Once upon an ancient time, a knight was preparing for the great battle in GridLand. The GridLand is divided into square grids. There are R horizontal and C vertical grids. Our particular knight in this case can always give an (M, N) move, i.e. he can move M squares horizontally and N squares vertically or he can move M squares vertically and N squares horizontally in a single move. In other words he can jump from square (a, b) to square (c, d) if and only if, either $(|a-c| = M$ and $|b-d| = N)$ or $(|a-c| = N$ and $|b-d| = M)$. However, some of the squares in the war field are filled with water. For a successful jump from one square to another, none of the squares should contain water. Now, the knight wants to have a tour in the war field to check if everything is alright or not. He will do the following:

a) He will start and end his tour in square $(0, 0)$ but visit as many squares as he can.

b) For each square s_i, he counts the number k_i of distinct squares, from which he can reach s_i in one jump (satisfying the jumping condition). Then he marks the square as an even square if k_i is even or marks it odd if k_i is odd. The squares he cannot visit remain unmarked.

c) After coming back to square $(0, 0)$ he counts the number of even and odd marked squares. He can visit a square more than once.

You, as an advisor of the knight, suggested that, he can do it without visiting all the squares, just by writing a program. So the knight told you to do so. He will check your result at the end of his visit.

Input

The first line of input will contain T (≤ 50) denoting the number of cases.

Each case starts with four integers R, C, M, N $(1 < R, C \leq 100,\ 0 \leq M, N \leq 50,\ M + N > 0)$. Next line contains an integer W $(0 \leq W < R * C)$, which is the number of distinct grids containing water. Each of the next W lines contains a pair of integer x_i, y_i $(0 \leq x_i < R,\ 0 \leq y_i < C,\ x_i + y_i > 0)$.

Output

For each case, print the case number and the number of even and odd marked squares.

Sample Input

```
2
3 3 2 1
0
4 4 1 2
2
3 3
1 1
```

Sample Output

```
Case 1: 8 0
Case 2: 4 10
```