

Data Structures and Algorithms

Semester 4 / 6 (2019/20)

**SCHOOL OF INFOCOMM TECHNOLOGY**

Diploma in Information Technology

Diploma in Information Security and Forensics

Diploma in Financial Informatics

**Assignment**

**Duration: 28** January to **10** February 2020 (Week 16 & 17)

**Weightage:** 20% of Module

**Individual/Team/Both:** Team of 2 Students

**Group Name:** Team Bees II

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| S10185580A | Seow Jia Wei Jerrald |
| S10187744E | Woo Weng Tai |

# Description of application

This application is developed for planning routes in the given metro dataset. It will be able to display all stations in a line, display relevant station information for a given station name, adding and saving of a new station, adding a new line into the dataset as well as finding the shortest route and its fare when given a starting station and a destination station.

# Roles and Contributions

Jerrald:

1. Storing of stations and routes into data structures
2. Implemented BST algorithm to display all stations in a line
3. Implemented Adjacency List and Queue algorithms to find and display a route and its price, given the source and destination stations
4. Implemented Dijkstra’s Algorithm to search for the shortest route and its price, given the source and destination stations

Weng Tai:

1. Creation of menu interface
2. File I/O to read and write from/to csv files
3. Display station information for a given station name
4. Add and save a new station on a given line
5. Adding of new lines

# Instructions

This is the Main Menu that will be shown when the user boots the program up. The user can enter the option of 1, 2, 3, 4, 5 or 0.

By entering 1, the user will be able to get all the stations of a line he/she enters. It will prompt the user what line he/she wants to see, and by entering a valid existing line, it will print out all the stations in the given line. For example, if the user enters “EW”, it will print out the stations in the EW line (refer to Figure 1).

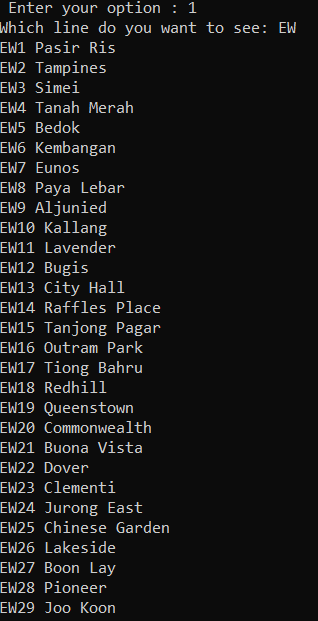


Figure : All stations in EW line

If the user enters 2 in the menu prompt, it will ask the user to enter a station name. If the station name exists, it will display the full station codes and whether it is an interchange. For example, when the user enters “Dhoby Ghaut”, it will display “NS4”, “CC1” and “NE6” (refer to Figure 2).

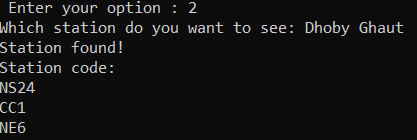


Figure : Station information of Dhoby Ghaut

If the user enters 3 in the menu prompt, the user will be able to add a new station. It will ask the user for a new station name and full station code. Then, it will prompt for the distance between the new station and the previous one (if it is located after a station) and/or the distance between the new station and the one after (if it is located before a station).

For example, if the user wants to add a “Tuas” station at EW30, it will only prompt for the distance between the new station and the previous one since it is located after the Joo Koon station and has no station after it (refer to Figure 3).

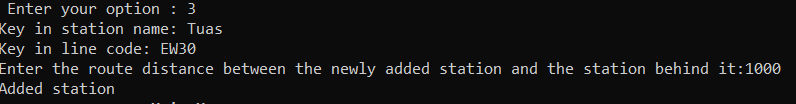


Figure : Adding station EW30 Tuas

If the user wants to add a “New Hill” station at DT4, it will prompt for both the distance between the new station and the previous one as well as the new station and the one after since it is located between Hillview and Beauty World (refer to Figure 4).

