CS542 Project Operations Manual

(Fall 2015)

Introduction

**Dijkstra's algorithm** is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest path tree.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined.

Dijkstra’s shortest path algorithm (also known as Link State Algorithm) is currently the most commonly used shortest path algorithm in GIS systems. Not only is it a popular shortest path algorithm in GIS systems, but also is also widely used in programs made for network modelling. One can find the shortest paths from a given source to all points in a graph in the same time; hence this problem is sometimes called the single-source shortest path problem.

In this we create a java project to implement the Link-State routing protocol. It is used to calculate the shortest path between any two given routers along with the direction. The programming language that has been used to implement this project is Java because of its object oriented design of the code, platform independence feature, rich set of API’s provided by the Swing library, and efficient portability.

Working of Dijkstra’s algorithm

This algorithm, generates a shortest path tree, with given node as the root or source node. Here, we generate two sets, one set contains vertices included in shortest path tree, and the other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has minimum distance from source node.

The various steps, which are involved in this algorithm are:

**1) Firstly,** a set spt (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated. Initially, this set is empty.

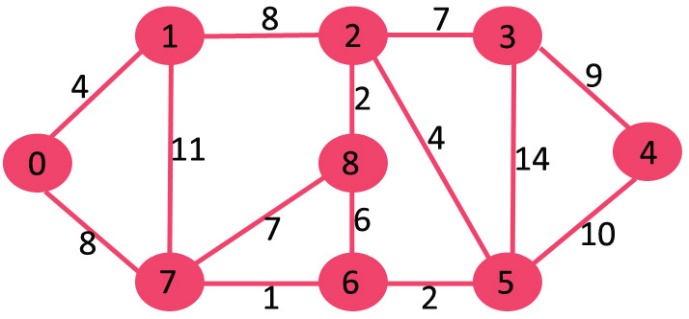
**2)** Secondly, Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source/root vertex so that it is selected first.

**3)** Repeat the steps mentioned below until all the vertices are present in the spt.

**a)** Pick a vertex ‘u’ which is not there in spt set and has minimum distance value.

**b)** Include ‘u’ to spt set.

**c)** Update distance value of all adjacent vertices of ‘u’. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex ‘v’, if sum of distance value of ‘u’ (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.



1. Here, the set spt is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite.
2. Here, we pick the vertex with minimum distance value. The vertex 0 is picked, and included it in spt set. So the set, becomes {0}. After that, update the distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. The vertices included in SPT are shown in green color.



1. Now, pick the vertex with minimum distance value and which is not present in the spt set. The vertex 1 is picked and added to spt set. So spt set now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.

[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/DIJ2.jpg)

1. Now again, pick the vertex with minimum distance value and not already included in spt set. Vertex 7 is picked. So spt set now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).

[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/DIJ3.jpg)

1. Now again, select the vertex with minimum distance value and not already included in spt set. Vertex 6 is picked. So spt set now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.

[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/DIJ4.jpg)

1. Here, now all the above steps are repeated again and again, till the minimum spanning tree (MST) is generated, with their respective shortest paths.



CODE DESIGN

Here, a single java file, called as the “networksclass” is taken. In this, initially the code is taken to launch the networks application, by using the void main class. Then, the class in initialized.

The code is then written in order to, initialize the frame contents. Here, the GUI is written, in which the buttons are created, and the corresponding event listeners are initialized, so that when the buttons are clicked by the users, the corresponding method is called.

Here, in that the code for “Network Topology” is written, in which the network topology file is loaded from the workspace, and the corresponding values are displayed to the users in the form of a matrix. This signifies the weight values between each and every node.

Then, the next part of the code is written for the “Connection Topology” in which the cost values between each and every nodes is displayed. This basically shows the interconnection between each and every node. It is also shown the node which is present next to that node.

In the next further part of the code, the user is asked to enter the source and destination router, for which the user wishes to find the shortest path between. If the user correctly specifies the router numbers, the shortest path is displayed, otherwise, a list of error messages are implemented or “error handling” is done, to make sure that the code runs properly, and the user corrects the incorrect information entered.

Then at the last, the Dijkstra’s algorithm is written, according to which the entire networking application works.

Link state routing protocol

**Link-state routing protocols** are one of the two main classes of routing protocols used in packet switching networks for computer communications, the other being distance-vector routing protocols. Examples are Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS).

The link-state protocol is performed by every *router* in the network. The basic concept of link-state routing is that every node constructs a *map* of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then calculates the next best logical *path* from it to every possible destination in the network. The collection of best paths will then form routing table of the node.

First step is to give a map of the network to every node. This is done with several subsidiary steps:

Determining the neighbors of each node

First, each node needs to determine what other ports it is connected to, over fully working links; using a *reachability protocol* which it runs periodically and separately with each of its directly connected neighbors.

### Distributing the information for the map

Next, each node periodically (and in case of connectivity changes) sends a short message, which is flooded throughout the network. Then, each node in the network remembers, for every other node in the network, the sequence number of the last message, it received from that node.

### Map Creation

Finally, with the complete set of link-state messages from every node in the network, each node produces the graph for the map of the network.

Second step is to calculate the values for the routing table.

### Calculating the shortest paths

A node here, maintains two data structures: a tree containing nodes which are "visited" already, and a list of *candidates*. The algorithm starts with both structures empty; it then adds to the first one the node itself. Then a & b steps are performed.

1. All neighbor nodes which are directly connected to the node are just added to the tree. Rest of the nodes are added to the candidate list.
2. Each node in the candidate list is compared to each of the nodes already in the tree. The candidate node which is closest to any of the nodes already in the tree is itself moved into the tree and attached to the appropriate neighbor node. When a node is moved from the candidate list into the tree, it is removed from the candidate list.

