动态规划入门1

赵宗昌

2018.5.13

每次遇到新题自己怎么也想不出来，
但是一看题解就明白！原来这么简单！

动态规划的基本概念

- 动态规划（Dynamic Programming 简称DP）。
- 解决“多阶段决策问题”的一种高效算法。
- 通过合理组合子问题的解从而解决整个问题解的一种算法。其中的子问题并不是独立的，这些子问题又包含有公共的子子问题。……
- 动态规划算法就是对每个子问题只求一次，并将其结果保存在一张表中(数组)，以后再用到时直接从表中拿过来使用，避免重复计算相同的子问题。
- “不做无用功”的求解模式，大大提高了程序的效率。
- 动态规划算法常用于解决统计类问题（统计方案总数）和最优值问题（最大值或最小值），尤其普遍用于最优化问题。

动态规划的术语：

1、 阶段：

把所给求解问题的过程恰当地分成若干个相互联系的阶段，以便于按一定的次序去求解，过程不同，阶段数就可能不同．描述阶段的变量称为阶段变量。在多数情况下，阶段变量是离散的，用k表示。

阶段的划分一般根据时间和空间来划分的。

2、状态：

某一阶段的出发位置成为状态，通常一个阶段有多个状态。

状态通常可以用一个或一组数来描述，称为状态变量。

3、决策：

一个阶段的状态给定以后，从该状态演变到下一阶段某个状态的一种选择（行动）称为决策。描述决策的变量称决策变量

4、策略和最优策略

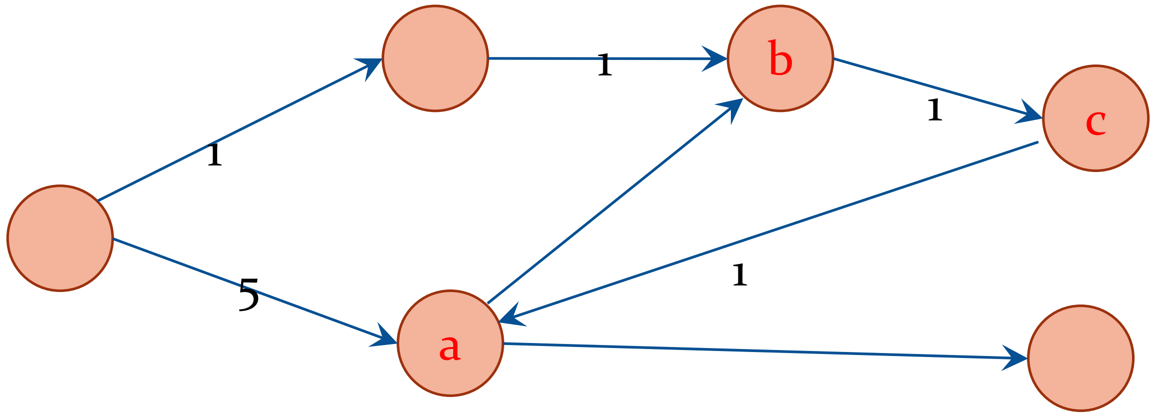
所有阶段的决策有序组合构成一个策略。

最优效果的策略叫最优策略。

动态规划的条件：

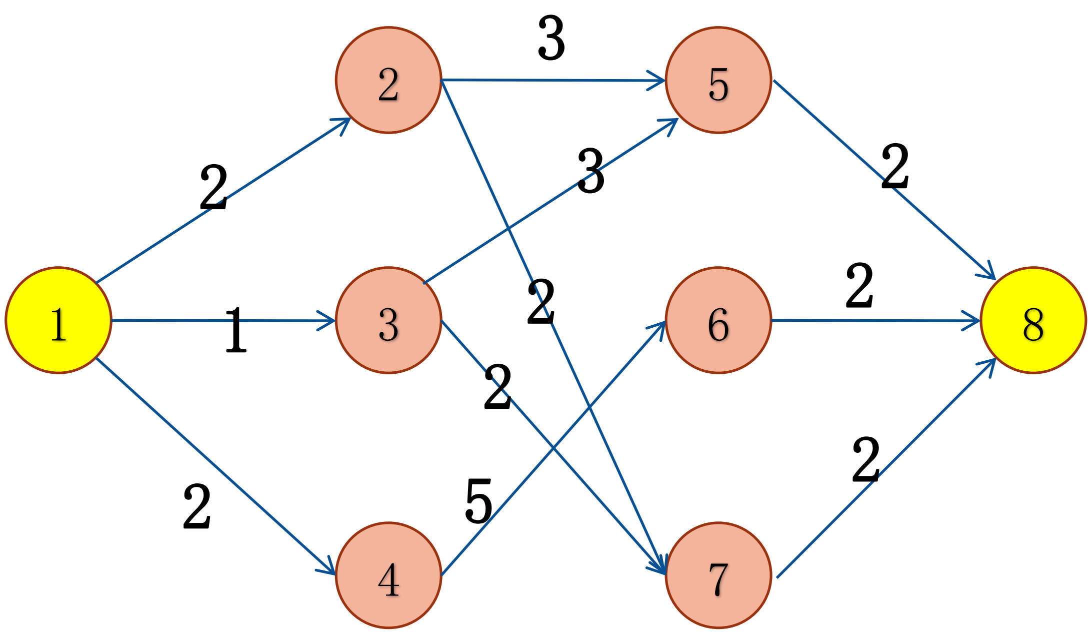
- 拓扑图（有向无环图）
- 无后效性 最优子结构

- 非拓扑图（可能有环）
- 有后效性　a→b→c？ b→c→a?



