Algorithms & Data Structures: Lab 14

week of 4th February 2019

1 Setup

1.1 Saving your work from last week

By now you should be familiar with the operations needed to save your work. Make sure you commit your work to version control often, and always have a backup copy, ideally remotely (for example in your own account on the department's gitlab installation.)

1.2 Downloading this week's distribution

Once you have successfully saved your changes from last week, you can get my updates by doing

```
git pull
```

which *should* automatically merge in new content. After the `git pull` command, you should have a new directory containing this week's material (named `14/`) alongside your existing directories.

2 Graphs

This lab requires you to implement undirected graphs, along with algorithms that work on those undirected graphs.

2.1 Implementing the class

I have provided you with skeleton implementations of `Vertex` and `Edge` classes, as well as the `Graph` class. The first part of the lab is to flesh out those implementations, so that the basic operations on graphs work:

- `addVertex()` and `getVertex()`, which add and retrieve vertices from a graph by name;
- `addEdge()` and `getEdge()`, which respectively create an edge in a graph (with a given weight) and retrieve an edge from a graph. Your implementations of these two methods must consistently implement **undirected** graph semantics;

Implement these methods, and verify that the tests provided for these basic operations pass.

2.2 Minimum spanning trees

You must implement two methods regarding minimum spanning trees; when doing so, you are permitted to assume that the graph is connected and that the edge weights are all distinct:

- `MSTCost()`, which should return the total cost of the minimum spanning tree of the graph;
- `MST()`, which should return a fresh `Graph` that represents the minimum spanning tree, but shares no structure (vertices or edges) with the original graph.

Implement these, using algorithms presented in lectures or any other technique, and verify that the corresponding tests pass.

2.3 Shortest path

You must implement methods to compute the cost and the edge sequence of the shortest path between two vertices. You should **first** write tests for and then implement the following methods:

- `SPCost(String from, String to)` / `SPCost(std::string from, std::string to)`, which should return the total cost of the shortest path between `from` and `to`;
- `SP(String from, String to)` / `SP(std::string from, std::string to)`, which should return a fresh `Graph` that represents the shortest path, but shares no structure (vertices or edges) with the original graph.

In writing your tests, you may wish to take inspiration from the provided minimum spanning tree tests.

3 Binary heap submission

Submit your work on implementing binary heaps to the lab submission system by **16:00** on **Friday 8th February**; as usual, you may submit more than once between now and then, and your highest score is retained.