

## Experiment 7: Floyd-Warshall Algorithm

**AIM:** To study & implement the Floyd-Warshall Algorithm

### **THEORY:**

Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).

A weighted graph is a graph in which each edge has a numerical value associated with it.

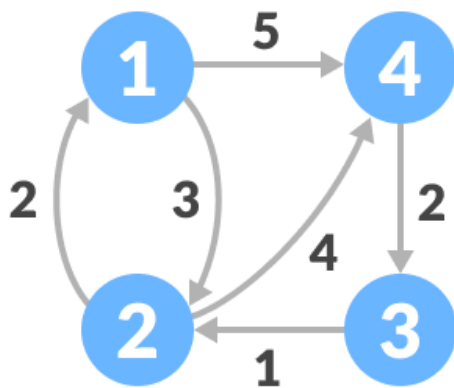
Floyd-Warshall algorithm is also called as Floyd's algorithm, Roy-Floyd algorithm, Roy-Warshall algorithm, or WFI algorithm.

This algorithm follows the [dynamic programming](#) approach to find the shortest paths.

---

### How Floyd-Warshall Algorithm Works?

Let the given graph be:



Initial graph

Follow the steps below to find the shortest path between all the pairs of vertices.

1. Create a matrix  $A_0$  of dimension  $n \times n$  where  $n$  is the number of vertices. The row and the column are indexed as  $i$  and  $j$  respectively.  $i$  and  $j$  are the vertices of the graph.

Each cell  $A[i][j]$  is filled with the distance from the  $i$ th vertex to the  $j$ th vertex. If there is no path from  $i$ th vertex to  $j$ th vertex, the

$$A^0 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{matrix}$$

cell is left as infinity.

Fill each cell with

the distance between  $i$ th and  $j$ th vertex

2. Now, create a matrix  $A_1$  using matrix  $A_0$ . The elements in the first column and the first row are left as they are. The remaining cells are filled in the following way.

Let  $k$  be the intermediate vertex in the shortest path from source to destination. In this step,  $k$  is the first vertex.  $A[i][j]$  is filled with  $(A[i][k] + A[k][j])$  if  $(A[i][j] > A[i][k] + A[k][j])$ .

That is, if the direct distance from the source to the destination is greater than the path through the vertex  $k$ , then the cell is filled with  $A[i][k] + A[k][j]$ .

In this step,  $k$  is vertex 1. We calculate the distance from source vertex to destination vertex through this vertex  $k$ .

$$A^1 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & & \\ \infty & & 0 & \\ \infty & & & 0 \end{bmatrix} \end{matrix} \longrightarrow \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & 9 & 4 \\ \infty & 1 & 0 & 8 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{matrix}$$

Calculate the distance from the source vertex to destination vertex through this vertex  $k$

For example: For  $A_1[2, 4]$ , the direct distance from vertex 2 to 4 is 4 and the sum of the distance from vertex 2 to 4 through vertex (ie. from vertex 2 to 1 and from vertex 1 to 4) is 7. Since  $4 < 7$ ,  $A_0[2, 4]$  is filled with 4.

3. Similarly,  $A_2$  is created using  $A_1$ . The elements in the second column and the second row are left as they are.

In this step,  $k$  is the second vertex (i.e. vertex 2). The remaining

steps are the same as in step 2.

$$A^2 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & & \\ 2 & 0 & 9 & 4 \\ & 1 & 0 & \\ & \infty & & 0 \end{bmatrix} \end{matrix} \longrightarrow \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & 9 & 5 \\ 2 & 0 & 9 & 4 \\ 3 & 3 & 1 & 0 \\ 4 & \infty & \infty & 2 \end{bmatrix} \end{matrix}$$

Calculate the distance from the source vertex to destination vertex through this vertex 2

4. Similarly,  $A_3$  and  $A_4$  is also created.

$$A^3 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & & \infty & \\ & 0 & 9 & \\ \infty & 1 & 0 & 8 \\ & & 2 & 0 \end{bmatrix} \end{matrix} \longrightarrow \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & 9 & 5 \\ 2 & 0 & 9 & 4 \\ 3 & 3 & 1 & 0 \\ 4 & 5 & 3 & 2 \end{bmatrix} \end{matrix}$$

Calculate the distance from the source vertex to destination vertex through this vertex 3

$$A^4 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & & & 5 \\ & 0 & & 4 \\ & & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{matrix} \longrightarrow \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & 7 & 5 \\ 2 & 0 & 6 & 4 \\ 3 & 3 & 1 & 0 \\ 4 & 5 & 3 & 2 \end{bmatrix} \end{matrix}$$

Calculate the distance from the source vertex to destination vertex through this vertex 4

5.  $A_4$  gives the shortest path between each pair of vertices.

---

## Floyd-Warshall Algorithm

**n = no of vertices**

**A = matrix of dimension n\*n**

**for k = 1 to n**

**for i = 1 to n**

**for j = 1 to n**

**Ak[i, j] = min (Ak-1[i, j], Ak-1[i, k] + Ak-1[k, j])**

**return A**

### CODE:

```
#include <stdio.h>
```

```
#define V 4
```

```
#define INF 99999
```

```
void printSolution(int dist[][V]);
```

```
void floydWarshall(int dist[][V]) {
```

```
    int i, j, k;
```

```
    for (k = 0; k < V; k++) {
```

```
        for (i = 0; i < V; i++) {
```

```
            for (j = 0; j < V; j++) {
```

```

        if (dist[i][k] + dist[k][j] < dist[i][j])
            dist[i][j] = dist[i][k] + dist[k][j];
    }
}

printSolution(dist);
}

void printSolution(int dist[][V]) {
    printf("The following matrix shows the shortest distances between
every pair of vertices\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int graph[V][V] = { {0, 5, INF, 10},

```

```
{INF, 0, 3, INF},  
{INF, INF, 0, 1},  
{INF, INF, INF, 0}};
```

```
floydWarshall(graph);  
return 0;  
}
```

### Output:

```
/tmp/UaQahY7KYf.o  
The following matrix shows the shortest distances between every pair of vertices  
    0    5    8    9  
INF    0    3    4  
INF  INF    0    1  
INF  INF  INF    0  
  
=== Code Execution Successful ===
```

### CONCLUSION:

**Successfully Implemented Floyd-Warshall Algorithm**