信息学奥林匹克竞赛

Olympiad in Informatics

# 动态规划之基础知识

陈 真

太原市第五中学校

# 基础知识复习

## DP的定义

**动态规划**（DynamicProgramming，简称DP）中的Programming指的是一种**表格法**，DP通过**合理安排子问题求解的顺序**，使得每个子问题只求解一次，并将**解保存在一个表格**中，需要的时候**查表**即可。

动态规划常用于解决多阶段决策最优化问题。这类特殊的问题可以按时间、空间等顺序分解成若干相互联系的阶段，在每个阶段都需要做出决策使得整个问题的解获得最优。

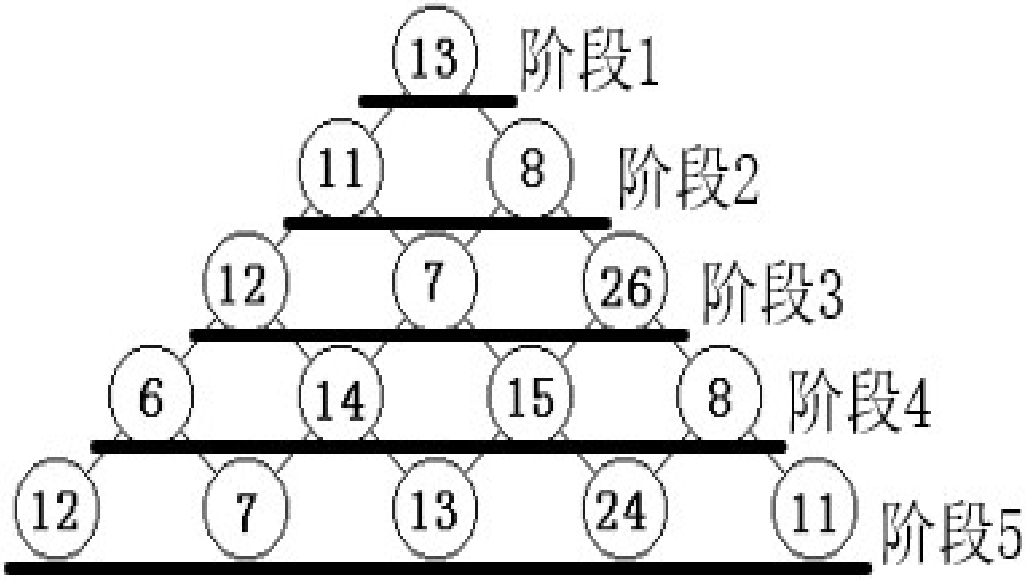
# 基础知识复习

## DP的相关概念

### 1.阶段

DP按照时间、空间等将问题的求解过程划分若干个相互联系的**阶段**，描述阶段的变量称为**阶段变量**。

可以用变量i代表第i个阶段，变量i也叫阶段变量。




以数字金字塔为例

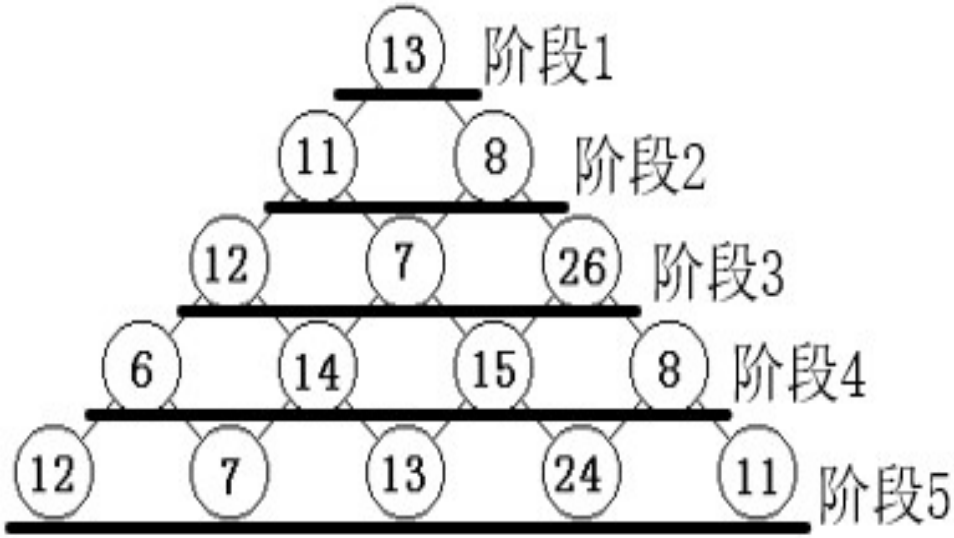
# 基础知识复习

## DP的相关概念

### 2.状态

每一个阶段中包含的“位置”为状态。这里的“位置”可以看作每个阶段所处的客观条件或自然状态，描述状态的变量称为**状态变量**。

二维状态变量（i, j）描述阶段里的状态，变量i同时也是阶段变量。




以数字金字塔为例

# 基础知识复习

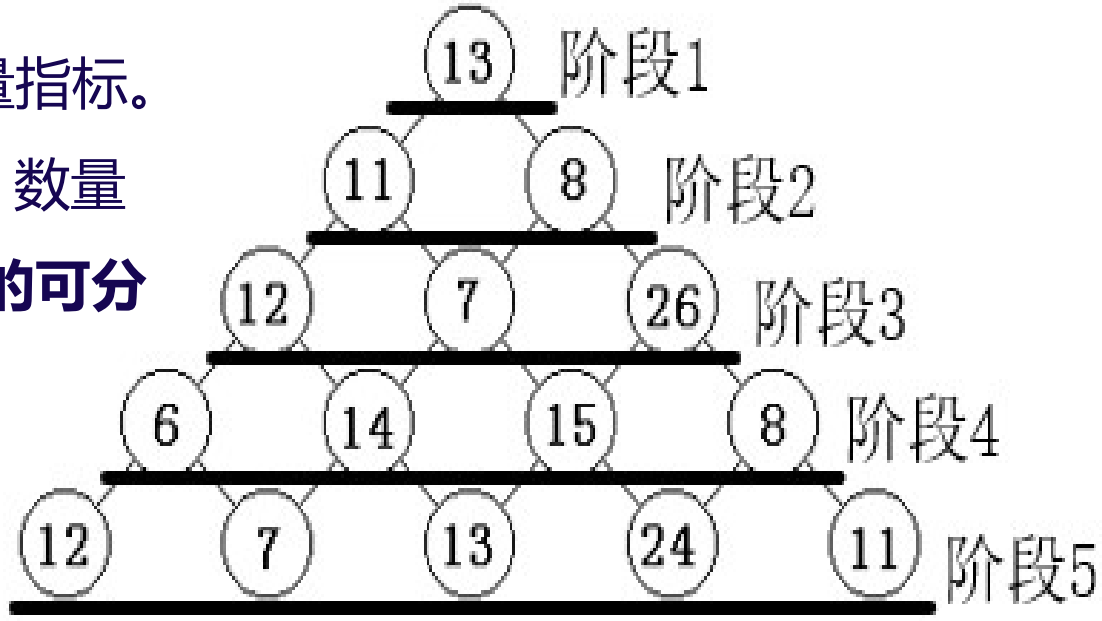
## DP的相关概念

最优值函数f可以表示极大值。设f(i,j)表示从顶端走到状态（i，j）获得最大值。

### 3.指标函数与最优值函数

指标函数是用于衡量过程优劣的一种数量指标。这种数量指标可以是距离、利润、成本、数量等等。DP的**指标函数必须是有关于阶段的可分离形式**，例如和、积或其它形式。

