MINOR PROJECT 1

Mid-Sem Report

on

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

Submitted 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
University of Petroleum and Energy Studies
Dehradun - 248007

Table of Content

Contents:

List of Figures

List of Figures

MID Semester Progress Report
Project title

1. 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.

2. Objectives defined (as in synopsis):-

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.

3. Objectives Achieved:-

- Implementation of code for Dijkstra’s, Bellman-Ford and Floyd-Warshall Algorithms in which all algorithms work properly , then Integrate those modules together.
- Provides the path and complexities of 2 algorithms (Dijkstra and Bellman-Ford Algorithm) in written form.
- Provides the time complexity and shortest distance to each node from each set of starting nodes of Floyd-Warshall Algorithm in written form.
- Shows the graphical representation of Dijkastra’s and Bellman-Ford Algorithms in which complete workings are shown step by step.

4. Objectives pending:-

- Providing the shortest path using the Floyd-Warshall Algorithm in written form.
- Showing the graphical representation of Floyd-Warshall Algorithm in which complete working will be shown step by step.
- Plotting the graph between x and y planes for comparing the time complexities of algorithms (Dijkstra’s, Bellman-Ford and Floyd-Warshall Algorithm).

5. 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[]
```

- **<u>Floyd-Warshall Algorithm:-</u>**

