#include <iostream>

#include <vector>

#include <queue>

#include <limits>

// Define infinity as a very large number

#define INF std::numeric\_limits<int>::max()

using namespace std;

// Structure to represent an edge

struct Edge {

int destination;

int weight;

};

// Dijkstra's Algorithm function

void dijkstra(vector<vector<Edge>>& graph, int startNode, int endNode) {

int numNodes = graph.size();

// Create vectors for distances and visited nodes

vector<int> distance(numNodes, INF);

vector<bool> visited(numNodes, false);

// Priority queue to store nodes based on their distance

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

// Initialize starting node

distance[startNode] = 0;

pq.push({0, startNode});

while (!pq.empty()) {

int current = pq.top().second;

pq.pop();

if (visited[current]) continue;

visited[current] = true;

for (const Edge& edge : graph[current]) {

int neighbor = edge.destination;

int weight = edge.weight;

if (distance[current] + weight < distance[neighbor]) {

distance[neighbor] = distance[current] + weight;

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

}

}

}

// Output the result

cout << "Shortest distance from node " << startNode << " to node " << endNode << ": " << distance[endNode] << endl;

// Print the path

cout << "Path: ";

int node = endNode;

while (node != startNode) {

cout << node << " <- ";

node = distance[node];

}

cout << startNode << endl;

}

int main() {

// Example graph represented as an adjacency list

vector<vector<Edge>> graph = {

{{1, 4}, {2, 2}},

{{3, 5}},

{{1, 1}, {3, 8}},

{}

};

int startNode, endNode;

// User input for start and end nodes

cout << "Enter the starting node: ";

cin >> startNode;

cout << "Enter the ending node: ";

cin >> endNode;

// Apply Dijkstra's Algorithm

dijkstra(graph, startNode, endNode);

return 0;

}