求1到8的最短距离?

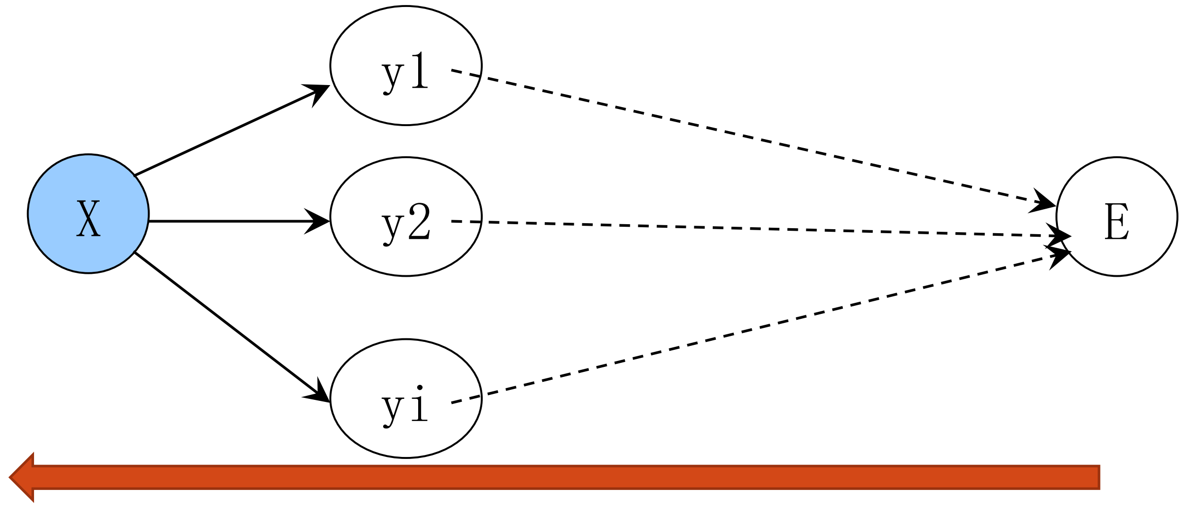
一本通1261类似的题目



- 拓扑图（有向无环图）
- 无后效性 最优子结构

倒推：

$$f_k[x]=\min\{f_{k+1}[y_i]+d[x,y_i]\}$$



倒推格式为：

$f[U_n]$=初始值；

for k←n-1 downto 1 do {枚举阶段}

for U取遍所有状态 do {枚举状态}

for X取遍所有决策 do {枚举决策}

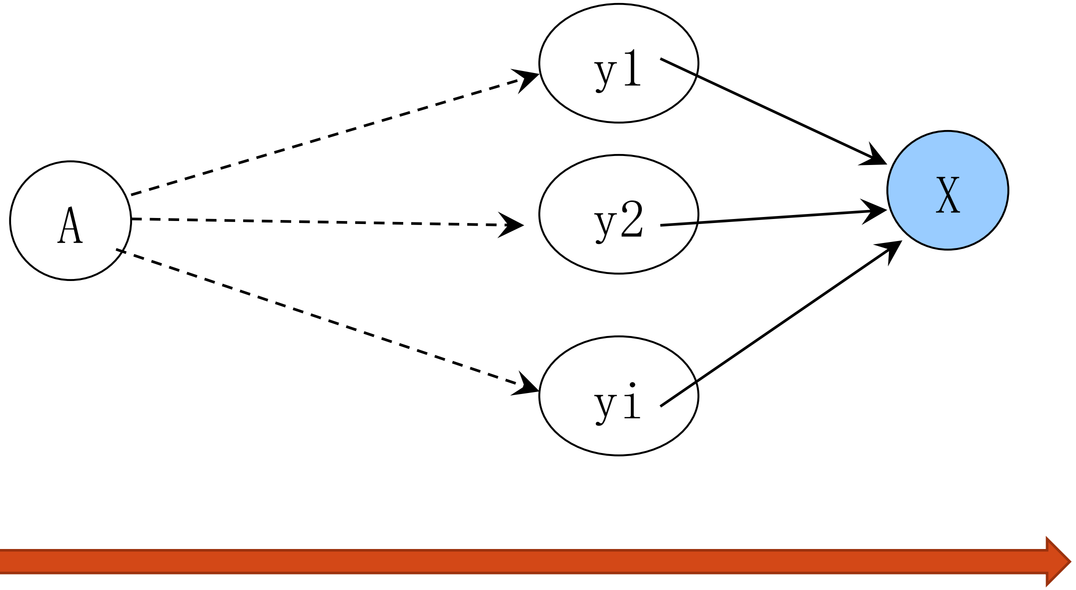
$f[U_k]=opt\{f[U_{k+1}]+L[U_k,X_k]\}$;

//$L[U_k, X_k]$}：状态U_k通过策略X_k到达状态U_{k+1}的费用输出：

$f[U_1]$:目标

顺推：

$$f_k[x]=\min\{f_{k-1}[y_i]+d\,[y_i,x]\}$$



顺推格式为：

$f[U_1]$=初始值；

for k←2 to n do   {枚举每一个阶段}

  for U取遍所有状态 do

    for X取遍所有决策 do

$f[U_k]=opt\{f[U_{k-1}]+L[U_{k-1},X_{k-1}]\}$;

//$L[U_{k-1},X_{k-1}]$}：状态U_{k-1}通过策略X_{k-1}到达状态U_k 的费用

输出：$f[U_n]$:目标

动态规划的关键:

状态转移方程的构造是动态规划过程中最重要的一步,也是最难的一步.对于大多数的动态规划,寻找状态转移方程有一条十分高效的通道,就是寻找变化中的不变量（已经求得的值）.