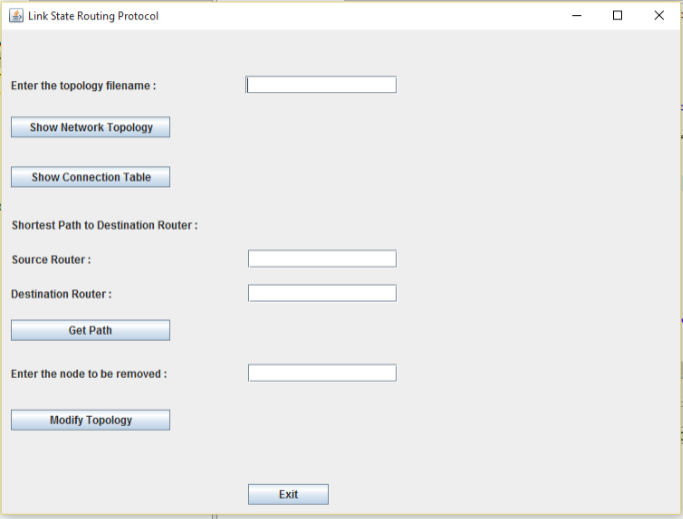
The above two steps are repeated as long as there are any nodes left in the candidate list. This procedure ends with the tree containing all the nodes in the network, The shortest path from that node to any other node is indicated by the list of nodes one traverses to get from the root of the tree, to the desired node in the tree.

### Routing Table Filling

### In order to create the routing table, it is only necessary to walk the tree, remembering the identity of the node at the head of each branch, and filling in the routing table entry for each node one comes across with that identity.

IMPLEMENTATION

1. Initial / Main Page



Here, the initial page of the networks application is displayed. It has various buttons, which implements different functionalities like-

1. Show Network Topology
2. Show Connection Table
3. Shortest Path to Destination Route :Get Path
4. Modify Topology
5. Exit

### Show Network Topology

### This option allows us to display the different interconnections between any two routers in the network. Here, the weight values between two routers are shown. Also, a value of “0” is used, to show connection between the same routers i.e. for the connection between R1 and R1, the weight value is initialized to 0. Also, the cost between two indirectly connected routers is initialized at -1.

### To implement this, the user is allowed to enter the file name into the application. Also, the file is pasted inside the workspace folder in order to be fully displayed. Then this filename is called using the code. However, if for some reason, the file can’t be displayed, then the “file not found” error message pops up.

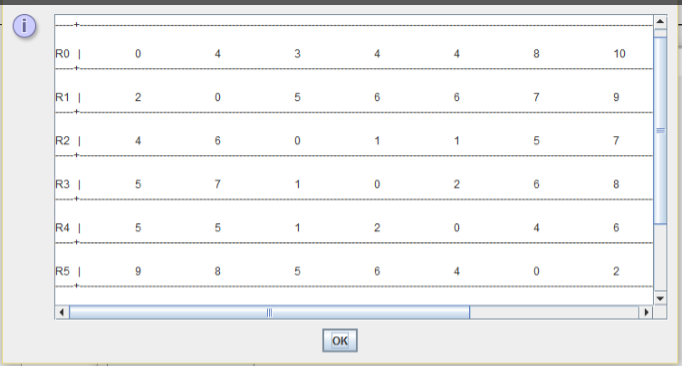
### C:\Users\verma\Desktop\Screenshots_CN\sa.PNG

### 

Here, the contents of the file”topo2.txt” is displayed.

1. Show Connection Path

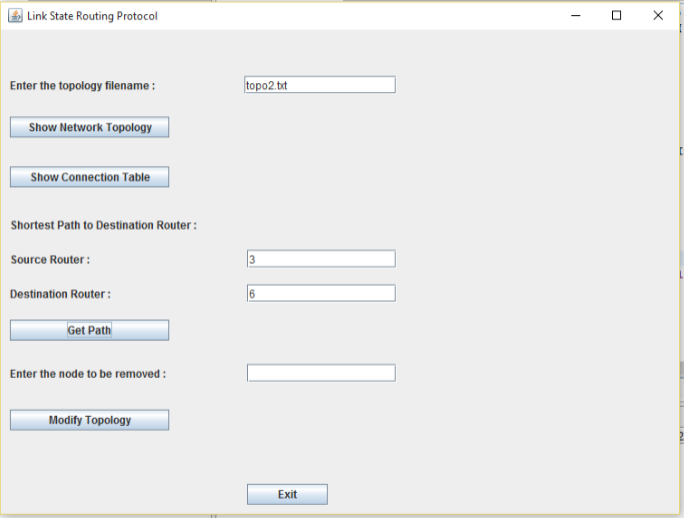
Here, using this option the Connection Path between two routers are shown. This basically helps us to know the interconnections between the source and destination router. It also helps us to know, the next router with respect to one of the routers. For instance, according to our connection topology, the node next to R2 is R3. Hence, this is shown here.



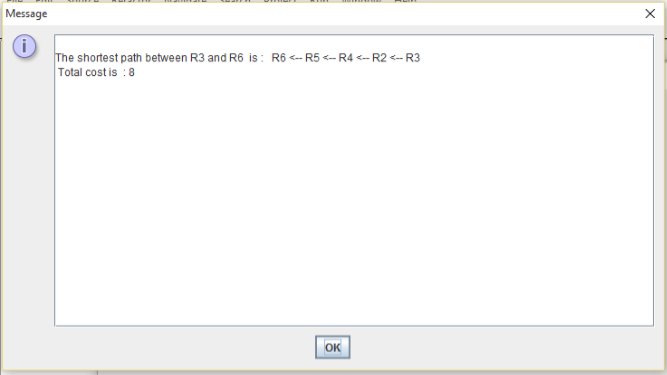
To implement this, the input file is read and displayed in the form of an “array”. Also, if the appropriate file is not found, then “file not found” error message is displayed.

1. Shortest Path between Destination and Source Router

This option allows us to, display the shortest path between source and destination router. Upon entering the source and destination router number, the “Get Path” button is pressed.

