

中山大学数据科学与计算机学院本科生实验报告

课程名称：算法设计与分析 任课教师：张子臻

年级	2017级	专业（方向）	软件工程
学号	17343100	姓名	仕润昊
电话	13280152626	Email	1056627011@qq.com
开始日期	2019/3/21	完成日期	2019/3/23

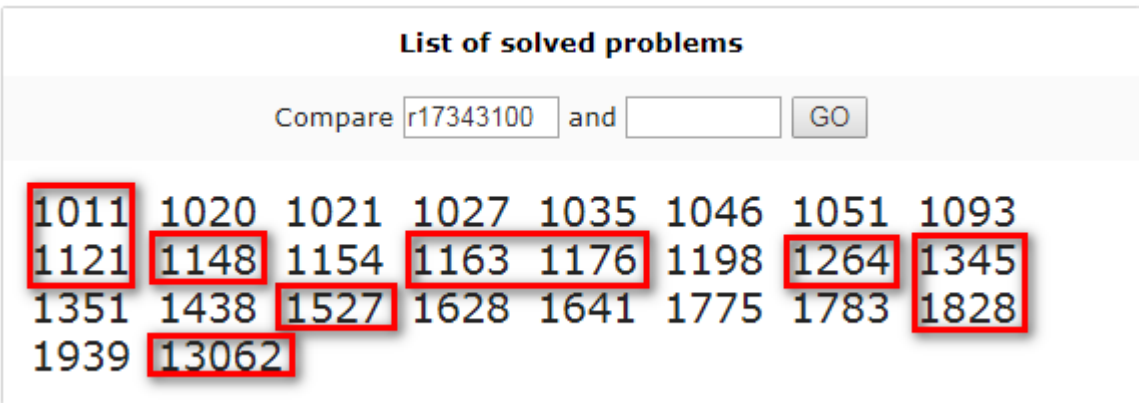
1.实验题目

完成情况——十道题

本次实验共完成十道题目

soj.acmm.club
1176 1011 1121 1264 1828
1527 1148 1163 1345 13062

Sicily截图如下



具体题目

1176 Two Ends

题意：给一个长度为n的数列 a_i ，两个人在上面做交替取数，每个人的每一轮从这个数列的两端中取出一个数（不能不操作）。先手可以自由选择策略，后手选择贪心策略。贪心策略是指，如果两端数大小不同，选择大的那个；如果相同选择左边那个。问最后先手能赢后手多少分。

约束：1 ≤ n ≤ 1000 且 n 为偶数，sum a_i ≤ 1,000,000

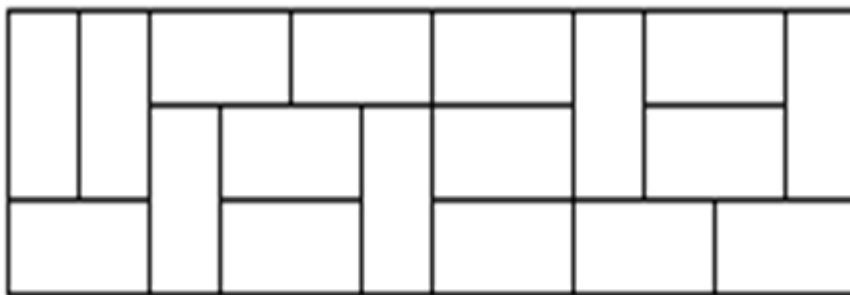
1011 Lenny's Lucky Lotto

题意：给出N，M，问有多少个长度为N的整数序列，满足所有数都在[1,M]内，并且每一个数至少是前一个数的两倍

约束：1 ≤ N ≤ 10，1 ≤ M ≤ 2000

1121 Tri Tiling

题意：用1*2的长方形铺满3*n的长方形，有多少种方法



约束：1 ≤ N ≤ 30

1264 Atomic Car Race[Special judge]

题意：在一次赛车比赛中，共有n个检查点，在每个检查点可以选择是否更换轮胎，更换轮胎需要花费b单位时间。在一次更换轮胎之后，赛车的速度先增加（轮胎变热），后减少（轮胎损坏），每单位公里行驶需要的时间可以表达为（x为离最近更换轮胎距离，从x公里跑到x+1公里）：

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

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

现在问从起点到终点的最少时间

约束：

$$0 < n \leq 100, 0 \leq r \leq a_n - 1, 0 < a_1 < a_2 < \dots < a_n \leq 10000, \\ v - e * (a_n - 1r) \geq 0.01, v - f * r \geq 0.01$$

