DATA STRUCTURES HOMEWORK 5 Maze generation

1 Introduction

This assignment concerns the maze generation process that we discussed in class. Your job is to write a program that will generate mazes; in doing so, you will need to implement and use a *Union-Find* data structure. The goals of this assignment are to improve your understanding of how Union-Find works and to see a fun application of the data structure.

1.1 The maze generation algorithm

A maze is an $n \times n$ grid; each square in this grid is called a *cell* and the borders between cells are called walls. The placement of walls must satisfy the following properties:

- The maze must have one entrance and one exit (that is, all but two of the "external" walls that only border one cell must exist). For simplicity, we will require the entrance to be at the top left and the exit to be at the bottom right of the grid.
- There must be exactly one path from the entrance to the exit.
- Every cell must be reachable from every other cell.
- There cannot be any cycles (that is, there must be exactly one path from every cell to every other cell).

In order to generate a maze, we do the following:

- 1. Begin with a grid that has all possible walls present (except for the walls leading to "outside" for the entrance and exit).
- 2. While there exists a wall that is safe to remove:
 - (a) Choose such a wall at random.
 - (b) Remove the chosen wall from the grid.

We said in class that a wall is safe to remove if doing so would not cause there to be any cycles in the maze. In other words, a wall is safe to remove if the two cells adjacent to it are not yet connected by a path. We can keep track of which cells are connected by a path using a Union-Find data structure.

1.2 Setup

Begin by downloading some starter code in a zip archive:

```
https://bit.ly/AMHCS-2019F-211-hw5
```

Decompress this archive to a folder with four files:

- Cell. java: each Cell contains pointers to its four adjacent Walls, as well as pointers to a LLAddOnly head and a Cell next; these last two pointers are meant to help you implement your Union-Find.
- Wall. java: each Wall contains pointers to its two adjacent Cells (note that one of these might be null if the Wall is an "external" wall), as well as a boolean visible that can be set to true if the Wall is still part of the maze and false if the Wall has been removed from the maze.
- LLAddOnly.java: an abbreviated version of a linked list of Cell objects that only allows you to add to the list. This is meant to help you implement your Union-Find.
- MazeGenerator.java: some starter code to help you generate mazes. Specifically, the code that I have given you does the following:
 - 1. Sets up a new $n \times n$ 2-dimensional array¹ of Cells, and calls the initializeCells () method to initialize them.
 - 2. Calls the getWalls () method to create an array of Walls and link adjacent Cells and Walls. The array of Walls is returned.
 - 3. Calls the print () method to print out the maze—in which, at this point, all of the walls are visible.

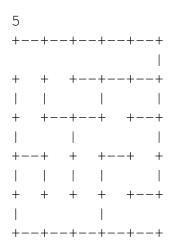
2 Your Tasks

Your job is to complete the program so that it generates mazes. Specifically:

- 1. Write a class called UnionFind that stores Cell objects. Your class should contain the following:
 - A method called makeset that takes a Cell as input and creates a new set containing one element, namely the input Cell.
 - A method called find that takes a Cell as input and returns a LLAddOnly that is the header of the set containing the input Cell.

¹Here n is an input parameter to the program. For example, if you run the program using the command <code>javaMazeGenerator 10</code>, this will create a 10×10 grid. If you don't specify an input parameter when running the program, a 5×5 grid is created by default.

- A method called union that takes two Cell objects as input and joins together the sets containing those Cells.
- 2. In the MazeGenerator class, write a method called createMaze that takes an array of Walls and a 2-dimensional array of Cells as input and produces a maze. Your method should not return anything; the Cells and Walls in the input arrays should contain all information needed to display the maze (which you can do by calling my print method).
- 3. Generate some mazes! You must submit a file called maze.txt that contains a value of n, followed by an $n \times n$ maze that your program generated. For example, your maze.txt file might contain:



The classes and methods that you write must have **exactly the names**, **parameters**, **and return types specified**, including capitalization. Note that you should not have to modify any of the provided code, except to fill in the createMaze method in MazeGenerator.

3 How to submit your work

Submit all of your *.java source code files, as well as your maze.txt file, using the CS submissions web page:

https://www.cs.amherst.edu/submit

This assignment is due on Wednesday, Dec-04, 11:59 pm.