坐标类DP

主讲老师：党东

【坐标型动态规划】

坐标型动态规划，顾名思义，此类动态规划与坐标位置有很大的关系。

它的状态划分与维数有关。如用二维坐标系f[i,j]表示状态，它的值一般代表x坐标为i，y坐标为j时的最佳值，它与(i-1,j)或(i,j-1)的值有很大关系，一般采用递推方法来实现。

【坐标型动态规划】

一、计算所有方案：f[i][j]=f[i-1][j]+f[i][j-1]

【例题】过河卒1209

【坐标型动态规划】

二、计算一条最佳路径：f[i][j]=min(f[i-1][j],f[i][j-1])+a[i][j]

【例题】最小伤害1271

【例题】数字金字塔1346

【例1】拾垃圾的机器人 --1495

【问题描述】在一个n*m的棋盘内，一些格子里有垃圾要拾捡。现在有一个能捡垃圾的机器人从左上格子里出发，每次只能向右或者向下走。每次他到达一个点，就会自动把这个点内的垃圾拾掉。

最多能拾多少垃圾。在最多的情况下，有多少种拾垃圾方案?

【数据范围】n<=100,m<=100

【例1】拾垃圾的机器人 --1495

【文件输入】

3 3
100
000
010

【文件输出】

2
3

【例1】拾垃圾的机器人 --1495

【核心代码】

状态：p[i][j]表示走到（i,j）这个位置时，最多可拾到的垃圾数

状态：q[i][j]表示走到（i,j）这个位置时，在拾到的垃圾数最多情况下的方案数

```
p[i][j]=x-'0';
p[i][j]+=max(p[i-1][j],p[i][j-1]);
if(a[i-1][j]<a[i][j-1]) q[i][j]+=q[i][j-1];
else if(p[i-1][j]>p[i][j-1]) q[i][j]+=q[i-1][j];
else q[i][j]+=q[i-1][j]+q[i][j-1];
```

【例2】方格取数 --1496

Description

给定一个 N * M 的矩阵， 记录左上角为 (1,1)， 右下角为 (N , M)， 现在从 (1,1)始取数， 每次只能向下或向右移动一个单位， 最终到达 (N , M)， 我们把路径上有的数相乘， 记为 C 。 使 C 的结果最大已经不能满足我们了， 现在我们想让 C尾的零最少 。

Ps. 11000 末尾有 3 个零， 100000100 末尾有 2 个零。

Input

第一行包含 两个 正整数 N， M 表示矩阵大小。

接下来 N 行每行 M 个正整数给出整个矩阵。

Output

包含一个整数表示所求最小值。

Sample Input

```
3 3
1 2 3
10 5 100
10 8 9
```

Sample Output

```
1
```

【例2】方格取数 --1496

【题目分析】

这道题是一个很明显的动态规划。我们不难得知，最后C末位的0的个数与我们路上取到的数的因数中2与5的个数有关。

我们知道，**一个2和一个5相乘会得到一个0**，所以，这道题也就变成了：

从（1,1）到（n,m）分别求出一条**含2的个数最少**与**含5的个数最少**的路径

在这两条路径中选择2或5的个数较小的那一条，这条路即是我们要求得的路径

需要注意的细节问题：

①方格中每个数因子中2和5的个数需要预处理

②递推前f[i][j]的边界情况一定要处理好

【例3】传纸条(NOIP2008) --1497

Description

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。

班里每个同学都可以帮他们传递，但只会帮他们一次。

每个同学愿意帮忙的好感度有高有低，可以用一个0-100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只和最大。现在，请你帮助小渊和小轩找到这样的两条路径。

Input

第一行有2个用空格隔开的整数m和n，表示班里有m行n列（1 < m,n <= 50）。

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。

Output

包含一个整数，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。

【例3】传纸条(NOIP2008) --1497

Sample Input

3 3
0 3 9
2 8 5
5 7 0

Sample Output

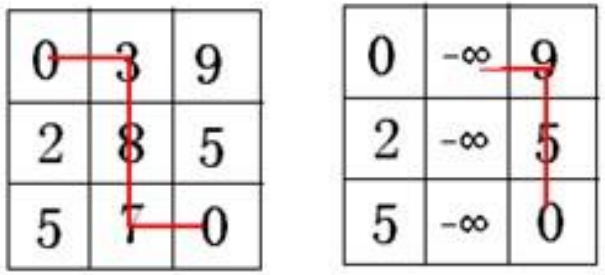
34

【例3】传纸条(NOIP2008) --1497

【题目分析】

贪心

- 很容易想到一个算法：
  - 求出1个纸条从（1，1）到（M,N）的路线最大值.
  - 删除路径上的点值
  - 再求出1个纸条从（M,N） 到（1，1）的路线最大值.
  - 统计两次和
- 上述算法很容易找出反例，如下图。



