动态规划-数字三角形模型

一、AcWing 1015. 摘花生

【题目描述】

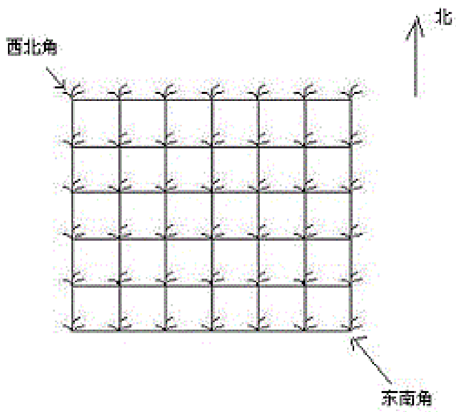
Hello Kitty想摘点花生送给她喜欢的米老鼠。

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。



【输入格式】

第一行是一个整数T，代表一共有多少组数据。

接下来是T组数据。

每组数据的第一行是两个整数，分别代表花生苗的行数R和列数C。

每组数据的接下来R行数据，从北向南依次描述每行花生苗的情况。每行数据有C个整数，按从西向东的顺序描述了该行每株花生苗上的花生数目M。

【输出格式】

对每组输入数据，输出一行，内容为Hello Kitty能摘到得最多的花生颗数。

【数据范围】

$1 \leq T \leq 100$

$1 \leq R, C \leq 100$

$0 \leq M \leq 1000$

【输入样例】

```
2
2 2
1 1
3 4
2 3
2 3 4
1 6 5
```

【输出样例】

```
8
16
```

【分析】

状态表示：

- 集合：$f[i][j]$为从$(1,1)$到达(i,j)的所有方案的集合。
- 属性：最大值

状态转移：

- 从(i,j)的上方$(i-1,j)$转移过来，即$f[i-1][j]$
- 从(i,j)的左方$(i,j-1)$转移过来，即$f[i][j-1]$

所以最终的状态转移方程为：$f[i][j]=max(f[i-1][j],f[i][j-1])+g[i][j]$，$g[i][j]$表示$(i,j)$上的花生数量。

【代码】

```
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 110;
int g[N][N], f[N][N];
int n, m;

int main()
{
    int T;
    cin >> T;
    while (T--)
    {

        cin >> n >> m;
```

```
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
                cin >> g[i][j];
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
                f[i][j] = max(f[i - 1][j], f[i][j - 1]) + g[i][j];
        cout << f[n][m] << endl;
    }
    return 0;
}
```

二、AcWing 1018. 最低通行费

【题目描述】

一个商人穿过一个$N \times N$的正方形的网格，去参加一个非常重要的商务活动。

他要从网格的左上角进，右下角出。

每穿越中间1个小方格，都要花费1个单位时间。

商人必须在$(2N-1)$个单位时间穿越出去。

而在经过中间的每个小方格时，都需要缴纳一定的费用。

这个商人期望在规定时间内用最少费用穿越出去。

请问至少需要多少费用？

注意：不能对角穿越各个小方格（即，只能向上下左右四个方向移动且不能离开网格）。

【输入格式】

第一行是一个整数，表示正方形的宽度N。

后面N行，每行N个不大于100的正整数，为网格上每个小方格的费用。

【输出格式】

输出一个整数，表示至少需要的费用。

【数据范围】

$1 \le N \le 100$

【输入样例】

```
5
1  4  6  8  10
2  5  7  15 17
6  8  9  18 20
10 11 12 19 21
20 23 25 29 33
```

【输出样例】

```
109
```

【分析】

首先需要注意题中要求必须在$(2N-1)$个单位时间穿越出去，不难证明，商人**不能走回头路**，即只能向右或向下走，因此与问题一相似。

状态表示：

- 集合：$f[i][j]$为从$(1,1)$到达(i,j)的所有方案的集合。
- 属性：最小值

状态转移：

- 从(i,j)的上方$(i-1,j)$转移过来，即$f[i-1][j]$
- 从(i,j)的左方$(i,j-1)$转移过来，即$f[i][j-1]$

所以最终的状态转移方程为：$f[i][j]=min(f[i-1][j],f[i][j-1])+g[i][j]$，$g[i][j]$表示$(i,j)$上的花费。

注意：本题需要求的属性为最小值，因此需要注意f的边界。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 110;
int g[N][N], f[N][N];
int n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cin >> g[i][j];
    memset(f, 0x3f, sizeof f);
```

```
    f[1][1] = g[1][1];//初始化第一个点
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (i != 1 || j != 1)//不更新第一个点
                f[i][j] = min(f[i - 1][j], f[i][j - 1]) + g[i][j];
    cout << f[n][n] << endl;
    return 0;
}
```

三、AcWing 1027. 方格取数

【题目描述】

设有$N \times N$的方格图，我们将其中的某些方格中填入正整数，而其他的方格中则放入数字0。如下图所示（见样例）：

```
A
   0  0  0  0  0  0  0  0
   0  0 13  0  0  6  0  0
   0  0  0  0  7  0  0  0
   0  0  0 14  0  0  0  0
   0 21  0  0  0  4  0  0
   0  0 15  0  0  0  0  0
   0 14  0  0  0  0  0  0
   0  0  0  0  0  0  0  0
                            B
```

某人从图的左上角的A点出发，可以向下行走，也可以向右走，直到到达右下角的B点。在走过的路上，他可以取走方格中的数（取走后的方格中将变为数字0）。

此人从A点到B点共走两次，试找出2条这样的路径，使得取得的数之和为最大。

【输入格式】

输入的第一行为一个整数N（表示$N \times N$的方格图），接下来的每行有三个整数，前两个表示位置，第三个数为该位置上所放的数。一行单独的0表示输入结束。

【输出格式】

只需输出一个整数，表示2条路径上取得的最大的和。

【数据范围】

$N \le 10$

【输入样例】

