K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch:-B-2	**Roll Number:- 16010122151**

Experiment No._7_

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of All Pair Shortest Path using Dynamic Programming

Objective To learn the All-Pair Shortest Path using Floyd-Warshall's algorithm

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://users.cecs.anu.edu.au/~Alistair.Rendell/Teaching/apac_comp3600/module4/all_pairs_shortest_paths.xhtml**
4. **https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/**
5. **http://www.cs.bilkent.edu.tr/~atat/502/AllPairsSP.ppt**

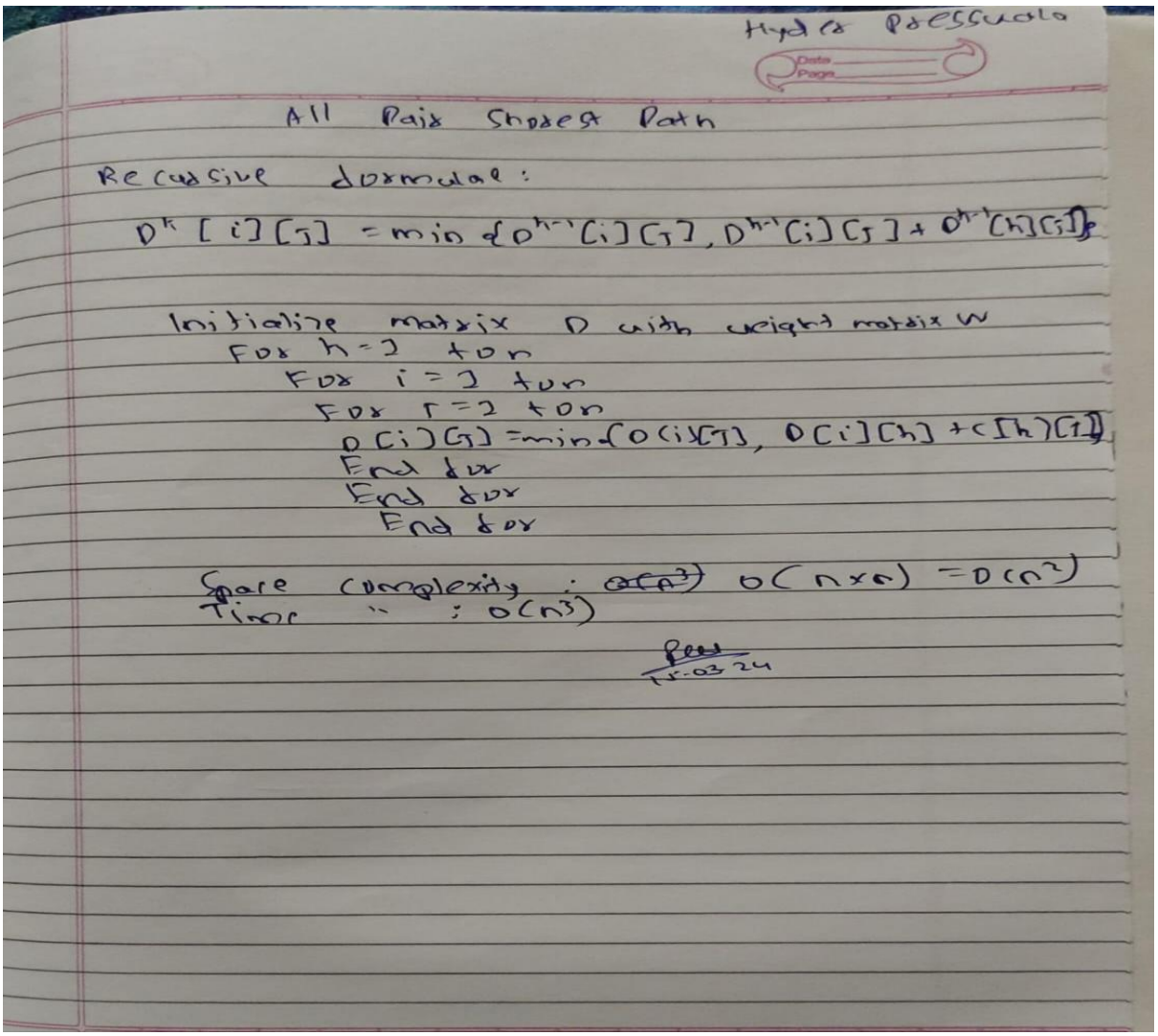
Theory:

It aims to figure out the shortest path from each vertex v to every other u.