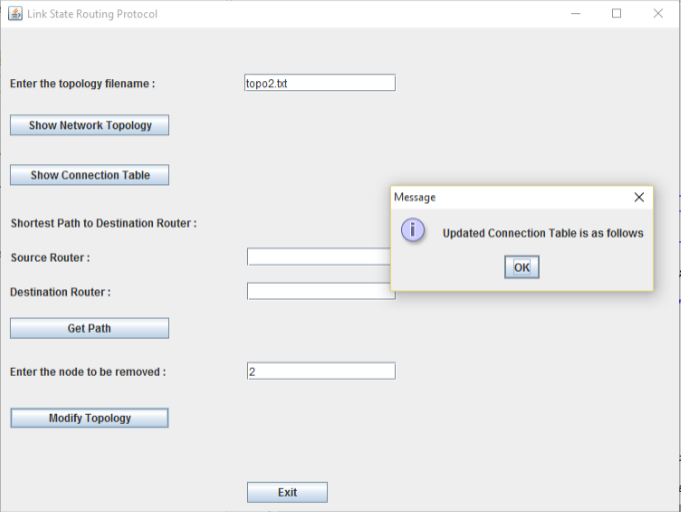


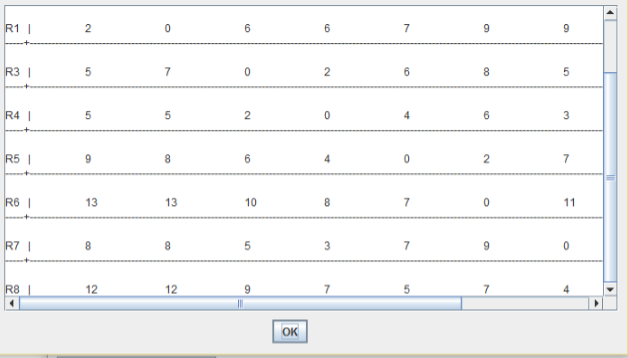
Hence, here after clicking “Get Path”, the shortest path between both the routers are displayed. Also, the overall cost between both the nodes is also displayed.



To implement this, contents of the input matrix are read, and displayed, which shows the shortest path.

1. Modify Topology

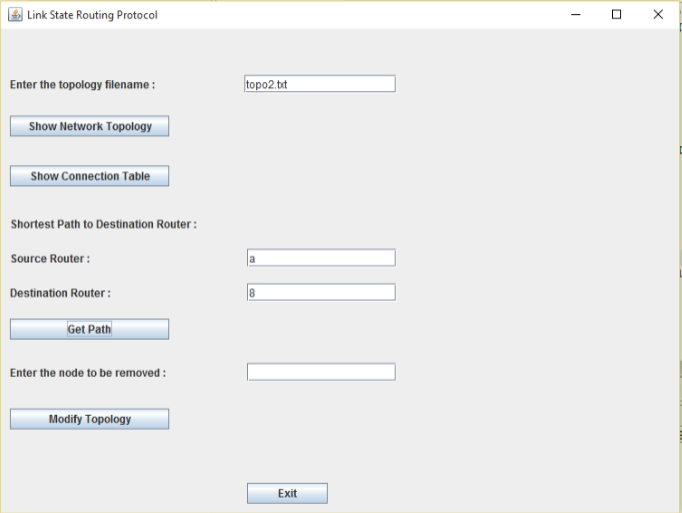


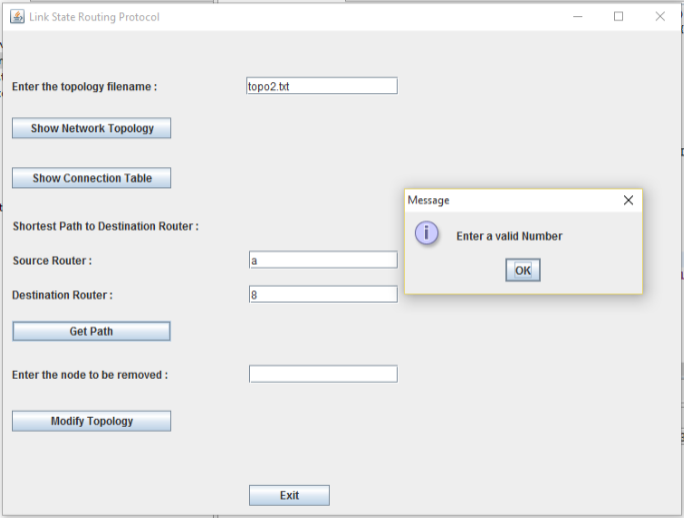


Here, the functionality is added in which, when the user clicks on “modify topology”, one of the nodes/routers gets deleted and the updated connection table is displayed.

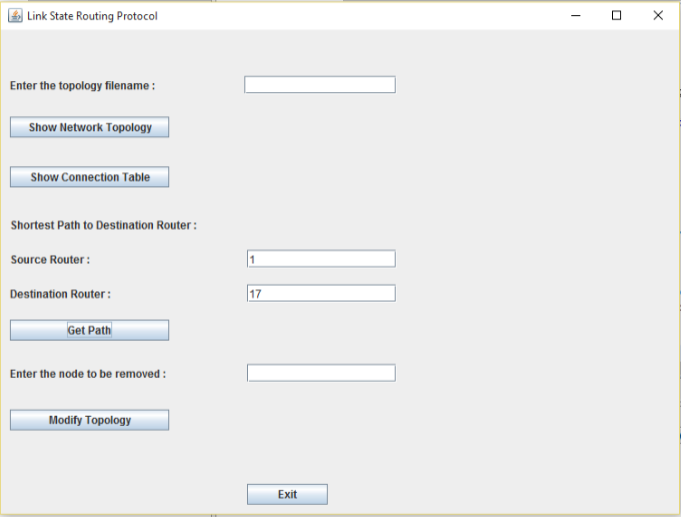
ERROR HANDLING

Here, the user has entered the source router as ”a” which is an invalid option. To overcome that, we have implemented the error handling feature, using which the “error message” i.e. a prompt is displayed to the user, to enter the correct value or number.

