

# 中山大学数据科学与计算机学院本科生实验报告

## (2016 学年春季学期)

课程名称: Algorithm design

任课教师: 张子臻

年级	15	专业(方向)	软件工程(移动信息工程)
学号	15352408	姓名	张稼伟
电话	13531810182	Email	709075442@qq.com
开始日期		完成日期	

### 目录

1.实验题目 .....	2
Soj 1176 Two Ends .....	2
Soj 1011 Lenny's Lucky Lotto .....	3
Soj 1121 Tri Tiling .....	3
Soj 1264 Atomic Car Race .....	3
Soj 1828 Minimal .....	3
Soj 1527 Tiling a Grid With Dominoes .....	3
Soj 1148 过河 .....	3
Soj 1163 Tour .....	4
Soj 1345 能量项链 .....	4
Soj 1687 Permutation .....	4
Soj 13062 SubDiagonal Paths .....	4
2.实验目的 .....	4
3.程序设计 .....	5
Soj 1176 Two Ends .....	5

Soj 1011 Lenny's Lucky Lotto .....	6
Soj 1121 Tri Tiling.....	6
Soj 1264 Atomic Car Race .....	8
Soj 1828 Minimal .....	9
Soj 1527 Tiling a Grid With Dominoes.....	10
Soj 1148 过河 .....	12
Soj 1163 Tour .....	13
Soj 1345 能量项链.....	14
Soj 1687 permutation .....	15
Soj 13062 SubDiagonal Paths .....	16
5.实验总结与心得 .....	17
附录、提交文件清单 .....	17

## 1.实验题目

### DP 专题

#### Soj 1176 Two Ends

**题意：**首先输入  $n(n \leq 1000)$ ,  $n$  为偶数，接着输入  $n$  个整数， $n$  个整数的和不超过 1,000,000.两个人每次只能从两端取数，第一个人 A 可以用任意策略，第二个人 B 用贪心策略取两段的较大数(左右数相等取左数)。求两人取数和之差的最大值。

### Soj 1011 Lenny's Lucky Lotto

**题意：**有一个  $N(N \leq 10)$  位数，用  $1-M(1 \leq M \leq 2000)$  去填，后一位数字至少是前一位数字的两倍，问有多少个这样的  $N$  位数。

### Soj 1121 Tri Tiling

**题意：**用  $2 \times 1$  的矩形去填满一个  $3 \times n$  的大矩形，有多少种填法。

$$0 \leq n \leq 30$$

### Soj 1264 Atomic Car Race

**题意：**有  $n$  个地点，分别距离起始点  $a_0$  的距离为  $a_1, a_2, \dots, a_n$ 。行驶时间由轮胎决定，新轮胎行驶从距离  $x$  行驶到  $x+1$  的时间开销关系如下：

$$1/(v - e * (x - r)) \text{ (if } x \geq r \text{)}$$

$$1/(v - f * (r - x)) \text{ (if } x < r \text{)}$$

在这  $n$  个地点你可以选择任意地点换轮胎，每次换胎增加  $b$  的时间开销。求从起始点到终点  $n$  的最短时间。

### Soj 1828 Minimal

**题意：**将一个有  $N+M$  个数的大集合  $\{x | 0 \leq x \leq 1000000, x \text{ 为整数}\}$  分成两个集合  $S_1$  和  $S_2$ ，其中  $S_1$  有  $N$  个元素， $S_2$  有  $M$  个元素。现给出  $S_1$  和  $S_2$ ，求  $F(S_1, S_2)$ ，满足  $F(S_1, S_2) = \min\{|a_1 - b_1| + |a_2 - b_2| + \dots + |a_n - b_n|\}$

其中  $a_i \in S_1, b_i \in S_2$

$$a_i \neq a_j \text{ if } i \neq j$$

$$b_i \neq b_j \text{ if } i \neq j$$

$$(i, j = 1, 2 \dots N, N = |S_1|)$$

### Soj 1527 Tiling a Grid With Dominoes

**题意：**有一个  $4 \times n$  的棋盘，无限个  $1 \times 2$  的多米诺骨牌。输入  $n$ ，求把棋盘完美覆盖的方法数，题目保证结果不超过 32 位整型。

### Soj 1148 过河

**题意：**一只青蛙想沿着独木桥从河的一侧跳到另一侧。在桥上有一些石子，青蛙很讨厌踩在这些石子上。桥的长度和青蛙一次跳过的距离都是正整数，把独木桥看成数轴上的一串整点：0, 1, …,  $L$ （其中  $L$  是桥的长度）。0 表示起点， $L$  表示终点。青蛙一次跳跃的距离是  $S$  到  $T$  之间的任意正整数（包括  $S, T$ ）。当青蛙跳到或跳过坐标为  $L$  的点时，就算青蛙已经跳出了独木桥。

