# [网络流进阶之转换对偶图](https://www.cnblogs.com/jinkun113/p/9495308.html)

**零、目录**

　　I、[网络流基础](https://www.cnblogs.com/jinkun113/p/9427825.html" \t "https://www.cnblogs.com/jinkun113/p/_blank)

**II、网络流进阶之转换对偶图**

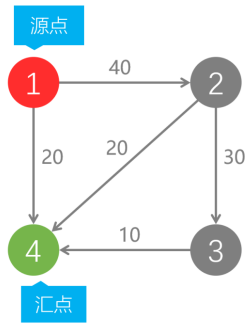
　　III、网络流进阶之费用流

**一、前言**

本文为上一篇文章《网络流基础》之续集，同样3年前已有一篇文章讲解转换对偶图，这里再次为其翻新一次，希望能够更好理解。

**二、最小割**

讲网络流不得不提一个概念——**最小割**。便于理解，上一篇文章并没有将其搅和进来。最小割是什么呢？**现在要求割断部分路径上的流量，使从源点没有任何流量可以到达汇点，而截取的流量最小值即最小割**。我们再次拿出上次的模型：



首先从1至4最直接的20流量必然需要截掉；从1至2理应截取40，但由于2-3-4路径上的最大流仅为10，加上2-4流量为20，故只需截取30；总计50流量。

看着看着就觉得有意思了——对于任意一条路径，其能够流通的流量最大值便是我们需要割掉的流量最小值，即**最大流=最小割。**

这里提及最小割的概念，能够更好的理解接下来的内容——转换对偶图。先看一道例题。

**三、例题**

# 狼抓兔子

**【题目描述】**

**话说灰太狼抓羊不到，但抓兔子还是比较在行的，**

**而且现在的兔子还比较笨，它们只有两个窝，现在你做为狼王，面对下面这样一个网格的地形：**

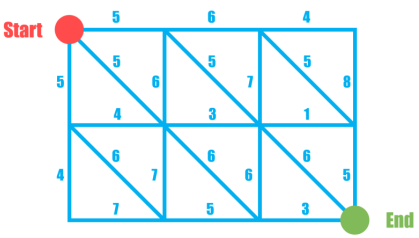
**左上角点为(1,1),右下角点为(N,M)(上图中N=4,M=5).有以下三种类型的道路：**