指标函数的最优值称为最优值函数。




以数字金字塔为例

**最优值函数常常直接体现在了DP的状态设计中。**

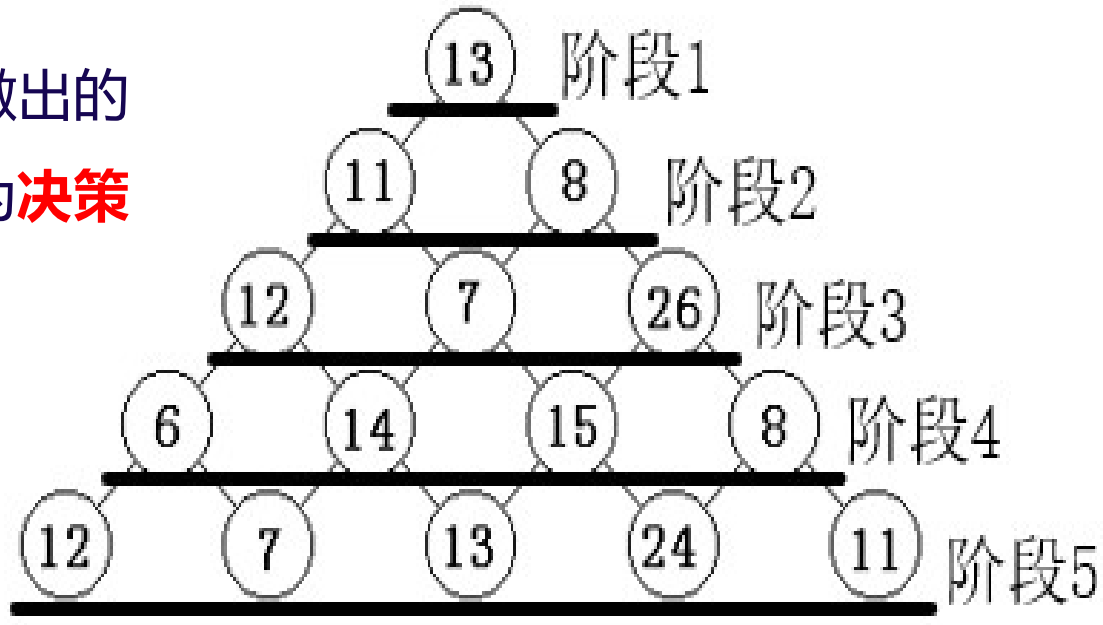
# 基础知识复习

## DP的相关概念

### 4.决策

在某一阶段，在给定的某个状态下可以做出的若干选择即为决策，描述决策的变量称为**决策变量。**

在数字金字塔问题中，当状态为(i,j)时，他可以做出的决策有两种，一是向左下方走到(i+1,j)，二是向右下方走(i+1,j+1)。



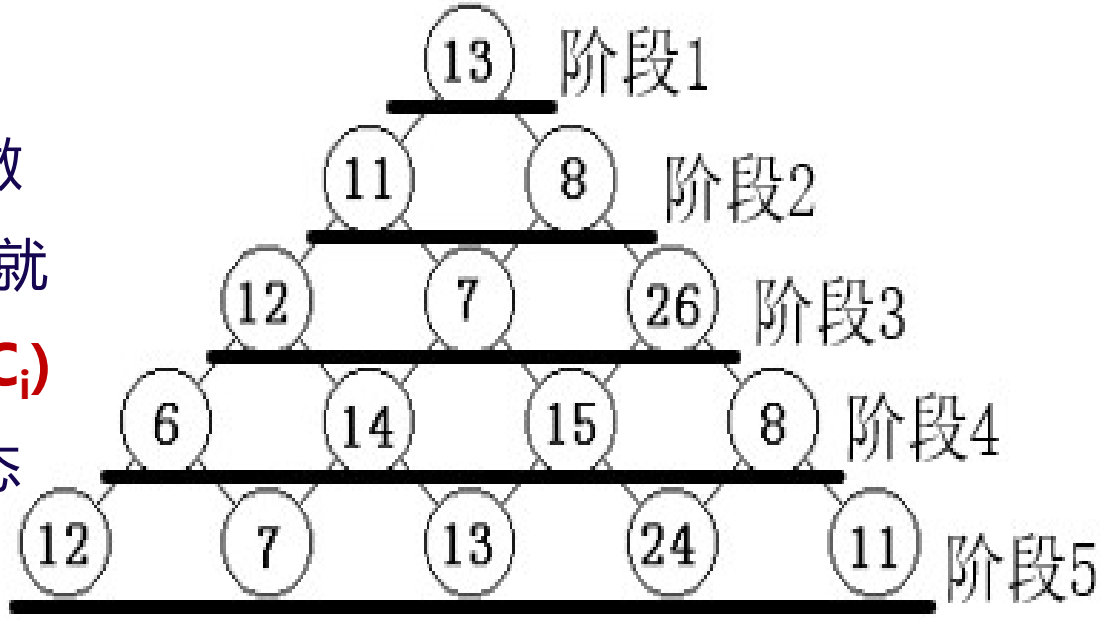

以数字金字塔为例

# 基础知识复习

## DP的相关概念

### 5.状态转移方程

若第i阶段的某个状态为$S_i$，在该状态下做出某种决策$C_i$后，下一阶段的状态$S_{i+1}$也就随之确定，我们可以用方程**$S_{i+1}=T(S_i，C_i)$**来描述这一过程。这种描述了状态与状态之间演变规律的方程即为**状态转移方程。**




以数字金字塔为例

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

# 基础知识复习

## DP的求解三要素

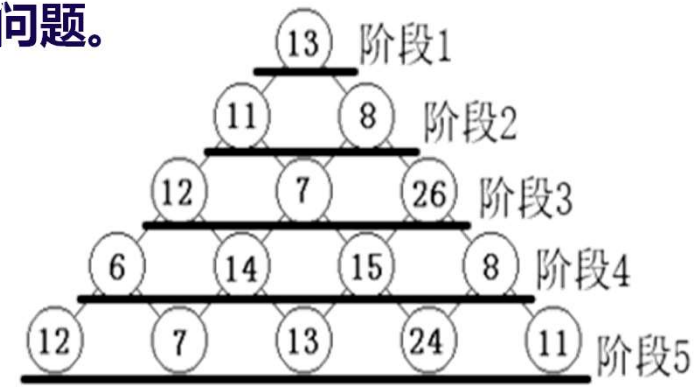
### 1.最优子结构

最优子结构是指**下一阶段的最优解**应该能够由前面各阶段**子问题的最优解**导出。

### 2.子问题重叠

子问题重叠是指不同阶段的状态或同一阶段的不同状态存在**相同的子问题**。DP将朴素搜索指数级复杂度变为多项式复杂度，其关键在计算过程中存储**中间状态**，以**空间换时间**，解决子问题重叠后的冗余问题。

| (i, j) | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 13 | | | | |
| 2 | 24 | 21 | | | |
| 3 | 36 | 31 | 47 | | |
| 4 | 42 | 50 | 62 | 55 | |
| 5 | 54 | 57 | 75 | 86 | 66 |



