

# 动态规划

河南师大附中双语国际学校 | 谢志佳

**2019年6月4日**



# 动态规划入门

简单引例

初学者的困惑：

每次遇到新题自己怎么也想不出来！

看一眼题解与方程，原来这么简单！

我怎么没想到！

第二部分 基础算法 --> 第九章 动态规划

题号	题目名称	通过数	提交数	题号	题目名称	通过数	提交数
第一节 动态规划的基本模型				1273	【例9.17】货币系统	468	1159
1258	【例9.2】数字金字塔	1498	2627	1290	采药	742	1169
1259	【例9.3】求最长不下降序列	971	3103	1291	数字组合	483	847
1260	【例9.4】拦截导弹(Noip1999)	648	1863	1292	宠物小精灵之收服	350	573
1261	【例9.5】城市交通路网	524	706	1293	买书	418	649
1262	【例9.6】挖地雷	543	1118	1294	Charm Bracelet	384	686
1263	【例9.7】友好城市	546	1023	1295	装箱问题	504	914
1264	【例9.8】合唱队形	619	1422	1296	开餐馆	528	1018
1265	【例9.9】最长公共子序列	620	1191	第三节 动态规划经典题			
1266	【例9.10】机器分配	376	781	1274	【例9.18】合并石子	588	908
1281	最长上升子序列	882	1745	1275	【例9.19】乘积最大	408	665
1282	最大子矩阵	369	568	1276	【例9.20】编辑距离	287	885
1283	登山	438	1272	1277	【例9.21】方格取数	402	568
1284	摘花生	475	745	1278	【例9.22】复制书稿(book)	242	469
1285	最大上升子序列和	460	957	1279	【例9.23】橱窗布置(flower)	239	576
1286	怪盗基德的滑翔翼	452	946	1280	【例9.24】滑雪	307	616
1287	最低通行费	424	568	1297	公共子序列	263	466
1288	三角形最佳路径问题	470	521	1298	计算字符串距离	207	426
1289	拦截导弹	490	776	1299	糖果	244	548
第二节 背包问题				1300	鸡蛋的硬度	204	307
1267	【例9.11】01背包问题	1548	2332	1301	大盗阿福	252	453
1268	【例9.12】完全背包问题	1123	2061	1302	股票买卖	162	310
1269	【例9.13】庆功会	927	1571	1303	鸣人的影分身	238	348
1270	【例9.14】混合背包	628	1139	1304	数的划分	183	438
1271	【例9.15】潜水员	499	831	1305	Maximum sum	164	383
1272	【例9.16】分组背包	382	601	1306	最长公共子上升序列	226	378

# 引例1：斐波那契数列 FIBONACCI

斐波那契数列：

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

通项公式：

$$f(1)=1$$

$$f(2)=1$$

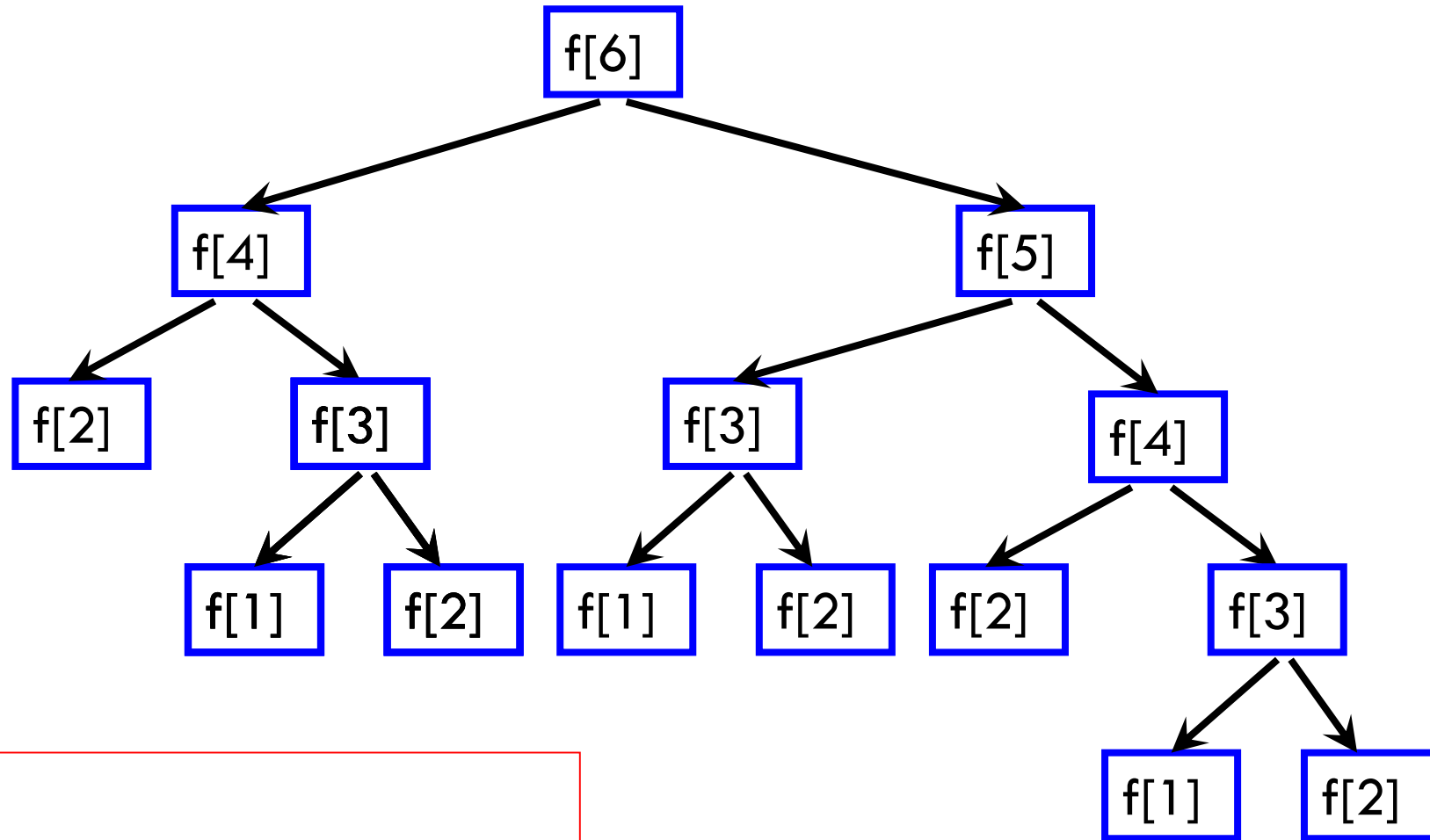
$$f(n)=f(n-2)+f(n-1) \quad (n>2)$$

输入：n ( $n \leq 1000$ )

输出：f(n)%10000

## 方法1:

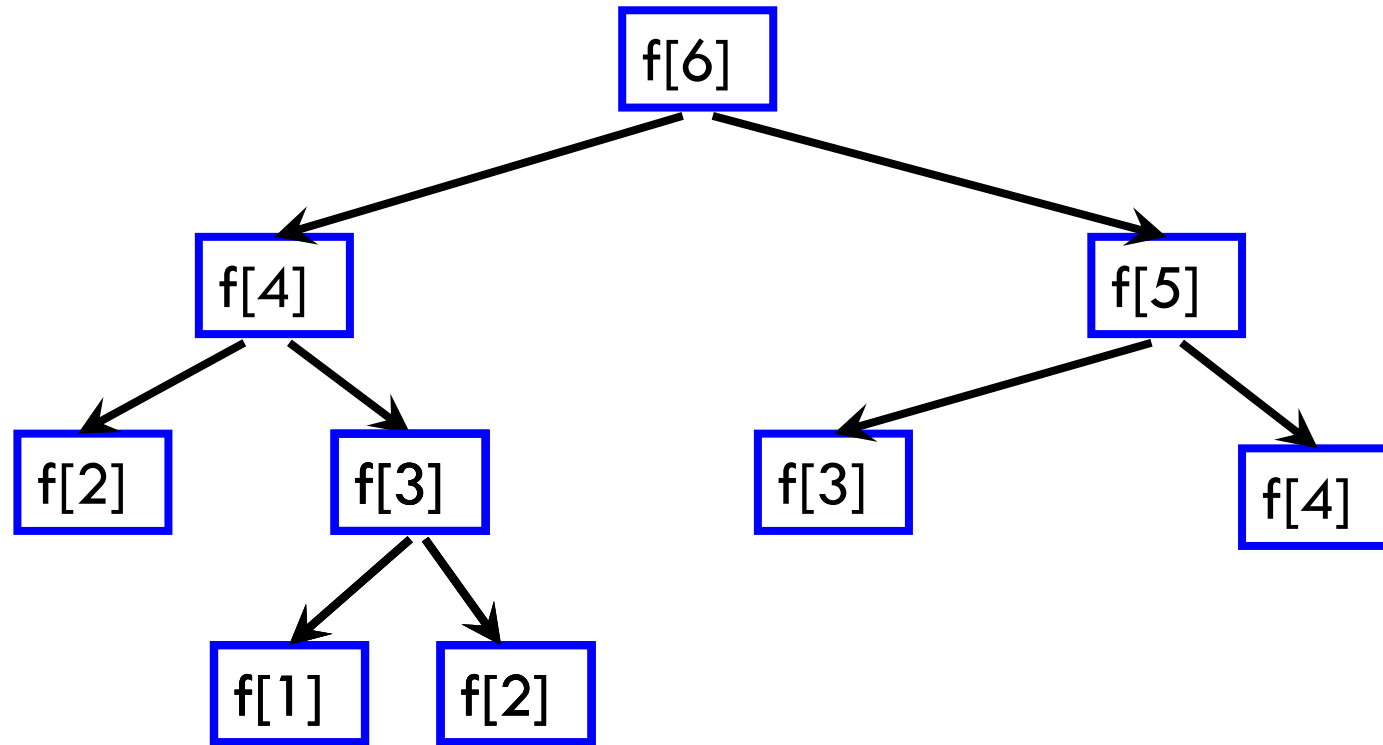
```
1  #include<cstdio>
2  #include<iostream>
3  using namespace std;
4  int n;
5  int dp(int i) {
6      if(i==1 || i==2) return 1;
7      return (dp(i-2)+dp(i-1))%10000;
8  }
9  int main() {
10     cin>>n;
11     cout<<dp(n)<<endl;
12     return 0;
13 }
```