```
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
```

6. 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 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. **Int main()**

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

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

7. Scope of Project:-

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

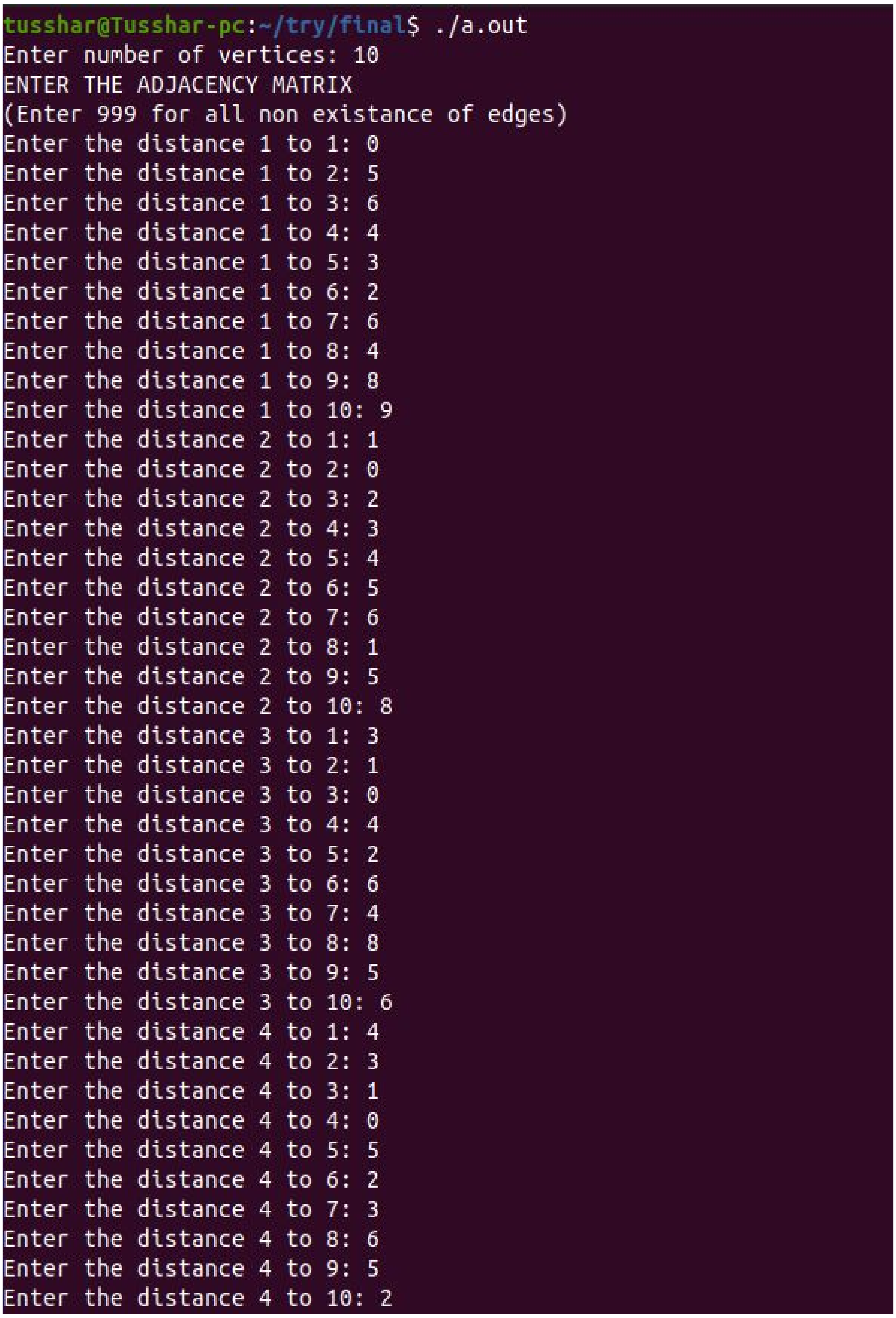
8. Participation and role of each student in work done:-

- **Siddhi Gupta:** Code implementation, Research about the Topic, Design Analysis.
- **Yogesh:** Code implementation, Research about the Topic, Testing code and Error Resolving.
- **Tushar Goyal:** Code implementation, Testing code and Error Resolving, Research about Topics, Installation of Necessary Header files.
- **Bhavuk Baluja:** Research about the Topic, Code implementation, Integrating the Modules, Testing the integration of modules.

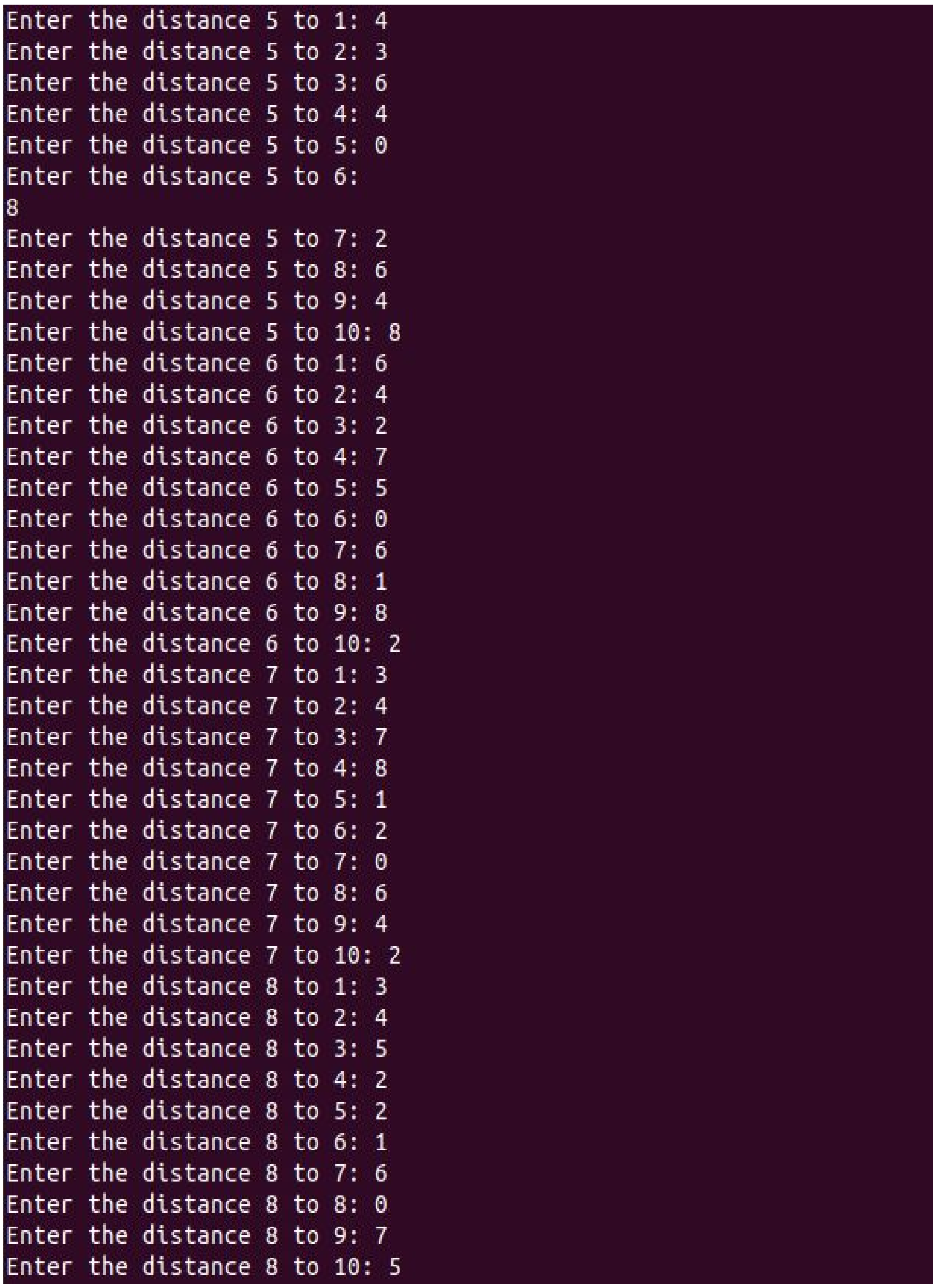
9. Outputs/Results of work done:-

- **Output with Positive weight:**

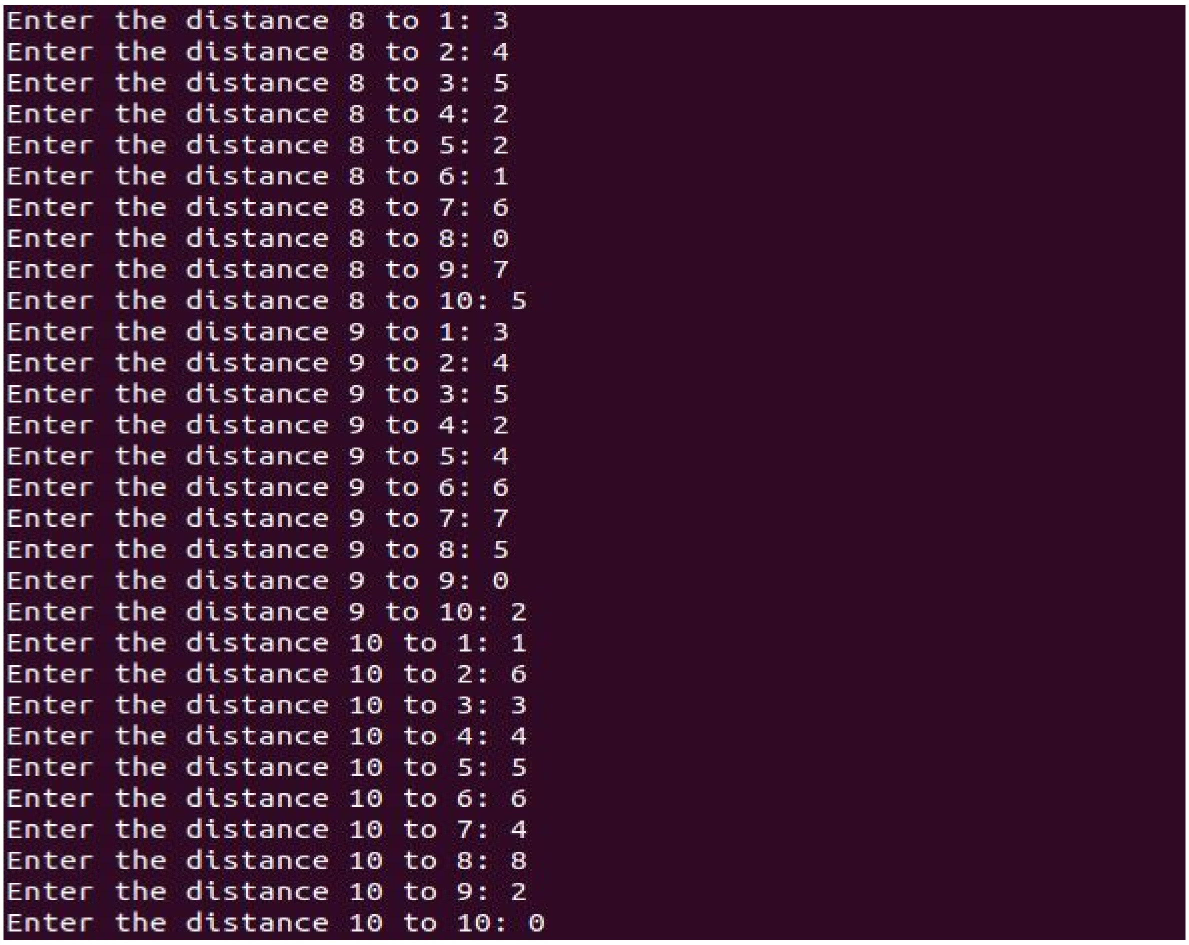
Input:



