

Comparative Analysis of shortest path algorithms on the basis of graphical plots

A

Project Report

*submitted in partial fulfillment of the
requirements for the award of the degree of*

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

by

Siddhi Gupta(61)

Yogesh(69)

Tushar Goyal(72)

Bhavuk Baluja(73)

500067967

500069549

500068373

500070089

Under the guidance of

Ms. Kalpana Rangra

Assistant Professor
Department of Cybernetics
School of Computer Science



**Department of Cybernetics,
School of Computer Science**



CANDIDATE'S DECLARATION

I/We hereby certify that the project work entitled **Comparative Analysis of shortest path algorithms on the basis of graphical plots** in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in Open Source and Open Standards and submitted to the Department of Computer Science & Engineering at Center for Information Technology, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my/ our work carried out during a period from **August, 2020** to **December, 2020** under the supervision of **Ms. Kalpana Rangra, Assistant Professor, Department of Cybernetics**.

The matter presented in this project has not been submitted by me/ us for the award of any other degree of this or any other University.

(Siddhi Gupta, Yogesh, Tushar Goyal, Bhavuk Baluja)
Roll No.- R100218061, R100218069, R100218072, R100218073

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 6 Dec, 2020

Ms. Kalpana Rangra
Project Guide

Dr. Monit Kapoor
Program Head- Department of Cybernetics
Center for Information Technology
University of Petroleum & Energy Studies
Dehradun – 248 001 (Uttarakhand)

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide **Ms. Kalpana Rangra**, for all the advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank our respected Program Head of the Department, **Dr. Monit Kapoor**, for his great support in doing our project in **Area (like Graphical interface, etc.)** at CIT.

We would like to thank all our **friends** for their help and constructive criticism during our project work. Finally we have no words to express our sincere gratitude to our **parents** who have shown us this world and for every support they have given us.

Name	Siddhi Gupta	Yogesh	Tushar Goyal	Bhavuk Baluja
Roll No.	R100218061	R100218069	R100218072	R100218073

ABSTRACT

The main reason to choose this topic as a Minor Project is to compare and analyse the algorithms for finding the Shortest Path by using the Dijkstra's, Bellman-Ford and Floyd-Warshall Algorithm Comparing, analyzing and plotting the graphs of each algorithm makes it easy to understand which algorithm is best for finding the shortest path for a particular type of question.

This project is created to find the shortest path very easily. Shortest Path Algorithm states that the algorithm to find the path which has minimal Distance(or path) between two nodes(or vertices). There are some Examples where the shortest path algorithm can be used like- telecommunications, network routing protocols, route planning, traffic control, path finding in social networks, computer games, and transportation systems, etc[1].

Keywords: Dijkstra's Algorithm, Bellman-Ford Algorithm, Floyd-Warshall Algorithm.

Table of Content

Contents:

1. Introduction	1
2. Problem Statement	1
3. Literature Review	1
4. Objectives	4
5. Methodology	4
6. System Requirements(Software/Hardware)	5
7. Implemented	
a. Pseudocode	5
b. Implemented Code	7
c. Output/ Result	10
8. Scope of Project	19
9. Conclusion	19
10. Schedule (Pert Chart)	20
11. References	21

List of Figures

List of Figures

1. Pert Chart	20
---------------	----

1. Introduction:

Today's requirement for the machine is less time to complete the work. Finding the shortest path or route from a source to a destination is important. The Shortest path problem is to find a path between two vertices or nodes in a graph such that the sum of the weights of its edges is minimized. There are some examples of using shortest path algorithms such as web mapping like: Google Maps, Transportation Systems, Computer Networks like the Internet etc. [1]The very common example of finding the shortest path is Rubik's cube, there are vertices or nodes to represent the states of puzzle and edges correspond to a single turn by which a person can understand the minimum possible number of moves. In this project, the shortest path is illustrated with the help of graphs. A graph is a mathematical or pictorial representation which contains vertices and edges. With the help of edges it is possible to walk from one vertex to another vertex because edges are a particular type of line segment joining two vertices[2]. By drawing the graph of any problem then it will be much easier to understand, so it helps people to get knowledge about the concept of the question in a short period of time.

For Example:

- A graph could be made for a routing protocol in which nodes have information about the network which is directly attached to it and directed arcs help to pass information and data packets between the nodes.
- In road networks, we make a graph in which nodes represent the road junctions and edges or directed arcs help to connect the segment between two junctions.
- It is the topology communication where graphs are made in which nodes having switching elements and the directed arcs help to link between switching elements.

2. Problem Statement:

There are several projects for finding the shortest path but they all give the output in written form, which is difficult to understand and it takes more time to understand the shortest path.

Therefore, there is a need for a project which compares all the algorithms on the basis of their time complexity and gives the output in written form as well as in graphical form step by step which makes it easy to understand the shortest path.

3. Literature:

To understand the shortest path algorithm we need to know in detail about Graph, Dijkstra's Algorithm, Bellman-Ford Algorithm and Floyd-Warshall Algorithm.

- 1. Graph representation and Explanation:** Graph is a mathematical representation that represents data in an organized manner. Graphs contain edges(E) and vertices(V). Edges

are the lines that connect two vertices and vertices are the points or the nodes. The shortest path problem is represented by the graph whether the graph is directed, undirected or mixed[2].

- 2. Dijkstra's Algorithm representation and explanation:** Dijkstra's Algorithm is a single-source shortest path problem algorithm which has non-negative edges weight. Single source shortest path states that finding the shortest path from the source vertex to all other vertices[3]. The node at which we are starting is called the initial node. It was discovered by the computer scientist **Edsger W. Dijkstra** in 1956.

Time complexity: $O(E \log V)$

1. Best Case Complexity = $O(E + V \log V)$
2. Average Case Complexity = $O(E + V \log V)$
3. Worst Case Complexity = $O(E + V \log V)$
4. Time complexity is $O(E + V^2)$, if the priority queue is not used.

Space complexity = $O(V)$

where, E is the number of edges and V is the number of vertices.

Dijkstra's algorithm is mainly used to find the shortest path in social networking, telephone network, finding the best route in the map etc.

There is several step to work on algorithm of Dijkstra's algorithm:

Step 1- Take the weighted graph.

Step 2- Take a starting vertex and assign value zero to it and assign all other vertices values infinite.

Step 3- Go to each vertex and update its path and distance.

Step 4- If the distance of adjacent is not more than the new path distance then, don't update.

