Laboratory 7 – Week 8

## Graph Traversal - MST

## 7.1 Introduction

**Firstly, this worksheet is one of the worksheets from which your laboratory worksheets portfolio of work will be assessed.**

This laboratory involves experimenting and testing with the *Minimum Spanning Tree* (MST) algorithm. You will be required to write a program that tests this algorithm on different sized datasets. You will also be required to create plots of the algorithm's performance. Make sure you save any programs and results.

## 7.2 Exercise 1: Preliminaries

Familiarise yourself with the lecture entitled “7.1 Classic Algorithms - Graph Traversal”. Pay particular attention to the section on the *Minimum Spanning Tree* algorithm and how to represent graphs in a computer program.

Create a project and associated class called Lab7. Extract the embedded objects in Appendix A (three classes) and include them into your project. These three classes are used to implement a *Minimum Spanning Tree* algorithm, specifically *Prim's Algorithm*.

Examine the Java code. What does each class do?

Class EDGE: Store the edges, from node i to j with weigh w.

Class COMPAREEDGE: Compare the edge’s weight and it is used to sort an ArrayList of edges

Class MST: sort the algorithm following the Minimum Spanning Tree.

You call the MST code as follows:

**double** mst[][] = MST.*PrimsMST*(g);

Where g is a weighted undirected graph in matrix form (two dimensional array) and mst is the *Minimum Spanning Tree* representing that graph. Both g and mst will be of the same size.

Imagine we have the following graph:

|  |  |
| --- | --- |
|  |  |

We have three nodes. We represent this as a three by three matrix (*G*) shown next to the graph above.

Here, the edge between each node *i* and *j* is represented by a number that indicates the edge weight. In the diagram, the edge between node 1 and 3 can be seen at row 1, column 3 and has value 2. Since the graph is an undirected graph each *gij* = *gji*, i.e. the matrix *G* is symmetric, i.e. *GT=G*, hence row 3, column 1 is also 2, representing the edge in the opposite direction. Note that we represent the absence of a link/edge by a zero, in the example above it can be seen that there are no edges between node 1 and itself (a zero at 1,1) , node 2 and itself (a zero at 2,2) and node 3 and itself (a zero at 3,3).

We can create the matrix as follows:

**double** g[][] = {{0,1,2},{1,0,3},{2,3,0}};

Here we create a 3 by 3 **double** array (matrix) and specify the contents on a row by row basis. We can then create the *Minimum Spanning Tree* as follows:

**double** mst[][] = MST.*PrimsMST*(g);

In order to see that the results you need to write a method PrintArray that displays to the screen the matrix as above. Test it by displaying g and then mst (after you have created the *Minimum Spanning Tree*). Verify the results are as expected:

## 7.3 Exercise 2: A Larger Example

Now try the MST algorithm on the example from the lecture:

|  |  |
| --- | --- |
|  |  |

The first diagram is the graph we want to create the MST from, and the second graph is the expected MST.

Again verify your results using the PrintArray method.

## 7.4 Exercise 3: Experimental Analysis

We are now going to time how well the MST algorithm scales according to input size.

Do the following:

1. Create a method RandomArray that takes in as parameter an integer *n* and returns an *n* by *n* double array where each element is a random **natural (integer ≥ 0)** number between zero and 100 inclusive. This will create a random graph. **It is vital that you make sure that the matrix is symmetric (see above)**.You should be able to reuse some of the code written for previous worksheets…
2. Verify that the method produces the expected results, and that the matrix is symmetric.
3. Conduct a set of experiments where you generate a number of random graphs varying in size from 100 nodes to 500 nodes, and then time how long the MST takes to run. Repeat each of these experiments a number of times, and then average the results.
4. Plot a graph of the average run time against the size of the graph. An example (similar) graph can be seen in the *Sorting Algorithms* worksheet.

## 7.5 MST Applications

Have a look at some of the application papers on the use of *Minimum Spanning Trees*:

[Detecting Actin Fibers in Cell Images using Minimal Spanning Trees](http://www.ri.cmu.edu/publication_view.html?pub_id=308)

[CfA redshift survey](http://iopscience.iop.org/1538-4357/553/2/L115/005808.text.html)

[Clustering Gene Expression Data](http://bioinformatics.oxfordjournals.org/content/18/4/536.full.pdf)

Finally the following link contains a discussion of [Minimum Spanning Trees](http://www.ams.org/samplings/feature-column/fcarc-trees).

## 7.6 Appendix A

The following classes contain the code you will need for this exercise sheet.

|  |
| --- |
|  |