KJSCE/IT/SYBTech/SEM IV/AA/2020-21



**Experiment No. : 6**

**Title:**Floyd-Warshall Algorithm using Dynamicprogramming approach

(Autonomous College Affiliated to University of Mumbai)

KJSCE/IT/SYBTech/SEM IV/AA/2020-21

**Batch:B2**

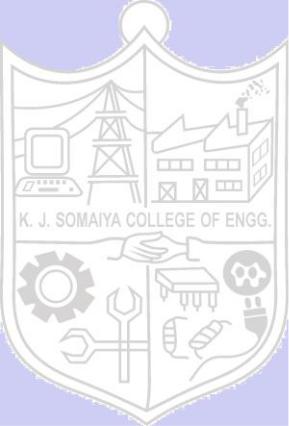
**Roll No.:1914078**

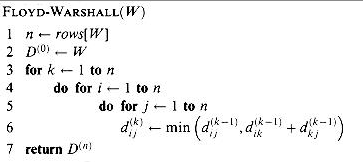
**Experiment No.:6**

**Aim:** Implement Floyd-Warshall Algorithm using Dynamic programming approach.

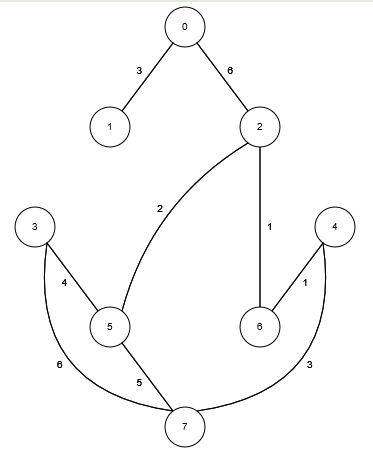


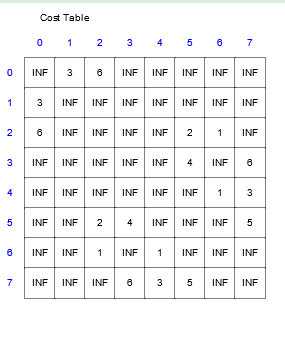
**Algorithm:**



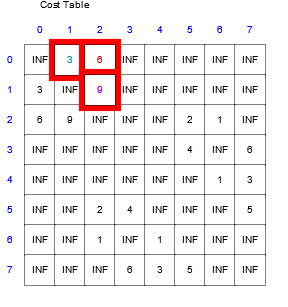


**Example of** Floyd-Warshall Algorithm (All pair shortest path) **Step by Step Execution**

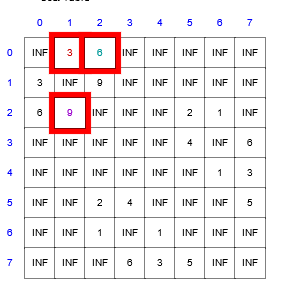




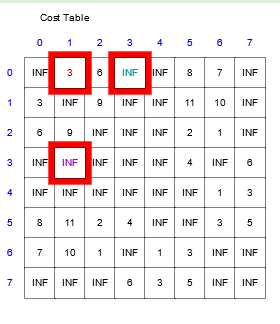
Step 1



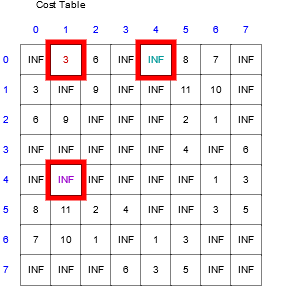
Step 2



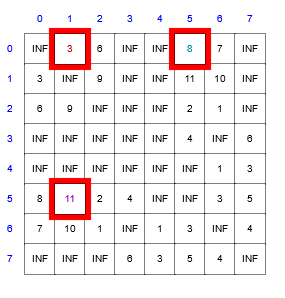
Step 3



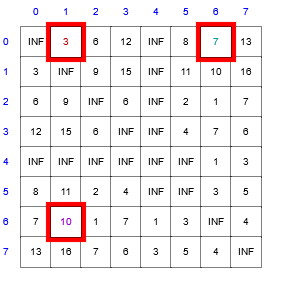
Step 4



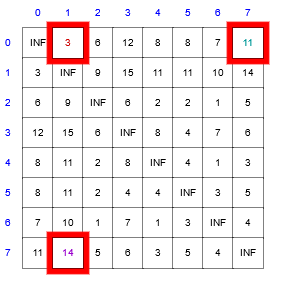
Step 5

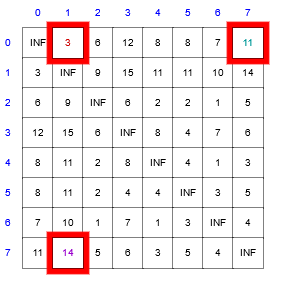


Step 7

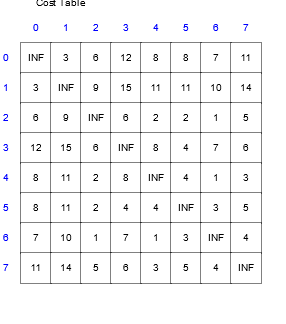


Step 8





Final cost matrix



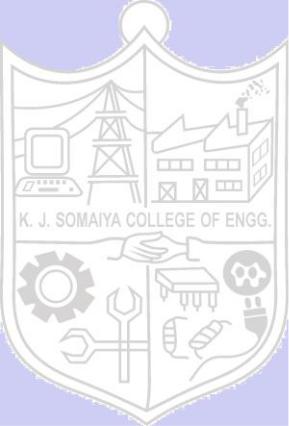
KJSCE/IT/SYBTech/SEM IV/AA/2020-21

**Derivation of Analysis:**

Time complexity.

Floyd Warshall Algorithm is a dynamic programming algorithm with *O*(|*V*|3) time complexity and *O*(|*V*|2) space complexity.

**Program(s):**



#include <bits/stdc++.h>

#define rep(i, n) for (int(i) = 0; (i) < (int)(n); ++(i))

#define reu(i, l, u) for (int(i) = (int)(l); (i) < (int)(u); ++(i))

#define each(it, o) for (auto it = (o).begin(); it != (o).end(); ++it)

#define ios                   \

    freopen("i", "r", stdin); \

    freopen("o", "w", stdout);

#define INF 99999

