部分和问题类DP

主讲老师：党东

【部分和问题类动态规划】

在信息学竞赛中，部分和问题是指在给定序列或者矩阵中，求出相应要求的子序列或者子矩阵，并得到最终答案。

典型的题目有：

一、最大连续子序列和

二、最大K个连续子序列和（不重叠）

三、最大最大子矩阵问题

【部分和问题类动态规划】

最大连续子序列的和

【例1】最大连续子序列的和 --1523

【问题描述】给你一个长度为 n 的整数序列（A1,A2,……,An），要求从中找出一个连续的子序列，使得这个子序列的和最大。

【文件输入】输入文件的第一行为一个整数n，第二行为n个整数；

【文件输出】输出文件仅一行为该序列中最大连续子序列的和，数据保证和不会超过int。

【样例输入】

6

1 2 -5 6 7 8

【样例输出】21

【数据范围】

对于10的数据，n<=300;

对于30的数据，n<=4000;

对于50的数据，n<=100000;

对于100的数据，n<=10^7;

【例1】最大连续子序列的和 --1523

【问题分析】

1、阶段和状态：

以第i个数结尾的最大连续子序列的和，只存在两种选择：

情形1：只包含a[i]

情形2：包含a[i]和以a[i-1]结尾的最大连续和序列

故设f[i]表示以a[i]结尾的最大连续子序列的和

2、状态转移方程：

转移方程：f[i]=max{a[i],f[i-1]+a[i]}(2<=i<=n)

初始化：f[1]=a[1]

Answer=max{f[i]|1<=i<=n}

【例1】最大连续子序列的和 --1523

【参考代码】

```
int a[100001],f[100001],i,j,n,maxx;
int main()
{  cin>>n;
   for(i=1;i<=n;i++)scanf("%d",&a[i]);
   f[1]=a[1];
   for(i=2;i<=n;i++)f[i]=max(f[i-1],0)+a[i];
   maxx=-0x7fffffff/2;
   for(i=1;i<=n;i++)if(f[i]>maxx)maxx=f[i];
   cout<<maxx<<endl;
}
```

【部分和问题类动态规划】

最大连续子序列的积

【思考】最大连续子序列的积

【问题描述】给你一个长度为 n 的整数序列（A1,A2,……,An），要求从中找出一个连续的子序列，使得这个子序列的积最大。

【文件输入】输入文件的第一行为一个整数n(n<=100000)，第二行为n个整数；

【文件输出】输出文件仅一行为该序列中最大连续子序列的积。

【样例输入】

5

-5 3 9 10 -5

【样例输出】6750

【思考】最大连续子序列的积

【思路点拨】

解决了最大连续子序列和问题，最大连续子序列积问题就容易解决了。但是要注意的是，由于负数的存在，我们不能只简单保存一个当前最大值，我们还需要保存当前最小值。因为有可能一个负数乘上一个最小值(也是负数)会得到一个较大的积。

令f[n]表示[0,n]区间内，以n结尾的最大积，

令g[n]表示[0,n]区间内，以n结尾的最小积。

a[i]为原序列，则动态规划方程为：

F[i]=max{f[i-1]*a[i],a[i],g[i-1]*a[i]}

G[i]=min{f[i-1]*a[i],a[i],g[i-1]*a[i]}

时间复杂度为：O(n)

【思考】最大连续子序列的积

【思路点拨】

```
int a[100001],f[100001],g[100001],i,j,n,maxx;
cin>>n;
 for(i=1;i<=n;i++)scanf("%d",&a[i]);
 f[1]=a[1];g[1]=a[1];
 for(i=2;i<=n;i++)
 {  f[i]=max(a[i],max(f[i-1]*a[i],g[i-1]*a[i]));
    g[i]=min(a[i],min(f[i-1]*a[i],g[i-1]*a[i]));
 }
 maxx=-0x7fffffff/2;
 for(i=1;i<=n;i++)if(f[i]>maxx)maxx=f[i];
 cout<<maxx<<endl;
```

