Low Level Design Document

Table of Contents

Introduction

- This Software Application asks users for a network topology in a matrix format from a text file.
- It will process the network topology and build a connection table for each router, which describes a path from that router to other routers in a topology.
- The application will provide an option to modify a topology, the user can add a router and add an edge.
- Using the connection table from each router, it will build a graph, and by processing that graph it will provide all the shortest path from a source router to a destination router.

Software Interfaces

- This Software Application is built using Java 1.7, and for GUI implementation we used Swing.
- Operating System: Mac OsX, Windows 10.

Software Design Description

- Below we mentioned a different classes and methods used by them.
- Vertex.Java:

  Variables:
  - String vertexId: Id of a vertex/node/router.
  - Boolean up: Whether the vertex/node/router is up or not.

  Methods:
  - Vertex(String vertexId): It will set the vertexId for a vertex.
  - Vertex(String vertexId, boolean up): It will set both vertexId and Status up/down for a vertex.
  - Boolean isup() : return True if the router is up else returns False;
  - Void setup(boolean up): set whether the router is up or not.
  - String getvertexId(): It return the VertexId of a Vertex.
  - Void setvertexId(String vertexId): It sets the vertexId of a vertex.
  - Boolean equals(Object obj): checks whether two objects are equal or not and returns a True/False value.

- Edge.Java:

  Variables:
  - String edgeId: represents an Id of an edge.
  - Vertex Source: Source/start node of an edge.
  - Vertex Destination: Destination/end node of an edge.
  - Int weight: Weight/Cost of an edge.

  Methods:
  - Edge(String edgeId, Vertex source, Vertex destination, int weight) : It will set the edgeId of an edge, source of an edge, destination of an edge, and weight of an edge.
  - String getEdgeId() : returns an edgeId.
  - Void setEdgeId(String edgeId) : It sets an Id of an edge.
  - Vertex getSource() : returns a source of an edge.
  - setSource(Vertex source) : set a source/start of an edge.
  - Vertex getDestination() : returns a destination of an edge.
  - Void setDestination(Vertex destination) : set a end/destination of an edge.
  - Void setWeight(int Weight) : Set a cost/weight of an edge.
  - Int getWeight() : returns a weight of an edge.

- Graph.Java:

  Variables:
  - ArrayList<Vertex> vertexes: Stores vertexes of a graph.
  - ArrayList<Edge> edges: Stores edges of a graph.

  Methods:
  - Graph(ArrayList<Vertex> vertexes, ArrayList<Edge> edges): It will create a graph with given vertexs, and edges.
  - ArrayList<Vertex> getvertexes() : It will return a vertexes associated with a graph.
  - ArrayList<Vertex> getEdges() : It will return an edges associated with a graph.
  - Void setVertexes(ArrayList<Vertex> vertexes) : It sets a vertexes in a graph.
  - Void setEdges(ArrayList<Edge> edges): It sets an edges in a graph.

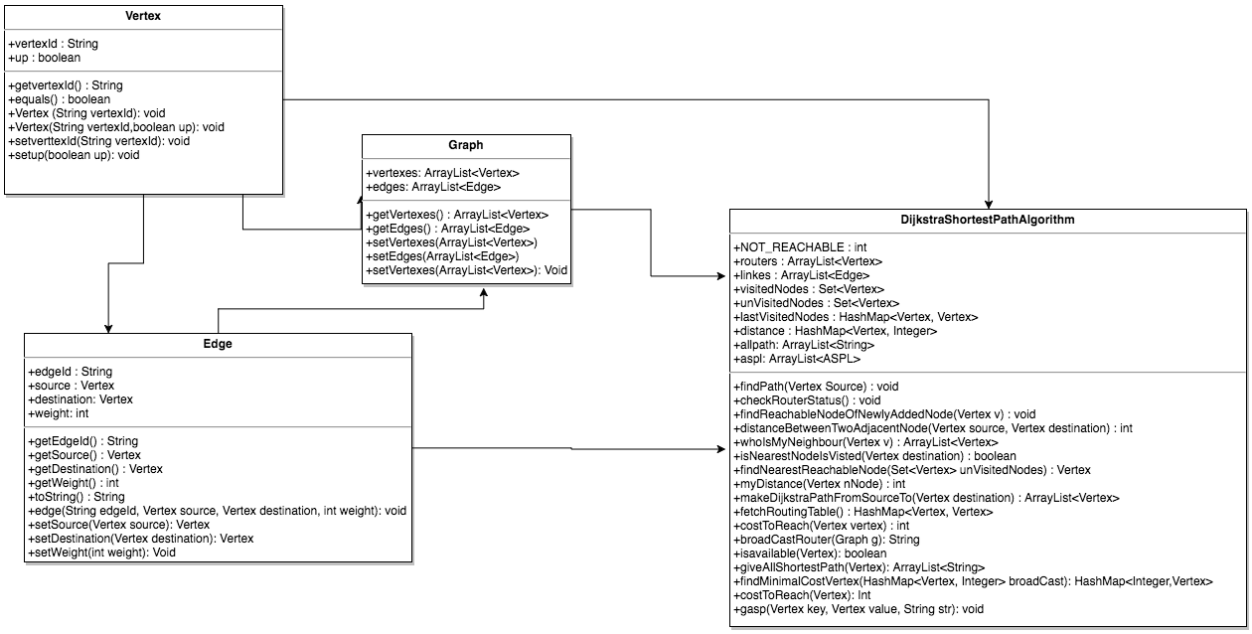
- DijkstraShortestpathAlgorithm.Java:
  Variables:
  - Int NOT_REACHABLE: It sets each router's default reachable distance to maximum integer value.