Step 5- Not visit the nodes which are already updated in the above steps.

Step 6- After visiting each vertex make the graph which contains the shortest path[4].

- 3. Bellman-Ford Algorithm representation and explanation:** Bellman-Ford Algorithm is a single-source shortest path problem algorithm which can have negative edge weight also. It was founded by Alfonso Shimbel in 1955, yet is rather named after Richard Bellman and Lester Ford Jr., who distributed it in 1958 and 1956. Bellman-Ford algorithm works on the path from the starting vertex to the other vertices. It is more versatile and has the capability to solve the problem by the graph which contains negative edge weights. In real life, some examples where negative weight edges can exist like cash flow, heat released in a chemical reaction, heat absorbed in chemical reaction etc[3]. If a graph contains a "negative cycle" (a cycle whose edges sum is negative value).

Time complexity :

1. Best Case Complexity = $O(E)$
2. Average Case Complexity = $O(VE)$
3. Worst Case Complexity = $O(VE)$

Space complexity = $O(V)$

where, E is the number of edges and V is the number of vertices.

Bellman-Ford algorithm is mainly used to find the shortest path, calculating shortest paths in routing etc.

There is several step to work on algorithm of Bellman-Ford:

Step 1- Take the weighted graph.

Step 2- Take a starting vertex and assign value to it zero and assign all other vertices values infinite.

Step 3- Use this formula (for each edge A-B)

If $\text{Distance}[B] > \text{Distance}[A] + \text{weight of edge A and B}$
then update $\text{Distance}[B]$
 $\text{Distance}[B] = \text{Distance}[A] + \text{weight of edge A and B}$

Step 4- We can do step 3 for (V-1) times and V is the number of vertices in the graph.

Step 5- It tells if there is a negative weight cycle in the graph. Check for each edge A-B:

If $\text{Distance}[B] > \text{Distance}[A] + \text{weight of edge A and B}$

Step6- After visiting each vertex make the graph which contains the shortest path.[5]

- 4. Floyd-Warshall Algorithm representation and explanation:** Floyd-Warshall Algorithm is used for finding the shortest path between all pairs of vertices. The Floyd-Warshall Algorithm was proposed by **Robert Floyd** in 1962. The graph used in Floyd-Warshall Algorithm is a weighted graph with positive or negative edge weights. The Floyd-Warshall Algorithm is used for both directed and undirected weighted graphs. It is applied in negative edges but not for negative cycles(sum of edges in cycle is equal to negative value)[3]. Floyd-Warshall Algorithm is also known by different names as Floyd's algorithm, Roy-Floyd algorithm, WFI algorithm or Roy-Warshall algorithm.

Time complexity: $O(V^3)$

1. Best Case Complexity = $O(V^3)$
2. Average Case Complexity = $O(V^3)$
3. Worst Case Complexity = $O(V^3)$

Space complexity = $O(V^2)$

where, V is the number of vertices.

Floyd-Warshall Algorithm is mainly used to find the shortest path in directed graphs, find the Inversion of real matrices, find the transitive closure of directed graphs, etc.

There are several step to work on algorithm of Bellman-Ford:

Step 1- Create a matrix of dimension (w*w) but here we take (4*4) matrix. where, w is the number of vertices. The row and also the column are unit indexed as I and j resp. where, i and j are unit vertices of the graph.

Make a matrix and fill the cell in the matrix with the distance between i^{th} and j^{th} vertex.

Step 2- Create a matrix B^1 using matrix B^0 and fill the element in matrix B^1 by help of formula but the elements in the first column and the first row are left as they are

$$B[i][j] = (B[i][k] + B[k][j]) \text{ if } (B[i][j] > B[i][k] + B[k][j])$$

where, k is the vertex=1.

Step 3- B^2 is created using B^3 . The elements in the second column and the second row are left as they are. Then, repeat the step of 2 by putting k vertex of value is 2.

Step 4- Same way B^3 and B^4 is also created.

Step 5- B^4 gives the shortest path.

Step 6- Make a graph of matrix $B^4[6]$.

4. Objective:

The main objective of this project is to compare all the algorithms by plotting their graphs on the basis of their time complexity and give the output in written form as well as in graphical form step by step which makes it easy to understand the shortest path and tells which algorithm should be used to solve the question more efficiently for a particular type of question.

5. Methodology:

In our project, we are comparing, analysing and plotting the graphs of each algorithm which algorithm is best for finding the shortest path for a particular type of question.

- ❖ Design and development of prerequisites:
 - Preparing the data flow diagram of the project.
- ❖ Implementation, Comparison and Analysis of Algorithms:
 - Implementing different algorithms.
 - Comparing the algorithms by plotting their graphs on the basis of their time complexity.
 - Plotting the graphs of shortest path step by step using different algorithms.
- ❖ Integrating the modules.

- ❖ Testing:
 1. Testing the integration of modules.
 2. Testing the system.

6. System Requirement:

- ❖ Operating System-
 - Linux/ Windows/ MacOS or any other OS.
- ❖ Software-
 3. gcc compiler or any other online 'C' Compiler like: GDB compiler.
- ❖ Hardware-
 4. Memory - 2GB (Recommended)
 5. Storage - 4GB (Recommended)

7. Implemented code:-

7.1: Pseudocode:- Pseudocode for the algorithms used in code are-

- **Dijkstra's Algorithm:-**

```

function dijkstra(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
    distance[S] <- 0
    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
  
```

```

        previous[V] <- U
    return distance[], previous[]

```

- **Bellman-Ford Algorithm:-**

```

function bellmanFord(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
    distance[S] <- 0
    for each vertex V in G
        for each edge (U,V) in G
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
    for each edge (U,V) in G
        If distance[U] + edge_weight(U, V) < distance[V}
            Error: Negative Cycle Exists
    return distance[], previous[]

```

- **Floyd-Warshall Algorithm:-**

```

n = no of vertices
A = matrix of dimension n*n

```

```

for i = 1 to n
    for j = 1 to n
        if there is an edge from i to j
            dist[i][j] = the length of the edge from i to j
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
             $A^k[i, j] = \min (A^{k-1}[i, j], A^{k-1}[i, k] + A^{k-1}[k, j])$ 
return A

```

7.2: Implemented code:-

1. void dijkstra(int G[Max][Max], int n , int startnode)

This function calculates and displays the shortest path and shortest distance to each node from the starting node and provides the time complexity of the Algorithm using the Dijkstra's Algorithm.