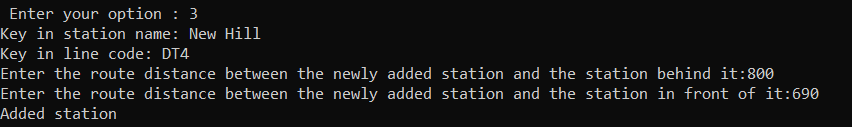


Figure : Adding station DT4: New Hill

The user can also add a new line if the line code the line they keyed in does not exist. For example, if the user adds a new station “NTU” at JE1, it will prompt the user whether they want to create a new line (refer to Figure 5).

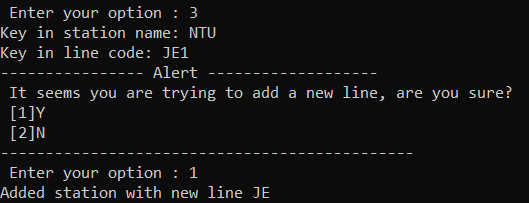


Figure : Adding of new line JE with station JE1: NTU

If the user enters option 4 in the menu prompt, it will calculate the *shortest route* and display the fare and train route when the user enters a starting and destination station. For example, if the user enters a source of “Ang Mo Kio” and a destination of “Beauty World”, it will display the route and train fare (refer to Figure 6).

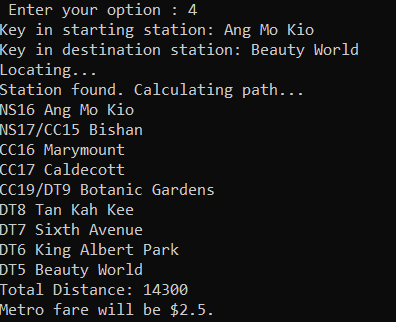


Figure : Shortest route of Ang Mo Kio to Beauty World

If the user enters option 5 in the menu prompt, it will save all the current data modified by the user in the current session into a csv file. It will update and save for both station and routes csv files.

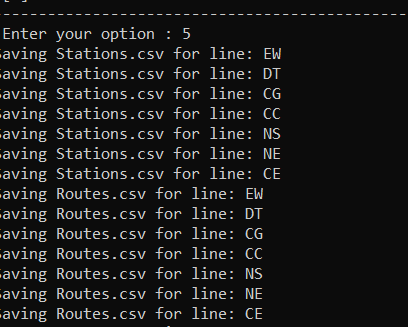
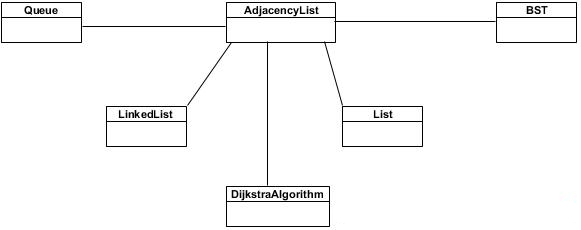