$$f(1)=1$$

$$f(2)=1$$

$$f(n)=f(n-2)+f(n-1)$$

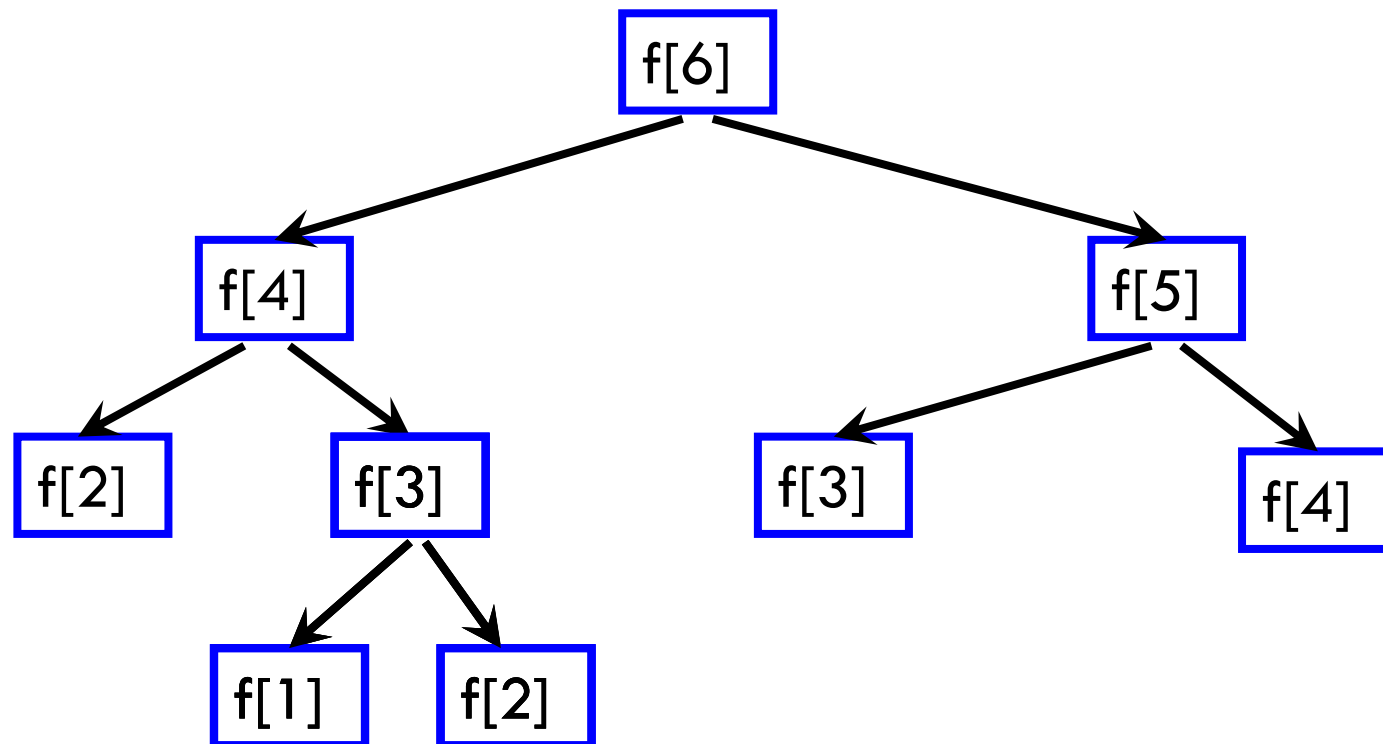


$$f(1)=1$$

$$f(2)=1$$

$$f(n)=f(n-2)+f(n-1)$$





$f(1)=1$   
 $f(2)=1$   
 $f(n)=f(n-2)+f(n-1)$

去掉重复的计算  
直接使用前面已经计算过的

# 改进1: 记忆化搜索

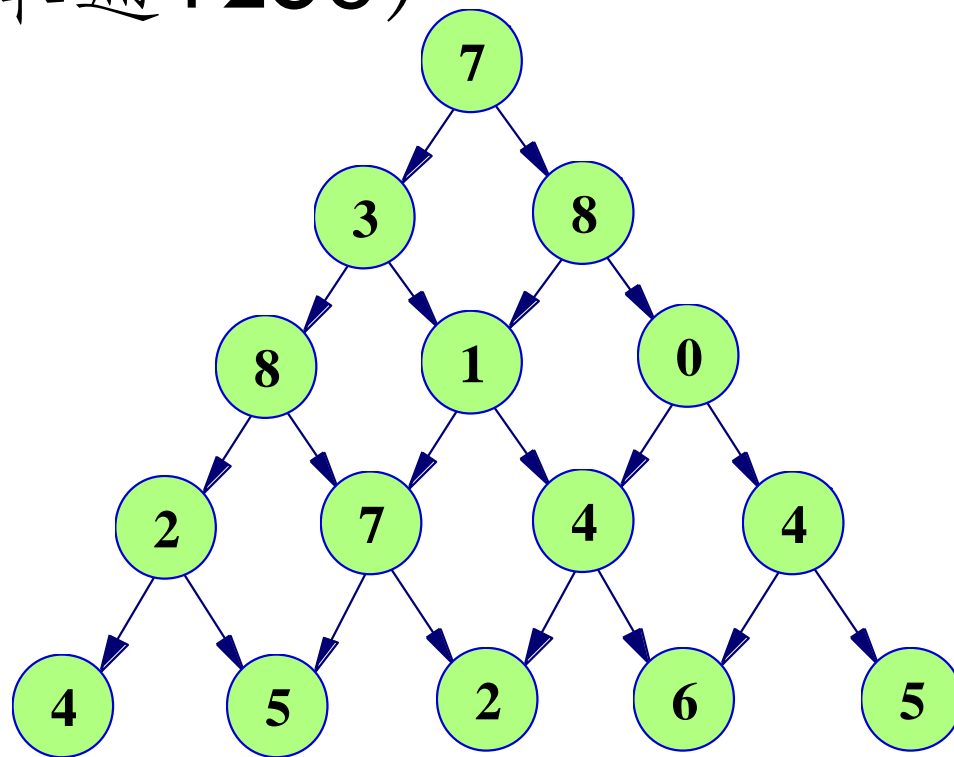
```
1  #include<cstdio>
2  #include<iostream>
3  using namespace std;
4  int n;
5  int f[1001];
6  int dp(int i){
7      if(f[i]>0) return f[i];
8      if(i==1||i==2) return 1;
9      return f[i]=(dp(i-2)+dp(i-1))%10000;
10 }
11 int main(){
12     cin>>n;
13     cout<<dp(n)<<endl;
14     return 0;
15 }
```

## 改进2：直接递推

```
1  #include<cstdio>
2  #include<iostream>
3  using namespace std;
4  int n;
5  int f[1001];
6  int main() {
7      cin>>n;
8      f[1]=f[2]=1;
9      for(int i=3;i<=n;i++)
10         f[i]=(f[i-2]+f[i-1])%10000;
11     cout<<f[n]<<endl;
12     return 0;
13 }
```

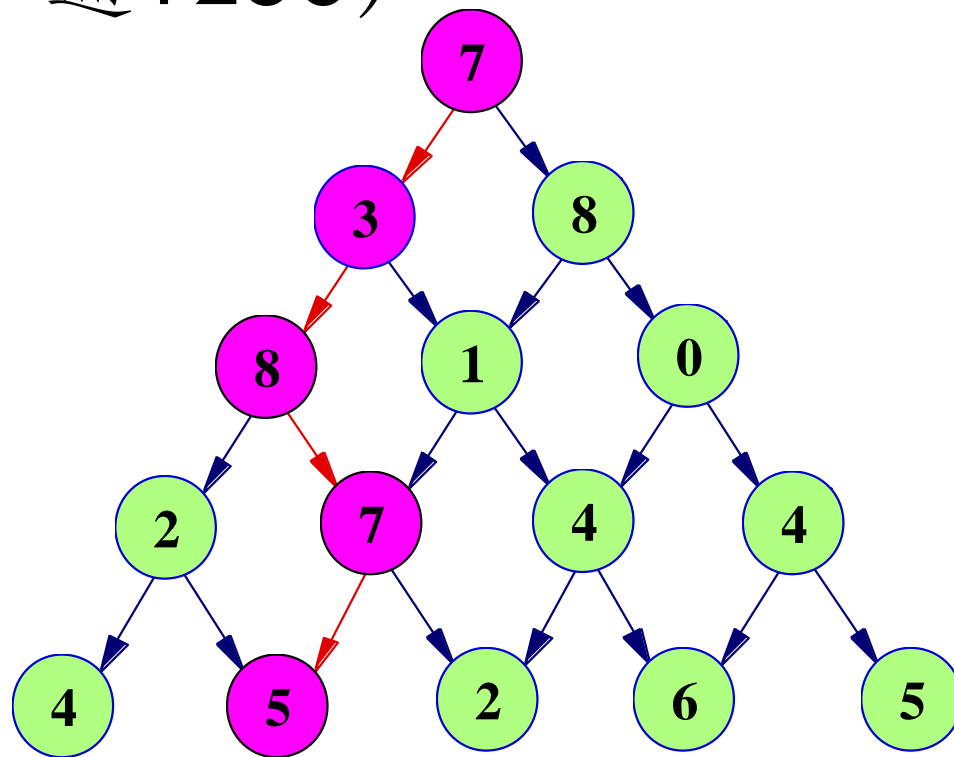
## 引例2: 数字金字塔 (一本通1258)

有一个数字三角形，编程求从最顶层到最底层的一条路所经过位置上数字之和的最大值。每一步只能向左下或右下方向走。



## 引例2: 数字金字塔 (一本通1258)

有一个数字三角形，编程求从最顶层到最底层的一条路所经过位置上数字之和的最大值。每一步只能向左下或右下方走。



路径数字最大和：  
 $7+3+8+7+5=30$

**输入样例:**

**5**

**7**

**3 8**

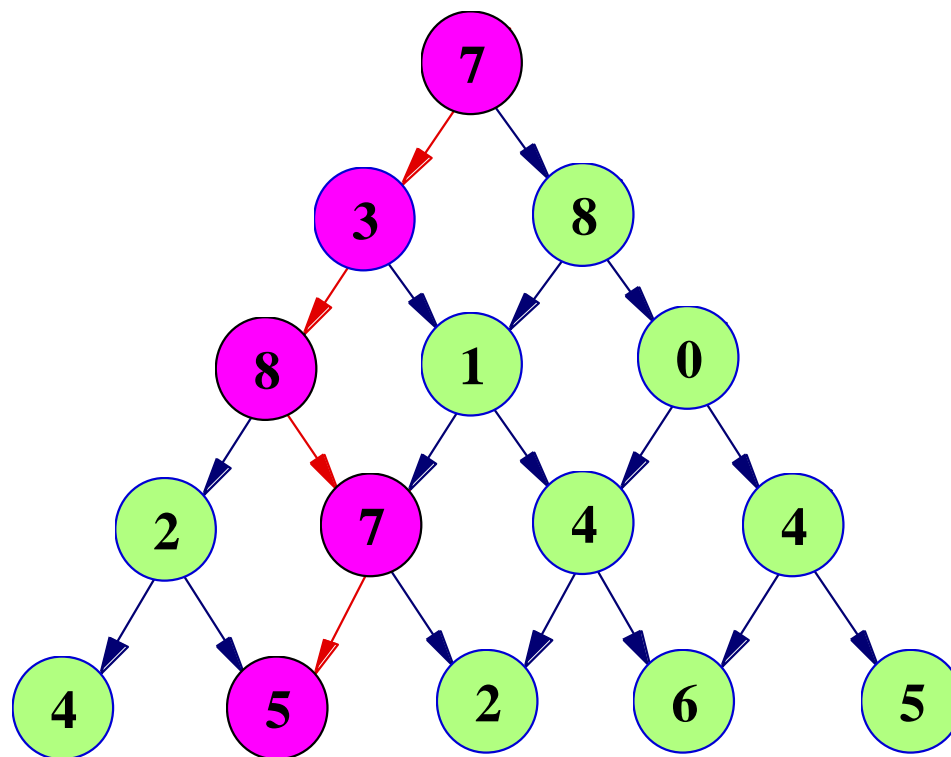
**8 1 0**

**2 7 4 4**

**4 5 2 6 5**

**输出样例:**

**30**



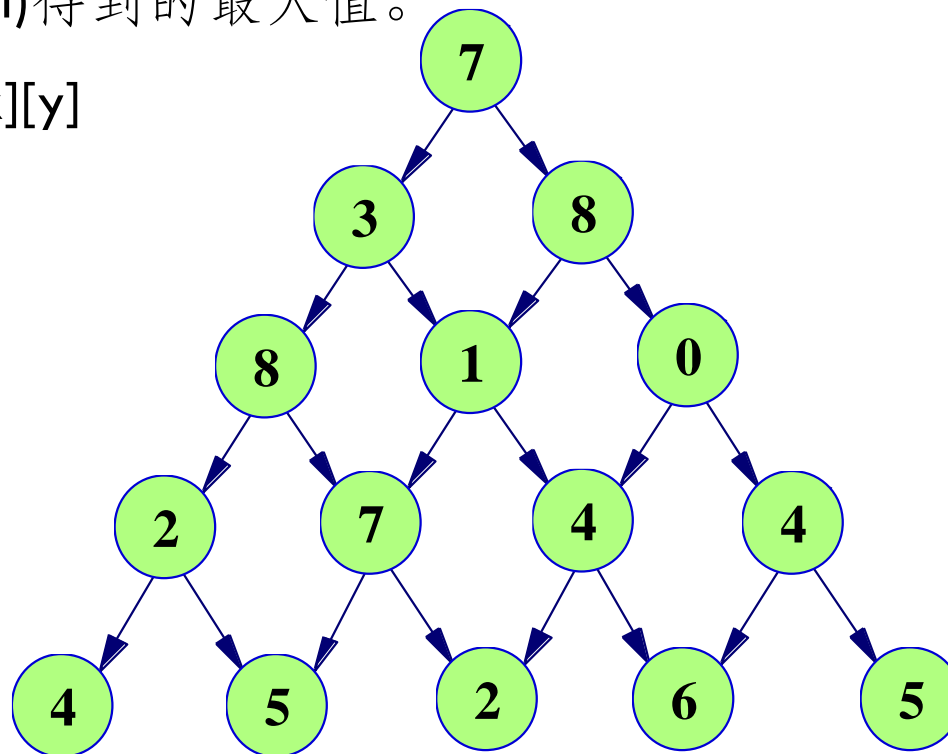
# 方法1：递归（搜索）

定义：函数 $\text{dfs}(x,y)$ 从 $(x,y)$ 走到最后一行 $(n,i)$ 得到的最大值。

$$\text{dfs}(x,y) = \max(\text{dfs}(x+1,y), \text{dfs}(x+1,y+1)) + a[x][y]$$

边界： $x=n$

起点 $\text{dfs}(1,1)$ ;



```
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];  
}
```

dfs(1,1)

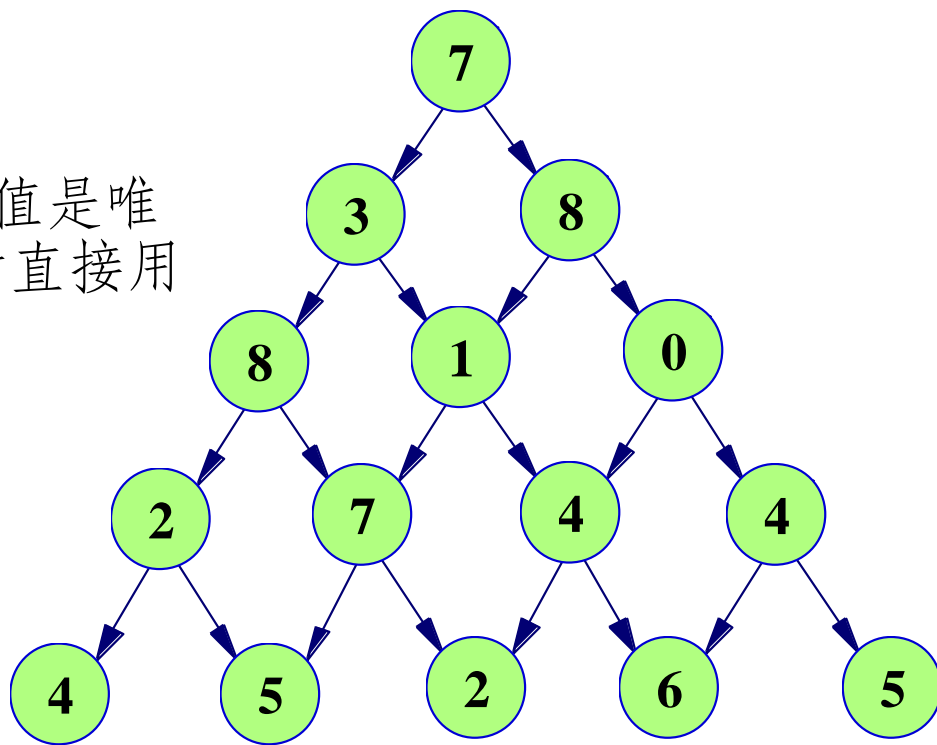


# 改进1:分析太慢的原因?

重复计算了很多的 $\text{dfs}(x,y)$ ;

