Jaeseung Lee

CS241

Project 3: Graph Project

jaeseungl@hotmail.com

Due 07/31/2016 11:59PM

**Section1 : ADT description**

I have several classes: Project3, Graph, Node, FileHandler, Interface, and they are clearly separated and managing each given tasks.

In the Project3 class, I have created 20 graphs, with starting index 1-20. In other words, ANAHEIM(1) to WRIGHTWOOD(20). So in that way, for example, when the user want to know all the edges of graph that starts from BREA CANYON(4), I can access graphs[3] and then give the whatever information they need. For each graph I created, I apply Dijkstra algorithm and pass starting index as argument. The important fact that I had to remember for this project was, to access cityIndex3, which is BOSSTOWN, I have to access nodes.get(2), not 3, which will cause ArrayIndexOutOfBound error.

Next, in the Node class, I do have local variable neighbor and distance, which are both ArrayList. Each nodes keep unique neighbor and distance info. For example, nodes.get(2).neighbor.get(3) will represent all the neighbors of node3(cityIndex3) that lead to node4(cityIndex5). Similarly for the distance will represent the neighbor's distance.

In FileHandler class, it handles all the proper storing process, after reading city.dat and road.dat. To increase the efficiency, I chose to use array, where I exactly know the size of given texts. Because for the row, there were only 3 inputs, I used single for loop, instead of using double for-loop. Such as explicitly stating as:[i][0] =3, [i][1]=BO , [i][2]=BOSSTOWN. I used helper methods: showRoadArray(), showCityArray(), which I decided to leave it there.

Next, I have the core class, Graph class. Each graph handles 20 nodes, which won't change depends on user's input. Here, I used the name "Node" instead of Vertex.

For the variables,

visitList – while loop keeps running until visitList is totally empty.

bestDist – the shortest path distance starting from starting index.

bestPath – Store all the indices that visited for the bestDist.

road- this will store user's inserted road.

traceDist – keep track the distance to get to the current node. Should update when you can't find available neighbors to go forth, and have to go back.

tracePath – keep track the path to get to the current node. Again, update when you can't find available neighbors.

Also In Dijkstra method, what I did was: while visitList isn't empty, figure out if current node has neighbors, if yes, find the neighbor that has shortest path distance. But that will be excluded if that neighbor is already visited. If you can't find the neighbors or neighbors are all visited, you find the leftover node from the visitList that has shortest path distance, and go there. In this process, I should re-update that bestPath, bestDist, traceDist, tracePath. Also initialize the unkown distance with infinity at the beginning. I used recursion to go to the next node, because there were only 20 nodes, and good thing about recursion is, it makes the implementation clear and easier. If the number nodes were bigger than this, I would've used more complex but efficiency data structure, such as priority queue.

Lastly, for Interface class, it handles all the prompting options for user. Prompt method gets argument from the array graphs[] from the Project3,  and it access by array index. And used switch statement which allows me to organize nicely for the user’s input.

## Section2: Testing Methodology

```
Command? H
 Q  Query the city information by entering the city code.
 D  Find the minimum distance between two cities.
 I  Insert a road by entering two city codes and distance.
 R  Remove an existing road by entering two city codes.
 H  Display this message.
 E  Exit.
Command? Q
City code: LV
12  LV    LEE VINING             8390   5983
Command? D
City codes:
CH PM
The minimum distance between CHINO HILLS and POMONA is 143 through the route: [CHINO
HILLS, PICO RIVERA, TORRANCE, POMONA].
Command? I
City codes and distance: GG BO 100
You have inserted a road from GARDEN GRPVE to BOSSTOWN with a distance of 100.
Command? R
City codes:
KV MP
The road from KERNVILLE and MOUNTAIN PASS doesn't exist.
Command? E
```

I exactly followed as the given example. By entering H, it shows you an instruction, Q allows user to show full-information that I obtained from the city.dat and road.dat. For D, I directly store user's input into the small array, using .split() to spate, and then try to find those in the cityArray that I stored, if I find both inputs, I  access to graphs[cityIndex1-1].get… and then do the task. I and R were simple. Using the road ArrayList in the Graph class, I initialized all the values to 0, and I let user store the value. If the user try to remove the value 0, print out, the road doesn't exist. Also I have prevented all the possible exception errors by properly using try-catch statement or throw exception.

## Section3: Lessons learned

I really learned a lot from this project. I wasn't familiar on using objects. Also I was able to cleanly seperated the classes depending on its given tasks. I'm also not familiar using ArrayList and all the newest data structure that are came from Java Library, such as InputStream, InputReader and etctera, and it was great opportunity to practice those. All the genetic types and other basics that we previously covered are on my homework to review and master those for my own good. And this project also motivated me to learn more algorithms, where it ignited my interest to lots of algorithms.