#include <iostream>

#include <vector>

#include <queue>

#include <climits>

using namespace std;

struct Edge {

int to;

int weight;

};

struct QueueEntry {

int vertex;

int distance;

bool operator>(const QueueEntry& other) const {

return distance > other.distance;

}

};

vector<int> dijkstra(int V, const vector<vector<Edge>>& graph, int src) {

priority\_queue<QueueEntry, vector<QueueEntry>, greater<QueueEntry>> pq;

vector<int> distances(V, INT\_MAX);

distances[src] = 0;

pq.push({src, 0});

while (!pq.empty()) {

QueueEntry current = pq.top();

pq.pop();

int u = current.vertex;

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

int v = edge.to;

int weight = edge.weight;

if (distances[v] > distances[u] + weight) {

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

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

}

}

}

return distances;

}

int findShortestDistance(const vector<vector<Edge>>& graph, const vector<int>& S, const vector<int>& T) {

int V = graph.size();

int superNode = V;

vector<vector<Edge>> extendedGraph = graph;

extendedGraph.push\_back({});

for (int s : S) {

extendedGraph[superNode].push\_back({s, 0});

}

vector<int> distances = dijkstra(V + 1, extendedGraph, superNode);

int minDistance = INT\_MAX;

for (int t : T) {

if (distances[t] < minDistance) {

minDistance = distances[t];

}

}

return minDistance == INT\_MAX ? -1 : minDistance;

}

int main() {

vector<vector<Edge>> graph = {

{{1, 2}, {3, 3}}, // Đỉnh 0

{{2, 1}}, // Đỉnh 1

{{3, 1}}, // Đỉnh 2

{} // Đỉnh 3

};

vector<int> S = {0};

vector<int> T = {3};

int shortestDistance = findShortestDistance(graph, S, T);

cout << "Shortest distance from S to T: " << shortestDistance << endl;

return 0;