第1次传递　　第2次传递

- 第1次找最优值传递后，导致第2次无法传递。

【例3】传纸条(NOIP2008) --1497

【题目分析】

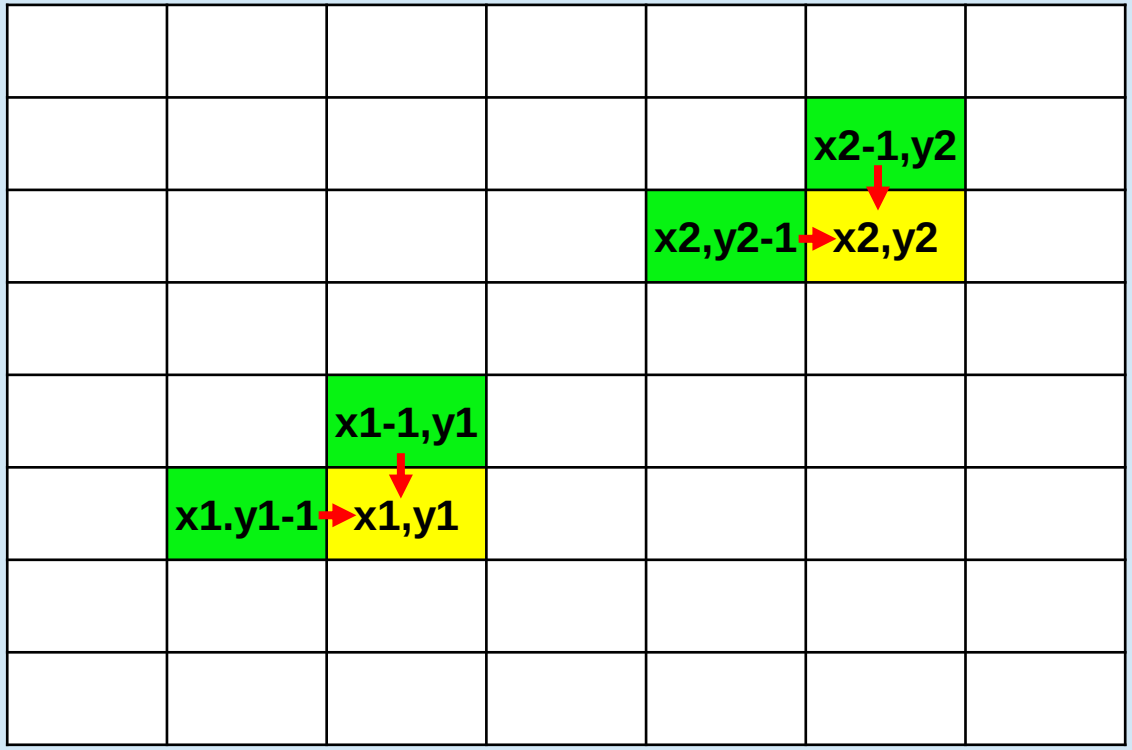
- 贪心算法错误，因此我们需要同时考虑两个纸条的传递。
- 由于小渊和小轩的路径可逆，因此，尽管出发点不同，但都可以看成同时从(1,1)出发到达(M,N)点。
- 设f(i_1,j_1,i_2,j_2)表示纸条1到达(i_1,j_1)位置，纸条2到达(i_2,j_2)位置的最优值。则有，

$$f(i_1,j_1,i_2,j_2)=\max\begin{Bmatrix} f(i_1-1,j_1,i_2-1,j_2) \\ f(i_1,j_1-1,i_2-1,j_2) \\ f(i_1-1,j_1,i_2,j_2-1) \\ f(i_1,j_1-1,i_2,j_2-1) \end{Bmatrix}+C[i_1,j_1]+C[i_2,j_2]$$