改进:

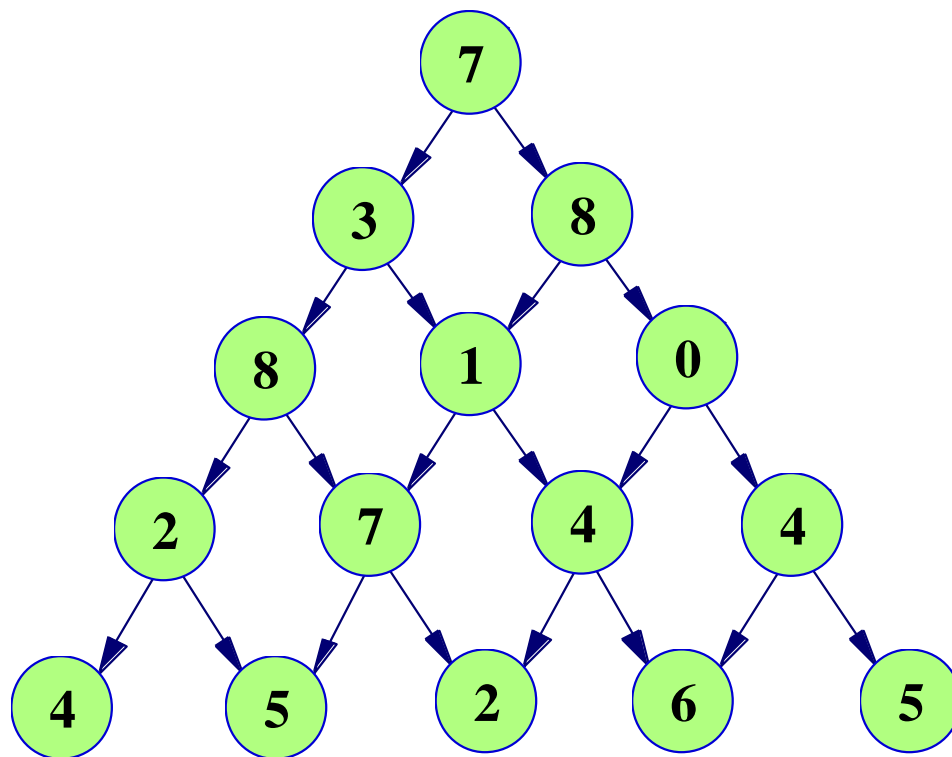
$f[x][y]$ :记录 $\text{dfs}(x,y)$ ,因为 $(x,y)$ 走到最后一行的最大值是唯一的,第一次走时记录下最优值,以后再用到时直接用 $f[x][y]$ 即可,不需要再递归求解。



```
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 $n, n-1, n-2, \dots, 1$ 行。



## 改进2：倒推

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;
```

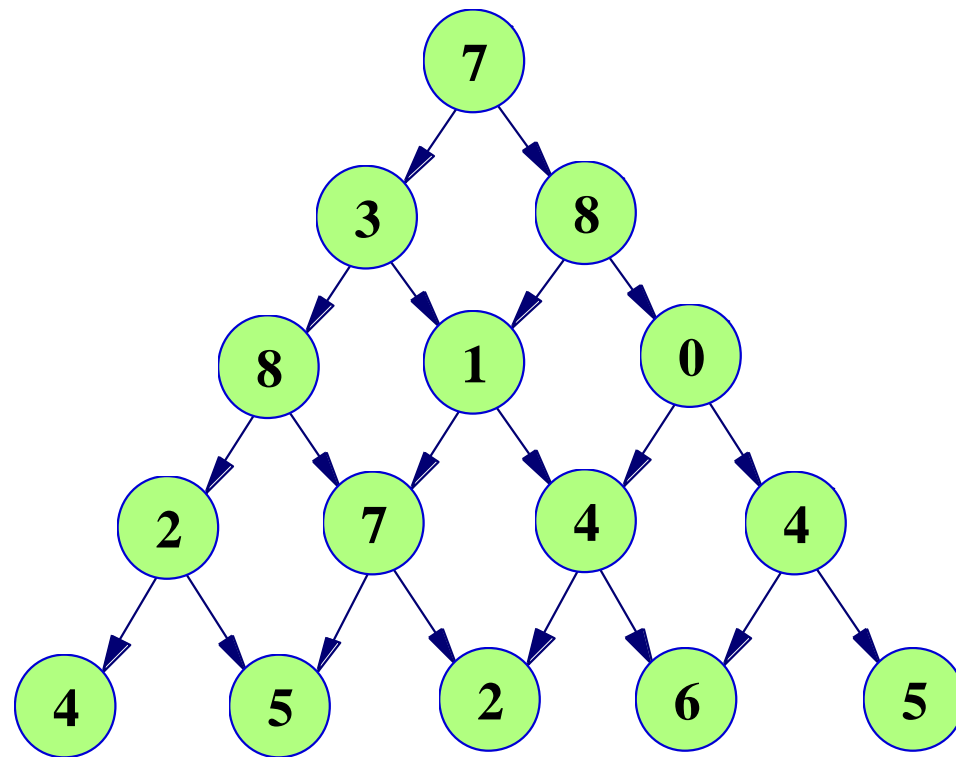
能否正向求？怎么定义？

# 正推（从第一行到最后一行）

$f[i][j]$ : 从  $(1,1)$  走到  $(i,j)$  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;
```

# 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。

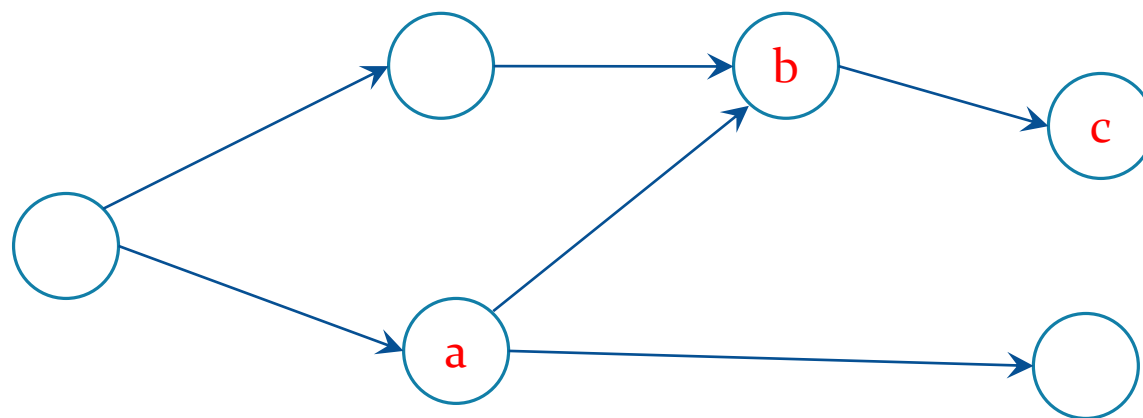
最优效果的策略叫最优策略。



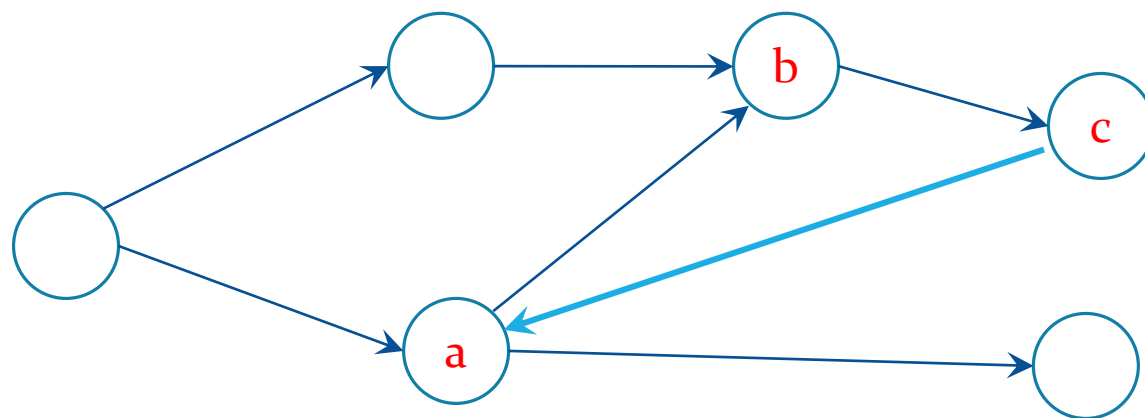
# 动态规划的条件

拓扑图（有向无环图）DAG  
图  
无后效性  
最优子结构

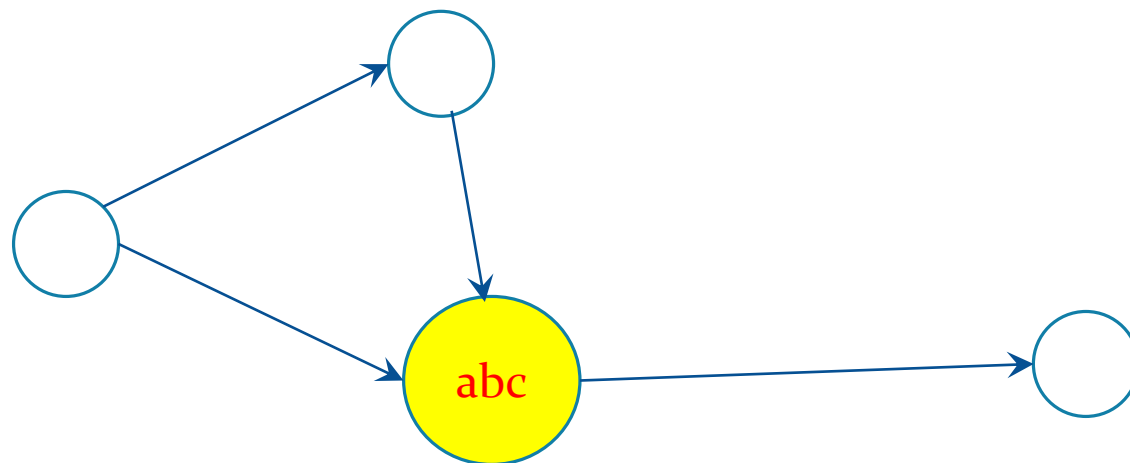
DAG图（有向无环图），可拓扑排序



有环，非拓扑图

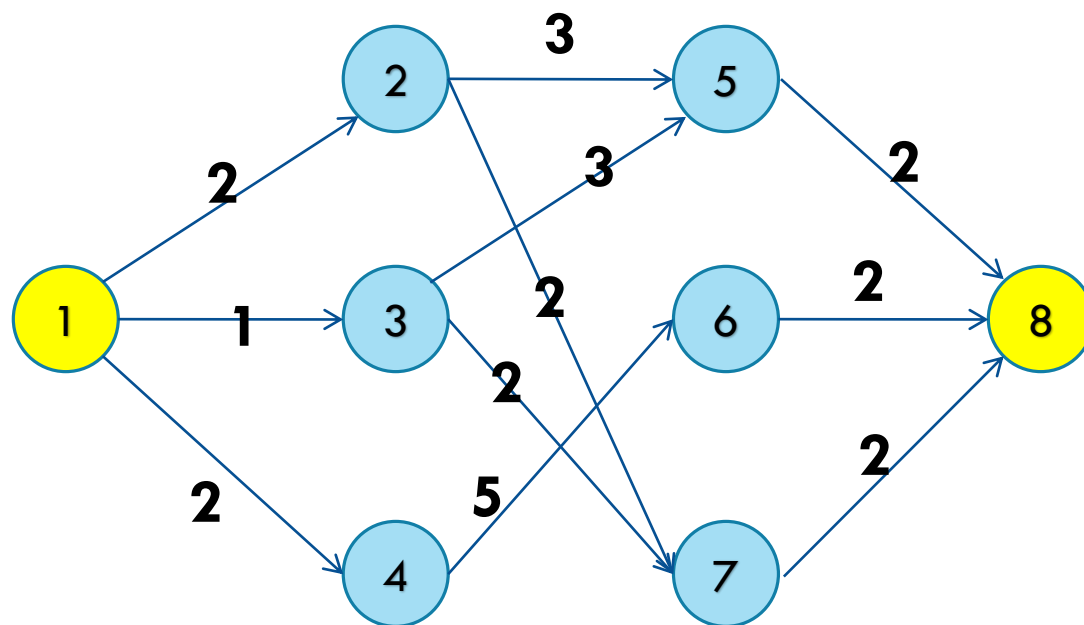


缩点，重新构造DAG



# 求1到8的最短距离？

一本通1261类似的题目

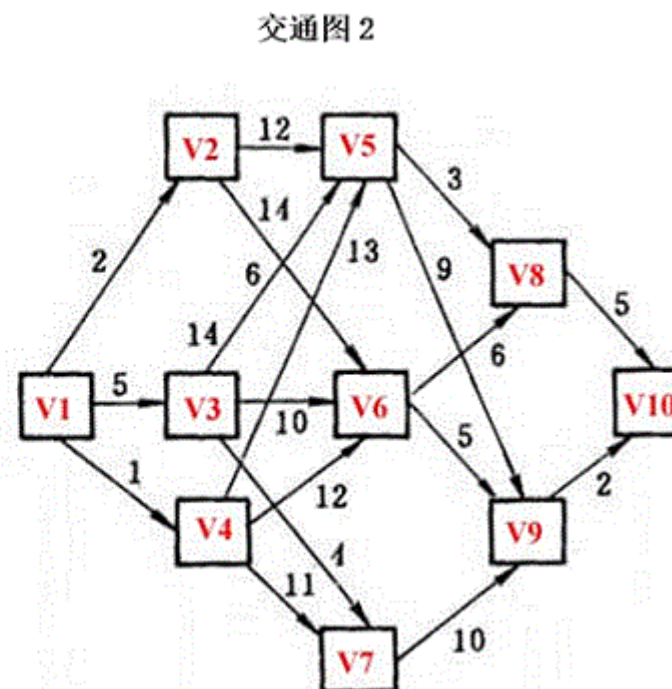
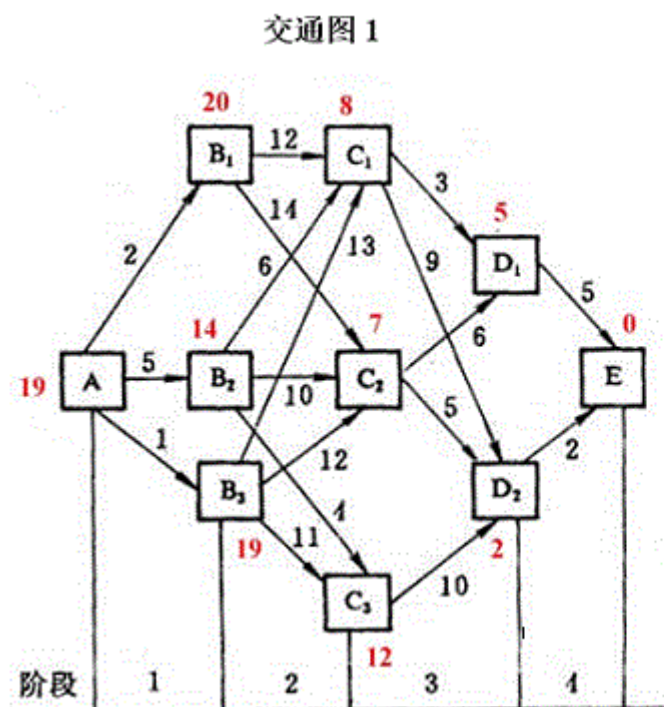


拓扑图（有向无环图）

无后效性 最优子结构

# 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。



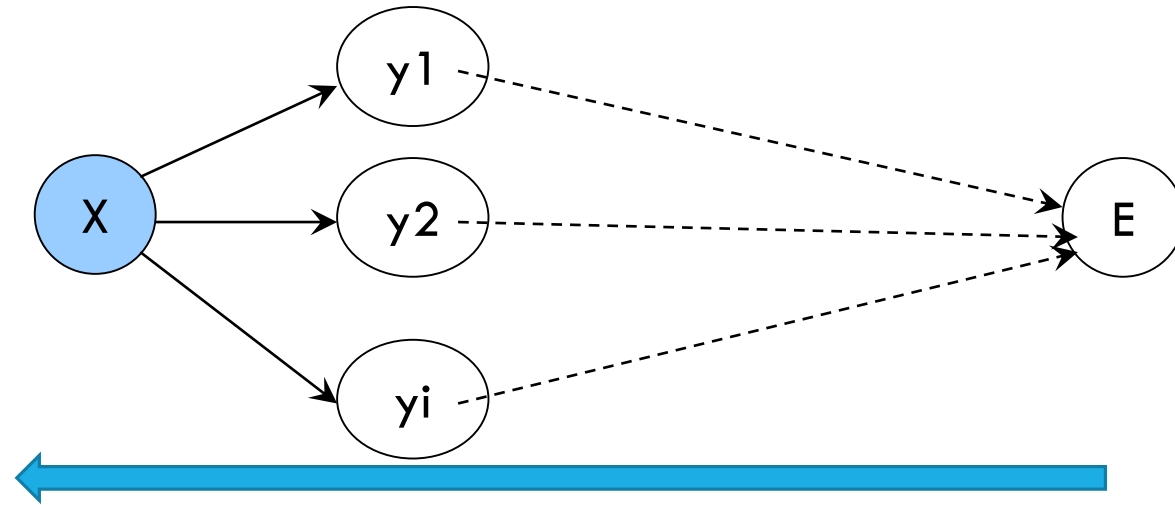
## $f[i]$ :以 $i$ 为起点的最短路

