CityGrail: Optimising Sydney Rail Network Repairs

COMP333 Assignment 1 Part 1

Macquarie University

Version 0.99 (Almost Ready Version)

September 12, 2019

Note: This is a draft version of the assignment, but a changelog will be provided at the end of the document to note any future changes.

1. Objectives

The objectives of this assignment is to test your ability to design efficient algorithms to solve the given problems, give an analysis of these algorithms, and then implement them using the Java programming language. The assignment also tests your ability to understand and follow instructions carefully and to implement your solutions under the given constraints. Finally, the assignment will evaluate your ability to work in a small team with other students.

2. Background

For this assignment, you will be asked to solve several graph-related questions using the Sydney Trains network as the motivating example.

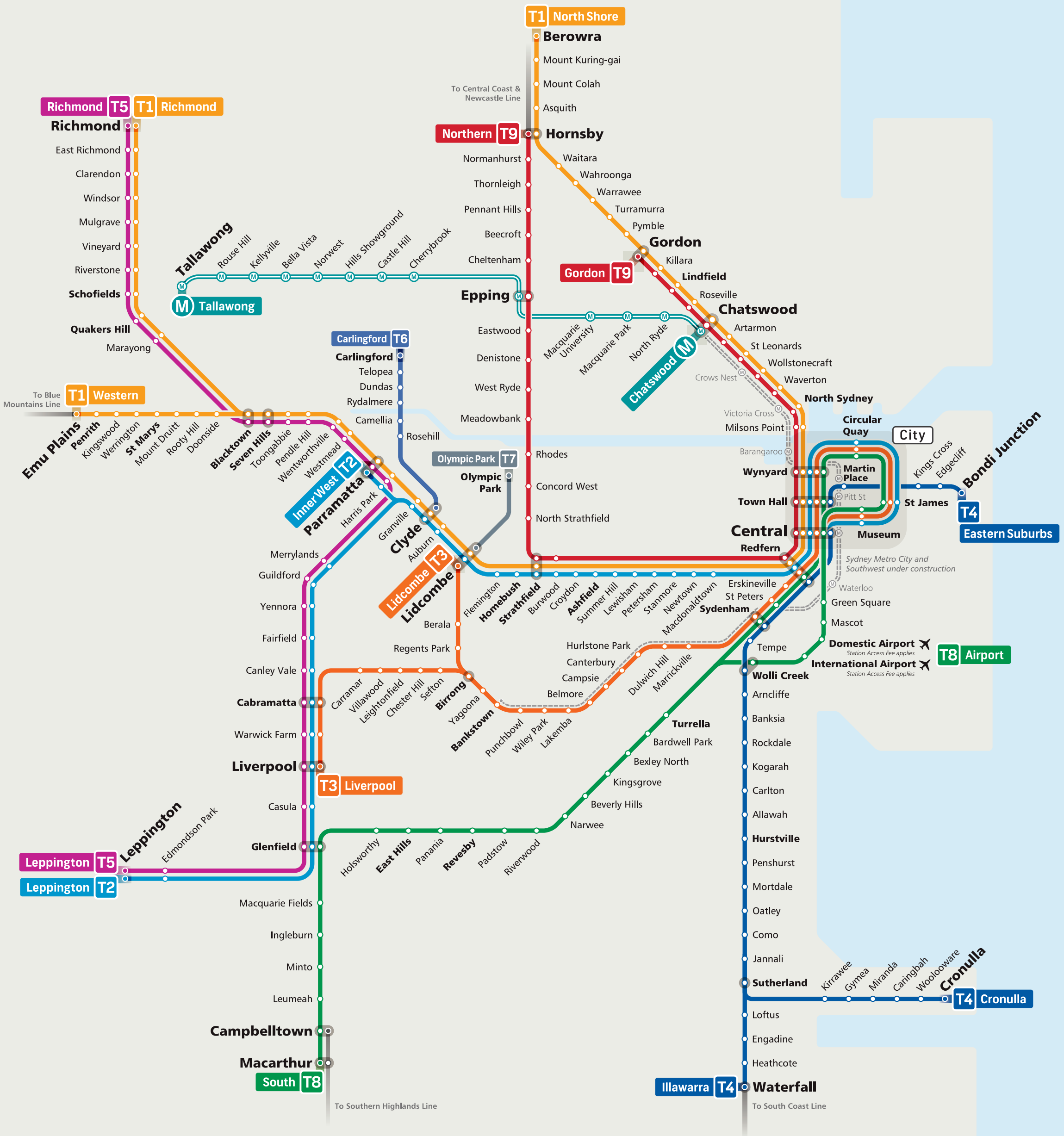
2.1 CityRail

The CityRail network consists of 307 stations running over 2,060 kilometres of track in and around Sydney, Newcastle and Wollongong. CityRail was abolished in June 2013 and was then superseded by Sydney Trains and NSW TrainLink, so technically we should not refer to Sydney train network as CityRail anymore, but I am sure most of us still fondly remember the term ‘CityRail’ and how it rhymes with ‘grail’. Sydney Trains operates 175 stations over nine lines (the metro lines are run separately by Sydney Metro). For this assignment, I have added three metro stations that were originally on the CityRail network: North Ryde, Macquarie Park, and Macquarie University.

Sydney rail network

M Metro T Trains

NSW GOVERNMENT | Transport



Sydney metro and train lines

M *Metro North West Line*
Chatswood
Tallawong

T1 *North Shore & Western Line*
North Shore
Western
Richmond

T2 *Inner West & Leppington Line*
Inner West
Leppington
City

T3 *Bankstown Line*
Liverpool
Lidcombe
City

T4 *Eastern Suburbs & Illawarra Line*
Eastern Suburbs
Illawarra
Cronulla

T5 *Cumberland Line*
Leppington
Richmond

T6 *Carlingford Line*
Carlingford
Clyde

T7 *Olympic Park Line*
Olympic Park