【部分和问题类动态规划】

求M个最大连续子序列和问题

【例2】求M个最大连续子序列和问题 --1524

Description

输入一个长度为 n 的整数序列（A1,A2,……,An），从中找出m个连续子序列，使得这m个连续子序列的和最大。

Input

第一行为用空格分开的两个整数n,m(n<=1000,m<=200)。第二行为n个用空格分开的整数序列,每个数的绝对值都小于1000。

Output

文件输出仅一个整数表示连续的长度不超过M的最大子序列和。

Sample Input

5 2

3 -2 1 2 3

Sample Output

9

【例2】求M个最大连续子序列和问题 --1524

【思路点拨】

该问题可以转换为了“资源分配类型动态规划问题”。设f[i][j]为以a[j]结尾的，分成i段子序列和的最大值，则需要枚举此段开头x和上一段结尾k，即f[i][j]=max{f[i-1][k]+a[x..j]}

每次需要枚举k<x<=j，决策量为O(n2)，状态为O(nm)，共$O(n^3m)$

注意到如果a[j-1]也是本段的，答案变成为f[i][j-1]+a[j]。设f[i][j]表示在数列前j个元素中，分成i个无公共元素的子序列的最大和，且必须包含第j个元素；因此方程优化为：f[i][j]=max{f[i][j-1]+a[j], f[i-1][k]+a[j]}, k<j，优化后状态仍然是二维的，但决策减少为O(n)，总$O(n^2m)$。

状态转移方程：f[i][j]=max{f[i][j-1]+a[j],f[i-1][k]+a[j] }

(2<=i<=m,i<=j<=n,i-1<=k<=j-1)

初始化：f[1][i]=max(f[1][i-1],0)+a[i]，(1<=i<=n)

时间复杂度：$O(n^3)$

【例2】求M个最大连续子序列和问题 --1524

【参考代码】

```
int i,j,k,n,m,best,a[1001],f[201][1001];
int main()
{  cin>>n>>m;
   for(i=1;i<=n;i++)cin>>a[i];
   for(i=1;i<=n;i++)f[1][i]=max(f[1][i-1],0)+a[i];//初始化
   for(i=2;i<=m;i++) //阶段
     for(j=i;j<=n;j++) //状态
     {  f[i][j]=f[i][j-1]+a[j];     //跟前j-1结尾的元素连续，直接累加在最后一个子序列上
        for(k=i-1;k<=j-1;k++) //不连续，则其 j 元素单独作为一个子序列
            if(f[i-1][k]+a[j]>f[i][j])f[i][j]=f[i-1][k]+a[j];
     }
   best=-0x7fffffff/2;
   for(i=1;i<=n;i++)if(f[m][i]>best)best=f[m][i];
   cout<<best<<endl;
}
```

【例2】求M个最大连续子序列和问题 --1524

【优化思考】

本题还可以继续优化，注意到时间主要耗费在对k的枚举上，计算max{f[i-1][k]}这个值…

我们把f的第一维称为“阶段”，则本题是典型的多阶段决策问题

- 计算一个阶段时，顺便记录本阶段最大值
- 只保留相邻两个阶段(滚动数组)

则时间降为O(nm)，空间降为O(n)。

【部分和问题类动态规划】

求指定大小的最大子矩形问题

【例3】轰炸 --1525

【问题描述】公元3030前，地球与某银河系星球COW为争夺资源开战，通过情报得知COW星球全部力量位于该星的最大平原Grass上，所有军事人员全部驻扎在分布在平原上的基地里，你的任务是去轰炸这些基地，消灭有生力量。

【输入文件】平原Grass呈方形，共有M行N列，基地分布在这些交叉点上，每个基地里有R个人（0<=R<=100）,你的炸弹威力也呈方形（有点怪!）大小为w行h列，范围内的所有人将被其消灭。

【输出文件】只有一行为一个炸弹可最多消灭多少人。

【样例输入】

2 2（表示m,n)