```
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];}
```



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  $k \leftarrow n-1$  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}

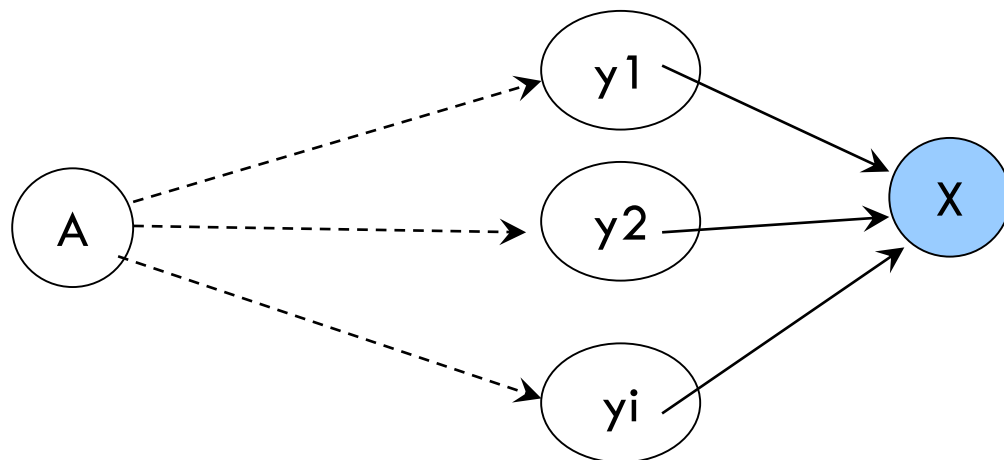
$f[U_k] = \text{opt}\{f[U_{k+1}] + L[U_k, X_k]\};$

        // $L[U_k, X_k]$ : 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  $k \leftarrow 2$  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

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

      //  $L[U_{k-1}, X_{k-1}]$ : 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

## 动态规划的步骤：

1.根据时间或空间确定要去的状态

2.写出动态规划方程

3.求解：记忆化搜索；递推



# DP 常见模型

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：

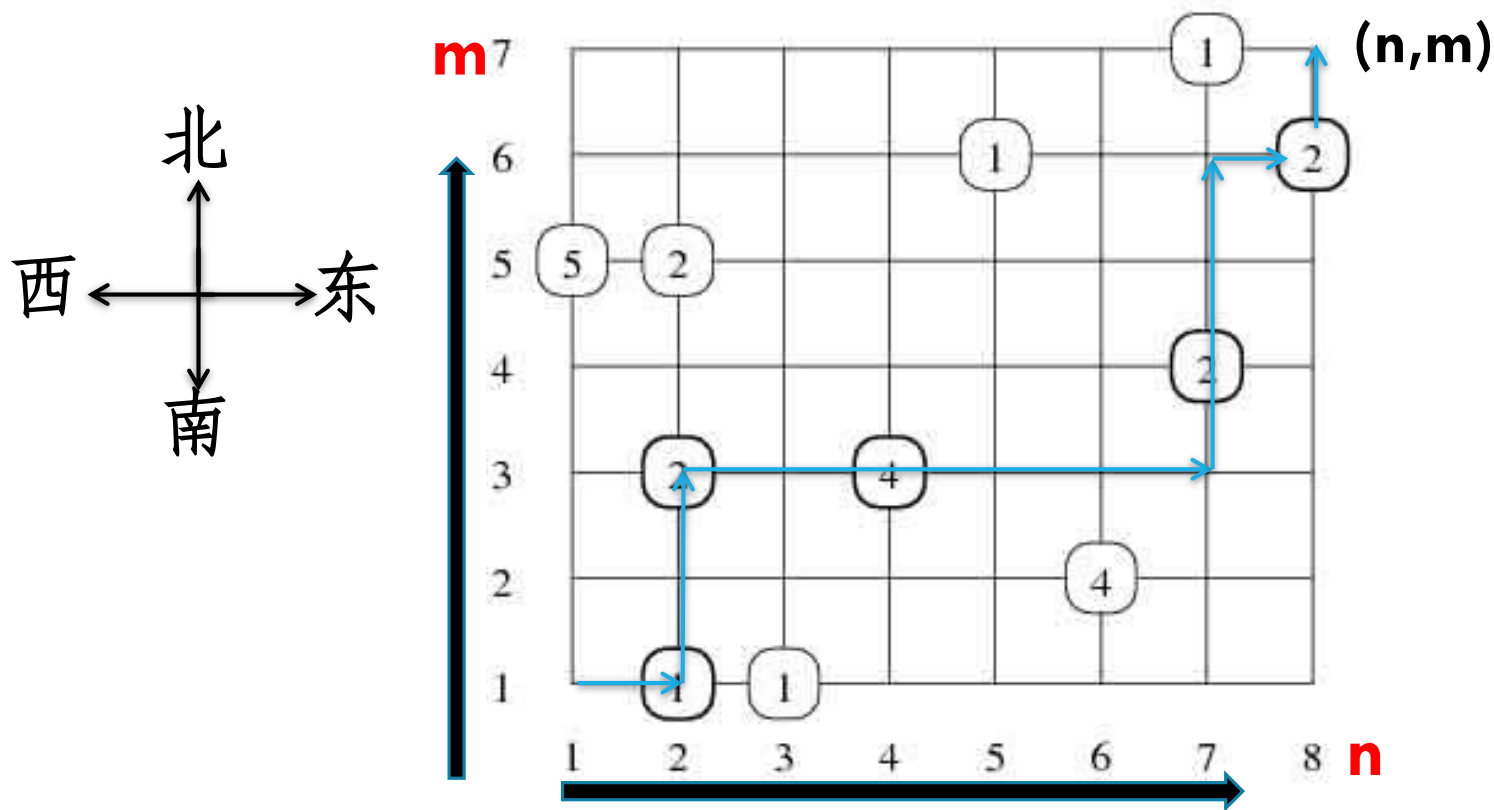
一般方程也是二维的  $f[i][j]$

# 例1：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  $1000000$ 。  $n, m \leq 1000$ 。

### 【输出】

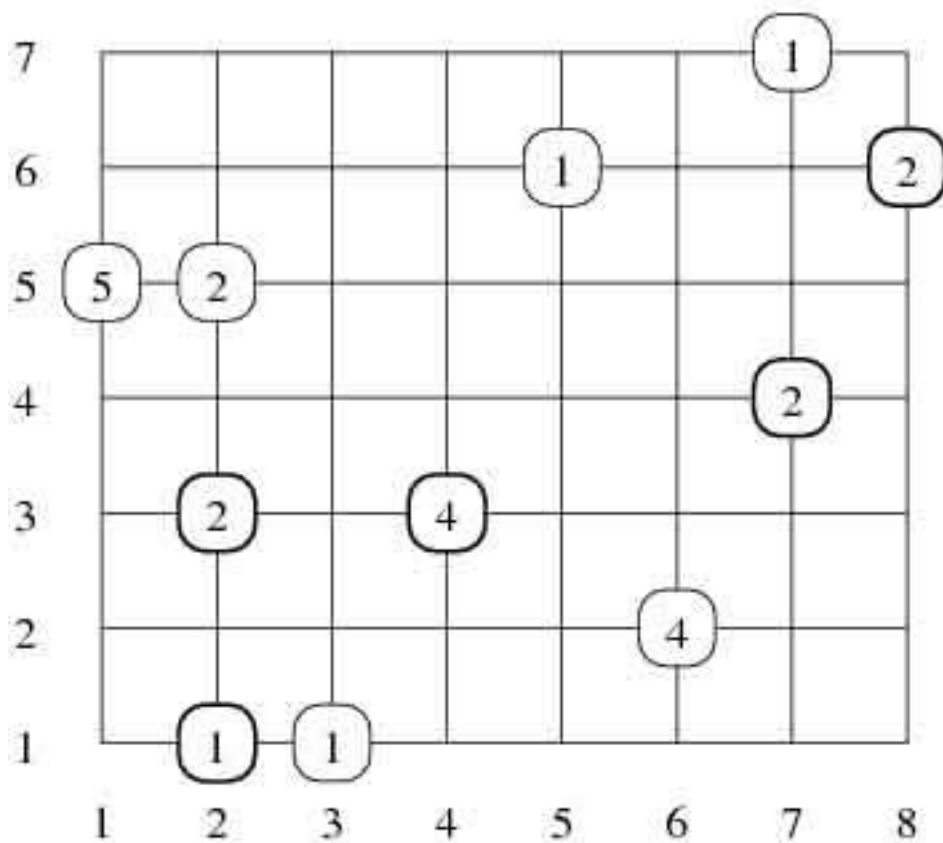
接到的最多的乘客数。

bus.in	bus.out
8 7 11	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	

$a[i,j]$  (i,j)位置的人数,

$f[i,j]$ :从 (1,1) 走到 (i, j) 能接的最多人数。

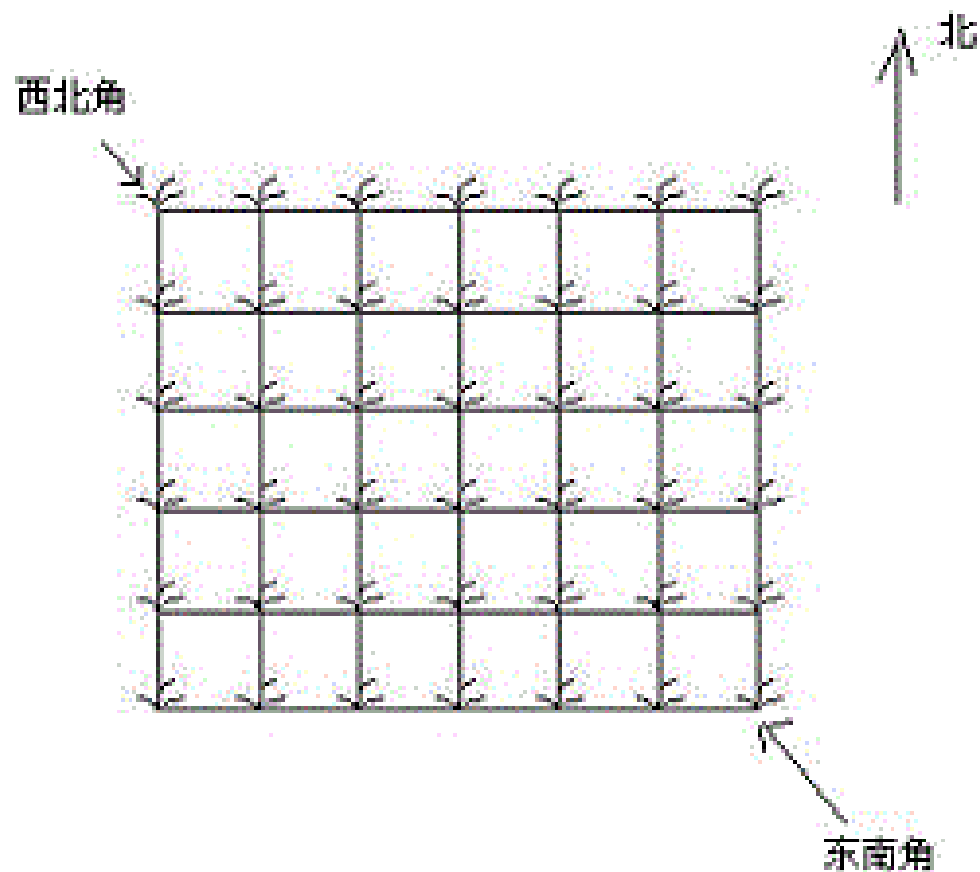
$$f[i,j] := \max\{f[i-1,j], f[i,j-1]\} + a[i,j]$$



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

## 1284：摘花生

**Hello Kitty**想摘点花生送给她喜欢的米老鼠。她来到一片有网格状道路的矩形花生地(如下图)，从西北角进去，东南角出来。地里每个道路的交叉点上都有种着一株花生苗，上面有若干颗花生，经过一株花生苗就能摘走该它上面所有的花生。**Hello Kitty**只能向东或向南走，不能向西或向北走。问**Hello Kitty**最多能够摘到多少颗花生。



## 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**)。

一个数的序列 $b_i$ ，当 $b_1 < b_2 < \dots < b_S$ 的时候，我们称这个序列是上升的。对于给定的一个序列 $(a_1, a_2, \dots, a_N)$ ，我们可以得到一些上升的子序列 $(a_{i_1}, a_{i_2}, \dots, a_{i_K})$ ，这里 $1 \leq i_1 < i_2 < \dots < i_K \leq N$ 。

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

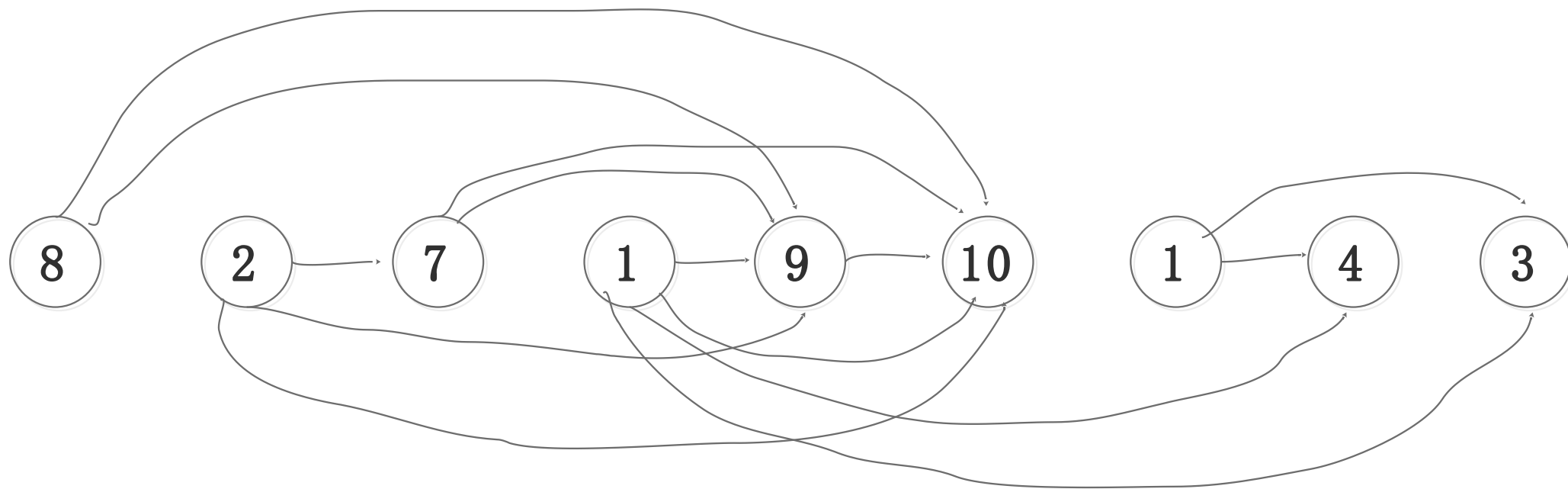
10

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）



$$f[i] = \max(f[j]) + 1 \quad (i < j \leq n \text{ \& \& } a[i] < a[j])$$

# 方法1：暴力搜索

8 2 7 1 9 10 1 4 3

找出以每个元素为起点的所有的上升子序列；  
然后选择最长的即可。

① 8 9 10

② 2 7 9 10

③ 7 9 10

④ 1 9 10

⑤ 9 10

⑥ 10

⑦ 1 4

⑧ 4

⑨ 3

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;
```