- ArrayList<Vertex> routers: This data structure stores all the nodes/routers of a graph.
- ArrayList<Edge> links: This data structure stores all the edges of a graph.
- Set<Vertex> visitedNodes: It stores the nodes/vertices that has been visited in a graph.
- Set<Vertex> unVisitedNodes: It stores the nodes/vertices that has not been visited in a graph.
- HashMap<Vertex, Vertex> lastVisitedNode: This data structure stores the information of lastVisitedNode in a path.
- HashMap<Vertex, Integer> distance: It stores the distance between two routers.
- ArrayList<String> allpath: It data structure will store all the paths between source and destination.
- ArrayList<ASPL> aspl: It stores the source and destination information along the all shortest path formation.

Methods:

- DijkstraShortestPathAlgorithm(Graph graph): It will take graph as an argument and initialize nodes and links according to the vertexs and edges of a graph.
- Void findPath(Vertex Source) : This method starts from a source node and tries to reach all the working nodes/routers.
- Void checkRouterStatus() : This method will add working router to the router data structure.
- Void findReachableNodeOfNewlyAddedNode(Vertex v) : This method will find reachable nodes of a newly added node in visited node. It will find minimum weight node from visited nodes to unvisited node and add node with minimum weight.
- ArrayList<Vertex> whoIsMyNeighbour(Vertex v): This method return an arraylist of directly reachable nodes from given node.
- Int distanceBetweenTwoAdjacentNode(Vertex source, Vertex destination): This method will return the weight between two adjacent nodes.
- Vertex findNearestReachableNode(Set<Vertex>, unVisitedNodes): This method will return a node with minimum weight from visited nodes.
- ArrayList<Vertex> makeDijkstraPathFromSourceTo(Vertex destination): This method will return the shortest path from source to destination.
- Boolean isNearesNodeIsVisited(Vertex destination): This method will return a true/false value based on whether the nearest node of the given node is visited or not.
- Int myDistance(Vertex nNode): This method return a node's distance from distance data structure, and if it is not yet inserted then it will return NOT_REACHABLE.

- HashMap<Vertex, Vertex> fetchRoutingTable(): This method will return the routing table of a router that call this method.
- String broadCastRouter(Graph g): This method will return a string which shows the broadcast router along with its cost based on a Graph argument.
- Void gasp(Vertex key, Vertex value, String str): This is a recursive method which will add all possible between source and destination to allPath variable.
- Boolean isAvailable(Vertex v): This method is for termination of a recursive function gasp(). It checks the availability of a router in a ASPL data structure.
- ArrayList<String> giveAllShortestPath(Vertex destination): This method will return all the available shortest path to the given destination from the source.
- HashMap<Integer,Vertex> findMinmalCostVertex(HashMap<Vertex, Integer> broadCast): This method will find minimal cost from broadcast router arraylist and return it.
- Int costToReach(Vertex vertex): This method will give the cost to reach destination router from the source router.



(Class Diagram)