```
tusshar@Tusshar-pc:~/try/final$ ./a.out
Enter number of vertices: 10
ENTER THE ADJACENCY MATRIX
(Enter 999 for all non existance of edges)
Enter the distance 1 to 1: 0
Enter the distance 1 to 2: 5
Enter the distance 1 to 3: 6
Enter the distance 1 to 4: 4
Enter the distance 1 to 5: 3
Enter the distance 1 to 6: 2
Enter the distance 1 to 7: 6
Enter the distance 1 to 8: 4
Enter the distance 1 to 9: 8
Enter the distance 1 to 10: 9
Enter the distance 2 to 1: 1
Enter the distance 2 to 2: 0
Enter the distance 2 to 3: 2
Enter the distance 2 to 4: 3
Enter the distance 2 to 5: 4
Enter the distance 2 to 6: 5
Enter the distance 2 to 7: 6
Enter the distance 2 to 8: 1
Enter the distance 2 to 9: 5
Enter the distance 2 to 10: 8
Enter the distance 3 to 1: 3
Enter the distance 3 to 2: 1
Enter the distance 3 to 3: 0
Enter the distance 3 to 4: 4
Enter the distance 3 to 5: 2
Enter the distance 3 to 6: 6
Enter the distance 3 to 7: 4
Enter the distance 3 to 8: 8
Enter the distance 3 to 9: 5
Enter the distance 3 to 10: 6
Enter the distance 4 to 1: 4
Enter the distance 4 to 2: 3
Enter the distance 4 to 3: 1
Enter the distance 4 to 4: 0
Enter the distance 4 to 5: 5
Enter the distance 4 to 6: 2
Enter the distance 4 to 7: 3
Enter the distance 4 to 8: 6
Enter the distance 4 to 9: 5
Enter the distance 4 to 10: 2
```

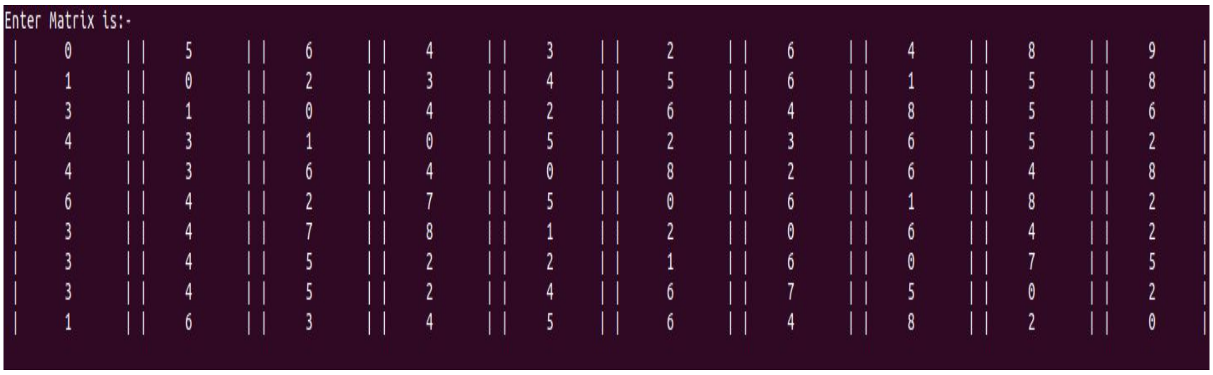


