Problem F: Path of Least Persistence

Bob Roberts owns a company that makes novelty games for parties. One of his favorites is a puzzle labyrinth, where players must navigate a path through a rectangular grid based on puzzles they must solve at each grid location. The solution to each puzzle indicates which grid square to go to next, and this continues until the player reaches the destination square. Bob has software to lay out the intended path, but there was a glitch in the algorithm and the resulting grids often have either very long paths or paths that don't lead to the destination. For example, consider the 3x3 grid shown on the left in Figure 1.

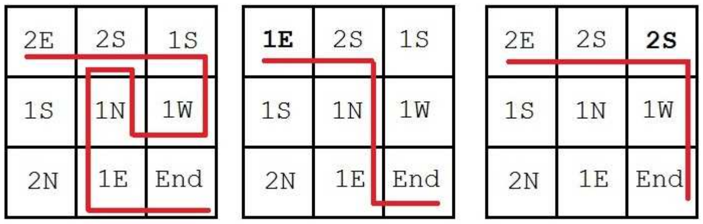

Figure 1

The solution to each puzzle is shown in each grid square, which gives the number of squares to move and the direction to move (either North, South, East or West). In this grid the path from the start square (always the upper left) to the destination square (always the lower right) has six segments, which is very unsatisfying to many customers (who aren't persistent enough to solve so many puzzles). Bob noticed this when setting up the grid, and because he didn't have much time, he looked to see if there was one grid square solution he could change to make the path shorter. After a quick look, he changed the puzzle in the upper left square so that its solution changed from 2E to 1E (the middle figure) which led to a path of length 3 – not ideal, but better than the original path. After the party was over, Bob realized that there was an even better solution, shown in the right figure, that led to a path of length 2 (note that by path length we mean the number of puzzles solved while on the path)

Changing a solution to a puzzle is somewhat complicated and time consuming, so Bob only wants to change at most one puzzle when setting up a grid. He has come to you to help solve the following problem: given a grid layout, which is the one square direction to change to minimize the path from the start to the destination. Bob realizes that sometimes there is no answer (since it might take two or more changes to actually get a path) and sometimes there is no need to change any square, since any change would result in a path no shorter than the one that already exists. He is confident that you can handle all these cases.

Input **Time Limit: 3 secs, No. of Test Cases: 76, Input File Size 1157K**

The first line of each case contains two positive integers n m indicating the number of rows and columns in the grid, where $n, m \leq 100$. Following that are $nm - 1$ direction specifications for the grid, given row-wise left to right starting at the topmost row. These are in the form r`D`, where r is the number of grid squares to move and `D` is the direction to move, either '`N`' (up), '`S`' (south), '`E`' (right), or '`W`' (you figure it out). Each grid will contain at least 2 grid squares. A line containing two 0's will terminate input.

Output

For each test case, output the case number followed by one of three possible answers. If there is no way to change one square to get a path to the destination, output `impossible`. If the existing path in the grid cannot be made shorter, output `none` l, where l = length of the existing path. Otherwise, output the row and column of the square to change (where the top row is row 0 and the leftmost column is column 0) followed by the new direction for that square, followed by the length of the new, shortest path. If there are two or more changes that could be used, display the one that has the lowest row number. If there are two or more with the lowest row number, display the one with the lowest column number. Finally, if there is still a tie use the change that comes first in the following ordering: `1E` < `1N` < `1S` < `1W` < `2E` < … etc.

Sample Input

```
3 3
2E 2S 1S 1S 1N 1W 2N 1E
2 4
1E 1W 1W 1W 1W 1W 1W
1 4
3E 1E 1E
0 0
```

Sample Output

```
Case 1: 0 2 2S 2
Case 2: impossible
Case 3: none 1
```