1828 Minimal

题意：给出两个集合S1和S2，在S2中选出一些不重复的数与S1的每个数匹配，使得匹配的数的差的绝对值之和尽量小

约束：集合中数的个数不超过500

1527 Tiling a Grid With Dominoes

题意：题目类似前面的1121，现在用骨牌填充 $4*n$ 的矩形，问有多少方案

约束：最终答案在32位整数范围内

1148 过河

题意：桥的起点为0，终点为L，其中地有M个石子。青蛙每次跳的范围为[S,T]，问要跳过桥最小踩到石子次数

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

1163 Tour

题意：旅行商问题的变种。一个想走一个环路，经过N个地点，并且先从左往右走，再从右往左走，求最短路程

约束：N题目没给出，但是可以认为 $1 \leq N \leq 50$ ，坐标的x值均不相同

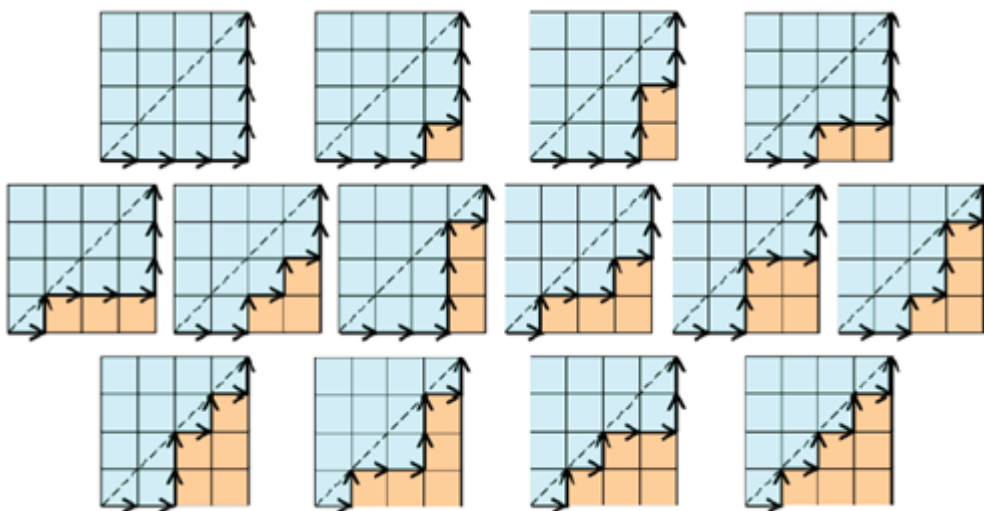
1345 能量项链

题意：给出一串项链，每次可以选相邻两个珠子进行聚合，释放出一定的能量，并产生一个新珠子，项链是头尾相接的，求释放的能量的总和的最大值。

限制：项链长度不超过100

13062 SubDiagonal Paths

题意：给出n，在 $n*n$ 的格子中，一开始你位于(0,0)，然后你每次可以从(x,y)走到(x+1,y)，或者(x,y+1)，全程需满足 $x \geq y$ ，问这样走到(n,n)处有几种走法。



2.实验目的

本次实验是要掌握基本的动态规划算法，学会分解熟练应用动态规划算法。虽然抽象后进行求解的思路并不复杂，但具体的形式千差万别，找出问题的子结构以及通过子结构重新构造最优解的过程很难统一。

熟练掌握 Top-down 方法与Bottom-up方法，掌握分析问题结构，确定子问题，写出状态转移方程最后解决问题的方法。

3.程序设计

1176 Two Ends

我们可以先来考虑赢的那一方能拿到的最大分数。后手是采用贪心策略，序列的最优值可以由子序列的最优值得到。假设 $F[a,b]$ 是区间 $[a,b]$ 能拿到的最大分数，那么你可能拿走最左边 $card[a]$ 或者最右边的一张卡 $card[b]$ ，那么

$$F[a,b] = \max\{F[a,b-1] + card[b], F[a+1,b] + card[a]\}$$

如果拿走 $card[a]$

$$\begin{aligned} F[a+1,b] &= F[a+2,b] \quad \text{if } card[a+1] \geq card[b] \\ F[a+1,b] &= F[a+1,b-1] \quad \text{if } card[a+1] < card[b] \end{aligned}$$

如果拿走 $card[b]$

$$\begin{aligned} F[a,b-1] &= F[a+1,b-1] \quad \text{if } card[a] \geq card[b-1] \\ F[a,b-1] &= F[a,b-2] \quad \text{if } card[a] < card[b-1] \end{aligned}$$

程序如下

```
int cal_a_b(int a, int b) {
    int leftsum;
    int rightsum;
    if (a > b) return 0;
    if (cal[a][b] != -1) {
        return cal[a][b];
    }
    if (arr[a+1] >= arr[b]) {
        leftsum = cal_a_b(a+2, b) + arr[a];
    }
    else {
        leftsum = cal_a_b(a+1, b-1) + arr[a];
    }
    if (arr[a] >= arr[b-1]) {
        rightsum = cal_a_b(a+1, b-1) + arr[b];
    }
    else {
        rightsum = cal_a_b(a, b-2) + arr[b];
    }
    cal[a][b] = max(leftsum, rightsum);
    return cal[a][b];
}
```

1011 Lenny's Lucky Lotto

我们用 $F[i,j]$ 表示从 j 个数中，按规则选 i 个数的方法数。

那么从 j 个数中选 i 个数可以分解成两种状况：选数字 j 或者不选数字 j 。

选择数字 j 的话，那么根据我们的规则，第 $i-1$ 个数一定要小于 $\text{floor}(j/2)$ ；

如果不选数字 j 的话，那么就可以用 $F[i, j-1]$ 来表示，所以状态转移方程如下：

$$F[i, j] = F[i-1][j/2] + F[i, j-1]$$

根据上面的状态转移方程，我们就可以很轻松的写出程序代码来了。

```
long long cal_n_m(int n, int m) {
    for (int i = 1; i <= m; i++) {
        list[1][i] = i;
    }

    for (int i = 2; i <= n; i++) {
        for (int j = i; j <= m; j++) {
            // 选择j或者不选j的和
            list[i][j] = list[i-1][j/2] + list[i][j-1];
        }
    }
    return list[n][m];
}
```

1121 Tri Tiling

首先我们知道 $1*2$ 的多米诺骨牌的面积为2，而 $3*n$ 的矩形要想完整容纳多米诺骨牌的话， n 必须要是偶数，因为只有这样，才能使 $3*n/2$ 为整数。

