

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

**Experiment-3.2**

**Student Name: Rajiv Paul UID: 20BCS1812**

**Branch: CSE Section/Group: 20BCS\_WM-702-A**

**Semester: 5th Date of Performance:31-10-2022**

**Subject Name: DAA lab Subject Code: 20-CSP-312**

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

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

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

Find shortest path in graph with positive edge weights using Dijkstra’s algorithm.

**3. Algorithm/Flowchart (For programming based labs):**

* 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 which is not there in sptSet and has a minimum distance value.
* Include u to sptSet.
* Then update 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.

**4. Steps for experiment/practical/Code:**

#include <bits/stdc++.h>

using namespace std;

vector<int> dijkstra(vector<vector<int>> &vec, int vertices, int edges, int source) {

unordered\_map<int, list<pair<int, int>>> adj; for (int i = 0; i < edges; i++)

{

int u = vec[i][0]; int v = vec[i][1]; int dist = vec[i][2];

// pair<int,int> p1 = make\_pair(u,dist),p2=make\_pair(v,dist);

adj[u].push\_back(make\_pair(v, dist)); adj[v].push\_back(make\_pair(u, dist));

}

vector<int> dist(vertices, INT\_MAX); set<pair<int, int>> st;

dist[source] = 0; st.insert(make\_pair(0, source)); while (!st.empty())

{

// fetch top record

auto top = \*(st.begin()); int nodeDist = top.first; int node = top.second;

// delete top st.erase(st.begin()); // traverse the adj

for (auto it : adj[node]) {

if (nodeDist + it.second < dist[it.first]) {

auto record = st.find(make\_pair(dist[it.first], it.first)); if (record != st.end())

st.erase(record);

// distance update

dist[it.first] = nodeDist + it.second; // insert value in set

st.insert(make\_pair(dist[it.first], it.first));

}

}

}

return dist;

}

int main()

{

int n, e;

// Undirected Graph

cout << "No of nodes: "; cin >> n;

cout << "No of edges: ";

cin >> e; vector<vector<int>> edges; for (int i = 0; i < e; i++)

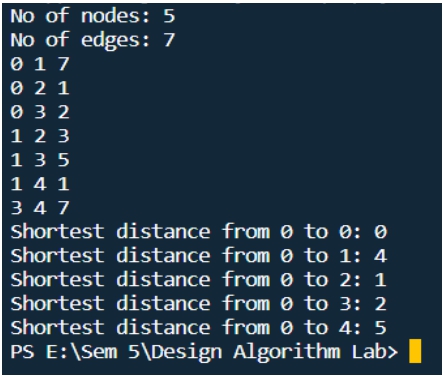
{

int u, v, wt; cin >> u; cin >> v; cin >> wt;

edges.push\_back({u, v, wt});

}

vector<int> distance = dijkstra(edges, n, edges.size(), 0);



for (int i = 0; i < n; i++)

{

cout << "Shortest distance from 0 to " << i << ": " << distance[i] << endl;

}

return 0;

}

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