongint;
k:=0;
for j:=1 to n do
if (dist[j]<min)and(d[j]=0) then begin
min:=dist[j];
k:=j;
end;
if k<>0 then begin
inc(ans,min);
d[k]:=1;
end;
for j:=1 to n do
if (map[k,j]<dist[j])and(d[j]<>1) then begin
dist[j]:=map[k,j];
end;
end;
end;
begin
readln(n);
for i:=1 to n do
begin
for j:=1 to n do
begin
read(map[i,j]);
if (i<>j)and(map[i,j]=0) then map[i,j]:=maxlongint;
end;
end;
d[1]:=1;
prim;
writeln(ans);
end.
```

2.KRUSKAL 算法

优化版，时间 O（N+MLOGM）；

```
type node=record
a,b,w:longint;
end;
var edge:array[0..10000] of node;
f:array[0..10000] of longint;
n,m,j,i,k,t,min:longint;
//procedure sort
function gefa(v:longint):longint;
begin
if  f[v]=v  exit(v);
f[v]:=gefa(f[v]);
exit(f[v]);
end;
procedure union(a,b,c:longint);
begin
a:=gefa(a);
b:=gefa(b);
if a<>b then begin
f[b]:=a;
min:=min+c;
inc(k);
end;
end;
procedure kruskal;
var i,k1,k2:longint;
begin
for i:=1 to n do
f[i]:=i;
for i:=1 to m do
begin
if k=n-1 then break;
union(edge[i].a,edge[i].b,edge[i].w);
end;
if k<n-1 then begin writeln('error');    exit;end;
end;
begin
readln(n,m);
for i:=1 to m do with edge[i] do
readln(a,b,w);
kruskal;
writeln(min);
end.
```

三．拓扑排序：

可作关键路径，找树的最长链。

```
var i,j,n,m,k,t,x,y:longint;
p,v:array[0..10000] of longint;
g:array[0..1000,0..1000] of longint;
begin
readln(n,m);
for i:=1 to m do
begin
        readln(x,y);
            g[x,y]:=1;
            inc(v[x]);
            end;

 i:=0;
 while (i<n) do
 begin
 j:=1;
 while (v[j]<>0) do inc(j);
 v[j]:=1000;
 for k:=1 to n do
 begin
 if g[k,j]=1 then begin dec(v[k]);g[k,j]:=0; end;end;
 inc(i);
 p[i]:=j;
 end;
 if i=0 then writeln('no')
 else for j:=i downto    2 do
 write(p[j],' ');
 writeln(p[1]);
 end.
```

四．欧拉回路问题

1.要找出欧拉路径，先要找出出发点，然后按以下步骤遍历：

如果该点没有相连的点，就将该点加进路径中然后返回;

如果该点有相连的点，对每一个连点，先删除连边，再遍历之,直到该点没有相连的点,把当前这个点加入路径中.

2.两个点之间有多条边的图,也可以使用相同的算法。

3. 有自环的图也可以使用这样的算法, 前提是我们认为自环给这

个节点增加的度为 2.

4.如果一个有向图是强连通的 (不考虑入度出度都是 0 的点) 并且每个点的入度等于出度。那么这个图有欧拉回路而这个算法仍然适用。

5.在有向图中寻找一个欧拉路径是十分困难的。

```
欧拉回路
const    maxn=100;
var
    g:array[1..maxn,1..maxn] of longint;
    du:array[1..maxn] of longint;
    circuit:array[1..maxn] of longint;
    n,circuitpos,i,j,start,oddnumber:longint;

procedure setIO;
begin
        assign(input,'one.in');
        reset(input);
        assign(output,'one.out');
        rewrite(output);
end;

procedure find_circuit(i:longint);
var j:longint;
begin
        for j:=1 to n do
            if g[i,j]=1 then
                begin
                        g[i,j]:=0;
                        g[j,i]:=0;
                        find_circuit(j);
                end;
        circuitpos:=circuitpos+1;
        circuit[circuitpos]:=i;
end;

begin
//        setIO;
```

```
    read(n);
    for i:=1 to n do
    begin
        du[i]:=0;
        for j:=1 to n do
        begin
            read(g[i,j]);
            du[i]:=du[i]+g[i,j];
        end;
    end;

    start:=1; oddnumber:=0;
    for i:=1 to n do
      if du[i] mod 2 =1 then
        begin
            start:=i;
            oddnumber:=oddnumber+1;
        end;

    if (oddnumber>2)or(oddnumber=1)
      then writeln('No Solution!')
      else begin
             circuitpos:=0;
             find_circuit(start);
             for i:=1 to circuitpos-1 do write(circuit[i],'--->');
             writeln(circuit[circuitpos]);
           end;
    close(input);close(output);
end.
```

五．图中最小环（floyd）：

```
    min:=maxn;
  for k:=1 to n do
    begin
      for i:=1 to n do
        for j:=1 to n do
          if (i<>j) and (f[i,j]+g[i,k]+g[j,k]<min) then
            min:=f[i,j]+g[i,k]+g[j,k];
      for i:=1 to n do
        for j:=1 to n do
          if (i<>j) and (f[i,k]+f[k,j]<f[i,j]) then
            f[i,j]:=f[i,k]+f[j,k];
```

```
        end;
    writeln(min);
```

六．强连通分量：

有向图中，u 可达 v 不一定意味着 v 可达 u．相互可达则属于同一个强连通分量(Strongly Connected Component，SCC)

有向图的强连通分量

最关键通用部分：强连通分量一定是图的深搜树的一个子树。

Trajan 算法

1．算法思路：

这个算法思路不难理解，由开篇第一句话可知，任何一个强连通分量，必定是对原图的深度优先搜索树的子树。那么其实，我们只要确定每个强连通分量的子树的根，然后根据这些根从树的最低层开始，一个一个的拿出强连通分量即可。那么身下的问题就只剩下如何确定强连通分量的根和如何从最低层开始拿出强连通分量了。

那么如何确定强连通分量的根，在这里我们维护两个数组，一个是 indx[1..n]，一个是 mlik[1..n]，其中 indx[i]表示顶点 i 开始访问时间，mlik[i]为与顶点 i 邻接的顶点未删除顶点 j 的 mlik[j]和 mlik[i]的最小值(mlik[i]初始化为 indx[i])。这样，在一次深搜的回溯过程中，如果发现 mlik[i]==indx[i]那么，当前顶点就是一个强连通分量的根，为什么呢？因为如果它不是强连通分量的跟，那么它一定是属于另一个强连通分量，而且它的根是当前顶点的祖宗，那么存在包含当前顶点的到其祖宗的回路，可知 mlik[i]一定被更改为一个比 indx[i]更小的值。

至于如何拿出强连通分量，这个其实很简单，如果当前节点为一个强连通分量的根，那么它的强连通分量一定是以该根为根节点的(剩下节点)子树。在深度优先遍历的时候维护一个堆栈，每次访问一个新节点，就压入堆栈。现在知道如何拿出了强连通分量了吧？是的，因为这个强连通分量时最先被压人堆栈的，那么当前节点以后压入堆栈的并且仍在堆栈中的节点都属于这个强连通分量。当然有人会问真的吗？假设在当前节点压入堆栈以后压入并且还存在，同时它不属于该强连通分量，那么它一定属于另一个强连通分量，但当前节点是它的根的祖宗，那么这个强连通分量应该在此之前已经被拿出。现在没有疑问了吧，那么算法介绍就完了。

例题：

Description

由于外国间谍的大量渗入，国家安全正处于高度的危机之中。如果 A 间谍手中掌握着关于 B 间谍的犯罪证据，则称 A 可以揭发 B。有些间谍收受贿赂，只要给他 们一定数量的美元，他们就愿意交出手中掌握的全部情报。所以，如果我们能够收买一些间谍的话，我们就可能控制间谍网中的每一分子。因为一旦我们逮捕了一个 间谍，他手中掌握的情报都将归我们所有，这样就有可能逮捕新的间谍，掌握新的情报。

我们的反间谍机关提供了一份资料，色括所有已知的受贿的间谍，以及他们愿意收受的具体数额。同时我们还知道哪些间谍手中具体掌握了哪些间谍的资料。假设总共有 n 个间谍(n 不超过 3000)，每个间谍分别用 1 到 3000 的整数来标识。

请根据这份资料，判断我们是否有可能控制全部的间谍，如果可以，求出我们所需要支付的最少资金。否则，输出不能被控制的一个间谍。

Input

第二行是整数 p。表示愿意被收买的人数，1<=p<=n。

接下来的 p 行，每行有两个整数，第一个数是一个愿意被收买的间谍的编号，第二个数表示他将会被收买的数额。这个数额不超过 20000。

紧跟着一行只有一个整数 r，1<=r<=8000。然后 r 行，每行两个正整数，表示数对(A, B)，A 间谍掌握 B 间谍的证据。

Output

如果可以控制所有间谍，第一行输出 YES，并在第二行输出所需要支付的贿金最小值。否则输出 NO，并在第二行输出不能控制的间谍中，编号最小的间谍编号。

Sample Input

【样例 1】

```
3
2
1 10
2 100
2
1 3
2 3
```

【样例 2】

```
4
```

2
1 100
4 200
2
1 2
3 4

Sample Output

【样例 1】
YES
110

【样例 2】
NO
3

解析

这道题是一道典型的求强连通分量的题，首先解释一下什么是有向图的强连通分量，如果一个图中的
任意节点与其它节点都有边连接，我们就把这样的图称作强连通。一个有向图中的强连通分量是它的
最大连通子图，如果每个强连通分量被缩成一个节点，由此产生的图是一个有向非循环图。而这道题
中，间谍与间谍之间的关系是有向的，所以说，每一个强连通分量中的任意一个间谍被收买，整个强
连通分量中的间谍都会玩完。因此，我们只需求出所有的强连通分量，并且满足入度为零（若不为零，
则证明可以通过收买其他强连通分量中的间谍达到目的），还要求出每一个强连通分量的最小收买代
价，相加即可。

```
    program  age;
type  list=^data;
  data=record
  num:integer;
  next:list;
  end;
var  e:array[1..3000]of  list;
  pred,a,b,c,g,f:array[1..3000]of  integer;
```

```
  mark,alive:array[1..3000]of boolean;
  n,m,k,i,j,t,top,v:integer;
  p:list;
  h:boolean;
  s:longint;
function min(x,y:integer):integer;
begin
  if x<y then exit(x);
  exit(y);
end;
procedure dfs(v:integer);
var i:integer;
  p:list;
begin
  pred[v]:=t;a[v]:=t;p:=e[v];
  inc(t);inc(top);b[top]:=v;
  mark[v]:=true;alive[v]:=true;
  while p<>nil do
  begin
  if mark[p^.num]=false then
  begin
  dfs(p^.num);
  pred[v]:=min(pred[v],pred[p^.num]);
  end
  else if alive[p^.num]=true then pred[v]:=min(pred[v],a[p^.num]);
  p:=p^.next;
  end;
  if a[v]=pred[v] then
  begin
  while b[top]<>v do
  begin
  c[b[top]]:=v;
  alive[b[top]]:=false;
  dec(top);
  end;
  c[v]:=v;
  alive[v]:=false;
  dec(top);
```

```
  end;
end;
begin
  readln(n);
  for i:=1 to n do
  begin
  e[i]:=nil;
  g[i]:=maxint;
  end;
  readln(m);
  for i:=1 to m do
  begin
  read(v);
  readln(g[v]);
  end;
  readln(k);
  for i:=1 to k do
  begin
  readln(v,s);
  new(p);
  p^.num:=s;
  p^.next:=e[v];
  e[v]:=p;
  end;
  for i:=1 to n do
  if mark[i]=false then dfs(i);
  fillchar(mark,sizeof(mark),false);
  for i:=1 to n do
  if g[i]<g[c[i]] then g[c[i]]:=g[i];
  for i:=1 to n do
  begin
  p:=e[i];
  while p<>nil do
  begin
  if c[i]<>c[p^.num] then inc(f[c[p^.num]]);
  p:=p^.next;
  end;
  end;
```

```
 fillchar(alive,sizeof(alive),false);
 for i:=1 to n do
 if mark[c[i]]=false then mark[c[i]]:=true;
 s:=0;
 for i:=1 to n do
 if(mark[i]=true)and(f[i]=0)then
 begin
 inc(s,g[i]);
 if g[i]=maxint then
 begin
 h:=true;
 alive[i]:=true;
 end;
 end;
 if h=false then
 begin
 writeln('YES');
 writeln(s);
 end;
 if h=true then
 for i:=1 to n do
 if alive[c[i]]=true then
 begin
 writeln('NO');
 writeln(i);
 break;
 end;
end.
```

七。LCA 问题

最近公共祖先(Least Common Ancestors)

对于有根树 T 的两个结点 u、v，最近公共祖先 LCA(T,u,v)表示一个结点 x，满足 x 是 u、v 的祖先且 x 的深度尽可能大。另一种理解方式是把 T 理解为一个无向无环图，而 LCA(T,u,v)即 u 到 v 的最短路上深度最小的点。

这里给出一个 LCA 的例子：

对于 T=<V,E>

V={1,2,3,4,5}

E={(1,2),(1,3),(3,4),(3,5)}

则有：

LCA(T,5,2)=1

LCA(T,3,4)=3

LCA(T,4,5)=3

```
type data=record a,b:longint;end;
var   A:array[1..100000*2]of data;
       T:array[1..100000*2]of data;
       N:longint;
       Star,fat:array[1..100000+1]of longint;
       Lavel:array[1..100000]of longint;
       Fa:array[0..100000]of longint;
       check:array[1..100000]of boolean;
       ans:longint;
function find(x:longint):longint;
  begin
  if fa[x]=0 then exit(x) else
     begin
     find:=find(fa[x]);
     fa[x]:=find;
     end;
  end;
procedure KP(f,e:longint);
var K,i,j:longint;Temp:data;
  begin
  K:=A[f+random(e-f+1)].A;
  i:=f;j:=e;
  repeat
  while (i<=j)and(A[i].A<K) do inc(i);
  while (i<=j)and(K<A[j].A) do dec(j);
  if i<=j then
     begin
     Temp:=A[i];A[i]:=A[j];A[j]:=Temp;
```

```
        inc(i);dec(j);
        end;
    until i>j;
    if F<j then KP(F,j);
    if i<E then KP(i,E);
    end;
procedure initAsk;
var M,R1,R,E,i:longint;
    begin
    readln(E);
    M:=0;R:=1;
    for i:=1 to E do
        begin
        R1:=R;
        readln(R);
            inc(M);a[M].a:=R;a[m].b:=R1;
            inc(M);a[M].a:=R1;a[m].b:=R;
        end;
    kp(1,M);
    Star[N+1]:=M+1;
    inc(M);
    for i:=N downto 1 do
        begin
        while (M>1)and(a[M-1].a=i) do dec(M);
        Star[i]:=M;
        end;
    M:=8;
    end;
procedure initTree;
var M:Longint;
        i,la,lb:longint;
    begin
readln(N);
M:=0;
for i:=1 to N-1 do
    begin
    readln(la,lb);
        inc(M);A[M].A:=la;A[M].b:=lb;
        inc(M);A[M].A:=lb;A[M].b:=la;
    end;
Kp(1,M);T:=A;
fat[N+1]:=M+1;
    inc(M);
    for i:=N downto 1 do
```

