1 Overview

This project requires you to model the problem below as graph and then use a known graph algorithm to solve the problem. **You are not allowed to use the internet or consult any references. This is an individual project. This policy will be strictly enforced.**

This problem is based on the "Apollo and Diana" maze problem (from "MAD MAZES: Intriguing Mind Twisters for Puzzle Buffs, Game Nuts and Other Smart People" by Robert Abbott). The text of the problem is quoted below. A diagram of the maze is provided separately.

> Apollo, god of the sun, stole a set of arrows from his sister Diana, goddess of the moon. When Diana caught him shooting moonbeams instead of sunbeams, she was furious! She cast a mighty curse, and all his arrows fell around him in a terrible maze. That maze is shown here. Apollo's sunbeams are the red arrows; Diana's moonbeams are the blue arrows.
>
> Diana spoke to Apollo: "You will remain here in this field of arrows until you solve this puzzle. You must find a route from the red arrow at the upper left to the bull's-eye at the lower right. The first red arrow points to two blue arrows. Choose either of those blue arrows and move to it. You are now on a blue arrow that points to one or more red arrows. Choose one of those red arrows. and move to it. Continue in this fashion, alternating between red and blue arrows. If you are truly wise you will arrive at an arrow that points to the bull's eye. You may then proceed to the bull's eye and escape from the maze. It's far more likely, though, that you will wander into a loop and find yourself going around endlessly. If that happens, you can admit that you are lost, go back to the red arrow at the upper left, and start the puzzle over again."
>
> So, how can Apollo find his way to the bull's eye (without getting stuck in too many loops)?

2 Modeling the problem

Before you write a program to solve this problem, you will first write a report describing (in English and pseudocode) how you will solve this problem. This report should answer four basic questions:

1. What type of graph would you use to model the problem input (detailed in the Section 3.1), and how would you construct this graph? (I.e., what do the vertices, edges, etc., correspond to?) Be specific here; we discussed a number of different types of graphs in class.
2. What algorithm will you use to solve the problem? You should describe your algorithm in pseudocode. Your algorithm must be correct, and it must have the minimum possible complexity.
3. How can you be sure that your algorithm is correct? (I.e., provide a proof of correctness.)
4. What is the time complexity of your algorithm?

3 Coding your solution

In addition to the report, you should implement your algorithm in C++ or Java so that it can solve "Apollo and Diana" mazes. Your code may be in C++ or Java, but it must compile and run on the C4 Linux Lab machines.

Your code may be split into any number of files. In addition, you are allowed to make use of any built-in library, and C++ users may use the Boost library in their implementations. Boost is a free, open-source library with a rich collection of mathematical functions, including several that deal with graphs. You may read more about Boost at `www.boost.org`.

3.1 Input format

Your program should read its input from the file `input.txt`, in the following format. The input begins with a two positive integers 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 in the corresponding row and column in the maze (R or B), followed by a hyphen (-), followed 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 N, E, S, W, NE, SE, SW, and NW represent "north," "east," "south," "west," "northeast," "southeast," "southwest," and "northwest," respectively. The goal square (bulls-eye) is represented by a letter O. You may assume that the bulls-eye will always be in the bottom-right corner of the maze.

For "Apollo and Diana" maze, the input is:

```
 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   O
```

3.2 Output format

Your program should write its output to the file `output.txt`, in the following format. The output should consist of a path from the top left square to the bottom right square (bulls-eye). It should be 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 space in between. The direction should be represented using N, E, S, W, NE, SE, SW, and NW, as in the input. Of course, the sequence of moves described in the file must solve the maze described in the input.

For example, if your first three moves take you 3 spaces east, 3 spaces southwest, and 4 spaces southeast, your output should begin as follows:

```
3E 3SW 4SE
```

You are welcome to try figuring out the solution to the "Apollo and Diana" puzzle on your own, but that won't get you any points. Your assignment is to model the maze as a graph and to solve the problem using an appropriate graph algorithm.

4 Submission

You must submit a zip archive containing 1) your report (described in Section 2) as a PDF document, 2) your code (described in Section 3), and 3) a README file describing how to compile and run your code to Canvas. If your code requires more than a simple command to compile and run then you must also provide a Makefile and/or shell script. A simple command might be something like:

```
g++ *.cpp -o maze
```

If you are using Boost in your solution, you must provide a Makefile and/or shell script that uses the environment variable `$BOOST_HOME` (pointing to the Boost installation directory) to compile your code.

As this is an *individual* project, your project report and code will be checked for plagiarism.

5 Grading

Report	**50 points**
Graph model	20
Algorithm pseudocode	15
Proof of correctness	10
Complexity	5
Code	**50 points**
README file	5
Follows input and output specs	10
Compiles and is correct	30
Good coding style	5

Note: if your code is unreasonably slow, you will lose points for both your algorithm design and your correct output grade.