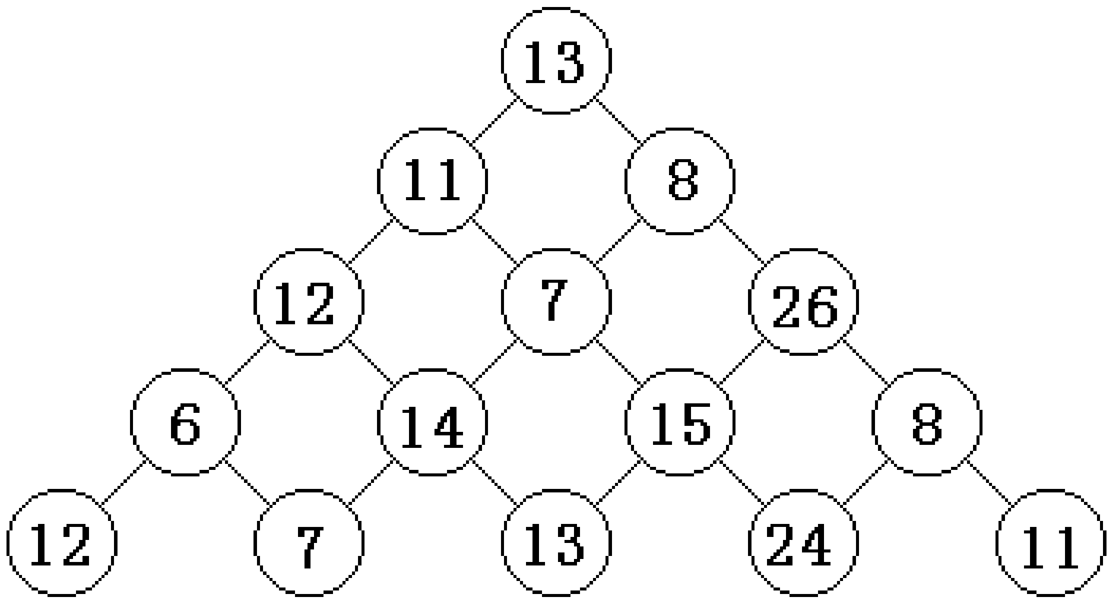
定量处理的过程也就是决策实施的过程.

题目：

1. 求最优值问题

2. 统计问题

例1 数字金字塔（一本通1258）



方法1：递归（搜索）

- 定义：函数dfs(x,y)从(x,y)走到最后一行(n,i)得到的最大值。
- d(x,y)=max(dfs(x+1,y),dfs(x+1,y+1))+a[x][y]
- 边界：x=n
- 起点dfs(1,1);

```
int dfs(int x,int y){
    //从(x,y)走到最后一行(n,i)得到的最大值
    if(x==n)return a[x][y];
    return max(dfs(x+1,y),dfs(x+1,y+1))+a[x][y];
}
```

方法2:分析太慢的原因？

- 重复计算了很多的dfs(x,y);
- 改进：
- f[x][y]:记录dfs(x,y),因为(x,y)走到最后一行的最大值是唯一的，第一次走时记录下最优值，以后再用到时直接用f[x][y]即可，不需要再递归求解。

```
int dfs(int x,int y){
     if(f[x][y]>0)return f[x][y];
     if(x==n)return f[x][y]=a[x][y];
     return f[x][y]=max(dfs(x+1,y),dfs(x+1,y+1))+a[x][y];
}
```

记忆化搜索

dfs(1,1):往下走，向上返回：真正计算是倒着从下向上计算的。依次计算第n,n-1,n-2,...,1行。

方法3：直接倒着从第n行开始向上推（递归的回退过程）：

f[i][j]:从(i,j)走到最后一行的最大值。

目标：f[1][1]

初始：f[n][i]=a[n][i]

```
for(int i=1;i<=n;i++)f[n][i]=a[n][i];
for(int i=n-1;i>0;i--)
    for(int j=1;j<=i;j++)
        f[i][j]=max(f[i+1][j],f[i+1][j+1])+a[i][j];
cout<<f[1][1]<<endl;
```

能否正向求？怎么定义？

方法4：正推（从第一行到最后一行）

- f[i][j]:从(1,1)走到(i,j)走到最后一行的最大值。
- 目标：max(f[n][i])
  初始：f[1][1]=a[1][1]

```
f[1][1]=a[1][1];
for(int i=2;i<=n;i++)
    for(int j=1;j<=i;j++)
        f[i][j]=max(f[i-1][j-1],f[i-1][j])+a[i][j];
int ans=0;
for(int i=1;i<=n;i++)ans=max(ans,f[n][i]);
cout<<ans<<endl;
```

