Experiment No. 8
To implement All pair shortest Path Algorithm **(Floyd Warshall Algorithm)**
Date of Performance:21/03/2024
Date of Submission:28/03/2024

Experiment No. 8

Title: All Pair Shortest Path

Aim: To study and implement All Pair Shortest Path Algorithm

Objective: To introduce dynamic programming-based algorithm

Theory: The Floyd-Warshall algorithm is a graph algorithm that is deployed to find the shortest path between all the vertices present in a weighted graph. This algorithm is different from other shortest path algorithms; to describe it simply, this algorithm uses each vertex in the graph as a pivot to check if it provides the shortest way to travel from one point to another.

Floyd-Warshall algorithm is one of the methods in All-pairs shortest path algorithms and it is solved using the Adjacency Matrix representation of graphs.

Floyd-Warshall Algorithm

Consider a graph, **G = {V, E}** where **V** is the set of all vertices present in the graph and E is the set of all the edges in the graph. The graph, **G**, is represented in the form of an adjacency matrix, **A**, that contains all the weights of every edge connecting two vertices.

Algorithm:

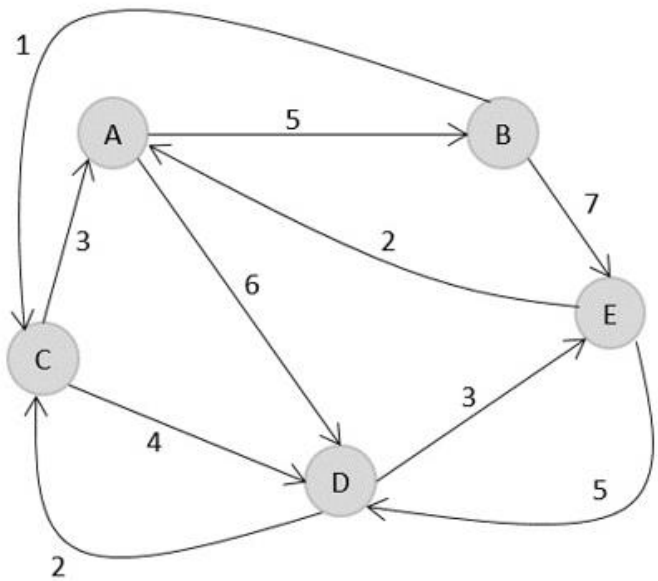
1. Construct an adjacency matrix **A** with all the costs of edges present in the graph. If there is no path between two vertices, mark the value as ∞.
2. Derive another adjacency matrix $\mathbf{A_1}$ from **A** keeping the first row and first column of the original adjacency matrix intact in $\mathbf{A_1}$. And for the remaining values, say $\mathbf{A_1[i,j]}$, if **A[i,j]>A[i,k]+A[k,j]** then replace $\mathbf{A_1[i,j]}$ with **A[i,k]+A[k,j]**. Otherwise, do not change the values. Here, in this step, **k = 1** (first vertex acting as pivot).
3. Repeat **Step 2** for all the vertices in the graph by changing the **k** value for every pivot vertex until the final matrix is achieved.
4. The final adjacency matrix obtained is the final solution with all the shortest paths.

Pseudocode:

```
Floyd-Warshall(w, n){ // w: weights, n: number of vertices
  for i = 1 to n do // initialize, D (0) = [wij]
    for j = 1 to n do{
      d[i, j] = w[i, j];
    }
    for k = 1 to n do // Compute D (k) from D (k-1)
      for i = 1 to n do
        for j = 1 to n do
          if (d[i, k] + d[k, j] < d[i, j]){
            d[i, j] = d[i, k] + d[k, j];
          }
    return d[1..n, 1..n];
}
```

Example:

Consider the following directed weighted graph **G = {V, E}**. Find the shortest paths between all the vertices of the graphs using the Floyd-Warshall algorithm.



Step 1: Construct an adjacency matrix **A** with all the distances as values.

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

Step 2: Considering the above adjacency matrix as the input, derive another matrix A_0 by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$A = \begin{matrix} 0 & 5 & \infty & 6 & \infty \\ \infty & & & & \\ 3 & & & & \\ \infty & & & & \\ 2 & & & & \end{matrix}$$

$$A_1 = \begin{matrix} 0 & 5 & \infty & 6 & \infty \\ \infty & 0 & 1 & \infty & 7 \\ 3 & 8 & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 2 & 7 & \infty & 5 & 0 \end{matrix}$$

Step 3:

Considering the above adjacency matrix as the input, derive another matrix A_0 by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$A_2 = \begin{matrix} & 5 & & & \\ \infty & 0 & 1 & \infty & 7 \\ & 8 & & & \\ & \infty & & & \\ & 7 & & & \end{matrix}$$

$$A_2 = \begin{matrix} 0 & 5 & 6 & 6 & 12 \\ \infty & 0 & 1 & \infty & 7 \\ 3 & 8 & 0 & 4 & 15 \\ \infty & \infty & 2 & 0 & 3 \\ 2 & 7 & 8 & 5 & 0 \end{matrix}$$

Step 4: Considering the above adjacency matrix as the input, derive another matrix A_0 by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$A_3 = \begin{matrix} & & 6 & & \\ & & 1 & & \\ 3 & 8 & 0 & 4 & 15 \\ & & 2 & & \\ & & 8 & & \end{matrix}$$

$$A_3 = \begin{matrix} 0 & 5 & 6 & 6 & 12 \\ 4 & 0 & 1 & 5 & 7 \\ 3 & 8 & 0 & 4 & 15 \\ 5 & 10 & 2 & 0 & 3 \\ 2 & 7 & 8 & 5 & 0 \end{matrix}$$

