| **Experiment No. 8** |
| --- |
| **To implement All pair shortest Path Algorithm**  **(Floyd Warshall Algorithm)** |
| Date of Performance: 4/4/24 |
| Date of Submission: 11/4/24 |

**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 **A1** from **A** keeping the first row and first column of the original adjacency matrix intact in **A1**. And for the remaining values, say **A1[i,j]**, if **A[i,j]>A[i,k]+A[k,j]** then replace **A1[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.

A diagram of a network

Description automatically generated

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

**A number and symbol on a white background

Description automatically generated**

**Step 2:** Considering the above adjacency matrix as the input, derive another matrix A0 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 number and equation on a white background

Description automatically generated with medium confidence**

**Step 3:**

Considering the above adjacency matrix as the input, derive another matrix A0 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 number grid with numbers and equations

Description automatically generated with medium confidence

**Step 4:** Considering the above adjacency matrix as the input, derive another matrix ***A0*** 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 math problem with numbers

Description automatically generated**

**Step 5:** Considering the above adjacency matrix as the input, derive another matrix ***A0*** 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 number grid with numbers and symbols

Description automatically generated with medium confidence**

**Step 6:** Considering the above adjacency matrix as the input, derive another matrix ***A0*** 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 math problem with numbers and equations Description automatically generated

## 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(n3)**, where ‘n’ is the number of vertices in the graph. The **space complexity** of the algorithm is **O(n2)**.

**Program:**

**#include <stdio.h>**

**#define INF 99999**

**#define V 4**

**void floydWarshall(int graph[][V], int vertices) {**

**int dist[V][V];**

**int i, j, k;**

**// Initialize the distance matrix with the given graph**

**for (i = 0; i < vertices; i++)**

**for (j = 0; j < vertices; j++)**

**dist[i][j] = graph[i][j];**

**// Main algorithm loop**

**for (k = 0; k < vertices; k++) {**

**for (i = 0; i < vertices; i++) {**

**for (j = 0; j < vertices; j++) {**

**if (dist[i][k] + dist[k][j] < dist[i][j])**

**dist[i][j] = dist[i][k] + dist[k][j];**

**}**

**}**

**}**

**// Print the shortest distances**

**printf("Shortest distances between every pair of vertices:\n");**

**for (i = 0; i < vertices; i++) {**

**for (j = 0; j < vertices; j++) {**

**if (dist[i][j] == INF)**

**printf("INF ");**

**else**

**printf("%d ", dist[i][j]);**

**}**

**printf("\n");**

**}**

**}**

**int main() {**

**int vertices;**

**printf("Enter the number of vertices in the graph: ");**

**scanf("%d", &vertices);**

**int graph[V][V];**

**printf("Enter the adjacency matrix of the graph:\n");**

**for (int i = 0; i < vertices; i++) {**

**for (int j = 0; j < vertices; j++) {**

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

**if (graph[i][j] == 0 && i != j)**

**graph[i][j] = INF;**

**}**

**}**

**floydWarshall(graph, vertices);**

**return 0;**

**}**

**Output:**

**Enter the number of vertices in the graph: 3**

**Enter the adjacency matrix of the graph:**

**0 4 11**

**6 0 2**

**3 INF 0**

**Shortest distances between every pair of vertices:**

**0 4 6**

**5 0 2**

**3 7 0**

**Conclusion:**

**Pairing shortest path algorithms, like Floyd-Warshall and Johnson's, compute shortest paths between all pairs of vertices in a weighted graph. Floyd-Warshall is straightforward but has a time complexity of O(V^3), suitable for dense graphs. Johnson's algorithm, combining Dijkstra and Bellman-Ford, handles negative weights in O(V^2 log V + VE), making it preferable for sparse graphs.**

**These algorithms are pivotal for network analysis, with Floyd-Warshall excelling in simplicity and Johnson's providing versatility for varied graph structures and edge weights.**