Programming Project 2: Routing Algorithms

This project includes two parts. For both parts, please use the same programming language (either C/C++, Python, or Java) to implement your algorithms.

For each, submit source code (.java, .py or .c file in plain ASCII text), and a readme file (plain ASCII text, PDF or plain HTML) that provides details on how to compile and run your code. It must be possible to compile on one of the common platforms.

The two parts are as follows. For further details, refer to lecture slides or any standard graph theory text. Assume that the inputs are undirected graphs (representing bi-directed networks), as in the examples in the lecture slides.

Part 1: write a program to implement the standard Dijkstra's algorithm for finding shortest paths from a given source node to a given destination node in graphs.

Part 2: write a program to find the shortest pair of edge-disjoint paths in a graph, from a given source node to a given destination node. The slides on possible algorithms for this part are given in the Moodle.

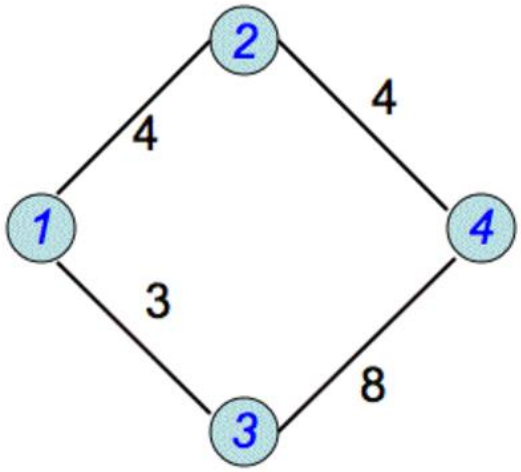
The programming assignment is due on 12/1/2016 9:30am. Submit your source files and readme files as a single zip file via Moodle2. No email submission will be accepted. Possible demonstration could be requested by TA. Late assignments are not accepted and will have a grade of zero.

Input Graph Instance Format

The graph instances will be supplied as plain ASCII text files, with the format described below. The first line of the graph contains a single number which gives the number N of vertices in the graph. The rest of the file is an adjacency list, one line per edge. In each line, there are four numbers, separated by spaces or tabs. The first two numbers are integers between 1 and Nand indicate source and destination vertices respectively. The third number is a real number indicating the cost of the edge. The fourth number is an integer specifying edge capacity - we do not need it for this programming project.

Note that this format can also represent directed graphs. But in this project, we are using the files to record undirected graphs, so all input files will specify an edge only once - indicating a bidirectional link.

For example, the following figure shows a graph and its file representation.




```
4
1 2 4.0000 192
1 3 3.0000 192
2 4 4.0000 96
3 4 8.0000 48
```

A slightly larger example of a graph with 10 vertices is given as an attachment (graph-10.txt). Your code should be able to take different graph files, source and destination pairs during testing.

Possible tests are as the follows:

Part 1: What is the shortest path from Node 1 to Node 6?

1->8->6 (cost:1.5)

Part 2: What is the shortest pair of link-disjoint paths from Node 1 to Node 6?

1->8>6 (cost: 1.5)

1->10->8->5->6 (cost:11)