Step 5: Considering the above adjacency matrix as the input, derive another matrix A_0 by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$A_4 = \begin{matrix} & & & 6 & \\ & & & 5 & \\ & & & 4 & \\ 5 & 10 & 2 & 0 & 3 \\ & & & 5 & \end{matrix}$$

$$A_4 = \begin{matrix} 0 & 5 & 6 & 6 & 9 \\ 4 & 0 & 1 & 5 & 7 \\ 3 & 8 & 0 & 4 & 7 \\ 5 & 10 & 2 & 0 & 3 \\ 2 & 7 & 7 & 5 & 0 \end{matrix}$$

Step 6: Considering the above adjacency matrix as the input, derive another matrix A_0 by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$A_5 = \begin{matrix} & & & & 9 \\ & & & & 7 \\ & & & & 7 \\ & & & & 3 \\ 2 & 7 & 7 & 5 & 0 \end{matrix}$$

$$A_5 = \begin{matrix} 0 & 5 & 6 & 6 & 9 \\ 4 & 0 & 1 & 5 & 7 \\ 3 & 8 & 0 & 4 & 7 \\ 5 & 10 & 2 & 0 & 3 \\ 2 & 7 & 7 & 5 & 0 \end{matrix}$$

Time Complexity Analysis:

The algorithm uses three for loops to find the shortest distance between all pairs of vertices within a graph. Therefore, the **time complexity** is **$O(n^3)$**, where 'n' is the number of vertices in the graph. The **space complexity** of the algorithm is **$O(n^2)$**.

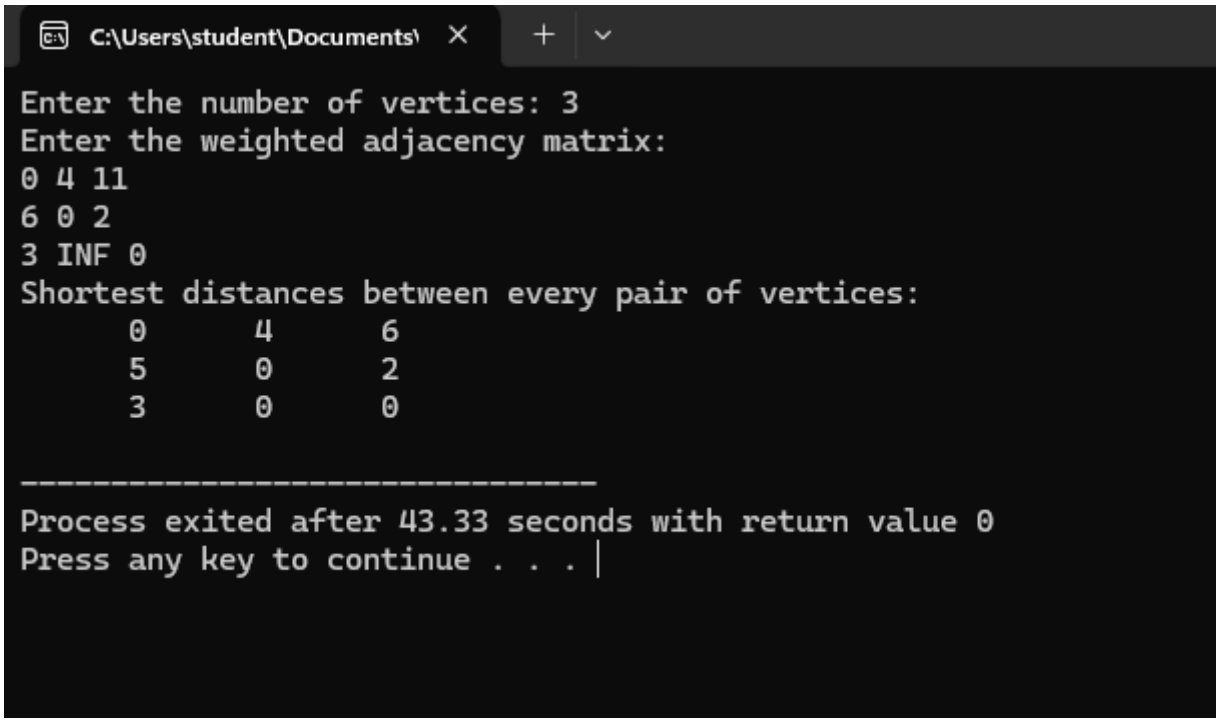
Program:

```
#include <stdio.h>

#define INF 99999
#define MAX_VERTICES 100 // Maximum number of vertices

void printSolution(int dist[][MAX_VERTICES], int V);

void floydWarshall(int graph[][MAX_VERTICES], int V) {
   int dist[MAX_VERTICES][MAX_VERTICES];
   int i, j, k;



   for (i = 0; i < V; i++) {
      for (j = 0; j < V; j++) {
         dist[i][j] = graph[i][j];
      }
   }



   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];
            }
         }
      }
```

```
    }

    // Print the shortest distance matrix
    printSolution(dist, V);
}

void printSolution(int dist[][MAX_VERTICES], int V) {
    printf("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 V;
    printf("Enter the number of vertices: ");
    scanf("%d", &V);

    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the weighted adjacency matrix:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
```

```
            scanf("%d", &graph[i][j]);
        }
    }

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

Output:



Conclusion:

In conclusion, the All Pair Shortest Path algorithm, such as Floyd Warshall, efficiently finds the shortest path between all pairs of vertices in a graph. It accomplishes this by considering all intermediate vertices and updating the distances between pairs of vertices accordingly. The time complexity of the Floyd Warshall algorithm is O(V^3), where V is the number of vertices in the graph. Despite its cubic time complexity, it is suitable for dense graphs or small graphs due to its simplicity and effectiveness in finding all shortest paths.s