```
Enter the distance 5 to 1: 4
Enter the distance 5 to 2: 3
Enter the distance 5 to 3: 6
Enter the distance 5 to 4: 4
Enter the distance 5 to 5: 0
Enter the distance 5 to 6:
8
Enter the distance 5 to 7: 2
Enter the distance 5 to 8: 6
Enter the distance 5 to 9: 4
Enter the distance 5 to 10: 8
Enter the distance 6 to 1: 6
Enter the distance 6 to 2: 4
Enter the distance 6 to 3: 2
Enter the distance 6 to 4: 7
Enter the distance 6 to 5: 5
Enter the distance 6 to 6: 0
Enter the distance 6 to 7: 6
Enter the distance 6 to 8: 1
Enter the distance 6 to 9: 8
Enter the distance 6 to 10: 2
Enter the distance 7 to 1: 3
Enter the distance 7 to 2: 4
Enter the distance 7 to 3: 7
Enter the distance 7 to 4: 8
Enter the distance 7 to 5: 1
Enter the distance 7 to 6: 2
Enter the distance 7 to 7: 0
Enter the distance 7 to 8: 6
Enter the distance 7 to 9: 4
Enter the distance 7 to 10: 2
Enter the distance 8 to 1: 3
Enter the distance 8 to 2: 4
Enter the distance 8 to 3: 5
Enter the distance 8 to 4: 2
Enter the distance 8 to 5: 2
Enter the distance 8 to 6: 1
Enter the distance 8 to 7: 6
Enter the distance 8 to 8: 0
Enter the distance 8 to 9: 7
Enter the distance 8 to 10: 5
```



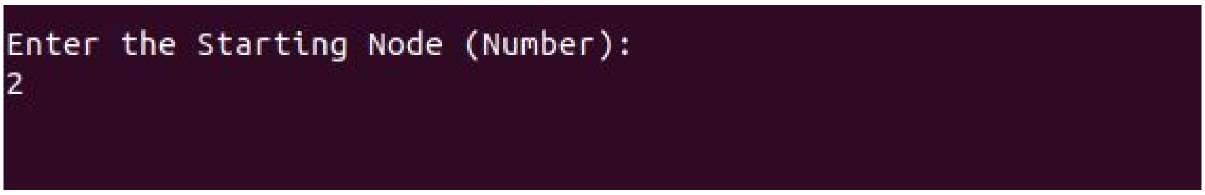
```
Enter the distance 8 to 1: 3
Enter the distance 8 to 2: 4
Enter the distance 8 to 3: 5
Enter the distance 8 to 4: 2
Enter the distance 8 to 5: 2
Enter the distance 8 to 6: 1
Enter the distance 8 to 7: 6
Enter the distance 8 to 8: 0
Enter the distance 8 to 9: 7
Enter the distance 8 to 10: 5
Enter the distance 9 to 1: 3
Enter the distance 9 to 2: 4
Enter the distance 9 to 3: 5
Enter the distance 9 to 4: 2
Enter the distance 9 to 5: 4
Enter the distance 9 to 6: 6
Enter the distance 9 to 7: 7
Enter the distance 9 to 8: 5
Enter the distance 9 to 9: 0
Enter the distance 9 to 10: 2
Enter the distance 10 to 1: 1
Enter the distance 10 to 2: 6
Enter the distance 10 to 3: 3
Enter the distance 10 to 4: 4
Enter the distance 10 to 5: 5
Enter the distance 10 to 6: 6
Enter the distance 10 to 7: 4
Enter the distance 10 to 8: 8
Enter the distance 10 to 9: 2
Enter the distance 10 to 10: 0
```

Output:-



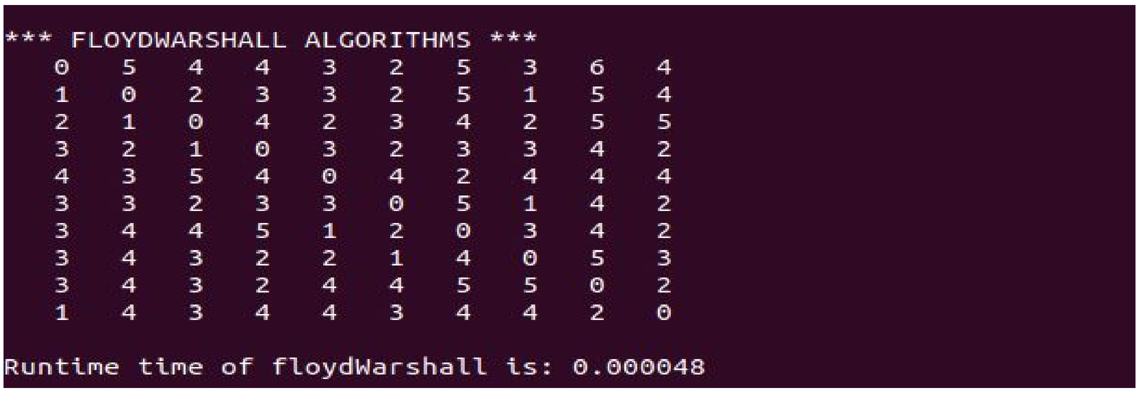
```
Enter Matrix is:-
|   0   ||   5   ||   6   ||   4   ||   3   ||   2   ||   6   ||   4   ||   8   ||   9   |
|   1   ||   0   ||   2   ||   3   ||   4   ||   5   ||   6   ||   1   ||   5   ||   8   |
|   3   ||   1   ||   0   ||   4   ||   2   ||   6   ||   4   ||   8   ||   5   ||   6   |
|   4   ||   3   ||   1   ||   0   ||   5   ||   2   ||   3   ||   6   ||   5   ||   2   |
|   4   ||   3   ||   6   ||   4   ||   0   ||   8   ||   2   ||   6   ||   4   ||   8   |
|   6   ||   4   ||   2   ||   7   ||   5   ||   0   ||   6   ||   1   ||   8   ||   2   |
|   3   ||   4   ||   7   ||   8   ||   1   ||   2   ||   0   ||   6   ||   4   ||   2   |
|   3   ||   4   ||   5   ||   2   ||   2   ||   1   ||   6   ||   0   ||   7   ||   5   |
|   3   ||   4   ||   5   ||   2   ||   4   ||   6   ||   7   ||   5   ||   0   ||   2   |
|   1   ||   6   ||   3   ||   4   ||   5   ||   6   ||   4   ||   8   ||   2   ||   0   |
```

Input:-



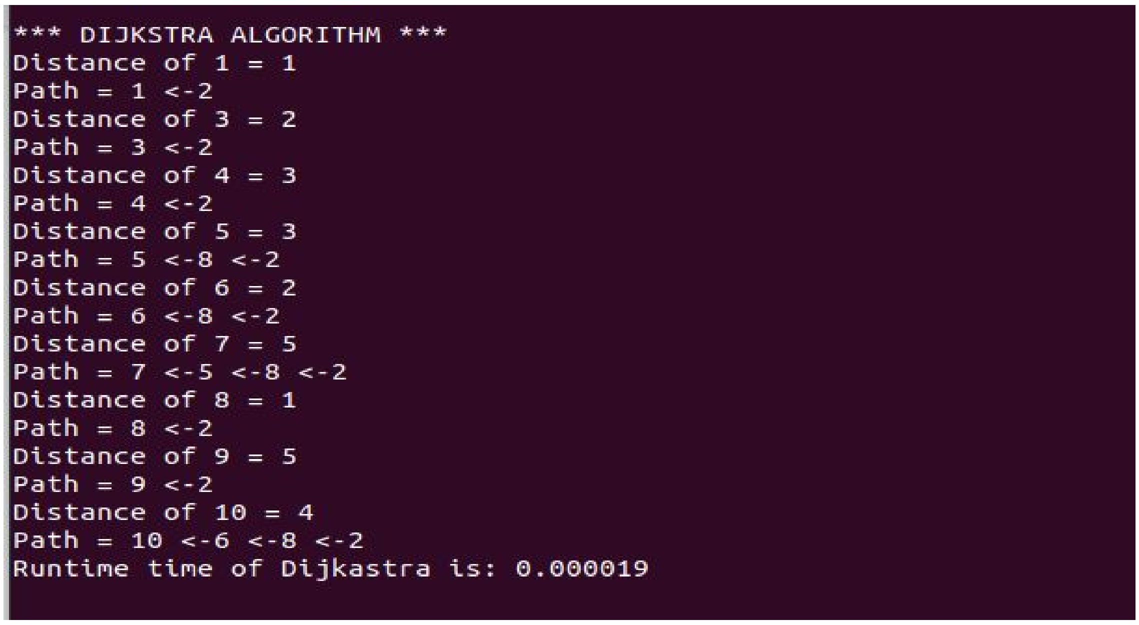
```
Enter the Starting Node (Number):
2
```

Output:-

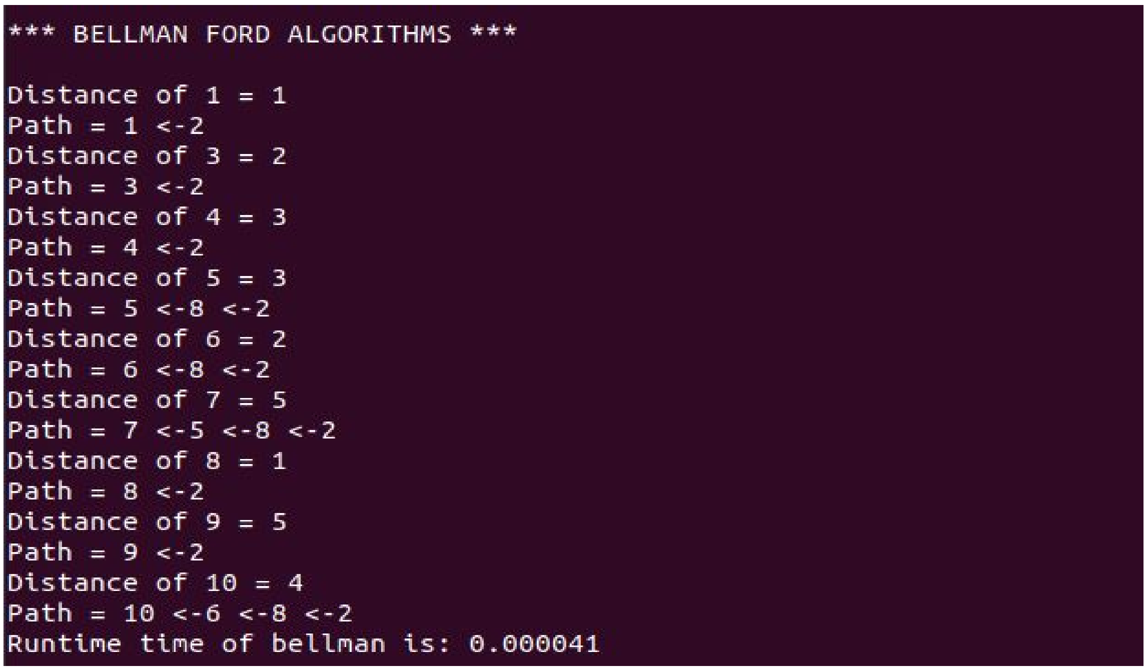