- 其中 (i_1,j_1)<> (i_2,j_2)
- 1<=i_1, i_2<=M, 1<=j_1 , j_2<=N
- 时间复杂度O(N^2M^2)

【例3】传纸条(NOIP2008) --1497

【题目分析】



【例3】传纸条(NOIP2008) --1497

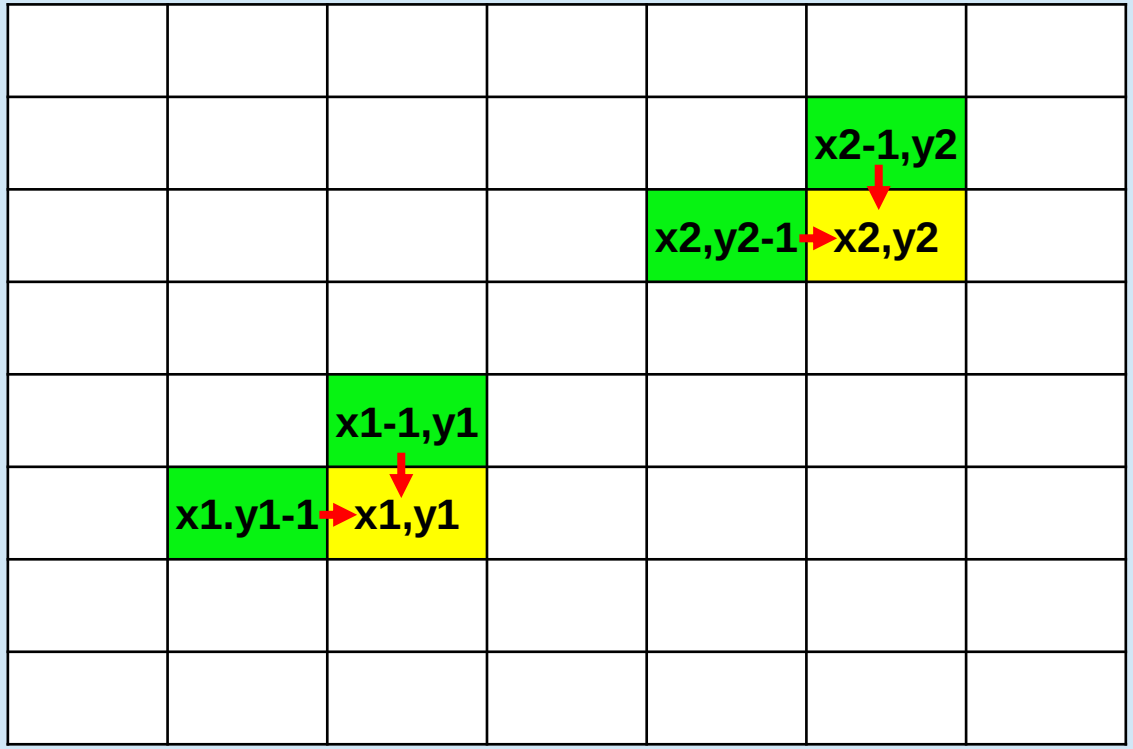
【核心代码】

```
f[1][1][1][1]=a[1][1];//初始化
for(i=1;i<=m;i++)
   for(j=1;j<=n;j++)
      for(x=1;x<=m;x++)
        for(y=1;y<=n;y++)
        {  t=0;
           if(f[i-1][j][x-1][y]>t)t=f[i-1][j][x-1][y];
           if(f[i][j-1][x-1][y]>t)t=f[i][j-1][x-1][y];
           if(f[i-1][j][x][y-1]>t)t=f[i-1][j][x][y-1];
           if(f[i][j-1][x][y-1]>t)t=f[i][j-1][x][y-1];
           if(i==x&&j==y)f[i][j][x][y]=t+a[i][j];
                 else f[i][j][x][y]=t+a[i][j]+a[x][y];
        }
cout<<f[m][n][m][n];
```