题目给出独木桥的长度  $L$ ，青蛙跳跃的距离范围  $S, T$ ，桥上石子的位置。你的任务是确定青蛙要想过河，最少需要踩到的石子数。

约束： $1 \leq L \leq 10^9, 1 \leq S \leq T \leq 10, 1 \leq M \leq 100$ 。

## Soj 1163 Tour

**题意：**给定平面上  $n$  个点 ( $n \leq 100$ )，求从最左边的点到最右边的点，再回到最左边的点的最短路程，中间不得经过重复的点且  $n$  个点都要经过。

## Soj 1345 能量项链

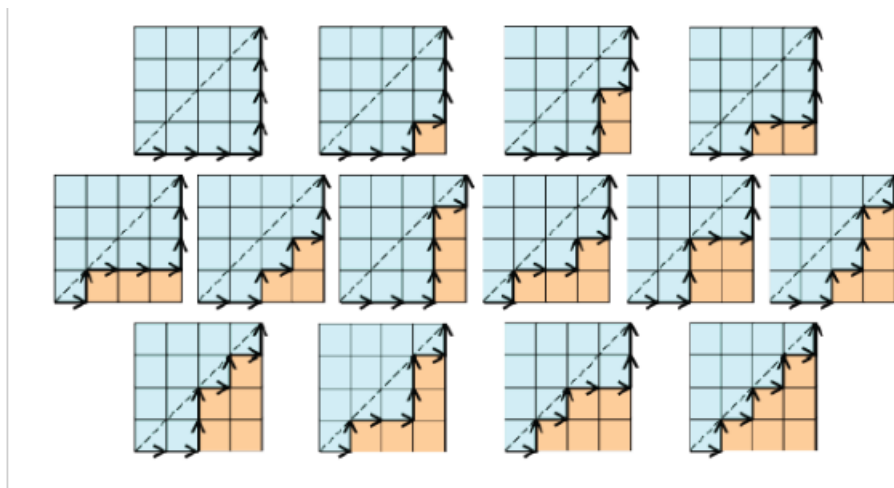
**题意：**一个首尾相连的项链有  $N$  颗能量珠 ( $4 \leq N \leq 100$ )。每颗能量珠有两个不超过 1000 的正整数作为头标记和尾标记，前一颗珠子的尾标记一定等于后一颗珠子的头标记，这样两颗珠子可以聚合在一起并释放能量。如果前一颗能量珠的头标记为  $m$ ，尾标记为  $r$ ，后一颗能量珠的头标记为  $r$ ，尾标记为  $n$ ，则聚合后释放的能量为  $m \times r \times n$ ，然后在原来位置得到一个头标记为  $m$ ，尾标记为  $n$  的新能量珠。现给出  $n$  个珠子的头标记 (尾标记就是后一个珠子的头标记) 现在请你设计一个聚合方案，使得最后剩下一颗能量珠，释放最多的能量，求这个最多能量是多少。

## Soj 1687 Permutation

**题意：** $n$  个数字有  $n!$  种排列，现根据排列后数字的大小关系插入〈和〉如 1 3 5 4 2 就变成  $1 < 3 < 5 > 4 > 2$ 。给定  $n$  和  $k$ ，( $0 < n \leq 100, 0 \leq k \leq 100$ ) 求  $n$  个数字的排列中，只插入  $k$  个〈的排列有多少个，答案 mod 2007

## Soj 13062 SubDiagonal Paths

**题意：**给一个  $n \times n$ ， $1 \leq n \leq 30$  的棋盘，求从左下角到右上角的所有路径数，只能向右走和向上走，且路径不能穿过对角线，如下图：



## 2.实验目的

1. 加深对动态规划的理解，熟悉各种动态规划的设计。
2. 提高运用课堂所学知识解决实际问题的能力。

### 3.程序设计

#### Soj 1176 Two Ends

**解题思路:**由于初始区间有偶数个数,两人轮流取数,则易知当区间长度为偶数时,第一个人取数,当区间长度为奇数时,第二个人取数。当第二个人取数时,只有一种选择,就是取两端较大的数(相等则取左边的数)。第一个人取数时有两种选择,显然,如果他能够知道取了某个数之后的最后结果,必然能做出最优抉择。设数组  $a$  存放原始的偶数个数,设  $f[i][j]$  表示在区间  $[i, j]$  两人取数和的最大值。由以上分析我们得出如下关系方程:

若  $(j-i+1)$  为奇数(即第二个人取数):

1.  $a[i] \geq a[j]$ :  
 $f[i][j] = f[i+1][j] - a[i];$
2.  $a[i] < a[j]$ :  
 $f[i][j] = f[i][j-1] - a[j];$

若  $(j-i+1)$  为偶数(即第一个人取数),则在两种取法中选最优:

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

最终答案就是  $f[1][n]$

时间复杂度  $O(n^2)$ , 我选择使用递归实现该 DP。

代码:

```
1 #include<iostream>
2 #include<cstdio>
3 using namespace std;
4 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
5 #define N 1010
6 #define INF 2000000000
7 int n;
8 int a[N],f[N][N];
9 void dfs(int l,int r){
10     int len=r-l+1;
11     if (f[l][r]!=-INF) return;
12     if (l==r){
13         f[l][r]=-a[l];
14         return;
15     }
16     if (len%2==1){ //第二个人取数,只取较大数
17         if (a[l]>=a[r]){
18             dfs(l+1,r);
19             f[l][r]=f[l+1][r]-a[l];
20         }
21         else {
22             dfs(l,r-1);
23             f[l][r]=f[l][r-1]-a[r];
24         }
25     }
26     else { //第一个人取数,两种选择
27         dfs(l+1,r);
28         dfs(l,r-1);
29         f[l][r]=max(f[l+1][r]+a[l],f[l][r-1]+a[r]);
30     }
31 }
32 int main(){
33     int T=0;
34     while(cin>>n,n>0){
35         rep(i,1,n) cin>>a[i];
36         rep(i,1,n)
37             rep(j,1,n)
38                 f[i][j]=-INF;
39         //f[i][j]为[i,j]的答案,先初始化为无穷小
40         dfs(1,n);
41         printf("In game %d, the greedy strategy might lose by as many as %d points.\n",++T,f[1][n]);
42     }
43     return 0;
44 }
```