```
        begin
        while (M>1)and(T[M-1].a=i) do dec(M);
        fat[i]:=M;
        end;
    end;
procedure LCA(X:longint);
var i:longint;
    begin
    fa[x]:=0;
    for i:=fat[x] to fat[x+1]-1 do if Lavel[T[i].b]>Lavel[x]+1 then
        begin
        Lavel[T[i].b]:=Lavel[x]+1;
        LCA(T[i].b);
        fa[T[i].b]:=x;
        end;
    for i:=Star[x] to Star[x+1]-1 do
    if check[A[i].b] then
        begin
        ans:=ans+Lavel[A[i].a]+Lavel[A[i].b]-2*Lavel[find(a[i].b)];
        end;
    check[x]:=True;
    end;
var i,la,lb:longint;
begin
fillchar(T,sizeof(T),0);
initTree;
initAsk;
fillchar(check,sizeof(check),False);
filldword(Lavel,sizeof(Lavel)shr 2,maxlongint);
ans:=0;
lavel[N shr 1]:=0;
LCA(N shr 1);
writeln(ans);
readln;readln;
end.
```

八．最优子矩形问题：

例题：

2、Candy

糖果盒（ Candy Box ）

问题描述：

一个被分为 n*m 个格子的糖果盒，第 i 行第 j 列位置的格子里面有 a[i][j] 颗

糖。本来 tenshi 打算送这盒糖果给某 PPMM 的，但是就在要送出糖果盒的前一天晚上，一只极其可恶的老鼠夜袭糖果盒，有部分格子被洗劫并且穿了洞。tenshi 必须尽快从这个糖果盒里面切割出一个矩形糖果盒，新的糖果盒不能有洞，并且 tenshi 希望保留在新糖果盒内的糖的总数尽量多。

任 务 ：

请帮 tenshi 设计一个程序 计算一下新糖果盒最多能够保留多少糖果。

输入格式：

从文件 CANDY.INP 读入数据。第一行有两个整数 n、m。第 i+1 行的第 j 个数表示 a[i][j]，如果这个数为 0 ，则表示这个位置的格子被洗劫过。其中：

1 ≤ n，m ≤ 1000

0 ≤ a[i][j]≤ 255

注意：本题提供 16 MB 内存，时间限制为 2 秒。

输出格式：

输出最大糖果数到 CANDY.OUT。

样例

CANDY.INP	CANDY.OUT
3 4 1 2 3 4 5 0 6 3 10 3 4 0	17

注：

10 3 4

这个矩形的糖果数最大

题解：

```
program candy;
const
  maxn=1000;
var
  left,right,high:array[1..maxn] of longint;
  s:array[0..maxn,0..maxn] of longint;
  now,res,leftmost,rightmost,i,j,k,n,m:longint;
  f:text;
begin
```

```
  assign(f,'candy.in');
  reset(f);
  readln(f,n,m);
  fillchar(s,sizeof(s),0);
  for i:=1 to m do
    begin
      left[i]:=1; right[i]:=m; high[i]:=0;
    end;
  res:=0;
  for i:=1 to n do
    begin
      k:=0; leftmost:=1;
      for j:=1 to m do
        begin
          read(f,now); k:=k+now;
          s[i,j]:=s[i-1,j]+k;
          if now=0 then
            begin
              high[j]:=0; left[j]:=1; right[j]:=m;
              leftmost:=j+1;
            end
          else
            begin
              high[j]:=high[j]+1;
              if leftmost>left[j] then left[j]:=leftmost;
            end;
        end;
      rightmost:=m;
      for j:=m downto 1 do
        begin
          if high[j]=0 then
            begin
              rightmost:=j-1;
            end
          else
            begin
              if right[j]>rightmost then right[j]:=rightmost;

now:=s[i,right[j]]+s[i-high[j],left[j]-1]-s[i-high[j],right[j]]-s[i,left[j]-1];
              if now>res then res:=now;
            end;
        end;
    end;
  writeln(res);
```

end.

十．二分图匹配：

给定一个二分图 G，M 为 G 边集的一个子集，如果 M 满足当中的任意两条边都不依附于同一个顶点，则称 M 是一个匹配。

极大匹配(Maximal Matching)是指在当前已完成的匹配下,无法再通过增加未完成匹配的边的方式来增加匹配的边数。最大匹配(maximum matching)是所有极大匹配当中边数最大的一个匹配。选择这样的边数最大的子集称为图的最大匹配问题。

如果一个匹配中，图中的每个顶点都和图中某条边相关联，则称此匹配为完全匹配，也称作完备匹配。

求最大匹配的一种显而易见的算法是：先找出全部匹配，然后保留匹配数最多的。但是这个算法的时间复杂度为边数的指数级函数。因此，需要寻求一种更加高效的算法。

增广路的定义(也称增广轨或交错轨)：

若 P 是图 G 中一条连通两个未匹配顶点的路径，并且属 M 的边和不属 M 的边(即已匹配和待匹配的边)在 P 上交替出现，则称 P 为相对于 M 的一条增广路径。

由增广路的定义可以推出下述三个结论：

1—P 的路径长度必定为奇数，第一条边和最后一条边都不属于 M。

2—P 经过取反操作可以得到一个更大的匹配 M’。

3—M 为 G 的最大匹配当且仅当不存在相对于 M 的增广路径。

用增广路求最大匹配(称作匈牙利算法，匈牙利数学家 Edmonds 于 1965 年提出)

算法轮廓：

(1)置 M 为空

(2)找出一条增广路径 P，通过取反操作获得更大的匹配 M’代替 M

(3)重复(2)操作直到找不出增广路径为止

时间复杂度 邻接矩阵：最坏为 O(n^3) 邻接表：O(nm)

空间复杂度 O(n^2) O(m+n)

Pascal:

```
Program matching;
Const
max = 1000;
Var
map : array [1..max, 1..max] of boolean; {邻接矩阵}
match: array [1..max] of integer; {记录当前连接方式}
chk : array [1..max] of boolean; {记录是否遍历过，防止死循环}
m, n, i, t1, t2, ans: integer;
Function dfs(p: integer): boolean;
var
i, t: integer;
```

```
begin
for i:=1 to n do
if map[p, i] and not chk[ i ] then
begin
chk[ i ] := true;
if (match[ i ] = 0) or dfs(match[ i ]) then {没有被连过 或寻找到增广路}
begin
match[ i ] := p;
exit(true);
end;
end;
exit(false);
end;
begin
readln(n, m); {N 为二分图左侧点数 M 为可连接的边总数}
fillchar(map, sizeof(map), 0);
for i:=1 to m do
begin
readln(t1, t2);
map[t1, t2] := true;
end;
fillchar(match, sizeof(match), 0);
ans := 0;
for i:=1 to n do
begin
fillchar(chk, sizeof(chk), 0);
if dfs(i) then inc(ans);
end;
writeln(ans);
for i:=1 to 1000 do
if match[ i ] <> 0 then
writeln(match[ i ], '-->', i);
end.
```

树

一． 并查集：

```
Var a: array[0..5000] Of string;
   fa: array[0..5000] Of integer;
   i,k,n,m,p,k1,k2: integer;
Function getfa(x:integer): integer;
```

```
Begin
   If fa[x]=0 Then getfa := x
   Else
      Begin
         fa[x] := getfa(fa[x]);
         getfa := fa[x]
      End;
End;
Procedure union(x,y:integer);
Var i,j: integer;
Begin
   i := getfa(x);
   j := getfa(y);
   If i<>j Then fa[i] := j;
End;
Begin
   readln(n,m,p);
   fillchar(fa,sizeof(fa),0);
   For i:=1 To m Do
      Begin
         readln(k1,k2);
         union(k1,k2);
      End;
   For i:=1 To p Do
      Begin
         readln(k1,k2);
         If getfa(k1)<>getfa(k2) Then writeln('No')
         Else    writeln('Yes');
      End;
End.
```

二． 哈夫曼树（最优二叉树）：

将树结构用于实际，常常要考虑一个问题，即如何设计一棵二叉树，使得执行路径最短，即算法的效率最高。哈夫曼树(Huffman Tree)，又称最优树，是一类带权路径长度最短的树，有着广泛的应用，在这里我们将要讨论哈夫曼二叉树(最优二叉村)的概念及其算法。

例如：现有一批球磨机上的铁球，需要将它分成四类：直径不大于 20 的属于第一类;直径大于 20 而不大于 50 的属于第二类;直径大于 50 而不大于 100 的属于第三类;其余的属于第四类;假定这批球中属于第一、二、三、四类铁球的个数之比例是 1:2:3:4。

我们可以把这个判断过程表示为下图中的两种方法：

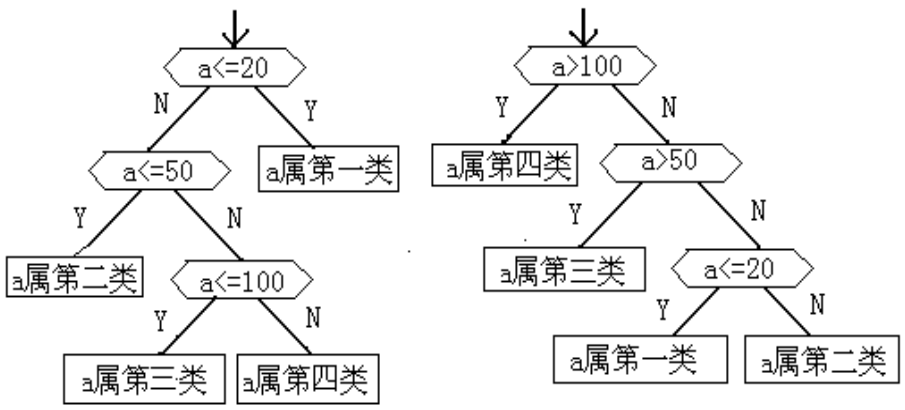


图 1　　图 2

那么究竟将这个判断过程表示成哪一个判断框，才能使其执行时间最短呢？让我们对上述判断框做一具体的分析。

假设有 1000 个铁球，则各类铁球的个数分别为：100、200、300、400；

对于图 1 和 图 2 比较的次数分别如下表所示：

图 1		
序号	比较式	比较次数
1	a<=20	1000
2	a<=50	900
3	a<=100	700
合计		2600

图 2		
序号	比较式	比较次数
1	a>100	1000
2	a>50	600
3	a<=20	300
合计		1900

通过上述分析可知，图 2 所示的判断框的比较次数远远小于图 1 所示的判断框的比较次数。为了找出比较次数最少的判断框，将涉及到树的路径长度问题。

一、哈夫曼树的基本术语

1）. **树的路径和路径长度**

若在一棵树中存在着结点序列 $k_1, k_2, \ldots, k_j$，使得 k_i 是 k_{i+1} 的双亲$(1<=i<j)$，则称此结点序列是从 k_1 到 k_j 的**路径**，因树中每个结点只有一个双亲结点，所以它也是这两个结点之间的唯一路径。从 k_1 到 k_j 所经过的分支数称为这两点之间的**路径长度**，它等于路径上的结点数减 1。（即从树中一个结点到另一个结点之间的分支构成这两个结点之间的路径，路径上的分支数目称做路径长度，树的路径长度是从树根到每一个结点的路径长度之和）

2）. **结点的权和带权路径长度**

在许多应用中，常常将树中的结点赋上一个有着某种意义的实数，我们称此实数为该结点的**权**。**结点的带权路径长度**规定为从根结点到该结点之间的路径长度与该结点上权的乘积。

3）. **树的带权路径长度**

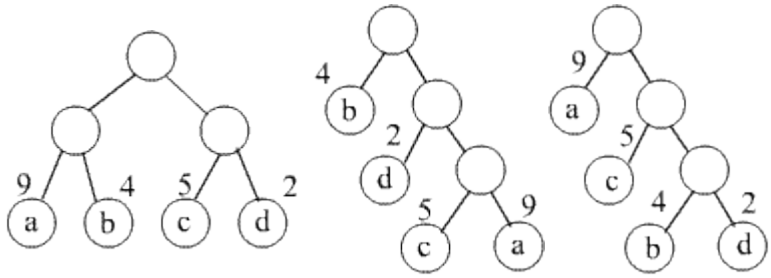
树的带权路径长度定义为树中所有叶子结点的带权路径长度之和，通常记为:

$$WPL=\sum_{i=1}^{n} w_i l_i$$

其中 n 表示叶子结点的数目，w_i 和 l_i 分别表示叶子结点 k_i 的权值和根到 k_i 之间的路径长度。

二、哈夫曼树

哈夫曼(Huffman)树又称最优二叉树。它是 n 个带权叶子结点构成的二叉树中，带权路径长度 WPL 最小的二叉树。因为构造这种树的算法是最早由哈夫曼于 1952 年提出的，所以被称之为哈夫曼树。 例如，由四个叶子结点 a,b,c,d,分别带权为 9、4、5、2，由它们构成的三棵不同的二叉树如下图 3 所示。



(a) (b) (c)

图 3 由四个叶子结点构成的三棵不同的带权二叉树

每一棵二叉树的带权路径长度 WPL 分别为：

(a)WPL=9×2+4×2+5×2+2×2=40

(b)WPL=4×1+2×2+5×3+9×3=50

(c)WPL=9×1+5×2+4×3+2×3=37

其中(c)树的 WPL 最小，此树就是哈夫曼树。

由上面可以看出，由 n 个带权叶子结点所构成的二叉树中，满二叉树或完全二叉树不一定就是最优二叉树，权值越大的结点离根越近的二叉树才是最优二叉树。

在给定一组具有确定权值的叶结点，可以构造出不同的带权二叉树。例如，给出 4 个叶结点，设其权值分别为 1，3，5，7，我们可以构造出形状不同的多个二叉树。这些形状不同的二叉树的带权路径长度将各不相同。图 4 给出了其中 5 个不同形状的二叉树。

这五棵树的带权路径长度分别为：

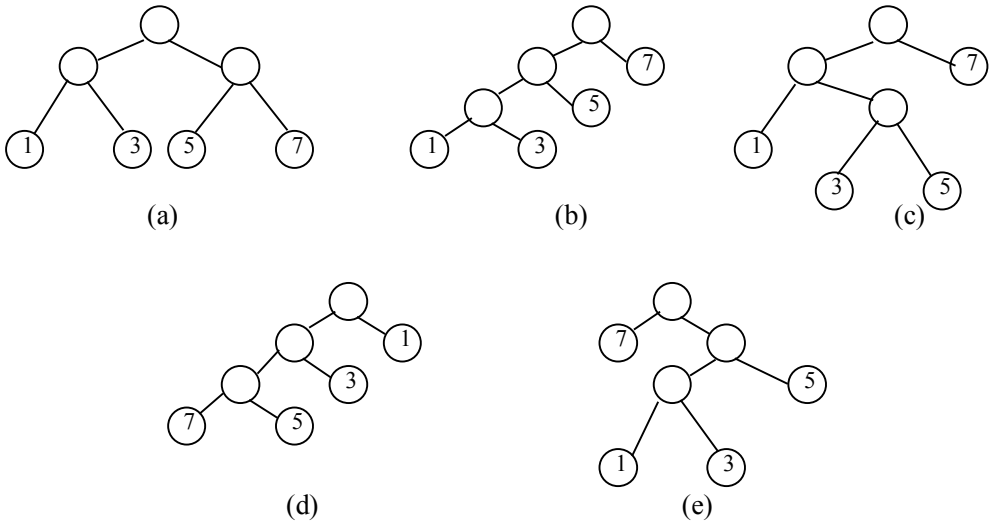
（a）WPL＝1×2＋3×2＋5×2＋7×2＝32

（b）WPL＝1×3＋3×3＋5×2＋7×1＝29

（c）WPL＝1×2＋3×3＋5×3＋7×1＝33

（d）WPL＝7×3＋5×3＋3×2＋1×1＝43

（e）WPL＝7×1＋5×2＋3×3＋1×3＝29



(a) (b) (c) (d) (e)

图 4　具有相同叶子结点和不同带权路径长度的二叉树

由此可见，由相同权值的一组叶子结点所构成的二叉树有不同的形态和不同的带权路径长度，那么如何找到带权路径长度最小的二叉树（即哈夫曼树）呢？根据哈夫曼树的定义，一棵二叉树要使其 WPL 值最小，必须使权值越大的叶结点越靠近根结点，而权值越小的叶结点越远离根结点。哈夫曼树的最主要的特点是带权的二叉树的路径长度最小的是权大的叶

子离根最近的二叉树，因此哈夫曼树又称为最优叶子二叉树。

注意：

① 叶子上的权值均相同时，完全二叉树一定是最优二叉树，否则完全二叉树不一定是最优二叉树。

② 最优二叉树中，权越大的叶子离根越近。

③最优二叉树的形态不唯一。

如何构造哈夫曼二叉树呢？D.A.Huffman 给出了一个简单而又漂亮的算法,这个算法称为哈夫曼算法．它的基本思想就是让权大的叶子离根最近，具体做法是：

(1)根据给定的 n 个权值同{W1,W2…，Wn}．构造 n 棵二叉树的集合 F{T1,T2,….Tn}，其中每棵二叉树中均只含一个带权值为 Wi 的根结点，其左、右子树为空树；

(2)在 F 中选取其根结点的权值为最小的两棵二叉树，分别作为左、右子树构造一棵新的二叉树，并置这棵新的二叉树根结点的权值为其左，右于树根结点的权值之和；

(3)从 F 中删去这两棵树，同时加入刚生成的新树；

(4)重复(2)和(3)两步，直到 F 中只含一棵树为止，

从上述算法中可以看出，F 实际上是森林，算法的目的是不断地对森林中的二叉树进行“合并”，最终得到哈夫曼二叉树。

图 5 给出了前面提到的叶结点权值集合为 W＝{1，3，5，7}的哈夫曼树的构造过程。可以计算出其带权路径长度为 29，由此可见，对于同一组给定叶结点所构造的哈夫曼树，树的形状可能不同，但带权路径长度值是相同的，一定是最小的。

程序如下：