题目号	评测结果
1281	<p>未通过 20分</p> <p>测试点1: 答案正确 432KB 0MS</p> <p>测试点2: 答案正确 420KB 0MS</p> <p>测试点3: 运行超时 416KB 996MS</p> <p>测试点4: 运行超时 408KB 996MS</p> <p>测试点5: 运行超时 404KB 996MS</p> <p>测试点6: 运行超时 408KB 996MS</p> <p>测试点7: 运行超时 404KB 996MS</p> <p>测试点8: 运行超时 408KB 1000MS</p> <p>测试点9: 运行超时 416KB 996MS</p> <p>测试点10: 运行超时 420KB 996MS</p>

怎样在此基础上改进这个暴力搜索？

## 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  $f[i] \geq 1$ ，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 $a[i]$ 为起点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)

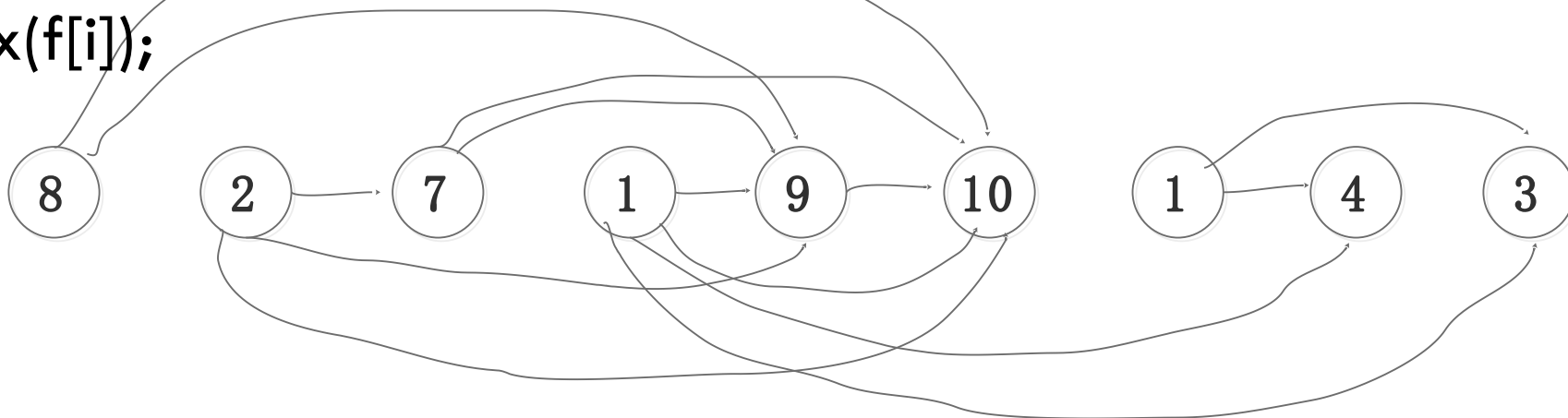
观察: 边的顺序:  $a[i]$ 向后的边 $i+1, i+1, \dots, n$ 中选择。

可以直接倒序求即可。

$f[n]=1;$

$f[i]=\max(f[j])+1 \quad (i < j \leq n \ \&\& \ a[i] < a[j])$

$\text{ans}=\max(f[i]);$



倒推求 $f[i]$ :

以 $a[i]$ 为起点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：8 2 7 1 9 10 1 4 3

$f[i]$ : 以  $a[i]$  为终点元素的最长子序列长度。

① 8

② 2

③ 2 7

④ 1

⑤ 2 7 9

⑥ 2 7 9 10

⑦ 1

⑧ 1 4

⑨ 1 3

正推求  $f[i]$ :

以  $a[i]$  为 终点 元素的最长子序列长度。

方程:

$$f[1]=1;$$

$$f[i]=\max(f[j])+1 \quad (1 \leq j < i \ \&\& \ a[j] < a[i])$$

$$\text{ans}=\max(f[i]);$$



# 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：

$$f[1]=1;$$

$$f[i]=\max(f[j])+1 \quad (1 \leq j < i \&\& a[j] < a[i])$$

$$\text{ans}=\max(f[i]);$$


$p[i]$ 记录 $f[i]$ 去最优值时的 $i$ 。

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 $a[i]$ 为起点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 $i+1, i+1, \dots, n$ 中选择。  
可以直接倒序求即可。

$f[n]=1;$

$f[i]=\max(f[j])+1 \quad (i < j \leq n \ \&\& \ a[i] < a[j])$

$ans=\max(f[i]);$

$p[i]$ 记录 $f[i]$ 去最优值时的 $j$ 。

找到最大的 $f[i]$ ,然后向 $\text{后}$ 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  $T_1, T_2, \dots, T_K$ ，则他们的身高满足  $T_1 < T_2 < \dots < T_i$ ,  $T_i > T_{i+1} > \dots > T_K$  ( $1 \leq i \leq K$ )。

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 $N$  ( $2 \leq N \leq 1000$ ), 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

$T_i$  ( $130 \leq T_i \leq 230$ ) 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  $n \leq 1000$ 。

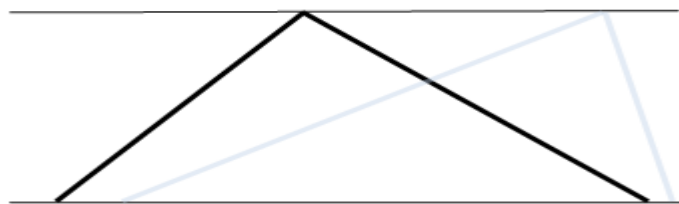
### 【样例输入输出】

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

你的任务，就是对于给定的序列，求出最大上升子序列和。

注意，最长的上升子序列的和不一定是最大的

6

10 2 188 6 7 8

最大上升子序列和为 $10+188=198$ ;

最长上升子序列为(2,6,7,8)。

# 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:

$f[n]=1;$

$f[i]=\max(f[j])+1 \quad (i < j \leq n \&\& a[i] < a[j])$

$ans=\max(f[i]);$

变为:

$f[n]=a[n];$

$f[i]=\max(f[j])+a[i] \quad (i < j \leq n \&\& a[i] < a[j])$

$ans=\max(f[i]);$

# 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。

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

初始：  $f[1] = a[1]$

目标：  $\max \{ f[i] \} \quad (1 \leq i \leq n)$

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:  $\max\{f[i]\} \quad 1 \leq i \leq n$

## 二维模型 $F[I][J]$



# 一.最长公共子序列模型LCS

最长公共子序列 (Longest Common Subsequence, LCS)

最长公共子串 (Longest Common Substring)

区别为：子串是串的一个连续的部分；子序列则是从改变序列的顺序，而从序列中去掉任意的元素而获得新的序列；也就是说，子串中字符的位置必须是连续的，子序列则可以不连续。

如：

abcbdad

bdcaba

2 1 4 5 6 7

3 2 5 4 7 6 8

## 1.1265 最长公共子序列 (LCS) (字符串 或 整数序列)

### 【问题描述】

给定两个字符序列: $X=\{x_1, x_2, \dots, x_n\}$ ;  $Y=\{y_1, y_2, \dots, y_m\}$

求X和Y的一个最长公共子序列长度。

举例:

$X=\{a,b,c,b,d,a,b\}$   $Y=\{b,d,c,a,b,a\}$

其中一个最长公共子序列为:  $LCS=\{b,c,b,a\}$ , 长度是4。LCS可能不止一个。

### 【输入】

第一行: 字符串X;

第二行: 字符串Y。

### 【输出】

最长公共子串的长度。

### 【样例输入】

a**bcb**dab

**bdc**aba

### 【样例输出】

4

$$X=\{x_1, \cdots, x_{i-1}, x_i\} \quad Y=\{y_1, \cdots, y_{j-1}, y_j\}$$

设 $f[i][j]$ 表示 $x[1..i]$ 与 $y[1..j]$ 的最长公共子序列的长度。

确定状态转移方程和边界条件：

分两种情况来考虑：

当 $x[i]=y[j]$ ： $x[i]$ 与 $y[j]$ 在公共子序列中，该情况下， $f[i][j]=f[i-1][j-1]+1$ 。

当 $x[i] \neq y[j]$ ：

$x[i]$ 不在公共子序列中：该情况下 $f[i][j]=f[i-1][j]$ ；

$y[j]$ 不在公共子序列中：该情况下 $f[i][j]=f[i][j-1]$ ；

$f[i][j]$ 取上述三种情况的最大值。综上：

$$\begin{aligned} \text{状态转移方程：} f[i][j] = \max \{ & f[i-1][j-1]+1; \quad x[i]=y[j] \\ & f[i-1][j]; \\ & f[i][j-1], \} \end{aligned}$$

边界条件： $f[0][j]=0, f[i][0]=0$ 。

目标： $f[n][m]$ ；



# 或者：

考虑：

$$X=\{x_1,\dots,x_{i-1}, x_i\}$$

$$Y=\{y_1,\dots,y_{j-1}, y_j\}$$

定义 $f[i,j]$ 为 $X$ 的前 $i$ 个字符和 $Y$ 的前 $j$ 个字符中最大公共子序列的长度。

当 $x_i=y_j$ 时： $f[i,j]=f[i-1,j-1]+1$ ；

当 $x_i\neq y_j$ 时： $f[i,j]=\max\{f[i,j-1], f[i-1,j]\}$ 。

注意字符串的下标和 $i$ 与 $j$ 的关系，字符下标从0开始。

## 2.1276：编辑距离

### 【题目描述】

设A和B是两个字符串。我们要用最少的字符操作次数，将字符串A转换为字符串B。  
这里所说的字符操作共有三种（对A而言）：

- 1、删除一个字符；
- 2、插入一个字符；
- 3、将一个字符改为另一个字符。

对任意的两个字符串A和B，计算出将字符串A变换为字符串B所用的最少字符操作次数。

### 【输入】

第一行为字符串A；第二行为字符串B；字符串A和B的长度均小于2000。

### 【输出】

只有一个正整数，为最少字符操作次数。

### 【输入样例】

sfdqxbw

gfdgw

### 【输出样例】

$X = \{x_1, \dots, x_{i-1}, x_i\}$

$Y = \{y_1, \dots, y_{j-1}, y_j\}$

设 $f[i][j]$ 表示把 $x[1..i]$ 变为 $y[1..j]$ 需要的最少操作次数。

状态转移方程：

$f[i][j] = \min\{$

    当 $x[i] = y[j]$  :  $f[i-1][j-1]$ ;

    当 $x[i] \neq y[j]$  :  $f[i-1][j] + 1$ ;   删除 $x[i]$

$f[i][j-1] + 1$ ;   在 $x[i]$ 后插入一个字符, 那么一定是 $=y[j]$ , 否则无意义

$f[i-1][j-1] + 1$ ; 将 $x[i]$ 变为 $y[j]$

$\}$

边界条件：

$f[0][i] = i$  全部插入

$f[i][0] = i$  全部删除

目标：  $f[n][m]$ ;

## 3.1298: 计算字符串距离

对于两个不同的字符串，我们有一套操作方法来把**他们变得相同**，具体方法为：

修改一个字符（如把“a”替换为“b”）；

删除一个字符（如把“traveling”变为“travelng”）。

比如对于“abcdefg”和“abcdef”两个字符串来说，我们认为可以通过增加/减少一个“g”的方式来达到目的。无论增加还是减少“g”，我们都仅仅需要一次操作。我们把这个操作所需要的次数定义为两个字符串的距离。

给定任意两个字符串，写出一个算法来计算出他们的距离。

# 1276和1298的不同点：

1276只对X操作，Y不变，让X变为Y

1298 X和Y都可以变，变化后相同

$X = \{x_1, \dots, x_{i-1}, x_i\}$

$Y = \{y_1, \dots, y_{j-1}, y_j\}$

设 $f[i][j]$ 表示把 $x[1..i]$ 变为 $y[1..j]$ 需要的最少操作次数。

状态转移方程：

$f[i][j] = \min\{$

    当 $x[i] = y[j]$ :  $f[i-1][j-1]$ ;

    当 $x[i] \neq y[j]$ :  $f[i-1][j] + 1$ ;   删除 $x[i]$

$f[i][j-1] + 1$ ;   删除 $y[j]$

$f[i-1][j-1] + 1$ ; 将 $x[i]$ 变为 $y[j]$ 或将 $y[j]$  变为 $x[i]$

$\}$

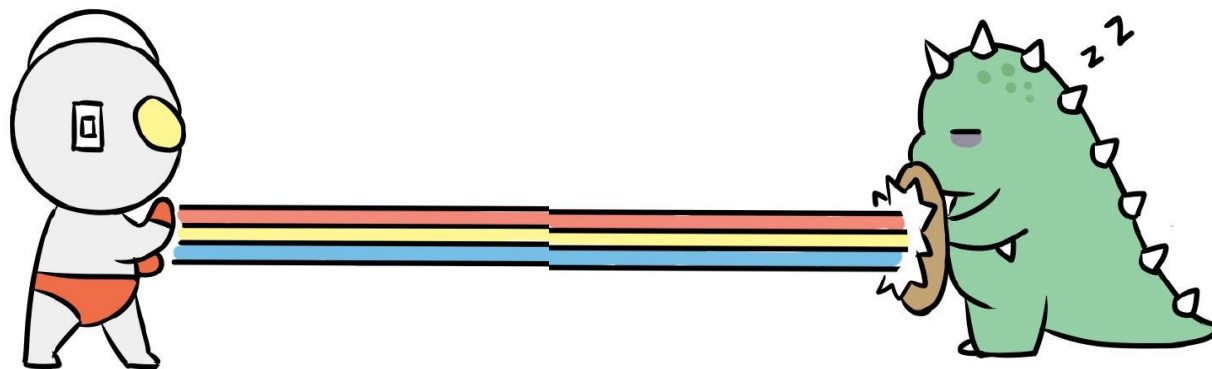
边界条件：

$f[0][i] = i$  Y全删

$f[i][0] = i$  X全除

目标:  $f[n][m]$ ;

## 二.资源分配问题



# 1.1266: 机器分配

$N$ 个分公司分配 $M$ 设备 $M$ 台.  $N \times M$ 的矩阵, 表明了第  $i$  个公司分配  $i$  台机器的盈利。

最大盈利值

如:

$n=7, m=6$

7 6

1 2 3 4 5 6

6 5 4 3 2 1

6 4 8 2 50 194

100 200 300 10 24 72

200 300 400 200 100 50

10 20 30 40 50 60

1 1 1 1 1 1

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



$f[i][j]$ : 前*i*个公司 (1..*i*) 分配*j*台设备最大获利。

考察前*i*-1个公司分配机器台数*k*,第*i*个公司分配*j*-*k*台

$$f[i][j] = \max\{f[i-1][k] + a[i][j-k]\} \quad (0 \leq k \leq j)$$

## 2.1275 乘积最大？

长度为 $n$ 的数串加 $m$ 个乘号，乘积最大。

【输入样例】

4 2

1231

【输出样例】

62

方法1:

$f[i][j]$ :  $i$  个乘号插入到前  $i$  个数字中的最大乘积。

9 4  
321044105

		3	2	1	0	4	4	1	0	5
	0	1	2	3	4	5	6	7	8	9
0		3	32	321	3210	...	...	...	...	...
1			?	?	?	?	?	?	?	?
2				?	?	?	?	?	?	?
3					?	?	?	?	?	?
4						?	?	?	?	?

$f[i][j]$ :  $i$  个乘号插入到前  $i$  个数字中的最大乘积。

考察第  $i$  个乘号的位置：第  $k$  和  $k+1$  个数字之间。  $(1,k)*(k+1,j)$

$f[i][j] = \max\{f[i-1][k] * \text{data}(k+1,j) \mid i \leq k < j\}$

初始值：  $f[0][i] = \text{data}(1,i)$

目标：  $f[m][n]$

行优先求

适合逐行求：

```
for(int i=1;i<=n;i++) f[0][i]=data(1,i);
for(int i=1;i<=m;i++)
    for(int j=i+1;j<=n;j++)
        for(int k=i;k<j;k++)
            f[i][j]=max(f[i][j],f[i-1][k]*data(k+1,j));
cout<<f[m][n]<<endl;
```

## 方法2:

$f[i][j]$ : 前  $i$  个数字中插入  $j$  个乘号到的最大乘积。

$f[i][j] = \max\{f[k][j-1] * \text{data}(k+1, i) \mid j \leq k < i\}$

初始值:  $f[i][0] = \text{data}(1, i)$

目标:  $f[n][m]$

		0	1	2	3	4
	0					
3	1	3				
2	2	32	?			
1	3	321	?	?		
0	4	3210	?	?	?	
4	5	...	?	?	?	?
4	6	...	?	?	?	?
1	7	...	?	?	?	?
0	8	...	?	?	?	?
5	9	...	?	?	?	?

列优先：

```
for(int i=1;i<=n;i++) f[i][0]=data(1,i);
for(int j=1;j<=m;j++) //列优先求
    for(int i=j+1;i<=n;i++)
        for(int k=j;k<i;k++)
            f[i][j]=max(f[i][j],f[k][j-1]*data(k+1,i));
cout<<f[n][m]<<endl;
```

## 行优先：

```
for(int i=1;i<=n;i++) f[i][0]=data(1,i);
for(int i=2;i<=n;i++) // 行优先
    for(int j=1;j<=min(m,i-1);j++)
        for(int k=j;k<i;k++)
            f[i][j]=max(f[i][j],f[k][j-1]*data(k+1,i));
cout<<f[n][m]<<endl;
```



## 4. 1278:复制书稿(BOOK)

复制时间最短。复制时间为抄写页数最多的人用去的时间。  
最大值最小问题。

【输入样例1】

9 3

1 2 3 4 5 6 7 8 9

【输出样例2】

1 5

6 7

8 9

样例2输入：

4 3

3 1 4 5

样例2输出：

1 1

2 3

4 4

# 先求出最小值

$f[i][j]$ :前 $i$ 人抄写前 $j$ 本书的最小值（抄的最多的人）。

考察第 $i$ 个人抄的书从第 $k+1$ 本到第 $j$ 本,即前 $i-1$ 个人抄前 $k$ 本书。

$i-1 \leq k \leq j-1$ :保证每人至少一本。

方程：

$f[i][j] = \min(f[i][j], \max(f[i-1][k], s[k+1][j]));$

$s[i][j]$ :第 $i$ 本书到第 $j$ 本书的页数。预先求出。

# 求出 $F[M][N]$ , 然后方案

从后先前分，尽量后面的人多抄，但还要保证前面每人至少保证一本。



# 区间型模型

区间型动态规划是线性动态规划的拓展，它将区间长度作为阶段，长区间的答案与短区间有关。在求解长区间答案前需先将短区间答案求出。区间型动态规划的典型应用有石子合并、乘积最大等。

## 1274: 石子合并

有 $N$ 堆石子( $N \leq 100$ )排成一行。现要将石子合并成一堆.规定每次只能选**相邻**的两堆合并成一堆新的石子,并将新的一堆的石子数,记为该次合并的得分.