【例3】传纸条(NOIP2008) --1497

【利用决策同步性优化代码】

因为两个坐标的位置是同步决策的状态：dp[x1][y1][x2][y2]，
x1+y1=x2+y2.将上述公式稍微变形：x1+y1-x2=y2。
即只要x1+y1决定当前阶段后，另一张纸条只要x2确定，则位置状态均确定。
故状态可以 dp[x1][y1][x2]来表示。



	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	10
3	4	5	6	7	8	9	10	11
4	5	6	7	8	9	10	11	12
5	6	7	8	9	10	11	12	13
6	7	8	9	10	11	12	13	14
7	8	9	10	11	12	13	14	15
8	9	10	11	12	13	14	15	16

【例3】传纸条(NOIP2008) --1497


```
int n,m,g[51][51];
int dp[51][51][51];  //代表传到坐标为(x1, y1)和(x2, y2)的同学的好感度最大值
  for(int x1=1;x1<=m;x1++)
    for(int y1=1;y1<=n;y1++)
      for(int x2=1;x2<=m;x2++)
      {
        int y2=x1+y1-x2;
        //因为两个坐标的位置是同步决策的状态：dp[x1][y1][x2][y2]，
        //且x1+y1=x2+y2.将上述公式稍微变形：x1+y1-x2=y2。
        if(y2<1) break; //超出边界，跳过
        int tmax=0;
        if(dp[x1-1][y1][x2-1]>tmax) tmax=dp[x1-1][y1][x2-1];
        if(dp[x1][y1-1][x2]>tmax) tmax=dp[x1][y1-1][x2];
        if(dp[x1-1][y1][x2]>tmax) tmax=dp[x1-1][y1][x2];
        if(dp[x1][y1-1][x2-1]>tmax) tmax=dp[x1][y1-1][x2-1];
        dp[x1][y1][x2]=tmax+g[x1][y1]+g[x2][y2];
        if(x1==x2&&y1==y2) dp[x1][y1][x2]-=g[x1][y1]; //减掉计算重复的位置
      }
  printf("%d",dp[m][n][m]);
```

【扩展】三取方格数 --1498

【题目背景】JerryZhou同学经常改编习题给自己做。 这天，他又改编了一题……

【问题描述】设有N*N的方格图，我们将其中的某些方格填入正整数，而其他的方格中放入0。某人从图得左上角出发，可以向下走，也可以向右走，直到到达右下角。在走过的路上，他取走了方格中的数。（取走后方格中数字变为0），　此人从左上角到右下角共走3次，试找出3条路径，使得取得的数总和最大。

【文件输入】第一行:N (4<=N<=20)，接下来一个N*N的矩阵，矩阵中每个元素不超过80，不小于0 。

【文件输出】一行，表示最大的总和。

【样例输入】

```
4
1 2 3 4
2 1 3 4
1 2 3 4
1 3 2 4
```

【样例输出】39

【扩展】三取方格数 --1498

【思路点拨】

```
//三取方格数 //dp[x][y][G][K]  X,Y一条路径，G,(x+y)-G一条路径，K,(x+y)-K一条路径
  for(int x1=1;x1<=n;x1++)
     for(int y1=1;y1<=n;y1++)
        for(int x2=1;x2<=n;x2++)
        {
           int y2=x1+y1-x2;
           if(y2<1) break;
           for(int x3=1;x3<=n;x3++)
           {
              int y3=x1+y1-x3;
              if(y3<1) break;
              int t=map[x1][y1]+map[x2][y2]+map[x3][y3];
              if(x1==x2&&y1==y2) t-=map[x1][y1];
              if(x1==x3&&y1==y3) t-=map[x1][y1];
              if(x2==x3&&y2==y3) t-=map[x2][y2];
              dp[x1][y1][x2][x3]=
              max(max(max(dp[x1-1][y1][x2-1][x3-1],dp[x1-1][y1][x2-1][x3]),max(dp[x1-1][y1][x2][x3-
1],dp[x1-1][y1][x2][x3])),max(max(dp[x1][y1-1][x2-1][x3-1],dp[x1][y1-1][x2-1][x3]),max(dp[x1][y1-
1][x2][x3-1], dp[x1][y1-1][x2][x3])))+t;
              }
           }
    printf("%d",dp[n][n][n][n]+map[1][1]+map[n][n]);//开头和结尾会被各多减一次
```