using namespace std;

void swap(int \*x, int \*y)

{

    int temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void display(int dist[][100], int size, int step)

{

    int n = size;

    cout << "Step " << step << endl;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (dist[i][j] == INF)

                cout << "INF"

                     << "     ";

            else

                cout << dist[i][j] << "     ";

        }

        cout << endl;

    }

}

void floyd(int graph[][100], int size)

{

    int n = size;

    int dist[size][100], i, j, k;

    for (i = 0; i < n; i++)

        for (j = 0; j < n; j++)

            dist[i][j] = graph[i][j];

    for (k = 0; k < n; k++)

    {

        for (i = 0; i < n; i++)

        {

            for (j = 0; j < n; j++)

            {

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

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

            }

        }

        display(dist, n, k + 1);

    }

}

int main()

{

    ios int n;

    cout << "Enter number: ";

    cin >> n;

    int graph[100][100];

    rep(i, n)

    {

        rep(j, n)

        {

            if (i == j)

            {

                graph[i][j] = 0;

                continue;

            }

            graph[i][j] = INF;

        }

    }

    int a, b, weight;

    while (cin >> a >> b >> weight)

    {

        if (a == b)

            continue;

        if (a < 0 || a >= n)

            continue;

        if (b < 0 || b >= n)

            continue;

        graph[a][b] = weight;

        graph[b][a] = weight;

    }

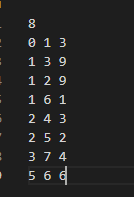
    floyd(graph, n);

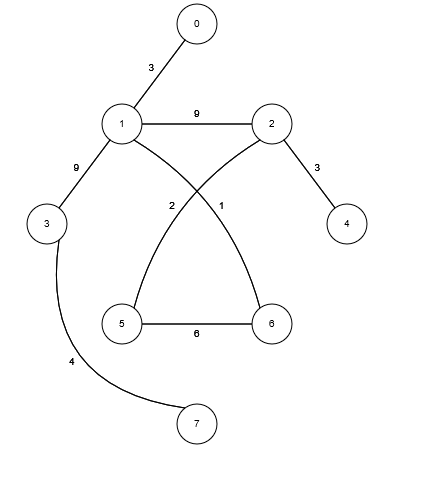
    return 0;