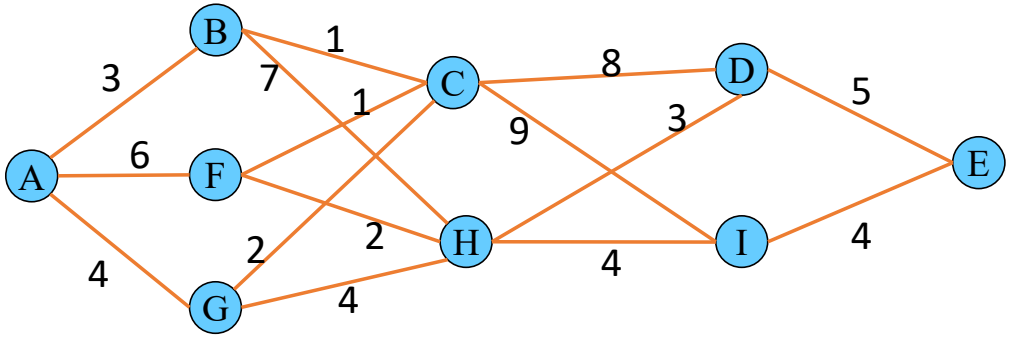
# 基础知识复习

## DP的求解三要素

### 3.无后效性

无后效性有**两层含义**，一是指某阶段的状态一旦确定，则此后过程的演变不再受此前各状态及决策的影响，即“未来与过去无关”，当前状态是**历史的完整总结**；二是指某阶段的状态一旦确定，则不受后续阶段状态及决策的影响，即“**当前与未来无关**”。

**【例题讲解01】如图所示，图中每条边上的数字表示该边的长度，则从A到E的最短距离是：_____ (2014NOIP初赛)**




动态规划求解：分五个阶段，最后求解得：D5(E):=16

**你能解释这是为什么?**

# 基础知识复习

## DP的实现方式

### 1.记忆化搜索（自顶向下）

自顶向下是指从原问题出发，逐步求解子问题，再由子问题的解合并成原问题的解，是一种从整体到局部的视角；

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[105][105];
int f[105][105];
//f[i][j]代表从顶点(1,1)走到(i,j)的最大路径和
int dfs(int x,int y){
    if(x==0)return 0;
    if(f[x][y]!=0xcfcfcfcf)return f[x][y];
    f[x][y]=max(dfs(x-1,y),dfs(x-1,y-1))+a[x][y];
    return f[x][y];
}
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=i;j++)cin>>a[i][j];
    }
    memset(f,0xcf,sizeof(f));
    int ans=-1e9;
    for(int j=1;j<=n;j++)ans=max(ans,dfs(n,j));
    cout<<ans<<endl;
    return 0;
}
```

# 基础知识复习

## DP的实现方式

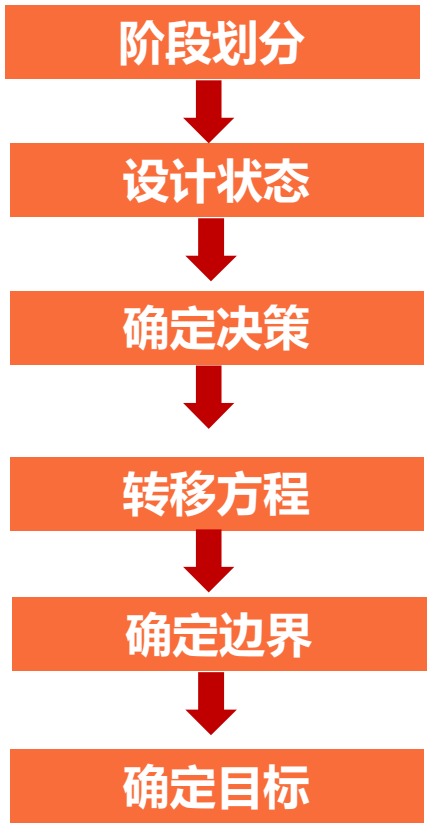
### 2.递推(自底向上)

自底向上则是指从边界、小规模问题出发，逐步推导到更大规模的问题，直到解决原问题，是一种从局部到整体的视角。

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int a[105][105];
4  int f[105][105];
5    //f[i][j]代表从顶点(1,1)走到(i,j)的最大路径和
6  int main(){
7      int n;
8      cin>>n;
9      for(int i=1;i<=n;i++){
10         for(int j=1;j<=i;j++)cin>>a[i][j];
11     }
12     f[1][1]=a[1][1];
13     for(int i=2;i<=n;i++){
14         for(int j=1;j<=i;j++){
15             f[i][j]=max(f[i-1][j],f[i-1][j-1])+a[i][j];
16             cout<<f[i][j]<<"";
17         }
18         cout<<endl;
19     }
20     int ans=-1e9;
21     for(int j=1;j<=n;j++)ans=max(ans,f[n][j]);
22     cout<<ans<<endl;
23     return 0;
24 }
```

# 基础知识复习

## DP的求解步骤



# 基础DP常见应用技巧

**应用技巧1：以求解次序或问题规模划分DP阶段。**

利用DP解决问题的过程中，阶段的划分是很关键的一步，阶段的划分直接影响问题求解的复杂性，阶段一般可按问题求解过程的先后次序或问题规模从小到大进行划分。以问题规模划分阶段时可以借鉴递归的思路将大问题拆解小问题，将小问题的逐步扩大依此划分DP阶段

**【例题01】核电站**（题目来源：OpenJudge9267）

**【问题描述】**一个核电站有N个放核物质的坑，坑排列在一条直线上。如果连续M个坑中放入核物质，则会发生爆炸，于是，在某些坑中可能不放核物质。

任务：对于给定的N和M，求不发生爆炸的放置核物质的方案总数。

**【输入格式】**只一行，两个正整数N，M( 1 < N < 50，2 ≤ M ≤ 5 )

**【输出格式】**一个正整数S，表示方案总数。

**【输入输出样例】**

| 样例输入 | 样例输出 |
| --- | --- |
| 4 3 | 13 |

# 基础DP常见应用技巧

**应用技巧1：以求解次序或问题规模划分DP阶段。**

**阶段划分**

**按照已处理的坑数将问题求解的过程划分为N个阶段，设阶段变量i代表已处理完前i个坑。**

**设计状态**

**设状态f(i,j)代表前i个坑，末尾放了j个核物质的方案数。**

**确定决策**

对于状态f[i][j]，问题进入了第i阶段，决策对象是第i个坑，可以做出的决策有：放和不放核物质。

**转移方程**

第i个坑不放核物质：f[i][0]+=f[i-1][j]

第i个坑放核物质：f[i][j]+=f[i-1][j-1]

**确定边界**

f[0][0]=1，即前i个坑末尾放了0个核物质的方案数为1。其余f[i][j]的值设为0。

**确定目标**

$\sum_{0<=j<=min(n,m-1)} f[n][j]$

前n个坑末尾放了j个核物质的方案数之和

**【01例题】核电站**（题目来源：OpenJudge9267）

**【问题描述】**一个核电站有N个放核物质的坑，坑排列在一条直线上。如果连续M个坑中放入核物质，则会发生爆炸，于是，在某些坑中可能不放核物质。

任务：对于给定的N和M，求不发生爆炸的放置核物质的方案总数。

**【输入格式】**只一行，两个正整数N，M( 1 < N < 50 ，2 ≤ M ≤ 5 )

**【输出格式】**一个正整数S，表示方案总数。

**【输入输出样例】**

| 样例输入 | 样例输出 |
| --- | --- |
| 4 3 | 13 |

# 基础DP常见应用技巧

**应用技巧2：依据问题阶段假设DP状态**

**利用DP解决问题的过程中，设计状态时，可直接将阶段变量先设为状态变量，若状态变量包含信息不足，再根据需要修改状态，定义或增加状态维度。**

**【例题02】最长上升子序列（题目来源： OpenJudge1759）**

**【问题描述】**一个数的序列$b_i$，当$b_1 < b_2 < \ldots < b_S$的时候，我们称这个序列是上升的。对于给定的一个序列$(a_1, a_2, \ldots, a_N)$，我们可以得到一些上升的子序列$(a_{i1}, a_{i2}, \ldots, a_{iK})$，这里$1 <= i_1 < i_2 < \ldots < i_K <= N$。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。