```
type
     jl=record
        l,r,st,pre:integer;
        end;

var
   n,i:integer;
   a:array[0..1000] of jl;

procedure hfm;
 var
      i,j,min1,min2:integer;
 begin
     for i:=n+1 to 2*n-1 do
          begin
            a[0].st:=maxint;
            min1:=0;min2:=0;
            for j:=1 to i-1 do
                  if (a[j].st<a[min1].st) and (a[j].pre=0) then min1:=j;
            for j:=1 to i-1 do
                  if (a[j].st<a[min2].st) and (a[j].pre=0) and (j<>min1) then min2:=j;
            a[i].st:=a[min1].st+a[min2].st;
```

```
                a[min1].pre:=i;
                a[min2].pre:=i;
                a[i].r:=min2;
                a[i].l:=min1;
            end;
  end;

procedure print(t:integer);
  var
      pr:string;
      p:integer;
  begin
      pr:='';
      p:=t;
      while    a[p].pre<>0 do
            begin
                if a[a[p].pre].l=p then pr:='0'+pr else pr:='1'+pr;
                p:=a[p].pre;
            end;
      writeln(a[t].st,' ',pr);
  end;

begin
  assign(input,'hfm.in');
  reset(input);

      readln(n);
      for i:=1 to n do
          read(a[i].st);
      hfm;
      for i:=1 to n do
         print(i);

  close(input);
end.
```

四．二叉树的性质：

性质 1 二叉树第 i 层上的结点数目最多为 2^{i-1}(i≥1)。

证明：用数学归纳法证明：

归纳基础：i=1 时，有 $2^{i-1}=2^0=1$。因为第 1 层上只有一个根结点，所以命题成立。

归纳假设：假设对所有的 j(1≤j<i)命题成立，即第 j 层上至多有 2^{j-1} 个结点，证明 j=i 时命题亦成立。

归纳步骤：根据归纳假设，第 i-1 层上至多有 2^{i-2} 个结点。由于二叉树的每个

结点至多有两个孩子，故第 i 层上的结点数至多是第 i-1 层上的最大结点数的 2 倍。即 j=i 时，该层上至多有 $2\times 2^{i-2}=2^{i-1}$ 个结点，故命题成立。

性质 2 深度为 k 的二叉树至多有 2^k-1 个结点(k≥1)。

证明：在具有相同深度的二叉树中，仅当每一层都含有最大结点数时，其树中结点数最多。因此利用性质 1 可得，深度为 k 的二叉树的结点数至多为：

$$2^0+2^1+\cdots+2^{k-1}=2^k-1$$

故命题正确。

性质 3 在任意一棵二叉树中，若终端结点的个数为 n_0，度为 2 的结点数为 n_2，则 $n_0=n_2+1$。

证明：因为二叉树中所有结点的度数均不大于 2，所以结点总数(记为 n)应等于 0 度结点数、1 度结点(记为 n_1)和 2 度结点数之和：

$$n=n_0+n_1+n_2 \quad (式子\ 1)$$

另一方面，1 度结点有一个孩子，2 度结点有两个孩子，故二叉树中孩子结点总数是：

$$n_1+2n_2$$

树中只有根结点不是任何结点的孩子，故二叉树中的结点总数又可表示为：

$$n=n_1+2n_2+1 \quad (式子\ 2)$$

由式子 1 和式子 2 得到：

$$n_0=n_2+1$$

满二叉树和完全二叉树是二叉树的两种特殊情形。

1、满二叉树(FullBinaryTree)

一棵深度为 k 且有 2^k-1 个结点的二叉树称为满二叉树。

满二叉树的特点：

（1） 每一层上的结点数都达到最大值。即对给定的高度，它是具有最多结点数的二叉树。

（2） 满二叉树中不存在度数为 1 的结点，每个分支结点均有两棵高度相同的子树，且树叶都在最下一层上。

【例】图(a)是一个深度为 4 的满二叉树。

2、完全二叉树(Complete BinaryTree)

若一棵二叉树至多只有最下面的两层上结点的度数可以小于 2，并且最下一层上的结点都集中在该层最左边的若干位置上，则此二叉树称为完全二叉树。

特点：

（1） 满二叉树是完全二叉树，完全二叉树不一定是满二叉树。

（2） 在满二叉树的最下一层上，从最右边开始连续删去若干结点后得到的二叉树仍然是一棵完全二叉树。

（3） 在完全二叉树中，若某个结点没有左孩子，则它一定没有右孩子，即该结点必是叶结点。

性质 4 具有 n 个结点的完全二叉树的深度为

$$\lfloor \lg n \rfloor+1 \text{(或} \lceil \lg(n+1) \rceil \text{)}$$

证明：设所求完全二叉树的深度为 k。由完全二叉树定义可得：

深度为 k 得完全二叉树的前 k-1 层是深度为 k-1 的满二叉树，一共有 $2^{k-1}-1$ 个结点。

由于完全二叉树深度为 k，故第 k 层上还有若干个结点，因此该完全二叉树的结点个数：

$$n>2^{k-1}-1。$$

另一方面，由性质 2 可得：

$$n\leqslant 2^{k}-1，$$

即：$2^{k-1}-1<n\leqslant 2^{k}-1$

由此可推出：$2^{k-1}\leqslant n<2^{k}$，取对数后有：

$$k-1\leqslant \lg n<k$$

又因 k-1 和 k 是相邻的两个整数，故有

$$k-1=\lfloor \lg n \rfloor,$$

由此即得：k=lg(n)+1

搜索

a) 基本深搜框架

```
procedure deep（k）;
begin
  for I:=1 to n do{n 个方向}
   begin
    if 符合条件
     then 入栈;
    if 是目标结点
     then 输出
    else deep（k+1）;
    栈顶结点出栈;
  end;
end;
```

b) 深搜的优化：剪枝、角度变换、展望等

例：九宫数独

From zhanghui

搜索-九宫数独

描述 Description

迷倒众生的「九宫数独」有人一开始玩就停不住了...「数独」看来很单纯，九直行九

横列，共分成九个九宫格，每行列都由 1 到 9 、不能重复的阿拉伯数字组成，每个九宫格亦然。如下图，在该图中填入空缺的数字可以组成九宫数独一种方案。

「数独」是数字解谜游戏。要用到算术吗？不必！除了 1 到 9 的阿拉伯数字以外，不必用到任何东西，要解数独，会用到的是推理与逻辑。先不论「数独」受欢迎的程度令它迅速「全球化」,「数独」本身的故事也蛮令人回味的。两百年前盲眼的瑞士数学家欧拉发明数独，但原始版本太容易，多年前日本人改进美国某版本的数独，创造出繁复的益智游戏。然后纽西兰裔英国香港退休法官高乐德近年把数独引进欧美而完成了「环球之旅」。这给了我们什么启示？也许是：这世界上，残障不影响发明、退休不影响发展；流行不代表创新，说不定是老东西，改进才能发扬光大；领航者不一定成功、后继者也有机会...你能否编程序解决九宫数独问题？
输入格式：共九行，每行 9 个 0~9 的数字，0 代表待填的数字
输出格式：输出解决方案，每个方案之后空一行，最后一行输出所有方案数。
输入样例：
081000000
020087000
700050006
900015400
000000030
500030000
003000028
004000700
000600005
输出样例：
381946257
625187394
749352816
936715482
478269531
512438679
163574928
254891763
897623145

1

```
var
  bh:array[1..9]of longint;
  bz:array[1..9]of longint;
  xf:array[1..9]of longint;
  g,m:array[1..9,1..9]of integer;
  i,j,tol,un:integer;
  ch:char;
```

```
function which(a,b:integer):integer;
   begin
   if (b<=3) and (a<=3) then exit(1);
   if (b<=6) and (a<=3) then exit(2);
   if (b<=9) and (a<=3) then exit(3);
   if (b<=3) and (a<=6) then exit(4);
   if (b<=6) and (a<=6) then exit(5);
   if (b<=9) and (a<=6) then exit(6);
   if (b<=3) and (a<=9) then exit(7);
   if (b<=6) and (a<=9) then exit(8);
   if (b<=9) and (a<=9) then exit(9);
   end;

procedure print;
   var i,j:integer;
   begin
   inc(tol);
   for i:=1 to 9 do begin
      for j:=1 to 9 do write(m[i,j]);
      writeln;
   end;
     writeln;
   end;

procedure try1(a,b:integer);
   var i,pos,p,d:integer;
   begin
   if b=10 then begin try1(a+1,1); exit; end;
   if a>9 then print else
         if g[a,b]<>0 then begin
              m[a,b]:=g[a,b];
              try1(a,b+1);end else
                begin
                  pos:=un and not (bh[a] or bz[b] or xf[which(a,b)]);
                  while pos<>0 do
                    begin
                      p:=pos and -pos;
                      pos:=pos-p;
                      d:=round(ln(p)/ln(2))+1;
                      inc(xf[which(a,b)],p);
                      inc(bh[a],p);
                      inc(bz[b],p);
                      m[a,b]:=d;
                      if b<10 then try1(a,b+1) else try1(a+1,1);
```

```
                dec(xf[which(a,b)],p);
                dec(bh[a],p);
                dec(bz[b],p);
                m[a,b]:=0;
              end;
            end;
  end;

begin
  fillchar(bh,sizeof(bh),0);
  fillchar(bz,sizeof(bz),0);
  fillchar(xf,sizeof(xf),0);
  fillchar(g,sizeof(g),0);
  un:=1 shl 9-1;
    for i:=1 to 9 do
    begin
      for j:=1 to 9 do begin
          read(ch);
          g[i][j]:=ord(ch)-ord('0');
          if g[i,j]<>0 then begin
          inc(xf[which(i,j)],1 shl (g[i,j]-1));
          inc(bh[i],1 shl (g[i,j]-1));
          inc(bz[j],1 shl (g[i,j]-1));
          end;
          end;
          readln;
      end;
  tol:=0;
  try1(1,1);
  writeln(tol);
end.
```

基本广搜框架

```
          pocedure bf;
          begin
            h:=0；t:=1；
            repeat
             inc（h）;
             for I：=1 to n do
              begin
                if 符合条件 and 不重复
                 then inc（t）；存放新结点；
                if 是目标结点
                 then 输出；
```

```
      end;
    until h>=t;
  end;
```

c) 广搜的优化：用布尔数组、哈希表、二叉排序树等提高判重效率，用双向搜索、滚动数组改善空间效率，用二进制改进产生式、存储空间以及判重效率

例题：

【问题描述】：

在 3 * 3 的棋盘上，摆 有八个棋子，每个棋子上标有 1 至 8 的某一数字。棋盘中留有一个空格。空格周围的棋子可以移到空格中。要求解的 问题是，给出一种初始布局 [初始状态] 和目标布局 [目标状态] ，输出从初始布局到目标布局的转变至少需要的步数。

【输入格式】

输入由两行组成，每行 8 个数，分别表示初始状态和目标状态：

【输出格式】

输出步数，若无解输出“No solution!”。

【输入输出样例】

输入文件名： 8num.in

283164705

123804765

输出文件名：8num.out

5

时限：前两组数据不超过 1 秒，第三组数据不超过 10 秒

```
program shuangguang8; {Made By   P.D.B (AYLA)}
{此程序可全部 AC,用时 0 ms}
const ji:array[1..8]of longint=(40320,5040,720,120,24,6,2,1);
        {康托展开时用到的常数，分别为为 9-1 的阶乘}
 var d1,d2:array[0..10000]of string[10];{两个队，分别存放双向广搜的一支}
      k1,k2:array[0..370000]of boolean;{Hash 表，康托展开的数组，判重用}
      kb1,kb2:array[0..370000]of integer;{Hash 表，康托展开的数组，存放每种情况的步数}
      fu1,fu2:array[0..10000]of integer;{两个队，存放当前步数}
      w1,w2,h1,t1,h2,t2,i,j,tol,lei,b:longint;m,n,z:string[10];l,c:char;{辅助变量}
procedure print;{输出}
begin
 writeln(kb1[lei]+kb2[lei]);{从起始到当前情况与从目标到当前情况的步数和}
 halt;
end;
function PDhash1(x:string):boolean;{判断当前情况在队列 1 中是否重复}
begin
 lei:=0;
 for i:=8 downto 1 do   {康托展开，i 表示当前位数}
  begin
   tol:=0;
```

