Aim-Dijkstra’s Algorithm

#include <iostream>

#include <unordered\_map>

#include <vector>

#include <queue>

#include <climits>

using namespace std;

class Graph {

public:

unordered\_map<int, vector<pair<int, int>>> adjList; // adjacency list (vertex, weight)

void addVertex(int vertex) {

adjList[vertex]; // Automatically creates an empty list for the vertex if it doesn't exist

}

void addEdge(int vertex1, int vertex2, int weight) {

adjList[vertex1].push\_back({vertex2, weight});

}

void dijkstra(int start) {

unordered\_map<int, int> dist; // Stores the shortest distance from the start vertex

for (const auto& pair : adjList) {

dist[pair.first] = INT\_MAX; // Initialize all distances as infinity

}

dist[start] = 0;

// Min-heap priority queue to select the vertex with the smallest distance

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

pq.push({0, start}); // Push the start vertex with distance 0

while (!pq.empty()) {

int u = pq.top().second;

int d = pq.top().first;

pq.pop();

// If the distance of the current vertex is greater than the stored one, skip it

if (d > dist[u]) continue;

// Explore all adjacent vertices

for (const auto& neighbor : adjList[u]) {

int v = neighbor.first;

int weight = neighbor.second;

// If a shorter path is found

if (dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

pq.push({dist[v], v});

}

}

}

// Display the shortest distances from the start vertex

for (const auto& pair : dist) {

cout << "Distance from " << start << " to " << pair.first << ": " << pair.second << endl;

}

}

};

int main() {

Graph g;

g.addVertex(1);

g.addVertex(2);

g.addVertex(3);

g.addVertex(4);

// Add weighted edges (directed graph)

g.addEdge(1, 2, 10);

g.addEdge(1, 3, 5);

g.addEdge(2, 3, 2);

g.addEdge(2, 4, 1);

g.addEdge(3, 4, 9);

g.addEdge(3, 2, 3);

// Perform Dijkstra's Algorithm starting from vertex 1

g.dijkstra(1);

return 0;

}