**Experiment 3.2**

**Student Name: Sahul Kumar Parida UID: 20BCS4919**

**Branch: CSE Section/Group: WM-904/B**

**Semester: 5th Date of Performance: 08/11/2022**

**Subject Name: Design and Analysis Algorithm Lab**

**Subject Code: 20CSP-312**

1. **Aim/Overview of the practical:**

Code and analyze to find the shortest paths in a graph with positive edge weights using Dijkstra’s algorithm.

1. **Task to be done/which logistics used:**

To implement Dijkstra’s algorithm to find the shortest paths in a graph with positive edge.

Dijkstra’s algorithm is also known as the shortest path algorithm. It is an algorithm used to find the shortest path between nodes of the graph. The algorithm creates the tree of the shortest paths from the starting source vertex from all other points in the graph. It differs from the minimum spanning tree as the shortest distance between two vertices may not be included in all the vertices of the graph. The algorithm works by building a set of nodes that have a minimum distance from the source. Here, Dijkstra's algorithm uses a greedy approach to solve the problem and find the best solution.

1. **Algorithm/Flowchart:**
2. First of all, we will mark all vertex as unvisited vertex
3. Then, we will mark the source vertex as 0 and all other vertices as infinity
4. Consider source vertex as current vertex
5. Calculate the path length of all the neighbouring vertex from the current vertex by adding the weight of the edge in the current vertex
6. Now, if the new path length is smaller than the previous path length then replace it otherwise ignore it
7. Mark the current vertex as visited after visiting the neighbour vertex of the current vertex
8. Select the vertex with the smallest path length as the new current vertex and go back to step 4.
9. Repeat this process until all the vertex are marked as visited.
10. **Steps for experiment/practical/Code:**

#include <bits/stdc++.h>

using namespace std;

int miniDist(int distance[], bool Tset[]) // finding minimum distance

{

int minimum=INT\_MAX,ind;

for(int k=0;k<6;k++)

{

if(Tset[k]==false && distance[k]<=minimum)

{

minimum=distance[k];

ind=k;

}

}

return ind;

}

void DijkstraAlgo(int graph[6][6],int src) // adjacency matrix

{

int distance[6]; // // array to calculate the minimum distance for each node

bool Tset[6];// boolean array to mark visited and unvisited for each node

for(int k = 0; k<6; k++)

{

distance[k] = INT\_MAX;

Tset[k] = false;

}

distance[src] = 0; // Source vertex distance is set 0

for(int k = 0; k<6; k++)

{

int m=miniDist(distance,Tset);

Tset[m]=true;

for(int k = 0; k<6; k++)

{

// updating the distance of neighbouring vertex

if(!Tset[k] && graph[m][k] && distance[m]!=INT\_MAX && distance[m]+graph[m][k]<distance[k])

distance[k]=distance[m]+graph[m][k];

}

}

cout<<"Vertex\t\tDistance from source vertex"<<endl;

for(int k = 0; k<6; k++)

{

char str=65+k;

cout<<str<<"\t\t\t"<<distance[k]<<endl;

}

}

int main()

{

cout << "Sahul Kumar Parida" << endl;

cout << "20BCS4919" << endl;

int graph[6][6]={

{0, 1, 2, 0, 0, 0},

{1, 0, 0, 5, 1, 0},

{2, 0, 0, 2, 3, 0},

{0, 5, 2, 0, 2, 2},

{0, 1, 3, 2, 0, 1},

{0, 0, 0, 2, 1, 0}};

DijkstraAlgo(graph,0);

return 0;

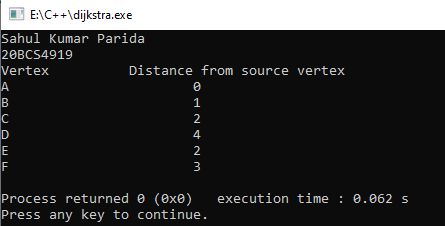
}

1. **Observations/Discussions/ Complexity Analysis:**

The time complexity of Dijkstra's algorithm is O(V^2), where V is the number of vertices in the graph. However, if the input graph is represented using an adjacency list (method of representation of graph), then the time complexity can be reduced to O(E log V) using a binary heap.

The space complexity of Dijkstra's algorithm is O(V), where V is the total number of vertices of the graph. This is because we have to store all these vertices in the list as an output.

1. **Output:**



**Learning outcomes (What I have learnt):**

1. Graphs.
2. Dijkstra’s algorithm.
3. Graphs are used as a connection between objects, people, or entities, and Dijkstra's algorithm will help you find the shortest distance between two points in a graph.