Demo 3

The YelpData class remains the same. This assignment saw the use of graphs, like assignment 2, I stored all of the nodes and the Edges in a separate text file. Each of the Node contains a YelpData obj, array of edges, a double for the haversine formula to calculate the real distance from a different location. The parent node and LinkedList of paths. The add edge function adds in edges to the new node, establishing a new line between the current node and the argument node.

The get all neighboring function loop through all the edges then, since Nodes connection or edges were stored in a separate text file, the read function takes in a long arg as ID then return Node’s information tied to the edge.

getDistanceTo function cal the gerDistanceToOtherNode using the 2 nodes latitude and longtitude. What follows is the calculation of haversine formula to find the real distance in Km between 2 locations, the use of radian conversion as needed since earth is round so, we need to find the angular distance in a spherical triangle.

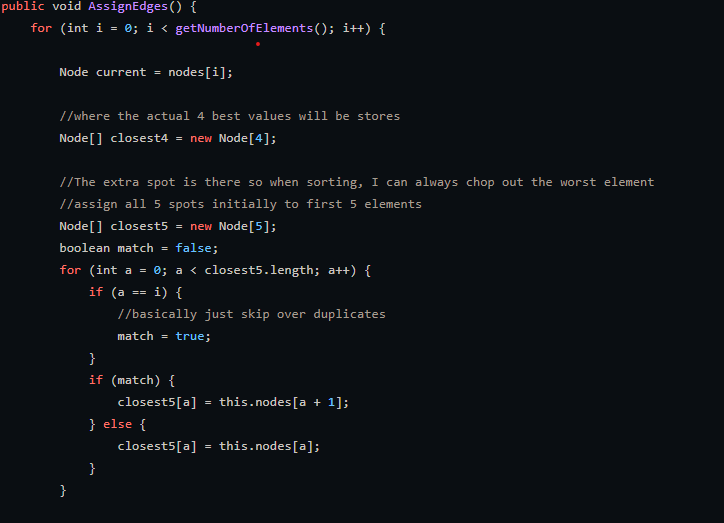
The Graph class contains a lot of small function so I will set them in bullet points

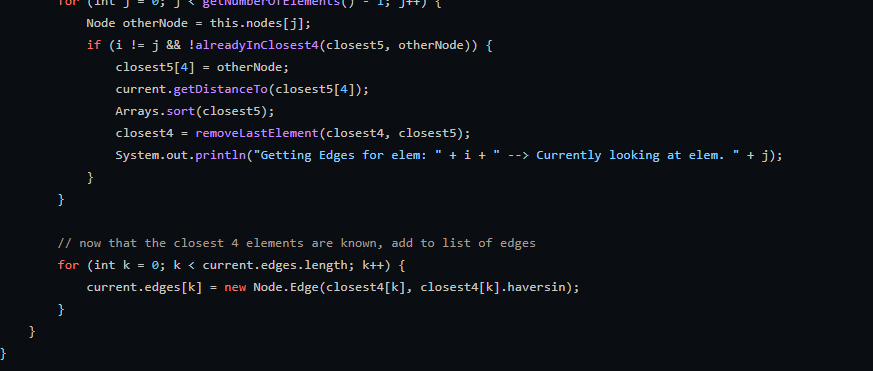
-add (Node n): add in new node if there is no existing node that contains the same IDnumber

-getNumberOfElement(): get the total number of nodes

-getIndexOfLastElement(): get the last element of the nodes array

-AssignEdges(): this function will loop through every possible Nodes then. It attemps to find the four closest neighbor, first however, I figured that I need a filled in list so that doing the comparison and looping through every other node will be easier.

