**★★★★★**

**題組：Contest Volumes (10000...)**

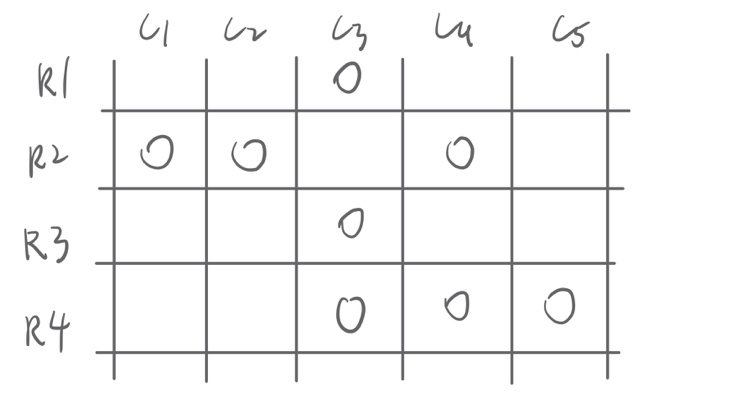
**題號：11419 SAM I AM**

**解題者：周霖**

**解題日期：2025年5月28日**

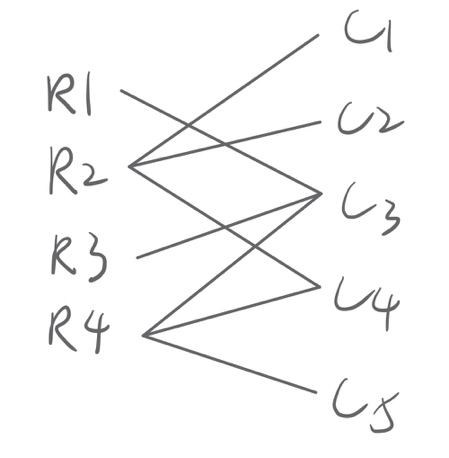
**題意：**

這題的題意是給訂R\*C的網格，給定N個敵人的座標(row, col)。一次發射炮彈可以打一整row或一整col，求最少花幾顆砲彈能解決所有敵人、並求該炮彈要發射在何rol或何col。