```
    for j:=i to 9 do {将当前位数前比当前位数大的数的个数与当前位数的阶乘相乘，并累加到 l 变量 lei 中}
      if x[i]>x[j] then inc(tol);
    lei:=lei+ji[i]*tol;
   end;
 PDhash1:=k1[lei];
end;
function PDhash2(x:string):boolean;{判断当前情况在队列 2 中是否重复}
begin
 lei:=0;
 for i:=8 downto 1 do
   begin
    tol:=0;
    for j:=i to 9 do
     if x[i]>x[j] then inc(tol);
    lei:=lei+ji[i]*tol;
   end;
 PDhash2:=k2[lei];
end;
procedure hash1(x:string[10]);
{将当前情况加入到队列 1 的 Hash 表中，可与 PDhash1 过程合并，为便于理解将其分开}
begin
 lei:=0;
 for i:=8 downto 1 do
   begin
    tol:=0;
    for j:=i to 9 do
     if x[i]>x[j] then inc(tol);
    lei:=lei+ji[i]*tol;
   end;
 k1[lei]:=true;
 kb1[lei]:=fu1[t1]; {将当前步数存入康托展开后的数组中}
 if k2[lei] then print; {如果反向搜索状态中由此情况，则输出，并结束程序}
end;
procedure hash2(x:string[10]);
{将当前情况加入到队列 2 的 Hash 表中，可与 PDhash2 过程合并，为便于理解将其分开}
begin
 lei:=0;
 for i:=8 downto 1 do
   begin
    tol:=0;
    for j:=i to 9 do
```

```
      if x[i]>x[j] then inc(tol);
    lei:=lei+ji[i]*tol;
   end;
  k2[lei]:=true;
  kb2[lei]:=fu2[t2]; {将当前步数存入康托展开后的数组中}
  if k1[lei] then print; {如果正向搜索状态中由此情况，则输出，并结束程序}
end;
procedure init;{读入}
begin
  fillchar(k1,sizeof(k1),false);
  fillchar(k2,sizeof(k2),false);
  {for i:=1 to 3 do
   begin
    for j:=1 to 3 do
     begin
      read(c); m:=m+c; read(c);
     end;
    readln;
   end;}
  m:='123456780';
  for i:=1 to 3 do
   begin
    for j:=1 to 3 do
     begin
      read(c); n:=n+c; read(c);
     end;
    readln;
   end;
end;
procedure WFS;{双向广搜}
begin
  t1:=1;t2:=1;h1:=0;h2:=0;
  repeat
   inc(h1);   {正向搜索}
   w1:=pos('0',d1[h1]);
   if w1-3>0 then {查找'0'的位置，判断'0'可移动的方向}
    begin
     z:=d1[h1];l:=z[w1];z[w1]:=z[w1-3];z[w1-3]:=l;{移动'0'}
     if not PDhash1(z) then {判断是否重复}
     begin
      inc(t1); d1[t1]:=z;
      fu1[t1]:=fu1[h1]+1; {当前情况步数等于其父节点的步数加 1}
      hash1(z); {加入 Hash 表}
     end;
```

```
  end;
if w1+3<10 then
  begin
    z:=d1[h1];l:=z[w1];z[w1]:=z[w1+3];z[w1+3]:=l;
    if not PDhash1(z) then
    begin
      inc(t1); d1[t1]:=z;
      fu1[t1]:=fu1[h1]+1;
      hash1(z);
    end;
  end;
case w1 mod 3 of {判断'0'可移动的方向}
  0: begin
        z:=d1[h1];l:=z[w1];z[w1]:=z[w1-1];z[w1-1]:=l;
        if not PDhash1(z) then
        begin
          inc(t1); d1[t1]:=z;
          fu1[t1]:=fu1[h1]+1;
          hash1(z);
        end;
      end;
  1: begin
        z:=d1[h1];l:=z[w1];z[w1]:=z[w1+1];z[w1+1]:=l;
        if not PDhash1(z) then
        begin
          inc(t1); d1[t1]:=z;
          fu1[t1]:=fu1[h1]+1;
          hash1(z);
        end;
      end;
  2:begin
        z:=d1[h1];l:=z[w1];z[w1]:=z[w1-1];z[w1-1]:=l;
        if not PDhash1(z) then
        begin
        inc(t1); d1[t1]:=z;
        fu1[t1]:=fu1[h1]+1;
        hash1(z);
        end;
        z:=d1[h1];l:=z[w1];z[w1]:=z[w1+1];z[w1+1]:=l;
        if not PDhash1(z) then
        begin
        inc(t1);d1[t1]:=z;
        fu1[t1]:=fu1[h1]+1;
        hash1(z);
```

```
        end;
       end;
    end;

inc(h2);{反向搜索，过程与正向搜索基本相同}
w1:=pos('0',d2[h2]);
if w1-3>0 then
  begin
    z:=d2[h2];l:=z[w1];z[w1]:=z[w1-3];z[w1-3]:=l;
    if not PDhash2(z) then
    begin
    inc(t2);d2[t2]:=z;
    fu2[t2]:=fu2[h2]+1;
    hash2(z);
  end;
  end;
if w1+3<10 then
  begin
    z:=d2[h2];l:=z[w1];z[w1]:=z[w1+3];z[w1+3]:=l;
    if not PDhash2(z) then
    begin
    inc(t2);d2[t2]:=z;
    fu2[t2]:=fu2[h2]+1;
    hash2(z);
    end;
  end;
case w1 mod 3 of
  0: begin
        z:=d2[h2];l:=z[w1];z[w1]:=z[w1-1];z[w1-1]:=l;
        if not PDhash2(z) then
        begin
        inc(t2);d2[t2]:=z;
        fu2[t2]:=fu2[h2]+1;
        hash2(z);
        end;
       end;
  1: begin
        z:=d2[h2];l:=z[w1];z[w1]:=z[w1+1];z[w1+1]:=l;
        if not PDhash2(z) then
        begin
        inc(t2); d2[t2]:=z;
        fu2[t2]:=fu2[h2]+1;
        hash2(z);
        end;
```

```
            end;
          2:begin
              z:=d2[h2];l:=z[w1];z[w1]:=z[w1-1];z[w1-1]:=l;
              if not PDhash2(z) then
              begin
              inc(t2); d2[t2]:=z;
              fu2[t2]:=fu2[h2]+1;
              hash2(z);
              end;
              z:=d2[h2];l:=z[w1];z[w1]:=z[w1+1];z[w1+1]:=l;
              if not PDhash2(z) then
              begin
              inc(t2); d2[t2]:=z;
              fu2[t2]:=fu2[h2]+1;
              hash2(z);
              end;
            end;
          end;
        until (h2>=t2)or(h1>=t1);
    end;
    begin {主程序，十分简洁}
      init;
      d1[1]:=m; {初始化，队中存入起始状态}
      d2[1]:=n; {初始化，队中存入目标状态}
      hash1(m);
      hash2(n);
      WFS;
      writeln('Impossible'); {无解则输出 Impossible}
end.
```

分治

一． 二分查找：

```
    procedure search(lx,rx,n: longint);
var i,j,mid:longint;
    f:boolean;
begin
   i:=lx;j:=rx;
   f:=false;
   repeat
     mid:=(i+j)div 2;
```

```
    if a[mid]=n then begin f:=true;break; end
    else if a[mid]<n then i:=mid +1
        else j:=mid -1;
  until i>j;
  if f then writeln(mid,'  found')
  else writeln('No found');
end;
```

二．　快速幂：

```
var n,m,k,t,c,d:qword;
a:array[1..10000] of 0..1;
i,j:longint;
begin
readln(n,m,k); //m de n cifang
c:=0;
d:=1;
repeat
inc(j);
a[j]:=n mod 2;
n:=n div 2;
until (n=1)or(n=0);
inc(j);
a[j]:=n;
for i:=j downto 1 do
begin
c:=c*2;
d:=(d*d) mod k;
if a[i]=1 then begin
inc(c);
d:=(d*m) mod k;
end;
end;
writeln(d);
end.
```

数论

d) 求两数的最大公约数

欧几里德算法求 a，b 的最大公约数

```
function gcd(a,b:longint):longint;
begin
 if b=0 then gcd:=a
```

```
  else gcd:=gcd(b,a mod b);
end;
```

e) 求两数的最小公倍数

```
acm=a*b div gcd(a,b);
```

其他:

扩展的欧几里德算法，求出 gcd(a,b)和满足 gcd(a,b)=ax+by 的整数 x 和 y

```
program ojilide;
 var a,b,c,x,y:longint;
function exgcd(a,b:longint;var x,y:longint):longint;
 var t:longint;
begin
 if b=0 then
  begin
   exgcd:=a;
   x:=1; y:=0;
  end
 else
  begin
  exgcd:=exgcd(b,a mod b,x,y);
  t:=x;
  x:=y;
  y:=t-(a div b)*y;
 end;
end;
begin
 readln(a,b);
 c:=exgcd(a,b,x,y);
 writeln(c,' ',x,' ',y);
end.
```

三． 求素数：

```
var pr:array[0..10000] of longint;
i,j,n,m,l:longint;
procedure getprime;// to build a less in 50000 prime blank
var i,j:longint;
p:array[0..50000] of boolean;
begin
fillchar(p,sizeof(p),true);
p[1]:=false;
i:=2;
while i<=50000 do
begin
```

```
if p[i] then begin
j:=i*2;
while j<=50000 do
begin
p[j]:=false;
inc(j,i);
end;
end;
inc(i);
end;
l:=0;
for i:=1 to 50000 do
if p[i] then begin inc(l);pr[l]:=i;
end;
end;
function prime(x:longint):boolean; //To Judge A longint is a prime or not
var i:longint;
begin
prime:=false;
for i:=1 to l do
if pr[i]>=x then break
else if x mod pr[i]=0 then exit;
prime:=true;
end;
begin
getprime;
for i:=1 to l do
write(pr[i],' ');
writeln;
end.
```

三． 与 Mod 运算相关的加减乘除法

规律	公式
结合率	**((a+b) mod p + c)mod p = (a + (b+c) mod p) mod p** **((a*b) mod p * c)mod p = (a * (b*c) mod p) mod p**
交换率	**(a + b) mod p = (b+a) mod p** **(a × b) mod p = (b × a) mod p**
分配率	**((a +b)mod p × c) mod p = ((a × c) mod p + (b × c) mod p) mod p**

四． 进制转化：见绿书；

f) 五． 中位数的计算与应用

代权中位数 ：

D[I]—第 I 个点的权值

DIST（I，J）—I 到 J 点的距离，即 DIST（I，J）=|NUM[I]-NUM[J]|

由定义式易知：DIST（I，J）=DIST（J，I）

算法：若 T 是最优点，则必有其左边的权值和加上 D[T]后大于右边的权值和，其右边的权值和加上 D[T]后大于左边的权值和；

排序

一． 堆排：

见绿书；

二． 归并排序：

```
var a,b:array[0..100000] of longint;
i,j,k,n,m,t:longint;
procedure merge(l,r:longint);
var mid,t1,t2:longint;
  begin
  if l=r then exit;
mid:=(l+r)div 2;
merge(l,mid);
merge(mid+1,r);
t1:=l;
t2:=mid+1;
for i:=l to r do
begin
  if(t1<=mid)and((a[t1]<=a[t2])or(t2>r)) then
  begin
  b[i]:=a[t1];
  inc(t1);
  end
  else begin
  b[i]:=a[t2];
  inc(t2);
  end;
  end;
  for i:=l to r do
  a[i]:=b[i];
  end;
begin
readln(n);
```

```
for i:=1 to n do
readln(a[i]);
merge(1,n);
for i:=1 to n do
write(a[i],' ');
writeln;
end.
```

动规

一．背包：

g) 0/1 背包

基本方程： **f[i][v]=max(f[i-1][v],f[i-1][v-c[i]]+w[i])**

```
for i:=1 to n do
 for j:=m downto w[i] do    {这里不能写成 for j:=w[i] to m}
   if f[j-w[i]]+c[i]>f[j] then f[j]:=f[j-w[i]]+c[i];
    { f(j)表示重量不超过 j 公斤的最大价值}
```

h) 完全背包

基本方程：**f[i][v]=max(f[i-1][v-k*c[i]]+k*w[i]) (0<=k*c[i]<=v)**

统计方案个数：f[I,v]:=f[I,v-w[i]]+f[I-1.v];//前 I 个物品组成重量为 V 的方案总数

```
for j:=1 to m do
  for i:=1 to n do
    begin
      if j>=w[i] then if f[j-w[i]]+u[i]>f[j] then f[j]:=
f[j-w[i]]+u[i] ;
    end;
```

i) 多维背包

基本方程： f[i][v]=max(f[i-1][v-k*c[i]]+k*w[i]) (0<=k<=n[i])

复杂度是 O(V*Σn[i])。

例题：

USACO3.3.2 商店购物

描述 Description

在商店中，每一种商品都有一个价格（用整数表示）。例如,一朵花的价格是 2 zorkmids （z），而一个花瓶的价格是 5z 。为了吸引更多的顾客，商店举行了促销活动。

促销活动把一个或多个商品组合起来降价销售，例如：

三朵花的价格是 5z 而不是 6z，

两个花瓶和一朵花的价格是 10z 而不是 12z。

编写一个程序，计算顾客购买一定商品的花费，尽量利用优惠使花费最少。尽管有时候添加其他商品可以获得更少的花费，但是你不能这么做。

对于上面的商品信息，购买三朵花和两个花瓶的最少花费是：以优惠价购买两个花瓶和一朵花（10z），以原价购买两朵花（4z）。

PROGRAM NAME: shopping

INPUT FORMAT

输入文件包括一些商店提供的优惠信息，接着是购物清单。

第一行 优惠商品的种类数（0 <= s <= 99）。

第二行..第 s+1 行 每一行都用几个整数来表示一种优惠方式。第一个整数 n （1 <= n <= 5），表示这种优惠方式由 n 种商品组成。后面 n 对整数 c 和 k 表示 k （1 <= k <= 5）个编号为 c （1 <= c <= 999）的商品共同构成这种优惠，最后的整数 p 表示这种优惠的优惠价（1 <= p <= 9999）。优惠价总是比原价低。

第 s+2 行 这一行有一个整数 b （0 <= b <= 5），表示需要购买 b 种不同的商品。

第 s+3 行..第 s+b+2 行 这 b 行中的每一行包括三个整数：c ，k ，和 p 。c 表示唯一的商品编号（1 <= c <= 999），k 表示需要购买的 c 商品的数量（1 <= k <= 5）。p 表示 c 商品的原价（1 <= p <= 999）。最多购买 5*5=25 个商品。

SAMPLE INPUT (file shopping.in)

