**Experiment 3.2**

**Aim:** *Develop a program and analyze complexity to find shortest paths in a graph with positive edge weights using Dijkstra’s algorithm.*

**Objectives:** *Code and analyze to find shortest paths in a graph with positive edge weights using Dijkstra.*

**Input/Apparatus Used:** *VS CODE*

# Procedure/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 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.*

# Code:

# *#include <bits/stdc++.h>*

# *using namespace std;*

# *#define INF 0x3f3f3f3f*

# *typedef pair<int, int> iPair;*

# *class Graph {*

# *int V;*

# *list<pair<int, int>>\* adj;*

# *public:*

# *Graph(int V);*

# *void addEdge(int u, int v, int w);*

# *void shortestPath(int src);*

# *};*

# *Graph::Graph(int V) {*

# *this->V = V;*

# *adj = new list<pair<int, int>>[V];*

# *}*

# *void Graph::addEdge(int u, int v, int w) {*

# *adj[u].push\_back(make\_pair(v, w));*

# *adj[v].push\_back(make\_pair(u, w));*

# *}*

# *void Graph::shortestPath(int src) {*

# *priority\_queue<iPair, vector<iPair>, greater<iPair>> pq;*

# *vector<int> dist(V, INF);*

# *pq.push(make\_pair(0, src));*

# *dist[src] = 0;*

# *while (!pq.empty()) {*

# *int u = pq.top().second;*

# *pq.pop();*

# *for (auto i = adj[u].begin(); i != adj[u].end(); ++i) {*

# *int v = (\*i).first;*

# *int weight = (\*i).second;*

# *if (dist[v] > dist[u] + weight) {*

# *dist[v] = dist[u] + weight;*

# *pq.push(make\_pair(dist[v], v));*

# *}*

# *}*

# *}*

# *printf("Vertex Distance from Source\n");*

# *for (int i = 0; i < V; ++i) {*

# *printf("%d \t\t %d\n", i, dist[i]);*

# *}*

# *}*

# *int main() {*

# *int V = 9;*

# *Graph g(V);*

# *g.addEdge(0, 1, 4);*

# *g.addEdge(0, 7, 8);*

# *g.addEdge(1, 2, 8);*

# *g.addEdge(1, 7, 11);*

# *g.addEdge(2, 3, 7);*

# *g.addEdge(2, 8, 2);*

# *g.addEdge(2, 5, 4);*

# *g.addEdge(3, 4, 9);*

# *g.addEdge(3, 5, 14);*

# *g.addEdge(4, 5, 10);*

# *g.addEdge(5, 6, 2);*

# *g.addEdge(6, 7, 1);*

# *g.addEdge(6, 8, 6);*

# *g.addEdge(7, 8, 7);*

# *g.shortestPath(0);*

# *return 0;*

# *}*Observations/Outcome :

# 

# Time Complexity:

* *Time Complexity: O(E \* logV), Where E is the number of edges and V is the number of vertices.*