Project 2 – Graph Traversal
COT 6405 – Fall 2021
Due 11/30/201 by 11:59pm

1 Introduction

This project requires you to model the problem below as a graph and then use a known graph algorithm to solve the problem.

You must traverse a field of arrows (red or blue). You must find a route from the arrow in the top left corner to the bullseye in the bottom right corner. You must follow the direction that the arrows point, and you can only stop on the other colored arrow or the bullseye. For example, start on red, then chose a blue arrow (in the direction that the red arrow is pointing), then from the blue arrow chose a red arrow in the direction the blue arrow is pointing. Continue in this fashion until you find the bullseye in the bottom right corner. It does not have to be the first opposite color that you find. You may find your-self in a loop and continuously visiting the same arrows; you need to account for this. You must find the correct path. You also need to handle what happens if you get to a node that you already visited within a path and you need to continue in that direction.

2 Program

This project must be written in python. You can have multiple python files; however, they must be run from **Project2.py** (this file/name is required). We have scripts that will test this so naming is important. For this project, you are allowed to use any graph structure and traversal algorithm. The program must take two command line parameters: (1) the input file name; and (2) the output file name. See below for an example.

python Project2.py input.txt output.txt

2.1 Comments

For this project, there is no required report. In place of this, comments will be graded/count as a "report." You do not have to comment every single variable/line, however, you do need to leave detailed comments for any function, methods, classes, etc. It needs to be clear that you understand what each of them are doing.

Note: Comments are required for this project. If there are no comments, you will receive a 0 for this project.

2.2 Input file format

Your program should read its input from a file with the following format. The input begins with two positive characters on a line indicating the number of rows r and columns c of the maze, respectively. The next r lines contain the color and directional information for each arrow in the maze. Each line has c values, where each value represents the color of the arrow by the direction of the arrow (N, E, S, W, NE, SE, SW, or NW). The color codes R and B represent red and blue, respectively, while the direction codes represent north, east, south, west, northeast, southeast, southwest, or northwest, respectively. The bulls-eye is represented by the letter O. You may

```
 8      8
R-E    R-SE   B-S    B-SW   R-S    R-SW   R-S    R-S
B-E    R-S    B-SE   R-E    B-SE   B-S    B-W    R-SW
R-N    B-W    B-SW   R-SE   R-NE   B-SW   B-W    R-W
R-SE   R-SE   B-SW   R-SE   R-S    B-NW   R-E    B-NW
B-NE   R-W    R-S    B-S    B-E    B-NE   B-NW   R-NW
R-S    B-SE   R-SE   R-SE   R-NW   R-NE   B-E    R-W
R-NE   B-W    B-SE   R-E    R-E    B-E    B-NW   R-SW
B-NE   R-E    B-N    R-NE   B-NE   B-N    B-NW   0
```

Figure 1: Example input graph.

assume that the bulls-eye will always be in the bottom-right corner of the maze. See Figure 1 for an example input file.

2.3 Output file format

You must write the output file in the following format. The output will consist of a path from the top left square to the bottom right square (bulls-eye). Write a single line consisting of a sequence of moves, separated by spaces. Each move should be represented by the number of spaces to move and the direction, with no spaces in between. The direction should be represented using N, E, S, W, NE, SE, SW, and NW, as in the input. The sequence of moves must solve the maze from the input. For example, if your first 3 moves take you 3 spaces east, 3 spaces southwest, and 4 spaces southeast, your output should begin as follows (note, the 3 moves below correspond to the first three possible moves to the graph in Figure 1): **3E 3SW 4SE**

You will be given multiple mazes to test, along a script that you can use to determine if your solution is correct. The output file must follow a particular naming convention: input_file_name-soln.txt (see example below). The script to test your solutions is **verifyGraph.py**. This script takes two command line parameters: (1) the input graph; and (2) and your solution file. See below for an example of running it.

python verifyGraph.py small.txt small-soln.txt

3 Submission

Submit a zip archive with the following naming format lastname_project2.zip. For example, my submission would be canavan_project2.zip. **If this naming convention is not used, 5 points will be deducted from your total grade**. The zip file must contain the following:

1. All python files needed to run your program
2. README file that lists any packages needed to run your program.

3. Sample output files that your program created from each test maze.

NOTE: Late submission will *NOT* be accepted for this project. Please be sure to allow enough time to upload your project to Canvas.

4 Grading

Algorithm Implementation: 80%
Comments: 20%