DP常见模型

动态规划有多种多样的题目，但通常按照状态可分为以下几类：

- 坐标型
- 线性型
- 区间型
- 背包型
- 树型

一、坐标型

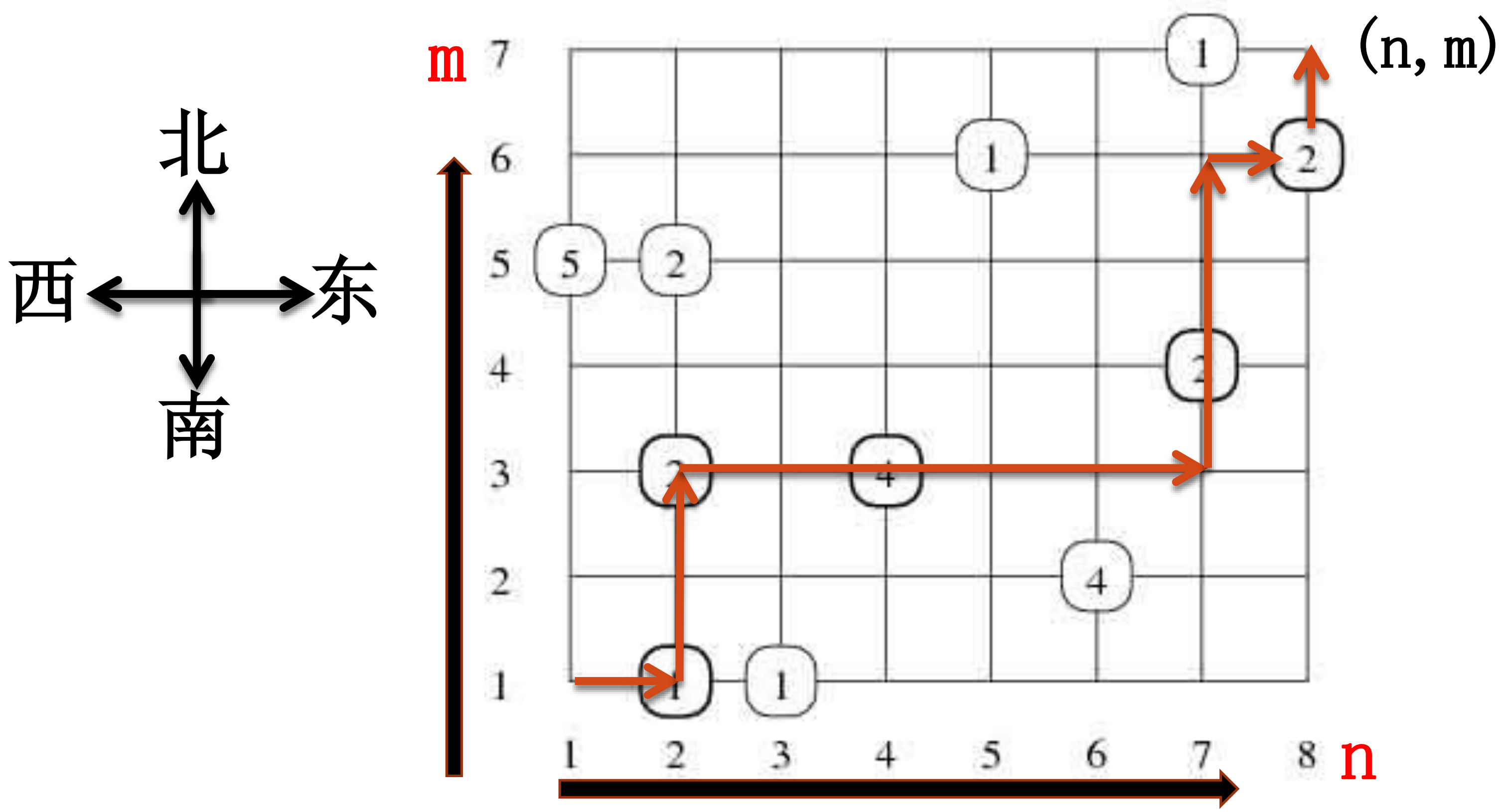
在二维坐标系内，规定了方向，求最优值问题。
比较容易根据方向写出动态规划方程：
一般方程也是二维的f[i][j]

例2：公共汽车

【问题描述】

一个城市的道路，南北向的路有n条，并由西向东从1标记到n，东西向的路有m条，并从南向北从1标记到m，每一个交叉点代表一个路口，有的路口有正在等车的乘客。一辆公共汽车将从(1,1)点驶到（n,m）点，车只能向东或者向北开.