```
8
2 3 13
2 6 6
3 5 7
4 4 14
5 2 21
5 6 4
6 3 15
7 2 14
0 0 0
```

【输出样例】

```
67
```

【分析】

本题的走法与问题一类似，但是需要走两次，因此可以进行类比，想象成两个人**同时走**，如果走到同一个点上那么特判一下即可。

状态表示：

- 集合：$f[i][j][k][l]$表示从$(1,1),(1,1)$分别走到$(i,j),(k,l)$的所有方案的集合。
- 属性：最大值

状态转移：

- 第一条路径从$(i-1,j)$走来，第二条路径从$(k-1,l)$走来，即$f[i-1][j][k-1][l]$
- 第一条路径从$(i-1,j)$走来，第二条路径从$(k,l-1)$走来，即$f[i-1][j][k][l-1]$
- 第一条路径从$(i,j-1)$走来，第二条路径从$(k-1,l)$走来，即$f[i][j-1][k-1][l]$
- 第一条路径从$(i,j-1)$走来，第二条路径从$(k,l-1)$走来，即$f[i][j-1][k][l-1]$

所以最终的状态转移方程为：$f[i][j][k][l]=max(f[i-1][j][k-1][l],f[i-1][j][k][l-1],f[i][j-1][k-1][l],f[i][j-1][k][l-1])+g[i][j]+g[k][l]$，如果两条路径走到了同一个位置（$i=k\,\&\&\,j=l$），那么需要减去一次该点的数，即$f[i][j][k][l]-=g[i][j]$。

优化：

如何保证两条路径同时走呢？答案是当两条路径的横坐标与纵坐标之和相等时。

因此我们可以用s来表示两条路径的横纵坐标之和，这样就只需要记录两条路径的横坐标即可。

状态表示：

- 集合：$f[s][i][k]$表示从$(1,1),(1,1)$分别走到$(i,s-i),(k,s-k)$的所有方案的集合。
- 属性：最大值

状态转移：思路同上，具体见以下代码。

【四维写法代码】

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 15;
int f[N][N][N][N];
int g[N][N];
int n;

int main()
{
    cin >> n;
    int a, b, c;
    while (cin >> a >> b >> c, a || b || c) g[a][b] = c;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            for (int k = 1; k <= n; k++)
                for (int l = 1; l <= n; l++)
                {
                    f[i][j][k][l] = max(max(f[i - 1][j][k - 1][l], f[i - 1][j][k][l - 1]),
                        max(f[i][j - 1][k - 1][l], f[i][j - 1][k][l - 1])) + g[i][j] + g[k][l];
                    if (i == k && j == l) f[i][j][k][l] -= g[i][j];//判断是否走到同一个格子
                }
    cout << f[n][n][n][n] << endl;
    return 0;
}
```

【三维写法代码】

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 15;
int f[N << 1][N][N];
int g[N][N];
int n;

int main()
{
    cin >> n;
    int a, b, c;
```

```cpp
    while (cin >> a >> b >> c, a || b || c) g[a][b] = c;
    for (int s = 2; s <= n << 1; s++)
        for (int i = 1; i <= n && i < s; i++)
            for (int k = 1; k <= n && k < s; k++)
            {
                int j = s - i, l = s - k;
                if (j <= n && l <= n)//如果纵坐标在边界内
                {
                    f[s][i][k] = max(max(f[s - 1][i - 1][k], f[s - 1][i - 1][k - 1]),
                        max(f[s - 1][i][k], f[s - 1][i][k - 1])) + g[i][j] + g[k][l];
                    if (i == k) f[s][i][k] -= g[i][j];//判断是否走到同一个格子
                }
            }
    cout << f[n << 1][n][n] << endl;
    return 0;
}
```

四、AcWing 275. 传纸条

【题目描述】

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，班上同学安排坐成一个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n和m，表示班里有n行m列。

接下来的n行是一个$n\times m$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m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。

【数据范围】

$1 \le n, m \le 50$

【输入样例】

```
3 3
0 3 9
2 8 5
5 7 0
```

【输出样例】

```
34
```

【分析】

本题的分析过程和第三题完全一样，将来回的两条路径看成从起点开始同时走的两条路径即可。

【四维代码写法】

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 55;
int f[N][N][N][N];
int g[N][N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> g[i][j];
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            for (int k = 1; k <= n; k++)
                for (int l = 1; l <= m; l++)
                {
                    f[i][j][k][l] = max(max(f[i - 1][j][k - 1][l], f[i - 1][j][k][l - 1]),
                        max(f[i][j - 1][k - 1][l], f[i][j - 1][k][l - 1])) + g[i][j] + g[k][l];
                    if (i == k && j == l) f[i][j][k][l] -= g[i][j];
                }
    cout << f[n][m][n][m] << endl;
    return 0;
```

```
}
```

【三维代码写法】

```
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 55;
int f[N << 1][N][N];
int g[N][N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> g[i][j];
    for (int s = 2; s <= n + m; s++)
        for (int i = 1; i <= n && i < s; i++)
            for (int k = 1; k <= n && k < s; k++)
            {
                int j = s - i, l = s - k;
                if (j <= m && l <= m)
                {
                    f[s][i][k] = max(max(f[s - 1][i - 1][k], f[s - 1][i - 1][k - 1]),
                        max(f[s - 1][i][k], f[s - 1][i][k - 1])) + g[i][j] + g[k][l];
                    if (i == k) f[s][i][k] -= g[i][j];
                }
            }
    cout << f[n + m][n][n] << endl;
    return 0;
}
```