```
2
1 7 3 5
2 7 1 8 2 10
2
7 3 2
8 2 5
```

OUTPUT FORMAT

只有一行，输出一个整数：购买这些物品的最低价格。

SAMPLE OUTPUT (file shopping.out)

```
14
```

程序：

```
var f:array[-5 ..5,-5..5,-5..5,-5..5,-5..5,-5..5] of longint;
   g:array[0..105,0..1000] of longint;
      num:array[1..5,1..2] of integer;
      m,i,j,a,b,c,d,e,s:longint;
function min(x,y:longint):longint;
 begin
  if x<y then exit(x) else exit(y);
 end;
begin
 fillchar(g,sizeof(g),0);
 fillchar(f,sizeof(f),$4f);
 readln(s);
 for i:=1 to s do
```

```
  begin
    read(a);
    for j:=1 to a do
      begin
        read(b,c);
        g[i,b]:=c;
      end;
    readln(g[i,1000]);
  end;
readln(m);
for i:=s+1 to s+m do
  begin
    read(num[i-s,1],num[i-s,2]);
    g[i,num[i-s,1]]:=1;
    readln(g[i,1000]);
  end;
f[0,0,0,0,0]:=0;
  {dp}
for a:=0 to num[1,2] do
  for b:=0 to num[2,2] do
    for c:=0 to num[3,2] do
      for d:=0 to num[4,2] do
        for e:=0 to num[5,2] do
          for i:=1 to s+m do
            f[a,b,c,d,e]:=min(f[a,b,c,d,e],
              f[a-g[i,num[1,1]],b-g[i,num[2,1]],c-g[i,num[3,1]],d-g[i,num[4,1]],e-g[i,num[5,1]]]+g[i,1000]);
writeln(f[num[1,2],num[2,2],num[3,2],num[4,2],num[5,2]]);
end.
```

j) 多重背包及优化（用二进制优化）(详见背包九讲)

既然 01 背包问题是最基本的背包问题，那么我们可以考虑把完全背包问题转化为 01 背包问题来解。最简单的想法是，考虑到第 i 种物品最多选 V/c[i]件，于是可以把第 i 种物品转化为 V/c[i]件费用及价值均不变的物品，然后求解这个 01 背包问题。这样完全没有改进基本思路的时间复杂度，但这毕竟给了我们将完全背包问题转化为 01 背包问题的思路：将一种物品拆成多件物品。

更高效的转化方法是：把第 i 种物品拆成费用为 c[i]*2^k、价值为 w[i]*2^k 的若干件物品，其中 k 满足 c[i]*2^k<=V。这是二进制的思想，因为不管最优策略选几件第 i 种物品，总可以表示成若干个 2^k 件物品的和。这样把每种物品拆成 O(log(V/c[i]))件物品，是一个很大的改进。

多位背包时，将第 i 种物品分成若干件物品，其中每件物品有一个系数，这件物品的费用和价值均是原来的费用和价值乘以这个系数。使这些系数分别为 1,2,4,...,2^(k-1),n[i]-2^k+1，且 k 是满足 n[i]-2^k+1>0 的最大整数。例如，如果 n[i]为 13，就将这种物品分成系数分别为 1,2,4,6 的四件物品。

分成的这几件物品的系数和为 n[i]，表明不可能取多于 n[i]件的第 i 种物品。另外这种方法也能保证对于

0..n[i]间的每一个整数，均可以用若干个系数的和表示，这个证明可以分 0..2^k-1 和 2^k..n[i]两段来分别讨论得出，并不难，希望你自己思考尝试一下。

这样就将第 i 种物品分成了 O(log n[i])种物品，将原问题转化为了复杂度为 O(V*Σlog n[i])的 01 背包问题，是很大的改进。

三． 最长 XX 子序列：

例题：

逢低吸纳”是炒股的一条成功秘诀。如果你想成为一个成功的投资者，就要遵守这条秘诀:

"逢低吸纳,越低越买"

这句话的意思是：每次你购买股票时的股价一定要比你上次购买时的股价低.按照这个规则购买股票的次数越多越好，看看你最多能按这个规则买几次。

给定连续的 N 天中每天的股价。你可以在任何一天购买一次股票，但是购买时的股价一定要比你上次购买时的股价低。写一个程序，求出最多能买几次股票。

以下面这个表为例, 某几天的股价是:

天数 1 2 3 4 5 6 7 8 9 10 11 12
股价 68 69 54 64 68 64 70 67 78 62 98 87

这个例子中, 聪明的投资者(按上面的定义), 如果每次买股票时的股价都比上一次买时低，那么他最多能买 4 次股票。一种买法如下(可能有其他的买法):

天数 2 5 6 10
股价 69 68 64 62

PROGRAM NAME: buylow
INPUT FORMAT
第 1 行： N (1 <= N <= 5000), 表示能买股票的天数。
第 2 行以下: N 个正整数 (可能分多行) ， 第 i 个正整数表示第 i 天的股价. 这些正整数大小不会超过 longint(pascal)/long(c++).

SAMPLE INPUT (file buylow.in)
12
68 69 54 64 68 64 70 67
78 62 98 87
OUTPUT FORMAT
只有一行，输出两个整数：

能够买进股票的天数
长度达到这个值的股票购买方案数量
在计算解的数量的时候，如果两个解所组成的字符串相同，那么这样的两个解被认为是相同的（只能算做一个解）。因此，两个不同的购买方案可能产生同一个字符串，这样只能计算一次。

SAMPLE OUTPUT (file buylow.out)

4 2
程序：

最长不下降序列长度+个数

```
program buylow
var i,n,j:longint;
      a:array[0..5000] of longint;
      f,g:array[0..5000] of longint;
      yes:array[0..maxint] of boolean;
begin
   assign(input,'buylow.in');
   reset(input);
   assign(output,'buylow.out');
   rewrite(output);

      readln(n);
      for i:=1 to n do
         read(a[i]);

      for i:=0 to n do f[i]:=1;
      a[0]:=maxint;
      for i:=n-1 downto 0 do
         for j:=i+1 to n do
            if (a[i]>a[j]) and (f[i]<f[j]+1) then
              f[i]:=f[j]+1;
    writeln(f[0]-1);{注意-1}

   for i:=n downto 0 do
        begin
           fillchar(yes,sizeof(yes),false);
           for j:=i+1 to n+1 do
              if (f[j]=f[i]-1)and (a[i]>a[j]) and (not yes[a[j]]) then
                   begin
                          inc(g[i],g[j]);
                          yes[a[j]]:=true;
                   end;
           if g[i]=0 then g[i]:=1;{f[i]=1 时上面的 for 循环不会给 g[i]赋值，特加此句}
```

```
        end;
  writeln(g[0]);
  close(input);
  close(output);
end.
```

四． 资源分配型动规：

资源分配模型

资源分配模型的动态规划,也是在信息学竞赛中经常使用的。这类型的题目一般是:给定 m 个资源,分配给 n 个部门,第 i 个部门获得 j 个资源有个盈利值,问如何分配这 m 个资源能使获得的盈利最大,求最大盈利。

这类型的题目一般用资源数做状态,数组 f[i,j]表示前个 i 个部门分配 j 个资源的最大盈利,则状态转移方程为:

f[i,j]:=max{f[i-1,k]+value[i,j-k]}　(0≤k≤j)

程序框架如下:

```
Procedure dynamic；
 Var i,j,k:longint；
Begin
 For i:=1 to n do
  For j:=0 to m do
   For k:=0 to j do
    If f[i-1l,k]+value[i,j-k]>f[i,j] then f [i,j]:=f[i-1,k]+value[I,j-k]；
  Wrneln(f[n,m])；
 End；
```

投资问题(invest.pas)

【问题描述】

有n 万元的资金，可投资于m 个项目，其中m和n 为小于100的自然数。对第i(1≤i≤m)个项目投资j 万元(1≤j≤n，且j 为整数)可获得的回报为Q(i,j)，请你编一个程序，求解并输出最佳的投资方案(即获得回报总值最高的投资方案)。

输入数据放在文件invest.in 中，格式如下：

```
m n
Q(1,0) Q(1,1)……Q(1,n)
Q(2,0) Q(2,1)……Q(2,n)
……
Q(m,0) Q(m,1)……Q(m,n)
```

输出数据放在文件invest.out中，格式为：

```
r(1) r(2) …… r(m) P
```

其中r(i)(1≤i≤m)表示对第i 个项目的投资万元数，P 为总的投资回报值，保留两位有效数字，任意两个数之间空一格。当存在多个并列的最佳投资方案时，只要求输出其中之一即可。

【输入样例】

invest.in：

2 3

0 1.1 1.3 1.9

0 2.1 2.5 2.6

【输出样例】

invest.out：

1 2 3.6

【算法分析】

本题可供考虑的递推角度只有资金和项目两个角度，从项目的角度出发，逐个项目地增加可以看出只要算出了对前k 个项目投资j 万元最大投资回报值(1≤j≤n)，就能推出增加第k+1 个项目后对前k+1 个项目投资j万元最大投资回报值(1≤j≤n)，设P[k,j]为前k个项目投资j 万元最大投资回报值，

则P[k+1,j]= Max(P[k,i]+Q[k+1,j-i]),0<=i<=j，k=1时，对第一个项目投资j 万元最大投资回报值(0≤j≤n)是已知的(边界条件)。

【算法设计】

动态规划的时间复杂度相对于搜索算法是大大降低了，却使空间消耗增加了很多。所以在设计动态规划的时候，需要尽可能节省空间的占用。本题中如果用二维数组存储原始数据和最大投资回报值的话，将造成空间不够，事实上，从前面的问题分析可知：在计算对前k+1个项目投资j 万元最大投资回报值时，只要用到矩阵Q 的第k+1 行数据和对前k 个项目投资j 万元最大投资回报值，而与前面的数据无关，后者也只需有一个一维数组存储即可，程序中用一维数组Q存储当前项目的投资回报值，用一维数组maxreturn存储对当前项目之前的所有项目投资j万元最大投资回报值(0≤j≤n)，用一维数组temp 存储对到当前项目为止的所有项目投资j 万元最大投资回报值(0≤j≤n)。为了输出投资方案，程序中使用了一个二维数组invest，invest[k,j]记录了对前k 个项目投资j 万元获得最大投资回报时投资在第k个项目上的资金数。

【参考程序】

```
program invest(input,output);
const
   maxm=100;
   maxn=100;
type
   arraytype=array [0..maxn] of real;
var
   i,j,k,m,n,rest:integer;
   q,maxreturn,temp:arraytype;
   invest:array[1..maxm,0..maxn] of integer;
   result:array[1..maxm] of integer;
begin
  assign(input,' invest.in' ); reset(input);
  readln(m,n);
  for j:=0 to n do read(q[j]);
  readln;
  for i:=1 to m do
   for j:=0 to n do invest[i,j]:=0;
  maxreturn:=q;
  for j:=0 to n do invest[1,j]:=j;
  for k:=2 to m do
   begin
    temp:=maxreturn;
    for j:=0 to n do invest[k,j]:=0;
```

```
   for j:=0 to n do read(q[j]);
   readln;
   for j:=0 to n do
    for i:=0 to j do
     if maxreturn[j-i]+q[i]>temp[j] then
      begin
        temp[j]:=maxreturn[j-i]+q[i];
        invest[k,j]:=i
      end;
     maxreturn:=temp
  end;
close(input);
assign(output,' invest.out' );rewrite(output);
rest:=n;
for i:=m downto 1 do
begin
 result[i]:=invest[i,rest];
 rest:=rest-result[i]
end;
for i:=1 to m do write(result[i],'  ' );
writeln(maxreturn[n]:0:2);
close(output)
end.
```

五． 区间动规：

例一：乘积最大

设有一个长度为 N 的数字串，要求选手使用 K 个乘号将它分成 K+1 个部分，找出一种分法，使得这 K+1 个部分的乘积能够为最大。

分析：数组 f[I,j,l] 存放从第 I 个数到第 j 个数，有 l 个乘号的最大值。

预处理：先求出有一个乘号时的解。

```
procedure YU;
begin
 for i:=1 to n do
  for j:=1 to k do
   f[i,i,j]:=0;
 for i:=1 to n-1 do
  for j:=i+1 to n do
   for l:=i to j-1 do
    begin
     t:=copy(s,i,l-i+1);
     r:=copy(s,l+1,j-l);
     val(t,p,o);
     val(r,q,o);
     f[i,j,1]:=max(f[i,j,1],p*q);
    end;
end;
```

动态规划：

```
procedure DP;
```

```
begin
 for l:=2 to k do
  for i:=1 to n-l-1 do
   for j:=i+l to n do
    for o:=i+1 to j do
     begin
      r:=copy(s,o,j-o+1);
      val(r,p,tol);
      f[i,j,l]:=max(f[i,j,l],f[i,o-1,l-1]*p);
     end;
 writeln(f[1,n,k]);
end;
```

例二：乘法游戏

奶牛们在玩一种乘法游戏，它的具体做法是在一行牌上进行的，每一张牌包括了一个正整数。在每一次移动中，玩家出一张牌，得分是用它的数字乘以它左边和右边的数字，所以不允许拿第 1 张和最后 1 张牌。最后一次移动后，这里只剩下两张牌。

奶牛们的目标是使得分的和最小。

分析：数组 f[I,j] 存放从第 I 个数到第 j 个数的最优解。

读入及预处理： 先求每相邻三个数的乘积

```
procedure init;
begin
 readln(n);
 for i:=1 to n do
  read(m[i]);
 fillchar(f,sizeof(f),$7f);
 for i:=1 to n-2 do
  f[i,i+2]:=m[i]*m[i+1]*m[i+2];
 for i:=1 to n do
  begin
   f[i,i]:=0;
   f[i,i+1]:=0;
  end;
end;
```

动态规划：

```
procedure DP;
begin
 for i:=n-3 downto 1 do  {注意是 Downto}
  for j:=i+3 to n do  {注意是 to}
   for k:=i+1 to j-1 do
    begin
     f[i,j]:=min(f[i,j],f[i,k]+f[k,j]+m[i]*m[k]*m[j]);
    end;
```

```
    writeln(f[1,n]);
  end;
```

例三：数字游戏

在你面前有一圈整数（一共 n 个），你要按顺序将其分为 m 个部分，各部分内的数字相加，相加所得的 m 个结果对 10 取模后再相乘，最终得到一个数 k。游戏的要求是使你所得的 k 最大或者最小。