Figure 7: Saving function in progress

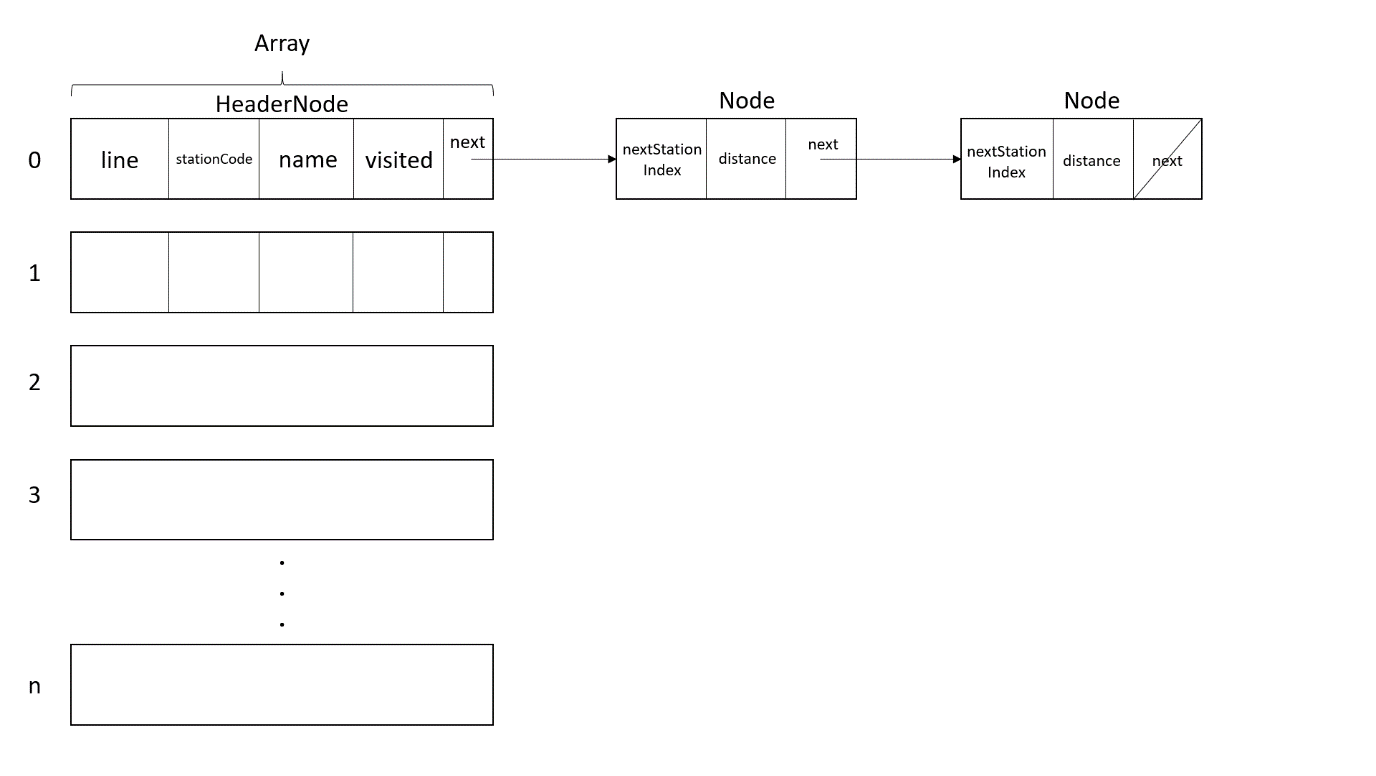
Lastly, if the user enters option 0 in the menu prompt, it will exit from the program.

# Class Diagram



# Description of Data Structures and Algorithms implemented

## 5.1 Adjacency List



The Adjacency List data structure consists of Header Node stored in an Array, and Node stored in a Linked List structure. The Adjacency List is used to store all the stations read from the Stations.csv and the path from Routes.csv.

The Header Node contains:

* line, which stores the line of the station
* stationCode, which stores the number associated with the station such as **1** for EW1
* name, which stores the name of the station
* visited, which is a Boolean that will be used for displaying of routes
* next, which links the HeaderNode to the Node

The Node contains:

