Experiment 3.2

Student Name: UTKARSH JOSHI **UID:** 21BCS9158
Branch: BE(CSE) **Section/Group:** 802-A
Semester: 5th **Date of Performance:**30/11/23
Subject Name: DAA **Subject Code:** 21CSH-311

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

2. **Objective:**
Develop a concise C++ program to find the shortest paths in a graph with positive edge weights using Dijkstra's algorithm and provide a time complexity analysis.

3. **Algorithm:**

- Initialize arrays dist and parent.
- Create a priority queue pq and push the source vertex with distance 0.
- While pq is not empty:
  - Pop the vertex with the smallest distance.
  - Update distances to neighboring vertices.
- dist now contains the shortest distances from the source vertex to all others.

4. **Input/Apparatus Used:**

   a. C++ Programming Language
   b. C++ Compiler

6. Sample Code and Outcome:

main.cpp

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <limits>

using namespace std;

const int INF = numeric_limits<int>::max();

struct Edge {
    int to;
    int weight;
};

void dijkstra(vector<vector<Edge>>& graph, int src) {
    int V = graph.size();
    vector<int> dist(V, INF);
    vector<int> parent(V, -1);
    dist[src] = 0;

    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
        int>>> pq;
    pq.push({0, src});

```

```cpp
    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        for (const Edge& edge : graph[u]) {
            int v = edge.to;
            int weight = edge.weight;

            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                parent[v] = u;
                pq.push({dist[v], v});
            }
        }
    }

    for (int i = 0; i < V; i++) {
        cout << "Shortest distance from " << src << " to " << i << " is " <<
            dist[i] << endl;
    }
}

int main() {
    int V = 6;
```

```
int main() {
    int V = 6;
    vector<vector<Edge>> graph(V);

    graph[0].push_back({2, 3});
    graph[0].push_back({3, 1});
    graph[1].push_back({0, 2});
    graph[1].push_back({2, 3});
    graph[1].push_back({3, 2});
    graph[2].push_back({1, 3});
    graph[2].push_back({4, 4});
    graph[3].push_back({0, 1});
    graph[3].push_back({1, 2});
    graph[3].push_back({4, 3});
    graph[4].push_back({2, 4});
    graph[4].push_back({3, 3});
    graph[4].push_back({5, 5});
    graph[5].push_back({4, 5});

    int source = 0;
    dijkstra(graph, source);
```

Outcome:

```
Output
/tmp/jvVLonWAl8.o
Shortest distance from 0 to 0 is 0
Shortest distance from 0 to 1 is 3
Shortest distance from 0 to 2 is 3
Shortest distance from 0 to 3 is 1
Shortest distance from 0 to 4 is 4
Shortest distance from 0 to 5 is 9
```

Time Complexity:

O(V*log(V) + E), where V is the number of vertices, and E is the number ofTime complexity: edges in the graph.