COMP 410 Fall 2018

Dijkstra's Single Source Shortest Path Algorithm

Summary

In this assignment you will use your basic graph data structure to solve an important graph problem. Write a Java program that implements Dijkstra's single source shortest path algorithm. The algorithm will use as input two things:

- a connected directed graph with weights on the edges (it may have cycles)
- a single vertex, the *start vertex*

For each vertex V in the graph, Dijkstra's algorithm finds the shortest path from the start vertex to V (including start vertex to itself, with path length 0).

What to Hand In

Instructions on how to prepare this for submission will appear in the Sakai assignment item.

Interface and Class Specifications

Class ShortestPathInfo

```
package DiGraph_A5;

public class ShortestPathInfo {
/*
 *
 * This class is to represent a single shortest path
 * from a source Node to a destination Node
 *
 * Description of each field you are to populate:
 *
 * String dest: the label of the destination node
 * long totalWeight: the sum of the edge weights on the shortest path
 *                   from source node to destination node
 *
 * Note on totalWeight: If a path from source node to the destination node
 *                      does not exist, set totalWeight to -1.
 *
 * Please leave this class untouched! You must populate your ShortestPathInfo object
 * using the given constructor
 * An array of populated ShortestPathInfo objects is to be returned by the
```

```
 * shortestPath method in your DiGraph
*/
  private String dest;
  private long totalWeight;

  public ShortestPathInfo(String dest, long totalWeight){
    this.dest=dest;
    this.totalWeight=totalWeight;
  }

  public String getDest() {
    return dest;
  }

  public long getTotalWeight() {
    return totalWeight;
  }

  public String toString(){
    return "dest: "+dest+"\ttotalWeight: "+totalWeight;
  }
}
```

Interface DiGraphInterface

```
package DiGraph_A5;

/**
 * COMP 410
 *
 * Make your class and its methods public!
 * Don't modify this file!
 *
 * Begin by creating a class that implements this interface.
 *
*/

public interface DiGraphInterface {
 /*
    Interface: A DIGRAPH will provide this collection of operations:

    addNode
      in: unique id number of the node (0 or greater)
          string for name
            you might want to generate the unique number automatically
            but this operation allows you to specify any integer
            both id number and label must be unique
      return: boolean
                returns false if node number is not unique, or less than 0
                returns false if label is not unique (or is null)
                returns true if node is successfully added
    addEdge
      in: unique id number for the new edge,
          label of source node,
          label of destination node,
          weight for new edge (use 1 by default)
          label for the new edge (allow null)
      return: boolean
                returns false if edge number is not unique or less than 0
                returns false if source node is not in graph
                returns false if destination node is not in graph
                returns false is there already is an edge between these 2 nodes
                returns true if edge is successfully added

    delNode
      in: string
            label for the node to remove
```

```
      out: boolean
             return false if the node does not exist
             return true if the node is found and successfully removed
    delEdge
      in: string label for source node
          string label for destination node
      out: boolean
             return false if the edge does not exist
             return true if the edge is found and successfully removed
    numNodes
      in: nothing
      return: integer 0 or greater
                reports how many nodes are in the graph
    numEdges
      in: nothing
      return: integer 0 or greater
                reports how many edges are in the graph

    shortestPath:
      in: string label for start vertex
      return: array of ShortestPathInfo objects (ShortestPathInfo)
              length of this array should be numNodes (as you will put in all shortest
              paths including from source to itself)
              See ShortestPathInfo class for what each field of this object should contain
  */

  // ADT operations

  boolean addNode(long idNum, String label);
  boolean addEdge(long idNum, String sLabel, String dLabel, long weight, String eLabel);
  boolean delNode(String label);
  boolean delEdge(String sLabel, String dLabel);
  long numNodes();
  long numEdges();
  ShortestPathInfo[] shortestPath(String label);
}
```

Class RunTests

```
package DiGraph_A5;
import gradingTools.comp410s19.assignment5.testcases.Assignment5Suite;

public class RunTests {
  public static void main(String[] args){ //runs Assignment 5 oracle tests
    Assignment5Suite.main(args);
  }
}
```

Sample Data

Here is an example:

```
     Vertices

       Raleigh, Durham, Pittsboro, Los_angeles, Graham,
       Cary, Chapel_hill, Hillsborough, Carrboro, Sanford

     Edges [ source --(weight)--> destination ]

       Raleigh     --(14)-->   Durham
```

```
  Durham      --(9)--->   Hillsborough
  Chapel_hill --(25)-->   Graham
  Chapel_hill --(1)--->   Carrboro
  Carrboro    --(32)-->   Cary
  Cary        --(3)--->   Raleigh
  Pittsboro   --(17)-->   Cary
  Pittsboro   --(15)-->   Sanford
  Sanford     --(3012)->  Los_angeles

Start Vertex: Pittsboro
```

Shortest Paths from start vertex Pittsboro to:

```
Raleigh: 20 (Raleigh, Cary, Pittsboro)
Durham: 34 (Durham, Raleigh, Cary, Pittsboro)
Hillsborough: 43 (Hillsborough, Durham, Raleigh, Cary, Pittsboro)
Chapel_hill: no path
Graham: no path
Carrboro: no path
Cary: 17 (Cary, Pittsboro)
Pittsboro: 0 (Pittsboro)
Sanford: 15 (Sanford, Pittsboro)
Los_angeles: 3027 (Los_angeles, Sanford, Pittsboro)
```

You can also use the examples from your text and check to see if you get the shortest paths given there.

JUnit Oracle Tests

JUnit oracle tests will be posted in Sakai about midway through the assignment period.

Miscellaneous Notes

- You may use the Java collections library for queues, stacks, lists, etc. that you need to program this solution.
- You will also need a priority queue (binary min heap) to make the algorithm efficient; you may use the one in the Java library, but I would encourage you to get wild and use your own code from the Assignment 3.