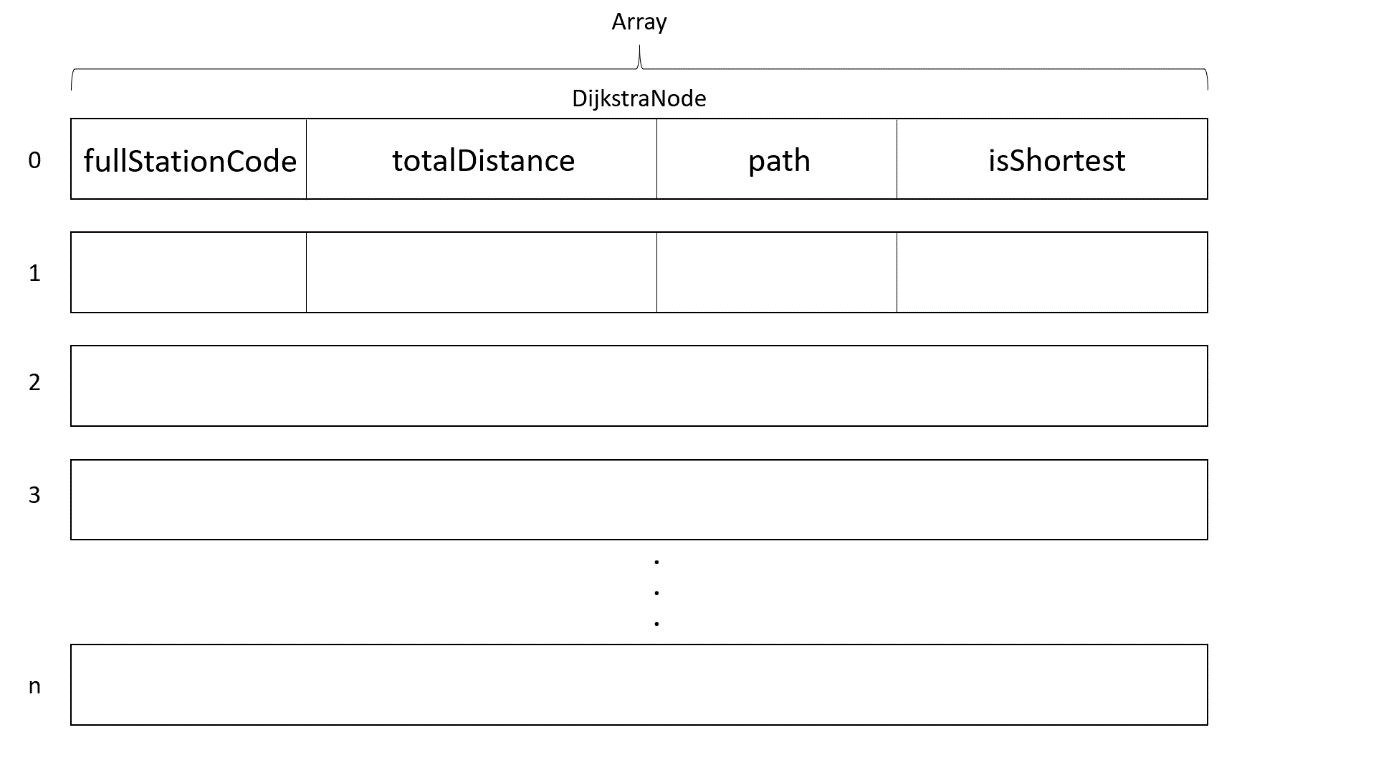
* nextStationIndex, which stores the Array index position of the station that is adjacent to the station of its Header Node
* distance, which stores the weight/distance between the Header Node’s station and the adjacent station
* next, which links to the next Node (other adjacent station to the Header Node’s station)

Note:

When an interchange station is added, it will be combined into one station. For example, a Jurong East Header Node will contain:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EW,NS | 24,1 | Jurong East | false | Node pointer |

## 5.2 Dijkstra’s Algorithm



In a DijkstraNode, it contains:

* fullStationCode, which stores the full station code of the station E.g. EW1
* totalDistance, which stores the total distance of the whole route
* path, which stores a Queue of the route
* isShortest, which is a Boolean that says whether the path is the shortest from the starting position to the station in fullStationCode

How does the algorithm work?

Let’s take for example a starting station of EW1 to a destination station of NS1. The function starts by creating a DijkstraNode that stores the starting station information(EW1, 0, [EW1ArrayIndex], true). The HeaderNode of EW1’s visited will be set to true.

Then going through the Node adjacent to EW1, it will create/update DijkstraNode to store the information of the adjacent station that is visited == false. For example, it will create a DijkstraNode for EW2 (EW2, 2400, [EW1ArrayIndex,EW2ArrayIndex], false). It will then look through the whole array in the Dijkstra Algorithm to find the next shortest totalDistance(i.e. EW2) and set the Node->isShortest to true. The HeaderNode will then now point to EW2. EW2->visited will be set to true.