Arguments passed by calling function,

G[Max][Max] is a 2D Array which holds the values of the distance between each set of nodes which is entered by the user.

' n ' is the no. of nodes entered by the user.

' startnode ' is the starting node entered by the user.

2. void bellman(int G[Max][Max], int n, int startnode)

This function calculates and displays the shortest path and shortest distance to each node from the starting node and provides the time complexity of the Algorithm using the Bellman Algorithm.

Arguments passed by calling function,

G[Max][Max] is a 2D Array which holds the values of the distance between each set of nodes which is entered by the user.

‘ n ‘ is the no. of nodes entered by the user.

‘ startnode ‘ is the starting node entered by the user.

3. **void floydWarshall(int G[Max][Max], int n)**

This function calculates and displays the shortest path and shortest distance to each node from each set of starting nodes and provides the time complexity of the Algorithm.

Arguments passed by calling function,

G[Max][Max] is a 2D Array which holds the values of the distance between each set of nodes which is entered by the user.

‘ n ‘ is the no. of nodes entered by the user.

4. **void Display(int Pred[], int startnode, int n, int f)**

This function shows the graphical representation of shortest path using Dijkstra’s and Bellman-ford Algorithms step by step.

Arguments passed by calling function,

Pred[] is a 1D Array which holds the values of the Parent node of each node of the shortest path which is calculated by each algorithm.

‘startnode’ is the starting node entered by the user.

‘n’ is the no. of nodes entered by the user.

‘f’ tells the function for which algorithm the function is calling. ‘1’ stands for Dijkstra’s Algorithm and ‘2’ stands for Bellman-ford Algorithm.

5. **void GG(int G[Max][Max], int n, int startnode)**

This function calculates the complexities for different no. of nodes by using all the 3 algorithms and stores the complexities in 3 different arrays (named as D[], B[], F[]).

Arguments passed by calling function,

G[Max][Max] is a 2D Array which holds the values of the distance between each set of nodes which is entered by the user.

‘ n ‘ is the no. of nodes entered by the user.

‘ startnode ‘ is the starting node entered by the user.

6. void TG(int n)

This function uses the arrays in which complexities are stored using GG functions and display the graph at those points according to the complexities by using different algorithms. And makes a complete graph of comparing the complexities of the 3 algorithms.

Argument passed by calling function,

‘ n ‘ is the no. of nodes entered by the user.

7. int main()

This function uses the file handling, fetch and set the values of the distance from each node to each other node in the matrix named as G[Max][Max].

This function calls the dijkstra, bellman, floydWarshall functions and displays the time complexities of each algorithm.

And Calls the Display Function for Dijkstra’s, Bellman-ford and Floyd- Warshall Algorithms.

It opens and closes the window opened to show the graphical representation of shortest path.

It calls the GG() function, TG() function.

7.2: Output/ Result:

Output:

```
bhavuk@bhavuk:~/Desktop$ gcc MinorFile.c -lgraph
bhavuk@bhavuk:~/Desktop$ ./a.out
No. you have entered is wrong. Enter value from 1 to 50
Enter number of vertices: █
```

Input:

5

Output:

```
bhavuk@bhavuk:~/Desktop$ gcc MinorFile.c -lgraph
bhavuk@bhavuk:~/Desktop$ ./a.out
No. you have entered is wrong. Enter value from 1 to 50
Enter number of vertices: 5
ENTER THE ADJACENCY MATRIX
(Enter 999 for all non existence of edges)
Enter Matrix is:-
|      0      | |      3      | |      4      | |      5      | |      999      |
|      2      | |      0      | |     12      | |     23      | |      7      |
|      9      | |      8      | |      0      | |     999      | |      7      |
|      6      | |     15      | |      4      | |      0      | |     12      |
|      5      | |      6      | |      7      | |      9      | |      0      |

Enter the Starting Node from 1 to 5 (Number):
█
```

Input:

1

Output: Written form of Floyd-Warshall Algorithm:

```
*** Floyd-Warshall ALGORITHM ***

  0   3   4   5  10
  2   0   6   7   7
  9   8   0  14   7
  6   9   4   0  11
  5   6   7   9   0

For starting node= 1:
Distance of 2 = 3
Path = 2-> 1
Distance of 3 = 4
Path = 3-> 1
Distance of 4 = 5
Path = 4-> 1
Distance of 5 = 10
Path = 5-> 2-> 1

For starting node= 2:
Distance of 1 = 2
Path = 1-> 2
Distance of 3 = 6
Path = 3-> 1-> 2
Distance of 4 = 7
Path = 4-> 1-> 2
Distance of 5 = 7
Path = 5-> 2

For starting node= 3:
Distance of 1 = 9
Path = 1-> 3
Distance of 2 = 8
Path = 2-> 3
Distance of 4 = 14
Path = 4-> 1-> 3
Distance of 5 = 7
Path = 5-> 3

For starting node= 4:
Distance of 1 = 6
Path = 1-> 4
Distance of 2 = 9
Path = 2-> 1-> 4
Distance of 3 = 4
Path = 3-> 4
Distance of 5 = 11
Path = 5-> 3-> 4

For starting node= 5:
Distance of 1 = 5
Path = 1-> 5
Distance of 2 = 6
Path = 2-> 5
Distance of 3 = 7
Path = 3-> 5
Distance of 4 = 9
Path = 4-> 5

Runtime time of floydWarshall is: 0.000035
```

Written form of Dijkstra's Algorithm:

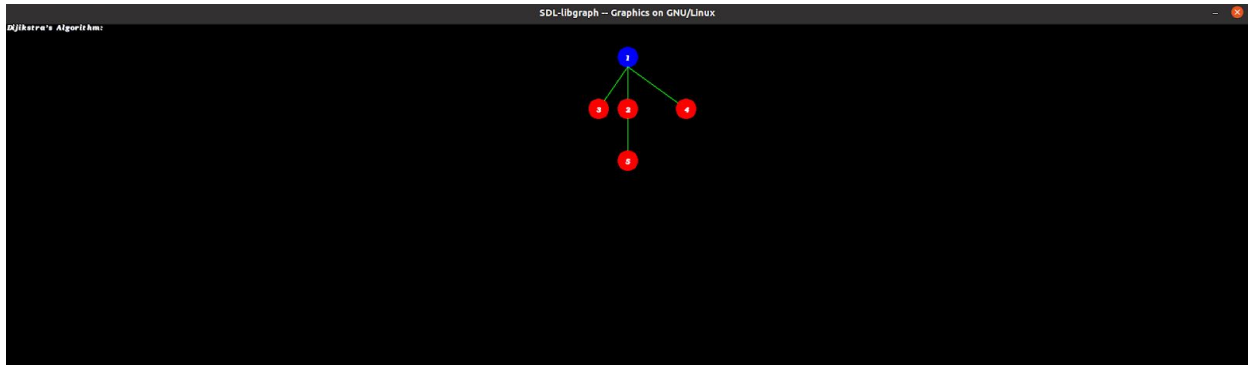
```
*** DIJKSTRA ALGORITHM ***  
  
Distance of 2 = 3  
Path = 2 <-1  
Distance of 3 = 4  
Path = 3 <-1  
Distance of 4 = 5  
Path = 4 <-1  
Distance of 5 = 10  
Path = 5 <-2 <-1  
Runtime time of Dijkstra is: 0.000017
```