选择一种合并石子的方案,使得做 $N-1$ 次合并,得分的总和**最少**。

输入数据:

第一行为石子堆数 $N$ ;

第二行为每堆石子数。

输出数据:

合并石子后得到的最小得分。

贪心算法：

每次合并相邻两堆和最小的那两堆。

如：

**4**

**1 3 5 2**

最小得分：**22**

反例：**7 4 4 7**

贪心： **$8+15+22=45$**

- 正确： **$11+11+22=44$**



应该怎么合并呢？

**8 3 6**

**3堆石子合并方案：**

$$11 + (11 + 6) = 28$$

$$9 + (8 + 9) = 26$$

$$\text{Ans} = \text{Min}(28, 26) = 26$$

8 5 5 8

**N=4**时，4堆一共合并了几次？

最后一次合并成一堆**前**的那**两**堆什么样？

8，18 或者 13,13 或者 18,8

哪种情况是理想的情况：

**Min (8+18,13+13,+18+8) =26**

子问题变成**3**堆和**2**堆的情况。

5堆石子:

$$\begin{aligned} (1,5) = \min\{ \\ (1,1) + (2,5); \\ (1,2) + (3,5); \\ (1,3) + (4,5); \\ (1,4) + (5,5)\} + \text{sum}[1,5] \end{aligned}$$

# N 堆石子： N-1次合并

- $(1,n)=\min\{a_1,a_2,a_3,\dots,a_{n-1},a_n$
- $(1,1)+(2,n);$
- $(1,2)+(3,n);$
- ...
- $(1,n-1)+(n,n)\}+\text{sum}[1,n]$
- $\text{Min}\{(1..k)+(k+1..n)\} + \text{sum}[1,n]$   
枚举：  $k=1..n-1$

# 动态规划算法：

定义  $f[i][j]$  表示从第  $i$  到第  $j$  堆间合并为一堆的最小代价。

$a[i], \dots, a[j]$  共有  $j-i+1$  堆石子

$sum[i, j] = a[i] + a[i+1] + \dots + a[j]$

状态转移方程： $f[i, j] := \{f[i, k] + f[k+1, j]\} + sum[i, j]$

枚举位置  $k$ ： $i \leq k \leq j-1$

初始： $f[i, i] = 0$ ;  $ans = f[1, n]$

## 前缀和：

$a[i]$ :记录第 $i$ 堆石子数量。

$s[i]=a[1]+a[2]+\dots+a[i]$ 。//前缀和

$sum[i,j]=s[j]-s[i-1]$ 。

# 实现方法1：记忆化搜索

```
int dp(int i, int j) {  
    if (i == j) return f[i][j] = 0;  
    if (f[i][j] > 0) return f[i][j];  
    f[i][j] = INF;  
    for (int k = i; k < j; k++)  
        f[i][j] = min(f[i][j], dp(i, k) + dp(k + 1, j) + s[j] - s[i - 1]);  
    return f[i][j];  
}
```

$f[i,j]$ : 第  $i$  到第  $j$  堆间合并为一堆的最小代价。

$$f[i,j] := \{f[i,k] + f[k+1,j]\} + \text{sum}[i,j]$$

	1	2	3	4	5	6
1	0	7	20	36	47	61
2		0	10	25	34	48
3			0	11	20	34
4				0	7	17
5					0	6
6						0



$$f[2,5] = \min(f[2,2] + f[3,5]; f[2,3] + f[4,5]; f[2,4] + f[5,5]) +$$

$$\text{sum}[2,5] = \min(20, 10 + 7, 25) + 17 = 34$$

	1	2	3	4	5	6
1	0	7	20	36	47	61
2		0	10	25	34	48
3			0	11	20	34
4				0	7	17
5					0	6
6						0

## 实现方法2：递推：合并第 I 堆 到第 J 堆

沿着对角线求：

外层循环变量d：从i开始的连续d堆石子

```
for(int d=2;d<=n;d++) //j-i+1=d
    for(int i=1;i<=n-d+1;i++) { //i+d-1<=n
        int j=i+d-1;
        for(int k=i;k<j;k++)
            f[i][j]=min(f[i][j],
                f[i][k]+f[k+1][j]+s[j]-s[i-1]);
    }
```

时间： $O(n^3)$

## 方法3：倒序按行优先求

	1	2	3	4	5	6
1	30	7	20	36	47	61
2		40	10	25	34	48
3			60	11	20	34
4				50	7	17
5					20	6
6						40

```
for(int i=n-1;i>=1;i--)  
    for(int j=i+1;j<=n;j++)  
        for(int k=i;k<j;k++)  
            f[i][j]=min(f[i][j],  
                f[i][k]+f[k+1][j]+s[j]-s[i-1]);  
cout<<f[1][n]<<endl;
```

# 总结本题：

- 1、前缀和的应用。
- 2、区间的**dp**的求解方法：  
是以区间长度的大小划分阶段。  
注意求解的顺序。

扩展一下：

**NOI95** 石子由一排改为围成一个环的形状？

**4 5 9 4**

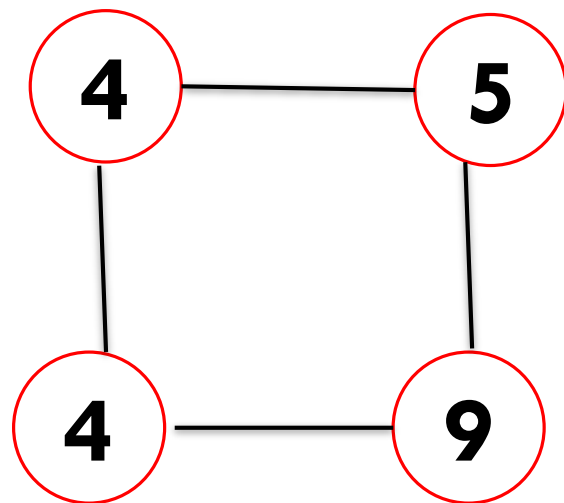
**4 5 9 4 4 5 9**

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_



**4 5 9 4 4 5 9**

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

# 环形石子合并算法：

环变成线性:长度:  $2n-1$

$a[1], a[2], \dots, a[n], a[n+1], \dots, a[2n-1]$ ;  $a[n+i] = a[i]$

$f[i, j]$ : 合并  $i$  到  $j$  堆的最小得分。

$$f[i, j] = \min\{f[i, k] + f[k+1, j]\} + s[j] - s[i-1] .$$

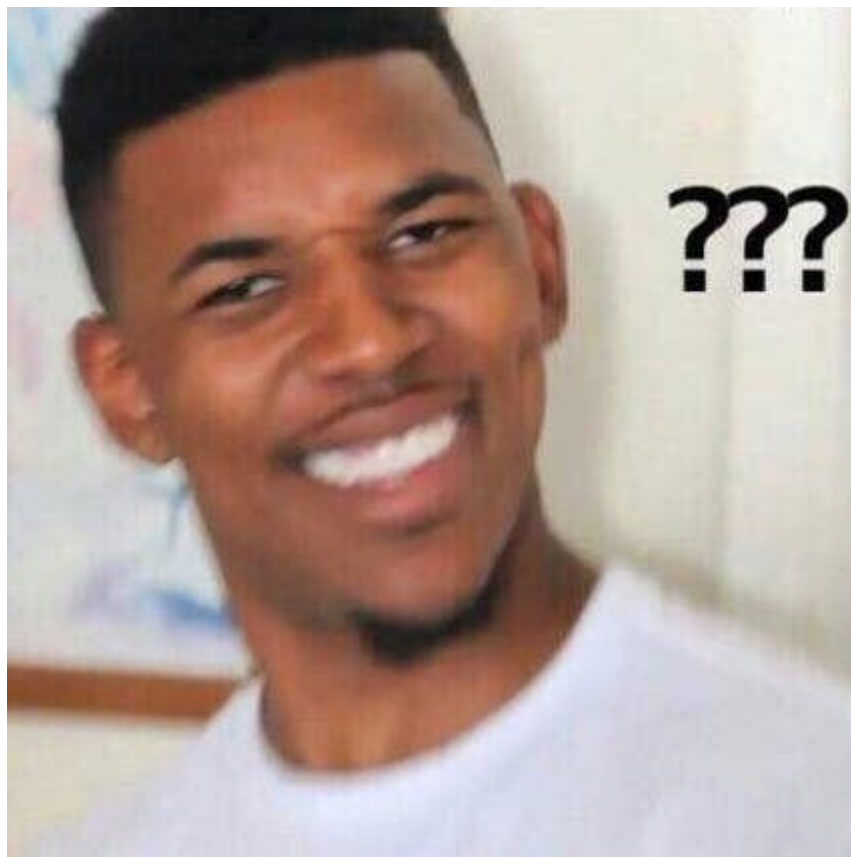
$(i \leq k \leq j-1)$

目标:  $ans = \min\{f[1, n], f[2, n+1], \dots, f[n, 2n-1]\}$

time  $O(n^3)$



# P1063 能量项链 (NOIP2006)

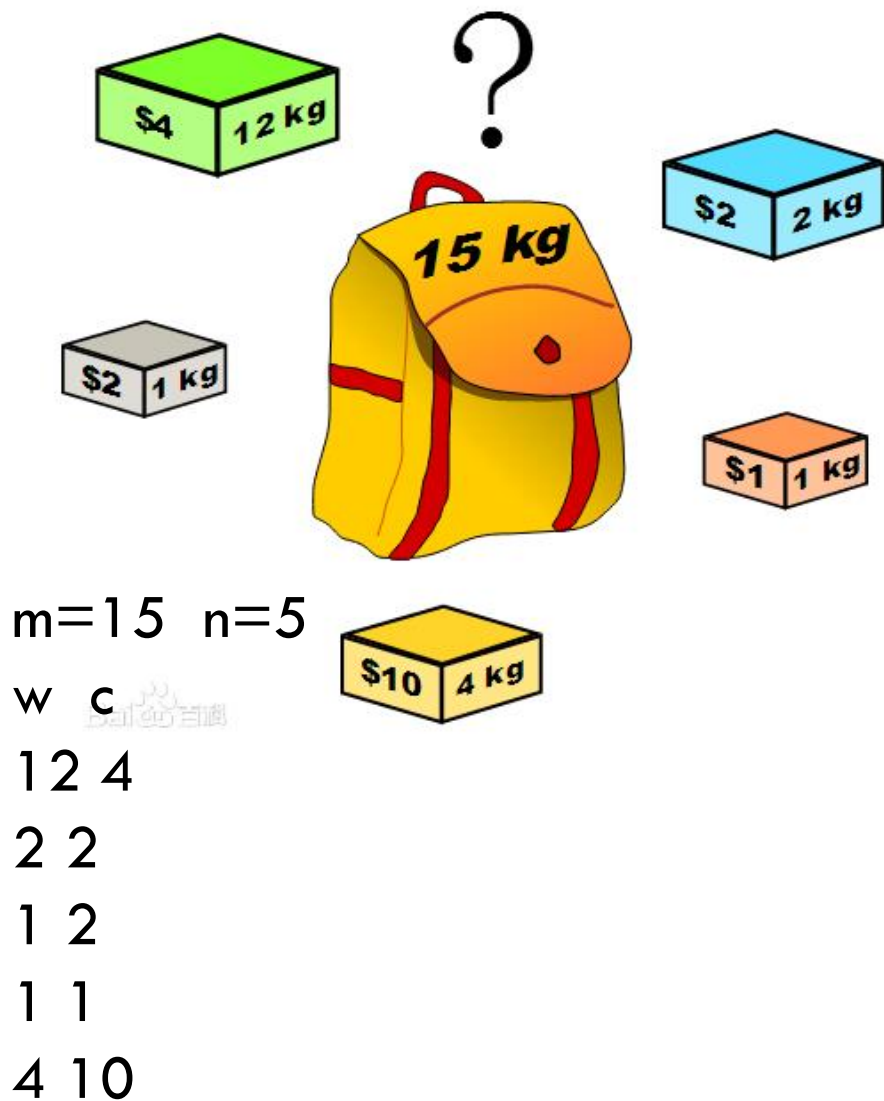




# 背包问题

0-1 背包  
完全背包  
多重背包  
混合背包

# 一. 0-1背包



有一个背包，最大载重量为 $m$ （或体积 $V$ ）。

有 $n$ 种货物：重量为  $w[i](<1000)$ （或体积）；  
价值为  $c[i](<1000)$ 。

今从  $n$  种物品中选取若干件放入背包，使其重量的和不超过 $m$ ，而所选货物的价值的和为最大。

$n \leq 1000, m < 1000$ . (有的  $n \leq 100, m < 10000$ )

求最大价值。

每件物品不放（0）或者放（1）（每种物品就1件）

输入：

**n**：物品种类

**m**：背包容积

**w[i]**：体积

**c[i]**：价值

输入样例1：

4 10

4 3 5 7

15 7 20 25

输出样例1：

35

输入样例2：

4 20

2 9 10 15

2 9 10 16

输出样例2：

19

价值大的？

单位价值大的？

# 方法1：暴力求解

每种物品都有选和不选两种决策：

递归枚举：

```
void dfs(int i,int j,int s){//对物品i进行决策，j剩余背包重量    if(i==n+1){  
        ans=max(ans,s);  
        return;  
    }  
    dfs(i+1,j,s);//不选物品i  
    if(j>=w[i])dfs(i+1,j-w[i],s+c[i]);//能装的下就选物品i  
}  
  
dfs(1,m,0);//从第一件物品开始枚举是放还是不放。
```

## 二进制枚举：

```
int k=1<<n;
for(int i=0;i<k;i++){
    int W=0,C=0;
    for(int j=0;j<n;j++){
        if(i&(1<<j)){
            W+=w[j+1];
            C+=c[j+1];
        }
        if(W<=m)ans=max(ans,C);
    }
}
cout<<ans<<endl;
```

枚举的时间：：时间  **$O(2^N)$**

适合  $n \leq 20$

物品04

**n=4   m=10**

编号	1	2	3	4
容量 $w$	4	3	5	7
价值 $v$	15	7	20	25

[illegible]



## 算法2:

设 $f[i][j]$ :从1到 $i$ 件物品中选若取若干件放到容量为 $j$ 的背包中, 获得的最大价值。目标是:  $f[n][m]$

$n=4$   $m=10$

编号	1	2	3	4
容量 $w$	4	3	5	7
价值 $v$	15	7	20	25

			体积										
序号	$w$	$c$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	4	15	0	0	0	0	15	15	15	15	15	15	15
2	3	7	0	0	0	7	15	15	15	22	22	22	22
3	5	20	0	0	0	7	15	20	20	22	27	35	35
4	7	25	0	0	0	7	15	20	20	25	27	35	35

用  $f[i][j]$  表示在第 1 到第  $i$  件物品中选择若干件到载重量为  $j$  的背包中所能获得的最大价值。

1)  $f[i-1][j]$ : 不放第  $i$  件物品获得的价值。

2)  $f[i-1][j-w[i]]+c[i]$ :

