# 迭代加深搜索

**描述：**

照明是一个拼图集在一个矩形板上，分成较小的方格。黑板上的一些方块是“空的”(下面的数字是白色方块)，有些方块是“障碍物”(下图中的暗方块)。一个障碍平方可能有一个与它相关的整数i(0<=i<=4)。

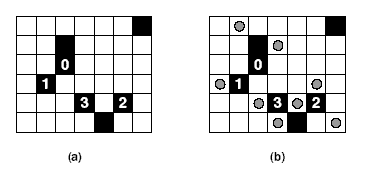


图2：(A)6行、7列和7栏的困惑；(B)谜题的解决方案。

在这个谜题中，目标是“点亮”所有的空方块，在其中一些地方放置灯(灯在图中被描绘成圆圈)。每盏灯照亮它所开的正方形，再加上所有与之平行的方格，水平或垂直，直至栅栏方或板端。

获胜的配置满足以下条件：

所有空旷的广场都必须点亮；

任何灯不得由另一盏灯点亮；

所有编号的阻隔方块必须有与其相邻的灯数(在上面、下面和侧面的四个方格中)；

没有编号的阻隔方块可以有任意数量的灯与他们相邻。

**输入：**

输入包含几个测试用例。测试用例的第一行包含两个整数N、M，分别表示板的行数和列数(1<=N<=7，1<=M<=7)。第二行包含一个整数B，表示势垒平方数(0<=B<=N×M)。下一个B行中的每一行描述一个屏障，包含三个整数R、C和K，分别表示行数(1<=R<=N)、列数(1<=C<=M)和屏障数(-1<=K<=4)；K=-1表示没有编号。输入的结尾用N=M=0表示。

**输出量：**

对于输入中的每个测试用例，您的程序必须生成一行输出，其中包含一个整数，该整数指示达到一个成功配置所需的最小灯具数，如果存在这样的配置，或者使用“无解决方案”字样。

#include<cstdio>

#include<cstring>

#include<cctype>

#include<algorithm>

using namespace std;

const int N = 9;

bool ok;

int flag[N][N];

int cover[N][N];

int dx[] = {1,-1,0,0};

int dy[] = {0,0,1,-1};

int n,m,b,ans;

struct node

{

int x,y,val;

}barrier[50];

int cmp(struct node a,struct node b)

{

return a.val < b.val;

}

int nextint()

{

int ret;

char c;

bool sig = false;

while(isspace(c = getchar()))

;

if(c == '-')

{

sig = true;

c = getchar();

}

ret = c - '0';

while((c = getchar()) >= '0' && c <= '9')

ret = ret \* 10 + c - '0';

return sig?-ret:ret;

}

bool jud(int x,int y)

{

return !(x >= 1 && y >= 1 && x <= n && y <= m);

}

int h()

{

int i,j;

int ret = 0;

for(i = 1;i <= n;i ++)

{

for(j = 1;j <= m;j ++)

{

if(!cover[i][j] && flag[i][j] != 2)

{

if((flag[i - 1][j] == 2 || jud(i - 1,j))

&& (flag[i + 1][j] == 2 || jud(i + 1,j))

&& (flag[i][j - 1] == 2 || jud(i,j - 1))

&& (flag[i][j + 1] == 2 || jud(i,j + 1)))

ret ++;

}

}

}

return ret;

}

int put(int x,int y)

{

int ret = 1;

int i,j;

i = x - 1;

j = y;

while(i > 0 && flag[i][j] != 2)

{

cover[i][j] ++;

if(cover[i][j] == 1)

ret ++;

i --;

}

i = x + 1;

j = y;

while(i <= n && flag[i][j] != 2)

{

cover[i][j] ++;

if(cover[i][j] == 1)

ret ++;

i ++;

}

i = x;

j = y - 1;

while(j > 0 && flag[i][j] != 2)

{

cover[i][j] ++;

if(cover[i][j] == 1)

ret ++;

j --;

}

j = y + 1;

while(j <= m && flag[i][j] != 2)

{

cover[i][j] ++;

if(cover[i][j] == 1)

ret ++;

j ++;

}

return ret;