问：司机怎么走能接到最多的乘客。

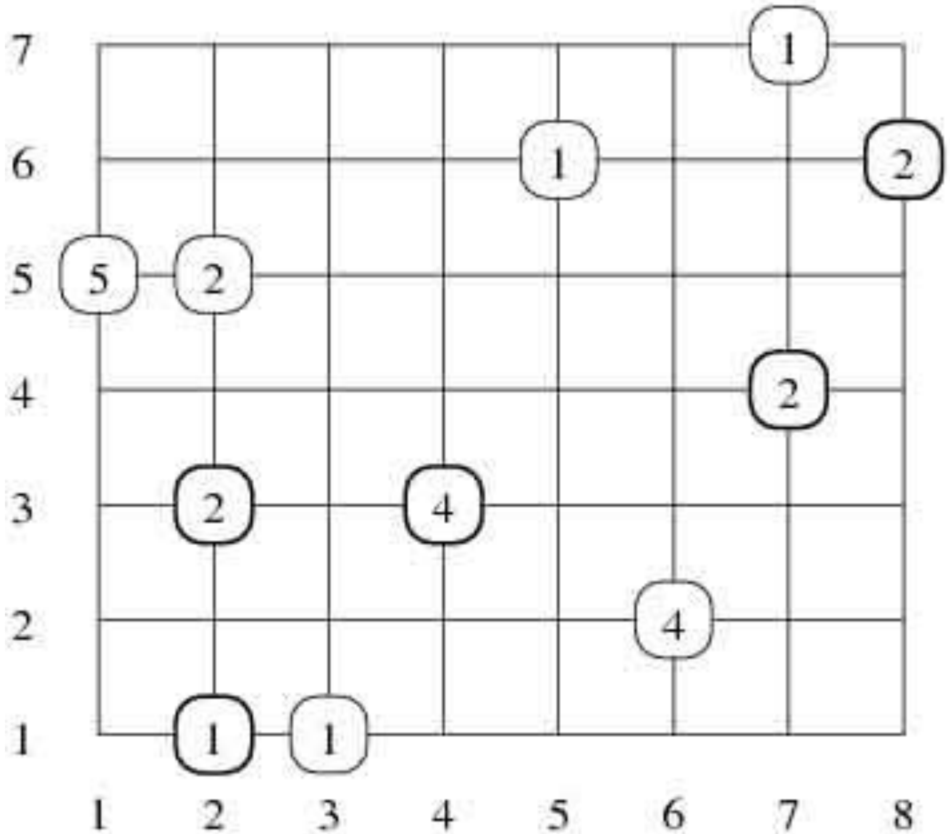


- 【输入】
- 第一行是n,m,和k,其中k是有乘客的路口的个数。以下k行是有乘客的路口的坐标和乘客的数量。已知每个路口的乘客数量不超过1000000。n,m<=1000.
- 【输出】
- 接到的最多的乘客数。

bus.in	bus.out
8 7 11 4 3 4 6 2 4 2 3 2 5 6 1 2 5 2 1 5 5 2 1 1 3 1 1 7 7 1 7 4 2 8 6 2	11

- a[i,j]（i,j)位置的人数，
- f[i,j]:从（1,1）走到（i，j）能接的最多人数。

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



- 训练：一本通：
- 1284
- 1287

二. 线性模型：

LIS（Longest Increasing Subsequence）最长上升子序列：

给定n个元素的数列，求最长的上升子序列长度(LIS)。

例题：一本通：1281

一个数的序列bi，当b1<b2<...<bSb1<b2<...<bS的时候，我们称这个序列是上升的。对于给定的一个序列(a1, a2, ..., aN)，我们可以得到一些上升的子序列(ai1, ai2, ..., aiK)，这里1≤i1<i2<...<iK≤N。
比如，对于序列(1, 7, 3, 5, 9, 4, 8)，有它的一些上升子序列，如(1, 7), (3, 4, 8)等等。这些子序列中最长的长度是4，比如子序列(1, 3, 5, 8)。

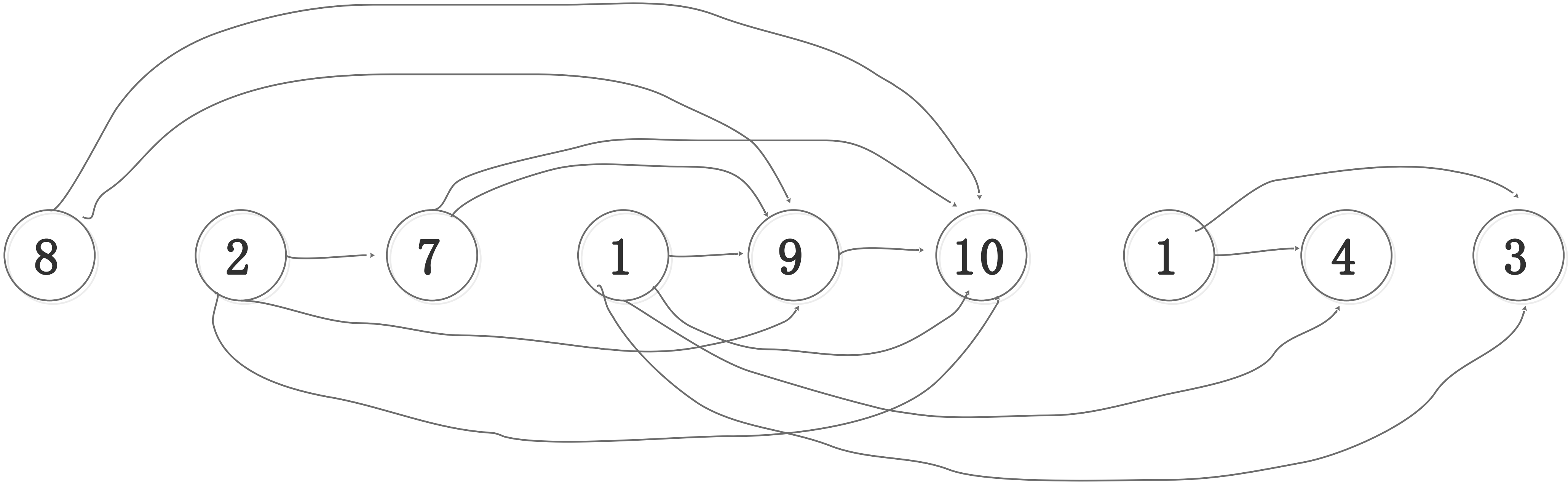
你的任务，就是对于给定的序列，求出最长上升子序列的长度。

最长上升子序列长度（LIS）：

8 2 7 1 9 10 1 4 3

- 找出以每个元素为**起点**的所有的上升子序列：
- 8 9 10
- 2 7 9 10
- 7 9 10
- 1 9 10
- 9 10
- 1 4
- 4
- 3