经过画图我们发现， $n=2$ 时不可分割的拼接情况有3中， $n=2*k$ ($k>2$) 时不可分割的拼接情况有两种，那么我们的状态转移方程就可以写为

$$F[k] = F[k-2] * 3 + F[k-4] * 2 + \dots + F[2] * 2 \quad (1)$$

所以

$$F[k-2] = F[k-4] * 2 + F[k-6] * 2 + \dots + F[2] * 2 \quad (2)$$

(1)-(2)得

$$F[k] = F[k-2] * 4 - F[k-4]$$

那么我们的状态方程便可以简化为上式的形式，根据状态转移方程，我们便可以很轻松的写出相应的程序，如下。

```
cal[0] = 1;
cal[2] = 3;
for (int i = 4; i <= 31; i += 2) {
    cal[i] = cal[i-2] * 4 - cal[i-4];
}
```

1264 Atomic Car Race[Special judge]

我们用 $T[k]$ 来表示汽车行驶到第 k 个服务站后耗费的最短时间，那么最短时间可以认为是，从到达第 i 个服务站耗费的最短时间，并且在第 i 个服务站更换轮胎，再加上从第 i 个服务站到第 k 个服务站耗费的时间之和的最小值。状态转移方程如下。

$$T[k] = \min\{T[i] + \text{tyre_cost} + \text{cost_time}(i, k)\} \\ i \in (0, k - 1)$$

这道题只是题目复杂，状态方程很普通，另外还需要注意的是求解时间 cost_time 时需要储存到数组中，不然会超时。

如果不换轮胎耗费的时间我们用数组 G 存起来

```
for (int i = 0; i <= a[n]; i++) {
    double temp = (i >= r) ? 1 / (v - e * (i - r)) : 1 / (v - f * (r - i));
    G[i + 1] = G[i] + temp;
}
```

1828 Minimal

我们定义 $P[i,j]$ 为第一个数组前 i 个数和第二个数组前 j 个数的最小差绝对值之和。那么这个可以分解为第二个数组选第 j 或者不选第 j 个数。如果不选第 j 个数，即为 $P[i,j-1]$ ，如果选第 j 个数即为 $P[i-1,j-1] + |\text{arr2}[j] - \text{arr1}[i]|$ 。所以状态转移 方程为：

$$P[i, j] = \min\{P[i, j - 1], P[i - 1, j - 1] + |\text{arr2}[j] - \text{arr1}[i]|\}$$

根据状态转移方程，我们便可以确定相应的程序代码 了。

```
for (int i = 2; i <= s1; i++) {
    for (int j = i; j <= s2; j++) {
        p[i][j] = min(p[i][j - 1], p[i - 1][j - 1] + abs(s_arr1[i - 1] - s_arr2[j - 1]));
    }
}
```

1527 Tiling a Grid With Dominoes

本题目有比较复杂的状态转移，具体转移如下：

```
m[i][0] = m[i - 1][15];
m[i][3] = m[i - 1][15] + m[i - 1][12];
m[i][6] = m[i - 1][15] + m[i - 1][9];
m[i][9] = m[i - 1][6];
m[i][12] = m[i - 1][15] + m[i - 1][3];
m[i][15] = m[i - 1][15] + m[i - 1][12] + m[i - 1][6] + m[i - 1][3] + m[i - 1][0];
```

$m[i,j]$ 表示前 $i-1$ 列都已经被铺满且第 i 列的状态为 j ， j 可以看做二进制序列：0000,0011,0110,1001,1100,1111。1表示被覆盖，0表示没有被覆盖。

根据状态转移可以直接得到结果。

1148 过河

本题的状态转移也较为简单，把青蛙到达距离*k*处所踩到的最少石子记为*F[i]*，那么状态转移方程为

$$F[i] = \min\{F[i - k] + \text{rock}[i]\}$$
$$k \in [\text{min_step}, \text{max_step}]$$

转化为代码即为

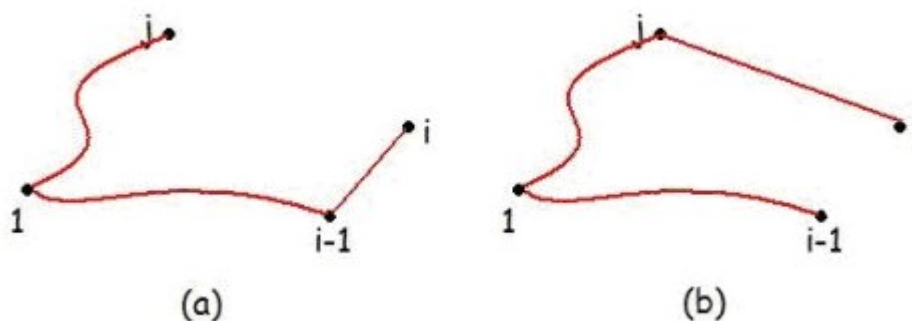
```
for (int i = max_step+1; i <= L; i++) {
    for (int k = min_step; k <= max_step && i-k >= 1; k++) {
        F[i] = min(F[i], F[i - k]);
    }
    if (rocks[i]) {
        F[i]++;
    }
}
```

但是这个题的数据量却有 10^9 ，所以我们要进行数据压缩，我们可以减少格子数，也就是只两个相邻石子之间只保留最多100个格子，因为进行了数据压缩，所以我们要进行特判min_step = max_step的情况。

```
// 特殊情况判断
if (min_step == max_step) {
    int ans = 0;
    for (int i = 1; i <= n; i++)
        if (stone_pos[i] % min_step == 0)
            ans++;
    cout << ans << endl;
    return 0;
}
```