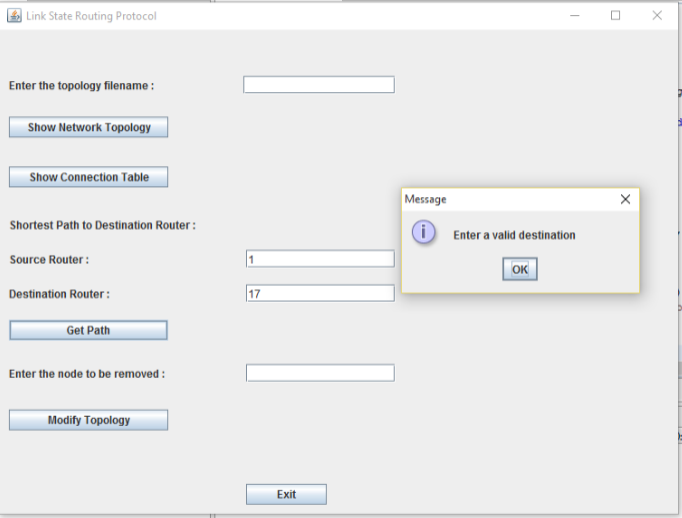




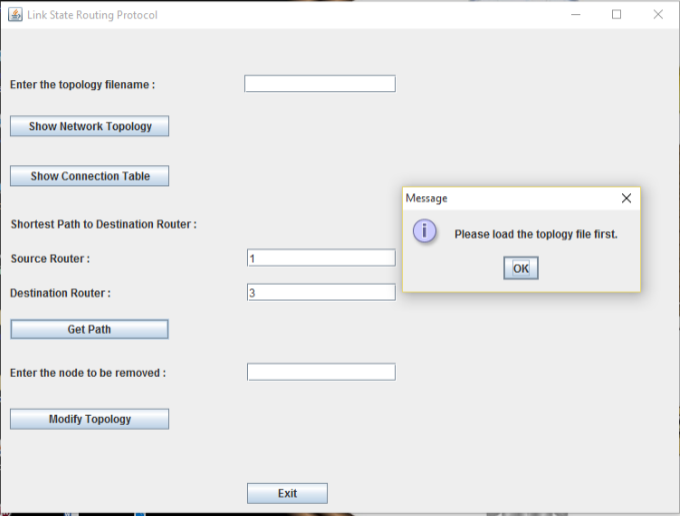
Now here, this kind of prompt is displayed to the user, in order to “enter a valid number”, so that the overall application works properly.



Here, the user has incorrectly entered the destination router. Hence, the dialog box is displayed for the user to enter correct destination number.



Hence, a prompt message like this is displayed to the user, to correct any previously entered incorrect data, in order to make sure that the code works perfectly.



Here, this is the option which is displayed when the user, tries to click the “Get Path” button, without clicking on the “Show Network Topology” button. This basically means that the topology file itself is not loaded, but the user is trying to retrieve it i.e. trying to display the path between two routers. Hence, the appropriate error message or prompt is displayed to the user.

EXECUTION STEPS

**IDE Used:** Eclipse Standard/SDK 64 bit version

**Version:** Mars Release (4.5.1)

**Java Version:** Java 8

**Operating System on PC:** Windows 10

To execute the project, follow the steps listed below:

1. Install Eclipse IDE. Launch eclipse in java perspective.
2. Create a local workspace.
3. From the File menu, import the project as a java project to the workspace, in order to load the entire java project.

1. Once the project is loaded with the respective packages and the classes, run the networksclass.java which is the single file created, in order to run the networks application.
2. Click on “run” command to execute the java project.
3. The GUI is displayed to the user in order to execute the application, where in the user should enter the file name present in “C:\file” path and click on “Show Network Topology” button. This will make sure that the topology file gets loaded properly.
4. Then the user should click on “Show Connection Topology”, to display the connection matrix between two nodes or routers.
5. Then the user, should enter the source and destination router number correctly, in order to find out the shortest path between them. If, any of the steps is not correctly executed then the corresponding error messages are displayed to the user.
6. Then the user can click on “Modify Topology”, if it user wants to delete a router from the network topology.
7. Close the project and eclipse IDE after successful execution.

WHY WE THINK OUR CODE WORK WORKS PERFECTLY?

We have come to the conclusion that our code runs perfectly fine, as we applied 2 different strategies, and both of these delivered positive results.

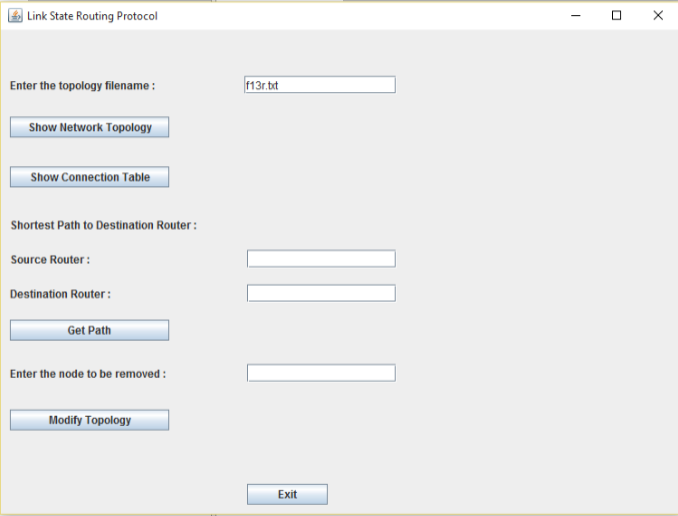
Strategy 1:

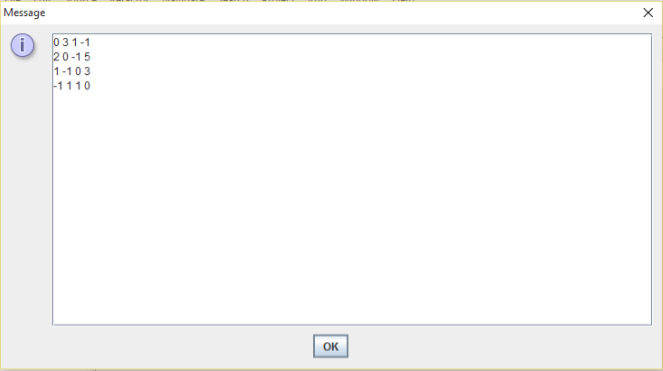
We did “dry” running of the code, with our sample input file. We manually calculated all the routing tables and shortest paths and checked if the results matched with the output of our code. The results did match. Strategy 1 was successful.