每个数向后面比他大的点建立有向边；
求最长路（顶点数最多）



f[i]=max(f[j])+1 (i<j<=n&&a[i]<a[j])

方法1：暴力搜索

8 2 7 1 9 10 1 4 3

找出以每个元素为**<u>起点</u>**的所有的上升子序列；
然后选择最长的即可。

① <u>8</u> 9 10
② <u>2</u> 7 9 10
③ <u>7</u> 9 10
④ <u>1</u> 9 10
⑤ <u>9</u> 10
⑥ <u>10</u>
⑦ <u>1</u> 4
⑧ <u>4</u>
⑨ <u>3</u>

怎么找出这些序列？

```
int dfs(int i){
    //以a[i]为开头的最长递增子序列长度
    int s=0;
    for(int j=i+1;j<=n;j++)
        if(a[i]<a[j])s=max(s,dfs(j));
    s++;
    return s;
}

ans=0;
for(int i=1;i<=n;i++)
    ans=max(ans,dfs(i));
cout<<ans<<endl;
```

过了几个点？
为什么超时？
怎样在此基础上改进这个保留搜索？

方法2：记忆化搜索

- 以每个元素为起点的LIS是固定不变的，每次求完可以记录下来，供后面直接使用，避免重复搜索。
- f[i]:以a[i]开始的最长上升子序列长度（初始为0），一旦求过f[i],一定是f[i]>=1,起码有a[i].

```
int dfs(int i){
	//以a[i]开始的最长序列长度
	if(f[i]>0)return f[i];
	f[i]=0;
	for(int j=i+1;j<=n;j++)
		if(a[i]<a[j])f[i]=max(f[i],dfs(j));
	f[i]++;
	return f[i];
}
```

方法3:倒序递推求f[i]

以a[i]为**<u>起点</u>**元素的最长上升子序列长度

每个数向后面比他大的点建立有向边;

求最长路（顶点数最多）

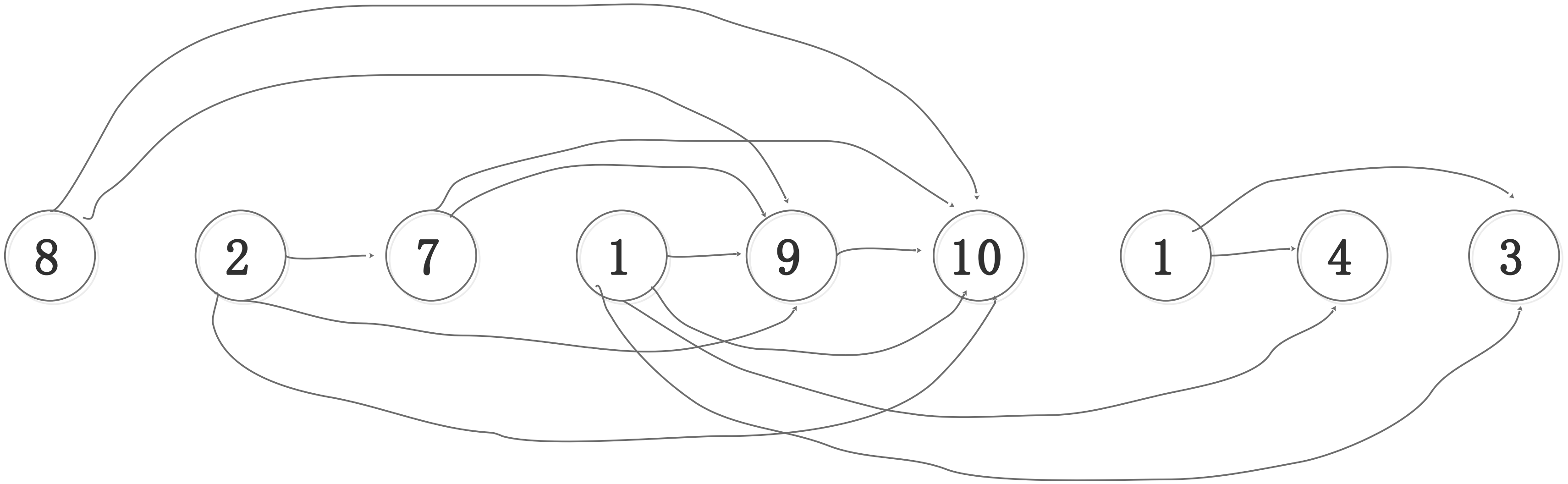
观察：边的顺序：a[i]向后的边i+1,i+1,..,n中选择。

可以直接倒序求即可。

f[n]=1;

f[i]=max(f[j])+1 (i<j<=n&&a[i]<a[j])

ans=max(f[i]);



倒推求f[i]:

以a[i]为**<u>起点</u>**元素的最长上升子序列长度

```
f[n]=1;
for(int i=n-1;i>=1;i--){
    f[i]=0;
    for(int j=i+1;j<=n;j++)
        if(a[i]<a[j])f[i]=max(f[i],f[j]);
    f[i]++;
}
```