1163 Tour

这是双调欧几里得旅行商问题，我们将*n*个点按照横坐标从小到大的顺序排列后，定义*F[i,j]*为第*i*个点到第1个点再到第*j*点的最短距离。



状态转移方程为

$$F[i, j] = F[i - 1, j] + dist[i, i - 1] \quad \text{if } j < i - 1$$

$$F[i, j] = \min\{F[i - 1, k] + dist[i, k]\} \quad \text{if } j \geq i - 1$$

根据状态转移方程得到相应的程序为

```
for (int i = 3; i <= n; i++) {
    for (int j = 1; j < i; j++) {
        if (j < i - 1) {
            cal[i][j] = cal[i - 1][j] + dist(p[i], p[i - 1]);
        }
        else {
            cal[i][j] = cal[i - 1][1] + dist(p[i], p[1]);
            for (int k = 2; k < i - 1; k++) {
                cal[i][j] = min(cal[i - 1][k] + dist(p[i], p[k]), cal[i][j]);
            }
        }
    }
}
```

1345 能量项链

实际就是矩阵连乘问题，只不过考虑到是个环，需要把这个环展开成两倍的链， $m[a, b]$ 表示从第a个珠子到第b个珠子所产生的最大能量，状态转移方程如下

$$m[a, b] = \max\{m[a, k] + m[k, b] + arr[a] * arr[k] * arr[b]\}$$

$$k \in [a + 1, b - 1]$$

根据状态转移方程得到程序如下。

```
for (int j = 3; j <= 2*n; j++) {
    for (int i = j - 2; i >= 1; i--) {
        energy[i][j] = 0;
        for (int k = i + 1; k < j; k++) {
            energy[i][j] = max(energy[i][k] + energy[k][j] + m[i] * m[j] * m[k], energy[i][j]);
        }
    }
}
```

13062 SubDiagonal Paths

这个实际就是一个进栈出栈问题，把向右走看做入栈，向上走看做出栈，即n个数执行出栈入栈操作后可能的排列组合个数。

状态转移方程为

$$F[n] = F[n - 1] * F[0] + F[n - 2] * F[1] + F[n - 3] * F[2] + \dots + F[0] * F[n - 1]$$

转为程序如下

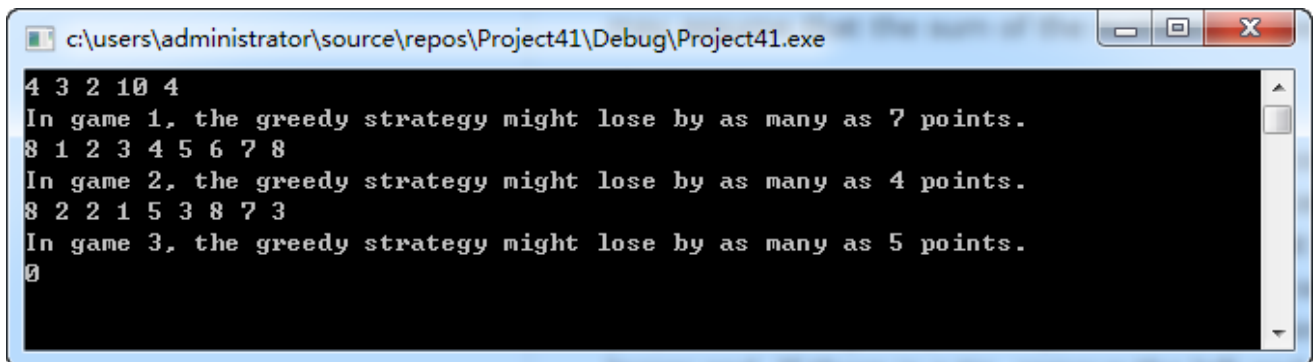

```

for (int i = 4; i <= 31; i++) {
    f[i] = 0;
    for (int j = 0; j < i; j++) {
        f[i] += f[i - j - 1] * f[j];
    }
}

```

4.程序运行与测试

1176 Two Ends

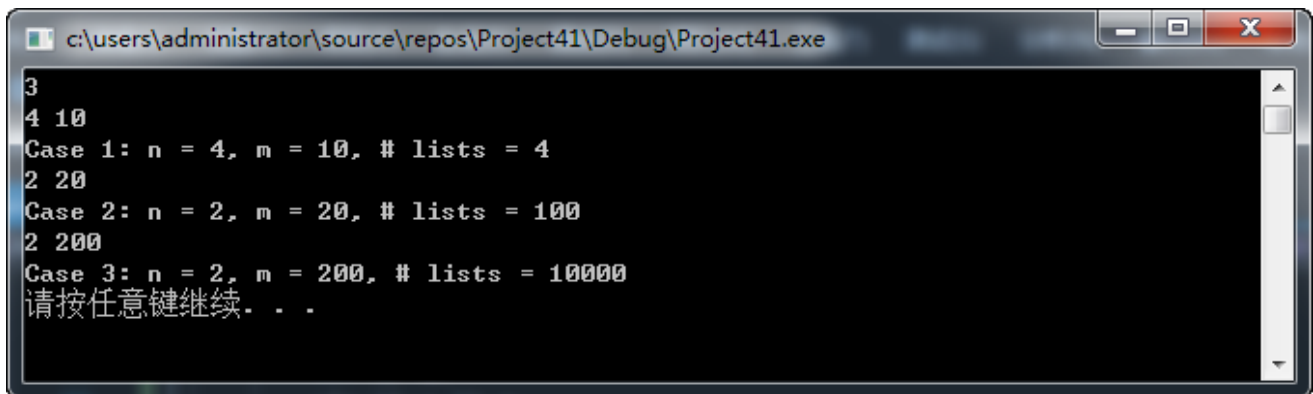


```

c:\users\administrator\source\repos\Project41\Debug\Project41.exe
4 3 2 10 4
In game 1, the greedy strategy might lose by as many as 7 points.
8 1 2 3 4 5 6 7 8
In game 2, the greedy strategy might lose by as many as 4 points.
8 2 2 1 5 3 8 7 3
In game 3, the greedy strategy might lose by as many as 5 points.
0

