Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL401**	**Course Name:**	**Analysis of Algorithm Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**8**
Title of the Experiment:	**To implement All Pairs Shortest Path Algorithm using Dynamic Method (Floyd Warshall).**
Date of Performance:	**06/03/2025**
Date of Submission:	**13/03/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mrs. Sneha Yadav

Signature :

Date:

Experiment No: 8

Title: All Pair Shortest Path: Floyd Warshall Algorithm

Aim: To study and implement All Pair Shortest Path using Dynamic Programming: Floyd Warshall

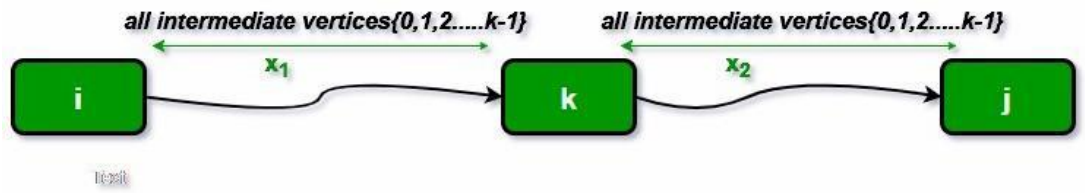
Objective: To introduce Floyd Warshall method

Theory:

The Floyd Warshall Algorithm is an all pair shortest path algorithm unlike Dijkstra and Bellman Ford which are single source shortest path algorithms. This algorithm works for both the directed and undirected weighted graphs can handle graphs with both positive and negative edge weights. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative). It follows Dynamic Programming approach to check every possible path going via every possible node in order to calculate shortest distance between every pair of nodes.

Suppose we have a graph G[][] with V vertices from 1 to N. Now we have to evaluate a shortestPathMatrix[][] where shortestPathMatrix[i][j] represents the shortest path between vertex i to j.

Obviously the shortest path between i to j will have some k number of intermediate nodes. The idea behind floyd warshall algorithm is to treat each and every vertex from 1 to N as an intermediate node one by one.



Algorithm

- Initialize the solution matrix same as the input graph matrix as a first step.
- Then update the solution matrix by considering all vertices as an intermediate vertex.
- The idea is to pick all vertices one by one and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.

- When we pick vertex number **k** as an intermediate vertex, we already have considered vertices **{0, 1, 2, .. k-1}** as intermediate vertices.
- For every pair **(i, j)** of the source and destination vertices respectively, there are two possible cases.
    - **k** is not an intermediate vertex in shortest path from **i** to **j**. We keep the value of **dist[i][j]** as it is.
    - **k** is an intermediate vertex in shortest path from **i** to **j**. We update the value of **dist[i][j]** as **dist[i][k] + dist[k][j],** if **dist[i][j] > dist[i][k] + dist[k][j]**

```
For k = 0 to n – 1
   For i = 0 to n – 1
      For j = 0 to n – 1
         Distance[i, j] = min(Distance[i, j], Distance[i, k] + Distance[k, j])
```

where i = source Node, j = Destination Node, k = Intermediate Node

Implementation:

```
#include <stdio.h>
#define INF 99999  // Large value to represent infinity
int n, i, j, k;
int graph[20][20];
void floydWarshall();
void main() {
   printf("Enter the number of vertices: ");
   scanf("%d", &n);

   printf("Enter the adjacency matrix (Enter %d for infinity):\n", INF);
   for (i = 0; i < n; i++) {
```

```
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    floydWarshall();
}

// Function to implement Floyd-Warshall Algorithm
void floydWarshall() {
    int dist[20][20];

    // Copy the input matrix to the distance matrix
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            dist[i][j] = graph[i][j];
        }
    }

    // Compute shortest paths
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j]
< dist[i][j]) {
```

```
                dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
}

// Print the final shortest distance matrix
printf("\nShortest Distance Matrix:\n");
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        if (dist[i][j] == INF)
            printf("INF\t");
        else
            printf("%d\t", dist[i][j]);
    }
    printf("\n");
}
}
```

Output:

```
Enter the number of vertices: 3
Enter the adjacency matrix (Enter 99999 for infinity):
0 4 11
6 0 2
3 99999 0

Shortest Distance Matrix:
0       4       6
5       0       2
3       7       0
```

Conclusion:

The Floyd-Warshall Algorithm is a dynamic programming technique used to find the shortest paths between all pairs of vertices in a weighted graph. Unlike Dijkstra's or Bellman-Ford algorithms, which solve for a single source, Floyd-Warshall systematically considers every possible intermediate vertex to update the shortest distances between each pair. It works by iteratively checking if a path from vertex `i` to `j` via an intermediate vertex `k` is shorter than the direct path from `i` to `j`, and if so, updates the distance. While effective for graphs with negative weights, it cannot handle graphs with negative weight cycles, as such cycles lead to endlessly decreasing path lengths.