

## AOA EXPERIMENT 7

**Name: Mohammed Arhaan Ali**

**Div: S11**

**Roll No: 02**

**Aim:** To study and implement the All Pair Shortest Path problem using 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).

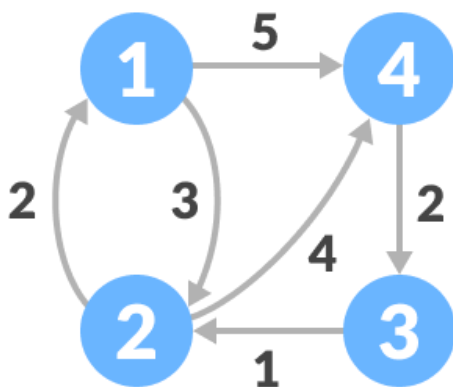
A weighted graph is a graph in which each edge has a numerical value associated with it.

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 does the 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 cell is

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

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.

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

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

4. Similarly, A3 and A4 is also created.

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

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

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

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

5. A4 gives the shortest path between each pair of vertices.

**Program:**

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

```
// defining the number of vertices
```

```
#define nV 4
```

```
#define INF 999
```

```
void printMatrix(int matrix[][nV]);
```

```
// Implementing floyd warshall algorithm
```

```
void floydWarshall(int graph[][nV]) {
```

```

int matrix[nV][nV], i, j, k;

for (i = 0; i < nV; i++)
    for (j = 0; j < nV; j++)
        matrix[i][j] = graph[i][j];

// Adding vertices individually
for (k = 0; k < nV; k++) {
    for (i = 0; i < nV; i++) {
        for (j = 0; j < nV; j++) {
            if (matrix[i][k] + matrix[k][j] < matrix[i][j])
                matrix[i][j] = matrix[i][k] + matrix[k][j];
        }
    }
}
printMatrix(matrix);
}

```

```

void printMatrix(int matrix[][nV]) {
    for (int i = 0; i < nV; i++) {
        for (int j = 0; j < nV; j++) {
            if (matrix[i][j] == INF)
                printf("%4s", "INF");
            else
                printf("%4d", matrix[i][j]);
        }
        printf("\n");
    }
}

```

```

int main() {
    int graph[nV][nV] = {{0, 3, INF, 5},
                        {2, 0, INF, 4},
                        {INF, 1, 0, INF},
                        {INF, INF, 2, 0}};
    floydWarshall(graph);
}

```

## Output:

```
PS C:\Users\arhaa\OneDrive\Desktop\A0A> cd "c:\Users\arhaa\OneDrive\Desktop\A0A"
} ; if ($?) { .\floydwarshall }
0   3   7   5
2   0   6   4
3   1   0   5
5   3   2   0
PS C:\Users\arhaa\OneDrive\Desktop\A0A>
```