Assignment 2 - Road Trip

Maguire Marion

Prof David Guy-Brizan

CS 245-1

Assignment 2 Design

Data Structures Explanation

Essentially, my project will be broken down into two primary data structures:

The first primary data structure will be used to handle the ‘attractions.csv’ file. To store these attractions in a way that will keep them connected to their relative city/location, I’ll be adding the items from 'attractions.csv' into a hashmap so that once given the array of attractions that need to be visited along the route by the user, I can reference their city/location by being given just the name (i.e. if the need-to-visit array contains "Boot Hill" I can simply do hashmap.get(“Boot Hill") which will return the value for that key, which would be the city that attraction’s located in, 'Dodge City KS’)

The second primary data structure will be used to handle ‘roads.csv’. To handle this file, I will be reading in each location and it’s relative distance to the next location in the csv file into a graph. This graph will have to be weighted in order to accommodate for finding the ‘shortest’ path in terms of distance (it will do so by keeping track of the provided distance/edges between cities) as well as be undirected so that cities (represented as the vertices of the graph) can be accessed going either way. This graph will be

represented as hashmaps; an ’adjacency hashmap’ if you would. After some thinking, I feel this structure, while seemingly unconventional in comparison an adjacency matrix or list, would perform the absolute fastest in terms of algorithmic running time since hashmaps run about all operations in O(1) constant.

Algorithms Explanation

My project focuses on a central algorithm with a minor additional one:

Starting with the minor algorithm, I will have a function ‘convertAttractionsToCities()’ which will linearly go through the user-provided attractions and check for the attractions ‘city value’ in the attractions hashmap. It will then add the city values to an ArrayList called ‘attractionsAsCities’ so that later the cities needed to be passed through can be easily referenced.

In terms of the algorithm driving the project, Dijkstra’s Algorithm is the way I will carry out the searching for the shortest path. Essentially, once the graph is established and the ‘attractionsAsCities’ ArrayList is ready, I will perform Dijstkra's Algorithm from the start point to the 1st item in the ‘attractionsAsCities’ ArrayList. Once the shortest path from the start to the 1st item is found, the city vertices passed through to achieve that shortest path will be added to another ArrayList called ‘citiesPassedThrough’. I'll then perform Dijkstra's from the 1st item in ‘attractionsAsCities’ to the 2nd item and append this path to the first one in ‘citiesPassedThrough'. Then I’ll do the same for the 2nd

to 3rd item, 3rd to 4th, etc and ultimately return that 'citiesPassedThrough' ArrayList which will contain all of the cities passed through to achieve the shortest path while passing through every city containing the supplied attractions. In a simplified manner, I'm performing an individual Dijkstra's algorithm from each point in the graph that needs to be visited and keeping track of the vertices being passed through.

Class Structure Explanation

In terms of how I will structure my Java classes, I will have a main/driver class, a few separate classes to represent the graph and its components (vertices and edges), as well as a class that performs Dijkstra's algorithm.

My main/driver class 'RoadTrip.java' will primarily read in the data from 'attractions.csv' and 'roads.csv' store those in their respective data structures (creating and filling the hashmap and graph with their data) and then accept user input for the starting city, ending city, and attractions to visit. As stated earlier, main() will also have a method to 'convertAttractionsToCities()' and then a 'route()' method, which will then actually perform Dijstkra's on the created graph.

My 'Graph.java' will be a class which constructs the 'adjacency hashmap' and will have methods to add and remove edges, add and remove vertices, and get the data for a given vertex/edge. This class will utilize two other classes I'll make, 'Vertex.java' and 'Edge.java' which will respectively hold data (Vertex holding the data of a city

and an edge holding data of pointing from one city to another city, and the distance between them). These two sub-classes will have no fancy methods and likely stick to just setters and getters.

The ‘Dijstkras.java’ class will perform the actual algorithm on the graph created in main() using the ‘Graph.java’ class. I plan to have a single method ‘findShortestPath()’ which will do exactly that by performing Dijstkra’s algorithm in the most literal way; by calculating the distance from the starting vertex to each unvisited neighboring vertices, and updating the distances if it finds a shorter than the previous shortest distance. This ‘findShortestPath()’ method will likely return some sort of data structure like an array, ArrayList, or Set containing that shortest path. There may be some other methods here and there to get and set, but nothing major planned apart from that for this class.