# Programming Assignment 4

Henry Z. Lo

Due: Tuesday, July 15 at midnight

## 1 Goals

This assignment will help you with the following:

1. Construct spanning trees for a graph-like structure.
2. Navigate a graph-like structure with traversal algorithms.

## 2 Overview

You will first write a spanning tree algorithm for constructing a maze. Then, you will use depth-first search to guide a traveller from one corner of the maze to another.

You will touch `Maze.java` and `Traveller.java`.

In `Maze.java`, you will need to implement the randomized Prim's algorithm. The algorithm is very similar to Prim's algorithm, but there are several modifications needed. The general idea:

1. Keep a collection of visited nodes, unvisited nodes, and candidate walls.
2. Pick a random node. Add it to the visited nodes, add its walls to the set of candidate walls.
3. Every other node is unvisited.
4. While there are unvisited nodes, do the following:
5. Pick a candidate wall randomly.
6. If it connects a visited node and unvisited node, mark the unvisited node as visited, add the wall to the edge set, and add the node's walls to the set.
7. Otherwise remove the wall from the candidate set.

Initially, the maze consists of a set of nodes, but no edges. You will need to implement randomized Prim's in the `makeEdges()` function to generate the walls of the maze (edges of the graph).

After doing this, implement the `travel()` function in the `Traveller` class. Use depth-first search to travel from the top left node to the bottom right node. Note that depending on where the traveler is, there is a limited set of legal moves.

Some functions of interest in `Maze`:

- `print()`: print out the entire `Maze`, showing all nodes, walls, and the position of the traveller, if marked.
- `getNode(int, int)`, `hasEdge(Node, Node)`: use these functions rather than manipulating internal representation of the graph directly.
- `addEdge(Node, Node)`: creates an edge (different from a wall!) between two nodes and stores it in the `Maze`.
- `go(Node)`: move the traveller if the node is valid, throws exception otherwise.
- `currentNode()`: returns current marked node (where the traveller is).
- `getMoves()`: returns an iterator for the nodes next to the current node.
- `atEnd()`: returns true if the traveller is at the bottom right corner.

There are also the subclasses `Node` and `Wall`, which have the following useful functions:

- `print()`: print out the entire `Maze`, showing all nodes, walls, and the position of the traveller, if marked.
- `Wall.getStartNode()`, `Wall.getEndNode()`: returns the `Node` at the start and end of the wall, respectively.
- `Node.walls()`: returns a `Collection` of `Wall` objects from this `Node`.

# 3 Instructions

1. Implement randomized Prim's algorithm in `Maze.makeEdges()` to create the walls of the maze.
    - Randomly select the starting position, and randomly select walls after that.
    - For every valid wall you select, add an edge to the `Maze` consisting of its connected nodes.
    - Use `print()` to visualize the state of your `Maze`.

- The resulting spanning tree should cover every node in the maze and not contain any cycles.

2. Implement depth-first search in `Traveller.travel()` to traverse the maze.
   - Use `getMoves()` to get the possible moves.
   - Call `go()` to move the traveller.
   - You will need to move backwards sometimes, or `go()` will throw an exception.
   - Be sure not to travel down the same path multiple times.
   - Use `print()` to visualize the movement of your `Traveller`.
   - The result should be a traversal from the top left to the bottom right.
3. Classes must compile.
4. Put all deliverables in your cs310/pa4 folder. Deliverables include:
   - Maze.java
   - Traveller.java
   - memo.txt
5. Answer the questions in section 4, put answers in memo.txt.

# 4 Questions

1. Pick 2 sizes of `Maze`s. Construct 2 `Maze`s, one of each size, for 4 in total. Show their construction step by step.
2. Show the action of the traveller for two of these `Maze`s.
3. It is possible to use breadth-first search for your traveller, but why is this a bad idea?
4. How can we apply Prim's algorithm to generate a maze instead of the randomized algorithm used here?
5. Describe how you could use breadth-first and depth-first traversal to *construct* a maze.

# 5 Extra Credit

You can make the traveller much more efficient by using the distance from the bottom right as a heuristic. In this case, instead of picking the next move randomly, you should pick the move which has the least heuristic value. For the heuristic, you can use either the Manhattan distance between the node and the destination, or the Euclidean distance.

If you do this correctly (you must use a heuristic function), you will get extra points. Traversal should be much faster. Email me if you have any questions.