```

1011 Lenny's Lucky Lotto

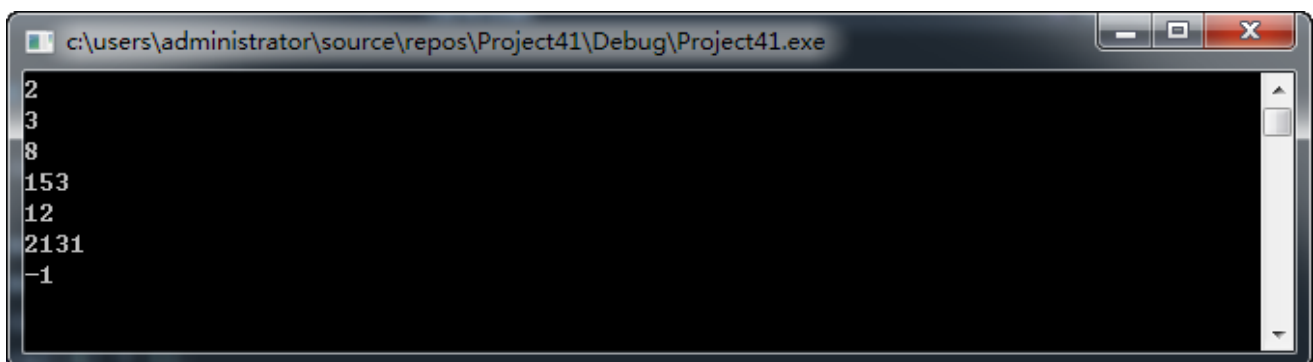


```

c:\users\administrator\source\repos\Project41\Debug\Project41.exe
3
4 10
Case 1: n = 4, m = 10, # lists = 4
2 20
Case 2: n = 2, m = 20, # lists = 100
2 200
Case 3: n = 2, m = 200, # lists = 10000
请按任意键继续. . .

```

1121 Tri Tiling

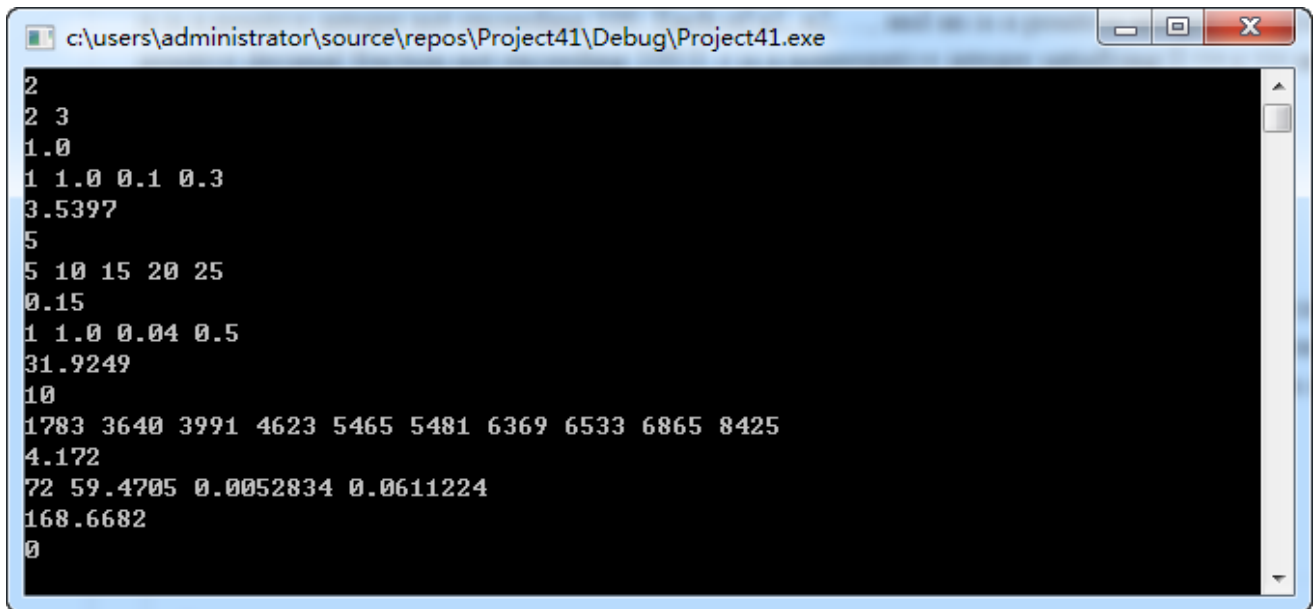


```

c:\users\administrator\source\repos\Project41\Debug\Project41.exe
2
3
8
153
12
2131
-1

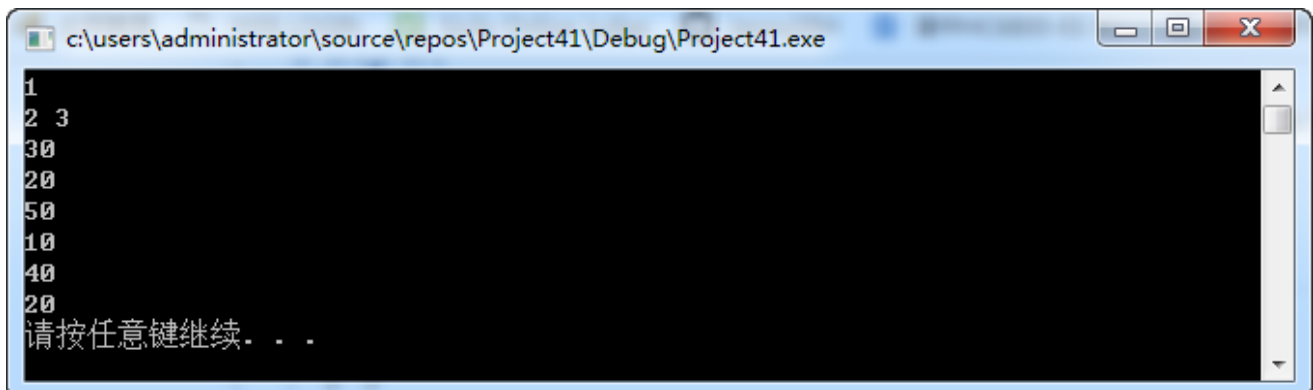
```

