Homework 3 Writeup

Many real-world problems can be represented abstractly through a graph. Solving these problems usually comes in two parts: The first step is to think about the problem as a graph problem (you saw some examples in the lecture slides). The next step usually involves finding a graph algorithm that can help solve the problem. In this homework, you will cover some applications of graph traversals.

Part 1: Understanding

You will be using 2 files:

1) Graph.java
2) Maze.java

1) Graph.java is an implementation of an undirected graph, similar to the one seen in the lecture slides. You will be adding a function to it that we'll go over later.

2) Maze.java is a file that you will use in order to solve a maze. It reads in a text file that represents a maze (explained in more detail below), and will use the graph class to solve the maze by traversing the graph.

Part 2: Mazes

In this part of the homework, you will implement a maze solver. The program will take as input a text file that represents a maze, and output a list of "moves" that can be taken to go from the start vertex to the end vertex.

Open up and read through Maze.java. Notice that the work of reading in the maze file is done for you. First, we convert the given text file into a 2d array. The next step is to create a graph from this 2d array. How can we represent our maze as a graph? To do that, you'll have to understand the format of the maze.

We've chosen to represent our maze as a grid, where each spot in the grid can have 4 values:

0 - this represents a "wall" block in the maze that cannot be travelled in
1 - this is a valid spot that can be taken in the maze
2 - this represents the starting point of the maze
3 - this represents the end point of the maze.

For example, a maze text file may look like this:

```
1 0 1 1 1 0 3
1 0 0 0 1 0 1
1 0 1 1 1 0 1
```

```
1 1 1 0 1 1 1
1 0 1 2 0 0 0
1 0 0 1 1 1 1
1 0 1 1 0 0 1
```

The above is a valid maze: there is a single path of 1s that can be taken to get from the 2 to 3. In general, you won't have to worry about invalid mazes. Note that you can only go in one of the four directions- no diagonals. In addition, you can always assume that the maze will be SQUARE. I.e., an nxn grid of numbers.

The path that can be taken to solve that maze should look like:

```
LEFT
UP
UP
RIGHT
RIGHT
DOWN
RIGHT
RIGHT
UP
UP
UP
```

The Maze class has 3 field variables:

g - the graph that will be used to represent the maze
startVertex - the starting point of the maze
endVertex - the ending point of the maze

You should start by finishing the maze constructor: create the maze graph (using the functions given to you in Graph.java), and identify the start and end vertices. A few hints:

1) How many vertices are in this graph?

2) A graph with k vertices will have vertices labelled from 1 to k. How will you choose to label the different blocks in the 2d array? How will you identify the start and end vertices?

3) When does it make sense to add an edge between two vertices in the graph?

After the graph has been created, we need to solve the maze by finding a path from the starting vertex to the ending vertex. You will output a list of moves that correspond to traversing this path, by writing the solveMaze() method.

The output of solveMaze() is a List of Moves (we've provided a Move enum for you in the Maze class, which is simply one of 4 directions). Thus, the actual output of your function in the example shown above would look something like:

```
        List<Direction> output = new LinkedList<>();
        output.add(Direction.LEFT);
        output.add(Direction.UP);
        output.add(Direction.UP);
        etc…
```

A few tips and ideas to get your started for this question:

1) You've seen two graph traversal methods. Put yourself in the place of someone walking through a maze. Will you attempt to explore all paths at once, slowly branching out? Or will you pick one path, see if it leads you to the end, and backtrack? Which one makes more sense when traversing a maze: depth-first-search, or breadth-first-search?

2) Make sure that you understand the two graph traversal algorithms in the lecture slides clearly before attempting to write them in code! There are animations of the two algorithms in the lecture slides.

3) When implementing a graph traversal, you'll need to make sure that you don't visit the same vertex twice. How can you accomplish this? It may be helpful to use an array of booleans that denote whether or not a given vertex has been visited.

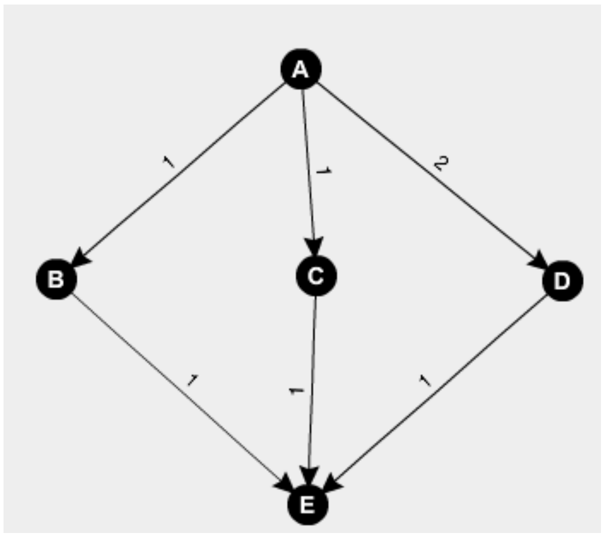
4) Graphs don't inherently have a sense of "direction", when going from one vertex to the next. The idea of moving RIGHT or DOWN when going from one vertex to another is specific to our maze class. How will you determine which direction must be travelled to go from a vertex u to vertex v?

5) How will you store this information during traversal?

Part 3: Shortest Paths problem

The next part of the homework is more theoretical. You will need to implement the method provided in the Graph class, numShortestPaths(int s, int t).

The problem is as follows. Given an undirected graph, and two vertices s and t, output the NUMBER of distinct shortest paths in the graph from s to t. Look at the example graph below:



From vertex a to vertex e, there are two shortest paths of length 2 and another path of length 3. The shortest path from a to e is of length 2, so there are a total of two shortest paths (the algorithm should output "2", given this graph)

Some tips:

1) Your algorithm must be efficient in order to get full credit! Even if you can't come up with the most efficient algorithm, try to figure out a less efficient solution for partial credit.

2) You've seen several graph traversal algorithms so far. Can any of them help you with finding shortest paths in a graph?

3) Try thinking about some of the programming paradigms you've seen in the course- namely, greedy and dynamic programming. You may need to keep track of some extra information as your traverse the graph in order to remember the number of shortest paths. Are there any strategies that can help you here? Can you put some of these ideas together?