}

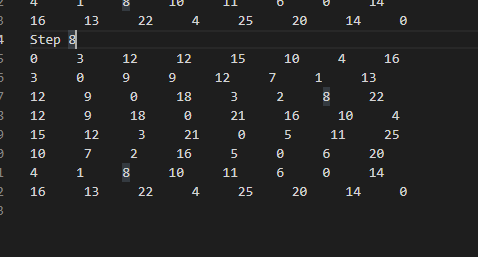
**Output(o):**

Input





Output



KJSCE/IT/SYBTech/SEM IV/AA/2020-21



**Questions:-** Explain dynamic programming approach for Floyd-Warshall algorithm and write the various applications of it.

**ANS.**

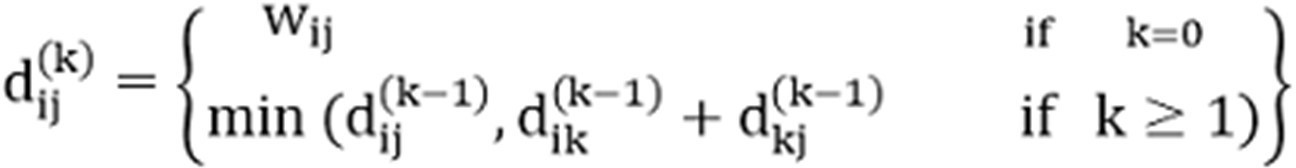
Let the vertices of G be V = {1, 2........n} and consider a subset {1, 2 ....... k} of vertices for some k. For any pair of vertices i, j  V, considered all paths from i to j whose intermediate vertices are all drawn from {1, 2 ...... k}, and let p be a minimum weight path from amongst them. The Floyd- Warshall algorithm exploits a link between path p and shortest paths from i to j with all intermediate vertices in the set {1, 2 ...... k-1}. The link depends on whether or not k is an intermediate vertex of path p.

If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set {1, 2 ...... k-1}. Thus, the shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2.......k-1} is also the shortest path i to j with all intermediate vertices in the set {1, 2 ...... k}.

If k is an intermediate vertex of path p, then we break p down into i → k → j.

Let dij(k) be the weight of the shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2 .. k}.

A recursive definition is given by



***Applications:***

The Floyd–Warshall algorithm can be used to solve the following problems:

* Finding a regular expression denoting the regular language accepted by a finite automaton (Kleene's algorithm, a closely related generalization of the Floyd–Warshall algorithm
* Inversion of real matrices (Gauss–Jordan algorithm)
* Optimal routing. In this application one is interested in finding the path with the maximum flow between two vertices. This means that, rather than taking minima as in the pseudocode above, one instead takes maxima. The edge weights represent fixed constraints on flow. Path weights represent bottlenecks; so the addition operation above is replaced by the minimum operation.
* Fast computation of Pathfinder networks.
* Widest paths/Maximum bandwidth paths
* Computing canonical form of difference bound matrices (DBMs)
* Computing the similarity between graphs
* Transitive closure in AND/OR/threshold graphs



**Outcome:**

Implement Greedy and Dynamic Programming algorithms

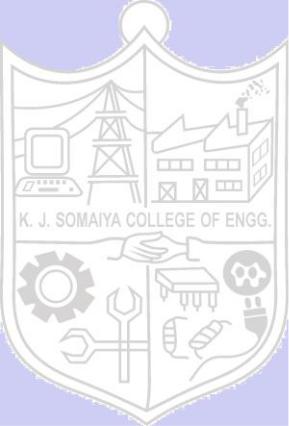
**Conclusion: (Based on the observations)**



Learned about Floyd Warshall Algorithm and used it to find shortest distances between all pairs of vertices

**References:**

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones &Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication



(Autonomous College Affiliated to University of Mumbai)