Lidcombe

T8 *Airport & South Line*
Airport
South
City

T9 *Northern Line*
Northern
Gordon

Check timetables and trip planners for train services and connections

Visit **transportnsw.info**

1812TMS-E-MWT-A4P

2.2 The Data

For this assignment you are given two CSV files: station_data.csv and adjacent_stations.csv. Both files can be found in package data in the Comp333 repository at

https://github.com/sutantyo/comp333_2019/tree/master/codes/src/data

The first file, station_data.csv, contains information about 178 stations on the Sydney Trains network (including the three I added above). Each row of the CSV file contains the name of the station and the latitude and longitude coordinates of its location. The CSV file also contains information about which line(s) the station belongs to, which is used for Stage 4 (see below on how they are used in conjunction with lines_data.csv).

The second file, adjacent_stations.csv, contains the list of all adjacent stations in the network. Each row lists exactly two stations which are adjacent to each other, but each pair in a row is unique, that is, if station a is adjacent to station b, then either (a, b) or (b, a) will be in the file, but not both.

The third file, lines_data.csv is only used for Stage 4, and it contains information about the train lines that are running on the Sydney Trains network. While Sydney Trains runs a total of 9 lines, some of the lines have different stops, for example:

- T1 towards Lidcombe from Berowra stops at Berowra, Mount Kuring-gai, Mount Colah, Asquith, Hornsby, Waitara, Wahroonga, Warrawee, Turramurra, Pymble, Gordon, Killara, Lindfield, Roseville, Chatswood, Artarmon, St Leonards, Wollstonecraft, Waverton, North Sydney, Milsons Point, Wynyard, Town Hall, Central, Redfern, Burwood, Strathfield, and Lidcombe.
- T1 towards Chatswood from Richmond stops at Richmond, East Richmond, Clarendon, Windsor, Mulgrave, Vineyard, Riverstone, Schofields, Quakers Hill, Marayong, Blacktown, Seven Hills, Toongabbie, Pendle Hill, Wentworthville, Westmead, Parramatta, Strathfield, Redfern, Central, Town Hall, Wynyard, Milsons Point, North Sydney, St Leonards, and Chatswood.