## Soj 1011 Lenny's Lucky Lotto

**解题思路:** 设  $a[i]$  表示第  $i$  位数字是什么。此题入手处在于后一位数字至少是前一位数字的 2 倍。由于首位数字最小是 1, 也就是说, 每一位数字的取值范围是  $[2^{(i-1)}, a[i+1]/2]$ ; 设  $dp[i][j]$  表示第  $i$  位数字是  $j$  时的方案数。我们枚举  $i, j, k$ , 对于  $dp[i][j]$ , 其值应为  $\sum dp[i-1][k]$ , 其中  $1 \leq k \leq j/2$ 。最终答案就是  $\sum dp[N][i]$ , 其中  $1 \leq i \leq M$ 。时间复杂度为  $O(n*m^2)$

代码:

```
1 #include<iostream>
2 #include<cstdio>
3 #include<cstring>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 long long dp[15][2010];
7 int main(){
8     int T,n,m;
9     long long ans;
10    cin>>T;
11    rep(t,1,T){
12        cin>>n>>m;
13        int p=1;
14        memset(dp,0,sizeof(dp));
15        rep(i,1,m) dp[1][i]=1;
16        rep(i,2,n){
17            rep(j,p*2,m){
18                rep(k,p,j/2)
19                    dp[i][j]+=dp[i-1][k];
20            }
21            p*=2;
22        }
23        ans=0;
24        rep(i,p,m) ans+=dp[n][i];
25        printf("Case %d: n = %d, m = %d, # lists = %I64d\n",t,n,m,ans);
26    }
27    return 0;
28 }
```

## Soj 1121 Tri Tiling

**分析和解法:** 注意到被填的矩阵是  $3*n$  的, 固定了是 3 行, 这种特性启示我们应该可以用递推来解决。由于小矩形的面积是  $2*1$  的, 是偶数, 也就是说, 被填矩形的面积也要是偶数才能被填满。因此, 当  $n$  为奇数时, 方案数为 0。下面重点考虑  $n$  为偶数的情况。首先考虑  $n=2$  时, 共有下列 3 种情况:

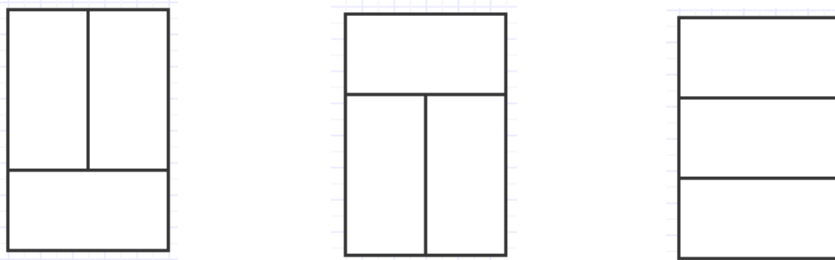


图 1

设  $dp[n]$  表示用  $2 \times 1$  的小矩形拼成  $3 \times n$  的大矩形的方案数。由以上分析得， $n$  为奇数时， $dp[n]=0$ ； $n=2$  时， $dp[n]=3$ ；

当  $n=4$  时，首先可以把  $3 \times 4$  的大矩形看成两个  $3 \times 2$  的小矩形，这样就有  $dp(2) \times dp(2) = 9$  种方案。除此之外，还有 2 种可以横跨两个  $3 \times 2$  的小矩形的边界的方案，如图 2，加起来一共 11 种方案。

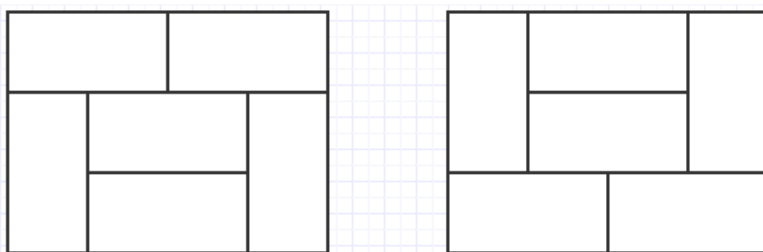
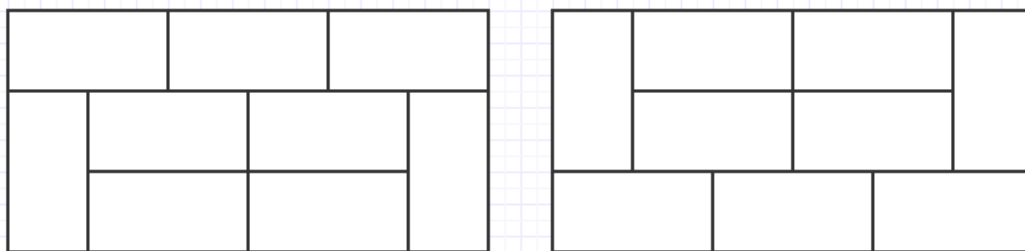