1264 Atomic Car Race[Special judge]



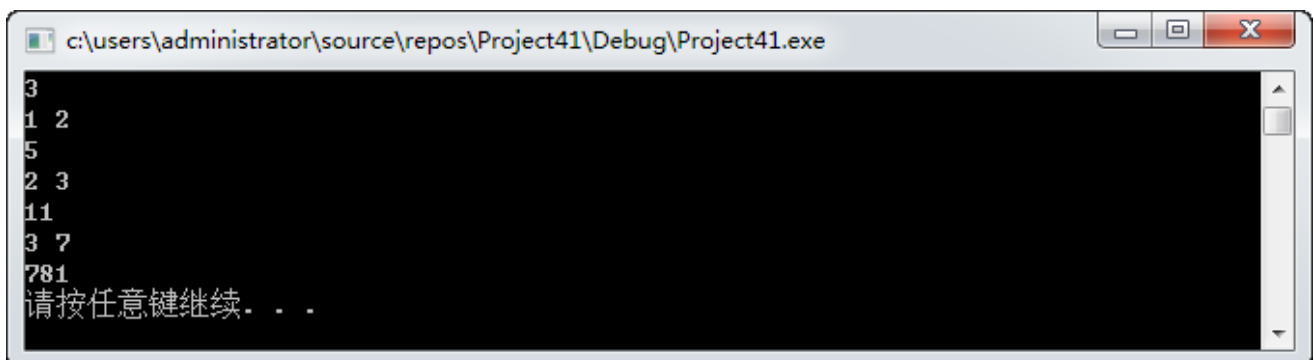
```
c:\users\administrator\source\repos\Project41\Debug\Project41.exe
2
2 3
1.0
1 1.0 0.1 0.3
3.5397
5
5 10 15 20 25
0.15
1 1.0 0.04 0.5
31.9249
10
1783 3640 3991 4623 5465 5481 6369 6533 6865 8425
4.172
72 59.4705 0.0052834 0.0611224
168.6682
0
```

1828 Minimal



```
c:\users\administrator\source\repos\Project41\Debug\Project41.exe
1
2 3
30
20
50
10
40
20
请按任意键继续. . .
```

1527 Tiling a Grid With Dominoes



```
c:\users\administrator\source\repos\Project41\Debug\Project41.exe
3
1 2
5
2 3
11
3 7
781
请按任意键继续. . .
```

1148 过河

```
c:\users\administrator\source\repos\Project41\Debug\Project41.exe
10
2 3 5
2 3 5 6 7
2
请按任意键继续. . .
```

1163 Tour

```
c:\users\administrator\source\repos\Project41\Debug\Project41.exe
3
1 1
2 3
3 1
6.47
4
1 1
2 3
3 1
4 2
7.89
```

1345 能量项链

```
c:\users\administrator\source\repos\Project41\Debug\Project41.exe
4
2 3 5 10
710
4
2 3 5 10
710
```

13062 SubDiagonal Paths

```
c:\users\administrator\source\repos\Project41\Debug\Project41.exe
1
1
2
2
3
5
4
14
0
```

5.实验总结与心得

本次实验比第一次实验困难很多，尤其是在头脑不清醒的时候特别容易把状态方程写错。学会动态规划最重要的就是要写出状态转移方程，这就需要我对状态的变化非常清晰，完成了这十道题目之后，我的能力也有了很大的提高，收获很大。

附录、提交文件清单

```
//1148
#include <iostream>
#include <string>
#include <vector>
#include <memory.h>
#include <algorithm>
#include <math.h>

using namespace std;

int rocks[20000];

int F[20000];

int min(int a, int b) {
    return (a < b ? a : b);
}

int stone_pos[200];

int L, min_step, max_step, n;

int main() {
    memset(rocks, 0, sizeof(rocks));
    cin >> L >> min_step >> max_step >> n;
    for (int i = 1; i <= n; i++) {
        cin >> stone_pos[i];
    }
    stone_pos[0] = 0;
    sort(stone_pos, stone_pos + n + 1);
    // 特殊情况判断
    if (min_step == max_step) {
        int ans = 0;
        for (int i = 1; i <= n; i++)
            if (stone_pos[i] % min_step == 0)
                ans++;
        cout << ans << endl;
        return 0;
    }
    // 压缩到rocks里
    int index;
    stone_pos[0] = 0;

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