1 1（表示w,h)

2 0 (以下两行两列表示基地的分布)

1 0

【样例输出】2

【数据规模】对于80%数据,1<=m,n<=1000

对于100%数据,1<=m,n<=3000

【例3】轰炸 --1525

【参考代码】

```
int m,n,w,h,ans,a[3001][3001],f[3001][3001]={0},s[3001][3001]={0},i,j;
int main()
{  scanf("%d%d%d%d",&m,&n,&w,&h);
   for(i=1;i<=m;i++)//读入数据并初始化
      for(j=1;j<=n;j++)
      {  scanf("%d",&a[i][j]);
         s[i][j]=s[i-1][j]+s[i][j-1]-s[i-1][j-1]+a[i][j];
      }
   ans=-0x7fffffff/2;
   for(i=w;i<=m;i++)
      for(j=h;j<=n;j++)
      {  f[i][j]=s[i][j]-s[i-w][j]-s[i][j-h]+s[i-w][j-h];
         ans=max(ans,f[i][j]);
      }
   printf("%d",ans);
}
```

任意大小的最大子矩形问题

【例4】最大子矩阵问题 --1356

【问题描述】给定一个二维的数组（含正数或负数），请从中找出和最大的子矩阵。例如：

0	−2	−7	0
9	2	−6	2
−4	1	−4	1
−1	8	0	−2

9	2
−4	1
−1	8

【文件输入】输入部分第一行是一个正整数n,说明矩阵的大小。下一行后面跟着n*n的矩阵。矩阵中每个数字的范围为 [-1000, 1000]。

【文件输出】输出其中最大子矩阵的和。

【样例输入】

4

0 -2 -7 0

9 2 -6 2

-4 1 -4 1

-1 8 0 -2

【样例输出】15

【数据范围】n<=300

【例4】最大子矩阵问题 --1356

【题目分析】

找一个矩形，使得里面所有数的和尽量大。如果仍然用刚才介绍的方法，需要枚举左上角和右下角，时间复杂度为O(n2m2)。

容易观察到，最大子矩阵问题是最大连续子序列和问题的提升，即将一条线换成一个面，将一维问题提升到二维问题。所以我们计算最大子矩阵的方法就是将一行行的数进行累加以求得最大值。

但是还有一个问题，那就是应该如何高效地存储矩阵?

我们可以想到：在一个一维的数列中，设数组b[i]表示从第1个元素到第i个元素的和，则如果想要求第i个元素到第j个元素的和，只需要计算b[j]-b[i-1]的值就行了。由此推广到二维矩阵，设b[i][j]表示矩阵第j列前i个元素的和，a[i][j]表示元素数据，则压缩存储：

```
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++){cin>>a[i][j];b[i][j]=b[i-1][j]+a[i][j];}
```

因此，我们可以使用三重循环求出所有的矩形值，即枚举起始行i和终止行j，压缩子矩形成为一行，变成一维求最大字段和问题。

即t[k]=max(t[k-1],0)+b[j][k]-b[i-1][k];

时间复杂度为O(n3)

【例4】最大子矩阵问题 --1356


```
int sum,n,i,j,k,b[301][301],a[301][301],t[301];
int main()
{  cin>>n;
   for(i=1;i<=n;i++)
      for(j=1;j<=n;j++){cin>>a[i][j];b[i][j]=b[i-1][j]+a[i][j];}
   sum=-0x7fffffff;
   for(i=1;i<=n;i++)  //阶段:起始行
    {  for(j=i;j<=n;j++)  //状态:结束行
       {  t[1]=b[j][1]-b[i-1][1];  //初始化第1列的值
          for(k=2;k<=n;k++)  //决策:第几列
          {  t[k]=max(t[k-1],0)+b[j][k]-b[i-1][k];
             if(t[k]>sum)sum=t[k];
          }
       }
    }
   cout<<sum<<endl;
}
```

