Given a map of gold mine which can be represented as a NxN matrix  
In this map, there are several gold mines that locates on random places (maximum 4 mines each map). A camp will be setup in the map for mining the gold. Every day, workers will go from the camp to gold mines location. In order to reduce the cost, company want to setup the camp so that the distance from it to mining location is optimized.  
Let consider below 5x5 map :  
  
- Worker can not go through the rock  
- It takes 1 hour to travel a cell in map  
There are two mines, from start point (camp location), it would take 4h to go to each gold mine. Let's suppose the final cost will be the highest cost among them, then if we put the camp as above figure, the final cost will be 4.  
Now, we consider other example :  
  
In the 3rd example, the final cost will be minimum (1)  
[Input]  
The first line of input will be the number of test case T (T ≤ 50)  
In each TC :  
- The first line will give the size of map N (N ≤ 20) and the number of gold mines G (2 ≤ G ≤ 4)  
- The next G lines will give location R (row) & C (column) of gold mine (1-base indexed)  
- The next N lines will give the map's data : 1 represents the road & gold mines, 0 represents the rock (can not go through)  
  
[Output]  
Your program should output the Minimum final cost for traveling from camp to gold mines.  
Case #1  
1  
Case #2  
2  
Case #3  
2  
Case #4  
7  
Case #5  
6  
  
[Constraints]  
- You can move on 4 directions (up/left/down/right)  
- You can not setup the camp on gold mine location or rock  
- You can go pass a gold mine then go to other gold mine  
- You must place camp so that workers can go to all gold mines

5  
5 2  
4 3  
3 4  
1 1 0 0 0  
1 1 0 0 0  
1 1 1 1 1  
1 1 1 0 1  
1 1 1 1 1  
8 2  
5 6  
6 4  
1 1 1 1 1 1 0 0  
1 1 1 1 1 1 1 0  
1 1 0 1 0 1 1 0  
1 1 1 1 0 1 1 0  
1 1 1 1 1 1 1 0  
1 1 1 1 1 1 1 0  
0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0  
10 3  
8 2  
5 3  
7 1  
0 0 0 1 1 1 1 1 1 0  
1 1 1 1 1 1 1 1 1 0  
1 0 0 1 0 0 0 0 1 0  
1 1 1 1 1 1 1 1 1 1  
1 1 1 1 0 1 0 0 1 1  
1 1 1 1 0 1 0 0 1 1  
1 1 1 1 0 1 0 0 1 1  
1 1 1 1 1 1 1 1 1 1  
1 1 1 0 0 1 0 0 1 1  
1 1 1 1 1 1 1 1 1 1  
8 3  
1 2  
2 8  
5 8  
0 1 0 0 0 0 0 0  
0 1 0 1 1 1 1 1  
0 1 0 1 0 0 0 1  
0 1 1 1 0 0 0 1  
0 1 0 1 0 0 0 1  
0 1 0 1 1 1 1 1  
0 1 0 0 1 0 0 0  
0 1 0 0 1 0 0 0  
10 4  
9 6  
1 9  
7 5  
7 3  
1 1 1 1 1 1 0 1 1 0   
1 1 1 0 1 1 0 1 1 1   
1 1 1 1 1 1 1 1 1 1   
1 1 1 0 1 1 0 1 1 1   
1 1 1 1 1 0 1 1 1 1   
1 0 1 1 0 1 1 1 1 1   
1 0 1 1 1 1 1 0 1 1   
1 0 1 1 1 1 0 1 1 1   
1 1 1 1 1 1 1 1 1 1   
1 0 1 0 1 1 1 1 0 1

package luyende;  
  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.util.Scanner;  
  