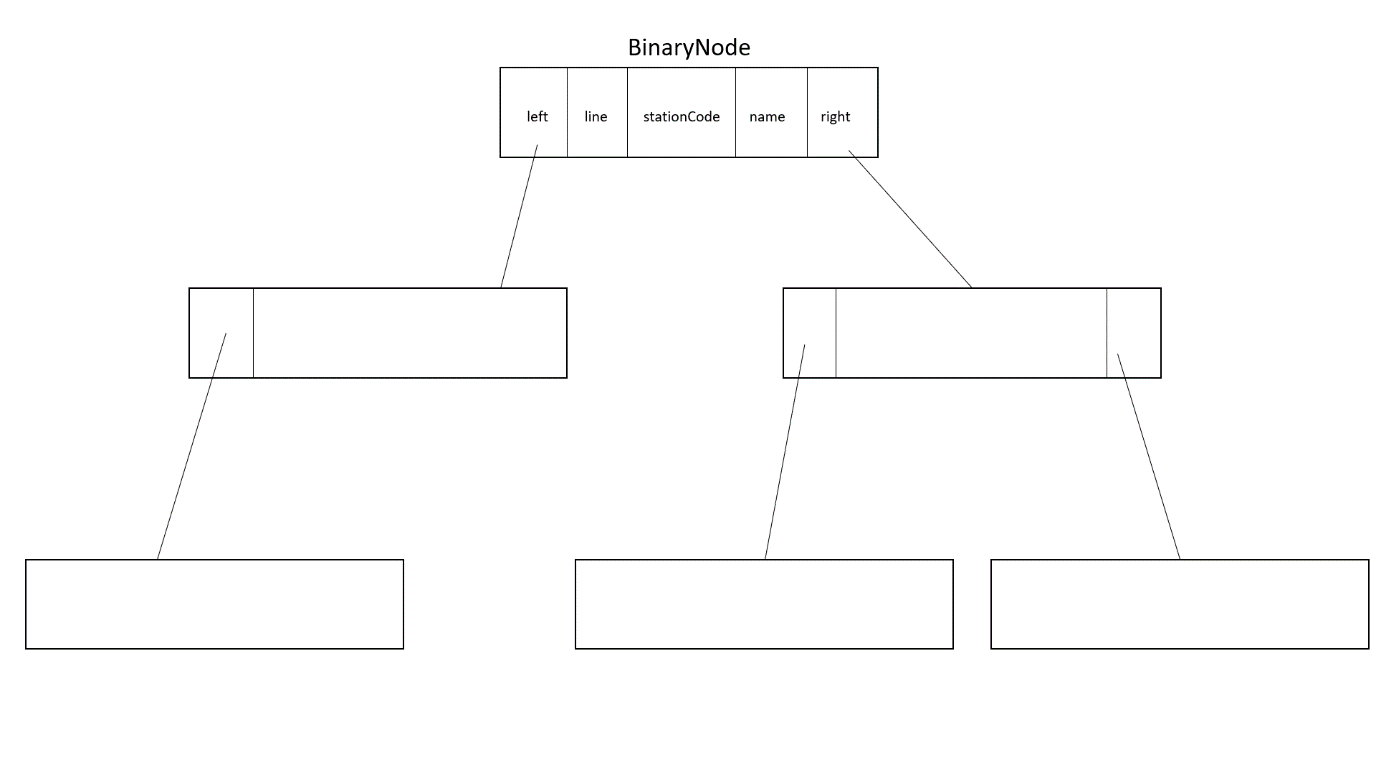
The algorithm will then check whether the HeaderNode is the destination station. If not, the loop continues, and it will then go through all the adjacent station of EW2 and create/update DijkstraNode for the adjacent station, finds the next shortest totalDistance in the array in Dijkstra Algorithm and set the HeaderNode to point to the new station with the shortest distance.

This loop continues until it hit the destination station that is selected as the next shortest totalDistance. (i.e. it may reach the station, but if it is NOT the shortest distance amongst those with isShortest == false in the Dijkstra Algorithm array , then it will not be selected as what the HeaderNode pointer is next pointed to and the loop continues with the next selected station)

Should the algorithm hit a station that is already in the Dijkstra Algorithm array, it will compare the total distance between the 2 paths. If the total distance of the new path is shorter, it will then update the DijkstraNode with the new totalDistance and the route(Queue).

When finding the next shortest totalDistance in the array in Dijkstra Algorithm, it will also check for the shortest distance that may not be adjacent to the current station in the HeaderNode.

