bzoj 2510: 弱题 循环矩阵 - mhy12345

Description

有M个球，一开始每个球均有一个初始标号，标号范围为1～N且为整数，标号为i的球有a_i个，并保证$\Sigma a_i = M$。每次操作等概率取出一个球（即取出每个球的概率均为1/M），若这个球标号为k（k < N），则将它重新标号为k + 1；若这个球标号为N，则将其重标号为1。（取出球后并不将其丢弃）现在你需要求出，经过K次这样的操作后，每个标号的球的期望个数。

Input

第1行包含三个正整数N，M，K，表示了标号与球的个数以及操作次数。第2行包含N个非负整数a_i，表示初始标号为i的球有a_i个。

Output

应包含N行，第i行为标号为i的球的期望个数，四舍五入保留3位小数。

Sample Input

```
2 3 2
3 0
```

Sample Output

```
1.667
1.333
```

HINT

【样例说明】

第1次操作后，由于标号为2球个数为0，所以必然是一个标号为1的球变为标号为2的球。所以有2个标号为1的球，有1个标号为2的球。

第2次操作后，有1/3的概率标号为2的球变为标号为1的球（此时标号为1的球有3个），有2/3的概率标号为1的球变为标号为2的球（此时标号为1的球 有1个），所以标号为1的球的期望个数为1/3*3+2/3*1 = 5/3。同理可求出标号为2的球期望个数为4/3。

【数据规模与约定】

对于10%的数据，N ≤ 5, M ≤ 5, K ≤ 10；

对于20%的数据，N ≤ 20, M ≤ 50, K ≤ 20；

对于30%的数据，N ≤ 100, M ≤ 100, K ≤ 100；

对于40%的数据，M ≤ 1000, K ≤ 1000；

对于100%的数据，N ≤ 1000, M ≤ 100,000,000, K ≤ 2,147,483,647。

这道题有两种解法，一中是预处理每一个位置经过k次到达另外位置的概率，七中运用到了类似于倍增的方法，求经过2^i次转移后的概率数组，然后在计算k次。

另一种解法在网上已经有提到，观察转移矩阵是一个“循环矩阵”，即每一行都是上一行通过右移得到，循环矩阵A、B满足A*B=C，那么C也是循环矩阵，则这样的矩阵做乘法只用O(n^2)，原因是一个矩阵只需要O(n)的空间就可以储存，而答案矩阵每一个元素都能用乘数矩阵通过O(n^2)错位相乘得出，具体细节自行脑补。总之我觉得非常神奇。

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>
using namespace std;
#define MAXN 1099
typedef double real;
int n,m,tt;
real mat[MAXN];
real res[MAXN];
real tmp[MAXN];
real ans[MAXN];
int tot[MAXN];
void mul(real res[],real m1[],real m2[])
{
        memset(res,0,sizeof(real)*MAXN);
        for (int i=0;i<m;i++)
                for (int j=0;j<m;j++)
```