【思考】最大环形子矩阵

【**问题描述**】给n*n矩阵，第一行和最后一行看作是连通的；第一列和最后一列看作是连通的，求和最大的子矩阵。

1	-1	0	0	-4
2	3	-2	-3	2
4	1	-1	5	0
3	-2	1	-3	2
-3	2	4	1	-4

【部分和问题类动态规划】

求两个不相交子矩阵问题

【例5】最大矩形 --1526

【问题描述】一个N*M的矩阵，每个格子里面有个整数（绝对值不大与10）,每个子矩阵（至少包含一个元素）的价值就是它所包含的格子内的数的和。 现在求两个不相交的子矩阵(不包含相同的格子),使得他们的价值的乘积最大。

例如：N=3，M=4，矩阵如图所示：

2 3 4 5

1 3 2 4

4 3 2 1

最大子矩阵值乘积为288(左边两列的和为16，右边两列的和为18，结果为16*18=288)。

【文件输入】第一行有两个数字n, m(n,m<100)。以后的n行，每行有m个整数。

【文件输出】输出文件仅一个数，即两不相交子矩阵价值乘积的最大值。

【样例输入1】

1 7

-9 -9 8 8 1 7 -4

【样例输出1】

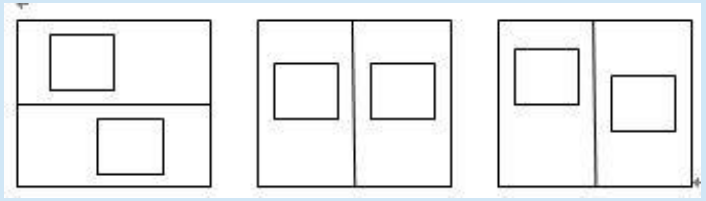
128

【例5】最大矩形 --1526

【题目分析】

在一个n*m二维的矩阵中，确定两个小的矩阵，使这两个小矩阵中所有元素的总和最大，且两个矩阵无公共元素。

既然是拓展，就与原问题有内在的联系，怎样的联系呢？我们先来看一下两个小矩阵的位置关系：



图中给出了三种，其实，第二种和第三种可以算为一种，所有的情况无非这两种而已。对于第一种情况，可以枚举分割线（图中横线）的坐标，设其坐标为x，x从2到n逐个枚举，则可求出1到x-1行的最优子矩阵、从x到n行的最优子矩阵，二者的和如果大于当前最优值则更新。对于第二种情况，只需将原始矩阵的行、列交换，然后重复第一种情况的步骤即可。

【例5】最大矩形 --1526

【参考代码】

```
#include<iostream>
#include<cstring>
using namespace std;
const int INF=0x7fffffff;
int n,m,ans=0,s[105][105]={0},a[105][105],fmax[105],fmin[105],t;
void Read()
{  int i,j;
   cin>>n>>m;
   for(i=1;i<=n;i++)
      for(j=1;j<=m;j++){cin>>a[i][j];s[i][j]=s[i-1][j]+a[i][j];}
}
```

【例5】最大矩形 --1526

【参考代码】

```
void Getans(int up,int down,int lef,int rig,int &mx,int &mi)
{  int i,j,k,maxn,minn;
   mx=-INF; mi=INF;
   fmax[lef-1]=0,fmin[lef-1]=0;    //每次只需初始化起点的前一个位置
   for(i=up;i<=down;i++)
      for(j=i;j<=down;j++)
      {
         for(k=lef;k<=rig;k++)
         {  t=s[j][k]-s[i-1][k];                    //利用前缀和压缩第i行(列)到第j行(列)
            fmax[k]=max(t,fmax[k-1]+t);  //转化为求最大连续子序列和问题
            fmin[k]=min(t,fmin[k-1]+t);
            if(fmax[k]>mx) mx=fmax[k];
            if(fmin[k]<mi) mi=fmin[k];
         }
      }
}
```

