Experiment No: 8

Student Name: Damandeep Kaur UID: 23BCS12255
Branch: BE-CSE Section/Group: 23BCS-DC-902(B)
Semester: 5 Date of Performance: 29-10-25
Subject Name: DAA Subject Code: 23CSH-301

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

Objective: Code and analyze to find shortest paths in a graph with positive edge weights using Dijkstra's

Algorithm:

1. Start
2. Initialize
   - Let the number of vertices be V.
   - Create an array dist[V] and set all distances to infinity (∞).
   - Set dist[src] = 0 (distance from source to itself is 0).
3. Create a Min-Priority Queue (Min-Heap)
   - Store pairs of (distance, vertex) to always extract the vertex with the smallest distance.
   - Initially, push (0, src) into the priority queue.
4. While the priority queue is not empty:
   a. Extract the vertex u with the minimum distance.
   b. For each adjacent vertex v of u with edge weight w:
   - If the path through u gives a shorter distance to v, i.e.
   - if dist[u] + w < dist[v]:
   - dist[v] = dist[u] + w
   - Push (dist[v], v) into the priority queue.
5. Repeat Step 4 until all vertices have been processed (i.e., the queue is empty).
6. Print the result:
   - Display all vertices and their shortest distance from the source.
7. End

Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;
```

```
class Graph {
int V;
```



```
    vector< vector< pair<int, int> > > adj;
public:
    Graph(int  vertices)
{           V = vertices;
adj.resize(V);
    }
    void addEdge(int u, int v, int w) {
adj[u].push_back(make_pair(v, w));
adj[v].push_back(make_pair(u, w));
    }     void dijkstra(int src) {
priority_queue< pair<int, int>,
vector<pair<int, int> >, greater<pair<int,
int>
> > pq;         vector<int> dist(V,
INT_MAX);
        dist[src] = 0;
pq.push(make_pair(0, src));         while
(!pq.empty()) {             int u =
pq.top().second;             pq.pop();             for
(size_t i = 0; i < adj[u].size(); ++i) {
int v = adj[u][i].first;                 int weight =
adj[u][i].second;

                if (dist[u] + weight < dist[v]) {
dist[v] = dist[u] + weight;
pq.push(make_pair(dist[v], v));
                }
            }
        }
        cout << "Vertex\tDistance from Source "
<< src << endl;         for (int i = 0; i < V;
i++) {             cout << i << "\t\t" << dist[i]
<< endl;
        }
    } }; int
main() {
int V = 5;
```

```
    g.addEdge(0, 3, 5);
    g.addEdge(0, 4, 3);
    g.addEdge(2, 1, 2);
    g.addEdge(2, 3, 4);
    g.dijkstra(0);
return 0;
}
```

Observations/Outcome:

```
Vertex  Distance from Source 0
0               0
1               8
2               6
3               5
4               3

--------------------------------
Process exited after 0.01835 seconds with return value 0
Press any key to continue . . .
```

Time Complexity: O((V + E) log V)

Learning outcomes (What I have learnt):

1) Understanding of Shortest Path Concept: Learned how to determine the shortest path from a single source to all other vertices in a graph with positive edge weights.
2) Implementation of Greedy Approach: Gained practical experience in implementing Dijkstra’s Algorithm using the greedy method and a priority queue (min-heap) in C++.
3) Graph Representation Skills: Understood how to represent a graph efficiently using an adjacency list and manage weighted edges.
4) Complexity Analysis: Analyzed the algorithm’s efficiency — with time complexity O((V + E) log V) and space complexity O(V + E).
5) Application Awareness: Identified real-world uses of Dijkstra’s Algorithm such as network routing, GPS navigation, and transportation planning.