#include <iostream>

#include <vector>

#include <queue>

#include <climits>

using namespace std;

// Structure to represent a graph edge

struct Edge {

    int dest;

    int weight;

};

// Structure to represent a graph node

struct Node {

    int vertex;

    int distance;

};

// Comparison function for priority queue

struct CompareNode {

    bool operator()(const Node& n1, const Node& n2) {

        return n1.distance > n2.distance;

    }

};

// Function to perform Dijkstra's algorithm

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

    int V = graph.size();

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

    vector<bool> visited(V, false);

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

    distance[source] = 0;

    pq.push({source, 0});

    while (!pq.empty()) {

        int u = pq.top().vertex;

        pq.pop();

        if (visited[u])

            continue;

        visited[u] = true;

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

            int v = edge.dest;

            int weight = edge.weight;

            if (!visited[v] && distance[u] + weight < distance[v]) {

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

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

            }

        }

    }

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

    for (int i = 0; i < V; ++i) {

        cout << "Vertex " << i << ": " << distance[i] << "\n";

    }

}

int main() {

    int V, E;

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

    cin >> V >> E;

    vector<vector<Edge>> graph(V);

    cout << "Enter the edges (source destination weight):\n";

    for (int i = 0; i < E; ++i) {

        int src, dest, weight;

        cin >> src >> dest >> weight;

        graph[src].push\_back({dest, weight});

    }

    int source;

    cout << "Enter the source vertex: ";

    cin >> source;

    dijkstra(graph, source);

    return 0;

}