方法4：正向递推：

- f[i]:以a[i]为<u>**终点**</u>元素的最长子序列长度。

8 2 7 1 9 10 1 4 3

① <u>8</u>

② <u>2</u>

③ 2 <u>7</u>

④ <u>1</u>

⑤ 2 7 <u>9</u>

⑥ 2 7 9 <u>10</u>

⑦ <u>1</u>

⑧ 1 <u>4</u>

⑨ 1 <u>3</u>

正推求f[i]：

以a[i]为**<u>终点</u>**元素的最长子序列长度。

方程：

f[1]=1;

f[i]=max(f[j])+1 (1<=j<i&&a[j]<a[i])

ans=max(f[i]);

正推：

```
f[1]=1;
for(int i=2;i<=n;i++){
        f[i]=0;
        for(int j=1;j<i;j++)
                if(a[j]<a[i])f[i]=max(f[i],f[j]);
        f[i]++;
}
```

输出最优方案：

- 一本通1259.
- 求最长不下降序列长度及输出改序列。

【输入样例】
14
13 7 9 16 38 24 37 18 44 19 21 22 63 15

【输出样例】
max=8
7 9 16 18 19 21 22 63

- 正推:
- 正推求f[i]:
- 以a[i]为**终点**元素的最长子序列长度。
- 方程:
- f[1]=1;
- f[i]=max(f[j])+1 (1<=j<i&&a[j]<a[i])
- ans=max(f[i]);

- p[i]记录f[i]去最优值时的j。
- 找到最大的f[i],然后向**前**找即可。

```
f[1]=1;
p[1]=0;
for(int i=2;i<=n;i++){
     f[i]=0;
     for(int j=1;j<i;j++)
         if(a[j]<=a[i]&&f[j]>f[i]){
             f[i]=f[j];
             p[i]=j;
         }
     f[i]++;
}
int ans=f[1],k=1;
for(int i=2;i<=n;i++)
     if(f[i]>ans)ans=f[k=i];
cout<<"max="<<ans<<endl;
dfs(k);
```

```
void dfs(int i){
	if(p[i]>0)dfs(p[i]);
	cout<<a[i]<<" ";
}
```

倒推：

倒序递推求f[i]

以a[i]为**<u>起点</u>**元素的最长上升子序列长度

每个数向后面比他大的点建立有向边；

求最长路（顶点数最多）

观察：边的顺序：a[i]向后的边i+1,i+1,...,n中选择。

可以直接倒序求即可。

f[n]=1;

f[i]=max(f[j])+1 (i<j<=n&&a[i]<a[j])

ans=max(f[i]);

p[i]记录f[i]去最优值时的j。

找到最大的f[i],然后向后找即可。

```
f[n]=1;
p[n]=0;
for(int i=n-1;i>0;i--){
    f[i]=0;
    for(int j=i+1;j<=n;j++)
        if(a[i]<=a[j]&&f[j]>f[i]){
            f[i]=f[j];
            p[i]=j;
        }
    f[i]++;
}
int ans=f[1],k=1;
for(int i=2;i<=n;i++)
    if(f[i]>ans)ans=f[k=i];
cout<<"max="<<ans<<endl;
while(k>0){cout<<a[k]<<" ";k=p[k];}
```

知识扩展：

- 最长上升子序列长度； <
- 最长不下降子序列长度； <=
- 最长下降子序列长度； >
- 最长不上升子序列长度。 >=
- 应用广泛！

课后训练：

① 1264 【例9.8】合唱队形

② 1283 登山

③ 1286 怪盗基德的滑翔翼

④ 1263 【例9.7】友好城市

⑤ 1260 拦截导弹NOIP999

合唱队形 [NOIP 2004]

- 【问题描述】
- N位同学站成一排，音乐老师要请其中的(N-K)位同学出列，使得剩下的K位同学排成合唱队形。
- 合唱队形是指这样的一种队形：设K 位同学从左到右依次编号为1，2，……，K，他们的身高分别为$T_1,T_2,……,Tk$,则他们的身高满足$T_1<T_2<……T_i$，$T_i>T_{i+1}>……>T_k$（$1<=i<=K$）。
- 你的任务是：
- 已知有N位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

【输入】

第一行是一个整数N(2<=N<=1000),表示同学的总数。

第二行有n个整数，用空格分隔，第i个整数Ti(130<=Ti<=230)是第i位同学的身高（厘米）。

【输出】

一个整数，表示最少需要几位同学出列。

【数据规模】

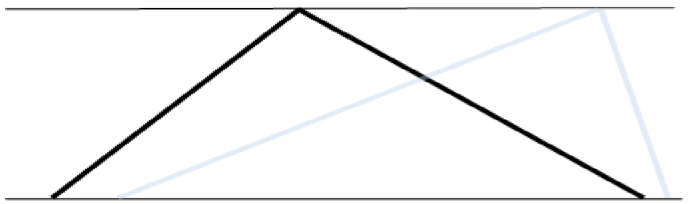
对与全部的数据，保证有n<=1000。

- 【样例输入输出】

chorus.in	chorus.out
8 186 186 150 200 160 130 197 220	4

问题分析：

- 计算最少需要几位同学出列，可转化为计算最多能留下多少同学。
- 将所有合唱队形同学排为一列，在二维坐标系中根据身高描点连线，图样就像一座山峰。



- 队列左边的身高依次上升，右边依次下降，中间有一个最高点，而这个最高点的左侧是上升子序列，右侧是下降子序列。人数最多是最优合唱队形。

算法描述：

- 对整个身高序列a[i]做：
- 一次最长上升子序列(f[i])：正推求；
- 一次最长下降子序列(g[i])：倒推求；
- 之后枚举最高点a[i]:

  ans=max(f[i]+g[i]-1)是保留最大值。
- 最少出队列人数:n-ans.