放第  $i$  件的价值。条件:  $j \geq w[i]$

方程 1:  $f[i][j] = \max\{ f[i-1][j] ,$

$f[i-1][j-w[i]]+c[i] : (j \geq w[i]) \}$

$(1 \leq i \leq n, 1 \leq j \leq m)$

目标:  $f[n, m]$ ;

非常重要，基本上所有跟背包相关的问题的方程都是由它衍生出来的！

设  $f[i][j]$ : 从 1 到  $i$  件物品中选若取若干件放到容量为  $j$  的背包中, 获得的最大价值。目标是:  $f[n][m]$

f	j=0	...	j-w[i]	...	j	...	m
i=0	0	...	0	...	0	...	0
...	...						
i-1	0		$f[i-1][j-w[i]]$	$+c[i]$	$f[i-1][j]$		
i					$f[i][j]$		
...	...						
n	0						$f[n][m]$

# 主程序：

实现1：

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

i的循环顺序无关紧要，因为依靠上一行

## 方程2:

$$f[i][j] = \max\{ f[i-1][j - k * w[i]] + k * c[i] \}$$

( $k=0..1$ : 不选与选;  $j \geq k * w[i]$ )

实现2:

```
for(int i=1; i<=n; i++)
```

```
    for(int j=0; j<=m; j++)
```

```
        for(int k=0; k<=1; k++)
```

```
            if(j >= k * w[i]) f[i][j] = max(f[i][j], f[i-1][j - k * w[i]] + k * c[i]);
```

# 空间优化:滚动数组实现:

$f[i][j] = \max(f[i-1][j], f[i-1][j-w[i]] + c[i])$

$f[i][j]$ 只与前一行有关系,所以可以滚动数组(随时更新,无需保留以前的): **floyed**也是用的滚动数组

$j$ 的枚举顺序必须从大到小,理解原因:

右边的 $f[j]$ 是原来的 $f[i-1][j]$ ,右边的 $f[j]$ 是更新后的是 $f[i][j]$

方程3:  $f[j] = \max\{f[j], f[j-w[i]] + c[i]\}$

实现3:

```
for(int i=1;i<=n;i++)
```

```
    for(int j=m;j>=w[i];j--) //重量要从大到小枚举
```

```
        f[j]=max(f[j],f[j-w[i]]+c[i]);
```

```
cout<<f[m]<<endl;
```

			体积										
序号	w	c	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	4	15	0	←									
2	3	7	0	←									
3	5	20	0								←		
4	7	25	0										

$$f[i] = \max(f[i], f[i-w[i]] + c[i])$$

如果*i*从小到大，则当*i*=1时：

$$f[4]=15, f[5]=15, f[6]=15, f[7]=15$$

$f[8] = \max(f[8], f[4] + 15) = \max(0, 15 + 15) = 30$ , 显然是错的, 物品1装了2次因为*f*[4]已经是15了, 已经装了一次了, 如果倒着求, 在求*f*[8]时, *f*[4]还是0, 就不会错了。

(正好有了后面的完全背包问题)

## 0-1 背包训练题目：

1290	采药	01 背包
1291	数字组合	01 背包计数
1294	Charm Bracelet	01 背包
1295	装箱问题	01 背包



## 二.完全背包

有 $n$ 种物品和一个容量为 $m$ 的背包，每种物品都有无限件可用。

第 $i$ 种物品的费用是 $w[i]$ ，价值是 $c[i]$ 。

求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

基本思路:

这个问题非常类似于01背包问题，所不同的是每种物品有无限件。也就是从每种物品的角度考虑，与它相关的策略已并非取或不取两种，而是有取0件、取1件、取2件.....等很多种。

如果仍然按照解01背包时的思路，令 $f[i][v]$ 表示前 $i$ 种物品恰放入一个容量为 $v$ 的背包的最大权值。仍然可以按照每种物品不同的策略写出状态转移方程，像这样：

$$f[i][v] = \max\{f[i-1][v-k*w[i]] + k*c[i] \mid 0 \leq k*w[i] \leq v\}。$$

将01背包问题的基本思路加以改进，可以推及其它类型的背包问题。

# 1268 【例9.12】 完全背包问题

## 【题目描述】

设有 $n$ 种物品，每种物品有一个重量及一个价值。但每种物品的数量是无限的，同时有一个背包，最大载重量为 $M$ ，今从 $n$ 种物品中选取若干件(同一种物品可以多次选取)，使其重量的和小于等于 $M$ ，而价值的和为最大。

## 【输入】

第一行：两个整数， $M$ (背包容量， $M \leq 200$ )和 $N$ (物品数量， $N \leq 30$ )；

第 $2..N+1$ 行：每行二个整数 $W_i, C_i$ ，表示每个物品的重量和价值。

```
10 4
2 1
3 5
4 6
7 9
max=16
```

$$2*5+6$$

## 0-1 背包的实现:

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

```
for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++)
        for(int k=0;k<=1;k++)
            if(j<=k*w[i]) f[i][j]=max(f[i][j], f[i-1][j-k*w[i]]+k*c[i]);
```

```
for(int i=1;i<=n;i++)
    for(int j=m;j>=w[i];j--)
        f[j]=max(f[j], f[j-w[i]]+c[i]);
```

# 完全背包的实现1:

方程1:  $f[i][j] = \max\{ f[i-1][j] ,$

$f[i-1][j-w[i]]+c[i] : (j \geq w[i]) \}$

0-1 背包



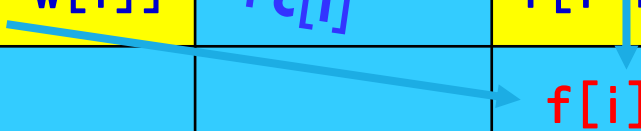
完全背包

方程1:  $f[i][j] = \max\{ f[i-1][j] ,$

$f[i][j-w[i]]+c[i] : (j \geq w[i]) \}$

# 0-1 背包

f	j=0	...	j-w[i]	...	j	...	m
i=0	0	...	0	...	0	...	0
...	...						
i-1	0		$f[i-1][j-w[i]]$	$+c[i]$	$f[i-1][j]$		
i					$f[i][j]$		
...	...						
n	0						$f[n][m]$



# 完全背包

f	j=0	...	j-w[i]	...	j	...	m
i=0	0	...	0	...	0	...	0
...	...						
i-1	0				$f[i-1][j]$		
i			$f[i][j-w[i]]$	$+c[i]$	$f[i][j]$		
...	...						
n	0						$f[n][m]$

# 完全背包的实现1:

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

0-1 背包



完全背包

区别在哪里?

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



## 完全背包的实现2:

$$f[i][j] = \max\{ f[i-1][j-k*w[i]] + k*c[i] \}$$

( $k=0..1$ : 不选与选;  $j \geq k*w[i]$ )

0-1 背包



完全背包

$$f[i][j] = \max\{ f[i-1][j-k*w[i]] + k*c[i] \}$$

( $k=0..j/w[i]$ ;  $j \geq k*w[i]$ )

## 完全背包的实现2:

```
for(int i=1;i<=n;i++)  
    for(int j=0;j<=m;j++)  
        for(int k=0;k<=1;k++)  
            if(j<=k*w[i]) f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*c[i]);
```

0-1 背包



完全背包

区别在哪里?

```
for(int i=1;i<=n;i++)  
    for(int v=0;v<=m;v++)  
        for(int k=0;k<=v/w[i];k++)  
            f[i][v]=max(f[i][v],f[i-1][v-k*w[i]]+k*c[i]);
```

# 完全背包的实现3:

```
for (int i=1; i<=n; i++)  
    for (int j=m; j>=w[i]; j--)  
        f[j]=max(f[j], f[j-w[i]]+c[i]);
```

0-1 背包



完全背包

区别在哪里?

```
for (int i=1; i<=n; i++)  
    for (int j=w[i]; j<=m; j++)  
        f[j]=max(f[j], f[j-w[i]]+c[i]);
```

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 $V/w[i]$ 件，于是可以把第 $i$ 种物品转化为 $V/w[i]$ 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z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。

高效的转化方法是：把第 $i$ 种物品拆成费用为 $w[i]*2^k$ 、价值为 $c[i]*2^k$ 的若干件物品，其中 $k$ 满足 $w[i]*2^k < V$ 。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 $O(\log(V/w[i])+1)$ 件物品，是一个很大的改进。后面多重背包也用到这种方法。

# 完全背包训练：

**1273： 【例9.17】 货币系统**

**1293： 买书**

## 三.多重背包

有 $n$ 种物品和一个容量为 $V$ 的背包。

第 $i$ 种物品最多有 $s[i]$ 件可用，每件费用是 $w[i]$ ，价值是 $c[i]$ 。

求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

## 【例9.13】庆功会 1269

### 【题目描述】

为了庆贺班级在校运动会上取得全校第一名成绩，班主任决定开一场庆功会，为此拨款购买奖品犒劳运动员。期望拨款金额能购买最大价值的奖品，可以补充他们的精力和体力。

### 【输入】

第一行二个数 $n(n \leq 500)$ ， $v(v \leq 6000)$ ，其中 $n$ 代表希望购买的奖品的种数， $v$ 表示拨款金额。

接下来 $n$ 行，每行3个数， $w$ 、 $c$ 、 $s$ ，分别表示第 $i$ 种奖品的价格、价值（价格与价值是不同的概念）和能购买的最大数量（买0件到 $s$ 件均可），其中 $w \leq 100$ ， $c \leq 1000$ ， $s \leq 10$ 。

### 【输出】

一行：一个数，表示此次购买能获得的最大的价值（注意！不是价格）。

### 【输入样例】      【输出样例】

5 1000	1040
80 20 4	
40 50 9	
30 50 7	
40 30 6	
20 20 1	

## 方法1：采用类似完全背包的方法（2）

第*i*种物品有*s[i]*+1种策略：取0件，取1件.....取*s[i]*件。

令*f[i][j]*表示前*i*种物品恰放入一个容量为*j*的背包的最大权值，则：

$f[i][j] = \max\{f[i-1][j-k*w[i]] + k*c[i] \mid 0 \leq k \leq s[i]\}$ 。复杂度是 $O(V * \sum s[i])$ 。



```
for(int i=1;i<=n;i++)
    for(int j=0;j<=v;j++)
        for(int k=0;k<=s[i];k++)
            if(j>=k*w[i]) f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*c[i]);
```

## 滚动数组实现：

```
for(int i=1;i<=n;i++)  
    for(int j=v;j>0;j--)  
        for(int k=0;k<=s[i];k++)  
            if(j<=k*w[i]) f[j]=max(f[j], f[j-k*w[i]]+k*c[i]);
```

## 方法2：转换为0-1背包（滚动数组）：

第*i*件物品，看做*s[i]*件一样的物品*i*，变成了*s[1]+..+s[n]*件物品的0-1背包，每个背包取还是不取，和方法1有区别的。

```
for(int i=1;i<=n;i++)  
    for(int j=1;j<=s[i];j--) //s[i] 个物品都是i的01背包  
        for(int k=v;k>=w[i];k--)  
            f[k]=max(f[k],f[k-w[i]]+c[i]);
```

复杂度是 $O(V * \sum s[i])$

## 方法3：方法2的改进

二进制思想

如： $s[i]=13$ ,没有必要增加13个物品，只需增加4件物品即可：

价格和价值分别乘系数：

1,2,4,6

$(1,2,\dots,2^{(k-1)},s[i]-(2^k-1))$  因为  $2^k-1=1+2+4+\dots+2^{(k-1)}$

1,2,4,6能组出1到13的任意数，等价13件物品

时间复杂度： $O(V*\sum(\log(s[i])))$

```
for (int i=1; i<=n; i++) {  
    int x, y, z, k=1;  
    cin >> x >> y >> z;  
    for (k=1; z>0; k=2*k) {  
        int d=min(k, z);  
        if (d>0) {  
            w[++sn]=d*x;  
            c[sn]=d*y;  
        }  
        z=z-k;  
    }  
}
```

```
for (int i=1; i<=sn; i++)  
    for (int j=v; j>=w[i]; j--)  
        f[j]=max(f[j], f[j-w[i]]+c[i]);
```

# 建议：

0-1 背包，完全背包，多重背包都采用一维的滚动数组实现。

0-1 背包

```
for (int i=1; i<=n; i++)  
    for (int j=m; j>=w[i]; j--)  
        f[j]=max(f[j], f[j-w[i]]+c[i]);
```

完全背包

```
for (int i=1; i<=n; i++)  
    for (int j=w[i]; j<=m; j++)  
        f[j]=max(f[j], f[j-w[i]]+c[i]);
```

多重背包

```
for (int i=1; i<=n; i++)  
    for (int j=v; j>0; j--)  
        for (int k=0; k<=s[i]; k++)  
            if (j>=k*w[i]) f[j]=max(f[j], f[j-k*w[i]]+k*c[i]);  
  
for (int i=1; i<=n; i++)  
    for (int j=1; j<=s[i]; j--) //s[i] 个物品都是i的01背包  
        for (int k=v; k>=w[i]; k--)  
            f[k]=max(f[k], f[k-w[i]]+c[i]);
```

## 四.混合背包问题

如果将0-1背包、完全背包、多重背包混合起来。

也就是说，有的物品只可以取一次（01背包），有的物品可以取无限次（完全背包），有的物品可以取的次数有一个上限（多重背包）。

应该怎么求解呢？



# 1270: 【例9.14】混合背包

如果有3种背包，可以：

完全背包单独处理

0-1背包和多重背包都可以化为0-1背包处理。

结构：

```
for(int i=1;i<=n;i++){
```

- if完全背包{
- }
- else{01或多重背包
- }

```
}
```