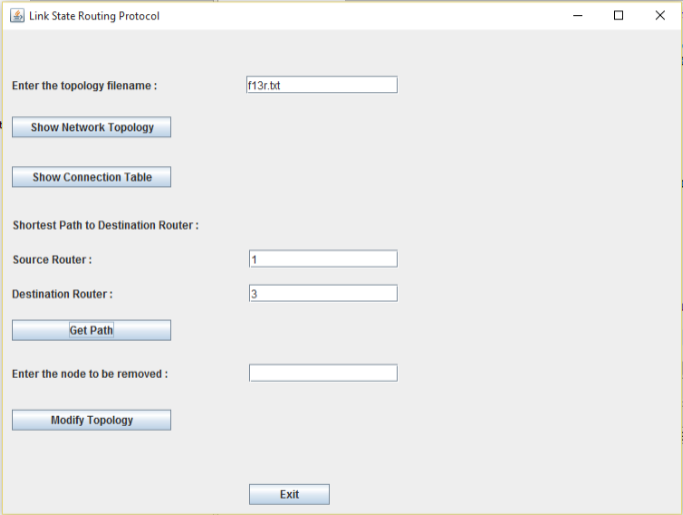
Strategy 2:

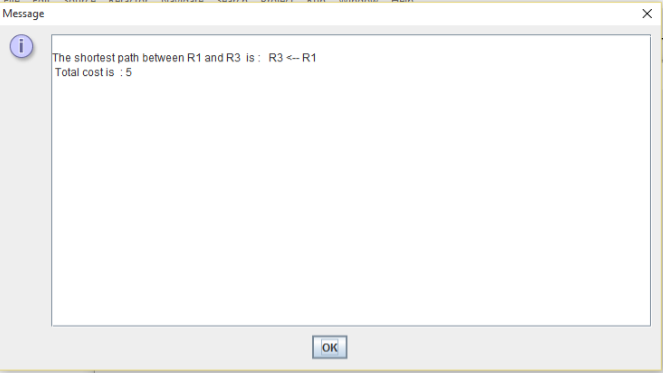
We then prepared 4 test files, and checked to see if the code worked perfectly arbitrary input data. The types of files taken and there testing results are as follows:

1. File with valid data in a valid format, having 4 routers.
2. File with valid data in a valid format, having 4 routers, but no interconnection is present between them.
3. File with weight values in 3 digits, having 5 routers.
4. File having different numbers of rows and columns.
5. File having all weight values as 0.
6. File with valid data in a valid format, having 4 routers



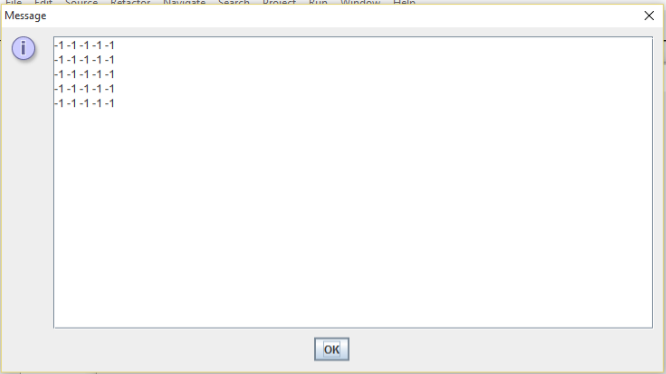


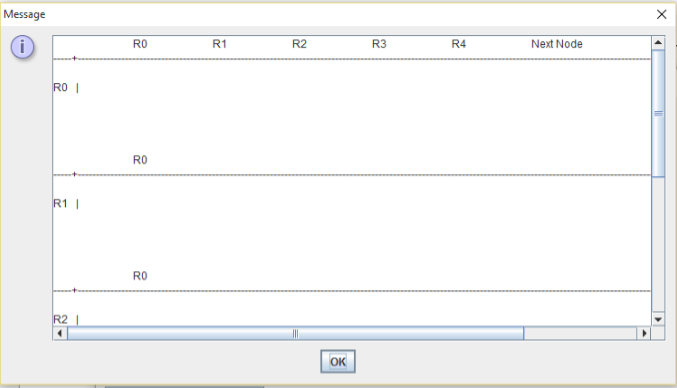


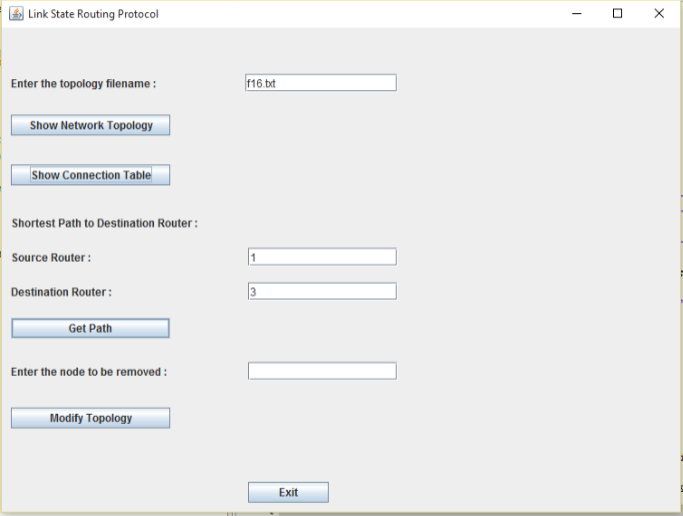


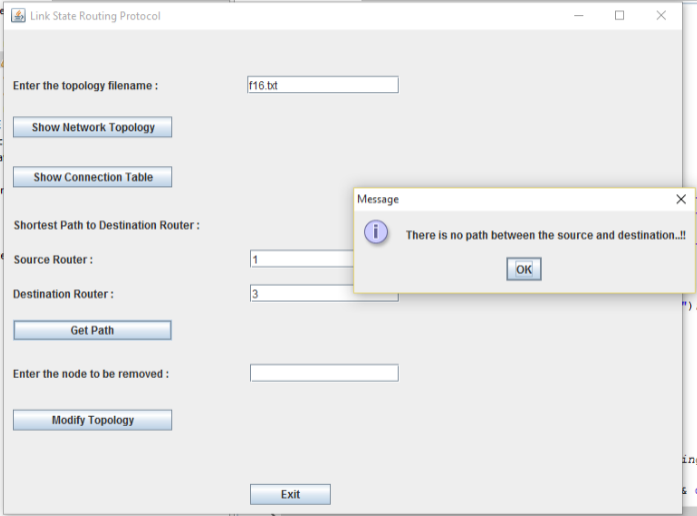
Here, an input file is taken wherein the file is present in valid format, but the number of routers are only 4. However, the code run was successful in this case.

