REMEMBER: The itertools and copy libraries in Python are banned.

1. Overview

In this project, you will be solving mazes. You will be reading in a maze definition from a file; these input files look like this:

```
####S###########   ####
#  #    #      #####  #
#  ###  #####         #
#    #  #   #         #
#  ###  ### #         #
#           #         #
#         ####   ######
#    E#####
```

Notice that a map file has exactly one S mark (your start point), and exactly one E mark (your end point). The hash marks represents paths that you can follow.

A classic strategy for solving a maze is the "left-hand rule." Imagine that you place your left hand on the wall, and you never let it leave; you follow the wall wherever it goes. If a doorway opens to the left; you go in; if the corridor turns right (with no left or straight) you follow that; if you hit a dead end, then you follow the wall around, and walk back the way you came.

The left-hand rule is guaranteed to solve the maze (not necessarily quickly, but eventually) so long as two things are true:

• The maze is **planar** - meaning that it's possible to lay it out flap on a map, like a piece of paper. It doesn't have any bridges or tunnels.

• The maze is **acylic** - meaning that it doesn't have any loops.

I will guarantee that every map file I ask you to solve will have both of these properties.

2. The Algorithm

It's pretty easy to imagine a person walking through a maze, using their hand to follow the wall. But how do we turn that into code, where the maze (and our position in it) is based on a fixed grid? As it turns out, there is a very simple algorithm, which your code must use.

To start with, you must keep track of the position of the solver, as (x, y) coordinates. ((0, 0) is the upper-left corner of the maze.) Also, keep track of the direction that the solver is facing (N, W, S, E).

Each time that you move forward (in your current direction), look at the spaces adjacent to where you land. Can you turn left? If that's possible, do so. If not, is it possible to keep going straight? Do so. If not, is it possible to turn right? Do so. If none of these are possible - if you have reached a dead end - then turn around.

Thus, after every move, the solver will be facing a position that they can walk.

2.1 Example

In this example, we will use a very small portion of the map to show you how to move. In the initial position, the solver is facing North:

```
####
#  #
   ###
   ^
########
#
########
```

(When we draw the map, we use a caret ^ to indicate when the solver is facing North.)

On the first move, the solver moves North by one step. It is not possible to turn left, but it is possible to continue straight, and so the solver does not change direction.

```
####
#  #
   ^##
   .
########
#
########
```

(Also note that, when you leave a place where you have been, change the hash sign to a period. It will make it easier to see where the solver is moving over time.)

On the second move, the solver contines North. But on the third move, we find that it’s possible to turn left. The solver does so:

```
####
#  ^
   .##
   .
########
#
########
```

then

```
###<
#  .
   .##
   .
########
#
########
```

After a few more moves, the solver turns left again:

```
v...
#   .
    .##
    .
########
#
########
```

and then eventually hits a dead end, where they turn around:

```
....
^   .
    .##
    .
########
#
########
```

After turning around at the dead end, the next move is to backtrack North, but the solver immediately hits a point where they must do a right turn.

```
>...
.   .
    .##
    .
########
#
########
```

They then continue back along the path, one space at a time, until they find a place where it's possible to turn left again.

```
....
.   .
    >##
    .
########
#
########
```

3. Your Program

You will write a program named maze_solver.py . It will first ask for the name of the “maze file” it should read - you’ve seen how this file looks, earlier in this spec. You may assume that the file exists, and that the contents of the file are a reasonable maze: it’s acyclic, it has exactly one start and exactly one end, and of course there’s a path that can be found from the start to the end.

You will start with the solver on the Start position (the S in the map). Face them North if possible - but if it won’t be possible for them to move North on the first move, rotate them left until they are facing a workable direction. Then display the map (see the .out files for details about what that looks like).

Now, move the solver around, according to the rules discussed above. Redraw the map after every single step, making sure to change hash marks to periods in places where the solver has been.

Terminate the program with a message (see the .out files) when you finally reach the end.

4. Data Structures

There are a variety of ways that you might store the map. I’m not going to tell you how you must store it. However, I’ll recommend that you store the data as a 2D array: that will make it easy to access (based on the current (x, y) location), and easy to modify (when you want to change hashes to periods).