LIS简单变形1：1285 最大上升子序列和

- 把加1变为加a[i]即可。
- LIS：
- f[n]=1;
- f[i]=max(f[j])+1 (i<j<=n&&a[i]<a[j])
- ans=max(f[i]);
- 变为：
- f[n]=a[n];
- f[i]=max(f[j])+a[i] (i<j<=n&&a[i]<a[j])
- ans=max(f[i]);

训练：

- 1262 【例9.6】挖地雷

LIS简单变形2：最大连续子序列的和

- 求给定序列的最大连续子序列和。
- 输入：第一行：n（N<100000）
- 第二行：n个整数（-3000，3000）。
- 输出：最大连续子序列的和。
- 样例：
- 输入：
- 7
- -6 4 -1 3 2 -3 2
- 输出：
- 8

分析：

-6 4 -1 3 2 -3 2

1、以a[i]为结束点和以a[i-1]为结束点的最大连续子序列和有没有联系？
有什么样的联系？

2、如果事先已经求得了以a[i-1]为结束点的最大连续子序列和为x，那么怎样求以a[i]为结束点的最大连续子序列？

顺推法：

a[i]:存储序列；
f[i]：以**a[i]**为终点（连续区间的右边界）的子序列的最大和。

$$f[i]=\max\{f[i-1]+a[i],a[i]\}$$
$$=\max\{f[i-1],0\}+a[i]$$

初始：$f[1]=a[1]$
目标：$\max\{f[i]\}$ $(1<=i<=n)$

时间：**O(n)**

倒推法：

-6 4 -1 3 2 -3 2

a[i]:存储序列；

f[i]：从第i项开始(以第i项为第1项)的最大连续子序列的和。

f[i]=max{f[i+1]+a[i],a[i]}

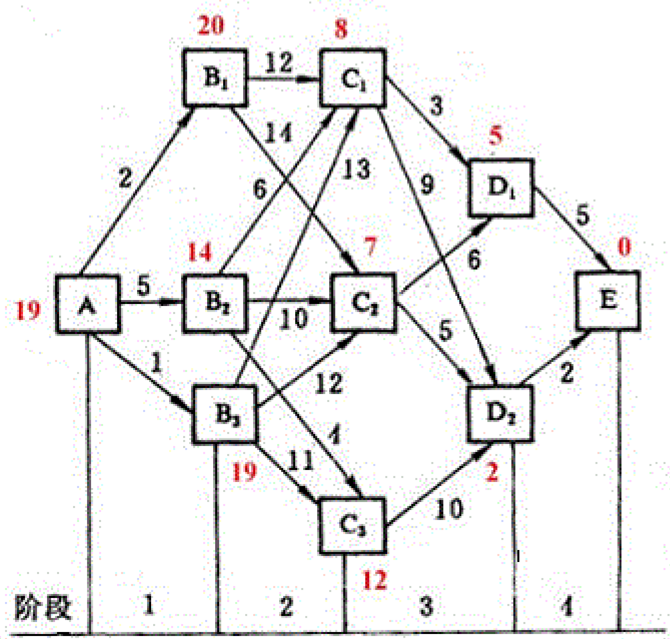
初始：f[n]=a[n]

目标：max{f[i]}　　　　1<=i<=n

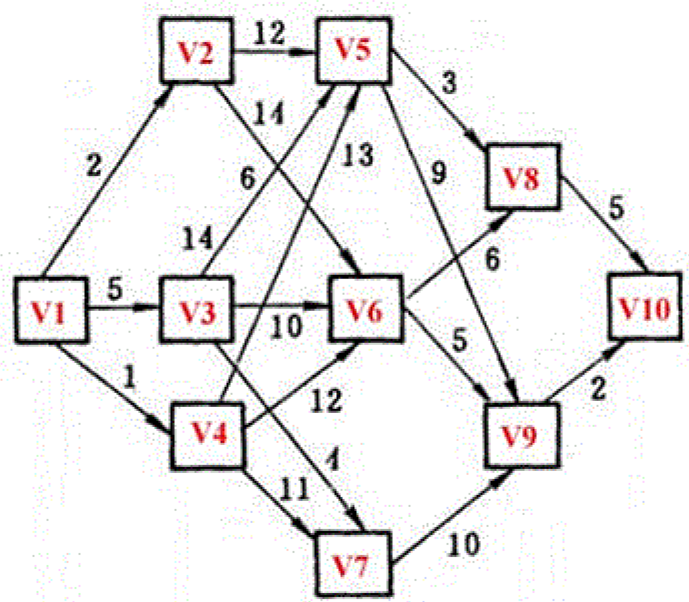
阶段状态决策演示：1261 城市交通路网（）

下图表示城市之间的交通路网，线段上的数字表示费用，单向通行由A->E。
试用动态规划的最优化原理求出A->E的最省费用。

交通图 1



交通图 2



```cpp
cin>>n;
for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)cin>>g[i][j];
f[n]=0;
p[n]=0;
for(int i=n-1;i>0;i--){
    f[i]=0x7fffffff;
    for(int j=i+1;j<=n;j++)
        if(g[i][j]>0&&g[i][j]+f[j]<f[i]){
            f[i]=min(f[i],f[j]+g[i][j]);
            p[i]=j;
        }
}
cout<<"minlong="<<f[1]<<endl;
cout<<1;
int k=p[1];
while(k>0){cout<<" "<<k;k=p[k];}
```