# CE204 Assignment 2

David Richerby

Faser deadline: Tuesday, 23rd March 2021 at 11:59:59am

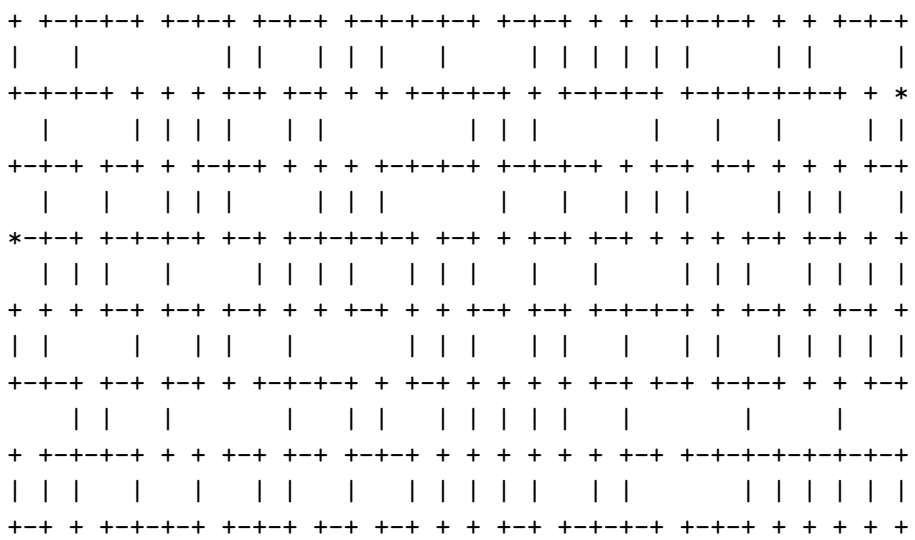
## Introduction

This assessed exercise constitutes 10% of your grade for CE204. Answer all five parts.

You may not use any libraries except for the standard Java runtime. The work that you submit must be your own. It is an academic offence to submit somebody else's work as your own. If you are unsure what this means, please consult the University's guidance or ask me.

## Generating mazes

This assignment is about generating random mazes like the one below. Note that the lines are the paths in the maze, not the spaces between them. The entrance and exit are the stars on the left and right sides.



The maze will be made by producing an undirected grid of edges with random weights and then computing a minimum spanning tree.

### 1. Getting started [15%]

Create a class called `Maze`. Its constructor should take integer arguments giving the width and height of the maze to be generated. It should create a `MatrixGraph` (see lecture 6, in unit 5) to store the graph and add the necessary weighted edges.

## 2. Printing [20%]

Add to `Maze` a method `void print()` that prints the graph to the screen, like the one above. Choose a random vertex on the left to be the entrance (marked with a star instead of plus) and a random one on the right to be the exit (ditto). If you print the graph made by `Maze`'s constructor, it should have a complete grid of edges.

## 3. Tracking graph components [20%]

In part 4 of the exercise, we will build up a spanning tree in stages. At each stage, the graph's components will be trees[1] and we'll need to track the components. Initially, each vertex will be its own component and, as we add edges to the graph, components will be merged.

We will pick a single vertex from each component to be the component's representative. We will store these in an array: for each vertex $x$, the array will store the representative vertex of $x$'s component.

i) Add an `int` array field to the `Maze` class.

ii) Add a method `void initializeComponents()` to `Maze` that initializes the array to the state in which each vertex is in a component on its own.

iii) Add a method `void mergeComponents(int x, int y)` to `Maze` that merges the components containing `x` and `y` by setting the representative of every vertex in `y`'s component to be the representative of `x`'s component.

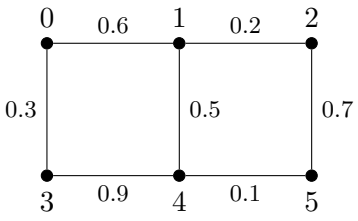
## 4. A spanning tree algorithm [40%]

The following algorithm to find a minimum spanning tree of a graph $G$ is an alternative to Prim and Kruskal's algorithms. First, initialize the component array, and initialize a graph $T$ with the same vertex set as $G$, to hold the spanning tree as we construct it. Now, repeat the following, until no edges are found:

- scan through all the edges of $G$ to find, for each component of $T$, the lowest-weight edge that has exactly one endpoint in that component;
- add all of these lowest-weight to the spanning tree;
- for each edge added, merge the components of its endpoints.

Note that we refer to components of $T$, the graph we are constructing, and which will eventually be the spanning tree. $G$, the original graph, will always have just one component.

For example, suppose we start with the following weighted graph $G$.

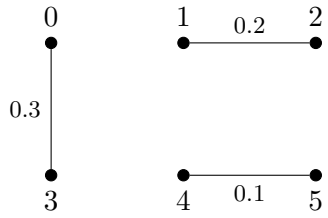



[1] Intuitively, such a graph is called a forest.

We initialize the spanning tree $T$ to be the graph with vertices 0–5 and no edges. Its components are $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$ and $\{5\}$. The least-weight edge in $G$ from each component of $T$ is:

| Component | Edge | Component | Edge |
|---|---|---|---|
| $\{0\}$ | $(0, 3, 0.3)$ | $\{3\}$ | $(0, 3, 0.3)$ |
| $\{1\}$ | $(1, 2, 0.2)$ | $\{4\}$ | $(4, 5, 0.1)$ |
| $\{2\}$ | $(1, 2, 0.2)$ | $\{5\}$ | $(4, 5, 0.1)$ |

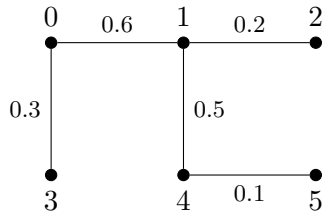
We add these to the spanning tree, to get




and merge the components of $T$ to get $\{0, 3\}$, $\{1, 2\}$ and $\{4, 5\}$. Now, we repeat. The least-weight edges in $G$ from each component of the new $T$ are:

| Component | Edge |
|---|---|
| $\{0, 3\}$ | $(0, 1, 0.6)$ |
| $\{1, 2\}$ | $(1, 4, 0.5)$ |
| $\{4, 5\}$ | $(1, 4, 0.5)$ |

We add these to the spanning tree, giving




Merging the components of $T$ gives the single component containing all the vertices. There are no edges in $G$ between components, so we are done.

Write a method `void spanningTree()` in `Maze` which implements the algorithm described above and replaces the graph stored in `Maze` with one the holds the minimum spanning tree.

### 5. Putting it all together [5%]

Write a `main` method in `Maze` that creates a `Maze` object with width and height 10, computes the minimum spanning tree of the graph and prints it on the screen.

## Submission

Please submit *only* the file `Maze.java` to Faser. Please do not submit a zip file. If you have written extra classes as part of your solution, please include them in the same file as the main class. You do not need to include the source of the `MatrixGraph` class.