COMP333 – Assignment 1

Group Work – Stage 3 & 4

1. Group Member:

Name	Student ID	Responsibility
Phong Lam Nguyen	44141610	Working on stage 4 and Merging both stages
Phan Hoai Nam	44183534	Working on stage 4
George Hayward	43692486	Working on stage 3

2. Stage 3

Algorithm Approach: The algorithm used is intended to increase the efficiency of the calculation of all the ratios between every combination of station. This is done by first gathering two hashmaps which are required to calculate ratios.

```
public HashMap<String, HashMap<String, Double>> generateAllDistances ()
```

This hashmap contains strings as keys which connect one station to another. The double is the distance between them. This is simply done by getting every combo of stations and computing the distance of them ONCE.

```
public HashMap<String, HashMap<String, List<String>>> generateAllRouteMinDistances
()
```

This hashmap contains strings as keys which connect on station to another. The List<Strings> is the best route from one station to another. This was calculated (although not successfully) similarly to the rod cutting problem. Instead of calculating the min distance for every single combination of stations. The algorithm first checks if a saved best route contains a subroute of these two stations. If so, add that as the best route for the origin and destination, instead of making the costly call of routeMinDistance.

The compute method simply divides the real bestRouteDistance by the directDistance. This gives the ratio.

Unfortunately, I could not complete this in time. However, I believe this approach should have passed tests if implemented correctly. It gathers all required variables only ONCE to get all ratios. The code was buggy and beyond repair at this time.

(Note: both implementation of function generateAllDistances() and generateAllRouteMinDistances() is commented, therefore, they will not slow down all the class and influence other implemented functions)

3. Stage 4

Stage 4 is asking students to find the shortest path between origin station and destination station in terms of number of stops based on the train lines and noting down any lines' transfer that is required.

3.1 Added data structure:

3.1.1 Station Class

To complete this stage, first of all, we need to modify 'Station' class in order to store more data about train lines that we are going to use.

- I have added two more constructors for 'station' class:
  - adjacentLineStation: is used to hold the data about adjacent stations (neighbours) of selected station. It is a treemap with key is a Station and value is String.
  - trainLine: is an ArrayList of String, which is used to store the train line value that the selected station is located on.
- In addition, I have added 'add-get' methods for both constructors above:
  - getAdjacentLineStation() and addLineNeighbour() for adjacentLineStation constructor.
  - getTrainLine() and addLine() for trainline constructors.

3.1.2 trainLineList constructor of RailNetworkAdvanced class

I have created a new constructor named trainLineList which is a treemap with each key is different train lines, and value of each key is the list of station's names in ascending order.

For example:

- key: T6
- value: [Carlingford, Telopea, Dundas, Rydalmere, Camellia, Rosehill, Clyde]

To generate the values for this new constructor, I have added code to function readLinesData() and modify the function readStationData().

- readLinesData () function is used to initialize the value for the constructors; it will pass all train lines' names into the constructor as the keys then initializing the value of each key with empty array of string, next, each array of string will be assigned in the first item and last item are the start station name and last station name of each train line.
- Then with the help of readStationData() function, the value of each key (train line) will be filled completely, and become the list of stations' names in ascending order corresponded to their locations or order in the selected line.

In addition, I was modifying readStationData() to assign the line value for each station using addLine() method from 'station' class.

In addition, I have created the populateNeighbors() function to give each station its adjacent station data based on train lines information by using addLineNeighbour() method from 'station' class.

3.2 routeMinStopWithRoutes() function:

Applying the similar strategy from stage 1 to this stage, we have found that the network of stations based on train lines information is also presented as a graph, where each station can be considered as different nodes, each railway is the line connecting each node. However, for this case, we do not need to compare any distances or number of stop between selected stations and origin, we just need to establish the connection between each station in the station list based on train lines information. Therefore, we was applying BFS strategy including queue and backtracking method to searching and creating the shortest path from origin station to destination station based on train lines information.

I have created 3 variables including:

- Result: is an Array List of string that will store the shortest path between origin station and destination station based on train lines information.
- Queue: is a Linked List of string that will hold the temporary station names, which will be used to process BFS and backtracking method until it reaches destination station.
- Path: is a treemap with key is a list of station name and value of each key is a nearest adjacent station between origin station and selected station based on train lines information.

First of all, we need to set up the condition to check whether the selected station has been visited. Therefore, at the start of the code, we have set all station to 'un-visited' state using function setUnmarked().

Then, we initialized the value for first station which is origin station:

- Set origin station to mark as we have visited to assign value.
- Add this station to the queue
- And set it nearest adjacent station between origin and selected station to 'None' as it is origin station.

Next, we started the while loop, which is similar to standard BFS:

- We get the first item of the queue out and remove it from the queue.
- Then getting its adjacent station data based on train line information using function getAdjacentLineStation().
- Next, we iterate through each adjacent station:
    - If the selected adjacent station is not visited yet, mark it as visited using setMarked() function
    - Update selected adjacent station's data of nearest adjacent station between origin and selected adjacent stations based on train line information (update path treemap).
    - Then, add selected adjacent station to the queue to use for backtracking method.
    - Last, if the selected adjacent station is the destination station, stop the loop.
- Then, we establish the path from origin to destination station based on train line information.
- I have created 4 more supporting functions in order to get the final result which including both the path from origin to destination station and train line information:
    - buildLine() function which is used to obtain the list of train lines corresponding with the path from origin to destination station based on train line information constructed from previous step.
    - buildRoute() function which is used to constructing the final result.
    - findPostition() function which is used to the locations/positions or orders of selected station on the selected line.
    - findDirection() function which is used to generate the train line information.

3.2.1 buildRoute() function:

I have created 4 variables in order to archive this function:

- result which is the array list of string, containing the final result for stage 4.

- Target which is the array list of integers holding the list of end and start location of each train lines, which is using to process the path. For example, the path has 5 station and contains 2 different train lines so that target = [0,3,4], 0 is the start of first line, 3 is the position of station in the path, where we need to change line, therefore 3 will be both end of first train line start of second train line, and 4 is the end of second train line.
- myRoute which is the array list of array lists containing the sub-list of station corresponding to different train lines from previous steps. For example, the path is [A-B-C-D-E], A-B-C is line T1 and we need to change line at station C then C-D-E is line T2, therefore, myRoute = [[A-B-C], [C-D-E]].
- myNote which is the array list of string storing the train line information, for example, myNote = [T1 towards C from A, T2 towards E from C].

First, we obtain the data for 'target' then using these data:

- Getting data for 'myNote'
- Constructing data for 'myRoute'

Finally, we combine data from both 'myNote' and 'myRoute' to create the final result for stage 4.

In addition, we have generated the different result for the path from Blacktown to Parramatta:

[T5 towards Leppington from Schofields, Blacktown, Seven Hills, T1E towards Chatswood from Emu Plains, Seven Hills, Westmead, T5 towards Leppington from Schofields, Westmead, Parramatta]

Our result for this path have distance is 4 stops (number of stops), compared to the result from documentation which is T1 line requires only 4 stops: Blacktown, Seven Hills, Westmead, Parramatta; both results have the same minimum number of stops but there are the different train lines information and path, therefore, we decided that our implementation for stage 4 is still correct.

(Note: Without the implementation for stage 3, which is slow down all the functions in stage 4, stage 4 are working instantly and correctly)