```
Program shuziyouxi;
var n,m,t10,t:integer;
    i,j,k:integer;
    a:array[0..49] of integer;
{储存初始值，因为同顺序无关，所以将 a[n]存入 a[0]；}
    min,max:array[0..49,1..50,0..2] of longint;
    m1,m2,temp:longint;
begin
    readln(n,m);
    for i:=0 to n-1 do
    readln(a[i]);
    for i:=0 to n-1 do
    begin
      temp:=0;
      for j:=1 to n do
      begin
        temp:=temp+a[(i+j-1) mod n];
        if temp>=0 then t10:=temp mod 10
        else t10:=(10-((-1)*temp) mod 10) mod 10;
        {t10 为从 a[i]开始的 j 个数的和，注意负数的处理}
        min[i,j,0]:=t10;{无断点时的状态}
        max[i,j,0]:=t10;
        min[i,j,1]:=t10;{有一个断点的状态}
        max[i,j,1]:=t10;
      end;
    end;
    for k:=1 to m-1 do{共寻找 m-1 次，搜索有 k+1 个断点的状态}
    begin
      for j:=k+1 to n do{区间宽度为 j}
      for i:=0 to n-1 do{从 i 开始开始的区间，[i,（i+j-1）mad n]}
      begin
        m1:=2000000000;
        m2:=-2000000000;
            for t:=k to j-1 do {以 t 为分界点搜索}
            begin
              if m1>min[i,t,1]*min[(i+t)mod n,j-t,0]
              then m1:=min[i,t,1]*min[(i+t)mod n,j-t,0];
```

```
                    if m2<max[i,t,1]*max[(i+t)mod n,j-t,0]
                    then m2:=max[i,t,1]*max[(i+t)mod n,j-t,0];
                  end;
              min[i,j,2]:=m1;
              max[i,j,2]:=m2;
            end;
            for i:=0 to n-1 do
            for j:=1 to n do
            begin{[i,j,1]更新为最新状态，[i,j,1]与[i,j,2]为滚动数组}
              min[i,j,1]:=min[i,j,2];
              max[i,j,1]:=max[i,j,2];
            end;
        end;
         m1:=2000000000;m2:=-2000000000;
         for i:=0 to n-1 do {搜索最值}
          begin
              if m1>min[i,n,1] then m1:=min[i,n,1];
              if m2<max[i,n,1] then m2:=max[i,n,1];
          end;
         writeln(m1);
         writeln(m2);
end.
```

NOIP2006TG_1_能量项链

描述 Description

在 Mars 星球上，每个 Mars 人都随身佩带着一串能量项链。在项链上有 N 颗能量珠。能量珠是一颗有头标记与尾标记的珠子，这些标记对应着某个正整数。并且，对于相邻的两颗珠子，前一颗珠子的尾标记一定等于后一颗珠子的头标记。因为只有这样，通过吸盘（吸盘是 Mars 人吸收能量的一种器官）的作用，这两颗珠子才能聚合成一颗珠子，同时释放出可以被吸盘吸收的能量。如果前一颗能量珠的头标记为 m，尾标记为 r，后一颗能量珠的头标记为 r，尾标记为 n，则聚合后释放的能量为 （Mars 单位），新产生的珠子的头标记为 m，尾标记为 n。

需要时，Mars 人就用吸盘夹住相邻的两颗珠子，通过聚合得到能量，直到项链上只剩下一颗珠子为止。显然，不同的聚合顺序得到的总能量是不同的，请你设计一个聚合顺序，使一串项链释放出的总能量最大。

例如：设 N=4，4 颗珠子的头标记与尾标记依次为(2，3) (3，5) (5，10) (10，2)。我们用记号 ⊕ 表示两颗珠子的聚合操作，(j ⊕ k)表示第 j，k 两颗珠子聚合后所释放的能量。则第 4、1 两颗珠子聚合后释放的能量为：

(4 ⊕ 1)=10*2*3=60。

这一串项链可以得到最优值的一个聚合顺序所释放的总能量为

((4 ⊕ 1) ⊕ 2) ⊕ 3）=10*2*3+10*3*5+10*5*10=710。

输入格式 Input Format

输入文件 energy.in 的第一行是一个正整数 N（4≤N≤100），表示项链上珠子的个数。第二行是 N 个用空格隔开的正整数，所有的数均不超过 1000。第 i 个数为第 i 颗珠子的头标记（1≤i≤N），当 i<N 时，第 i 颗珠子的尾标记应该等于第 i+1 颗珠子的头标记。第 N 颗珠子的尾标记应该等于第 1 颗珠子的头标记。

至于珠子的顺序，你可以这样确定：将项链放到桌面上，不要出现交叉，随意指定第一颗珠子，然后按顺时针方向确定其他珠子的顺序。

输出格式 Output Format

输出文件 energy.out 只有一行，是一个正整数 E（E≤2.1*109），为一个最优聚合顺序所释放的总能量。

样例输入 Sample Input

```
4
2  3  5  10
```

样例输出 Sample Output

```
710
```

时间限制 Time Limitation

各个测试点 1s

```
var f:array[0..201,0..201] of longint;
    a:array[0..200,1..2] of longint;
    i,j,k,p,n,ans:longint;

function max(x,y:longint):longint;
 begin
  if x>y then exit(x) else exit(y);
 end;

begin
 readln(n);
 for i:=1 to n do
  begin
```

```
    read(a[i,1]);
    a[i-1,2]:=a[i,1];
    a[i+n,1]:=a[i,1];
    a[i+n-1,2]:=a[i,1];
  end;
  a[n,2]:=a[1,1];a[n+n,2]:=a[n,2];
  ans:=0;
  fillchar(f,sizeof(f),0);
  for i:=n*2 downto 1 do
    for j:=i+1 to n*2 do
      for k:=i to j-1 do
        f[i,j]:=max(f[i,j],f[i,k]+f[k+1,j]+a[i,1]*a[j,2]*a[k+1,1]);

  for i:=1 to n do
    ans:=max(f[i,i+n-1],ans);
  writeln(ans);
end.
```

六. 树型动规：

k) 树型动规基本模型

根—>叶

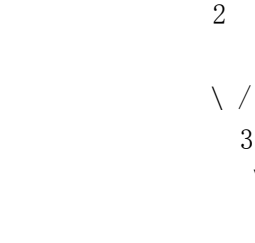
叶—>根：既根的子节点传递有用的信息给根，完后根得出最优解的过程。

给出一棵树，每个结点有一个权，求一棵子树，使得其顶点的权值最大；或者给出一棵树．树边有一个权代表结点间的距离，对每个结点，求出它到其他结点的最远距离。这两个问题用树型动态规划来做的话都是 O(n)的。

例：二叉苹果树

问题描述

有一棵苹果树，如果树枝有分叉，一定是分 2 叉（就是说没有只有 1 个儿子的结点）这棵树共有 N 个结点（叶子点或者树枝分叉点），编号为卜 N，树根编号一定是 1。我们用一根树枝两端连接的结点的编号来描述一根树枝的位置。



现在这颗树枝条太多了，需要剪枝。但是一些树枝有苹果。给定需要保留的树枝数量，求出最多能留住多少苹果。

问题分析：

f[k，j]表示以结点 k 为父结点的子树中，保留 j 个树枝能留住的最多苹果数量，f[k,0]=0;f[k,1]=c[k]; 子结点树枝和为 j-1;

则状态转移方程为：f[k，j]:=max{f[son[k,1], i]+f[son[k,2], j-I-1]+c[k]}　(2≤J≤q+1;0≤i≤J;),

f[1,q+1]为最优解。

```
var
  a,b:array[0..100,0..100]of integer;
  son:array[1..100,1..2]of integer;
  s,c:array[1..100]of integer;
  n,q,i,j,si,x,y:integer;
procedure maketree(k:integer);{建树并在s数组中储存树的后序遍历}
var
  i,j:integer;
begin
  if k=0 then exit;{表示k的父亲已没有儿子}
  j:=0;
  for i:=1 to n do
  if a[k,i]>=0
  then begin
       inc(j);
       son[k,j]:=i;{记录第j个儿子}
       a[i,k]:=-1;
       c[i]:=a[k,i];{记录I点的值}
       if j=2 then break;
       end;
  maketree(son[k,1]);
  maketree(son[k,2]);
  inc(si);
  s[si]:=k;{s数组记录数的后序}
end;
procedure tree(k:integer);
var
  i,j:integer;
begin
  if son[k,1]=0
    then begin{k没有孩子}
       for i:=1 to q+1 do
        b[k,i]:=c[k];
       end
  else begin
       b[k,1]:=c[k];
       for j:=2 to q+1 do
       for i:=0 to j-1 do
        if b[k,j]<b[son[k,1],i]+b[son[k,2],j-i-1]+c[k]
        then b[k,j]:=b[son[k,1],i]+b[son[k,2],j-i-1]+c[k];
       end;
end;
```

```
begin
  readln(n,q);
  for i:=1 to n do
  for j:=1 to n do
    a[i,j]:=-1;
  fillchar(son,sizeof(son),0);
  fillchar(b,sizeof(b),0);
  for i:=1 to n-1 do
    begin
      readln(x,y,a[x,y]);
      a[y,x]:=a[x,y];
    end;
  si:=0;
  maketree(1);
  for i:=1 to n do
   tree(s[i]);
  writeln(b[1,q+1]);
end.
```

例题2：

NOIP2003TG_3_加分二叉树

描述 **Description**

设一个**n**个节点的二叉树**tree**的中序遍历为（**l,2,3,…,n**），其中数字**1,2,3,…,n**为节点编号。每个节点都有一个分数（均为正整数），记第**i**个节点的分数为**di**，**tree**及它的每个子树都有一个加分，任一棵子树**subtree**（也包含**tree**本身）的加分计算方法如下：

subtree的左子树的加分× **subtree**的右子树的加分＋**subtree**的根的分数

若某个子树为空，规定其加分为**1**，叶子的加分就是叶节点本身的分数。不考虑它的空子树。

试求一棵符合中序遍历为（**1,2,3,…,n**）且加分最高的二叉树**tree**。要求输出；

（**1**）**tree**的最高加分

（**2**）**tree**的前序遍历

输入格式 **Input Format**

第**1**行：一个整数**n**（**n**＜**30**），为节点个数。

第**2**行：**n**个用空格隔开的整数，为每个节点的分数（分数＜**100**）。

输出格式 **Output Format**

第**1**行：一个整数，为最高加分（结果不会超过**4,000,000,000**）。

第**2**行：**n**个用空格隔开的整数，为该树的前序遍历。

样例输入 **Sample Input**

5
5 7 1 2 10

样例输出 **Sample Output**

145
3 1 2 4 5

时间限制 **Time Limitation**

每点**1s**

参考程序：

```
var i,j,k,n,m,t:longint;
a:array[0..10000] of longint;
f,g:array[0..1000,0..1000] of longint;
procedure treedp(l,r:longint);
var i,j:longint;
begin
if l>r then begin f[l,r]:=1;exit;end;
if l=r then begin f[l,r]:=a[l];g[l,r]:=l;exit;end;
if f[l,r]<>0 then exit;
for i:=l to r do
begin
treedp(l,i-1);
treedp(i+1,r);
t:=a[i]+f[l,i-1]*f[i+1,r];
if t>f[l,r] then begin f[l,r]:=t;
g[l,r]:=i;end;
end;
end;
procedure pre(l,r:longint);
begin
write(g[l,r],' ');
if l<=g[l,r]-1   then pre(l,g[l,r]-1);
if g[l,r]+1<=r then pre(g[l,r]+1,r);
end;
procedure print;
begin
writeln(f[1,n]);
pre(1,n);
writeln;
```

```
end;
begin
readln(n);
for i:=1 to n do
begin
read(a[i]);
end;
fillchar(f,sizeof(f),0);
fillchar(g,sizeof(g),0);
treedp(1,n);
print;
end.
```

USACO2.3.2奶牛家谱

描述 **Description**

农民约翰准备购买一群新奶牛。在这个新的奶牛群中，每一个母亲奶牛都生两小奶牛。这些奶牛间的关系可以用二叉树来表示。这些二叉树总共有**N**个节点**(3 <= N < 200)**。这些二叉树有如下性质**:**

每一个节点的度是**0**或**2**。度是这个节点的孩子的数目。

树的高度等于**K(1 < K < 100)**。高度是从根到任何叶子的最长的路径上的节点的数目**;** 叶子是指没有孩子的节点。

有多少不同的家谱结构**?** 如果一个家谱的树结构不同于另一个的，那么这两个家谱就是不同的。输出可能的家谱树的个数除以**9901**的余数。

PROGRAM NAME: nocows

INPUT FORMAT

第**1**行**:** 两个空格分开的整数**,** **N**和**K**。

SAMPLE INPUT (file nocows.in)

5 3

OUTPUT FORMAT

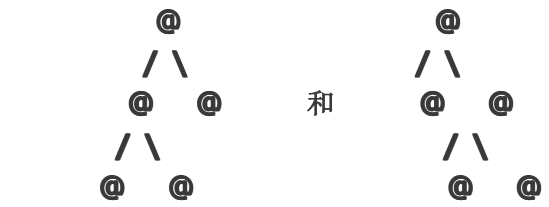
第 **1** 行**:** 一个整数，表示可能的家谱树的个数除以**9901**的余数。

SAMPLE OUTPUT (file nocows.out)

2

OUTPUT DETAILS:

有**5**个节点，高为**3**的两个不同的家谱：



参考程序：

```
var n,m,i,j,k:longint;
   f:array[0..201,0..101] of longint;

begin
```

```
readln(n,m);
for i:=1 to m do f[1,i]:=1;

for i:=1 to n do
 for j:=1 to m do
  for k:=1 to i-2 do
   f[i,j]:=(f[i,j]+f[k,j-1]*f[i-k-1,j-1]) mod 9901;

 writeln((f[n,m]-f[n,m-1]+9901)mod 9901);
end.
```

七． 枚举型动规：

例题：

题 3、排列小球（ball.pas/c/cpp）

[问题描述]

LG 意外收到了一份礼物,是一大堆小球..准确地说,小球共有 3 种颜色:红黄白.LG 决定从这些小球中选出 n 个来排成一列.如何使得这 n 个排成一排的小球看着顺眼,这是个问题..

对于连续放置的三个小球,LG 都会定义一个”舒适度”.比如他认为三个颜色分别为红红黄的小球连在一起,舒适度为 12.

而对于全部 n 个小球,会有 n-2 个连续的三个小球.这 n-2 个组合的舒适度之和,定义为整个序列的舒适度.现在 LG 想知道,整个序列的舒适度的最大值是多少.

三个小球总共有 27 种摆法,舒适度分别为:

红红红 4	红红白 12	红红黄 17
红白红 11	红白白 22	红白黄 24
红黄红 27	红黄白 37	红黄黄 30
白红红 6	白红白 18	白红黄 21
白白红 12	白白白 3	白白黄 16
白黄红 19	白黄白 22	白黄黄 24
黄红红 10	黄红白 13	黄红黄 7
黄白红 6	黄白白 3	黄白黄 4
黄黄红 7	黄黄白 15	黄黄黄 2

你可以认为,LG 手里有无数小球,每个颜色都是.

突然天将画外音！LG 说，呃，这样的话万一暴力能做的话不久囧了，于是乎，ben 对 LG 说：“你加一点东东不就行了么。”，“加什么？”LG 问道。

于是 XX 就大胆的改起了题目。

听好下面是 XX 对题目的赘述。

呃，XX 说我有特殊能力，我能给你额外的合适度，你愿意要么？

你不会告诉我你说你不愿意要吧？那也太扯了！

然后 XX 就叽里咕噜了半天，大概意思就是，如果你在第 i 个位置放球 j 的话，XX 会给你额外的合适度。

[输入描述]

一个整数,n(5<=n<=100000),表示序列的长度。

接下来有 n 行，每行三个数。第 i 行分别表示在第 i 个位置放红、白、黄时，ben 给你的附加合适度。

其他输入不超过 40

[输出描述]

一个整数,表示舒适度的最大值。

[样例输入]

3
2 3 4
3 4 2
1 2 1

[样例输出]

43

[数据说明]

30%　5<=n<=10000
100%　5<=n<=100000

程序如下：