```
*** FLOYDWARSHALL ALGORITHMS ***
   0   5   4   4   3   2   5   3   6   4
   1   0   2   3   3   2   5   1   5   4
   2   1   0   4   2   3   4   2   5   5
   3   2   1   0   3   2   3   3   4   2
   4   3   5   4   0   4   2   4   4   4
   3   3   2   3   3   0   5   1   4   2
   3   4   4   5   1   2   0   3   4   2
   3   4   3   2   2   1   4   0   5   3
   3   4   3   2   4   4   5   5   0   2
   1   4   3   4   4   3   4   4   2   0

Runtime time of floydWarshall is: 0.000048
```



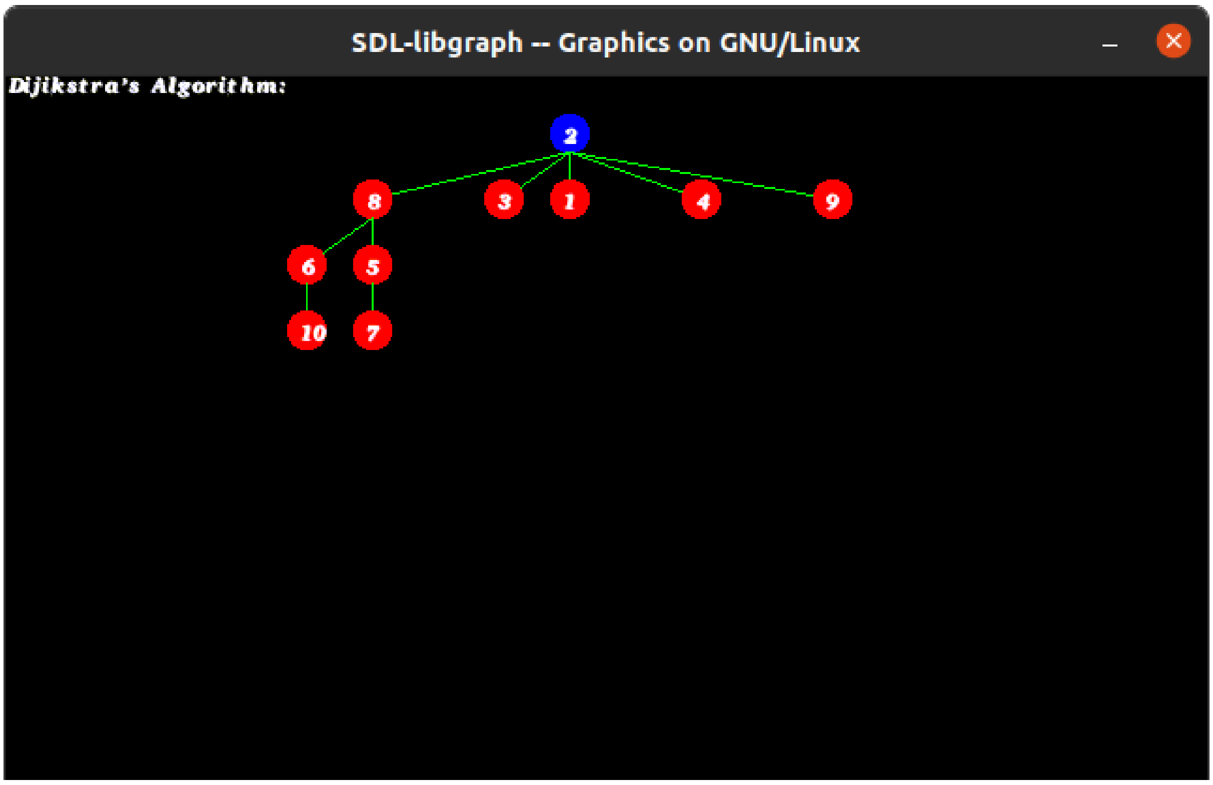
```
*** DIJKSTRA ALGORITHM ***
Distance of 1 = 1
Path = 1 <-2
Distance of 3 = 2
Path = 3 <-2
Distance of 4 = 3
Path = 4 <-2
Distance of 5 = 3
Path = 5 <-8 <-2
Distance of 6 = 2
Path = 6 <-8 <-2
Distance of 7 = 5
Path = 7 <-5 <-8 <-2
Distance of 8 = 1
Path = 8 <-2
Distance of 9 = 5
Path = 9 <-2
Distance of 10 = 4
Path = 10 <-6 <-8 <-2
Runtime time of Dijkastra is: 0.000019
```

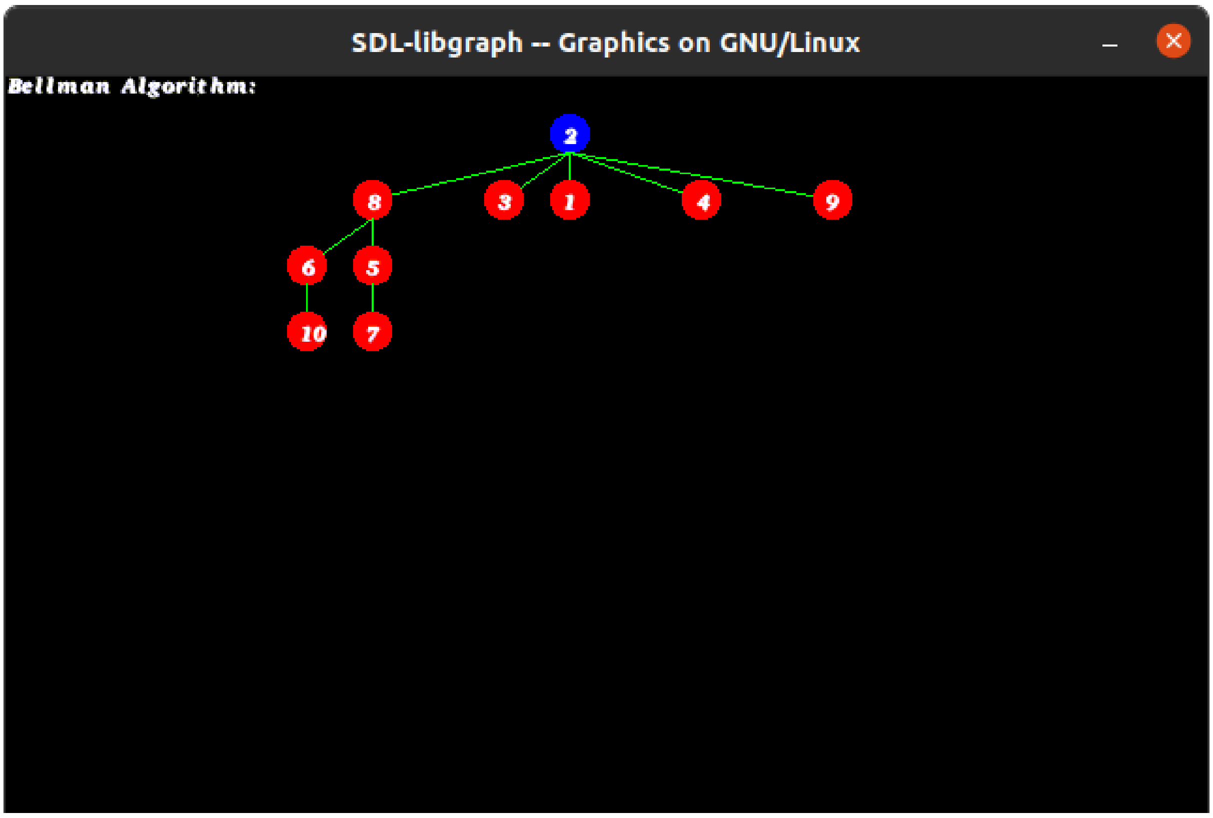