To accommodate this, the file lines_data.csv contains 16 different train lines instead of 9. For example, T1 is now split into three different train lines:

- T1 towards Chatswood from Emu Plains
- T1 towards Chatswood from Richmond
- T1 towards Lidcombe from Berowra

Each row on lines_data.csv contains the code for the train line (e.g. T1E, T1R, T1B respectively for the above three lines), followed by the actual line name (e.g. T1), the starting and ending stations, and the number of stations on the line.

The file station_data.csv contains columns where the headings are the train line codes (e.g. T1E, T1R, T1B). If a station is part of a train line, then its corresponding column will have a positive integer signifying the position of the station on the train line (i.e. its stop number). For example, Blacktown has 9 for T1E, 11 for T1B, and 20 for T5, meaning that

- it is the 9th stop on T1 towards Chatswood from Emu Plains
- it is the 11th stop on T1 towards Chatswood from Richmond
- it is the 20th stop on T5 towards Leppington from Schofield

The final column on station_data.csv contains the number of train lines the station belongs to (3 in the case of Blacktown).

3. Assignment Structure

The starter template for the assignment can be found in

https://github.com/sutantyo/comp333_2019/tree/master/codes/src/assg1p1

If you have done previous workshops tasks, then the files should be added to your Eclipse project when you do a pull. If you have not set up git with Eclipse as requested in Week 1, then you can do so now (the instructions are on the main page of the repository).

Note: you are not required to use Eclipse for this assignment and you can download the files directly from github, but it is strongly recommended that you practice the use of git since you may be required to use it later.

The starter template for this assignment can be found inside the assg1p1 and assg2p2 packages. The package assg1p1 contains the starter code for Stages 1 and 2, while the package assg1p2 contains the starter code for Stages 3 and 4. Here is a brief description of the files in these two packages:

3.1 Package assg1p1

3.1.1 Station.java

The Station Class represents a station in the rail network. Each Station object has the name of the station (as String), and the latitude and longitude coordinates (as Double) representing the location of the station. Each Station object can also be considered as a vertex in the rail network graph, and so each Station object contains a list of other Station objects adjacent to it in the form of a TreeMap<Station,Integer> (referring to the adjacent station and its distance).

You should be able to complete Stage 1-3 of the assignment without having to modify Station.java, although you are allowed to make minor modifications if it helps. For Stage 4, it might be necessary to modify Station.java, and for this, the package assg1p2 also contains a Station.java file.

3.1.2 RailNetwork.java

The RailNetwork Class represents the rail network itself, and contains one important attribute: stationList, which is a mapping of String objects to Station objects (implemented as a TreeMap).

The rail network can be represented as a graph with the Station objects being its vertices, thus stationList is the set of vertices of this graph, with the information of the edges contained within each Station object in the set. stationList is implemented as a mapping from String to Station objects as a convenience, since we can obtain any Station object from this using its name, e.g. stationList.get("Epping").

The RailNetwork Class contains the methods that you need to implement to satisfy the requirements for Stages 1 and 2 of the assignment. See the task description for these stages for more information. Please do not modify the constructor since it is going to be used by the JUnit test cases.

3.2 Package assg1p2

3.2.1 Station.java

The Station.java file is unchanged from the one in package assg1p1, but as mentioned before, you may want to make some modification to Station.java for Stage 4 (e.g. by adding information about the train line).

3.2.2 RailNetworkAdvanced.java

Similar to RailNetwork Class, the RailNetworkAdvanced Class contains the methods that you need to implement for Stage 3 and Stage 4 of the assignment. To finish Stage 3 and 4, you may need methods that you have written for Stage 1 in RailNetwork.java, and these would be repeated in RailNetworkAdvanced.java.

Since Stage 4 requires an extra input file, the constructor takes an extra String variable that contains the path to the lines_data.csv file. As in RailNetwork.java, please do not modify this constructor.

4. Task Description

Your task for this assignment is divided into four stages, plus a preliminary stage that is not going to be marked (albeit necessary for the rest of the assignment). The preliminary stage and stages 1 and 2 are to be done individually while stages 3 and 4 can be done in a group.