【例5】最大矩形 --1526

【参考代码】

```
void Dp()
{  int i,j,k,max1,max2,min1,min2;
   ans=-INF;
   for(k=2;k<=m;k++)          //枚举分割列
   {  Getans(1,n,1,k-1,max1,min1);
      Getans(1,n,k,m,max2,min2);
      if(max1*max2>ans) ans=max1*max2;
      if(min1*min2>ans) ans=min1*min2;
   }
   for(k=2;k<=n;k++)          //枚举分割行
   {  Getans(1,k-1,1,m,max1,min1);
      Getans(k,n,1,m,max2,min2);
      if(max1*max2>ans) ans=max1*max2;
      if(min1*min2>ans) ans=min1*min2;
   }
   cout<<ans;
}
```

【部分和问题类动态规划】

最大空正方形问题

【例6】盖房子 --1527

【问题描述】永恒の灵魂最近得到了面积为n*m的一大块土地(高兴ING^_^),他想在这块土地上建造一所房子，这个房子必须是正方形的。

但是，这块土地并非十全十美，上面有很多不平坦的地方（也可以叫瑕疵）。这些瑕疵十分恶心，以至于根本不能在上面盖一砖一瓦。

他希望找到一块最大的正方形无瑕疵土地来盖房子。

不过，这并不是什么难题，永恒の灵魂在10分钟内就轻松解决了这个问题。现在，您也来试试吧。

【文件输入】输入文件第一行为两个整数n,m(1<=n,m<=1000)，接下来n行，每行m个数字，用空格隔开。0表示该块土地有瑕疵，1表示该块土地完好。

【文件输出】一个整数，最大正方形的边长。

【样例输入】

```
4 4
0 1 1 1
1 1 1 0
0 1 1 0
1 1 0 1
```

【样例输出】2

【例6】盖房子 --1527

【方法一】f[i][j]：表示以(i,j)为右下角盖房子的最大边长；
f[i][j]=min{f[i-1][j-1],f[i-1][j],f[i][j-1]}+1；(a[i][j]=1)
answer=扫描所有点的最大值

```
#include<iostream>
using namespace std;
int a[1001][1001],f[1001][1001],ans,n,m,i,j;
int main()
{  cin>>n>>m;
   for(i=1;i<=n;i++)
      for(j=1;j<=m;j++)scanf("%d",&a[i][j]);
   ans=0;
   for(i=1;i<=n;i++)
     for(j=1;j<=m;j++)
        if(a[i][j]==1)
        {  f[i][j]=min(f[i-1][j],min(f[i-1][j-1],f[i][j-1]))+1;
           if(ans<f[i][j])ans=f[i][j];
        }
  cout<<ans<<endl;
}
```

【例6】盖房子 --1527

【方法二】

f[i][j]：表示以(i,j)为右下角盖房子的最大边长；

lf[i][j]：表示从(i,j)这点向左最长为1的区间；

up[i][j]：表示从(i,j)这点向上最长为1的区间；

f[i][j]=min{f[i-1][j-1]+1, lf[i][j],up[i][j]}；（a[i][j]=1）

answer=扫描所有点的最大值

【例6】盖房子 --1527


```
int i,j,m,n,ans,f[1001][1001]={0};
int lf[1001][1001]={0},up[1001][1001]={0},map[1001][1001]={0};
int main()
{  cin>>n>>m;
   for(i=1;i<=n;i++)
      for(j=1;j<=m;j++) scanf("%d",&map[i][j]);
   for(i=1;i<=n;i++)lf[i][1]=map[i][1];
   for(i=1;i<=n;i++)
      for(j=2;j<=m;j++)
         if(map[i][j]==0)lf[i][j]=0;else lf[i][j]=lf[i][j-1]+1;
   for(i=1;i<=m;i++)up[1][i]=map[1][i];
   for(i=2;i<=n;i++)
      for(j=1;j<=m;j++)
         if(map[i][j]==0)up[i][j]=0;else up[i][j]=up[i-1][j]+1;
   ans=0;
   for(i=1;i<=n;i++)
      for(j=1;j<=m;j++)
      {  f[i][j]=min(f[i-1][j-1]+1,min(lf[i][j],up[i][j]));
         if(f[i][j]>ans)ans=f[i][j];
      }
   cout<<ans<<endl;
}
```