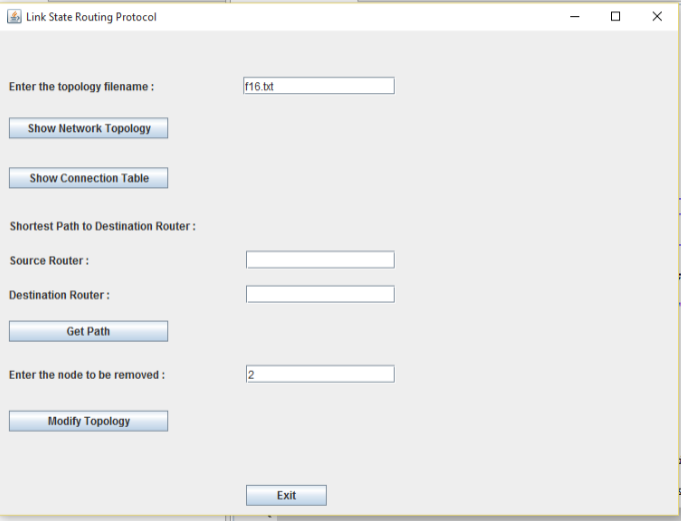
1. File with valid data in a valid format, having 4 routers, having no interconnection between them.

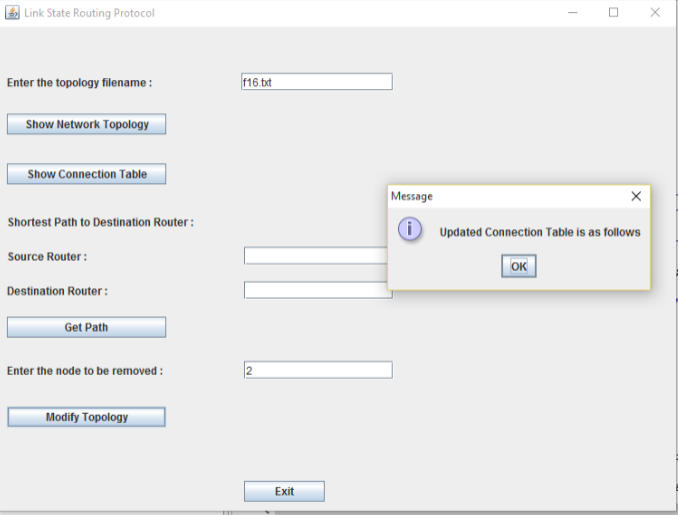


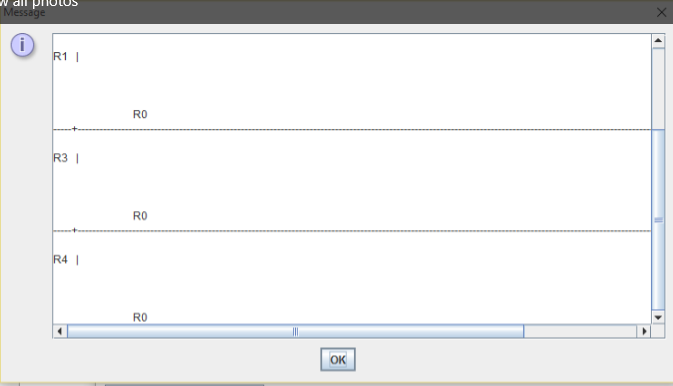






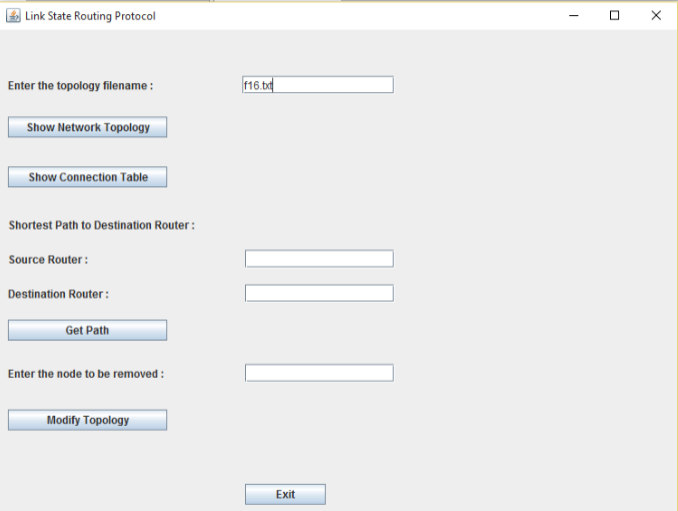


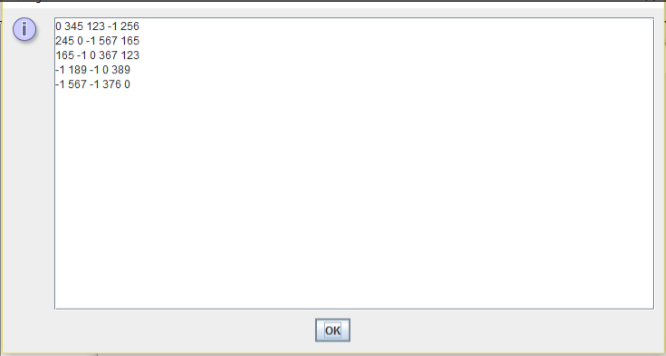


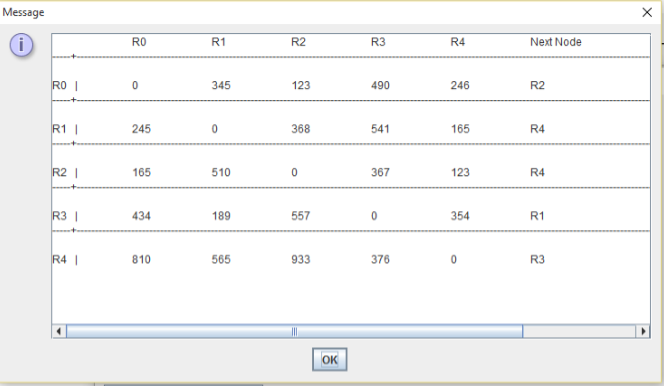


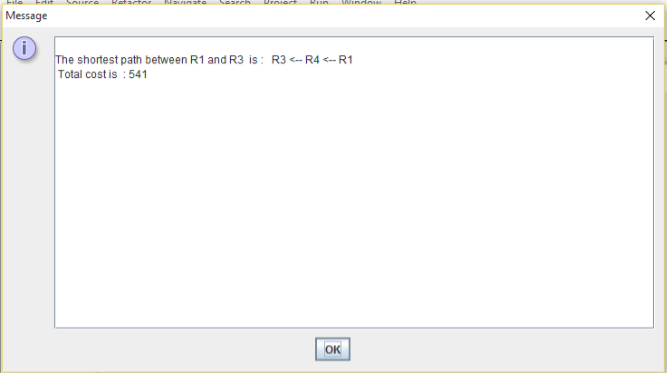
