// Name : Hoshank Mali
// Id : 801254416

Introduction

Consider a data communication network that must route data packets (email, MP3 files, or video files, for example). Such a network consists of routers connected by physical cables or links. A router can act as a source, a destination, or a forwarder of data packets. We can model a network as a graph with each router corresponding to a vertex and the link or physical connection between two routers corresponding to a pair of directed edges between the vertices. A network that follows the OSPF (Open Shortest Path First) protocol routes packets using Dijkstra's shortest path algorithm. The criteria used to compute the weight corresponding to a link can include the time taken for data transmission, reliability of the link, transmission cost, and available bandwidth. Typically each router has a complete representation of the network graph and associated information available to it.
For the purposes of this project, each link has associated with it the transmission time taken for data to get from the vertex at one end to the vertex at the other end. You will compute the best path using the criterion of minimizing the total time taken for data to reach the destination. The shortest time path minimizes the sum of the transmission times of the links along the path. The network topology can change dynamically based on the state of the links and the routers.
For example, a link may go down when the corresponding cable is cut, and a vertex may go down when the corresponding router crashes. In addition to these transient changes, changes to a network occur when a link is added or removed.

Design and Execution

The code is in a single file - Graph.java . It consists of a class each for Vertex, Edge and Graph.

The Vertex class takes name, list of adjacent vertices, previous vertex, availability status, distance and visited status as attributes.

- Linked list of type Edge is used to store all the adjacent edges of a vertex.
- The status of a vertex gives the availability- if it is up or down.
- The distance of the vertex is used to compute the total cost needed to reach the vertex using Dijkstra's algorithm.
- The previous variable of type vertex is used to store the predecessor vertex when finding the shortest path. This is useful for printing the shortest path after running Dijkstra’s algorithm.
- The visited boolean variable is used to maintain if the vertex has been already visited during the BFS algorithm. This works similar to color attribute implementation for a vertex.

The Edge class has destination name, edge availability status and edge cost as attributes.

- The status is used to store whether an edge is available i.e if it is up or down.
- The cost is used to store the cost associated with every edge.

Graph class is the main class which does all the main functions such as building the graph by using add Edge, delete Edge, Edge down, Edge up, Vertex down, Vertex up. Also it prints the contents of the graph using the print function.

- Add Edge : Used to build the graph when the input file is read. It is also used for addedge query. If the edge already exists then only the cost of the edge is updated. It also calls getVertex method which returns the vertex details from a Vertices Hashmap and adds a new vertex if it does not exist.
- Delete Edge : Delete edge method is used when deleteedge query is inputted. It deletes an edge if it exists and returns an error message if the edge does not exist. It also returns an error message if the source or destination vertex does not exist.
- Edge Down : Edge down method is called when edgedown query is inputted. It sets the status of the edge to false.
- Edge Up : This method is called when edgeup query is inputted. It sets the status of the edge to true.
- Vertex Up : This method is called when vertexup query is inputted. It sets the status of the vertex to true.
- Vertex Down : This method is called when vertexdown query is inputted. It sets the status of the vertex to false.
- Dijkstra : The Graph class also has a method called dijkstras which is used to compute the shortest path from the given source to destination. It uses Dijkstra's algorithm to compute the shortest path between two vertices with time complexity of O(V+ElogV) with V being the no of vertices and E being the no of edges in the graph.
- Reachable : To print the reachable vertices, the Graph class uses a method called reachable. This method uses the breadth first search(BFS) algorithm to print all the reachable vertices from a given vertex. The BFS gives a time complexity of O(V+E) and we are performing it for V vertices. Hence the total time complexity is O(V(V+E)).
- Print : print method is used when the query print is inputted. It prints all the contents of the graph using treeset to alphabetically order the vertices.

Compiler Used

java 17.0.1 2021-10-19 LTS

What works and What fails:

- The queries mentioned in the project1.pdf work as expected given that the input queries are in the expected format.
- Works as expected on Windows OS.

Data Structure Design:

1. Priority Queue : Priority Queue based on minimum binary heap is used to perform Dijkstra’s algorithm and also used in BFS.
2. HashMap : used to store the vertex details.
3. TreeSet : TreeSet is used to store the vertex names in order to print them in alphabetical order.
4. Linked List : used to store all the adjacent vertices of a particular vertex.
5. ArrayList : Used arraylist to store visited vertices during Dijkstra’s algorithm.

References:

1. https://www.tutorialspoint.com/java/java_treeset_class.htm
2. https://stackoverflow.com/questions/683041/how-do-i-use-a-priorityqueue
3. https://www.geeksforgeeks.org/linkedlist-listiterator-method-in-java/
4. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm