Project code:

import java.util.\*;

// Represents a graph node (intersection)

class Intersection {

String name; // Intersection name (e.g., "A", "B", "C")

Map<Intersection, Integer> roads; // Roads to neighboring intersections (adjacency list with travel time as weight)

public Intersection(String name) {

this.name = name;

this.roads = new HashMap<>();

}

// Add a road to a neighboring intersection

public void addRoad(Intersection neighbor, int travelTime) {

roads.put(neighbor, travelTime);

}

// Get neighbors (all intersections connected by a road)

public Set<Intersection> getNeighbors() {

return roads.keySet();

}

// Get the travel time to a neighboring intersection

public int getTravelTime(Intersection neighbor) {

return roads.getOrDefault(neighbor, Integer.MAX\_VALUE);

}

}

// Represents the entire map (city)

class CityMap {

private Map<String, Intersection> intersections;

public CityMap() {

intersections = new HashMap<>();

}

// Add a new intersection to the city map

public void addIntersection(String name) {

intersections.putIfAbsent(name, new Intersection(name));

}

// Add a road between two intersections

public void addRoad(String from, String to, int travelTime) {

Intersection fromIntersection = intersections.get(from);

Intersection toIntersection = intersections.get(to);

if (fromIntersection != null && toIntersection != null) {

fromIntersection.addRoad(toIntersection, travelTime);

toIntersection.addRoad(fromIntersection, travelTime); // Undirected road

}

}

// Dijkstra's Algorithm to find the shortest path from source to destination

public List<Intersection> findShortestPath(String sourceName, String destinationName) {

Intersection source = intersections.get(sourceName);

Intersection destination = intersections.get(destinationName);

if (source == null || destination == null) {

throw new IllegalArgumentException("Source or destination intersection not found");

}

// Create a priority queue with a custom comparator to prioritize intersections by their distance

PriorityQueue<Intersection> pq = new PriorityQueue<>(Comparator.comparingInt(distances::get));

Map<Intersection, Integer> distances = new HashMap<>();

Map<Intersection, Intersection> previous = new HashMap<>();

// Initialize all intersections with a distance of infinity except the source

for (Intersection intersection : intersections.values()) {

distances.put(intersection, Integer.MAX\_VALUE);

previous.put(intersection, null);

}

distances.put(source, 0);

// Start the algorithm from the source intersection

pq.add(source);

while (!pq.isEmpty()) {

Intersection current = pq.poll();

// If we've reached the destination, reconstruct the path

if (current.equals(destination)) {

List<Intersection> path = new ArrayList<>();

for (Intersection at = destination; at != null; at = previous.get(at)) {

path.add(at);

}

Collections.reverse(path);

return path;

}

// Explore neighbors and update distances

for (Intersection neighbor : current.getNeighbors()) {

int newDist = distances.get(current) + current.getTravelTime(neighbor);

if (newDist < distances.get(neighbor)) {

distances.put(neighbor, newDist);

previous.put(neighbor, current);

pq.add(neighbor);

}

}

}

return null; // If no path is found

}

// Update travel time (simulating real-time traffic)

public void updateRoadTime(String from, String to, int newTime) {

Intersection fromIntersection = intersections.get(from);

Intersection toIntersection = intersections.get(to);

if (fromIntersection != null && toIntersection != null) {

fromIntersection.addRoad(toIntersection, newTime);

toIntersection.addRoad(fromIntersection, newTime); // Update both directions

}

}

}

// Main class to simulate and test the city map and navigation

public class NavigationSystem {

public static void main(String[] args) {

CityMap cityMap = new CityMap();

// Add intersections

cityMap.addIntersection("A");

cityMap.addIntersection("B");

cityMap.addIntersection("C");

cityMap.addIntersection("D");

// Add roads (edges) with travel time (in minutes)

cityMap.addRoad("A", "B", 5);

cityMap.addRoad("A", "C", 10);

cityMap.addRoad("B", "C", 2);

cityMap.addRoad("B", "D", 7);

cityMap.addRoad("C", "D", 1);

// Find the shortest path from A to D

List<Intersection> shortestPath = cityMap.findShortestPath("A", "D");

System.out.println("Shortest Path from A to D:");

for (Intersection intersection : shortestPath) {

System.out.print(intersection.name + " ");

}

System.out.println();

// Simulate real-time traffic update (change the road time from B to D)

System.out.println("\nUpdating road time from B to D due to traffic...");

cityMap.updateRoadTime("B", "D", 4);

// Recalculate the shortest path with the updated traffic

List<Intersection> newShortestPath = cityMap.findShortestPath("A", "D");

System.out.println("Updated Shortest Path from A to D after traffic update:");

for (Intersection intersection : newShortestPath) {

System.out.print(intersection.name + " ");

}

System.out.println();

}

}