Project Report
Dijkstra's Algorithm

Subject Name – Design and Analysis of Algorithms

Subject Code – 23CSH-301

Submitted To:

Er. Mohammad Shaqlain

(E17211)

Submitted By:

Name: Harjit Singh

UID: 23BCS10849

Section: KRG_2-B

An implementation of Dijkstra's Algorithm to find the Shortest route

1. Aim: To develop and analyse the complexity of a program to find the shortest route using Dijkstra’s Algorithm.

2. Objective: The objective is to implement a Dijkstra’s Algorithm that finds the shortest path in a Weighted Graph :

- Each edge shows a path and weight of each path gives the distance of that path.
- The program efficiently finds the shortest route

3. Input / Software Used:

- **Programming Language:** C++
- **IDE used:** Clion ide
- **Input:** Weighted Graph

4. Algorithm:

1. Start
2. Set dist[source]=0 and all other distances as infinity
3. Push the source node into the min heap as a pair <distance, node> → i.e., <0, source>
4. Pop the top element (node with the smallest distance) from the min heap
   - For each adjacent neighbor of the current node:
   - Calculate the distance using the formula:
     i. **dist[v] = dist[u] + weight[u][v]**
     ii. If this new distance is shorter than the current dist[v], update it.
     iii. Push the updated pair <dist[v], v> into the min heap
5. Repeat step 3 until the min heap is empty.
6. Return the distance array, which holds the shortest distance from the source to all nodes.
7. Stop

5. C++ Program:

```
#include <iostream>

#include <vector>

#include <queue>

#include <climits>

using namespace std;
```

```
// Function to construct adjacency
vector<vector<vector<int>>> constructAdj(vector<vector<int>>
                        &edges, int V) {

    // adj[u] = list of {v, wt}
    vector<vector<vector<int>>> adj(V);

    for (const auto &edge : edges) {
        int u = edge[0];
        int v = edge[1];
        int wt = edge[2];
        adj[u].push_back({v, wt});
        adj[v].push_back({u, wt});
    }

    return adj;
}

//Driver Code Ends
// Returns shortest distances from src to all other vertices
vector<int> dijkstra(int V, vector<vector<int>> &edges, int src){

    // Create adjacency list
    vector<vector<vector<int>>> adj = constructAdj(edges, V);

    // Create a priority queue to store vertices that
    // are being preprocessed.
    priority_queue<vector<int>, vector<vector<int>>,
                greater<vector<int>>> pq;
```

```
// Create a vector for distances and initialize all
// distances as infinite
vector<int> dist(V, INT_MAX);

// Insert source itself in priority queue and initialize
// its distance as 0.
pq.push({0, src});
dist[src] = 0;

// Looping till priority queue becomes empty (or all
// distances are not finalized)
while (!pq.empty()){

    // The first vertex in pair is the minimum distance
    // vertex, extract it from priority queue.
    int u = pq.top()[1];
    pq.pop();

    // Get all adjacent of u.
    for (auto x : adj[u]){

        // Get vertex label and weight of current
        // adjacent of u.
        int v = x[0];
        int weight = x[1];

        // If there is shorter path to v through u.
        if (dist[v] > dist[u] + weight)
        {
```

```
                // Updating distance of v
                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }

    return dist;
}
int main(){
    int V = 5;
    int src = 0;

    // edge list format: {u, v, weight}
    vector<vector<int>> edges = {{0, 1, 4}, {0, 2, 8}, {1, 4, 6},
                                 {2, 3, 2}, {3, 4, 10}};

    vector<int> result = dijkstra(V, edges, src);

    // Print shortest distances in one line
    for (int dist : result)
        cout << dist << " ";

    return 0;
}
```

OUTPUT: 0 4 8 10 10

6. Complexity Analyses:

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

ii. **Auxiliary Space:** O(V), Where V is the number of vertices, We do not count the adjacency list in auxiliary space as it is necessary for representing the input graph.

7. Result: The program finds find the Shortest route successfully and satisfies all the constraints.