```
const
  b:array[1..3,1..3,1..3]of integer=
  (((4,12,17),(11,22,24),(27,37,30)),
   ((6,18,21),(12,3,16),(19,22,24)),
   ((10,13,7),(6,3,4),(7,15,2)));
var
  a:array[1..100000,1..3]of integer;
  f,g:array[1..3,1..3]of longint;
  x:array[1..3,1..3]of integer;
  i,j,k,m,n,s,t:longint;
begin
  assign(input,'ball.in');
  reset(input);
  assign(output,'ball.out');
  rewrite(output);
  readln(n);
  for i:=1 to n do readln(a[i,1],a[i,2],a[i,3]);
  for i:=1 to 3 do
   for j:=1 to 3 do
    f[i,j]:=a[1,i]+a[2,j];
  for t:=3 to n do
   begin
```

```
      fillchar(g,sizeof(g),0);
      for i:=1 to 3 do
       for j:=1 to 3 do
        for k:=1 to 3 do
         if g[i,j]<f[k,i]+b[k,i,j]+a[t,j] then g[i,j]:=f[k,i]+b[k,i,j]+a[t,j];
      f:=g;
    end;
  s:=0;
  for i:=1 to 3 do
   for j:=1 to 3 do
    if f[i,j]>s then s:=f[i,j];
  writeln(s);
  close(input);
  close(output);
end.
```

描述 **Description**

设有**N*N**的方格图，我们将其中的某些方格填入正整数，
而其他的方格中放入**0**。
某人从图得左上角出发，可以向下走，也可以向右走，直到到达右下角。
在走过的路上，他取走了方格中的数。（取走后方格中数字变为**0**）
此人从左上角到右下角共走**3**次，试找出**3**条路径，使得取得的数总和最大。

输入格式 **Input Format**

第一行**:N (4<=N<=20)**
接下来一个**N*N**的矩阵，矩阵中每个元素不超过**80**，不小于**0**

输出格式 **Output Format**

一行，表示最大的总和。

样例输入 **Sample Input**

```
4
1 2 3 4
2 1 3 4
1 2 3 4
1 3 2 4
```

样例输出 **Sample Output**

```
39
```

程序：

```
var
 f   : array[0..200,0..200,0..200]of longint;
 map : array[0..200,0..200]of longint;
 i, j, l, r, m, n, ans : longint;
function max(a,b,c,d,z,x,y,v:longint):longint;
          begin
               max := a;
               if max < b then max := b;
               if max < c then max := c;
```

```
            if max < d then max := d;
            if max < z then max := z;
            if max < x then max := x;
            if max < y then max := y;
            if max < v then max := v;
        end;
begin
    readln(n);
    for i := 1 to n do
      for j := 1 to n do read(map[i,j]);
    fillchar(f,sizeof(f),0);
    for i := 3 to n*2-3 do
     for j := i-2 downto 1 do
      for l := i-1 downto j+1 do
        for r := i    downto l+1 do
         f[j,l,r] := max(f[j,l,r],
                         f[j,l-1,r-1],
                         f[j,l,r-1],
                         f[j,l-1,r],
                         f[j-1,l-1,r-1],
                         f[j-1,l,r],
                         f[j-1,l-1,r],
                         f[j-1,l,r-1])+map[i-j+1,j]+map[i-l+1,l]+map[i-r+1,r];
    ans := f[n-2,n-1,n]+map[1,1]+map[1,2]+map[2,1]+map[n,n]+map[n,n-1]+map[n-1,n];
    writeln(ans);
end.
```

描述 **Description**

设有**N*N**的方格图**(N<=10,**我们将其中的某些方格中填入正整数**,**而其他的方格中则放入数字**0**。如下图所示（见样例）：

某人从图的左上角的**A** 点出发，可以向下行走，也可以向右走，直到到达右下角的**B**点。在走过的路上，他可以取走方格中的数（取走后的方格中将变为数字**0**）。

此人从**A**点到**B** 点共走两次，试找出**2**条这样的路径，使得取得的数之和为最大。

输　入

输入的第一行为一个整数**N**（表示**N*N**的方格图），接下来的每行有三个整数，前两个表示位置，第三个数为该位置上所放的数。一行单独的**0**表示输入结束。

输　出

只需输出一个整数，表示**2**条路径上取得的最大的和。

样　例　：

输 入

```
8
2  3  13
2  6   6
3  5   7
```

4 4 14
5 2 21
5 6 4
6 3 15
7 2 14
0 0 0

输 出

67

程序：

```
const maxn=10;
type arraytype=array [0..maxn,0..maxn] of longint;
var i,j,k,n,i1,i2,j1,j2:longint;
    data:arraytype;
    sum:array [0..maxn,0..maxn,0..maxn,0..maxn] of longint;
function max(x,y:longint):longint;
begin
     if x>y then max:=x else max:=y;
end;
BEGIN  {main}
      for i:=1 to maxn do
          for j:=1 to maxn do data[i,j]:=0;
      readln(n);
      repeat
            readln(i,j,k);
            data[i,j]:=k
      until (i=0) and (j=0) and (k=0);
      fillchar(sum,sizeof(sum),0);
      for i1:=1 to n do
          for j1:=1 to n do
              for i2:=1 to n do
                  for j2:=1 to n do
                  begin
                       if sum[i1-1,j1,i2-1,j2]>sum[i1,j1,i2,j2]
                          then
sum[i1,j1,i2,j2]:=sum[i1-1,j1,i2-1,j2];
                       if sum[i1-1,j1,i2,j2-1]>sum[i1,j1,i2,j2]
                          then
sum[i1,j1,i2,j2]:=sum[i1-1,j1,i2,j2-1];
                       if sum[i1,j1-1,i2-1,j2]>sum[i1,j1,i2,j2]
                          then
sum[i1,j1,i2,j2]:=sum[i1,j1-1,i2-1,j2];
                       if sum[i1,j1-1,i2,j2-1]>sum[i1,j1,i2,j2]
                          then
sum[i1,j1,i2,j2]:=sum[i1,j1-1,i2,j2-1];
```

```
                                sum[i1,j1,i2,j2]:=sum[i1,j1,i2,j2]
+data[i1,j1];
                                if (i1<>i2) or (j1<>j2)
                                   then sum[i1,j1,i2,j2]:=sum[i1,j1,i2,j2]
+data[i2,j2]
                            end;
        writeln(sum[n,n,n,n]);
        END.
```

1. 资源问题 1
   -----机器分配问题
   F[I,j]:=max(f[i-1,k]+w[i,j-k])

2. 资源问题 2
   ------01 背包问题
   F[I,j]:=max(f[i-1,j-v[i]]+w[i],f[i-1,j]);

3. 线性动态规划 1
   -----朴素最长非降子序列
   F[i]:=max{f[j]+1}

4. 剖分问题 1
   -----石子合并
   F[i,j]:=min(f[i,k]+f[k+1,j]+sum[i,j]);

5. 剖分问题 2
   -----多边形剖分
   F[I,j]:=min(f[i,k]+f[k,j]+a[k]*a[j]*a[i]);

6. 剖分问题 3
   ------乘积最大
   f[i,j]:=max(f[k,j-1]*mult[k,i]);

7. 资源问题 3
   -----系统可靠性(完全背包)
   F[i,j]:=max{f[i-1,j-c[i]*k]*P[I,x]}

8. 贪心的动态规划 1
   -----快餐问题
   F[i,j,k]:=max{f[i-1,j',k']+(T[i]-(j-j')*p1-(k-k')*p2) div
p3}

9. 贪心的动态规划 2
   -----过河 f[i]=min{{f(i-k)} (not stone[i])
                        {f(i-k)}+1} (stone[i]); +贪心压缩状态

10. 剖分问题 **4**

-----多边形-讨论的动态规划

F[i,j]:=max{正正 **f[I,k]*f[k+1,j];**

负负 **g[I,k]*f[k+1,j];**

正负 **g[I,k]*f[k+1,j];**

负正 **f[I,k]*g[k+1,j];}** **g** 为 **min**

11. 树型动态规划 **1**

-----加分二叉树 **(**从两侧到根结点模型**)**

F[I,j]:=max{f[I,k-1]*f[k+1,j]+c[k]}

12. 树型动态规划 **2**

-----选课 **(**多叉树转二叉树**,**自顶向下模型**)**

F[I,j]表示以 **i** 为根节点选 **j** 门功课得到的最大学分

f[i,j]:=max{f[t[i].l,k]+f[t[i].r,j-k-1]+c[i]}

13. 计数问题 **1**

-----砝码称重

f[f[0]+1]=f[j]+k*w[j];

(1<=i<=n; 1<=j<=f[0]; 1<=k<=a[i];)

14. 递推天地 **1**

------核电站问题

f[-1]:=1; f[0]:=1;

f[i]:=2*f[i-1]-f[i-1-m]

15. 递推天地 **2**

------数的划分

f[i,j]:=f[i-j,j]+f[i-1,j-1];

16. 最大子矩阵 **1**

-----一最大 **01** 子矩阵

f[i,j]:=min(f[i-1,j],v[i,j-1],v[i-1,j-1])+1;

ans:=maxvalue(f);

17. 判定性问题 **1**

-----能否被 **4** 整除

g[1,0]:=true; g[1,1]:=false; g[1,2]:=false; g[1,3]:=false;

g[i,j]:=g[i-1,k] and ((k+a[i,p]) mod 4 = j)

18. 判定性问题 **2**

-----能否被 **k** 整除

f[I,j±n[i] mod k]:=f[i-1,j]; **-k<=j<=k; 1<=i<=n**

20. 线型动态规划 **2**

-----方块消除游戏

```
f[i,i-1,0]:=0
f[i,j,k]:=max{f[i,j-1,0]+sqr(len(j)+k),
             f[i,p,k+len[j]]+f[p+1,j-1,0]}
ans:=f[1,m,0]
```

21. 线型动态规划 **3**

-----最长公共子串，**LCS** 问题

```
f[i,j]={0        (i=0)&(j=0);
    f[i-1,j-1]+1    (i>0,j>0,x[i]=y[j]);
    max{f[i,j-1]+f[i-1,j]}} (i>0,j>0,x[i]<>y[j]);
```

22. 最大子矩阵 **2**

-----最大带权 **01** 子矩阵 **O(n^2*m)**

枚举行的起始，压缩进数列，求最大字段和，遇 **0** 则清零

23. 资源问题 **4**

-----装箱问题(判定性 **01** 背包)

```
f[j]:=(f[j] or f[j-v[i]]);
```

24. 数字三角形 **1**

-----朴素の数字三角形

```
f[i,j]:=max(f[i+1,j]+a[I,j],f[i+1,j+1]+a[i,j]);
```

25. 数字三角形 **2**

-----晴天小猪历险记之 **Hill**

同一阶段上暴力动态规划

```
if[i,j]:=min(f[i,j-1],f[I,j+1],f[i-1,j],f[i-1,j-1])+a[i,j]
```

26. 双向动态规划 **1**

数字三角形 **3**

-----小胖办证

```
f[i,j]:=max(f[i-1,j]+a[i,j],f[i,j-1]+a[i,j],f[i,j+1]+a[i,j])
```

27. 数字三角形 **4**

-----过河卒