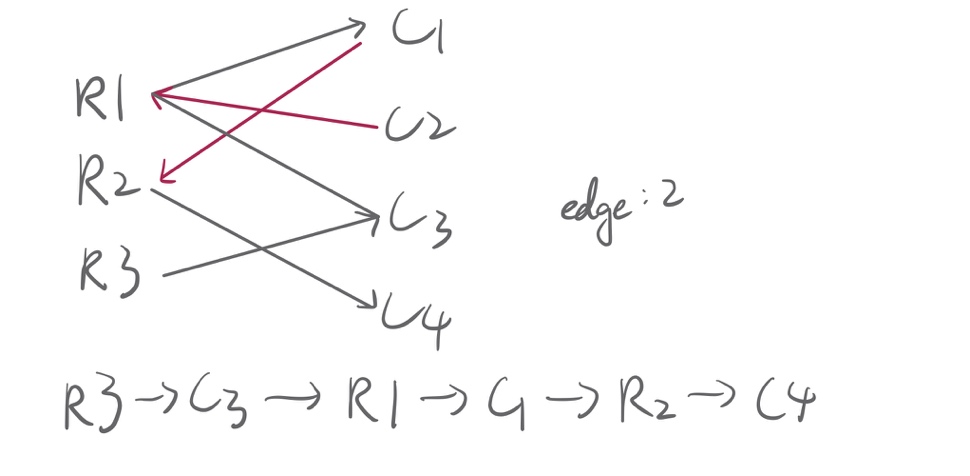
**題意範例：**

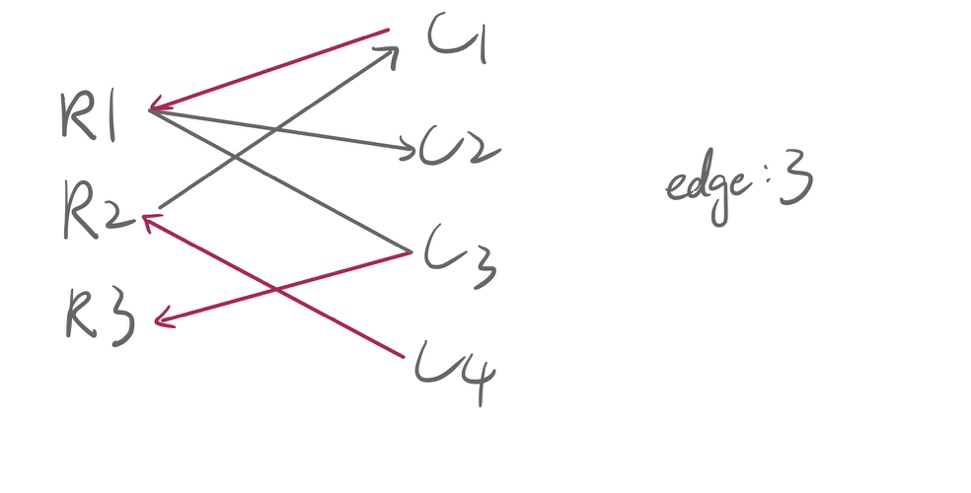
R = 4, C = 5, N = 8

**解法：**

先將每個點座標化為二分圖

交替路：從未匹配點出發，依序走未匹配邊、匹配邊、未匹配邊。

增廣路：從未匹配點出發，走交替路，若能達到另一個未匹配點，就是增廣路

已知反覆走增廣路可以增加匹配邊數量（範例由2變為3）。

Edmonds-Karp演算法：透過BFS或DFS反覆走增廣路，直到走到最大匹配。

找出的最大匹配 = 最大流量值(max flow) = 最小點覆蓋(min vertex cover) = 最小要發射幾顆炮彈

**解法範例：**

如上

**討論：**

當邊最大流量都是1時，Edmonds-Karp的複雜度為O(V·E)，每次BFS為O(E)，最多找V次增廣路，故worst case: O(2002 \* 10^6)，最多約一兩秒，不會超過Uva限制3s。

此外，此題還可以使用匈牙利演算法或是Hopcroft-Karp會更快。

**程式：**

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

struct Edge {

int dest, rev\_edge, capacity;

Edge(int d, int rev, int c) : dest(d), rev\_edge(rev), capacity(c) {}

};

struct MaxFlow{

vector<vector<Edge>> bipartite;

int N;

MaxFlow(int n) : N(n), bipartite(n){}

void add\_edge(int u, int v) {//容量固定為 1

bipartite[u].push\_back(Edge(v, (int)bipartite[v].size(), 1));//正向邊

bipartite[v].push\_back(Edge(u, (int)bipartite[u].size() - 1, 0));//反向邊 剛剛push過所以size-1

}

int bfs(int s, int t, vector<int>& vertex, vector<int>& edge){//vertex: 前驅點；parent\_edge：紀錄到哪條邊

fill(vertex.begin(), vertex.end(), -1);//-1: 未造訪

queue<int> q;

q.push(s);

vertex[s] = s;

while(!q.empty() && vertex[t] == -1){

int u = q.front();

q.pop();

for(int i = 0; i < (int)bipartite[u].size(); ++i){

Edge &tmp = bipartite[u][i];

if(tmp.capacity && vertex[tmp.dest] == -1){

vertex[tmp.dest] = u;

edge[tmp.dest] = i;

q.push(tmp.dest);

if(tmp.dest == t){

break;

}

}

}

}

return vertex[t] != -1;

}

int edmonds\_karp(int s, int t){

vector<int> vertex(N), edge(N);

int flow = 0;

while(bfs(s,t,vertex,edge)){

//所有邊容量皆為1，走增廣路

for(int v=t; v!=s; v=vertex[v]){

Edge &tmp = bipartite[vertex[v]][edge[v]];

tmp.capacity -= 1;

bipartite[v][tmp.rev\_edge].capacity += 1;

}

++flow;

}

return flow;

}

//residual network中從src可達的集合

vector<bool> reachable(int src){

vector<bool> visited(N, 0);

queue<int> q;

q.push(src);//從source開始

visited[src] = 1;

while(!q.empty()){//bfs

int u = q.front();

q.pop();

for(auto &tmp : bipartite[u])//遍歷residual network中每個邊

if(tmp.capacity && !visited[tmp.dest]){//還有residual capacity但但還沒造訪

visited[tmp.dest] = 1;

q.push(tmp.dest);

}

}

return visited;//1 -> 能在residual network中從source走到v

}

};

int main(){

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

int R, C, N;

while(cin >> R >> C >> N, R || C || N){

const int SRC = 0;//source

const int ROW = 1;

const int COL = ROW + R;

const int SNK = COL + C;//sink

//source -> row -> col -> sink

MaxFlow mf(SNK + 1);

//create edge: source -> row, col -> sink

for(int r = 0; r < R; ++r){

mf.add\_edge(SRC, ROW + r);

}

for(int c = 0; c < C; ++c){

mf.add\_edge(COL + c, SNK);

}

vector<vector<int>> enemy(R);

for(int i = 0; i < N; ++i){

int r,c;

cin >> r >> c;

--r;

--c;

mf.add\_edge(ROW+r, COL+c);

}

mf.edmonds\_karp(SRC, SNK);

//找min vertex cover

auto reach = mf.reachable(SRC);

vector<pair<char,int>> ans; //('r', row) and ('c', col)

for(int r = 0; r < R; ++r){

if(!reach[ROW + r]){

ans.push\_back(pair<char, int>('r', r + 1));

}

}

for(int c = 0;c < C; ++c){

if(reach[COL + c]){

ans.push\_back(pair<char, int>('c', c + 1));

}

}

cout << ans.size();

for(auto [tp,id] : ans){

cout << ' ' << tp << id;

}

cout << "\n";

}

}