最大空矩形问题

【例7】月饼盒 --1528

【题目背景】中秋节了，CCC老师决定去送礼。

【问题描述】一个被分为 n*m　个格子的月饼盒，第 i 行第 j 列位置的格子里面有 a [i , j]个月饼。本来CCC老师打算送这盒月饼给某人的，但是就在要送出月饼盒的前一天晚上，一只极其可恶的老鼠夜袭月饼盒，有部分格子被洗劫并且穿了洞。CCC老师必须尽快从这个月饼盒里面切割出一个矩形月饼盒，新的月饼盒不能有洞，并且CCC老师希望保留在新月饼盒内的月饼的总数尽量多。

【题目任务】请帮CCC老师设计一个程序 计算一下新月饼盒最多能够保留多少月饼。

【文件输入】第一行有两个整数 n、m。第i+1行的第j个数表示a [i,j]，如果这个数为 0 ，则表示这个位置的格子被洗劫过。其中0≤ a[i,j]≤1000

【文件输出】输出最大月饼数。

【样例输入】

3 4
1 2 3 4
5 0 6 3
10 3 4 0

【样例输出】17

【例7】月饼盒 --1528

【思路点拨】

首先需要注意的是：本题的模型是一个数字矩阵而不是几何矩形。在数字矩阵的情况下，由于点的个数是有限的，所以又产生了一个新的问题：**最大权值子矩阵**。

定义：内部不包含任何障碍点的子矩形为有效子矩阵。与有效子矩形不同的是，有效子矩阵的边界上也不能包含障碍点。有效子矩阵的权值(只有有效子矩形才有权值)为这个子矩阵包含的所有点的权值和。最大权值有效子矩阵为所有有效子矩阵中权值最大的一个。以下简称为最大权值子矩阵。

本题的数学模型就是正权值条件下的最大权值子矩阵问题。再一次利用极大化思想，因为矩阵中的权值都是正的，所以**最大权值子矩阵一定是一个极大子矩阵**。所以我们只需要枚举所有的极大子矩阵，就能从中找到最大权值子矩阵。同样，枚举法和递推法只需稍加修改就可以解决本题。下面分析两种算法应用在本题上的优劣。

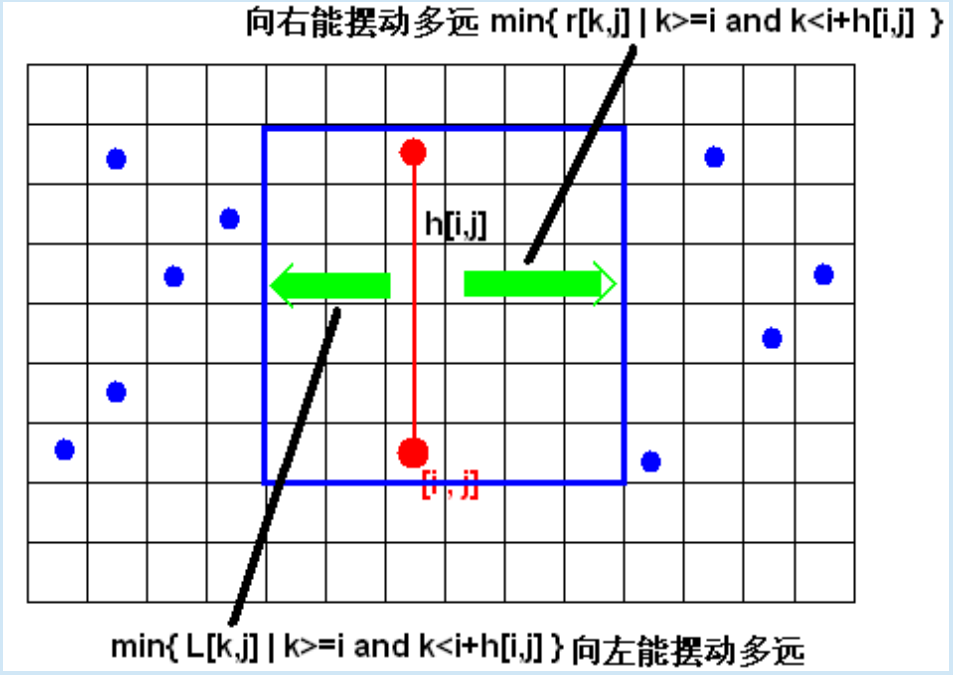
对于枚举法，由于矩形中障碍点的个数是不确定的，而且最大有可能达到N×M,这样时间复杂度有可能达到$O(N^2M^2)$，空间复杂度为O(NM)。此外，由于矩形与矩阵的不同，所以在处理上会有一些小麻烦。