```
                        res[(i+j+m)%m]+=m1[i]*m2[j];
}


int main()
{
        //freopen("input.txt","r",stdin);
        scanf("%d%d%d",&m,&n,&tt);
        for (int i=0;i<m;i++)
                scanf("%d",tot+i);
        mat[1]=1.0/n;
        mat[0]=(n-1.0)/n;
        res[0]=1;
        while (tt)
        {
                if (tt&1)
                {
                        mul(tmp,res,mat);
                        memcpy(res,tmp,sizeof(tmp));
                }
                mul(tmp,mat,mat);
                memcpy(mat,tmp,sizeof(tmp));
                tt>>=1;
        }
        for (int i=0;i<m;i++)
        {
                for (int j=0;j<m;j++)
                {
                        ans[i]+=tot[j]*res[(i-j+m)%m];
                }
                printf("%.3lf\n",ans[i]);
        }
        return 0;
}
```


Description

打地鼠是这样的一个游戏：地面上有一些地鼠洞，地鼠们会不时从洞里探出头来很短时间后又缩回洞中。玩家的目标是在地鼠伸出头时，用锤子砸其头部，砸到的地鼠越多分数也就越高。游戏中的锤子每次只能打一只地鼠，如果多只地鼠同时探出头，玩家只能通过多次挥舞锤子的方式打掉所有的地鼠。你

认为这锤子太没用了，所以你改装了锤子，增加了锤子与地面的接触面积，使其每次可以击打一片区域。如果我们把地面看做M*N的方阵，其每个元素都代表一个地鼠洞，那么锤子可以覆盖R*C区域内的所有地鼠洞。但是改装后的锤子有一个缺点：每次挥舞锤子时，对于这R*C的区域中的所有地洞，锤子会打掉恰好一只地鼠。也就是说锤子覆盖的区域中，每个地洞必须至少有1只地鼠，且如果某个地洞中地鼠的个数大于1，那么这个地洞只会有1只地鼠被打掉，因此每次挥舞锤子时，恰好有R*C只地鼠被打掉。由于锤子的内部结构过于精密，因此在游戏过程中你不能旋转锤子（即不能互换R和C）。你可以任意更改锤子的规格（即你可以任意规定R和C的大小），但是改装锤子的工作只能在打地鼠前进行（即你不可以打掉一部分地鼠后，再改变锤子的规格）。你的任务是求出要想打掉所有的地鼠，至少需要挥舞锤子的次数。Hint：由于你可以把锤子的大小设置为1*1，因此本题总是有解的。

Input

第一行包含两个正整数M和N；

下面M行每行N个正整数描述地图，每个数字表示相应位置的地洞中地鼠的数量。

Output

输出一个整数，表示最少的挥舞次数。

Sample Input

3 3

1 2 1

2 4 2

1 2 1

Sample Output

4

【样例说明】

使用2*2的锤子，分别在左上、左下、右上、右下挥舞一次。

【数据规模和约定】

对于100%的数据，1<=M,N<=100，其他数据不小于0，不大于10^5

HINT

反过来考虑，如果我们知道了击打位置，能否快速反推棋盘，想到将一次击打(x,y)(x+r,y+x)加1，(x+r,y)(x,y+c)减1用二维前缀和搞，所以我们现在已经有前缀和，还原原来的矩阵，然后就可以暴力枚举r,c然后暴力判断，时间复杂度常数极小的O(n^4)

样例大坑，输出最大面积，最小次数均能过样例 TAT

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
#define MAXN 103
#define update(i,j) if (vis[i][j]!=vistime)tnow[i][j]=tot[i][j],vis[i][j]=vistime;
int tot[MAXN][MAXN];
int vis[MAXN][MAXN],vistime=0;
int tnow[MAXN][MAXN];

int main()
{
        freopen("input.txt","r",stdin);
        int n,m;
        scanf("%d%d",&n,&m);
        int sum=0;
        for (int i=1;i<=n;i++)
                for (int j=1;j<=m;j++)
                        scanf("%d",&tot[i][j]),sum+=tot[i][j];
        for (int i=n+1;i>=1;i--)
                for (int j=m+1;j>=1;j--)
                        tot[i][j]-=tot[i-1][j]+tot[i][j-1]-tot[i-1][j-1];
        bool flag;
        int ans=0;
        for (int r=1;r<=n;r++)
        {
                for (int c=1;c<=m;c++)
                {
                        ++vistime;
                        flag=true;
                        if (r*c<=ans)continue;
```

```
int i,j;
for (i=1;i+r-1<=n;i++)
{
        for (j=1;j+c-1<=m;j++)
        {
                update(i,j);
                update(i+r,j);
                update(i,j+c);
                update(i+r,j+c);
                if (tnow[i][j]<0)
                {
                        flag=false;
                        break;
                }
                tnow[i+r][j]+=tnow[i][j];
                tnow[i][j+c]+=tnow[i][j];
                tnow[i+r][j+c]-=tnow[i][j];
                tnow[i][j]=0;
        }
        if (!flag)break;
        for (;j<=m;j++)
        {
                update(i,j);
                if (tnow[i][j])
                {
                        flag=false;
                        break;
                }
        }
        if (!flag)break;
}
for (;i<=n;i++)
{
        for (j=1;j<=m;j++)
        {
                update(i,j);
                if (tnow[i][j])
                {
                        flag=false;
                        break;
                }
        }
        if (!flag)break;
}
```

```
                    if (flag)
                    {
                            ans=r*c;
                    }
            }
    }
    printf("%d\n",sum/ans);
}
```