CIT 596 - Programming 2

This programming homework deals with the creation and solution of mazes using some of the graph theoretic algorithms seen this semester. A lot of the code comes from a similar assignment handed out at the University of Washington.

The HW should be done in Java.

You do not have to stick to the provided design. If you want to design this yourself, that is totally fine and will not cost you any points.

You are allowed to work on this in a group of 2. Please indicate very clearly who you are working with. Only one of you needs to submit. Both of you get the same score.

Goal

The goal is to make a maze and then solve the created maze. A maze can be considered to be a graph in the following way. Each room in the maze is a vertex of the graph. A room is normally connected to a room to the north, the south, the east and the west. However, the presence of a wall between two rooms means that there is no edge between the vertices that represent those two rooms.

The maze creation part of the assignment is equivalent to making a spanning tree (details on this later) using a slightly modified version of traditional Kruskal's algorithm.

Once the maze is created then you are required to solve the maze - we have given you one methods of maze solving - a random walker (this is a pretty slow way of solving a maze). You are required to complete a BFS and a DFS implementation to solve the maze.

Files provided

We have provided you with 4 files: Maze.java, MazeViewer.java, MazeCell.java and DisjointSet.java.

Since the emphasis with this assignment is to expose you to implementation of the algorithms, we have deliberately provided you with all the code required to create the UI.

However to get a basic understanding of how the code is organized we have deliberately deleted some lines of code and replaced them with TODOs. Your first step in this assignment will be to go through the code and figure out how the different classes are interacting with each other.

The main function is to be found in the MazeViewer file. The main function makes calls to generate a maze and then solve the maze. Both of those functions are in the Maze.java file.

To complete the assignment you will have to work your way through the TODOs. We strongly recommend that you use Eclipse's Tasks view and go through and finish these one by one.

While working through the TODOs should get you to the point where the assignment works, it is only a necessary step in accomplishing the assignment and might not be sufficient. Depending upon how you choose to implement certain portions of the assignment, you might have to write more code to complete it.

Design of the graph

The maze is being maintained as a graph which consists of vertices that are a connected in a grid like manner. You will realize that we do not actually maintain this graph as an adjacency matrix or an adjacency list. Since we know that we have neighbours only to the north, south, east and west, these neighbours are being maintained via instance variables with those exact names.

Every cell of the grid is considered to be a vertex (think of it as a room in a maze). To create a maze, we use an algorithm that knocks down walls in a manner such that there is one unique path from the start vertex (starting room) to the end vertex (the destination vertex of the maze).

Creating a maze using Kruskal's algorithm

A slightly modified version of Kruskal's algorithm can be used as follows to create a maze. Instead of creating a minimum spanning tree, this modified version treats all edges as equal.

```
While (number of  walls knocked down < Total Cells - 1)
    Choose random cell current
    Choose random neighbor of current that has a wall up between it and current
    If such a neighbor exists
       Find the labels of current and neighbour (using the find operation)
       If current and neighbour are in different sets
          union them
          knock down the wall
          increment number of walls knocked down
```

Disjoint set

As you have seen in lecture, the disjoint set data structure is important for any implementation of Kruskal's algorithm. You need to implement this in DisjointSet.java. Please implement this in the recommended manner with a parent pointer and doing union by rank and find with path compression (we will cover both of these on Monday, Feb 23).

Suggested approach to writing the code

Since completing this assignment requires you to write a few different pieces of code, here is a suggested approach to solving the assignment. This is merely a suggestion and you do not have to do things in this strict order.

1. Go through all the code outside of the MazeViewer.java file. As you go through the code make sure that you develop some understanding about what the functions are. While you should not have to change any of the code in this file, you will be making use of the methods and it is good to know what is available to you.

2. Go through the TODOs and make sure you fill out the really basic ones like the simple getters and setters. That should give you and idea of the class design.

3. The one class that you have to design completely by yourself is a disjoint set - DisjointSet.java. Implement the makeset, union and find methods. You will find that to do this you will also have to ensure that MazeCell is working.

4. Use the modified Kruskal's algorithm to create a maze. The modified algorithm is given to you in makeKruskalMaze is the method you want to be filling.

5. Once you have a maze maker, you should be able to run the code with command line arguments. That single command line argument can be 'random' and it will call into our random walker solver that uses the random walker to solve the maze. Obviously, this random walker solution is going to take a long time to solve the maze.

6. Write a breadth first solution to the maze. The room currently being visited should be marked in blue and the vertices that have been visited in the past should be gray.

7. Write a depth first solution to the maze. The room currently being visited should be marked in blue and the vertices that have been visited in the past should be gray.

Evaluation

The primary part that would be evaluated is your implementation of the DisjointSet data structure, the creation of the maze using Kruskal's algorithm, the DFS implementation, and the BFS implementation.

You are not required to write unit tests. Writing them might be useful though.

The overall weightage of this HW is 25 points. It is worth more than any of your other HWs.

What to submit

Submit all 4 files. You can choose to upload them individually on canvas. You can choose to upload a zip file. If you make any additional files, please make sure to submit those as well.