Name: Rishi Tiku

Class: SE DS

UID: 2021700067

Subject: DAA LAB

Experiment: 7

Aim: To implement Dijkstra's algorithm for finding shortest distance between two vertices

Problem:

Given an undirected graph with 'v' vertices, assume the source as the 0^{th} vertex and find the shortest distance from the source to each vertex using Dijkstra's algorithm.

Theory:

Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph.

Dijkstra's Algorithm works on the basis that any sub-path `B -> D` of the shortest path `A -> D` between vertices A and D is also the shortest path between vertices B and D.

Dijkstra used this property in the opposite direction i.e., we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbours to find the shortest sub-path to those neighbours.

The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

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 a given source as a root. Maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, find a vertex that is in the other set (set not yet included) and has a minimum distance from the source.

Time Complexity: $O(V^2)$
Auxiliary Space: O(V)

Algorithm

- Create a set **sptSet** (shortest path tree set) that keeps track of vertices included in the shortest path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
- Assign a distance value to all vertices in the input graph. Initialize all distance values as **INFINITE**. Assign the distance value as 0 for the source vertex so that it is picked first.
- While **sptSet** doesn't include all vertices
    - Pick a vertex **u** that is not there in **sptSet** and has a minimum distance value.
    - Include u to **sptSet**.
    - Then update the distance value of all adjacent vertices of u.
        - To update the distance values, iterate through all adjacent vertices.
        - For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Code

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

int minDistance(int V, int dist[], bool sptSet[])
{
    int min = INT_MAX, min_index;

    for (int ver = 0; ver < V; ver++)
    {
        if (sptSet[ver] == false && dist[ver] <= min)
        {
            min = dist[ver];
            min_index = ver;
        }
    }
    return min_index;
}

void printSolution(int V, int dist[])
{
    printf("\n\nVertex \t\t| Distance from Source\n");
    printf("-----------------------------------------\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t|\t %d\n", i, dist[i]);
}

void dijkstra(int V, int weight[V][V], int src)
{
    int dist[V];
    bool sptSet[V];
    for (int i = 0; i < V; i++)
    {
        dist[i] = INT_MAX, sptSet[i] = false;
    }

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(V, dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
        {
            if (!sptSet[v] && weight[u][v] && dist[u] != INT_MAX && dist[u] +
weight[u][v] < dist[v])
            {
```

```
                dist[v] = dist[u] + weight[u][v];
            }
        }
    }

    printSolution(V, dist);
}

int main()
{
    int s, e, w, v, ch;

    printf("Enter no. of vertices: ");
    scanf("%d", &v);

    int weight[v][v];

    printf("\nEnter edges and their weights:-");
    do{
        printf("\nStart Point: ");
        scanf("%d",&s);
        printf("End Point: ");
        scanf("%d",&e);
        printf("Weight: ");
        scanf("%d",&w);
        weight[s][e] = w;
        weight[e][s] = w;
        printf("\nEnter 0 to stop: ");
        scanf("%d",&ch);
    }while(ch!= 0);

    dijkstra(v, weight, 0);
    return 0;
}

//5 0 1 2 1 0 3 6 1 1 2 3 1 1 3 8 1 1 4 5 1 2 4 7 1 3 4 9 0
```

Output

```
Enter no. of vertices: 5

Enter edges and their weights:-
Start Point: 0
End Point: 1
Weight: 2

Enter 0 to stop: 1

Start Point: 0
End Point: 3
Weight: 6

Enter 0 to stop: 1

Start Point: 1
End Point: 2
Weight: 3

Enter 0 to stop: 1
```

```
Start Point: 1
End Point: 3
Weight: 8

Enter 0 to stop: 1

Start Point: 1
End Point: 4
Weight: 5

Enter 0 to stop: 1

Start Point: 2
End Point: 4
Weight: 7

Enter 0 to stop: 1

Start Point: 3
End Point: 4
Weight: 9

Enter 0 to stop: 0
```

Vertex	Distance from Source
0	0
1	2
2	5
3	6
4	7

Conclusion

Shortest distance for each vertex from the source in an undirected graph can be thus obtained using Dijkstra’s algorithm.