## 5.3 Binary Search Tree (BST)



Binary Search Tree contains BinaryNode, consisting of:

* left, which is a pointer to the BinaryNode to the left of the current BinaryNode. It points to BinaryNode that has a stationCode that is less that the current BinaryNode stationCode
* line, which stores the line of the station in the BinaryNode
* stationCode, which stores the number associated with the station such as 1 for EW1
* name, which stores the name of the station in the BinaryNode
* right, which is a pointer to the BinaryNode to the right of the current BinaryNode. It points to the BinaryNode that has a stationCode that is more than the current BinaryNode stationCode

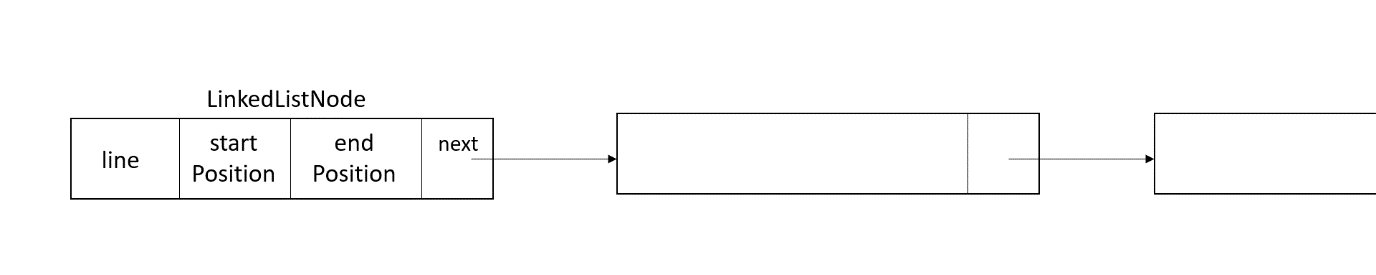
By using an Inorder traversal, it will loop through the whole BST from the smallest stationCode to the largest stationCode. The Inorder traversal will go to the left-most BinaryNode towards the right-most BinaryNode. (left-Root-right).

## 5.4 Queue

A QueueNode will contain a stationIndex, which stores the Array index of the station (e.g. 0 for EW1) and a next, which points to the QueueNode behind the current QueueNode.

By dequeuing the Queue, it will show the path taken by the Dijkstra’s Algorithm to reach the destination station of the shortest route.

## 5.5 Linked List



Linked List Node structure consists of:

* line, which is a line of the Metro
* startPosition, which is the first index in the Adjacency List Array of a station in that line
* endPosition, which is the last index in the Adjacency List Array of a station in that line

## 5.6 List

A List is an array of string datatype of a fixed size.

# Why these Data structures and Algorithms

When we were first thinking of how we can implement the finding of route function, we thought of using a doubly linked list so that we can show the train route in both directions, but that is quickly dismissed due to there being interchange. Therefore, we would need a way to show multiple train path in our data structure. We settled on using graph, but we do not know what kind of structure a graph is like.

List and linked list data structures are used for information storing for specific functions for reference or usage if needed.

After researching a bit, we came across Adjacency Matrix and Adjacency List. We find the Adjacency List structure to be similar to that of Dictionary and Hash Table, so we’re confident that we can implement it. Also, the Adjacency List will also allow us to traverse through and display the path, hence will be useful for storing of train routes.

Dijkstra’s Algorithm is used, as I learnt from Discrete Mathematics that it can be used to find the shortest distance between 2 points of a graph. Using these, we can hit basic Question 4 and advance Question 2 at once, hence helping us in this short and busy timeframe for the assignment. By making use of Queue, we are able to modify the algorithm slightly and store the path taken by the algorithm and hence find the shortest distance between 2 stations as well as displaying the shortest route.

Binary Search Tree is used because by using an Inorder traversal, it will display all nodes in ascending order. Therefore, it can be used to display all the stations in a given line.

