Given the map of BTS consists of N x M cells. Each cell in map have a cost and can connect to other 6 cells around it. Two cells are connected if they share a same edge.  
We call a group of 4 connected cells a cluster. Your program should find the cluster with highest cost sum and print out the square value of its sum  
Ex: M = 5 , N = 3  
  
In above example, we have a cluster with square of sum   
(300 + 410 + 185 + 95)2 = 980100  
  
  
In above example, the selected cells is not a cluster.  
  
[Input]  
- The first line is the number of test cases T (T <= 50)  
- In each TC :  
+ The first line give the size of the map M, N ( 3 ≤ N, M ≤ 15 )  
+ In the next N lines, the cost C (0 ≤ C ≤ 1000) of each cell is given as below rule  
  
5  
5 3  
300 410 150 55 370  
120 185 440 190 450   
165 70 95 420 50   
5 5  
356 55 41 453 12   
401 506 274 506 379   
360 281 421 311 489   
425 74 276 371 164   
138 528 461 477 470  
  
[Output]  
Print out the square value of maximum cluster's sum  
Case #1  
2250000  
Case #2  
3748096  
Case #3  
3928324  
Case #4  
7236100  
Case #5  
13104400

package luyende;  
  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.util.Scanner;  
  
/\*  
\* chọn tổ hợp của 4 ô liền nhau có giá trị lớn nhất trong n\*m ô  
\* in ra tổng bình phương của 4 ô đó.  
\* duyệt backtracking 2 lần, duyệt theo độ sâu (kề với đỉnh) vs duyệt đi xung quanh  
\* cách 2: duyệt backtracking 1 lần  
\*   
\* backTracking 2 lần  
1 lần duyệt 1 đỉnh có 6 đỉnh kề xung quanh, chọn 4 trong 6 đỉnh có tổng lớn nhất  
1 lần duyệt 1 đỉnh chọn 4 đỉnh liền kề vs đỉnh đó bất kỳ trong map rồi sao cho có tổng lớn nhất  
\*/  
public class BaseStation {  
Scanner sc = new Scanner([System.in](http://system.in/));  
int t, n, m;  
long max;  
int map[][];  
boolean visit[][];  
int dxChan[] = { -1, -1, -1, 0, 0, 1 };  
int dyChan[] = { -1, 0, 1, -1, 1, 0 };  
int dxLe[] = { -1, 0, 0, 1, 1, 1 };  
int dyLe[] = { 0, -1, 1, -1, 0, 1 };  
  
void init() {  
max = 0;  
map = new int[n][m];  
visit = new boolean[n][m];  
}  
  
// backTrack theo chiều sâu  
void backTrack(int x, int y, int num, int s) {  
if (num == 3) {  
if (s > max) {  
max = s;  
}  
return;  
}  
visit[x][y] = true;  
if (y % 2 == 0) {  
for (int i = 0; i < 6; i++) {  
int a = x + dxChan[i];  
int b = y + dyChan[i];  
if (a >= 0 && a < n && b >= 0 && b < m) {  
if (visit[a][b] == false) {  
visit[a][b] = true;  
backTrack(x, y, num + 1, s + map[a][b]);  
visit[a][b] = false;  
}  
}  
}  
} else {  
for (int i = 0; i < 6; i++) {  
int a = x + dxLe[i];  
int b = y + dyLe[i];  
if (a >= 0 && a < n && b >= 0 && b < m) {  
if (visit[a][b] == false) {  
visit[a][b] = true;  
backTrack(x, y, num + 1, s + map[a][b]);  
visit[a][b] = false;  
}  
}  
}  
}  
visit[x][y] = false;  
}  
  
// backTrack đỉnh liền kề  
void backTrack2(int x, int y, int num, int s) {  
if (num == 3) {  
if (s > max) {  
max = s;  
}  
return;  
}  
visit[x][y] = true;  
  
if (y % 2 == 0) {  
for (int i = 0; i < 6; i++) {  
int a = x + dxChan[i];  
int b = y + dyChan[i];  
if (a >= 0 && a < n && b >= 0 && b < m) {  
if (visit[a][b] == false) {  
visit[a][b] = true;  
backTrack2(a, b, num + 1, s + map[a][b]);  
visit[a][b] = false;  
}  
}  
}  
} else {  
for (int i = 0; i < 6; i++) {  
int a = x + dxLe[i];  
int b = y + dyLe[i];  
if (a >= 0 && a < n && b >= 0 && b < m) {  
if (visit[a][b] == false) {  
visit[a][b] = true;  
backTrack2(a, b, num + 1, s + map[a][b]);  
visit[a][b] = false;  
}  
}  
}  
}  
  
visit[x][y] = false;  
  
}  
  
void solution() {  
t = sc.nextInt();  
sc.nextLine();  
for (int tc = 1; tc <= t; tc++) {  
m = sc.nextInt();  
n = sc.nextInt();  
  
init();  
  
for (int i = 0; i < n; i++) {  
for (int j = 0; j < m; j++) {  
map[i][j] = sc.nextInt();  
}  
}  
  
for (int i = 0; i < n; i++) {  
for (int j = 0; j < m; j++) {  
backTrack(i, j, 0, map[i][j]);  
backTrack2(i, j, 0, map[i][j]);  
}  
}  
  
System.out.println("Case #" + tc);  
System.out.println(max \* max);  
}  
}  
  
public static void main(String[] args) throws Exception {  
System.setIn(new FileInputStream("BaseStation.txt"));  
BaseStation b = new BaseStation();  
b.solution();  
}  
  
}  
// void bt(int x, int y, int step, int sum){  
// if()  
// for(int i = 0 ; i < 6; ++i){  
// int tx;  
// int ty;  
// if(){  
// visit[tx][ty]=true;  
// bt(x, y, step+1)  
  
// }  
//  
//  
// }  
//  
// }

// lý do sai là điều kiện dừng ở 4 phải sửa lại thành 3,sinh tổ hợp  
sinh nhị phân  
đường đi có chi phí nhỏ nhất

đáng ra phải nghĩ ngay đó là giá trị bị vượt quá đáp án (lỗi logic)  
thì điều kiện dừng bị sai