G. H. RAISONI COLLEGE OF ENGG., NAGPUR
(An Autonomous Institute under UGC Act 1956)
Department of Computer Science & Engg.

Date: 08/08/2020

Practical Subject: Design and Analysis of Algorithms
Session: 2020-21

Student Details:

Roll Number	58
Name	Shivam Tawari
Semester	3
Section	A
Branch	Artificial Intelligence

Practical Details: Practical Number- 7

Practical Aim	To implement and analyze time complexity of Single source shortest path Algorithm.
Theory & Algorithm	**Theory:** ***Single Source Shortest Path:*** The shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. The problem of finding the shortest path between two intersections on a road map may be modeled as a special case of the shortest path problem in graphs, where the vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of the segment. ***Dijkstra's Algorithm:***

	Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source. **Algorithm:** Step 1: START Step 2: Input Number of Vertices and Source Node Step 3: Input Graph Step 4: Create an empty set sptSet Step 5: Initialize all distance value to infinity. Set distance value of source node to 0 Step 6: Select a vertex u not in sptSet having min distance value Step 7: Add vertex to sptSet Step 8: Update distance value of all adjacent vertices of u Step 9: Display Path Step 10: STOP
Complexity	Time Complexity of Dijkstra's Algorithm: O (V^2)
Program	

main.cpp

Run

```
#include<iostream>
#include<stdio.h>
using namespace std;
#define INFI 9999
#define max 9

void dpathcal(int a[max][max], int n, int source) {
    int c[max][max], dist[max], pred[max];
    int done[max], flag, mdis, nextnode;

    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            if(a[i][j] == 0)
                c[i][j] = INFI;
            else
                c[i][j] = a[i][j];
    for(int i=0; i<n; i++)
    {
        dist[i] = c[source][i];
        pred[i] = source;
        done[i] = 0;
    }
    dist[source] = 0;
    done[source] = 1;
    flag = 1;
    while(flag < n-1) {
```

```
        mdis = INFI;
        for(int i=0; i<n; i++)
            if(dist[i]<mdis && !done[i]) {
                mdis = dist[i];
                nextnode = i;
            }
        done[nextnode] = 1;
        for(int i=0; i<n; i++)
            if(!done[i])
                if(mdis+c[nextnode][i] < dist[i]) {
                    dist[i] = mdis + c[nextnode][i];
                    pred[i] = nextnode;
                }
        flag++;
    }
    for(int i=0;i<n;i++)
        if(i!=source) {
            cout << endl;
            cout << "\n Vertex " << i << ": " << dist[i];
            cout << "\n *Path to vertex*: " << i;
            int j = i;
            do {
                j = pred[j];
                cout << " <- " << j;
            } while(j != source);
    }

}
int main()
{
    cout << "\nName: Shivam Tawari";
    cout << "\nSection: A";
    cout << "\nRoll Number: 58";
    int graph[max][max];
    int n, source;
    cout << "\nEnter Vertices and Source: ";
    cin >> n >> source;
    cout << " Enter Adjacency Matrix(" << n << "x" << n << "): ";
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            cin >> graph[i][j];
        }
    }
    dpathcal(graph, n, source);
    return 0;
}
```

Output	Output Clear g++ -o /tmp/6m8VY0duxJ.o /tmp/6m8VY0duxJ.cpp /tmp/6m8VY0duxJ.o Name: Shivam Tawari Section: A Roll Number: 58 Enter Vertices and Source: 3 0 Enter Adjacency Matrix(3x3): 2 6 3 5 2 0 3 6 4 Vertex 1: 6 *Path to vertex*: 1 <- 0 Vertex 2: 3 *Path to vertex*: 2 <- 0