**【输入格式】**输入的第一行是序列的长度N (1 <= N <= 1000)。第二行给出序列中的N个整数，这些整数的取值范围都在0到10000。

**【输出格式】**最长上升子序列的长度。

**【输入输出样例】**

| 样例输入 | 样例输出 |
| --- | --- |
| 7<br>1 7 3 5 9 4 8 | 4 |

# 基础DP常见应用技巧

## 应用技巧2：依据问题阶段假设DP状态

**阶段划分** 设阶段变量i表示前i个数字

**设计状态** 设f[i]代表前i个数字必须选择a[i]能得到的最长上升子序列长度。

**确定决策** 对于状态f[i]，决策对象为数字a[i],讨论a[i]和前面的哪一个数去构成最长上升子序列，设决策变量为j，决策变量范围为1<=j<i且a[j]<a[i]。

**转移方程** f[i]=max(f[j]+1) (1<=j<i且a[j]<a[i])

**确定边界** 边界为f[i]=1

**确定目标** 目标解为max(f[i])

**【例题02】最长上升子序列（题目来源： OpenJudge1759）**

**【问题描述】**一个数的序列$b_i$，当$b_1<b_2<\ldots<b_S$的时候，我们称这个序列是上升的。对于给定的一个序列$(a_1,a_2,\ldots,a_N)$，我们可以得到一些上升的子序列$(a_{i1},a_{i2},\ldots,a_{iK})$，这里$1<=i_1<i_2<\ldots<i_K<=N$。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。

**【输入格式】**输入的第一行是序列的长度N (1 <= N <= 1000)。第二行给出序列中的N个整数，这些整数的取值范围都在0到10000。

**【输出格式】**最长上升子序列的长度。

**【输入输出样例】**

| 样例输入 | 样例输出 |
| --- | --- |
| 7<br>1 7 3 5 9 4 8 | 4 |

# 基础DP常见应用技巧

**应用技巧2：依据问题阶段假设DP状态**

对于状态f[i]，为了能说明数字a[i]的选择情况，我们还可**增加状态维度**

设状态**f[i][0/1]代表前i个数字第i个数字不选/选能得到的最长上升子序列长度**。那么有：

**f[i][0]=max(f[i-1][0],f[i-1][1]);**

**f[i][1]=max(f[j][1]+1)，1<=j<i且a[j]<a[i]。**

边界为f[i][0]=0，f[i][1]=1，即前i个数字第i个数字不选得到的最长上升子序列长度为0，第i个数字选得到的最长上升子序列长度为1。

目标解为**max(f[n][0],f[n][1]),**即前n个数字第n个数字不选/选得到的最长上升子序列长度的最大值。

# 基础DP常见应用技巧

**【例题03】公共子序列（题目来源： OpenJudge1808）**

**【问题描述】**我们称序列Z=<$z_1$,$z_2$,...,$z_k$>是序列X = < $x_1$, $x_2$, ..., $x_m$ >的子序列，当且仅当存在严格上升的序列<$i_1$,$i_2$,...,$i_k$>，使得对j = 1,2, ... ,k,有$x_{ij}$= $z_j$。比如Z = <a,b,f,c> 是X = <a,b,c,f,b,c>的子序列。

现在给出两个序列X和Y，你的任务是**找到X和Y的最大公共子序列**，也就是说要找到一个最长的序列Z，使得Z既是X的子序列也是Y的子序列。

**【输入格式】**输入包括多组测试数据。每组数据包括一行，给出两个长度不超过200的字符串，表示两个序列。两个字符串之间由若干个空格隔开。

**【输出格式】**对每组输入数据，输出一行，给出两个序列的最大公共子序列的长度。

**【输入输出样例】**

| 样例输入 | 样例输出 |
| --- | --- |
| abcfbc abfcab<br>programming contest<br>abcd mnp | 4<br>2<br>0 |

# 基础DP常见应用技巧

【例题03】公共子序列（题目来源： OpenJudge1808）

**阶段划分** 设阶段变量i代表a串的前i个字符，阶段变量j代表b串的前j个字符。

**设计状态** 设f[i][j]代表a串的前i个字符，b串的前j个字符构成的最大公共子序列长度。

**确定决策** 对于状态f[i][j]，决策对象为a[i]和b[j]，对a[i]和b[j]的值分类讨论即可。

**转移方程** 若a[i]=a[j]，则f[i][j]=f[i-1][j-1]+1；若a[i]≠a[j]，则f[i][j]=max(f[i-1][j],f[i][j-1])。

**确定边界** 边界为f[i][j]=0，1<=i<=n,1<=j<=m

**确定目标** 目标解为f[n][m]

# 基础DP常见应用技巧

**应用技巧3：采用向后递推和向前递推的方式确定DP方程。**

确定状态转移方程时，采用递推的思路，常见方式为**向后递推（填表法）**和**向前递推（刷表法）**。递推的前后方式是根据**问题推导的顺序定义**的。

**向后递推**是指考虑当前状态可以由此前阶段的哪些状态转移得到，即放置某一物品达到某状态是由上一阶段的哪个状态影响的；

**向前递推**则是指考虑当前状态可以更新后续阶段的哪些状态，即放置某一物品对状态的影响下一阶段的状态；向前递推造成结果，向后递推寻找原因，在实战训练中进一步从阶段间状态的变化体会转移方程向前递推与向后递推的思维过程。

# 基础DP常见应用技巧

| 样例输入 | 样例输出 |
| --- | --- |
| 5 7<br>1<br>2<br>3<br>4<br>5 | 14 |

**应用技巧3：采用向后递推和向前递推的方式确定DP方程。**

**【例题04】糖果**（题目来源：OpenJudge2989）

**【问题描述】**由于在维护世界和平的事务中做出巨大贡献，Dzx被赠予糖果公司2010年5月23日当天无限量糖果免费优惠券。在这一天，Dzx可以从糖果公司的N件产品中任意选择若干件带回家享用。糖果公司的N件产品每件都包含数量不同的糖果。Dzx希望他选择的产品包含的糖果总数是K的整数倍，这样他才能平均地将糖果分给帮助他维护世界和平的伙伴们。当然，在满足这一条件的基础上，糖果总数越多越好。Dzx最多能带走多少糖果呢?

