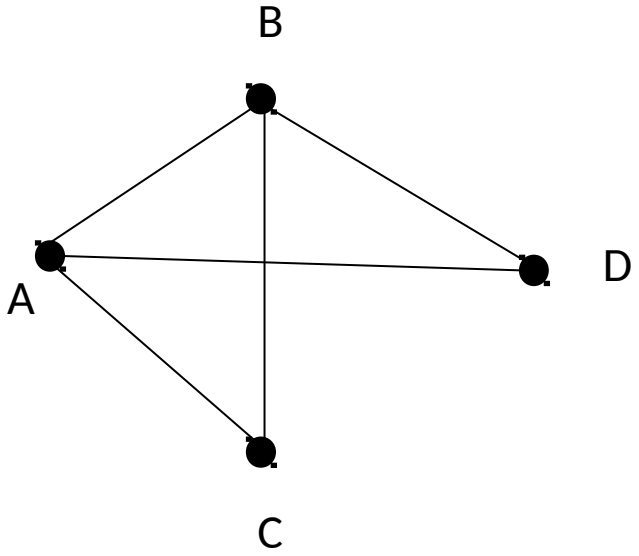
CS420 Computer Communication and Networks

Assignment 7
Assigned: 11/16/15
Due: 12/3/15 11:59 PM

In this assignment, you will implement the Djikstra's Shortest Path algorithm to find a minimum spanning rooted at a particular node. The minimum spanning tree should then be used to compute the routing table of that node.

Program Input

Your program should read and manipulate the graph represented as an adjacency list. An adjacency list contains the list of nodes in the graph, and the edges incident on each node. For e.g., for the graph shown below, the adjacency list would be listed as follows:



A	A→B	A→C	A→D
B	B→A	B→C	B→D
C	C→A	C→B	
D	D→A	D→B	

Note that this is an undirected graph, and hence each edge appears twice in the list. For e.g., A→B and B→A represent the same edge in the graph. Also, there is no edge weight included in this representation. For this assignment, your program should read in the graph as a set of edges in which each edge between two nodes with edge weight *w* is represented by the following format:

$Node_1$—$Node_2$:w

The root node is an argument to the program, and the program should find the minimum spanning tree rooted at the specified node input to the program.

Java Classes (optional)

Three basic classes have been written for you, and may be used as the starting point for your assignment. The classes **Node.java** and **Edge.java** represent a node and an edge in your program. The class **AdjacencyList.java** represents the adjacency list as a Hash Map of nodes, and a list of edges incident on each node. A HashMap is a Java Collections object to store a (*key*, *value*) pair such that a *value* object can be obtained by referring to the *key* value corresponding to that object. For e.g., (WIU ID, Student Name) can be a HashMap of storing student names identified by a unique WIU ID. Objects can be placed in the HashMap by the *put(Key, Value)* method, while retrieving the *get(Key)* method retrieves *Value* referenced by the *Key*.

Iterators are a great way to run through the objects stored in a HashMap. You can use the *for-each* construct or an *Iterator* to loop through the objects. The following examples show how you can use the for-each and iterator to loop through a Map m:

```
// For-each
for (KeyType key : m.keySet())
    System.out.println(key);

// Iterator
for (Iterator<Type> it = m.keySet().iterator(); it.hasNext(); )
    KeyType key = it.next();
```

The printAdjacency() method in AdjacencyList.java is a good example how this is done. Since our adjacency list `adjacency` is a HashMap of (Node, List<Edge>), we would need to retrieve the Edge list for each Node, and print the Edge nodes, and weights.

```
public void printAdjacency()
{
    for (Node node : adjacency.keySet())
    {
        List<Edge> list = adjacency.get(node);
        for (ListIterator<Edge> it = list.listIterator(); it.hasNext();)
        {
            Edge edge = it.next();
            System.out.print(edge.start.getAddr() + "-" +
edge.end.getAddr() + "(" + edge.weight + ") ");
        }
        System.out.println();
    }
}
```

The outer loop retrieves the list of Nodes, while the inner loop retrieves the Edge list for that node.

If you wish to refresh your understanding of the Collections framework, follow this tutorial:

http://docs.oracle.com/javase/tutorial/collections/interfaces/index.html

The logic to read the input file, and create an adjacency list has already been done for you. You can use/modify this code as per your discretion. You can apply Djikstra's algorithm on the adjacency list object to find the shortest path tree rooted at a node specified by your program.

NOTE: You may ignore the classes provided, and design your program differently to compute the shortest path. For example, instead of an adjacency list, you might choose to use an adjacency matrix. In such cases, ensure that you strictly follow the specifications as far as the input goes, and implement your logic differently than that given in the provided class files.

Specifications

Write a Java program **ShortestPathFinderApp.java** that reads the input graph as a text file, and finds the Minimum Spanning Tree rooted at the node specified as a run time argument. Your program should be invoked as

java ShortestPathFinderApp *input-file root-node*

where input-file is the name of a text-file that contains the weighted adjacency list and root-node is the starting point to the shortest path algorithm. Your program should compute the shortest path, and print the routing table of the root node represented as a three column table as follows:

Node Cost Next-Hop

where **Node** column is the set of destination nodes, **Cost** column is the total cost to reach those nodes, and the **Next-Hop** column is the next node to forward packets in order to reach that node.

What to submit

Upload your code on WesternOnline as well as toolman.wiu.edu. Use the *submit_cs420* program to submit your source code (ShortestPathFinderApp.java and others, if modified) only on toolman.

For this assignment, you should use the command

submit_cs420 –s hw7 ShortestPathFinderApp.java

to submit your program. To verify submission, use the command

submit_cs420 –l hw7