/\*  
\* Gold Mining: cách làm giống bài Clearning Robot  
ý tưởng: đặt trại tại 1 trong các vị trí (bài toán sinh hoán vị)  
rồi so sánh xem khoảng cách từ điểm đó tới mỏ vàng ở xa nhất với chi phí nhỏ nhất  
Phải tự đặt trại.  
Theo bài cho thì ko được đặt trại ở vị trí mỏ vàng. Nếu đặt ở vị trí mỏ vàng thì break luôn,  
nếu đặt trại ở vị trí mà xung quanh nó toàn là đá => ko đi ra ngoài được => loại  
đặt trại ở tất cả vị trí trong map (for từ trái qua phải, từ trên xuống dưới)  
1 đại diện cho các mỏ vàng và đường, 0 đại diện cho đá (không thể đi qua)  
\*/  
public class GoldMining {  
Scanner sc = new Scanner([System.in](http://system.in/));  
int t, n, g, st, en;  
int map[][], gold[][], khoangCach[][];  
int dx[] = { 1, -1, 0, 0 };  
int dy[] = { 0, 0, 1, -1 };  
int queueX[] = new int[1000000];  
int queueY[] = new int[1000000];  
  
void init() {  
map = new int[n][n];  
gold = new int[g][2];  
khoangCach = new int[n][n];  
}  
  
void resetKhoangCach() {  
khoangCach = new int[n][n];  
}  
  
void input() {  
n = sc.nextInt();  
g = sc.nextInt();  
init();  
for (int i = 0; i < g; i++) {  
gold[i][0] = sc.nextInt()-1;  
gold[i][1] = sc.nextInt()-1;  
}  
for (int i = 0; i < n; i++) {  
for (int j = 0; j < n; j++) {  
map[i][j] = sc.nextInt();  
}  
}  
}  
  
void BFS(int startX, int startY) {  
st = en = 0;  
queueX[en] = startX;  
queueY[en++] = startY;  
khoangCach[startX][startY] = 1;  
while (st != en) {  
int x = queueX[st];  
int y = queueY[st++];  
for (int i = 0; i < 4; i++) {  
int a = x + dx[i];  
int b = y + dy[i];  
if (a >= 0 && a < n && b >= 0 && b < n) {  
if (map[a][b] != 0 && khoangCach[a][b] == 0) {  
queueX[en] = a;  
queueY[en++] = b;  
khoangCach[a][b] = khoangCach[x][y] + 1;  
}  
}  
}  
}  
}  
void resultKhoangCach(){  
for(int i=0; i<n; i++){  
for(int j=0; j<n; j++){  
System.out.print(khoangCach[i][j]+" ");  
}  
System.out.println();  
}  
}  
void solution() {  
t = sc.nextInt();  
for (int tc = 1; tc <= t; tc++) {  
input();  
int min = 999999;  
// sinh hoán vị tất cả các trường hợp đặt trại  
for(int i=0; i<n; i++){  
for(int j=0; j<n; j++){  
int max=0;  
boolean check=true, check2=false;  
// vị trí đặt trại ko phải là mỏ vàng  
for(int v=0; v<g; v++){  
if(gold[v][0] == i && gold[v][1]==j){  
check = false;  
}  
}  
// vị trí đặt trại là đường đi, tránh mỏ vàng, mỏ đá  
for(int h=0; h<4; h++){  
int a = i + dx[h];  
int b = j + dy[h];  
if(a>=0 && a<n && b>=0 && b<n){  
if(map[a][b] == 1){  
check2 = true;  
break;  
}  
}  
}  
if(map[i][j] == 1 && check==true && check2==true){  
BFS(i,j);  
System.out.println("khoang cach");  
resultKhoangCach();  
// g mỏ vàng  
for(int s=0; s<g; s++){  
if(khoangCach[gold[s][0]][gold[s][1]] > max){  
max = khoangCach[gold[s][0]][gold[s][1]];  
}  
}  
if(min > max){  
min = max;  
}  
resetKhoangCach();  
}  
}  
  
}  
System.out.println("Case #"+tc);  
System.out.println(min-1);  
}  
}  
  
public static void main(String args[]) throws Exception {  
System.setIn(new FileInputStream("GoldMining.txt"));  
GoldMining g = new GoldMining();  
g.solution();  
}  
}