注意：Dzx只能将糖果公司的产品整件带走。

**【输入格式】**第一行包含两个整数N(1<=N<=100)和K(1<=K<=100)，以下N行每行1个整数，表示糖果公司该件产品中包含的糖果数目，不超过1000000。

**【输出格式】**符合要求的最多能达到的糖果总数，如果不能达到K的倍数这一要求，输出0。

# 基础DP常见应用技巧

**阶段划分**

设阶段变量i表示前i件产品。

**设计状态**

设f[i][j]表示前i件商品选择若干使得糖果数量模k余数为j时最多能得到的糖果数量。

**确定决策**

对于状态f[i][j]，决策对象为第i件产品，决策集合为选或不选。

**转移方程**

**向后递推方式**，即当前阶段状态如何由上一阶段的状态更新。

不选第i件商品：f[i][j]=f[i-1][j]；选第i件商品：f[i][j]=f[i-1][((j-a[i])%k+k)%k]+a[i]。

向后递推时，第i件商品的决策带来的影响在第i阶段体现。

**向前递推方式**，即当前阶段的状态更新下一阶段的状态。

不选第i件商品：f[i+1][j]=f[i][j]；选第i件商品：f[i+1][(j+a[i])%k]=f[i][j]+a[i]。

向前递推时第i件商品的决策带来的影响在第i+1阶段才体现。

**确定边界**

**边界为f[0][0]=0，其余f[i][j]的值设为负无穷。**

**确定目标**

**向后递推方式**：目标解为f[n][0]，即前n件产品选择选择若干使得糖果数量模k余数为0时最多能得到的糖果数量。

**向前递推方式**：目标解为f[n+1][0]，即前n+1件产品选择选择若干使得糖果数量模k余数为0时最多能得到的糖果数量。

# 基础DP常见应用技巧

## 向后递推（填表法）

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[105];
int f[105][105];
//f[i][j]代表前i件商品选择若干使得糖果数量模k余数为j时最多能得到的糖果数量
int main(){
    int n,k;
    cin>>n>>k;
    for(int i=1;i<=n;i++)cin>>a[i];
    memset(f,0xcf,sizeof(f));
    f[0][0]=0;
    for(int i=1;i<=n;i++){
        for(int j=0;j<k;j++){
            f[i][j]=max(f[i-1][j],f[i-1][((j-a[i])%k+k)%k]+a[i]);//逆推
        }
    }
    if(f[n][0]<0)cout<<0<<endl;
    else cout<<f[n][0]<<endl;
    return 0;
}
```

## 向前递推（刷表法）

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[105];
int f[105][105];
 //f[i][j]代表前i件商品选择若干使得糖果数量
 //模k余数为j时最多能得到的糖果数量
int main(){
    int n,k;
    cin>>n>>k;
    for(int i=1;i<=n;i++)cin>>a[i];
    memset(f,0xcf,sizeof(f));
    f[0][0]=0;
    for(int i=0;i<=n;i++){//顺推时，阶段从0开始
        for(int j=0;j<k;j++){
            f[i+1][j]=max(f[i+1][j],f[i][j]);
            f[i+1][(j+a[i])%k]=max(f[i+1][(j+a[i])%k],f[i][j]+a[i]);
        }
    }
    if(f[n+1][0]<0)cout<<0<<endl;
    else cout<<f[n+1][0]<<endl;
    return 0;
}
```

# 基础DP常见应用技巧

**应用技巧4：巧设数组，解决DP问题重构解。**

**【例题05】机器分配 （题目来源：洛谷P2066）**

**【问题描述】**总公司拥有高效设备M台，准备分给下属的N个分公司。各分公司若获得这些设备，可以为国家提供一定的盈利。问：如何分配这M台设备才能使国家得到的盈利最大？求出最大盈利值。其中M≤15，N≤10。分配原则：每个公司有权获得任意数目的设备，但总台数不超过设备数M。

**【输入格式】**第一行有两个数，第一个数是分公司数N，第二个数是设备台数M。

接下来是一个N*M的矩阵，表明了第 I个公司分配 J台机器的盈利。

**【输出格式】**第1行为最大盈利值；

第2到第n为第i分公司分x台，要求答案的字典序最小。

| 样例输入 | 样例输出 |
| --- | --- |
| 3 3<br>30 40 50<br>20 30 50<br>20 25 30 | 70<br>1 1<br>2 1<br>3 1 |

# 基础DP常见应用技巧

**应用技巧4：巧设数组，解决DP问题重构解。**

**【例题】机器分配 （题目来源：洛谷P2066）**

**阶段划分** 设阶段变量i代表前i家公司。

**设计状态** 设f[i][j]代表前i加公司分配j台设备能产生的最大盈利。

**确定决策** 对于状态f[i][j]，决策对象为第i家公司，做出的决策即为i家公司分配多少台设备。设决策变量为k，那么决策变量范围为0<=k<=j。

**转移方程** f[i][j]=max(f[i-1][k]+a[i][k])（1<=j<=m,0<=k<=j）。

**确定边界** f[i][0]=0，1<=i<=n，即前i家公司分配0台设备能产生的最大盈利为0。

**确定目标** 目标解为f[n][m]，即前n家公司分配m台设备能产生的最大盈利。关于输出方案，需要重构解。可用g[i][j]维护当状态f[i][j]取得最优解的决策k，然后递归输出即可。

# 多维动态规划

**【题目06】乌龟棋**（题目来源：NOIP2010提高组）

**【问题描述】**乌龟棋的棋盘是一行N个格子，每个格子上一个分数（非负整数）。棋盘第1格是唯一的起点，第N格是终点，游戏要求玩家控制一个乌龟棋子从起点出发走到终点。

乌龟棋中M张爬行卡片，分成4种不同的类型（M张卡片中不一定包含所有4种类型的卡片，见样例），每种类型的卡片上分别标有1,2,3,4四个数字之一，表示使用这种卡片后，乌龟棋子将向前爬行相应的格子数。游戏中，玩家每次需要从所有的爬行卡片中选择一张之前没有使用过的爬行卡片，控制乌龟棋子前进相应的格子数，每张卡片只能使用一次。

游戏中，乌龟棋子自动获得起点格子的分数，并且在后续的爬行中每到达一个格子，就得到该格子相应的分数。玩家最终游戏得分就是乌龟棋子从起点到终点过程中到过的所有格子的分数总和。很明显，用不同的爬行卡片使用顺序会使得最终游戏的得分不同，小明想要找到一种卡片使用顺序使得最终游戏得分最多。

现在，告诉你棋盘上每个格子的分数和所有的爬行卡片，你能告诉小明，他最多能得到多少分吗？

**【输入格式】**每行中两个数之间用一个空格隔开。

第1行2个正整数N,M，分别表示棋盘格子数和爬行卡片数。

第2行N个非负整数，a1,a2,...,an，其中ai表示棋盘第i个格子上的分数。

第3行M个整数，b1,b2,..,bm，表示M张爬行卡片上的数字。

输入数据保证到达终点时刚好用光M张爬行卡片。

**【输出格式】**1个整数，表示小明最多能得到的分数。

| 样例输入 | 样例输出 |
| --- | --- |
| 9 5<br>6 10 14 2 8 8 18 5 17<br>1 3 1 2 1 | 73 |

# 多维动态规划

**【题目06】乌龟棋**（题目来源：NOIP2010提高组）假设字母A，B，C，D代表四种卡片（卡片数字为1,2,3,4）的总张数，a，b，c，d代表四种卡片已使用的张数，x[i]代表位置i的分数。

**阶段划分** 设阶段变量i代表使用了i张卡片（或跳了i次）。

**设计状态** 当乌龟使用某张卡片从一个位置跳到下一个位置时，变化的客观条件有：位置、4种卡片的已使用张数，因此设f[i][j][a][b][c][d]代表乌龟跳了i次，位于位置j，4种卡片分别使用了a，b，c，d张得到的最多分数

首先，阶段变量i=a+b+c+d，所以i可以省略；其次j=1a+2b+3c+4d，所以j可以省略。因此修改状态为：设f[a][b][c][d]代表4种卡片分别使用了a，b，c，d张得到的最多的分数。

**确定决策** **对于状态f[a][b][c][d]，上一次使用的卡片可能是第1种，第2种，第3种，第4种。**

**转移方程** **f[a][b][c][d]=max(f[a-1][b][c][d],f[a][b-1][c][d],f[a][b][c-1][d],f[a][b][c][d-1])+x[1+1*a+2*b+3*c+4*d]**

**确定边界** **f[a][b][c][d]=-∞，f[0][0][0][0]=x[1],**即玩家最开始站在位置1，各种卡片已使用张数均为0张，初始获得分数为x[1]

**确定目标** 目标解为**f[A][B][C][D]**

# 多维动态规划

**【题目06】乌龟棋**（题目来源：NOIP2010提高组）

**// f[a][b][c][d]代表4种卡片分别使用了a，b，c，d张得到的最多的分数**

```
cin>>n>>m;
for(int i=1;i<=n;i++)cin>>x[i];
for(int i=1;i<=m;i++){
    cin>>c;
    card[c]++;
}
memset(f,0xcf,sizeof(f));
f[0][0][0][0]=x[1];
for(int a=0;a<=card[1];a++){
    for(int b=0;b<=card[2];b++){
        for(int c=0;c<=card[3];c++){
            for(int d=0;d<=card[4];d++){
                int score=x[1+1*a+2*b+3*c+4*d];
    if(a>=1)f[a][b][c][d]=max(f[a][b][c][d],f[a-1][b][c][d]+score);
    if(b>=1)f[a][b][c][d]=max(f[a][b][c][d],f[a][b-1][c][d]+score);
    if(c>=1)f[a][b][c][d]=max(f[a][b][c][d],f[a][b][c-1][d]+score);
    if(d>=1)f[a][b][c][d]=max(f[a][b][c][d],f[a][b][c][d-1]+score);
            }
        }
    }
}
cout<<f[card[1]][card[2]][card[3]][card[4]];
```