```
*** BELLMAN FORD ALGORITHMS ***

Distance of 1 = 1
Path = 1 <-2
Distance of 3 = 2
Path = 3 <-2
Distance of 4 = 3
Path = 4 <-2
Distance of 5 = 3
Path = 5 <-8 <-2
Distance of 6 = 2
Path = 6 <-8 <-2
Distance of 7 = 5
Path = 7 <-5 <-8 <-2
Distance of 8 = 1
Path = 8 <-2
Distance of 9 = 5
Path = 9 <-2
Distance of 10 = 4
Path = 10 <-6 <-8 <-2
Runtime time of bellman is: 0.000041
```

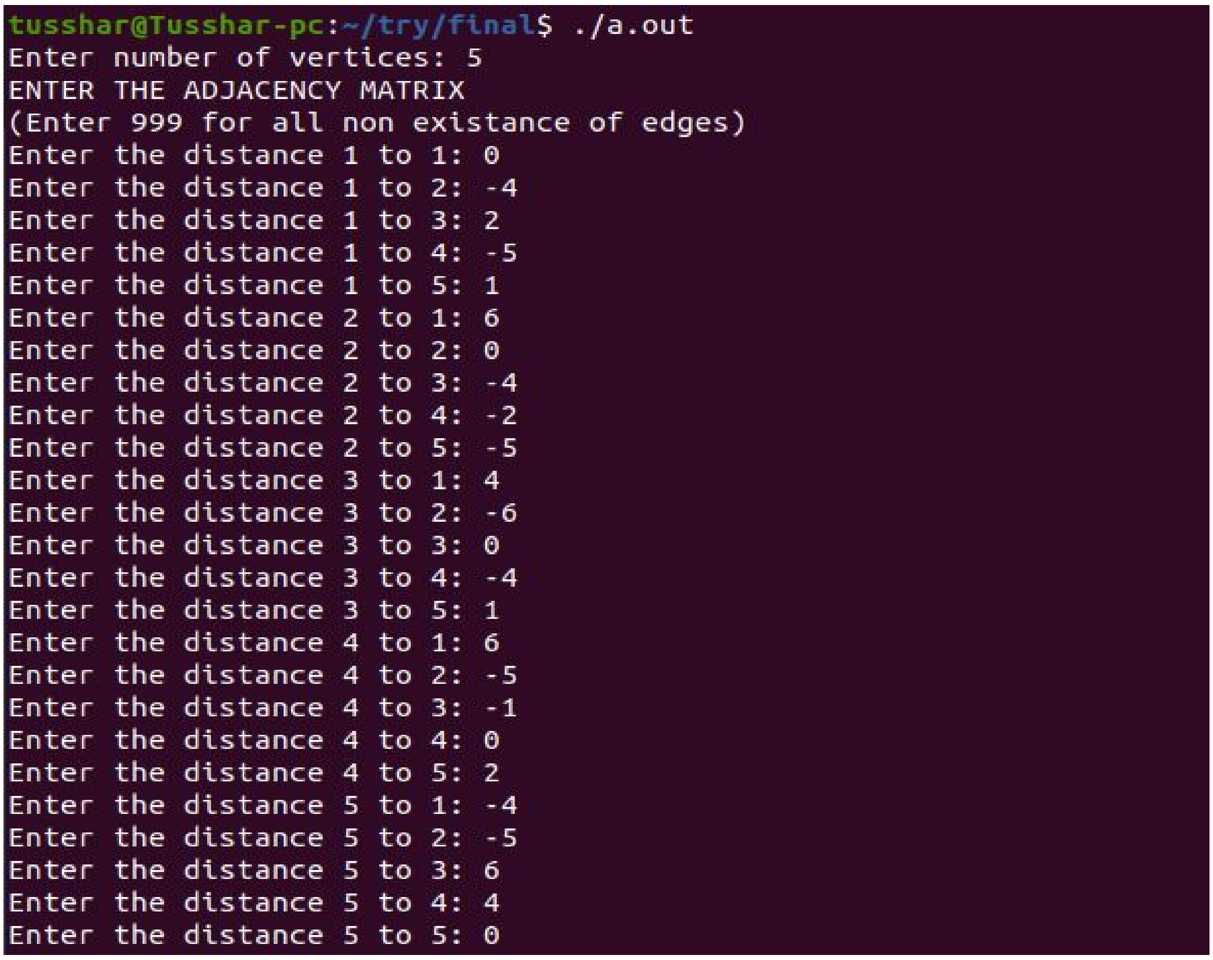
Graphical Output:-





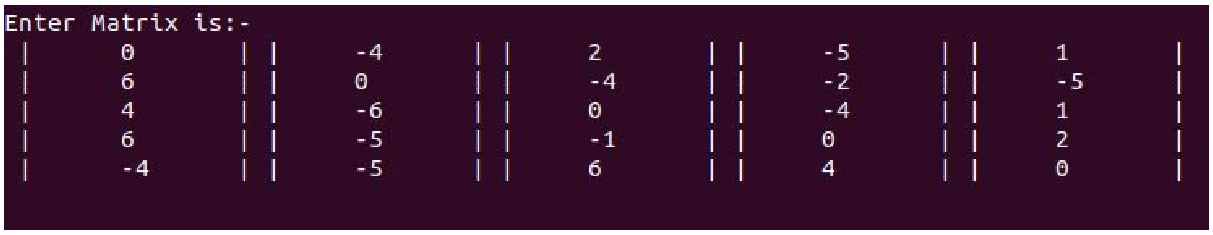
- **Output with Negative weight:**

  Input:



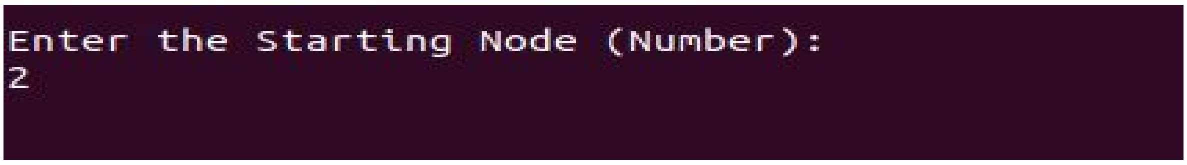
```
tusshar@Tusshar-pc:~/try/final$ ./a.out
Enter number of vertices: 5
ENTER THE ADJACENCY MATRIX
(Enter 999 for all non existance of edges)
Enter the distance 1 to 1: 0
Enter the distance 1 to 2: -4
Enter the distance 1 to 3: 2
Enter the distance 1 to 4: -5
Enter the distance 1 to 5: 1
Enter the distance 2 to 1: 6
Enter the distance 2 to 2: 0
Enter the distance 2 to 3: -4
Enter the distance 2 to 4: -2
Enter the distance 2 to 5: -5
Enter the distance 3 to 1: 4
Enter the distance 3 to 2: -6
Enter the distance 3 to 3: 0
Enter the distance 3 to 4: -4