图 2

当  $n=6$  时，我们可以看成左边一个  $3 \times 4$  的矩形和右边一个  $3 \times 2$  的矩形组成。这样则有  $dp[4] \times dp[2] = 33$  种方案。然后我们可以看成左边一个  $3 \times 2$  的矩形和右边一个  $3 \times 4$  的矩形，注意此时不是新增 33 种方案，因为这样摆其实是和前面有重复的，它只比第一种摆法多了图二的两种方案，因此新增了  $f[2] \times 2 = 6$  种方案。最后，将整个大矩形看成一个整体，还有图 3 的两种方案，一共有  $33 + 6 + 2 = 41$  种。



依次类推，当求  $n$  时，可以依次把矩形划分为(1)  $3 \times (n-2)$  和  $3 \times 2$  的矩形，(2)  $3 \times (n-4)$  和  $3 \times 4$  的矩形，(3)  $3 \times (n-6)$  和  $3 \times 6$  的矩形.....和一个大矩形

然后对每种情况求解相对前面几种划分没有出现的新增的方案数，即(1)  $dp[n-2] \times dp[2]$ , (2)  $dp[n-4] \times 2$  (因为这种分法相对(1)的分法右边的矩形只是新增了两种，而左边的矩形可以  $dp[n-4]$  种，根据乘法原理，可知新增了  $dp[n-4] \times 2$  种方案)，(3)  $dp[n-6] \times 2$ , .....，2。然后依次把这些数相加即可得到总的方案数。

由上可得状态转移方程：

$$dp[n] = dp[n-2] \times dp[2] + dp[n-4] \times 2 + dp[n-6] \times 2 + \dots + 2$$

等号右边的式子可使用前缀和优化。可设  $sum[n]=dp[n]*2+dp[n-2]*2+...+2$ 。则动态规划方程可化为：

$$dp[n]=dp[n-2]*dp[2]+sum[n-4].$$

代码：

```
1 #include<iostream>
2 #include<cstdio>
3 #include<cstring>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 int dp[31],sum[31];
7 int main(){
8     int n;
9     dp[0]=1;
10    sum[0]=dp[0]*2;
11    dp[2]=3;
12    sum[2]=dp[2]*2+sum[0];
13    rep(i,3,30){
14        if (i%2==1){//n为奇数无方案
15            dp[i]=0;
16            sum[i]=sum[i-1];
17        }
18        else //n为偶数
19            dp[i]=dp[i-2]*dp[2]+sum[i-4];
20            sum[i]=sum[i-1]+dp[i]*2;
21        }
22    }
23    while (cin>>n){
24        if (n==-1)break;
25        cout<<dp[n]<<endl;
26    }
27    return 0;
28 }
29
```

### Soj 1264 Atomic Car Race

**解题思路：**这题较简单，数据规模较小，可以暴力 dp。设  $dp[n]$  为从起点到  $n$  的最快时间。要求  $dp[n]$ ，显然需要知道在  $1-n$  中哪个地点换胎，是最优的，当然也可能不换。所以我们只要枚举换胎地点，算出时间，再从中取最小值即可。设  $t[i]$ ，为从起始点开始，不换胎行驶  $i$  距离所需时间。那么由以上分析可得状态转移方程：

$$dp[n]=\min(dp[i]+t[a[n]-a[i]]), \text{ 其中 } 0 \leq i < n;$$

当然，对  $n$  之前每个点都是如此处理即可。



时间复杂度  $O(n^2)$

代码:

```
1 #include<iostream>
2 using namespace std;
3 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
4 #define N 102
5 double dp[N],t[10010];
6 int a[N];
7 double b,e,v,f;
8 int r;
9 double calc(int x){
10     if (x>=r)
11         return (double)1.0/(v - e * (x - r));
12     else
13         return (double)1.0/(v - f * (r - x));
14 }
15 int main(){
16     int n;
17     while (cin>>n&&n){
18         rep(i,1,n)cin>>a[i];
19         cin>>b>>r>>v>>e>>f;
20         dp[0]=0; //起点到自己时间为0
21         rep(i,1,a[n])t[i]=t[i-1]+calc(i-1);
22         rep(i,1,n){
23             dp[i]=t[a[i]]; //将用时初始化为全程不换胎的时间
24             rep(j,1,i-1) //枚举换胎地点, 求时间开销最小值。
25                 dp[i]=min(dp[i],dp[j]+t[a[i]-a[j]]+b);
26         }
27         printf("%.4lf\n",dp[n]);
28     }
29     return 0;
30 }
```