Written form of Bellman Ford Algorithm:

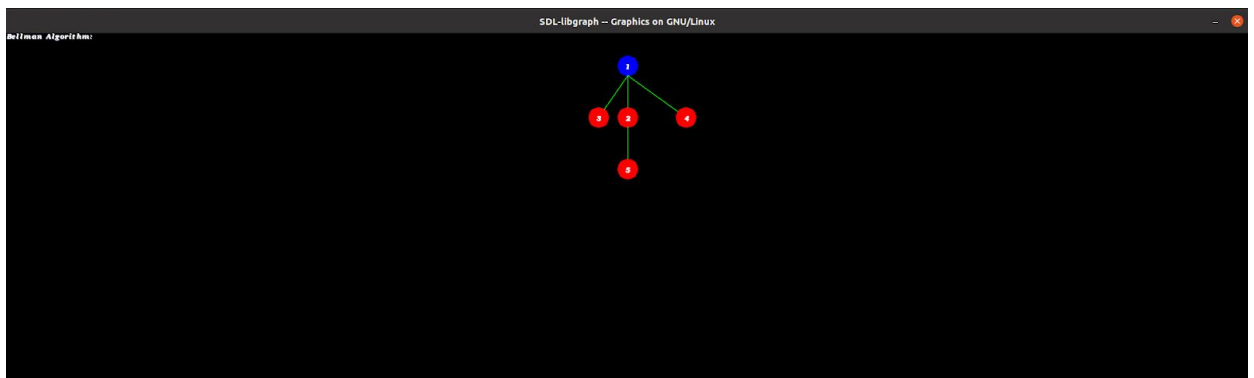
```
*** BELLMAN FORD ALGORITHMS ***  
  
Distance of 2 = 3  
Path = 2 <-1  
Distance of 3 = 4  
Path = 3 <-1  
Distance of 4 = 5  
Path = 4 <-1  
Distance of 5 = 10  
Path = 5 <-2 <-1  
Runtime time of Bellman is: 0.000008
```

Output:

Dijkstra's Algorithm:

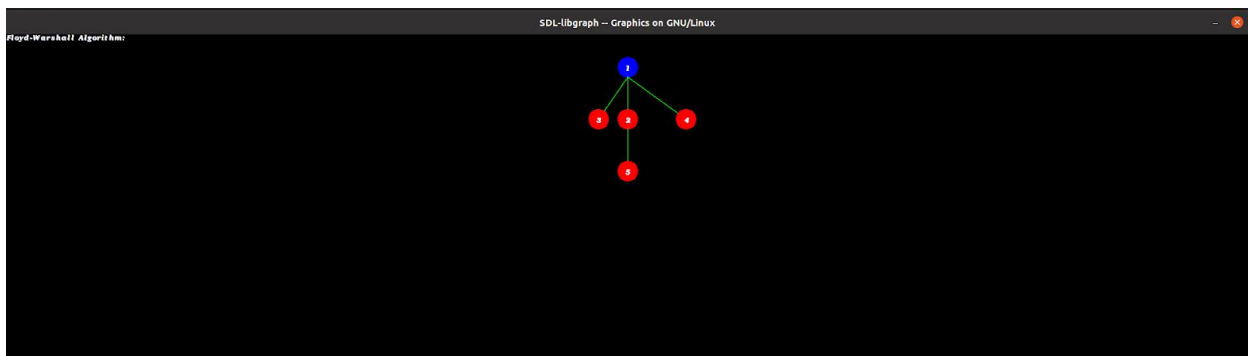


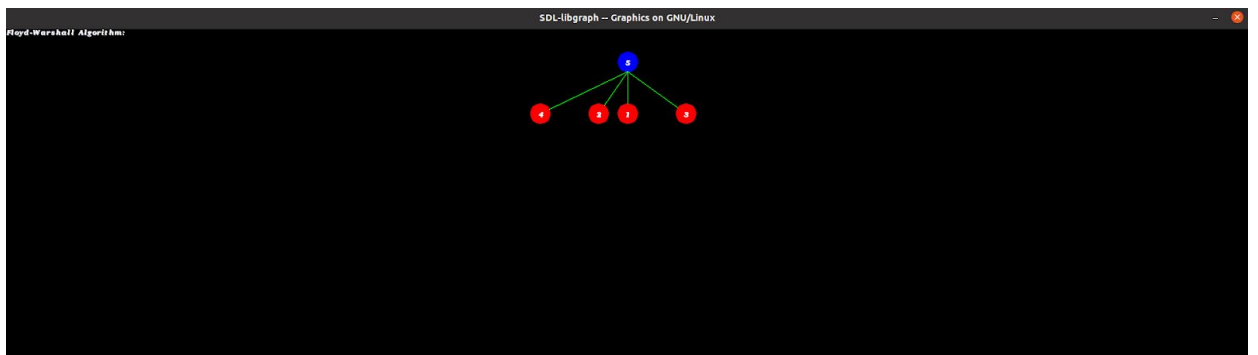
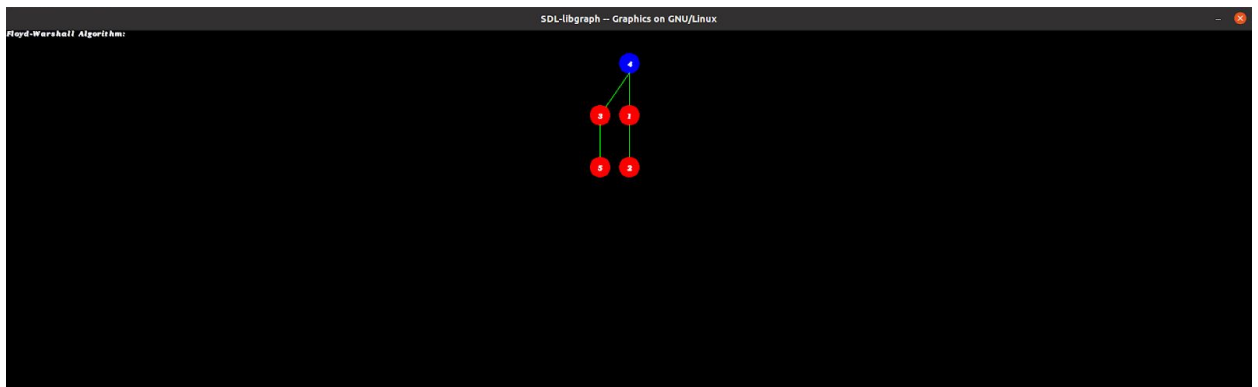
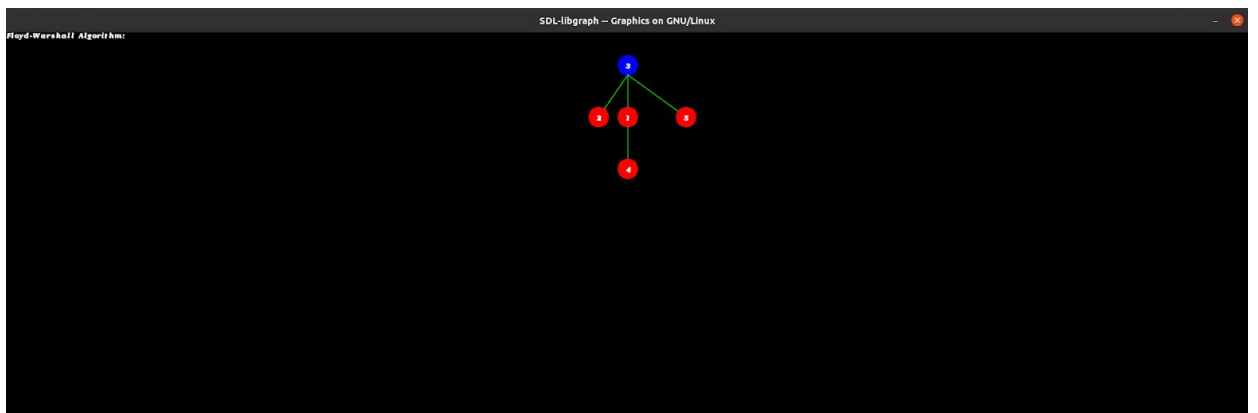
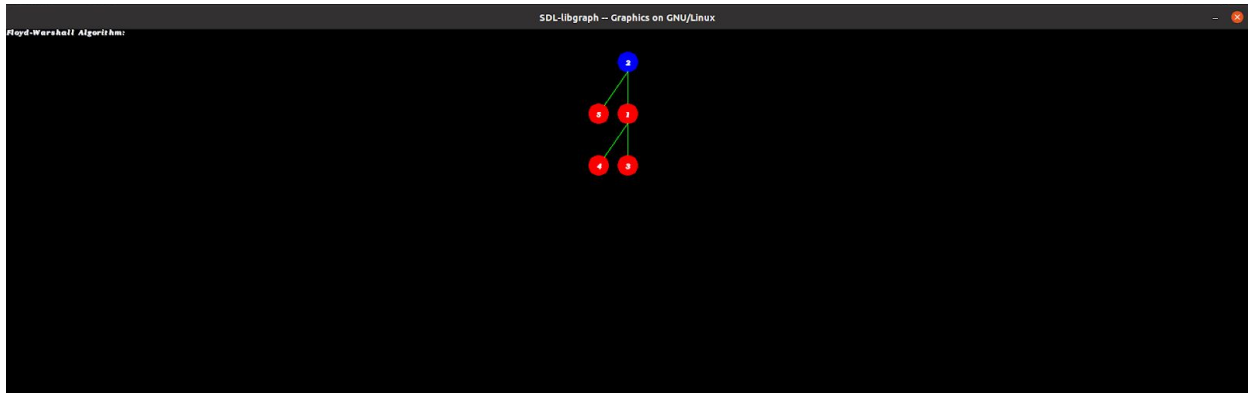
Bellman-Ford Algorithm:



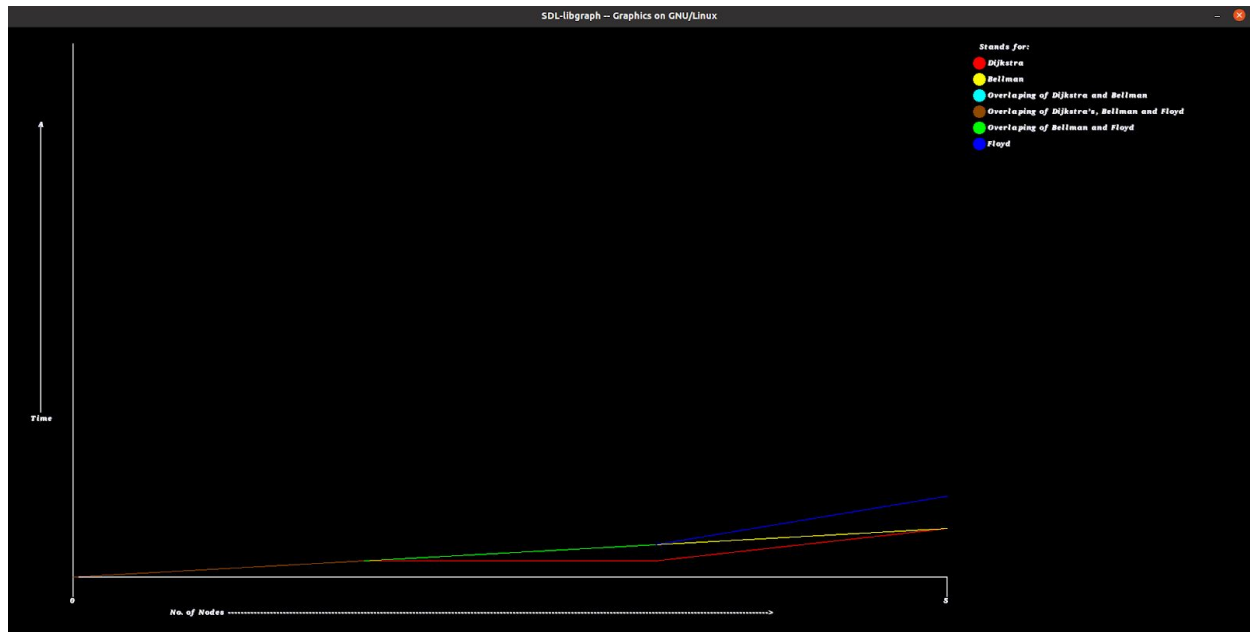
Floyd-Warshall algorithm:

[Note: In Floyd-warshall Algorithm, it gives the output for each node as a starting node, i.e. It makes the no. of graphs as same that of the no. of nodes entered by the user.]