# 多维动态规划

**应用技巧1：从阶段出发巧设DP状态。**

解决DP问题设计状态时，先划分阶段，可先直接将阶段变量设为状态变量，若状态变量包含信息不足，再根据需要增加状态维度。具体来讲，一方面是观察所求问题受哪些维度限制，另一方面是思考从某个阶段进入到下一阶段时哪些客观条件在发生变化。

**【题目07】子串（NOIP2015提高组）**

**【题目描述】**有两个仅包含小写英文字母的字符串 A和 B。

现在要从字符串 A中取出 k个互不重叠的非空子串，然后把这 k个子串按照其在字符串 A中出现的顺序依次连接起来得到一个新的字符串。请问有多少种方案可以使得这个新串与字符串 B 相等?

注意：子串取出的位置不同也认为是不同的方案。

**【输入格式】**

第一行是三个正整数 n,m,k，分别表示字符串 A的长度，字符串 B的长度，以及问题描述中所提到的 k，每两个整数之间用一个空格隔开。

第二行包含一个长度为 n的字符串，表示字符串 A。

第三行包含一个长度为 m的字符串，表示字符串 B。

**【输出格式】**一个整数，表示所求方案数。

由于答案可能很大，所以这里要求输出答案对 1000000007取模的结果。

# 多维动态规划

**应用技巧1：从阶段出发巧设DP状态。**

## 【题目07】子串（NOIP2015提高组）

全国信息学奥林匹克联赛（NOIP2015）复赛　　提高组 day2

【输入输出样例说明】
所有合法方案如下：（加下划线的部分表示取出的子串）
样例 1：**aab** aab / aab **aab**
样例 2：**a ab** aab / **a** aba **ab** / a **a** ba **ab** / aab **a ab**
**aa b** aab / **aa** baa **b** / aab **aa b**
样例 3：**a a b** aab / **a a** baa **b** / **a** ab **a** a **b** / **a** aba **a b**
a **a** b **a** a **b** / a **a** ba **a b** / aab **a a b**

【输入输出样例 4】
见选手目录下 substring/substring4.in 与 substring/substring4.ans。

【数据规模与约定】
对于第 1 组数据：1≤n≤500，1≤m≤50，k=1；
对于第 2 组至第 3 组数据：1≤n≤500，1≤m≤50，k=2；
对于第 4 组至第 5 组数据：1≤n≤500，1≤m≤50，k=m；
对于第 1 组至第 7 组数据：1≤n≤500，1≤m≤50，1≤k≤m；
对于第 1 组至第 9 组数据：1≤n≤1000，1≤m≤100，1≤k≤m；
对于所有 10 组数据：1≤n≤1000，1≤m≤200，1≤k≤m。

# 多维动态规划

**应用技巧1：从阶段出发巧设DP状态。**

**【题目07】子串（NOIP2015提高组）** 假设字符串A、B长度分别为n，m，下标均从1开始。

**阶段划分** 对A串从前向后逐一考虑每个字符，按A串已处理的字符数量划分阶段，设阶段变量i代表已处理A串的前i个字符。

**设计状态** 设f[i][j][k]代表从A串的前i个字符选择k个子串组成的字符串和B串的前j个字符一样的方案数。

**确定决策** 对于状态f[i][j][k]，决策对象为字符A[i]，对它做出的可能决策有：不选，选（由于是选择一个子串，a[i]可以和之前的某些字符一起选形成一个子串）。

**转移方程**

**若不选A[i]，那么f[i][j][k]=f[i-1][j][k];**

**若选择了A[p]...A[i]这一子串，该子串长度为i-p+1，那么f[i][j][k]=f[p-1][j-(i-p+1)][k-1]。**

**综上：f[i][j][k]=f[i-1][j][k]+f[p-1][j-(i-p+1)][k-1]，k<=p<=i。**

**确定边界** **边界为f[i][j][k]=0，f[i][0][0]=1，即从A的前i个字符中选择0个子串和B的前0个字符一样的方案数为1，其余状态的方案数为0。**

**确定目标** 目标解为f[n][m][k]，即从A的前n个字符中选择k个子串和B的前m个字符一样的方案数。

# 多维动态规划

**应用技巧1：从阶段出发巧设DP状态。**

**【题目07】子串（NOIP2015提高组）**

假设字符串A、B长度分别为n，m，下标均从1开始。

**【优化思考】**选择字符A[i]时，若只选A[i]这个子串，则f[i][j][k]+=f[i-1][j-1][k-1]；若是让A[i]和之前的某些字符一起选，那么A[i]至少会和A[i-1]一起选，至于A[i-1]是单独选还是和之前的字符一起选，之前已经处理过了。

修改状态，设f[i][j][k][0/1]代表从A串的前i个字符选择k个子串组成的字符串和B串的前j个字符一样且A[i]不选/选的方案数。若不选A[i]，那么f[i][j][k][0]=f[i-1][j][k][0]+f[i-1][j][k][1]；若选A[i]，前提是A[i]=B[j]，分两种情况讨论：

（1）只选A[i]，那么f[i][j][k][1]=f[i-1][j-1][k-1][0]+f[i-1][j-1][k-1][1]；

（2）A[i]和之前的字符一起选，那么f[i][j][k][1]=f[i-1][j-1][k][1]；

所以[i][j][k][1]=f[i-1][j-1][k-1][0]+f[i-1][j-1][k-1][1]+f[i-1][j-1][k][1]。

边界为f[i][j][k][0/1]=0，f[i][0][0][0]=1，即从A串的前0个字符中选0个非空子串得到B的前0个字符，A[0]不选的方案数为1，其余所有状态的方案数为0。

**目标解为f[n][m][k][0]+f[n][m][k][1]**，即从A的前n个字符中取出k个非空子串得到B的前m个字符，A[n]选和不选的方案数之和。

# 多维动态规划

**应用技巧1：从阶段出发巧设DP状态。**

**//f[i][j][k][0/1]代表从A的前i个字符中取出k个非空子串得到B的前j个字符，第i个字符选/不选的方案数。**

```
int main(){
    int n,m,K;
    cin>>n>>m>>K;
    cin>>a+1>>b+1;
    for(int i=0;i<=n;i++)f[i%2][0][0][0]=1;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            for(int k=0;k<=K;k++){
                //滚动数组时使用了之前的解，先初始化为0
                f[i%2][j][k][0]=f[i%2][j][k][1]=0;
                //不选第i个字符
                f[i%2][j][k][0]=(f[(i-1)%2][j][k][0]+f[(i-1)%2][j][k][1])%mod;
                //选第i个字符
                if(a[i]==b[j]){
                    f[i%2][j][k][1]=(f[i%2][j][k][1]+f[(i-1)%2][j-1][k][1])%mod;
                    if(k>=1)
                        f[i%2][j][k][1]=(f[i%2][j][k][1]+f[(i-1)%2][j-1][k-1][0]+f[(i-1)%2][j-1][k-1][1])%mod;
                }
                else f[i%2][j][k][1]=0;//滚动数组优化后，这里要赋0
            }
        }
    }
    cout<<(f[n%2][m][K][0]+f[n%2][m][K][1])%mod<<endl;
    return 0;
}
```

