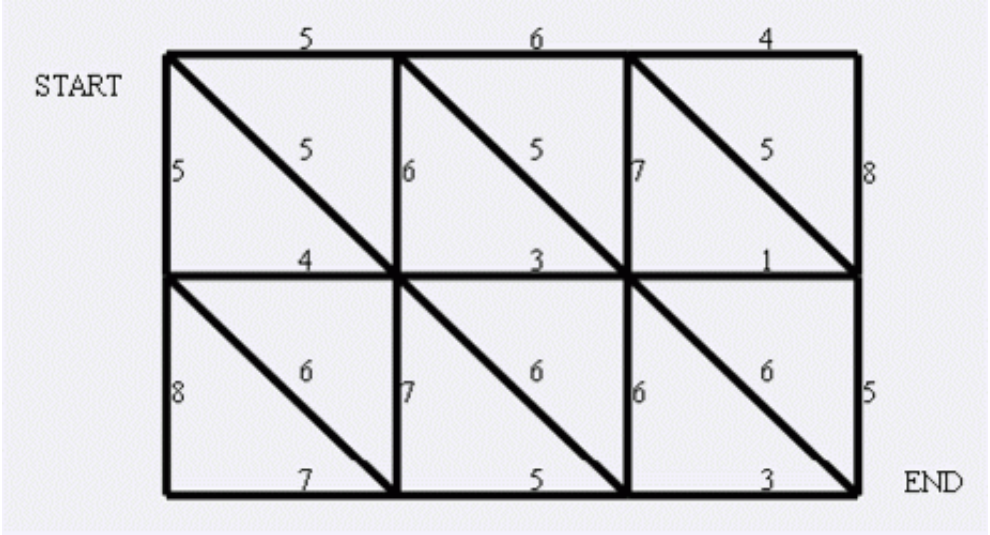
题意

面对下面这样一个网格的地形：



左上角点为(1,1),右下角点为(N,M)(上图中N=4,M=5).有以下三种类型的道路
1:(x,y)<==>(x+1,y)
2:(x,y)<==>(x,y+1)
3:(x,y)<==>(x+1,y+1)
道路上的权值表示这条路上最多能够通过的兔子数，道路是无向的. 左上角和右下角为兔子的两个窝，
开始时所有的兔子都聚集在左上角(1,1)的窝里，现在它们要跑到右下解(N,M)的窝中去，狼王开始伏击
这些兔子.当然为了保险起见，如果一条道路上最多通过的兔子数为K，狼王需要安排同样数量的K只狼，
才能完全封锁这条道路，你需要帮助狼王安排一个伏击方案，使得在将兔子一网打尽的前提下，参与的
狼的数量要最小。

思路

要求最少的狼截住所有兔子，即为图的最小割
转化为求最大流
将网格图所有格点标序号（0 ~ n*m-1），第x行第y列点为（x - 1) * m + y - 1;

前向星邻接表建图
跑DINIC最大流即可

代码

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
#define typec int
const int N = 1000005;
const int E = 6000005;
const int inf = 1000000000;
struct edge { int x, y, nxt; typec c; } bf[E];
int ne, head[N], cur[N], ps[N], dep[N];
void addedge(int x, int y, typec c)
{ // add an arc(x -> y, c); vertex: 0 ~ n-1;
    bf[ne].x = x; bf[ne].y = y; bf[ne].c = c;
    bf[ne].nxt = head[x]; head[x] = ne++;
    bf[ne].x = y; bf[ne].y = x; bf[ne].c = c;
    bf[ne].nxt = head[y]; head[y] = ne++;
}
typec flow(int n, int s, int t)
{
    typec tr, res = 0;
    int i, j, k, f, r, top;
    while (1)
    {
        memset(dep, -1, n * sizeof(int));
        for (f = dep[ps[0] = s] = 0, r = 1; f != r; )
        for (i = ps[f++], j = head[i]; j; j = bf[j].nxt)
        {
            if (bf[j].c && -1 == dep[k = bf[j].y])
            {
                dep[k] = dep[i] + 1; ps[r++] = k;
                if (k == t) { f = r; break; }
            }
        }
        if (-1 == dep[t]) break;
        memcpy(cur, head, n * sizeof(int));
        for (i = s, top = 0; ; )
        {
```

```
            if (i == t)
             {
                for (k = 0, tr = inf; k < top; ++k)
                    if (bf[ps[k]].c < tr)
                        tr = bf[ps[f = k]].c;
                for (k = 0; k < top; ++k)
                    bf[ps[k]].c -= tr, bf[ps[k]^1].c += tr;
                res += tr; i = bf[ps[top = f]].x;
            }
            for (j=cur[i]; cur[i]; j = cur[i] = bf[cur[i]].nxt)
                if (bf[j].c && dep[i]+1 == dep[bf[j].y]) break;
            if (cur[i])
            {
                ps[top++] = cur[i];
                i = bf[cur[i]].y;
            }
            else
            {
                if (0 == top) break;
                dep[i] = -1; i = bf[ps[--top]].x;
            }
        }
    }
    return res;
}
int main()
{
    ios::sync_with_stdio(false);
    int n,m;
    cin>>n>>m;
    memset(head,0,sizeof(head));
    ne = 2;
    int x;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<m;j++)
        {
            cin>>x;
            int u = (i-1)*m+j-1;
            int v = u+1;
            addedge(u,v,x);
        }
    }
    for(int i=1;i<n;i++)
    {
        for(int j=1;j<=m;j++)
```

```
83          {
84              cin>>x;
85              int u = (i-1)*m+j-1;
86              int v = u+m;
87              addedge(u,v,x);
88          }
89      }
90      for(int i=1;i<n;i++)
91      {
92          for(int j=1;j<m;j++)
93          {
94              cin>>x;
95              int u = (i-1)*m+j-1;
96              int v = u+m+1;
97              addedge(u,v,x);
98          }
99      }
100     int ans = flow(n*m,0,n*m-1);
101     cout<<ans<<endl;
102 }
```