Here, an input file is taken, consisting of 4 routers, and having no interconnection between them. The code run was successful in this case.

1. File with weight values in 3 digits, having 5 routers



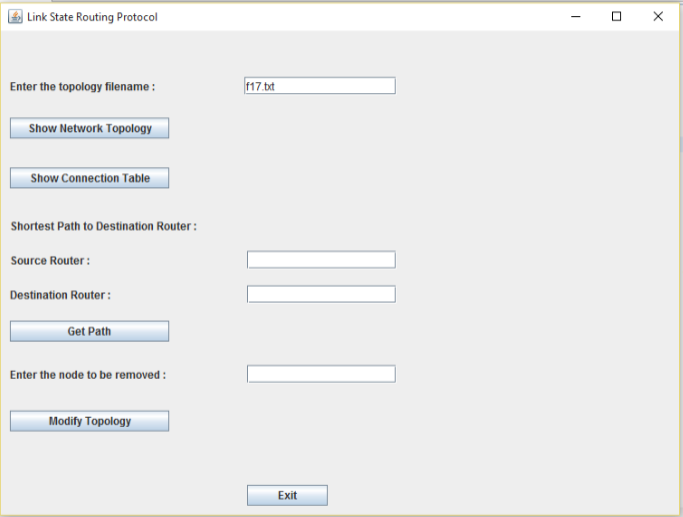


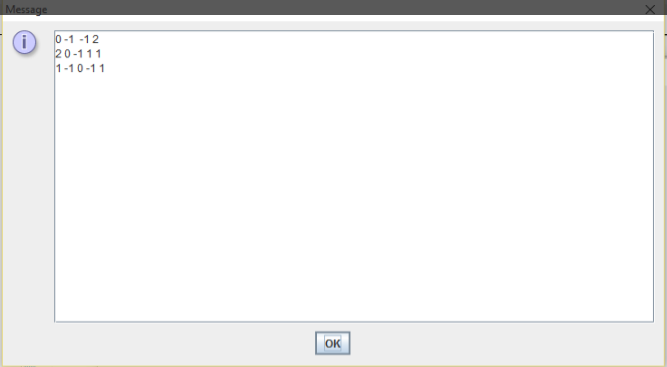


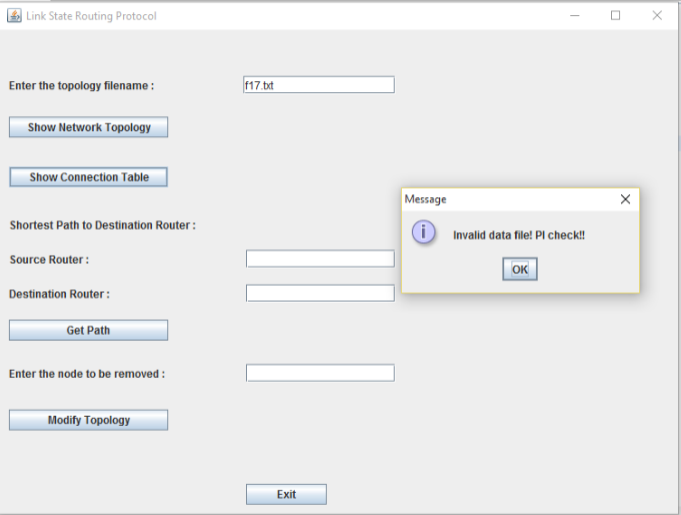


Here, an input file is taken in which the weight values are present in 3 digits, consisting of 5 routers. The code run was successful in this case.