}

void deput(int x,int y)

{

int ret = 1;

int i,j;

i = x - 1;

j = y;

while(i > 0 && flag[i][j] != 2)

{

cover[i][j] --;

i --;

}

i = x + 1;

while(i <= n && flag[i][j] != 2)

{

cover[i][j] --;

i ++;

}

i = x;

j = y - 1;

while(j > 0 && flag[i][j] != 2)

{

cover[i][j] --;

j --;

}

j = y + 1;

while(j <= m && flag[i][j] != 2)

{

cover[i][j] --;

j ++;

}

}

int check()//0 irllegal 1 ok 2 right

{

int i,j;

int tmp;

int ret = 2;

for(i = 0;i < b;i ++)

{

if(barrier[i].val == -1)

continue;

tmp = 0;

for(j = 0;j < 4;j ++)

{

int tx = barrier[i].x + dx[j];

int ty = barrier[i].y + dy[j];

if(tx < 1 || ty < 1 || tx > n || ty > m)

continue;

if(flag[tx][ty] == 3)

tmp ++;

}

if(tmp > barrier[i].val)

return 0;

if(tmp != barrier[i].val)

ret = 1;

}

return ret;

}

void print()

{

int i,j;

for(i = 1;i <= n;i ++)

{

for(j = 1;j <= m;j ++)

printf("%d ",flag[i][j]);

putchar(10);

}

putchar(10);

for(i = 1;i <= n;i ++)

{

for(j = 1;j <= m;j ++)

printf("%d ",cover[i][j]);

putchar(10);

}

}

void Dfs(int x,int y,int cnt,int ca)

{

if(ca > ans)

return;

int i,j;

if(ok)

return;

if(cnt == m \* n - b)

{

ok = true;

return;

}

for(i = x;i <= n;i ++)

{

for(j = 1;j <= m;j ++)

{

if(flag[i][j] == 0 && cover[i][j] == 0)

{

flag[i][j] = 3;

if(check() != 2)

{

flag[i][j] = 0;

continue;

}

int tp = put(i,j);

Dfs(i,j,cnt + tp,ca + 1);

deput(i,j);

flag[i][j] = 0;

}

}

}

}

void dfs(int id,int cnt,int ca)

{

if(ca > ans)

return;

if(ok)

return;

if(id == b)

{

if(check() == 2)//right

{

Dfs(1,1,cnt,ca);

}

return;

}

int i,j;

for(i = j = 0;i < 4;i ++)

{

if(flag[barrier[id].x + dx[i]][barrier[id].y + dy[i]] == 3)

j ++;

}

if(j > barrier[id].val)

return;

else

{

if(j == barrier[id].val)

dfs(id + 1,cnt,ca);

else

{

for(i = 0;i < 4;i ++)

{

int tx = barrier[id].x + dx[i];

int ty = barrier[id].y + dy[i];

if(tx < 1 || ty < 1 || tx > n || ty > m)

continue;

if(flag[tx][ty] == 0 && cover[tx][ty] == 0)

{

flag[tx][ty] = 3;

if(check() == 0)

{

flag[tx][ty] = 0;

continue;

}

int tp = put(tx,ty);

dfs(id,cnt + tp,ca + 1);

deput(tx,ty);

flag[tx][ty] = 0;

}

}

}

}

}

int main()

{

int i,j;

while(n = nextint())

{

m = nextint();

if(m + n == 0)

break;

b = nextint();

memset(flag,0,sizeof(flag));

memset(cover,0,sizeof(cover));

for(i = 0;i < b;i ++)

{

barrier[i].x = nextint();

barrier[i].y = nextint();

barrier[i].val = nextint();

flag[barrier[i].x][barrier[i].y] = 2;//

}

sort(barrier,barrier + b,cmp);

ok = false;

for(i = 0;i < b;i ++)

if(barrier[i].val > 0)

break;

ans = h();

//print();

while(1)

{

if(ok || ans > m \* n - b)

break;

dfs(i,0,0);

if(ok)

break;

ans ++;

}

if(ok == false)

printf("No solution\n");

else

printf("%d\n",ans);

}

return 0;

}

