**Square and Tower**

In a city, there are N mobile signal towers. Raj is creating an application to visualize the coverage area of the signal towers.

        For simplicity, the following are assumed:   
1) the shape of the city is square.   
2) the signal coverage is in the form of a square.   
3) the side length of the signal coverage is **3 meters**.

        Given side length of the city, number of towers, and coordinates of towers, find whether whole city is under coverage.  If the whole city is not under signal coverage, find how much area is not under coverage.   
  
Assume that the two of the end of points of the city are  (0,0) and (n-1,n-1).   
**Input Format:**   
First input corresponds to n, length of the city.   
Second input corresponds to the number of towers, t.   
The next n lines of the input follow the following format: x and y coordinates of each tower, respectively.   
  
Consider that tower is always placed at valid position.   
  
**Output Format:**   
Print "Yes" if the whole city is under coverage. Else, print "No", along with the left out area.   
  
**Sample Input 1:**

5

3

2 2

0 1

4 1   
  
**Sample output 1:**

No

8   
  
**Sample Input 2:**   
3   
1   
1 1   
**Sample Output 2:**   
Yes   
  
  
**Explanation for sample 1:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 |  | **T3** |  |  |  |
| 3 |  |  |  |  |  |
| 2 |  |  | **T1** |  |  |
| 1 |  |  |  |  |  |
| 0 |  | **T2** |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 |

Green colour indicates the coverage area.   
Red colour indicates the area which is not under coverage.   
Total red area = 8.

**package** tower;

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** App {

**public** **static** **void** main(String[] args) {

Scanner sc=**new** Scanner(System.***in***);

**int** cityLength=sc.nextInt();

**int** city[][]=**new** **int**[cityLength][cityLength];

**int** noOfTowers=sc.nextInt();

**int** tower[][]=**new** **int**[noOfTowers][2];

**for**(**int** i=0;i<noOfTowers;i++)

{

tower[i][0]=sc.nextInt();

tower[i][1]=sc.nextInt();

}

// System.out.println(Arrays.deepToString(city));

//loop for each tower, and fix 1s in the city array

**for**(**int** i=0;i<noOfTowers;i++)

{

**int** x=tower[i][0];

**int** y=tower[i][1];

city[x][y]=1; //tower is here. So coverage is here.

**for**(**int** j=-1;j<=1;j++)

{

**for**(**int** k=-1;k<=1;k++)

{

**int** x1=x+j;

**int** y1=y+k;

**if**(x1<cityLength && y1<cityLength && x1>=0 && y1>=0)

{

city[x+j][y+k]=1;

}

}

}

}

**int** zeroCount=0;

**for**(**int** i=0;i<cityLength;i++)

{

**for**(**int** j=0;j<cityLength;j++)

{

// System.out.print(city[i][j]+"\t");

**if**(city[i][j]==0)

{

zeroCount++;

}

}

// System.out.println();

}

**if**(zeroCount==0)

{

System.***out***.println("Yes");

}**else**

{

System.***out***.println("No");

System.***out***.println(zeroCount);

}

sc.close();

}

}

**Mobile Lock Pattern** 

Given the coordinates of  a lock pattern configuration, check whether it is a Valid Lock Pattern or not. A valid Lock pattern will be formed only by joining the adjacent neighbours.  If it is a valid lock pattern and if there is   
       1)   Above 75% coverage, print "Excellent",   
       2)  Above 50% coverage, print "Good",   
       3)  Above 25% coverage, print "Average",   
       4)  Else print "Poor".   
  
(Coverage need to be calculated with respect to the total grid size)   
  
        If it is an  invalid lock pattern, print "Invalid".   
  
**Assume** that the pattern grid will always be a square.   
**Note:**   
To make the problem less complicated the following relaxations are made:   
1) The testcases will not contain any overlap through a point in the pattern.   
2) The pattern will always end at a new point.   
3) The points given in the testcases always lie within the pattern grid. No need to check if the point is outside the grid.

**Input Format:**   
The first line is an integer that corresponds to the side length of the pattern grid.   
The second line is an integer that corresponds to the number of coordinates in the lock pattern, n.   
The following n lines have the following pattern : x-coor y-coor   
  
**Output Format:**   
Print "Invalid" if there is the pattern is not properly linked, else print "Excellent " or "Good" or "Average" or "Poor", based on the number of points covered.   
  
**Sample Input 1:**   
3   
5   
1 1   
2 1   
3 1   
2 2   
3 3   
**Sample Output 1:**   
Good   
  
**Sample Input 2:**   
3   
6   
1 1   
2 1   
3 1   
2 2   
3 1   
3 3   
**Sample Output 2:**   
Invalid