1. File having different numbers of rows and columns.

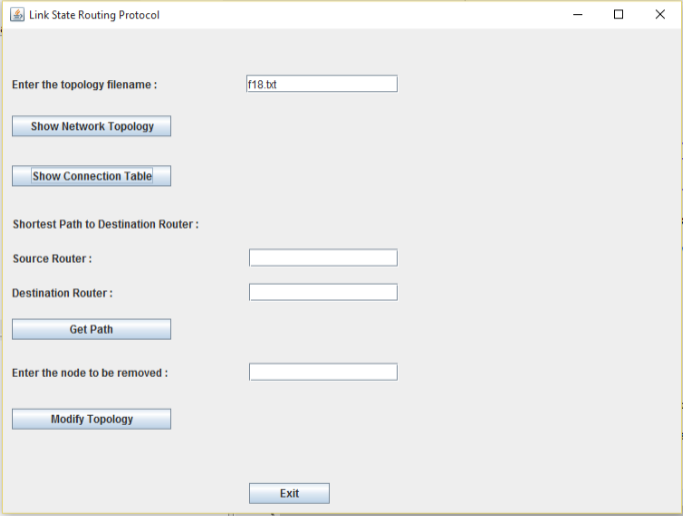


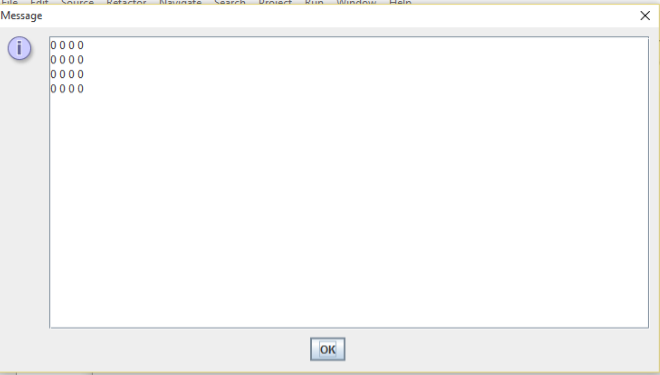


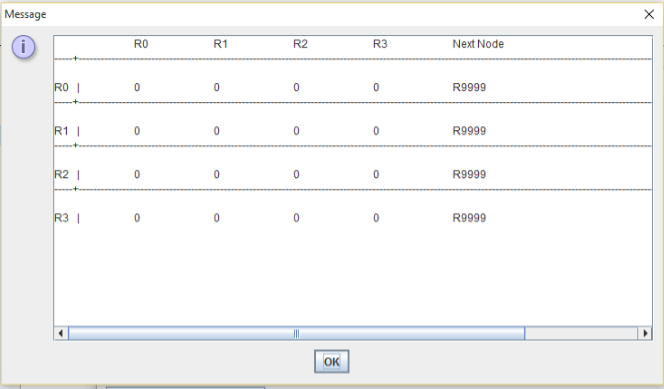


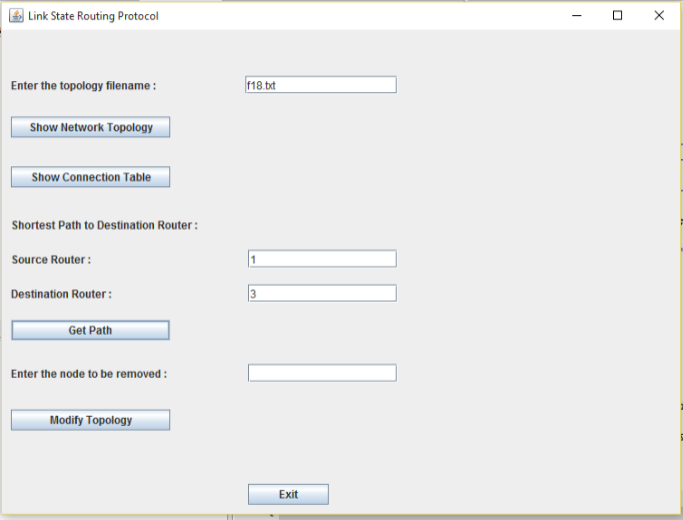
Here, an input file is taken in which the number of rows and columns are different. This means that the input file is not valid, hence a message dialog box is displayed, so that the proper file is given as input by the user.

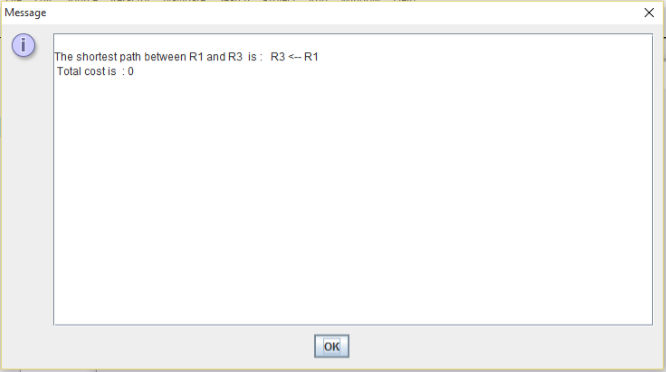
1. File having all weight values as 0.

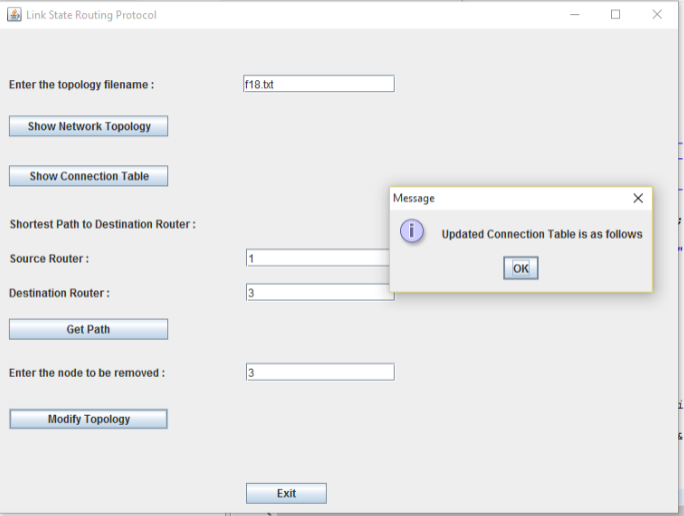


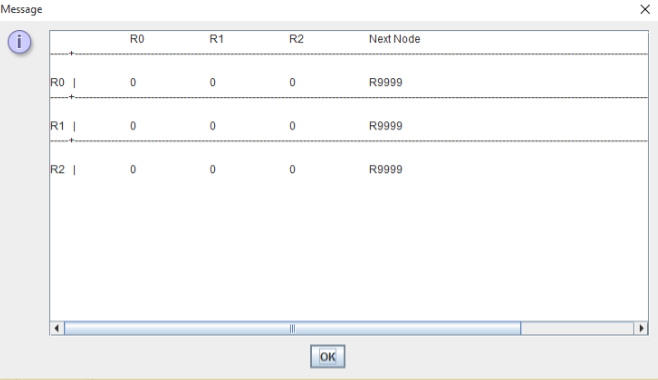












Here, an input file is taken in which the weight values are all 0. Also, the code run was successful in this case.

SCHEDULING AND RESOURCE ALLOCATION

**SCHEDULING**

Coding

10/21/2015-10/25/2015: Phase 1

10/26/2015-10/31/2015: Phase 2

11/1/2015-11/10/2015: Phase 3

Testing

11/11/2015- 11/17/2015

Documentation & PPT Preparation

11/18/2015-11/20/2015

**RESOURCE ALLOCATION**

Phase 1: Mansi

Phase 2: Pratishtha

Phase 3: Mansi

Testing: Pratishtha

Documentation (Project Operations Manual): Pratishtha

PPT Preparation: Mansi

REFERENCES

1. http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/
2. https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm
3. https://en.wikipedia.org/wiki/Link-state\_routing\_protocol
4. http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf
5. https://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html