## Soj 1828 Minimal

**分析和解法:**这题要求点对差的和最小。很明显,与点  $x$  的差最小的点,必然是  $x$  附近的点,所以一个直觉的想法就是将  $S1$  和  $S2$  的点先分别按照从小到大的顺序排序。排号序后,我们设  $dp[i][j]$  表示  $S1$  中前  $i$  个点与  $S1$  中前  $j$  个点配对的最优结果。接下来我们思考如何得出状态转移方程。由于题目要求我们配对  $N$  对点对,也就是说  $S1$  中的每个点都要配对。所以我们只要考虑  $S1$  的点的配对情况。现在我们假设前  $i-1$  个点都已经配对成功,这意味着  $S2$  中至少前  $i-1$  个点已经不能做配对了。考虑  $S1$  中的第  $i$  个点,  $S2$  中与它做配对的点只能是  $i$  到  $m$  的某个点,我们设为  $j$ 。那么  $i$  有 3 种选择:

1.  $j=i$ ,  $S1$  中的第  $i$  个点与  $S2$  的第  $i$  个点配对。此时有:

$$dp[i][i]=dp[i-1][i-1]+abs(s1[i]-s2[i]);$$

2.  $j>i$ ,  $i$  与  $j$  配对。则有:

$$dp[i][i]=dp[i-1][i-1]+abs(s1[i]-s2[j]);$$

3.  $j>i$ ,  $i$  不与  $j$  配对,相当于  $S1$  的前  $i$  个点与  $S2$  的前  $j-1$  个点配对,则有:

$$dp[i][j]=dp[i][j-1];$$

显然  $dp[i][j]$  的答案就是从上面三者中取最小值。

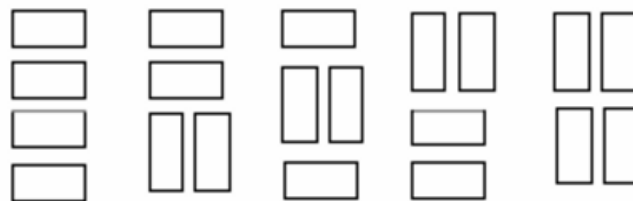
所以我们只要枚举  $i$  和  $j$ ，按上述方法求得最终的答案  $dp[N][M]$  即可。  
时间复杂度  $O(NM)$

代码：

```
1 #include<iostream>
2 #include<algorithm>
3 #include<cmath>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 int s1[502],s2[502],dp[502][502];
7 int main(){
8     int T,n,m;
9     cin>>T;
10    while (T--){
11        cin>>n>>m;
12        rep(i,1,n)cin>>s1[i];
13        rep(i,1,m)cin>>s2[i];
14        sort(s1+1,s1+1+n);
15        sort(s2+1,s2+1+m);
16        rep(i,1,n)
17            rep(j,i,m){
18                if (i==j)dp[i][i]=dp[i-1][i-1]+abs(s1[i]-s2[i]);
19                else dp[i][j]=min(dp[i-1][j-1]+abs(s1[i]-s2[j]),dp[i][j-1]);
20            }
21        cout<<dp[n][m]<<endl;
22    }
23    return 0;
24 }
```

### Soj 1527 Tiling a Grid With Dominoes

**分析和解法：**题目说了结果不会超过 32-bit integer，那么  $n$  最多是多少才不会使结果超过 32-bit integer 呢？题中给出了  $4 \times 2$  的棋盘的方案，如下图，共 5 种：



那我们可以做如下简单估算，将  $4 \times n$  的棋盘拆成  $k$  个  $4 \times 2$  的棋盘，使  $5^k \leq 2^{31}-1$ ，得  $13 < k < 14$ ，暂且令  $k_{\max}=14$ ，则， $n \leq 28$ 。

此题与 1121 类型，也可以使用递推，不过考虑递推情况太麻烦，这里我使用状态压缩 dp 完成。此题实际上是经典状态压缩 dp 求用  $1 \times 2$  的方块覆盖  $n \times m$  的棋盘的方案数的简化版，下面我们基于  $n \times m$  的棋盘来设计算法。

我们设横着铺砖，则被铺的两格值为 1；若竖着铺，则上面一格的值为 0，下面一格的值为 1。设有状态  $f[i][k]$ ， $i$  表示格子的编号（从上到下，同行内从左到右）， $k$  化为二进制后表示编号为  $i-m+1$  到  $i$  的  $m$  个格子的状态是否被覆盖，而且此时前  $i-m$  格都已被覆盖。比如  $n=m=4$  时状态

$f[5][5]=f[5][0101(2)]$ 对应下图的情形。

0	1	2	3
		0	1
4	5	6	7
0	1		
8	9	10	11
12	13	14	15

然后就可以分情况讨论来转移了：

1. 若  $k \& 1 = 0$ , 即当前格为 0, 说明它要打竖铺, 且自己作为竖着这块砖的上面那格, 这时必有第  $i-m$  格已经铺了, 即为 1, 所以  $f[i][k] = f[i-m][k \gg 1]$ ;
2. 若  $k \& 1 = 1$ , 即当前格为 1, 则有两种情况, 说明第  $i$  格和第  $i-1$  格或第  $i-m$  格共用了一个  $1 \times 2$  的方块, 把这两种情况的方案数加起来就是这个状态的方案数了, 于是有  $f[i][k] \leftarrow f[i-1][k \gg 1]$  和  $f[i][k] \leftarrow f[i][k] + f[i-1][((k \gg 1) \wedge 1) | (1 \ll (m-1))]$ , 当然还要判断一下这两种情况是否可行。