Enter the distance 3 to 5: 1
Enter the distance 4 to 1: 6
Enter the distance 4 to 2: -5
Enter the distance 4 to 3: -1
Enter the distance 4 to 4: 0
Enter the distance 4 to 5: 2
Enter the distance 5 to 1: -4
Enter the distance 5 to 2: -5
Enter the distance 5 to 3: 6
Enter the distance 5 to 4: 4
Enter the distance 5 to 5: 0
```

Output:-



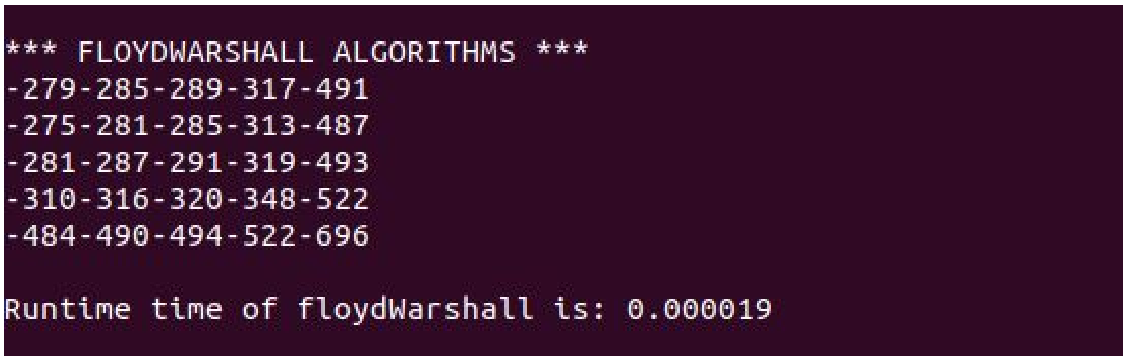
```
Enter Matrix is:-
 |      0       | |     -4      | |     2       | |     -5      | |     1       |
 |      6       | |     0       | |     -4      | |     -2      | |     -5      |
 |      4       | |     -6      | |     0       | |     -4      | |     1       |
 |      6       | |     -5      | |     -1      | |     0       | |     2       |
 |      -4      | |     -5      | |     6       | |     4       | |     0       |
```

Input:-



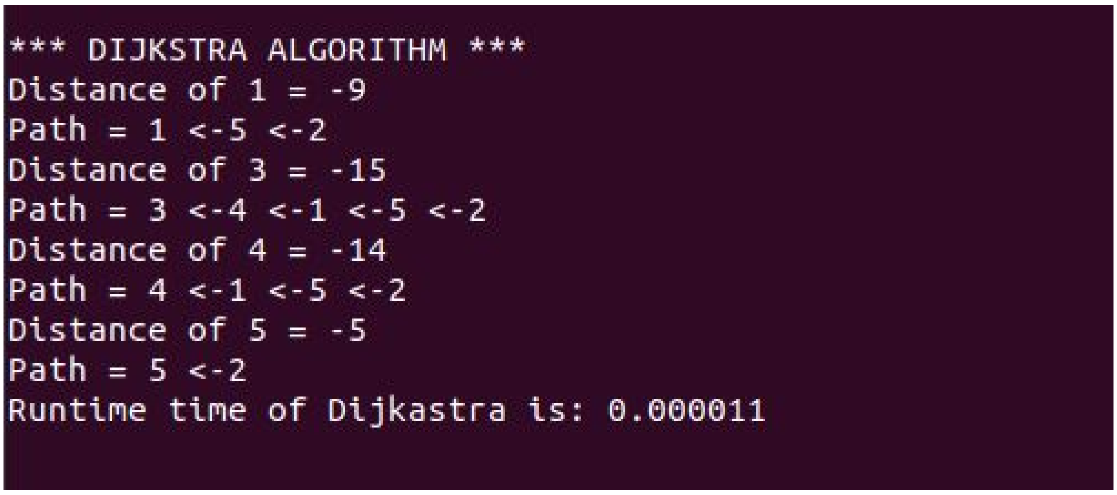
```
Enter the Starting Node (Number):
2
```

Output:-

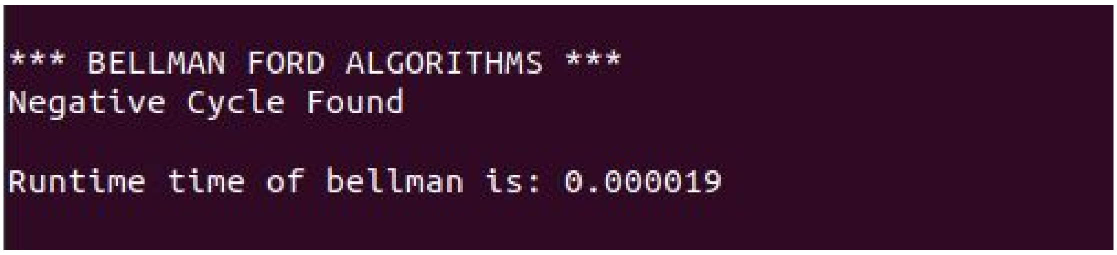


```
*** FLOYDWARSHALL ALGORITHMS ***
-279-285-289-317-491
-275-281-285-313-487
-281-287-291-319-493
-310-316-320-348-522
-484-490-494-522-696

