**Code:**

#include <bits/stdc++.h>

using namespace std;

struct Node {

int vertex;

int distance;

};

struct CompareNode {

bool operator()(const Node& a, const Node& b) {

return a.distance > b.distance;

}

};

class Dijkstra {

private:

int vertices;

vector<vector<pair<int, int>>> adjList;

public:

Dijkstra(int V) : vertices(V), adjList(V) {}

void addEdge(int u, int v, int w) {

adjList[u].push\_back(make\_pair(v, w));

adjList[v].push\_back(make\_pair(u, w));

}

void dijkstra(int src) {

vector<int> distance(vertices, INT\_MAX);

distance[src] = 0;

priority\_queue<Node, vector<Node>, CompareNode> pq;

pq.push({src, 0});

while (!pq.empty()) {

int u = pq.top().vertex;

pq.pop();

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

int v = neighbor.first;

int w = neighbor.second;

if (distance[v] > distance[u] + w) {

distance[v] = distance[u] + w;

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

}

}

}

cout << "Shortest distances from source " << src << ":\n";

for (int i = 0; i < vertices; ++i) {

cout << "Vertex " << i << ": ";

if (distance[i] == INT\_MAX) cout<<"INF\n";

else cout<<distance[i]<< "\n";

}

}

};

int main() {

auto start = chrono::steady\_clock::now();

Dijkstra graph(5);

graph.addEdge(0, 1, 2);

graph.addEdge(0, 2, 4);

graph.addEdge(1, 2, 1);

graph.addEdge(1, 3, 7);

graph.addEdge(2, 4, 3);

graph.addEdge(3, 4, 1);

graph.dijkstra(0);

auto end = chrono::steady\_clock::now();

auto diff = end - start;

cout <<"Execution time: "<< chrono::duration <double, milli> (diff).count() << " ms\n";

return 0;

}

**Output:**

