#include <iostream>

#include <vector>

#include <climits>

#include <algorithm>

#include <map>

using namespace std;

const int INF = INT\_MAX;

// Structure to represent a weighted edge

struct Edge {

int to, weight;

};

// Function to find the vertex with the minimum distance in the "unvisited" set

int findMinDistanceVertex(vector<int>& distance, vector<bool>& visited) {

int minDistance = INF;

int minVertex = -1;

for (int i = 0; i < distance.size(); i++) {

if (!visited[i] && distance[i] < minDistance) {

minDistance = distance[i];

minVertex = i;

}

}

return minVertex;

}

// Function to perform Dijkstra's Algorithm on the graph

void dijkstra(vector<vector<Edge>>& graph, int start, int end) {

int numVertices = graph.size();

vector<int> distance(numVertices, INF);

vector<int> parent(numVertices, -1);

vector<bool> visited(numVertices, false);

distance[start] = 0;

for (int i = 0; i < numVertices - 1; i++) {

int u = findMinDistanceVertex(distance, visited);

if (u == -1) break; // All remaining vertices are unreachable

visited[u] = true;

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

int v = edge.to;

int w = edge.weight;

if (!visited[v] && distance[u] != INF && distance[u] + w < distance[v]) {

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

parent[v] = u;

}

}

}

if (distance[end] == INF) {

cout << "No path from " << start << " to " << end << " exists." << endl;

return;

}

cout << "Shortest path from " << start << " to " << end << " with a cost of " << distance[end] << " is:" << endl;

// Construct and print the path

vector<int> path;

int cur = end;

while (cur != -1) {

path.push\_back(cur);

cur = parent[cur];

}

reverse(path.begin(), path.end());

for (int i = 0; i < path.size(); i++) {

cout << path[i];

if (i != path.size() - 1) {

cout << " -> ";

}

}

cout << endl;

}

int main() {

int numVertices, numEdges;

cout << "Enter the number of vertices and edges: ";

cin >> numVertices >> numEdges;

vector<vector<Edge>> graph(numVertices);

cout << "Enter edges (source, destination, weight):" << endl;

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

int source, destination, weight;

cin >> source >> destination >> weight;

graph[source].push\_back({destination, weight});

}

int start, end;

cout << "Enter the starting and ending nodes: ";

cin >> start >> end;

dijkstra(graph, start, end);