对于递推法，稍加变换就可以直接使用，时间复杂度为O(NM),空间复杂度为O(NM) 。显然枚举法并不适合这道题，最好还是采用递推法。

【例7】月饼盒 --1528

【核心思想】

对于(i,j)这个格子，我们如果将以它为某矩形底边的一个格子，它向上可以达到高为h[i][j]，我们把这个看成一条高为h[i][j]的一条线段，让这条线段向左右尽可能长地摆动，假设他能够摆动的长度分别为L[i][j]和R[i][j]，这样能够得到一个以(i,j)为底边一点的最大矩阵（即可以确定矩形的左上脚坐标和右下脚坐标），这样我们用递推就可以一次性算出这个矩阵的权值。如下图。



h[i,j]、L[i,j]、R[i,j]都可以递推得到：

当格子中的数为0的时候h[i][j]=0，否则h[i][j]=h[i-1][j]+1；

当格子中的数为0的时候L[i][j]=0，否则L[i][j]=L[i][j-1]+1；

当格子中的数为0的时候R[i][j]=0，否则R[i][j]=R[i][j+1]+1；

【例7】月饼盒 --1528

【核心思想】

```
int a[301][301],L[301][301],R[301][301],h[301][301],n,m,ans;
void init()
{  int i,j,c;
   cin>>n>>m;
   for(i=1;i<=n;i++)
   {  a[i][0]=0; h[i][0]=0;
      for(j=1;j<=m;j++)
      {  scanf("%d",&c);
         a[i][j]=c+a[i-1][j]+a[i][j-1]-a[i-1][j-1];
         if(c==0)h[i][j]=0;else h[i][j]=h[i-1][j]+1;//计算h[i][j]
      }
      L[i][0]=0;
      for(j=1;j<=m;j++)//计算 L
           if(h[i][j]==0)L[i][j]=0;else L[i][j]=L[i][j-1]+1;
      R[i][m+1]=0;
      for(j=m;j>=1;j--)//计算r
           if(h[i][j]==0)R[i][j]=0;else R[i][j]=R[i][j+1]+1;
   }
}
```

【例7】月饼盒 --1528

【核心思想】

```
void dp()
{  int x1,y1,x2,y2,maxx,i,j; ans=0;
   for(i=1;i<=n;i++)
     for(j=1;j<=m;j++)
       if(h[i][j]>0)
       {  if(h[i][j]>1)
          {  if(L[i][j]>L[i-1][j])L[i][j]=L[i-1][j];//得到最小的L
             if(R[i][j]>R[i-1][j])R[i][j]=R[i-1][j];//得到最小的r
          }
          x2=i;y2=j+R[i][j]-1;        //右下角
          x1=i-h[i][j]+1;y1=j-L[i][j]+1; //左上角
          maxx=a[x2][y2]-a[x2][y1-1]-a[x1-1][y2]+a[x1-1][y1-1];
          if(ans<maxx)ans=maxx;
       }
}
```