1. In all pair shortest path, when a weighted graph is represented by its weight matrix W then objective is to find the distance between every pair of nodes.
2. Apply dynamic programming to solve the all pairs shortest path.
3. In all pair shortest path algorithm, we first decomposed the given problem into sub problems.
4. In this principle of optimally is used for solving the problem.
5. It means any sub path of shortest path is a shortest path between the end nodes.

Algorithm:

```
Algorithm All_pair(W, A)
{
For i = 1 to n do
For j = 1 to n do
A [i , j] = W [i , j]
For k = 1 to n do
        {
        For i = 1 to n do
                {
                For j = 1 to n do
                {
                A [i , j] = min(A [i, j], A [i, k] + A [k, j])
                }
                }}}
```



CODE:-

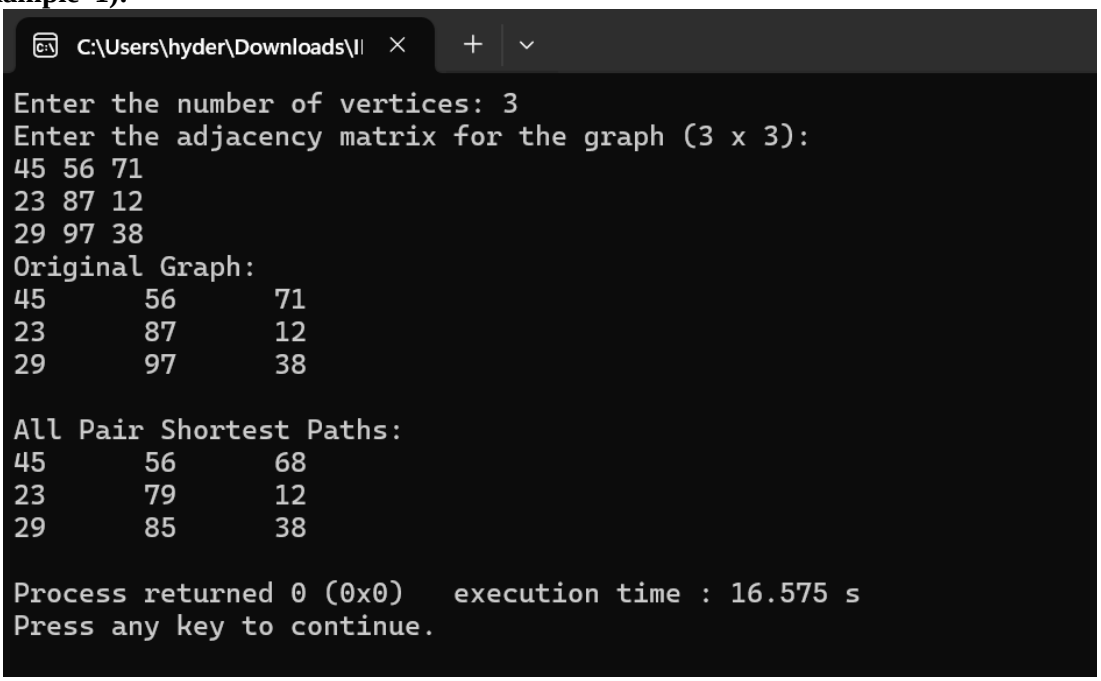
```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

void printGraph(int **graph, int V) {
    printf("Original Graph:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (graph[i][j] == INT_MAX) {
                printf("INF\t");
            } else {
                printf("%d\t", graph[i][j]);
            }
        }
        printf("\n");
    }
}

void printShortestPaths(int **dist, int V) {
    printf("All Pair Shortest Paths:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INT_MAX) {
                printf("INF\t");
            } else {
                printf("%d\t", dist[i][j]);
            }
        }
        printf("\n");
    }
}

void floydWarshall(int **graph, int V) {
    int **dist = (int **)malloc(V * sizeof(int *));
    for (int i = 0; i < V; i++) {
        dist[i] = (int *)malloc(V * sizeof(int));
        for (int j = 0; j < V; j++) {
            if (graph[i][j] == -1) {
                dist[i][j] = INT_MAX; // Treat -1 as infinity
            } else {
                dist[i][j] = graph[i][j];
            }
        }
```

```
    }

    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][k]
+ dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    printGraph(graph, V);
    printf("\n");
    printShortestPaths(dist, V);

    for (int i = 0; i < V; i++) {
        free(dist[i]);
    }
    free(dist);
}

int main() {
    int V;

    printf("Enter the number of vertices: ");
    scanf("%d", &V);

    int **graph = (int **)malloc(V * sizeof(int *));
    for (int i = 0; i < V; i++) {
        graph[i] = (int *)malloc(V * sizeof(int));
    }

    printf("Enter the adjacency matrix for the graph (%d x %d):\n", V, V);
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    floydWarshall(graph, V);

    for (int i = 0; i < V; i++) {
        free(graph[i]);
    }
    free(graph);
```

