



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment – 8

Student Name: Ayush Ranjan

Branch: BE-CSE

Semester: 5th

Subject Name: DAA

UID: 23BCS10187

Section/Group: KRG-2-B

Date of Performance: 14/8/25

Subject Code: 23CSH-301

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

2. Procedure:

- Define the problem of finding shortest paths in a weighted graph with non-negative edge weights.
- Take input for the number of vertices, edges, and the source vertex.
- Represent the graph using an adjacency list to store edges and their weights.
- Initialize all distances as infinity, except for the source vertex set to zero.
- Use a min-priority queue to iteratively select the vertex with the smallest tentative distance.
- Update the distances of adjacent vertices if a shorter path is found through the current vertex.
- Display the shortest distance from the source to all other vertices.

3. Code:

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

void dijkstra(int V, vector<vector<pair<int, int>>> &adj, int src) {
    vector<int> dist(V, 1e9);
    dist[src] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    pq.push({0, src});
    while (!pq.empty()) {
        int u = pq.top().second;
        int d = pq.top().first;
        pq.pop();
        if (d > dist[u]) continue;
        for (auto &edge : adj[u]) {
            int v = edge.first;
            int weight = edge.second;
```

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

cout << "Vertex\tDistance from Source\n";
for (int i = 0; i < V; i++)
    cout << i << "\t" << dist[i] << "\n";
}

int main() {
    int V, E;
    cout << "Enter number of vertices: ";
    cin >> V;
    cout << "Enter number of edges: ";
    cin >> E;
    vector<vector<pair<int, int>>> adj(V);
    cout << "Enter edges (u v weight):\n";
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    }
    int src;
    cout << "Enter source vertex: ";
    cin >> src;
    dijkstra(V, adj, src);
    return 0;
}
```

4. Output:

```
Enter number of vertices: 5
Enter number of edges: 6
Enter edges (u v weight):
0 1 2
0 2 4
1 2 1
1 3 7
2 4 3
3 4 2
Enter source vertex: 0
Vertex  Distance from Source
0        0
1        2
2        3
3        8
4        6
```

5. Learning Outcomes:

- Gained understanding of Dijkstra's algorithm and its use in shortest path problems.
- Learned to implement priority queues and adjacency lists in graph-based algorithms.
- Developed ability to analyze time complexity using data structures like heaps.
- Gained insight into greedy algorithm design principles.
- Learned to apply Dijkstra's algorithm in network and routing problem scenarios.