在此题中,  $m=4$ , 时间复杂度为  $O(4n \times 2^4)$

代码:

```

1 #include<iostream>
2 #include<cstring>
3 #include<cstdio>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 int f[4*30][16];
7 void solve(int T,int n,int m) {
8     memset(f,0,sizeof(f));
9     f[0][(1<<m)-2]=1; //假设网格的上面还有一行且已经填满
10    for (int i=1;i<=n*m;i++) //枚举格子
11        for (int k=0;k<(1<<m);k++) //枚举二进制状态
12            if (k&1) { //如果第i格被覆盖
13                if (i/m>0) f[i][k]=f[i-1][k>>1]; //如果有上一行, 就可以竖着放一个
14                if (i%m>0&&(k&2)) f[i][k]+=f[i-1][((k>>1)^1)|(1<<(m-1))]; //如果左边一格空着, 就可以横着放一个
15            }
16            else f[i][k]=f[i-1][((k>>1)|(1<<(m-1)))];
17            //如果第i格没有被覆盖, 则第i-m个格子一定是1, 其他格子状态相同
18    cout<<T<<" "<<f[n*m-1][(1<<m)-1]<<endl;
19 }
20 }
21 int main() {
22     int T,n,m;
23     cin>>T;
24     rep(t,1,T){
25         cin>>n;
26         solve(t,n,4);
27     }
28     return 0;
29 }
30

```

## Soj 1148 过河

**分析和解法:**这题题意还是很容易懂，设  $dp[i]$  表示调到第  $i$  个点最少需要踩到的石子数，明显  $dp[i] = \min(dp[j] + i \text{ 位置是石子?} : 1:0)$ ，其中  $i-T \leq j \leq i-S$ 。然而数据范围  $L \leq 10^9$ ，显然不可能直接这样  $dp$ 。在这个数据范围的动态规划只有 3 种选择，第一种是可以推导出直接求的公式，第二种是可以使用矩阵乘法加速，第三种就是状态压缩。明显前两种都不适用，那么我们来考虑状态压缩是否可行。

仔细观察数据，我们发现虽然桥很长，但是石头最多只有 100 个，很稀疏，那么是不是可以把这些石头之间的距离压短呢？设石头位置为  $x$ ，与这个石头可能有关的位置就是  $[x-T, x-S]$  和  $[x+S, x+T]$  而已。假设  $S=2, T=3$ ，那么从 0 开始能到达的位置就是  $\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 \dots\}$

假设  $S=3, T=4$ ，那么从 0 开始能到达的位置就是  $\{3, 4, 6, 7, 8, 9, 10, 11 \dots\}$

假设  $S=4, T=5$ ，那么从 0 开始能到达的位置就是  $\{4, 5, 8, 9, 10, 12, 13, 14 \dots\}$

假设  $S=4, T=9$ ，那么从 0 开始能到达的位置就是  $\{4, 8, 9, 12, 13, 16, 17, 18, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, \dots\}$

可以发现，序列总是从某一个数开始变得连续，也就是说从这个数开始后面的状态其实都是这样。而通过观察可以发现，这个数总是比  $S$  和  $T$  的最小公倍数要小。那我们就将这个数取  $S$  和  $T$  的积  $k$ 。将终点和起点也看成石头，然后两个石头间的距离，如果大于  $k$  就设成  $k$ ，因为  $k$  之后的状态和  $k$  是一样的。然后我们再使用一开始的  $dp$  式即可。由于可以跳过终点，最后应该在  $dp[L] \sim dp[L+T]$  之间取最小值作为答案。

时间复杂度为  $O(M^2)$

代码:

```
1 #include<iostream>
2 #include<algorithm>
3 #include<cstring>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 int w[110];
7 int dp[11000],bo[11000];
8 int main(){
9     int L,S,T,M;
10    cin>>L>>S>>T>>M;
11    rep(i,1,M)cin>>w[i];
12    sort(w+1,w+1+M);
13    if (S==T){//单独处理S==T的情况
14        int ans=0;
15        rep(i,1,M)
16            if (w[i]%S==0)ans++;
17        cout<<ans<<endl;
18    }
```

```

19     else {
20         int k=S*T;
21         w[0]=0;w[M+1]=L;
22         rep(i,1,M+1){
23             if (w[i]-w[i-1]>k){
24                 int delta=w[i]-w[i-1]-k;
25                 rep(j,i,M+1)
26                     w[j]-=delta;
27             }
28             bo[w[i]]=1;
29         }
30         bo[w[M+1]]=0;
31         memset(dp,-1,sizeof(dp));
32         dp[0]=0;
33         rep(i,0,w[M+1]){
34             if (dp[i]==-1)continue;
35             rep(j,i+S,i+T){
36                 if (dp[j]==-1||dp[i]+bo[j]<dp[j])
37                     dp[j]=dp[i]+bo[j];
38             }
39         }
40         int ans=110;
41         rep(i,w[M+1],w[M+1]+T)ans=min(ans,dp[i]);
42         cout<<ans<<endl;
43     }
44     return 0;
45 }

