Lab Exercise: Implementing a Graph algorithm

In this exercise you will learn how a graph can be implemented. You will extend an existing implementation of a node and a graph toolkit to include an implementation of Dijkstra's algorithm (and others, perhaps). By doing this, you will gain an insight into the details of implementing high-level algorithms.

By purpose, this exercise is defined loosely. You will have to evaluate and make the critical decisions yourselves. Team up and discuss.

Exercise 1: Study the handout

A handout for this exercise, `WeightedDirectedGraphHandout.zip`, is available from BB. This comprises three header files and one source file:

`GraphNode.h`	Implementation of a graph node
`Edge.h`	Implementation of a weighted, directed graph edge
`GraphToolkit.h`	Skeleton implementation of a graph toolkit
`__TESTER__.cpp`	A simple test of the above files.

Investigate these three files and explain the following to the person next to you:

1. What member variables does a graph node contain?
2. What is the purpose of each member variable in the graph node?
3. Why is the edge declared as a stand-alone class, and not just as a pointer as in the linked list or the binary tree?
4. What is the purpose of the type definitions (typedef's) in the top of the graph toolkit?
5. What is the purpose of each of the methods in the graph toolkit

Exercise 2: Implement Dijkstra's algorithm

Using the handout as a starting point, implement Dijkstra's algorithm. You will need to make additions to the Node class and the toolkit class. Test your implementation by defining the same graph as in the slides for Dijkstra's algorithm and check the result of executing your implementation of Dijkstra's algorithm on it.