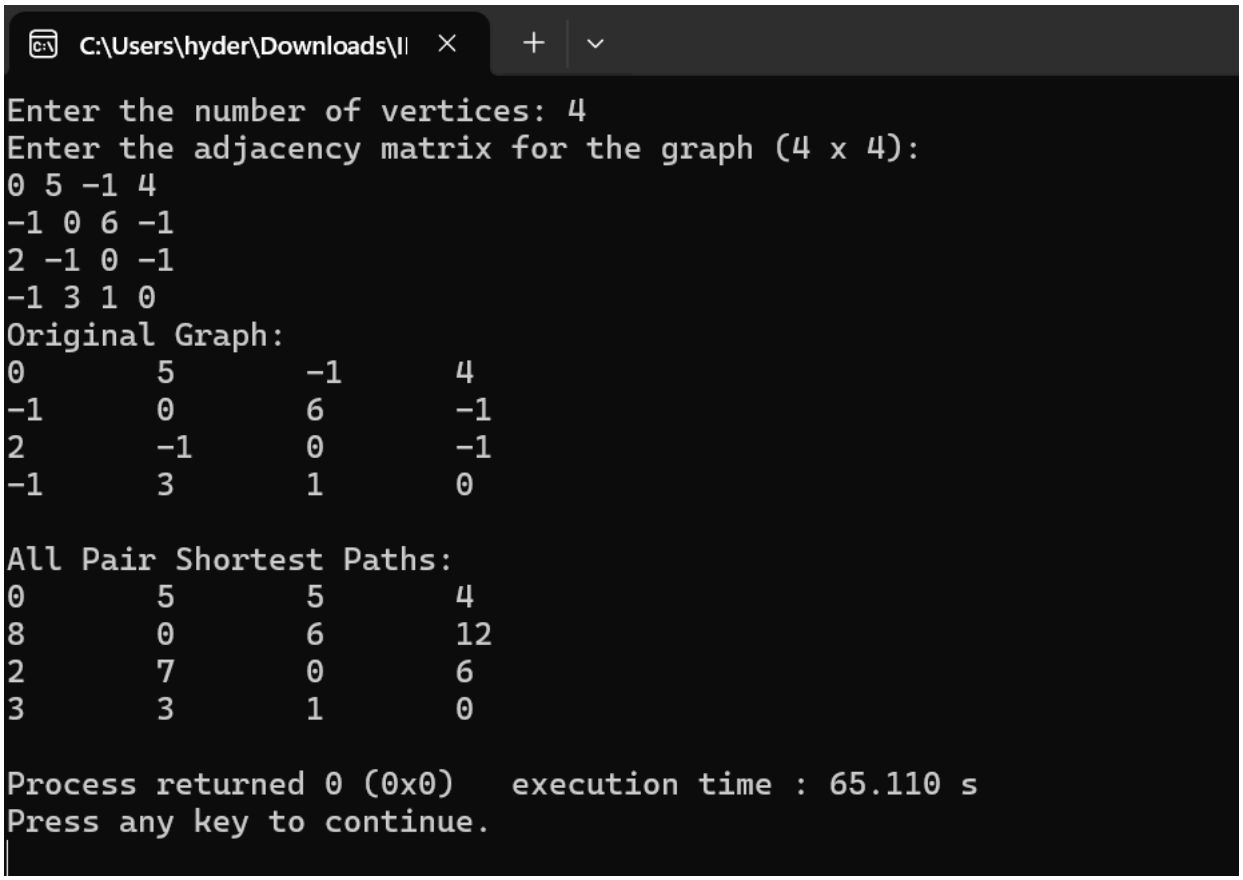
```
    return 0;
}
```

Output :-

Example 1):-



Example 2:-



NOTE:- HERE WE ARE TRAETING (-1) AS INFINITY

CONCLUSION:- The code utilizes the Floyd-Warshall algorithm to find the shortest paths between all pairs of vertices in a directed graph. It prompts the user to input the number of vertices and the adjacency matrix, treating -1 as infinity. The algorithm computes the shortest paths and prints both the original graph and the shortest paths matrix. Proper memory management is implemented to avoid memory leaks.