



Experiment-8

Student Name: Gourav Sharma

Branch: BE-CSE

Semester: 5th

Subject Name: DAA

UID: 23BCS10857

Section/Group: 23BCS_KRG_3A

Subject Code: 23CSH-301

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

2. Objective: To implement Dijkstra's algorithm in C++ for finding shortest paths from a source vertex to all other vertices in a weighted graph and understand its working and complexity.

3. Procedure:

1. Start
2. Input the number of vertices V and the graph (weighted adjacency matrix).
3. Initialize:
 - dist[] array with INFINITE values except for dist[src] = 0
 - sptSet[] array (boolean) to keep track of vertices whose shortest distance is finalized
4. Repeat V times:
 - Pick a vertex u not in sptSet with the minimum distance value
 - Include u in sptSet
 - For every adjacent vertex v of u:
 - If v not in sptSet and dist[u] + weight(u,v) < dist[v]
 - Update dist[v] = dist[u] + weight(u,v)
5. After the loop ends, dist[] contains the shortest distance from source to all vertices.
6. Optionally, use a parent array to track the actual shortest paths.

4. Code:

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

```
int minDistance(vector<int>& dist, vector<bool>& sptSet, int V){
    int min = INT_MAX, min_index;
    for(int v = 0; v < V; v++){
        if(!sptSet[v] && dist[v] <= min){
            min = dist[v];
            min_index = v;
        }
    }
    return min_index;
}

int main(){
    int V;
    cout << "Enter number of vertices: ";
    cin >> V;
    vector<vector<int>> graph(V, vector<int>(V));
    cout << "Enter adjacency matrix (0 if no edge):\n";
    for(int i = 0; i < V; i++)
        for(int j = 0; j < V; j++)
            cin >> graph[i][j];
    int src;
    cout << "Enter source vertex (0 to " << V-1 << "): ";
    cin >> src;
    vector<int> dist(V, INT_MAX);
    vector<bool> sptSet(V, false);
    dist[src] = 0;
    for(int count = 0; count < V-1; count++){
        int u = minDistance(dist, sptSet, V);
        sptSet[u] = true;
        for(int v = 0; v < V; v++){
            if(!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v]){
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

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

```
    return 0;  
}
```

5. Observations:

```
Enter number of vertices: 5  
Enter adjacency matrix (0 if no edge):  
0 10 0 30 100  
0 0 50 0 0  
0 0 0 0 10  
0 0 20 0 60  
0 0 0 0 0  
Enter source vertex (0 to 4): 0  
Vertex    Distance from Source  
0          0  
1          10  
2          50  
3          30  
4          60
```

6. Time Complexity:

- Using simple adjacency matrix and linear search for min distance: $O(V^2)$
- Using a min-priority queue / min-heap: $O(E \log V)$

7. Learning Outcome:

- ❖ Learned how to implement Dijkstra's algorithm using C++.
- ❖ Understood the concept of shortest path tree (SPT) and how to select the next vertex.
- ❖ Gained practical experience in updating distances and maintaining a visited set.
- ❖ Strengthened understanding of graph representations (adjacency matrix, adjacency list) and traversal.
- ❖ Learned to analyze the time and space complexity of Dijkstra's algorithm.