Runtime time of floydWarshall is: 0.000019
```



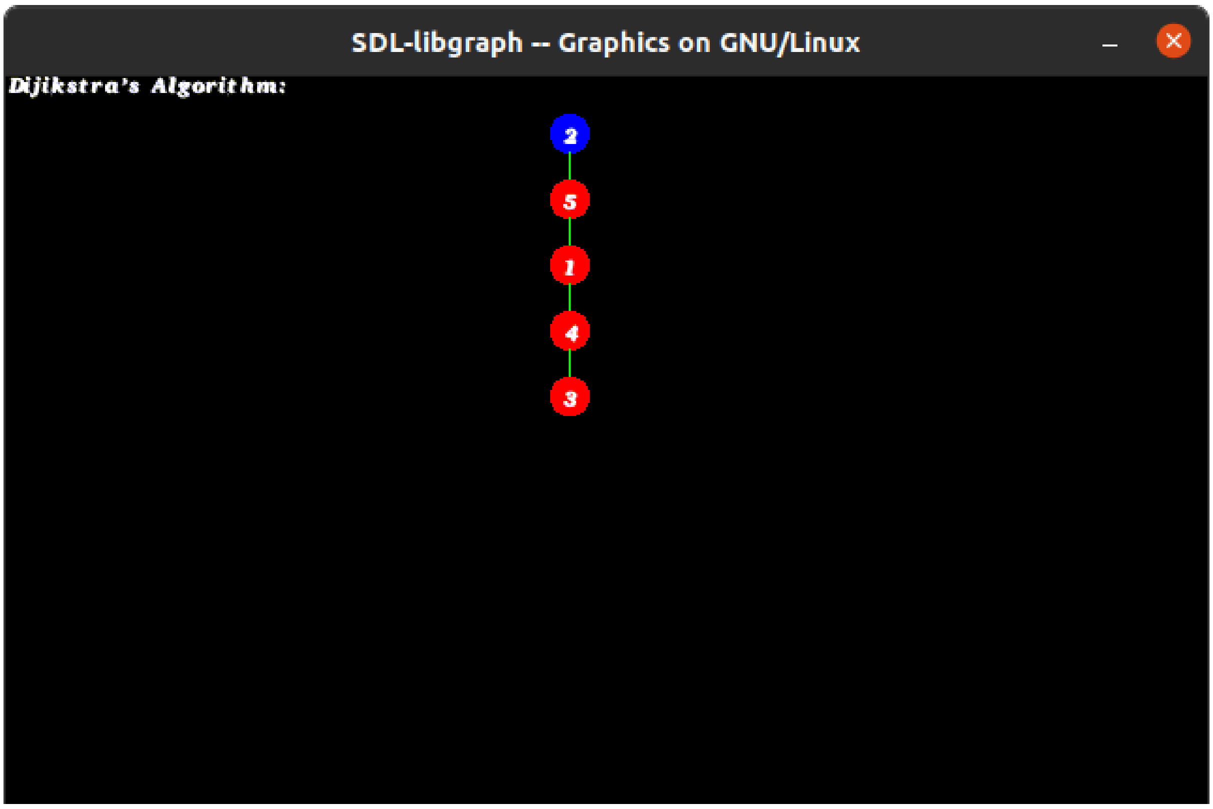
```
*** DIJKSTRA ALGORITHM ***
Distance of 1 = -9
Path = 1 <-5 <-2
Distance of 3 = -15
Path = 3 <-4 <-1 <-5 <-2
Distance of 4 = -14
Path = 4 <-1 <-5 <-2
Distance of 5 = -5
Path = 5 <-2
Runtime time of Dijkastra is: 0.000011
```



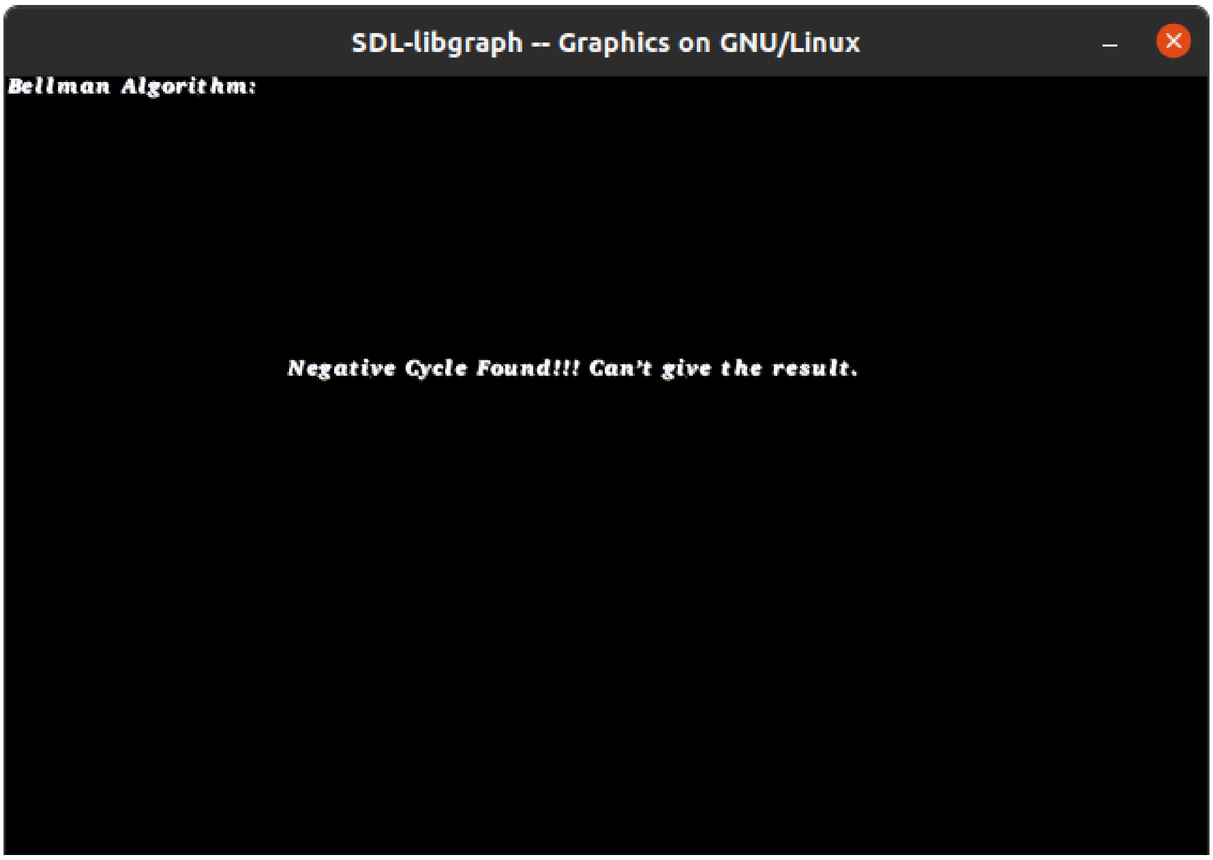
```
*** BELLMAN FORD ALGORITHMS ***
Negative Cycle Found

Runtime time of bellman is: 0.000019
```

Graphical Output:-


SDL-libgraph -- Graphics on GNU/Linux

Dijikstra's Algorithm:


SDL-libgraph -- Graphics on GNU/Linux

Bellman Algorithm:

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