```

## Soj 1163 Tour

**分析和解法:**既然题目有严格的方向规定，我们不妨先将所有点按横坐标排序。然后题目要求从左到右再回来，这等价于两个人从起点出发经过不同的点到达终点。这样看的话状态就很容易定了。直接设  $dp[i][j]$  表示第一个人走到点  $i$ ，第二个人走到点  $j$  且  $i$  和  $j$  前面的位置都不被重复走过的最短距离和，由于这两个人具有对称性，即 A 走到  $x_1$  再到  $x_2$ ，B 走到  $x_3$  再到  $x_4$  与 B 走到  $x_1$  再到  $x_2$ ，A 走到  $x_3$  再到  $x_4$  是一样的，所以我们不妨假设第一个人总是走在前面，即  $i \geq j$ 。状态转移有下面 3 种情况：

1.  $i=j$ ，两个人处于同一个点，那么

$$dp[i][j]=\min(dp[i-1][k]+\text{dist}(i,i-1)+\text{dist}(k,j));$$

2.  $i=j+1$ ，即 A 在 B 前一个点，那么  $dp[i][j]=\min(dp[k][j]+\text{dist}(i,k))$  ( $1 \leq k \leq j$ )

3.  $i > j+1$ ，A 在 B 前至少两个点，那么  $dp[i][j]=dp[i-1][j]+\text{dist}(i,i-1)$

在这里显然不用考虑 A 和 B 在中间走到同一个点，因为肯定不是最优。

最终答案就是  $dp[n][n]$ 。

时间复杂度  $O(n^3)$

代码：



```

1 #include<iostream>
2 #include<cstring>
3 #include<cstdio>
4 #include<cmath>
5 #include<algorithm>
6 using namespace std;
7 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
8 #define INF 100000000.0
9 struct node{
10     double x,y;
11 }a[110];
12 int n;
13 double dp[110][110];
14 bool cmp(const node &n1,const node &n2){
15     return n1.x<n2.x;
16 }
17 double dist(node &n1,node &n2){
18     double x=n1.x-n2.x;
19     double y=n1.y-n2.y;
20     return sqrt(x*x+y*y);
21 }
22 int main(){
23     while(cin>>n){
24         rep(i,1,n)
25             cin>>a[i].x>>a[i].y;
26         sort(a+1,a+1+n,cmp);
27         rep(i,0,n)
28             rep(j,0,n)
29                 if (!i || !j) dp[i][j]=0;
30                 else dp[i][j]=INF;
31         dp[1][1]=0; //起点到自己是0
32         rep(i,2,n) dp[1][i]=dp[i][1]=dp[i-1][1]+dist(a[i],a[i-1]);
33         //上面这步是设置边界,即一个人走完全程
34
35         rep(i,1,n) //枚举第一个人走的点
36             rep(j,1,i) { //枚举第二个人走的点
37                 if (i>j+1) dp[i][j]=dp[j][i]=dp[i-1][j]+dist(a[i],a[i-1]);
38                 else if (i==j+1){
39                     rep(k,1,j-1) //枚举第一个人是从k点走到i点的
40                         dp[i][j]=dp[j][i]=min(dp[i][j],dp[k][j]+dist(a[i],a[k]));
41                 }
42                 else{//i==j
43                     rep(k,1,j-1)
44                         dp[i][j]=dp[j][i]=min(dp[i][j],
45                             dp[i-1][k]+dist(a[i],a[i-1])+dist(a[k],a[j]));
46                 }
47             }
48         printf("%.21f\n",dp[n][n]);
49     }
50     return 0;
51 }

```

## Soj 1345 能量项链

**分析和解法:**首先我们不考虑环,我们考虑一条链的合并情况。设  $f[i][j]$  表示将区间  $[i, j]$  这段珠子合并所释放的最大能量。首先如果只有两个珠子,那么合并情况是唯一的,如果有三个珠子 A、B、C,那么有两种情况,先合并 AB,再合 C 或者先合并 BC 再合 A。所以我们只需要枚举区间的断点,

断点之前的珠子先合并，断点之后的珠子也先合并，然后这两部分再合并，取所有情况的最小值即可。设  $a[i]$  表示第  $i$  个珠子的头标记，那么转移方程就是  $f[i][j]=f[i][k]+f[k+1][j]+a[i]*a[k+1]*a[j+1]$ 。现在来看一个环的情况，其实，我们只要枚举这个环从哪里断开，就可以变成一条链了。为什么呢，因为最后肯定会变成两个珠子合并，而这两个珠子就相当于把记录第一珠子的头标记的那个原始珠子看成链的头，把记录第二个珠子尾标记得那个原始珠子看成链的尾。在所有断开情况中取最小值就是我们要的答案。为了方便，我们令  $a[i+n]=a[i]$ ，即将初始那个链复制一边接在后面。