# Appendices

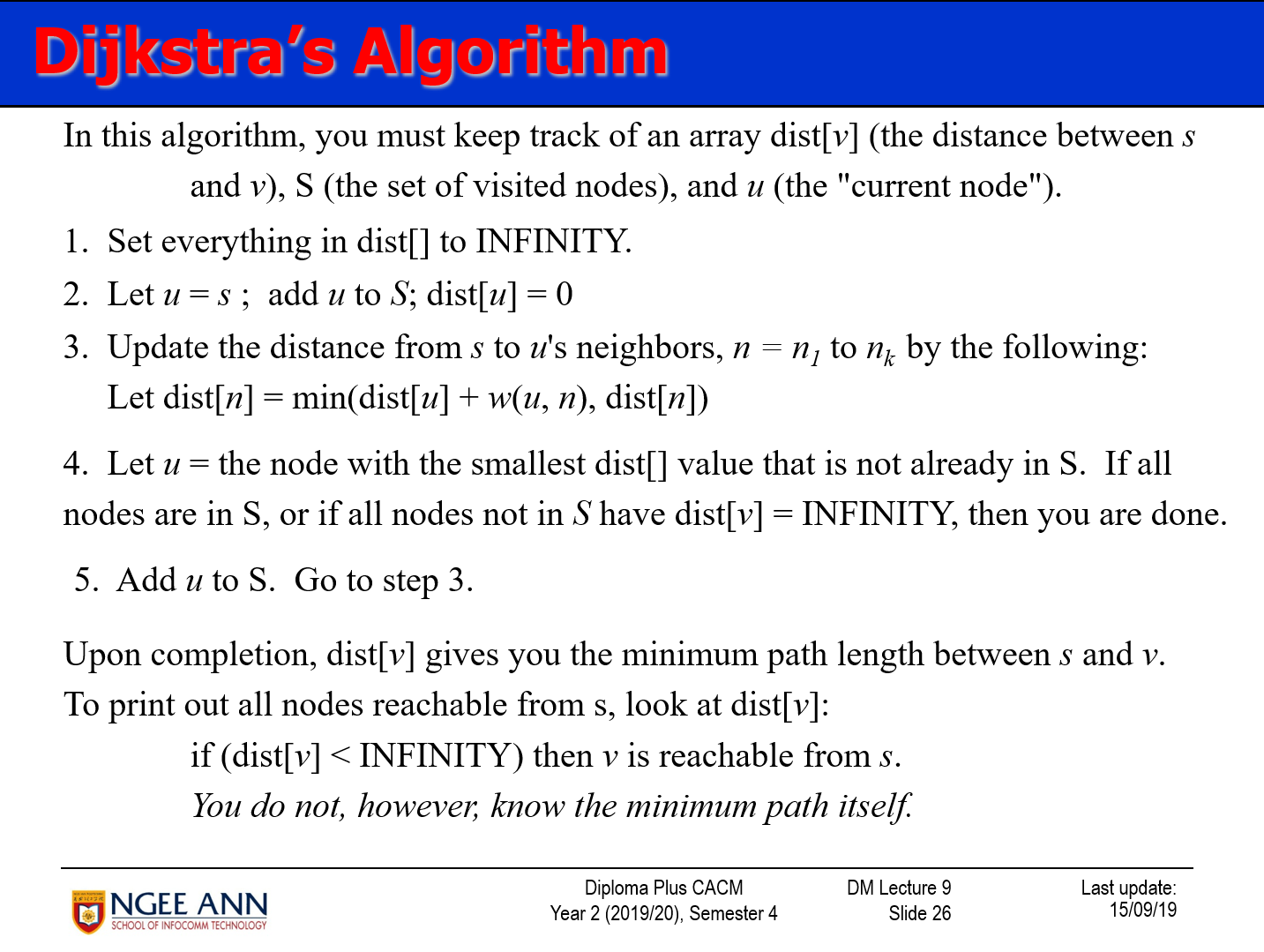


Figure : Dijkstra's Algorithm steps referenced

# 8. References

Discrete Mathematics (CACM3)- Diploma Plus Certificate in Advanced Computing Mathematics. (2020). *Graphs & Graph Algorithms(Part 2).*

GeeksforGeeks. (n.d.). *Graph and its representations.* Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/graph-and-its-representations/