Time Complexity Graph:



For Negative Cycle:

Output:

Dijkstra's Algorithm:

```
*** DIJKSTRA ALGORITHM ***  
  
Distance of 1 = 1  
Path = 1 <-12 <-28 <-10  
Distance of 2 = 4  
Path = 2 <-1 <-12 <-28 <-10  
Distance of 3 = 0  
Path = 3 <-12 <-28 <-10  
Distance of 4 = 3  
Path = 4 <-10  
Distance of 5 = 3  
Path = 5 <-12 <-28 <-10  
Distance of 6 = 2  
Path = 6 <-12 <-28 <-10  
Distance of 7 = 2  
Path = 7 <-34 <-1 <-12 <-28 <-10  
Distance of 8 = 4  
Path = 8 <-1 <-12 <-28 <-10
```

```

Distance of 9 = 3
Path = 9 <-10
Distance of 11 = 2
Path = 11 <-21 <-12 <-28 <-10
Distance of 12 = -2
Path = 12 <-28 <-10
Distance of 13 = 1
Path = 13 <-12 <-28 <-10
Distance of 14 = 4
Path = 14 <-1 <-12 <-28 <-10
Distance of 15 = 0
Path = 15 <-34 <-1 <-12 <-28 <-10
Distance of 16 = 3
Path = 16 <-10
Distance of 17 = 3
Path = 17 <-12 <-28 <-10
Distance of 18 = 5
Path = 18 <-34 <-1 <-12 <-28 <-10
Distance of 19 = 1
Path = 19 <-12 <-28 <-10
Distance of 20 = 4
Path = 20 <-1 <-12 <-28 <-10
Distance of 21 = 0
Path = 21 <-12 <-28 <-10
Distance of 22 = 3
Path = 22 <-10
Distance of 23 = 3
Path = 23 <-12 <-28 <-10
Distance of 24 = 5
Path = 24 <-34 <-1 <-12 <-28 <-10
Distance of 25 = 1
Path = 25 <-12 <-28 <-10

```

```

Path = 26 <-1 <-12 <-28 <-10
Distance of 27 = 0
Path = 27 <-12 <-28 <-10
Distance of 28 = 3
Path = 28 <-10
Distance of 29 = 3
Path = 29 <-12 <-28 <-10
Distance of 30 = 5
Path = 30 <-34 <-1 <-12 <-28 <-10
Distance of 31 = 1
Path = 31 <-12 <-28 <-10
Distance of 32 = 4
Path = 32 <-1 <-12 <-28 <-10
Distance of 33 = 0
Path = 33 <-12 <-28 <-10
Distance of 34 = -4
Path = 34 <-1 <-12 <-28 <-10
Distance of 35 = 3
Path = 35 <-12 <-28 <-10
Runtime time of Dijkstra is: 0.000079

```

Bellman-Ford Algorithm:

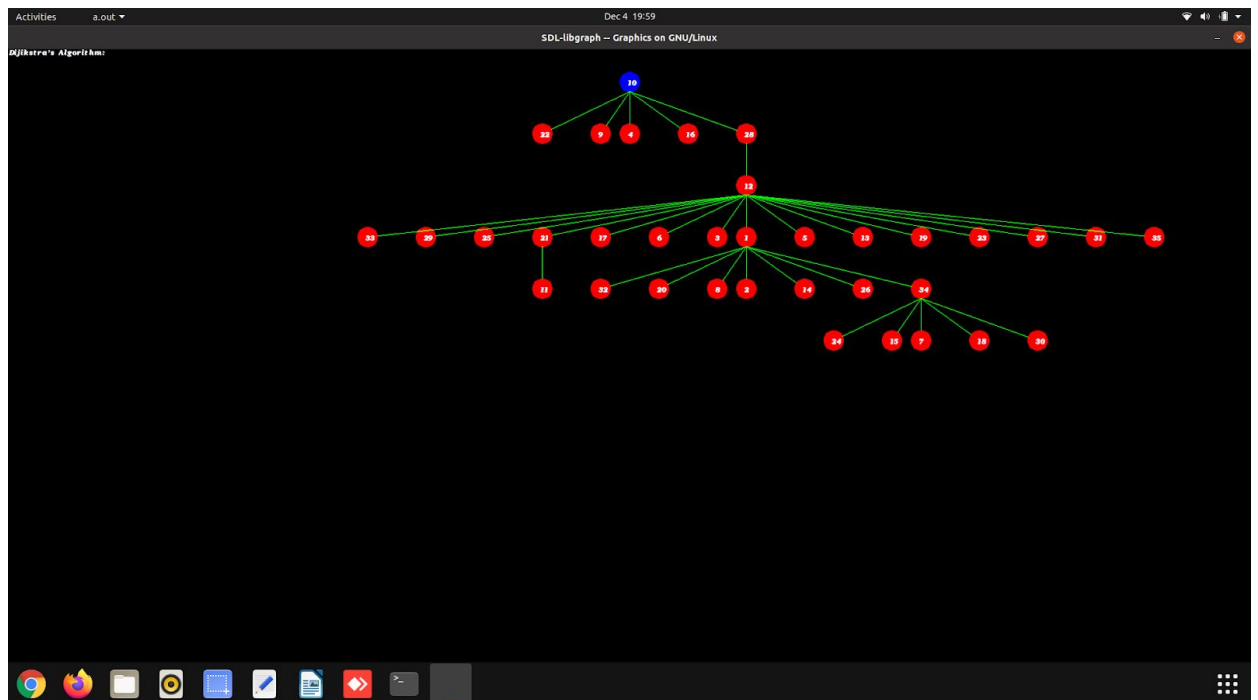
```
*** BELLMAN FORD ALGORITHMS ***
Negative Cycle Found
Runtime time of Bellman is: 0.001097
```

Floyd-Warshall algorithm:

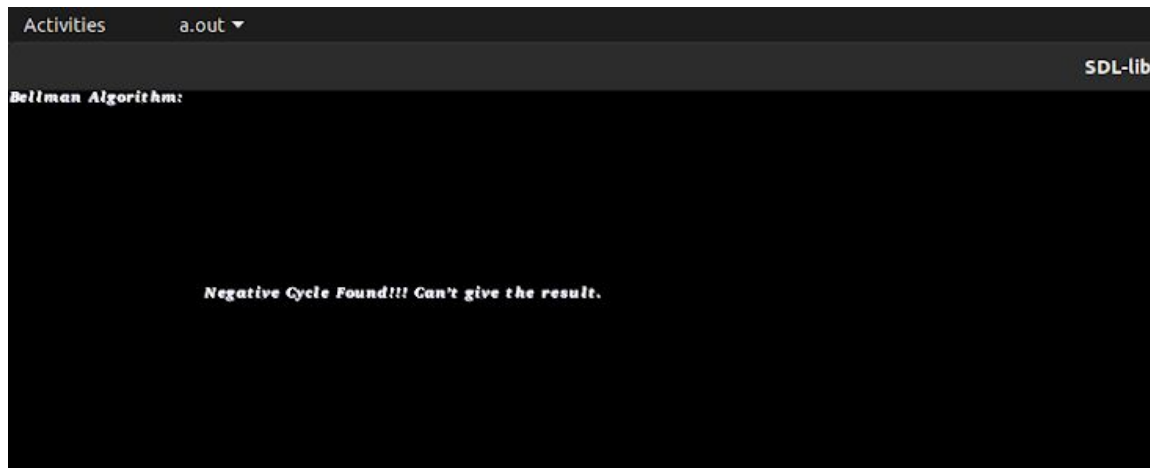
```
*** Floyd-Warshall ALGORITHM ***
Negative Cycle Found
Runtime time of floydWarshall is: 0.001712
```

Output:

Dijkstra's Algorithm (Graphical Representation):



Bellman-Ford Algorithm (Graphical Representation):

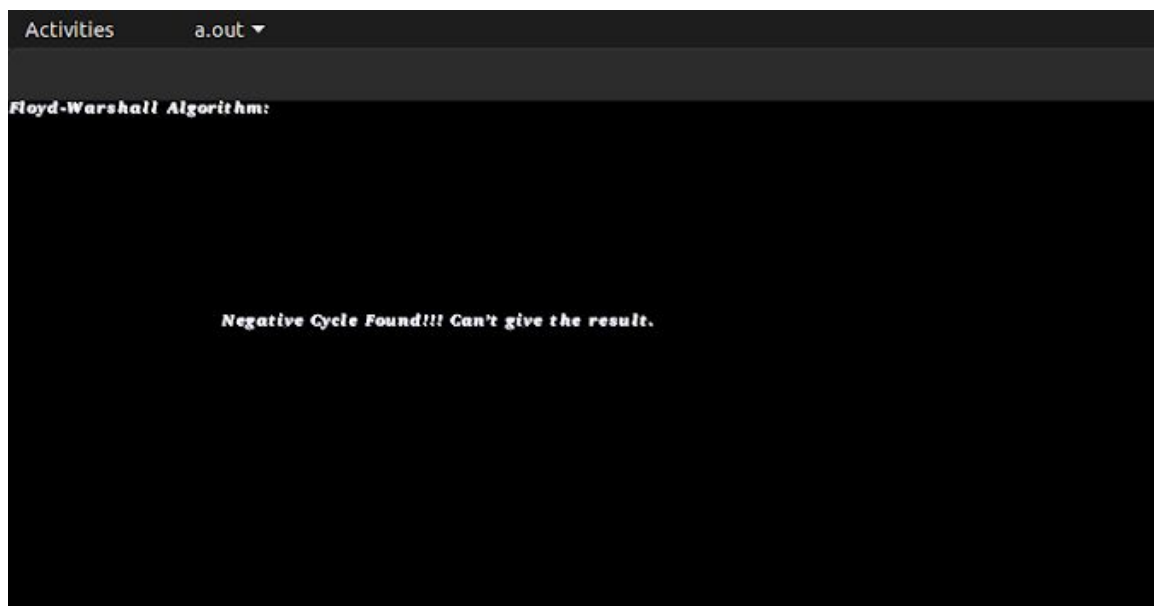


```
Activities    a.out ▼
Bellman Algorithm:

Negative Cycle Found!!! Can't give the result.
```

The image shows a terminal window with a dark background. At the top, there is a header bar with 'Activities' and 'a.out ▼'. Below this, the text 'Bellman Algorithm:' is displayed. The main output of the program is 'Negative Cycle Found!!! Can't give the result.'.

Floyd-Warshall algorithm (Graphical Representation):

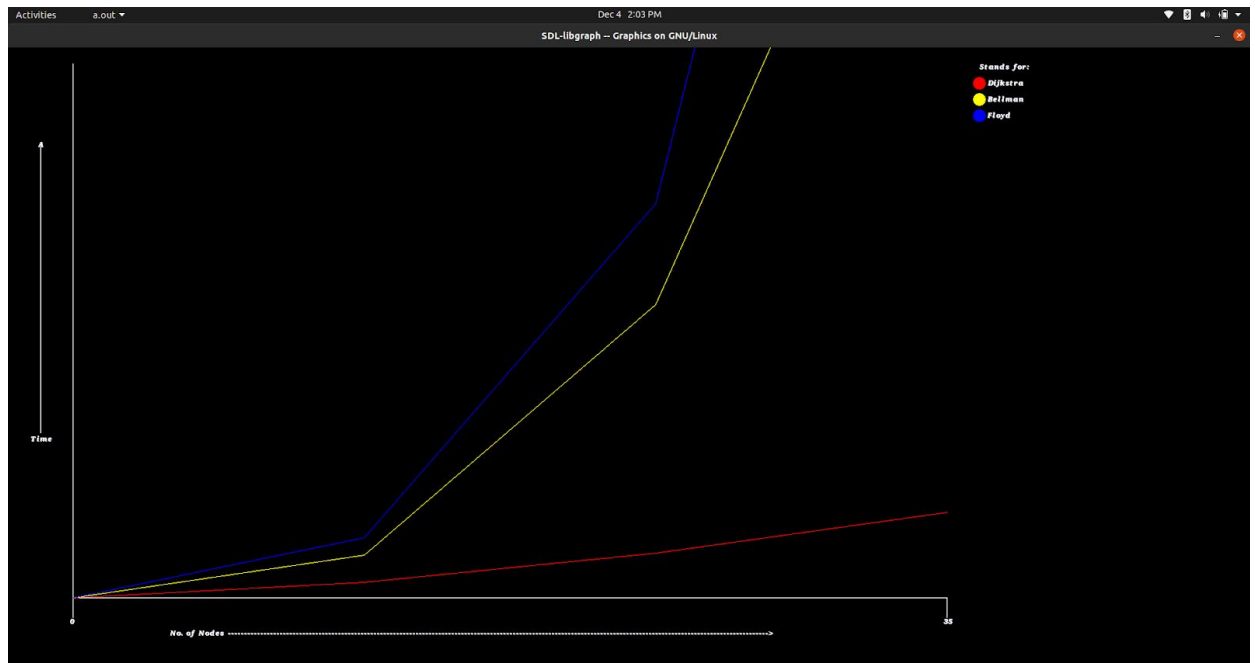


```
Activities    a.out ▼
Floyd-Warshall Algorithm:

Negative Cycle Found!!! Can't give the result.
```