时间复杂度  $O(n^3)$

代码：

```

1 #include<iostream>
2 #include<cstring>
3 using namespace std;
4 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
5 int a[210];
6 int f[210][210];
7 int dp(int l,int r){
8     if (f[l][r])return f[l][r];
9     if (l+1==r)return f[l][r]=a[l]*a[l+1]*a[r+1];
10    int maxx=0;
11    rep(i,l,r-1){
12        int tmp=dp(l,i)+dp(i+1,r)+a[l]*a[i+1]*a[r+1];
13        maxx=max(maxx,tmp);
14    }
15    return f[l][r]=maxx;
16 }
17 int main(){
18     int n;
19     while (cin>>n){
20         rep(i,1,n){
21             cin>>a[i];
22             a[i+n]=a[i];
23         }
24         int ans=0;
25         memset(f,0,sizeof(f));
26         rep(i,1,n)
27             ans=max(ans,dp(i,i+n-1));
28         cout<<ans<<endl;
29     }
30     return 0;
31 }

```

## Soj 1687 permutation

**分析和解法：**我们设  $f[n][k]$  就是  $n$  个数插  $k$  个小于号的答案。假设我们已经知道了  $n-1$  个数的排列  $a[1], a[2], a[3] \cdots a[n-1]$ ，且已经有了  $k$  个 ' $<$ ' 现在插入第  $n$  个数，有  $n$  个位置可以插：

1. 插在  $a[1]$  前面，则 ' $<$ ' 没有增加，这种插法有 1 个位置可以插
2. 若  $a[i] < a[i+1]$ ，把  $n$  插入到  $a[i+1]$  前面，' $<$ ' 没有增加，这种插

法有  $k$  个位置可以插

3. 插入到其他位置 ‘<’ 都增加

由以上分析可得状态转移方程：

$$f[n][k] = f[n-1][k] * (k+1) + f[n-1][k-1] * (n - (k-1+1))$$

时间复杂度  $O(n^2)$

代码：

```
1 #include<iostream>
2 using namespace std;
3 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
4 int f[101][101];
5 int main() {
6     int n,k;
7     rep(i,0,100) {
8         f[i][0]=1;
9         f[0][i]=0;
10    }
11    rep(i,1,100)
12        rep(j,1,100)
13            f[i][j]=(f[i-1][j]*(j+1)+f[i-1][j-1]*(i-j))%2007;
14    while (cin>>n>>k)
15        cout<<f[n][k]<<endl;
16    return 0;
17 }
18
```

### Soj 13062 SubDiagonal Paths

**分析和解法：**这题其实以前做过的一道趣味数学题，从左下角到右上角有多少条路。设  $f[i][j]$  表示从  $(1, 1)$  (这里我把左下角看成  $(1, 1)$ , 把右上角看成  $(n+1, n+1)$ , 点比格子多 1) 到  $(n+1, n+1)$  有多少条路径。由于只能向右和向上走，显然转移方程如下：

$$f[i][j] = f[i-1][j] + f[i][j-1];$$

然后题目限制路径不能超过对角线，那么这里要保证  $j \geq i$ 。

时间复杂度  $O(n^2)$


代码：

```
1 #include<iostream>
2 using namespace std;
3 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
4 #define dto(i,u,v) for (int i=(u);i>=(v);i--)
5 long long f[32][32];
6 int main() {
7     f[1][1]=1;
8     rep(i,1,31)
9         rep(j,i,31) {
10             if (i-1>0) f[i][j]+=f[i-1][j];
11             if (j-1>=i) f[i][j]+=f[i][j-1];
12         }
13     int n;
14     while (cin>>n&&~n)
15         cout<<f[n+1][n+1]<<endl;
16     return 0;
17 }
```



## 4.程序运行与测试

11 道题均通过成功运行并通过 sicily 的测试。

5062394		smie15352408	1527	C++	Accepted	0.01sec	316 KB	989 Bytes
5062387		smie15352408	13062	C++	Accepted	0sec	316 KB	402 Bytes
5062371		smie15352408	1687	C++	Accepted	0.11sec	348 KB	358 Bytes
5062318		smie15352408	1345	C++	Accepted	0.01sec	480 KB	659 Bytes
5062258		smie15352408	1163	C++	Accepted	0sec	404 KB	1491 Bytes
5062112		smie15352408	1148	C++	Accepted	0sec	396 KB	1058 Bytes
5061282		smie15352408	1828	C++	Accepted	0sec	1296 KB	604 Bytes
5061275		smie15352408	1828	C++	Accepted	0sec	1296 KB	583 Bytes
5061205		smie15352408	1264	C++	Accepted	0sec	388 KB	777 Bytes
5061138		smie15352408	1121	C++	Accepted	0sec	308 KB	577 Bytes
5060699		smie15352408	1011	C++	Accepted	0.16sec	564 KB	635 Bytes
5060628		smie15352408	1176	C++	Accepted	0.02sec	4296 KB	910 Bytes

## 5.实验总结与心得

通过这次动态规划的训练，我加深了对动态规划的理解。动态规划的中心思想是由局部最优解递推产生全局最优解。设计动态规划一般来说分为以下几步：1.建立数学模型来描述问题。

2.把求解的问题分成若干子问题。

3.对最小的子问题求解，然后逐步递推求出更大的子问题的解，最后得到全局最优解。

要注意，要应用动态规划一定要分析解决问题的过程是否满足无后效性，若有后效性则设计的动态规划是错误的。

## 附录、提交文件清单

实验报告.pdf

1176.cpp

1011.cpp

1121.cpp

1264.cpp

1828.cpp

1527.cpp

1148.cpp

1163. cpp  
1345. cpp  
1687. cpp  
13062. cpp