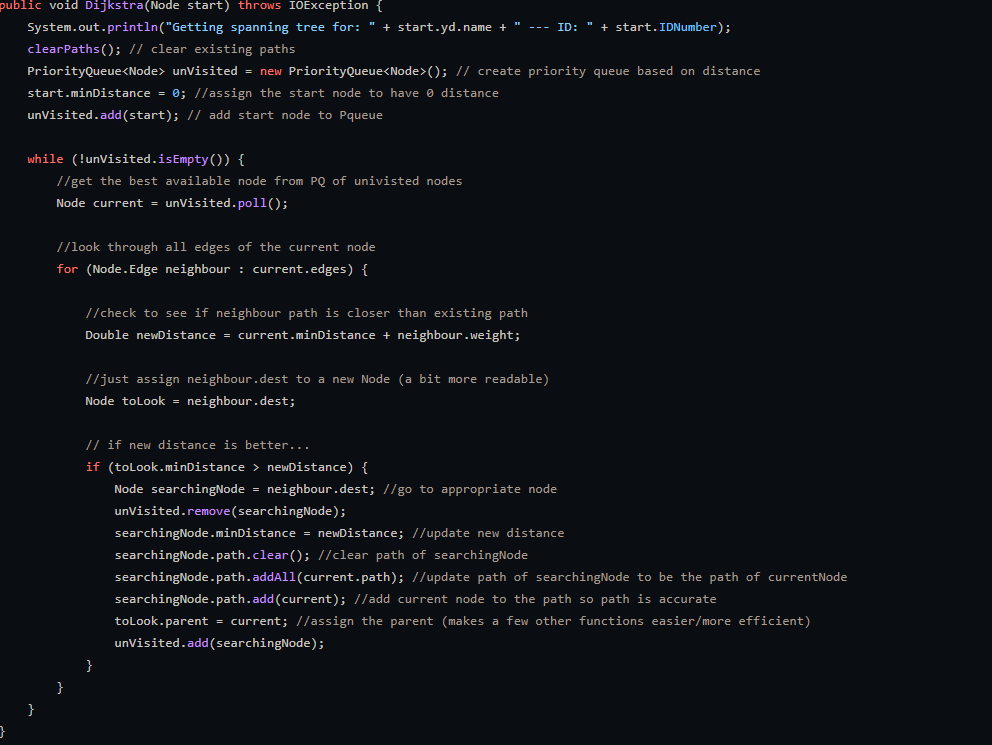


At the end of then second loop, we should have the complete version of closest5 containing the first 5 nodes. The reason why I chose to find the first 5 note instead of 4 will be discuss later. 

It made another loop then, for every node, it will find the haversine distance and added to the last elem of the closest5 ,then sort so that the longest distance will be at the end and the shortest will be at index 0. Each time that happens, I assigned the first 4elem of the closest5 to the closet for since we are loong for 4 closest nodes. Having a closest 5 means I could easily chop off the last elem.

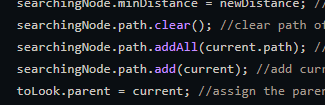
At the end of the iteration I should have added the edged to every nodes.

Moving on to the Dijkstra,



The function uses ProrityQueue as a collection because the poll() function returns the top of the queue while also removing it, making the operation easier to handle. It starts by clearing path which will be discuss later. Placing the Node start, the first interation will be comparing the start node to the all of the neigboring finding the shortest length to every other nodes. The start nodes will then be compare to all of its edges by a loop. For every loop it makes or distance that is smaller to that of the old path which is toLook.minDistance, then clear the path and add it to the searching not.path along with the current node’s path as well.

The pathFromTo requires a starting Node and a destination. We then run the Dijkstra function for the path from the starting location to loop and find the shortest way to every other nodes, including the destination as it was updated with following lines:



We then get the path list and return it. This however needs to be revised as, given the sheer length of the number of Nodes, we are only look for one destination each events, however, because of the design, we need to look at every other path thtt won’t be use for the events, making the program runs longer and more resources consuming. The rest of the function serve as tools to write and read edges and nodes into the 2 separate files. However, I did tried to create a another files only containings the edges which points destinations to an ID of type Long so as to make the text file readable. These long can later be referef to the orginal Yelp datas, but Json loading time is heavy and converting it to csv should be more efficient but it is quite to late.