```
//边界初始化
f[i,j]:=f[i-1,j]+f[i,j-1];
```

28. 数字三角形 **5**

-----朴素的打砖块

f[i,j,k]:=max(f[i-1,j-k,p]+sum[i,k],f[i,j,k]);

29. 数字三角形 **6**

-----优化的打砖块

f[I,j,k]:=max{g[i-1,j-k,k-1]+sum[I,k]}

30. 线性动态规划 **3**

-----打鼹鼠’

f[i]:=f[j]+1;(abs(x[i]-x[j])+abs(y[i]-y[j])<=t[i]-t[j])

31. 树形动态规划 **3**

-----贪吃的九头龙

$$f[i,j,k]=\min\begin{Bmatrix} f[l,j',1]+f[r,j-j'-1,k]+d[k,1]*w[i,p[i]] \\ f[l,j',0]+f[r,j-j',k]+d[k,0]*w[i,p[i]] \end{Bmatrix}$$

$$d[i,j]=\begin{Bmatrix} 1((i=1)\&(j=1))or((i=0)\&(j=0)and(m=2)) \\ 0 \end{Bmatrix}$$

32. 状态压缩动态规划 **1**

-----炮兵阵地

Max(f[Q*(r+1)+k],g[j]+num[k])

If (map[i] and plan[k]=0) and

((plan[P] or plan[q]) and plan[k]=0)

33. 递推天地 **3**

-----情书抄写员

f[i]:=f[i-1]+k*f[i-2]

34. 递推天地 **4**

-----错位排列

f[i]:=(i-1)(f[i-2]+f[i-1]);

f[n]:=n*f[n-1]+(-1)^(n-2);

35. 递推天地 **5**

-----直线分平面最大区域数

f[n]:=f[n-1]+n

:=n*(n+1) div 2 + 1;

36. 递推天地 **6**

-----折线分平面最大区域数

f[n]:=(n-1)(2*n-1)+2*n;

37. 递推天地 **7**

-----封闭曲线分平面最大区域数

f[n]:=f[n-1]+2*(n-1)
:=sqr(n)-n+2;

38 递推天地 8

-----凸多边形分三角形方法数

f[n]:=C(2*n-2,n-1) div n;

对于 k 边形

f[k]:=C(2*k-4,k-2) div (k-1); //(k>=3)

39 递推天地 9

-----Catalan 数列一般形式

1,1,2,5,14,42,132

f[n]:=C(2k,k) div (k+1);

40 递推天地 10

-----彩灯布置

排列组合中的环形染色问题

f[n]:=f[n-1]*(m-2)+f[n-2]*(m-1); (f[1]:=m; f[2]:=m(m-1);

41 线性动态规划 4

-----找数

线性扫描

sum:=f[i]+g[j];

(if sum=Aim then getout; if sum<Aim then inc(i) else inc(j);)

42 线性动态规划 5

-----隐形的翅膀

min:=min{abs(w[i]/w[j]-gold)};

if w[i]/w[j]<gold then inc(i) else inc(j);

43 剖分问题 5

-----最大奖励

f[i]:=max(f[i],f[j]+(sum[j]-sum[i])*i-t

44 最短路 1

-----Floyd

f[i,j]:=max(f[i,j],f[i,k]+f[k,j]);

ans[q[i,j,k]]:=ans[q[i,j,k]]+s[i,q[i,j,k]]*s[q[i,j,k],j]/s[i,j];

45 剖分问题 6

-----小 H 的小屋

F[l,m,n]:=f[l-x,m-1,n-k]+S(x,k);

46 计数问题 2

-----陨石的秘密（排列组合中的计数问题）

Ans[l1,l2,l3,D]:=f[l1+1,l2,l3,D+1]-f[l1+1,l2,l3,D];

F[l1,l2,l3,D]:=Sigma(f[o,p,q,d-1]*f[l1-o,l2-p,l3-q,d]);

47 线性动态规划

------合唱队形

两次 **F[i]:=max{f[j]+1}**＋枚举中央结点

48 资源问题

------明明的预算方案：加花的动态规划

f[i,j]:=max(f[i,j],f[l,j-v[i]-v[fb[i]]-v[fa[i]]]+v[i]*p[i]+v[fb[i]]*p[fb[i]]+v[fa[i]]*p[fa[i]]);

49 资源问题

-----化工场装箱员

$$f[n,i,j]:=\min\left\{\begin{array}{l} f[n+i,getA[n+1,n+i],j+getB[n+1,n+i]] \\ f[n+j,i+getA[n+1,n+j],getB[n+1,n+j]] \\ f[n+10-i-j,i+getA[n+1,n+10-i-j],j+getB[n+1,n+10-i-j]] \end{array}\right\}+1$$

50 树形动态规划

-----聚会的快乐

f[i,2]:=max(f[i,0],f[i,1]);

f[i,1]:=sigma(f[t[i]^.son,0]);

f[i,0]:=sigma(f[t[i]^.son,3]);

51 树形动态规划

-----皇宫看守

f[i,2]:=max(f[i,0],f[i,1]);

f[i,1]:=sigma(f[t[i]^.son,0]);

f[i,0]:=sigma(f[t[i]^.son,3]);

52 递推天地

-----盒子与球

f[i,1]:=1;

f[i,j]:=j*(f[i-1,j-1]+f[i-1,j]);

53 双重动态规划

-----有限的基因序列

f[i]:=min{f[j]+1}

g[c,i,j]:=(g[a,i,j] and g[b,i,j]) or (g[c,i,j])

54 最大子矩阵问题

-----居住空间

f[i,j,k]:=min(min(min(f[i-1,j,k],f[i,j-1,k]),
min(f[i,j,k-1],f[i-1,j-1,k])),
min(min(f[i-1,j,k-1],f[i,j-1,k-1]),
f[i-1,j-1,k-1]))+1;

55 线性动态规划

------日程安排

f[i]:=max{f[j]}+P[I]; (e[j]<s[i])

56 递推天地

------组合数

C[I,j]:=C[i-1,j]+C[I-1,j-1]

C[I,0]:=1

57 树形动态规划

-----有向树 k 中值问题

F[I,r,k]:=max{max{f[l[i],I,j]+f[r[i],I,k-j-1]},f[f[l[i],r,j]+f[r[i],r,k-j]+w[I,r]]}

58 树形动态规划

-----CTSC 2001 选课

F[I,j]:=w[i](if i ∈ P)+f[l[i],k]+f[r[i],m-k](0 ≤ k ≤ m)(if l[i]<>0)

59 线性动态规划

-----多重历史

f[i,j]:=sigma{f[i-k,j-1]}(if checked)

60 背包问题(+-1 背包问题+回溯)

-----CEOI1998 Substract

f[i,j]:=f[i-1,j-a[i]] or f[i-1,j+a[i]]

61 线性动态规划(字符串)

-----NOI 2000 古城之谜

f[i,1,1]:=min{f[i+length(s),2,1], f[i+length(s),1,1]+1}

f[i,1,2]:=min{f[i+length(s),1,2]+words[s],f[i+length(s),1,2]+words[s]}

62 线性动态规划

-----最少单词个数

f[i,j]:=max{f[I,j],f[u-1,j-1]+l}

63 线型动态规划

-----APIO2007 数据备份

状态压缩＋剪掉每个阶段 **j** 前 **j*2** 个状态和 **j*2+200** 后的状态贪心动态规划

f[i]:=min(g[i-2]+s[i],f[i-1]);

64 树形动态规划

-----APIO2007 风铃

f[i]:=f[l]+f[r]+{1 (if c[l]<c[r])}

g[i]:=1(d[l]<>d[r]) 0(d[l]=d[r])

g[l]=g[r]=1 then Halt;

65 地图动态规划

-----NOI 2005 adv19910

F[t,i,j]:=max{f[t-1,i-dx[d[[t]],j-dy[d[k]]]+1],f[t-1,i,j];

66 地图动态规划

-----优化的 **NOI 2005 adv19910**

F[k,i,j]:=max{f[k-1,i,p]+1} j-b[k]<=p<=j;

67 目标动态规划

-----CEOI98 subtra

F[I,j]:=f[I-1,j+a[i]] or f[i-1,j-a[i]]

68 目标动态规划

----- Vijos 1037 搭建双塔问题

F[value,delta]:=g[value+a[i],delta+a[i]] or g[value,delta-a[i]]

69 树形动态规划

-----有线电视网

f[i,p]:=max(f[i,p],f[i,p-q]+f[j,q]-map[i,j])

leaves[i]>=p>=l, 1<=q<=p;

70 地图动态规划

-----vijos 某题

F[I,j]:=min(f[i-1,j-1],f[I,j-1],f[i-1,j]);

71 最大子矩阵问题

-----最大字段和问题

f[i]:=max(f[i-1]+b[i],b[i]); f[1]:=b[1]

72 最大子矩阵问题

-----最大子立方体问题

枚举一组边 **i** 的起始，压缩进矩阵 **B[I,j]+=a[x,I,j]**

枚举另外一组边的其实，做最大子矩阵

73 括号序列

-----线型动态规划

f[I,j]:=min(f[I,j],f[i+1,j-1](s[i]s[j]=”()”or(”[]”)),

f[I+1,j+1]+1 (s[j]=”(”or”[”] ,

f[I,j-1]+1(s[j]=”)”or”]”)

74 棋盘切割

-----线型动态规划

f[k,x1,y1,x2,y2]=min{min{f[k-1,x1,y1,a,y2]+s[a+1,y1,x2, y2],

f[k-1,a+1,y1,x2,y2]+s[x1,y1,a, y2]

min{}}

75 概率动态规划

-----聪聪和可可**(NOI2005)**

x:=p[p[i,j],j]

f[I,j]:=(f[x,b[j,k]]+f[x,j])/(l[j]+1)+1

f[I,i]=0

f[x,j]=1

76 概率动态规划

-----血缘关系

F[A, B]=(f[A_0, B]+P[A_1, B])/2

f[I,i]=1

f[I,j]=0(I,j 无相同基因**)**

77 线性动态规划

-----决斗

F[I,j]=(f[I,j] and f[k,j]) and (e[I,k] or e[j,k]),i<k<j

78 线性动态规划

-----舞蹈家

F[x,y,k]=min(f[a[k],y,k+1]+w[x,a[k]],f[x,a[k],k+1]+w[y, a[k]])

79 线性动态规划

-----积木游戏

F[I,a,b,k]=max(f[I,a+1,b,k],f[i+1,a+1,a+1,k’],f[I,a+1,a+1, k’])

80 树形动态规划（双次记录）

-----NOI2003 逃学的小孩

朴素的话枚举节点 **i** 和离其最远的两个节点 **j,k O(n^2)**

每个节点记录最大的两个值，并记录这最大值分别是从哪个相邻节点传过来的。当遍历到某个孩子节点的时候，只需检查最大值是否是从该孩子节点传递来的。如果是，就取次大，否则取最大值

81 树形动态规划**(**完全二叉树**)**

-----NOI2006 网络收费

F[I,j,k]表示在点 **i** 所管辖的所有用户中，有 **j** 个用户为 **A**，在 **I** 的每个祖先 **u** 上，如果 **N[a]>N[b]**则标 **0** 否则标 **1**，用二进制状态压缩进 **k** 中，在这种情况下的最小花费

F[I,j,k]:=min{f[l,u,k and (s[i]<<(i-1))]+w1,f[r,j-u,k and(s[i]<<(i-1))]}

82 树形动态规划

-----IOI2005 河流

F[i]:=max

83 记忆化搜索

-----Vijos 某题,忘了

F[pre,h,m]:=sigma{SDP(I,h+1,M+i)} (pre<=i<=M+1)

84 状态压缩动态规划

-----APIO 2007 动物园

f[I,k]:=f[i-1,k and not (1<<4)] + NewAddVal

85 树形动态规划

-----访问术馆

f[i,j-c[i]×2]:= max (f[l[i],k], f[r[i],j-c[i]×2-k])

86 字符串动态规划

-----Ural 1002 Phone

if exist(copy(s,j,i-j)) then f[i]:=min(f[i],f[j]+1);

87 多进程动态规划

-----CEOI 2005 service

Min(f[i,j,k], f[i-1,j,k] + c[t[i-1],t[i]])

Min(f[i,t[i-1],k], f[i-1,j,k] + c[j,t[i]])

Min(f[i,j,t[i-1]], f[i-1,j,k] + c[k,t[i]])

88 多进程动态规划

-----Vijos1143 三取方格数

```
max(f[i,j,k,l],f[i-1,j-R[m,1],k-R[m,2],l-R[m,3]]);
if (j=k) and (k=l) then inc(f[i,j,k,l],a[j,i-j]) else
if (j=k) then inc(f[i,j,k,l],a[j,i-j]+a[l,i-l]) else
if (k=l) then inc(f[i,j,k,l],a[j,i-j]+a[k,i-k]) else
```

if (j=l) then inc(f[i,j,k,l],a[j,i-j]+a[k,i-k]) else inc(f[i,j,k,l],a[j,i-j]+a[k,i-k]+a[l,i-l]);

89 线型动态规划

-----IOI 2000 邮局问题

f[i,j]:=min(f[I,j],f[k,j-1]+d[k+1,i]);

90 线型动态规划

-----Vijos 1198 最佳课题选择

if j-k>=0 then Min(f[i,j],f[i-1,j-k]+time(i,k));

91 背包问题

----- USACO Raucous Rockers

多个背包，不可以重复放物品，但放物品的顺序有限制。

F[I,j,k]表示决策到第 **i** 个物品、第 **j** 个背包，此背包花费了 **k** 的空间。

f[I,j,k]:=max(f[I-1,j,k],f[I-1,j,k-t[i]]+p[i],f[i-1,j-1,maxtime-t[i]])

92 多进程动态规划

-----巡游加拿大（**IOI95**、**USACO**）

d[i,j]=max{d[k,j]+1(a[k,i] & j<k<i),d[j,k]+1(a[I,j] & (k<j))}。

f[i,j]表示从起点出发，一个人到达 **i**，另一个人到达 **j** 时经过的城市数。

d[i,j]=d[j,i]，所以我们限制 **i>j**

分析状态**(i,j)**，它可能是**(k,j)(j<k<i)**中 **k** 到达 **i** 得到（方式 **1**），也可能是**(j,k)(k<j)**中 **k** 超过 **j** 到达 **i** 得到(方式 **2**)。但它不能是**(i,k)(k<j)**中 **k** 到达 **j** 得到，因为这样可能会出现重复路径。即使不会出现重复路径，那么它由**(j,k)**通过方式 **2** 同样可以得到，所以不会遗漏解 时间复杂度 **$O(n^3)$**

93 动态规划

-----ZOJ cheese

f[i,j]:=f[i-kk*zl[u,1],j-kk*zl[u,2]]+a[i-kk*zl[u,1],j-kk*zl[u,2]]

94 动态规划

-----NOI 2004 berry 线性

F[I,1]:=s[i]

F[I,j]:=max{min{s[i]-s[l-1]},f[l-1,j-1]} (2≤j≤k, j≤l≤i)

95 动态规划

-----NOI 2004 berry 完全无向图

F[I,j]:=f[i-1,j] or (j≥w[i]) and (f[i-1,j-w[i]])

96 动态规划

-----石子合并 四边形不等式优化

m[i,j]=max{m[i+1,j], m[i,j-1]}+t[i,j]

97 动态规划

-----CEOI 2005 service

（**k≥long[i]**，**i≥1**）**g[i, j, k]=max{g[i-1,j,k-long[i]]+1**，**g[i-1,j,k]}**

（**k<long[i]**，**i≥1**） **g[i, j, k]=max{g[i-1,j-1,t-long[i]]+1**，**g[i-1,j,k]}**

(0≤j≤m， 0≤k<t) g[0,j,k]=0;

ans:=g[n,m,0]。

状态优化：**g[i, j]=min{g[i-1,j]**，**g[i-1,j-1]+long[i]}**

其中**(a, b)+long[i]=(a', b')**的计算方法为：

当 **b+long[i] ≤t** 时： **a'=a; b'=b+long[i];**

当 **b+long[i] >t** 时： **a'=a+1; b'=long[i];**

规划的边界条件：

当 **0≤i≤n** 时，**g[i,0]=(0,0)**

98 动态规划

-----AHOI 2006 宝库通道

f[k]:=max{f[k-1]+x[k,j]-x[k,i-1], x[k,j]-x[k,i-1]}

99 动态规划

-----Travel

A) 费用最少的旅行计划。

设 **f[i]**表示从起点到第 **i** 个旅店住宿一天的最小费用；**g[i]**表示从起点到第 **i** 个旅店住宿一天，在满足最小费用的前提下所需要的最少天数。那么：

f[i]=f[x]+v[i], g[i]=g[x]+1

x 满足：

1、 **x<i**，且 **d[i] – d[x] <= 800**（一天的最大行程）。

2、 对于所有的 **t < i, d[i] – d[t] <= 800**，都必须满足：

A. g[x] < g[t](f[x] = f[t]时**)** **B. f[x] < f[t] (**其他情况**)**

f[0] = 0，**g[0] = 0**。 **Ans:=f[n + 1]**，**g[n+1]**。

B). 天数最少的旅行计划。

方法其实和第一问十分类似。

设 **g'[i]**表示从起点到第 **i** 个旅店住宿一天的最少天数；**f'[i]**表示从起点到第 **i** 个旅店住宿一天，在满足最小天数前提下所需要的最少费用。那么：

g'[i] = g'[x] + 1, f'[i] = f'[x] + v[i]

x 满足：

1、 **x<i**，且 **d[i] – d[x] <= 800**（一天的最大行程）。

2、 对于所有的 **t < i, d[i] – d[t] <= 800**，都必须满足：

f'[x] < f'[t]　　g'[x] = g'[t]时
g'[x] < g'[t]　　其他情况
f'[0] = 0，g'[0] = 0。 Ans:=f'[n + 1]，g'[n+1]。

100 动态规划
-----NOI 2007 cash
y:=f[j]/(a[j]*c[j]+b[j]);
g:=c[j]*y*a[i]+y*b[i];
f[i]:=max(f[i],g)