**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。**

**【输入描述】**

**第一行为N、M，表示网格的大小，N、M均小于等于1000；**

**接下来分三部分：**

**第一部分共N行，每行M-1个数，表示横向道路的权值；**

**第二部分共N-1行，每行M个数，表示纵向道路的权值；**

**第三部分共N-1行，每行M-1个数，表示斜向道路的权值。**

**【输出描述】**

**输出一个整数，表示参与伏击的狼的最小数量。**

**【输入样例】**

**3 4**

**5 6 4**

**4 3 1**

**7 5 3**

**5 6 7 8**

**8 7 6 5**

**5 5 5**

**6 6 6**

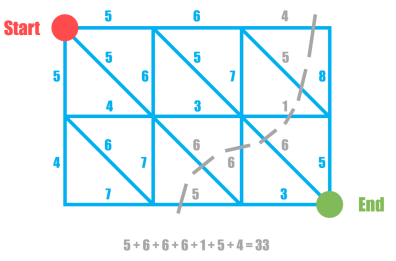
**【输出样例】**

**14**

首先，裸网络流的正确性毋庸置疑，此处不再赘述，因为数据并没有允许这种无脑的方式通过，所以我们现在带着脑子想一个更好的办法。

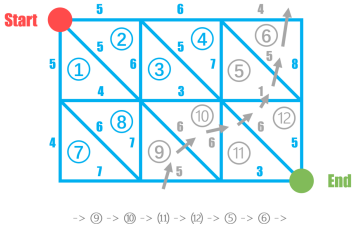
**四、转换对偶图**

每一次狼的派遣其本质其割流。本题虽不是传统网格图，但同样可以看作一种特殊的三角网格图。目的同样是求最小割，我们来先随手割一刀看是什么效果——

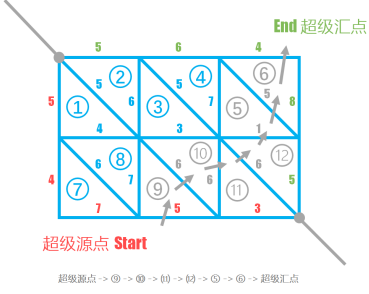


如图为一种非最小割方案，需要割掉33流量。我们发现，当且仅当所有割线将图划分成两部分时，源点没有流量流向汇点。了解这一项规律后，满足最小割的方案可以轻易得到——将图中直接与汇点相连的三条边割掉，可以最小割为3 + 5 + 6 = 14，同样将图割成两部分。

现在，我们能否将这张图进行一定改进——对于每一次割，可以理解为走过一条连通这两边两侧区域的边，其权值即流量，而我们求的最小割即最小权值。所以现在我们将原图转换一下——将每一块区域看作一点，两个区域之间的边看作两点之间相连的边，称之为**对偶图**。



 看起来大功告成，不过我们忽视了两个更大的区域。所有边界似乎无法处理？这时我们将所有左下方的边界与一个点视作相连，此点称之为超级源点；同理将右上方区域视作超级汇点。



这样我们每一次求最小割，其实本质是在新的对偶图上跑最短路，起点为超级源点，终点为超级汇点。综上，上述方案即为：

IMG_261

**五、代码**

要注意的是，N/M <=1000的条件下，其对偶图最多会存在6\*10^6个节点，Dijkstra算法时间复杂度为O(n^2)，显然无法通过；起初我尝试了SPFA算法，但由于其复杂度的不稳定，第10个点TLE了；最终修改成了**优先队列优化的Dijkstra算法**，其复杂度为O(nlogn)**。**

#include <cstdio>

#include <cstring>

#include <vector>

#include <queue>

using namespace std;

#define MAXN 6000005#define INF 0x3f3f3f3f

int n, m, t, h[MAXN], dis[MAXN], w, o;

struct Edge {

int v, next, w;

} e[MAXN];

struct Node {

int n, w;

};

struct cmp {

bool operator () (Node a, Node b) {

return a.w > b.w;

}

};

priority\_queue <Node, vector<Node>, cmp> Q;

void add(int u, int v, int w) {

o++, e[o] = (Edge) {v, h[u], w}, h[u] = o;

o++, e[o] = (Edge) {u, h[v], w}, h[v] = o;

}

void init() {

scanf("%d %d", &n, &m), t = (n - 1) \* (m - 1) \* 2 + 1;

memset(h, -1, sizeof(h)), memset(dis, INF, sizeof(dis));

for (int i = 1, tot = 2; i <= n; i++)

for (int j = 1; j <= m - 1; j++, tot += 2) {

int u = i == 1 ? t : tot - m \* 2 + 1, v = i == n ? 0 : tot;

scanf("%d", &w), add(u, v, w);

}

for (int i = 1, tot = 1; i <= n - 1; i++) {

for (int j = 1; j <= m - 1; j++, tot += 2) {

int u = j == 1 ? 0 : tot - 1;

scanf("%d", &w), add(u, tot, w);

}

scanf("%d", &w), add(tot - 1, t, w);

}

for (int i = 1, tot = 1; i <= n - 1; i++)

for (int j = 1; j <= m - 1; j++, tot += 2) scanf("%d", &w), add(tot, tot + 1, w);

}

void work() {

Q.push((Node) {0, 0}), dis[0] = 0;

while (!Q.empty()) {

Node o = Q.top();

for (int x = h[o.n]; x != -1; x = e[x].next) {

int v = e[x].v;

if (dis[v] > dis[o.n] + e[x].w) dis[v] = dis[o.n] + e[x].w, Q.push((Node) {v, dis[v]});

}

Q.pop();

}

}

int main() {

init();

work();

printf("%d", dis[t]);

return 0;

}