Gold Minin

Gold Mining

package luyende;  
  
import java.io.FileInputStream;  
import java.util.Scanner;  
  
public class GoldMining {  
static int T;  
static int M, N;  
static int[][] A;  
static int[][] B;  
static int[][] visit;  
static int[][] khoangcach;  
static int[] dx = { -1, 0, 0, 1 };  
static int[] dy = { 0, 1, -1, 0 };  
static int[][] Q;  
static int min;  
  
static void BFS(int i, int j) {  
int f, r;  
f = r = 0;  
  
Q[r][0] = i;  
Q[r][1] = j;  
khoangcach[i][j] = 1;  
r++;  
  
while (f != r) {  
// pop ()  
int x = Q[f][0];  
int y = Q[f][1];  
f++;  
  
for (int v = 0; v < 4; v++) {  
int a = x + dx[v];  
int b = y + dy[v];  
if (a >= 0 && b >= 0 && a < N && b < N) {  
if (khoangcach[a][b] == 0 && A[a][b] == 1) {  
khoangcach[a][b] = khoangcach[x][y] + 1;  
//System.out.println(khoangcach[a][b]);  
Q[r][0] = a;  
Q[r][1] = b;  
r++;  
  
}  
}  
}  
}  
  
}  
  
public static void main(String args[]) throws Exception {  
System.setIn(new FileInputStream("GoldMining.txt"));  
Scanner sc = new Scanner([System.in](http://system.in/));  
Q = new int[100000][2];  
T = sc.nextInt();  
for (int tc = 1; tc <= T; tc++) {  
N = sc.nextInt();  
M = sc.nextInt();  
A = new int[N][N];  
B = new int[M][2];  
khoangcach = new int[N][N];  
for (int i = 0; i < M; i++)  
for (int j = 0; j < 2; j++)  
B[i][j] = sc.nextInt() -1;  
  
for (int i = 0; i < N; i++)  
for (int j = 0; j < N; j++)  
A[i][j] = sc.nextInt();  
min = 1000000;  
  
for (int i = 0; i < N; i++)  
for (int j = 0; j < N; j++) {  
boolean check = true;  
int max = 0;  
// đánh dấu là mỏ vàng  
for (int v = 0; v < M; v++) {  
if (B[v][0] == i && B[v][1] == j)  
check = false;  
}  
// nếu xung quanh là đường đi hoặc là vàng  
boolean check2 = false;  
for (int v = 0; v < 4; v++) {  
int a = i + dx[v];  
int b = j + dy[v];  
if (a >= 0 && b >= 0 && a < N && b < N) {  
if (A[a][b] == 1) {  
check2 = true;  
break;  
}  
  
}  
}  
if (A[i][j] == 1 && check && check2) {  
BFS(i, j);  
// Chứng tỏ vừa xong mình in nhầm mảng khoảng cách  
// System.out.println("khoang cach");  
// for (int m = 0; m < N; m++){  
// for (int n = 0; n < N; n++){  
// System.out.print(khoangcach[m][n]+" ");  
// }  
// System.out.println();  
// }  
  
// xét khoảng cách ở các vị trí mỏ vàng  
for (int s = 0; s < M; s++) {  
// không thể thiết lập trại trên vị trí mỏ vàng hoặc đá  
// phải đặt trại để công nhân có thể đi đến tất cả các mỏ vàng  
if (khoangcach[B[s][0]][B[s][1]] > max) {  
max = khoangcach[B[s][0]][B[s][1]];  
//System.out.println(max);  
// if (max == 0) {  
// System.out.println(i + " " + j);  
// System.out.println(B[s][0] + " " + B[s][1]);  
// }  
}  
  
}  
// lấy khoảng cách chi phí nhỏ nhất  
if (min > max)  
min = max;  
}  
// đi xong reset lại mảng khoangcach[][]  
for (int m = 0; m < N; m++)  
for (int n = 0; n < N; n++)  
khoangcach[m][n] = 0;  
}  
System.out.println("Case #" + tc);  
System.out.println(min - 1);  
  
}  
}  
}