# 多维动态规划

## 应用技巧2：巧设状态，降低DP维度

设计DP状态时，分析选择最少维度的状态变量，进而分析DP状态下的不同维度之间的关系，若状态变量中的某一维可以由其它维度导出，则可以省略这一维度，以降低DP维度。

**【题目08】传纸条**（题目来源：NOIP2008提高组 ）

**【问题描述】**小渊和小轩是好朋友也是同班同学，他们在一起总有谈不完的话题。一次素质拓展活动中，班上同学安排做成一个 m 行 n 列的矩阵，而小渊和小轩被安排在矩阵对角线的两端，因此，他们就无法直接交谈了。幸运的是，他们可以通过传纸条来进行交流。纸条要经由许多同学传到对方手里，小渊坐在矩阵的左上角，坐标(1,1)，小轩坐在矩阵的右下角，坐标 (m,n)。从小渊传到小轩的纸条只可以向下或者向右传递，从小轩传给小渊的纸条只可以向上或者向左传递。

在活动进行中，小渊希望给小轩传递一张纸条，同时希望小轩给他回复。班里每个同学都可以帮他们传递，但只会帮他们一次，也就是说如果此人在小渊递给小轩纸条的时候帮忙，那么在小轩递给小渊的时候就不会再帮忙。反之亦然。

还有一件事情需要注意，全班每个同学愿意帮忙的好感度有高有低（注意：小渊和小轩的好心程度没有定义，输入时用 0 表示），可以用一个[0,100] 内的自然数来表示，数越大表示越好心。小渊和小轩希望尽可能找好心程度高的同学来帮忙传纸条，即找到来回两条传递路径，使得这两条路径上同学的好心程度之和最大。现在，请你帮助小渊和小轩找到这样的两条路径。

**【输入格式】**第一行有两个用空格隔开的整数m 和n，表示班里有 m 行 n 列。

接下来的 m 行是一个m×n 的矩阵，矩阵中第 i 行 j 列的整数表示坐在第 i 行 j 列的学生的好心程度。每行的 n 个整数之间用空格隔开。

**【输出格式】**输出文件共一行一个整数，表示来回两条路上参与传递纸条的学生的好心程度之和的最大值。

# 多维动态规划

| 样例输入 | 样例输出 |
| --- | --- |
| 3 3<br>0 3 9<br>2 8 5<br>5 7 0 | 34 |

## 应用技巧2：巧设状态，降低DP维度

**【题目08】传纸条**（题目来源：NOIP2008提高组 ）

**阶段划分** 行走过程可以按照已走的步数将两人行走过程划分为n+m-2个阶段，设阶段变量i代表两人已走了i步。

**设计状态** 设**f[i][x1][y1]**代表两人已走了**i**步，小渊位于第**x1**行，小轩位于第**x2**行两人得到的最大路径和。

**确定决策** 对于状态**f[i][x1][x2]**，由于小渊可以选择向下、向右走，小轩也可以选择向下、向右走，可能的决策有**4**种组合：下下，下右，右下，右右。

**转移方程** 对于状态f[i][x1][x2]，小渊位于(x1,i+2-x1)，小轩位于(x2,i+2-x2)，

若x1=x2且　i+2-x1=i+2-x2，设val=a[x1][i+2-x1]；否则val=a[x1][i+2-x1]+a[x2][i+2-x2]。

则f[i][x1][x2]=max(f[i-1][x1-1][x2-1],f[i-1][x1-1][x2],f[i-1][x1][x2-1],f[i-1][x1][x2])+val。

**确定边界** f[i][x1][x2]=-∞，f[0][1][1]=a[1][1]，即初始时两人走了0步，小渊位于第1行，小轩位于第1行两人的最大路径和为0，其余值为-∞代表该状态暂时无解。

**确定目标** 目标解为f[n+m-2][m][m]，即两人走了n+m-2步，小渊位于第m行，小轩位于第m行两人得到的最大路径和。

# 多维动态规划

**应用技巧1：从阶段出发巧设DP状态。**

**【题目08】传纸条**（题目来源：NOIP2008提高组 ）

| 样例输入 | 样例输出 |
| --- | --- |
| 3 3<br>0 3 9<br>2 8 5<br>5 7 0 | 34 |

```
int maxx(int i,int j,int x,int y){
    return max(max(i,j),max(x,y));
}
int a[105][105],f[105][105][105];
//f[i][x1][y1]代表两人已走了i步，小渊位于第x1行，小轩位于第x2行两人得到的最大路径和。
int main(){
    int m,n,tmp;
    cin>>m>>n;
    for(int i=1;i<=m;i++)
    for(int j=1;j<=n;j++)
    cin>>a[i][j];
    f[0][1][1]=a[1][1];
    for(int k=1;k<=n+m-2;k++){
        for(int x1=1;x1<=m;x1++){
            for(int x2=1;x2<=m;x2++){
                if(k+2-x1<=0||k+2-x2<=0)break;
                if(x1==x2)tmp=a[x1][k+2-x1];
                else tmp=a[x1][k+2-x1]+a[x2][k+2-x2];
                f[k][x1][x2]=maxx(f[k-1][x1][x2],f[k-1][x1-1][x2],f[k-1][x1][x2-1],f[k-1][x1-1][x2-1])+tmp;

            }
        }
    }
    cout<<f[m+n-2][m][m];
    return 0;
}
```

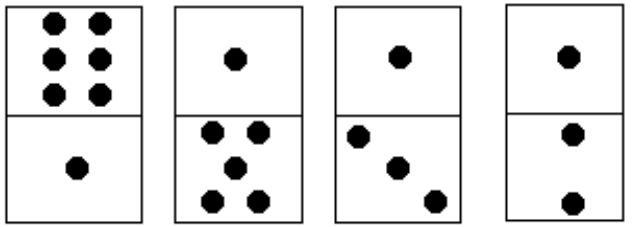
# 多维动态规划

**应用技巧3：以最优性问题设定可行性状态。**

**若题目所求最优解有两种及以上，可以尝试将其中几种最优解作为状态变量参与状态设定。**

**【题目09】多米诺骨牌**（题目来源：洛谷P1282）

**【问题描述】**多米诺骨牌有上下2个方块组成，每个方块中有1~6个点。现有排成行的上方块中点数之和记为S1，下方块中点数之和记为S2，它们的差为|S1-S2|。例如在图8-1中，S1=6+1+1+1=9，S2=1+5+3+2=11，|S1-S2|=2。每个多米诺骨牌可以旋转180°，使得上下两个方块互换位置。 编程用最少的旋转次数使多米诺骨牌上下2行点数之差达到最小。



对于图中的例子，只要将最后一个多米诺骨牌旋转180°，可使上下2行点数之差为0。

**【输入格式】**输入文件的第一行是一个正整数n(1≤n≤1000)，表示多米诺骨牌数。接下来的n行表示n个多米诺骨牌的点数。每行有两个用空格隔开的正整数，表示多米诺骨牌上下方块中的点数a和b，且1≤a，b≤6。

**【输出格式】**输出文件仅一行，包含一个整数。表示求得的最小旋转次数。

| 样例输入 | 样例输出 |
| --- | --- |
| 4<br>6 1<br>1 5<br>1 3<br>1 2 | 1 |

# 多维动态规划

**应用技巧3：以最优性问题设定可行性状态。**

**【题目09】多米诺骨牌**（题目来源：洛谷P1282）

**阶段划分**

**按已处理的骨牌数量将问题求解过程划分为n个阶段，设阶段变量i表示前i个骨牌。**

**设计状态**

设f[i][j]代表前i个骨牌旋转j次得到的最小的上下两行点数和之差，该状态不具有最优子结构，因为第i阶段的最小点数和之差并不能有第i-1阶段最小的点数和之差转移。