```

        if (stone_pos[i] - stone_pos[i - 1] > 100) {
            index = stone_pos[i - 1] + 100;
            int pos = stone_pos[i] - (stone_pos[i - 1] + 100);
            for (int j = i; j <= n; j++) {
                stone_pos[j] -= pos;
            }
        }
        else {
            index = stone_pos[i];
        }
        rocks[index] = 1;
    }
    if (L - index > 100) {
        L = index + 100;
    }

    for (int i = 0; i <= L; i++) {
        F[i] = 999;
    }

    for (int i = min_step; i <= max_step; i++) {
        if (rocks[i] == 0) {
            F[i] = 0;
        }
        else {
            F[i] = 1;
        }
    }

    for (int i = max_step+1; i <= L; i++) {
        for (int k = min_step; k <= max_step && i-k >= 1; k++) {
            F[i] = min(F[i], F[i - k]);
        }
        if (rocks[i]) {
            F[i]++;
        }
    }
    cout << F[L] << endl;
    system("pause");
}

```

//1163

```

#include <iostream>
#include <string>
#include <vector>
#include <memory.h>
#include <algorithm>
#include <math.h>
#include <iomanip>

```

```

using namespace std;

```

```

double cal[100][100];

struct point {
    double x, y;
    point() {

    }
    point(double x, double y) {
        this->x = x;
        this->y = y;
    }
};

point p[100];

double dist(point a, point b) {
    return sqrt(double((a.x - b.x)*(a.x - b.x)) + double((a.y - b.y)*(a.y - b.y)));
}

double min(double a, double b) {
    return (a < b ? a : b);
}

int main() {
    int n;
    while (cin >> n) {
        for (int i = 1; i <= n; i++) {
            double x, y;
            cin >> x >> y;
            p[i].x = x;
            p[i].y = y;
        }
        cal[1][1] = 0;
        cal[2][1] = dist(p[2], p[1]);
        for (int i = 3; i <= n; i++) {
            for (int j = 1; j < i; j++) {
                if (j < i - 1) {
                    cal[i][j] = cal[i - 1][j] + dist(p[i], p[i - 1]);
                }
                else {
                    cal[i][j] = cal[i - 1][1] + dist(p[i], p[1]);
                    for (int k = 2; k < i-1; k++) {
                        cal[i][j] = min(cal[i - 1][k] + dist(p[i], p[k]), cal[i][j]);
                    }
                }
            }
        }
        cout << fixed << setprecision(2) << cal[n][n - 1] + dist(p[n], p[n - 1]) << endl;
    }
}

```

```

//1264
#include <iostream>
#include <string>
#include <vector>
#include <memory.h>
#include <algorithm>
#include <math.h>
#include <iomanip>
using namespace std;

int a[110];
double t[110];
double b, r, v, e, f;

double min(double a, double b) {
    return (a < b ? a : b);
}

int distance(int k, int i) {
    return abs(a[i] - a[k]);
}

double G[10010];

int main() {
    int n;
    while (cin >> n) {
        if (n == 0) {
            return 0;
        }
        a[0] = 0;
        for (int i = 1; i <= n; i++) {
            cin >> a[i];
        }
        cin >> b >> r >> v >> e >> f;
        t[0] = 0;
        G[0] = 0;
        for (int i = 0; i <= a[n]; i++) {
            double temp = (i >= r) ? 1 / (v - e * (i - r)) : 1 / (v - f * (r - i));
            G[i + 1] = G[i] + temp;
        }
        t[1] = G[a[1]];
        for (int i = 2; i <= n; i++) {
            t[i] = G[a[i]];
            for (int k = i - 1; k >= 1; k--) {
                t[i] = min(t[i], t[k] + b + G[distance(k, i)]);
            }
        }
        cout << fixed << setprecision(4) << t[n] << endl;
    }
}

```

```

//1345
#include <iostream>
#include <string>
#include <vector>
#include <memory.h>
#include <algorithm>
#include <math.h>

using namespace std;

long long energy[210][210];

int m[210];

long long max(long long a, long long b) {
    return (a > b ? a : b);
}

int main() {
    int n;
    while (cin >> n) {
        for (int i = 1; i <= n; i++) {
            cin >> m[i];
            m[i + n] = m[i];
        }
        m[2*n + 1] = m[1];
        for (int j = 3; j <= 2*n; j++) {
            for (int i = j-2; i >= 1; i--) {
                energy[i][j] = 0;
                for (int k = i + 1; k < j; k++) {
                    energy[i][j] = max(energy[i][k] + energy[k][j] + m[i] * m[j] * m[k],
energy[i][j]);
                }
            }
        }
        int max_num = 0;
        for (int i = 1; i <= n; i++) {
            max_num = max(max_num, energy[i][i + n]);
        }
        cout << max_num << endl;
    }
}

```

```

//13062
// 实际就是进栈出栈问题
#include <iostream>
#include <string>
#include <vector>
#include <memory.h>

#include <algorithm>

```



```

#include <math.h>

using namespace std;

int main() {
    long long f[32];
    f[0] = 1;
    f[1] = 1;
    f[2] = 2;
    f[3] = 5;
    for (int i = 4; i <= 31; i++) {
        f[i] = 0;
        for (int j = 0; j < i; j++) {
            f[i] += f[i - j - 1] * f[j];
        }
    }
    int n;
    while (true) {
        cin >> n;
        if (n == 0) {
            return 0;
        }
        cout << f[n] << endl;
    }
}

```

//1527