The image shows a terminal window with a dark background. At the top, there is a header bar with 'Activities' and 'a.out ▼'. Below this, the text 'Floyd-Warshall Algorithm:' is displayed. The main output of the program is 'Negative Cycle Found!!! Can't give the result.'.

Time Complexity Graph:



8. Scope of Project:-

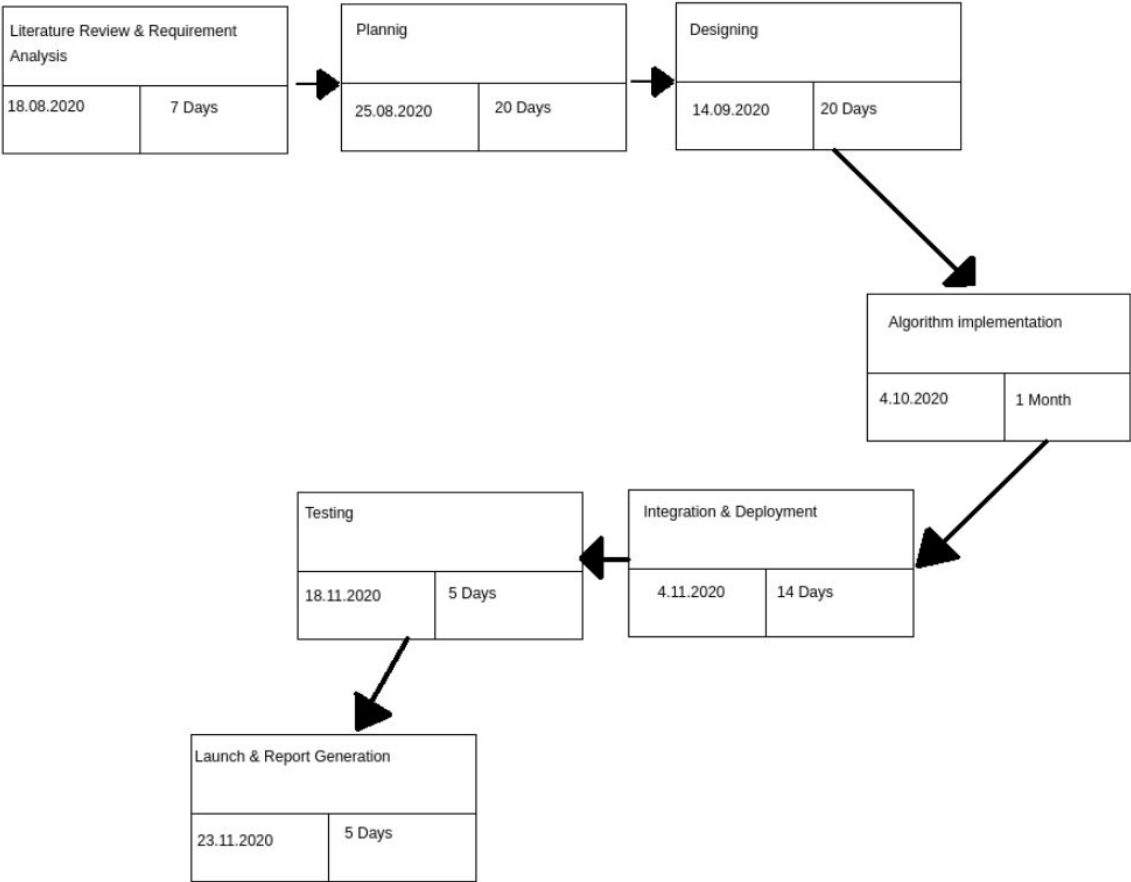
- ❖ Least-cost paths are calculated for instance to establish tracks of electricity lines and oil pipelines.
- ❖ Network Routing Protocol.
- ❖ Road Networks.

9. Conclusion:

By plotting the graphs of all 3 Algorithms on the basis of their time complexities it is easy to compare the algorithms. All of the 3 Algorithms provides a solution with having some time complexities. The time complexity for each of the 3 Algorithms are on the basis of their efficiency to solve the shortest path problem.

For more convenience it gives the output in written form as well as in graphical form step by step which makes it easy to understand the shortest path.

10. Schedule (PERT Chart) :



11. References:

1. Amgad Madkour, Walid G. Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, Saleh Basalamah. *A Survey of Shortest-Path Algorithms*. Purdue University, West Lafayette, USA, Umm Al-Qura University, Makkah, KSA, may 8, 2017.
2. Kairanbay Magzhan, Hajar Mat Jani — *A Review And Evaluations Of Shortest Path Algorithms*, the international journal of scientific & technology research volume 2, issue 6, june 2013, pp. 99-104.
3. 'Shortest path problem'(8 September 2020)*Wikipedia*. Available at: https://en.wikipedia.org/wiki/Shortest_path_problem(Accessed: 25 August 2020).
4. 'Dijkstra's shortest path algorithm'(22 April 2020)*GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>(Accessed: 26 August 2020).
5. 'Bellman–Ford Algorithm'(23 April 2020)*GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/>(Accessed: 28 August 2020).
6. 'Floyd Warshall Algorithm'(18 July 2019)*GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/>(Accessed: 28 August 2020).

Approved By

(Ms. Kalpana Rangra)
Project Guide

(Dr. Monit Kapoor)
Head of Department