设**f[i][j]**代表前i个骨牌上下两行点数和之差为j时最少的旋转次数，差值j可能为负数，不是很好处理。

**设f[i][j]代表前i个骨牌第一行骨牌之和为j时最少的旋转次数。**

**转移方程**

**对于状态f[i][j]，问题进入第i阶段，决策对象为第i个骨牌，可以做出的决策为转或不转。**

**确定决策**

**若不转第i个骨牌：f[i][j]=f[i-1][j-a[i]]，a[i]为第i个骨牌上方点数。**

**若转第i个骨牌：f[i][j]=f[i-1][j-b[i]]+1，b[i]为第i个骨牌下方点数。f[i][j]=min(f[i-1][j-a[i]),f[i-1][j-b[i]]+1)。**

**确定目标**

**枚举f[n][j] 其中j的取值范围为0<=j<=6*n**

**确定边界**

**f[i][j]=+∞，f[0][0]=0**

# 多维动态规划

**应用技巧3：以最优性问题设定可行性状态。**

**【题目09】多米诺骨牌**（题目来源：洛谷P1282）

若f[n][j]不等于无穷大。则上下两行点数和之差为abs(j-(sum-j))，sum代表n个骨牌点数之和，再用该差值更新点数和之差最小值的同时，更新最少旋转次数。

//f[i][j]代表前i个骨牌第一行骨牌之和为j时最少的旋转次数

```
5  int main(){
6      int n,sum=0;
7      cin>>n;
8      for(int i=1;i<=n;i++){
9          cin>>a[i]>>b[i];
10         sum+=a[i];
11         sum+=b[i];
12     }
13     memset(f,0x3f,sizeof(f));
14     f[0][0]=0;
15     for(int i=1;i<=n;i++){
16         for(int j=0;j<=n*6;j++){
17             if(j>=a[i])f[i][j]=f[i-1][j-a[i]];
18             if(j>=b[i])f[i][j]=min(f[i][j],f[i-1][j-b[i]]+1);
19         }
20     }
21     int ans1=1e9,ans2=1e9;//记录最小差值和最少旋转次数
22     for(int j=0;j<=6*n;j++){
23         if(f[n][j]!=0x3f3f3f3f){
24             if(abs(j-(sum-j))<ans1){
25                 ans1=abs(j-(sum-j));
26                 ans2=f[n][j];
27             }
28             else if(abs(j-(sum-j))==ans1){
29                 ans2=min(ans2,f[n][j]);
30             }
31         }
32     }
33     cout<<ans2<<endl;
```

# 多维动态规划

**应用技巧4：转化DP状态，降低空间复杂度。**

DP问题中对DP状态常通过开辟数组进行存储， 然后遇到状态变量取值范围太大会导致在给定空间复杂度限制下无法开辟相应空间的数组，为节省空间，可以交换DP问题最优解与状态变量，或者对状态变量范围作离散化处理。

**【题目10】Parking Ships**（题目来源：HDU4205）

**【问题描述】**有n（n<=1000）个海盗，每个海盗有一条船，长度为Li（1<=li<=$10^9$）。他们的家沿着海边排成一条直线，每一个海盗家的坐标为xi（-$10^9$<=xi<=$10^9$）。他们会将船也停在海边（船相当于一个闭区间，两两不能相交）。如果某个海盗的船停在家门口（即他家的坐标包含在船所停的区间中），他会很高兴。

有个海盗是老大，他一定要自己的船放在家门口正当中（即他家坐标是他的船的区间中点）。请问怎样安排这些船只停靠才能让最多能让多少的海盗开心呢?

**【输入格式】**第1行有一个整数，代表测试数据组数。对于每组测试数据而言包含2行，其中：

第1行有一个整数n，代表海盗的个数。

第2-n行，每行包含两个整数xi和li，含义如题目所述,第一对xi，li即海盗老大的家的坐标和船的长度。

**【输出格式】**对每组测试数据，输出一个整数，代表最多能让多少海盗开心。你可以假设海岸是无限长的。

| 样例输入 | 样例输出 |
| --- | --- |
| 2<br>5<br>0 6<br>-5 2<br>-4 1<br>4 2<br>5 3<br>4<br>0 4<br>-5 4<br>3 4<br>5 3 | 5<br>3 |

# 多维动态规划

**应用技巧4：转化DP状态，降低空间复杂度。**

DP问题中对DP状态常通过开辟数组进行存储， 然后遇到状态变量取值范围太大会导致在给定空间复杂度限制下无法开辟相应空间的数组，为节省空间，可以交换DP问题最优解与状态变量，或者对状态变量范围作离散化处理。

**【题目10】Parking Ships**（题目来源：HDU4205）

**阶段划分**

**对所有海盗按家的坐标xi从小到大排序，我们可以逐一处理每一个海盗的船只应该如何放置，因此按已处理的海盗数量划分阶段，设阶段变量i代表前i个海盗。**

**设计状态**

**设f[i][j]代表前i个海盗第i个海盗船右端坐标为j时能让海盗开心的最大人数。**

**设f[i][j]代表前i个海盗必须要让j个海盗开心最右的船右端点坐标的最小值。**

**确定决策**

对于状态f[i][j]，问题进入第i阶段，决策对象为第i个海盗，可以做出的决策是不让他开心（忽略他的船只）、让他开心（将船只停在他家门口）。

**转移方程**

若不让第i个海盗开心，则f[i][j]=f[i-1][j]。注意，如果第i个海盗是船长，则必须要让他开心。

若要让第i个海盗开心，如果f[i-1][j-1]>=xi，此时是无法让第i个海盗开心的；

如果f[i-1][j-1]+li<xi，则f[i][j]=xi；否则f[i][j]=f[i-1][j-1]+li。

**确定边界**

若排序后船长在位置1，则f[1][1]=a[1].x+(a[1].l+1)/2；

若船长不在位置1，则f[1][0]=-∞，f[1][1]=a[1].x，其余f[i][j]=+∞。

**确定目标**

若f[n][j]!=+∞，则j就是最多能开心的海盗数，输出最大的j即可

# 多维动态规划

**应用技巧4：转化DP状态，降低空间复杂度。**

**【题目10】Parking Ships**（题目来源：HDU4205）

**//f[i][j]**代表前i个海盗必须要让j个海盗开心最右边的船右端点坐标的最小值。

```
struct node{
    int x,l,no;
}a[1010];
bool cmp(node p,node q){
    if(p.x!=q.x)return p.x<q.x;
    else return q.l<p.l;
}
```

```
while(t--){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i].x>>a[i].l,a[i].no=i;
    x0=a[1].x;l0=a[1].l;
    sort(a+1,a+1+n,cmp);
    for(int i=1;i<=n;i++){
        if(a[i].no==1)pos=i;
        for(int j=0;j<=n;j++)
            f[i][j]=inf;
    }
    if(pos==1)f[1][1]=x0+(l0+1)/2;
    else{
        f[1][0]=-inf;
        f[1][1]=a[1].x;
    }
    for(int i=2;i<=n;i++){
        if(i==pos){
            for(int j=1;j<=i;j++)
            if(f[i-1][j-1]<=x0-(l0+1)/2)f[i][j]=x0+(l0+1)/2;
            continue;
        }
        for(int j=0;j<=i;j++){
            //不让第i个海盗开心
            f[i][j]=f[i-1][j];
            //让第i个海盗开心
            if(j>=1&&f[i-1][j-1]<=a[i].x){
        f[i][j]=min(f[i][j],max(a[i].x,f[i-1][j-1]+a[i].l));
            }
        }
    }
    for(int j=n;j>=1;j--){
        if(f[n][j]!=inf){
            cout<<j<<endl;
            break;
        }
    }
}
```