```

#include <iostream>
#include <string>
#include <vector>
#include <memory.h>
#include <algorithm>
#include <math.h>

using namespace std;

int m[1010][16];
int main() {
    int T;
    cin >> T;
    int count = 0;

    m[1][0] = m[1][3] = m[1][6] = m[1][12] = m[1][15] = 1;
    for (int i = 2; i <= 1000; i++) {
        m[i][0] = m[i - 1][15];
        m[i][3] = m[i - 1][15] + m[i - 1][12];
        m[i][6] = m[i - 1][15] + m[i - 1][9];
        m[i][9] = m[i - 1][6];
        m[i][12] = m[i - 1][15] + m[i - 1][3];
        m[i][15] = m[i - 1][15] + m[i - 1][12] + m[i - 1][6] + m[i - 1][3] + m[i - 1][0];
    }
}

```

```

        while (T--) {
            count++;
            cout << count << " ";
            int n;
            cin >> n;
            cout << m[n][15] << endl;
        }
    }
}

```

//1828

```

#include <iostream>
#include <string>
#include <vector>
#include <memory.h>
#include <algorithm>
#include <math.h>

using namespace std;

int min(int a, int b) {
    return (a < b ? a : b);
}

int p[510][510];
int s_arr1[510];
int s_arr2[510];
int main() {
    int T;
    cin >> T;
    while (T--) {
        int s1, s2;
        cin >> s1 >> s2;
        for (int i = 0; i < s1; i++) {
            cin >> s_arr1[i];
        }
        for (int i = 0; i < s2; i++) {
            cin >> s_arr2[i];
        }
        sort(s_arr1, s_arr1 + s1);
        sort(s_arr2, s_arr2 + s2);
        memset(p, 9999999, sizeof(p));
        int min_num = 9999999;
        for (int i = 1; i <= s2; i++) {
            if (abs(s_arr1[0] - s_arr2[i - 1]) < min_num) {
                min_num = abs(s_arr1[0] - s_arr2[i - 1]);
            }
            p[1][i] = min_num;
        }
        for (int i = 2; i <= s1; i++) {

            for (int j = i; j <= s2; j++) {

```

```

        p[i][j] = min(p[i][j - 1], p[i - 1][j - 1] + abs(s_arr1[i - 1] - s_arr2[j -
1]));
    }
}
cout << p[s1][s2] << endl;
}
}

```

//1121

```

#include <iostream>
#include <string>
#include <vector>
#include <memory.h>
#include <algorithm>

```

```
using namespace std;
```

```
int cal[40];
```

```

int main() {
    memset(cal, 0, sizeof(cal));
    cal[0] = 1;
    cal[2] = 3;
    for (int i = 4; i <= 31; i += 2) {
        cal[i] = cal[i - 2] * 4 - cal[i - 4];
    }
    int n;
    while (true) {
        cin >> n;
        if (n == -1) {
            return 0;
        }
        cout << cal[n] << endl;
    }
}

```

// 1011

```

#include <iostream>
#include <string>
#include <vector>
#include <memory.h>
#include <algorithm>

```

```
using namespace std;
```

```

int max(int a, int b) {
    return (a > b ? a : b);
}

```

```
long long list[20][2020];
```

```

long long cal_n_m(int n, int m) {
    for (int i = 1; i <= m; i++) {
        list[1][i] = i;
    }

    for (int i = 2; i <= n; i++) {
        for (int j = i; j <= m; j++) {
            // 选择j或者不选j的和
            list[i][j] = list[i - 1][j / 2] + list[i][j - 1];
        }
    }
    return list[n][m];
}

int main() {
    int T;
    cin >> T;
    int count = 0;
    while (T--) {
        count++;
        int n, m;
        cin >> n >> m;
        memset(list, 0, sizeof(list));
        long long ans = cal_n_m(n, m);
        cout << "Case " << count << ": n = " << n << ", m = " << m << ", # lists = " << ans <<
endl;
    }
}

// 1176
#include <iostream>
#include <string>
#include <vector>
#include <memory.h>
#include <algorithm>

using namespace std;
int arr[1010];
int cal[1010][1010];

int max(int a, int b) {
    return (a > b ? a : b);
}

int cal_a_b(int a, int b) {
    int leftsum;
    int rightsum;
    if (a > b) return 0;
    if (cal[a][b] != -1) {
        return cal[a][b];
    }

    if (arr[a + 1] >= arr[b]) {

```

```

        leftsum = cal_a_b(a + 2, b) + arr[a];
    }
    else {
        leftsum = cal_a_b(a + 1, b - 1) + arr[a];
    }
    if (arr[a] >= arr[b - 1]) {
        rightsum = cal_a_b(a + 1, b - 1) + arr[b];
    }
    else {
        rightsum = cal_a_b(a, b - 2) + arr[b];
    }
    cal[a][b] = max(leftsum, rightsum);
    return cal[a][b];
}

int main() {
    int n;
    int count = 0;
    while (true) {
        cin >> n;
        if (n == 0) {
            return 0;
        }
        memset(cal, -1, sizeof(cal));
        count++;
        int sum = 0;
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
            sum += arr[i];
        }
        int win_sum = cal_a_b(0, n - 1);
        int lose_sum = sum - win_sum;
        cout << "In game " << count << ", the greedy strategy might lose by as many as " <<
win_sum - lose_sum << " points." << endl;
    }
}

```