We describe these stages in this section so that you have an understanding of what you need to do. The allocation of the marks will be discussed in the next section.

4.1 Preliminary Stage

Your first task is to process the data files and populate stationList in RailNetwork.java. As described earlier, stationList is the graph representation of the rail network, and so it is the most important data structure in the assignment. To populate stationList, you need to complete two methods:

```
public void readStationData(String infile) throws IOException
public void readConnectionData(String infile) throws IOException
```

In method readStationData, you need to read the CSV file containing the station names and its latitude and longitude (see 2.2) and then create a Station object for each row of data (i.e. the vertices of the graph). You can assume that the file is correct, that is, a station name will not be repeated on different rows, and each row contains the proper values for the name, latitude and longitude, separated by commas.

Once you have added the list of stations into stationList, the next step is to add the neighbours for each station, and this should done in method readConnectionData. As mentioned in 2.2, each row of data contains two station names, signifying that they are adjacent to each other. You can assume that each pair of names is unique, i.e. it will not be repeated in another row.

For each neighbouring station, you also need to set its distance in metres as the weight of the edge (see 3.1.1). To find the distance between two stations, two helper methods have been written for you:

```
public int computeDistance(String a, String b)
public int computeDistance(Station a, Station b)
```

Both of these methods in turn call the static method

```
public static int computeDistance(double lat1, double lon1, double lat2,
    double lon2)
```

which computes the distance in metres between sets of latitude and longitude coordinates. Please DO NOT MODIFY these three methods because the JUnit tests use the distance computed by these methods.

As you may have noticed, the distance between two stations is computed as the most direct path between two geo-coordinates, hence it is only an approximation of the actual distance between two stations (i.e. the length of the rail track).

4.2 Stage 1

Once you have created the graph representation of the rail network, your next task is to find the shortest route between any two stations in the Sydney Trains network. Here, the *shortest route* can either mean the the least number of stops or the least amount of distance travelled between the origin and the destination stations (where the distance between two stations is the value returned by computeDistance method, as discussed earlier). You need to write one method for each definitions:

```
// find the shortest route (in terms of the distance travelled)
// between the origin station and the destination station
public ArrayList<String> routeMinDistance(String origin, String destination)

// find the shortest route (in terms of the number of stops)
// between the origin station and the destination station
public ArrayList<String> routeMinStop(String origin, String destination)
```

The above methods should return an ArrayList<String> containing the name of the stations in the shortest route, including the origin and the destination stations. For example, the shortest distance in terms of the number of stops between Hornsby and Chatswood is 11 stops, but there are two possible routes:

- Hornsby, Waitara, Wahroonga, Warrawee, Turramurra, Pymble, Gordon, Killara, Lindfield, Roseville, Chatswood, total distance 13,182 metres
- Hornsby, Normanhurst, Thornleigh, Pennant Hills, Beecroft, Cheltenham, Epping, Macquarie University, Macquarie Park, North Ryde, Chatswood, total distance 19,678 metres

You are also required to write two more methods (similar to the above ones) that return the shortest routes between two stations if one or more stations are removed from the network (to model a blocked route).

```
// find the shortest route (in term of the distance travelled)
// between the origin station and the destination station that
// does not pass through stations in TreeSet<String> failures
public ArrayList<String> routeMinDistance(String origin, String destination,
     TreeSet<String> failures)

// find the shortest route (in terms of the number of stops)
// between the origin station and the destination station that
// does not pass through stations in TreeSet<String> failures
public ArrayList<String> routeMinStop(String origin, String destination,
    TreeSet<String> failures)
```

For example, the shortest route between Hornsby and Epping if Beecroft is removed is 15 stops (23,157 metres):

- Hornsby, Waitara, Wahroonga, Warrawee, Turramurra, Pymble, Gordon, Killara, Lindfield, Roseville, Chatswood, North Ryde, Macquarie Park, Macquarie University, Epping.

Finally, you need to write a method to compute the length of a route, that is the total distance between the first and last stations in the route.

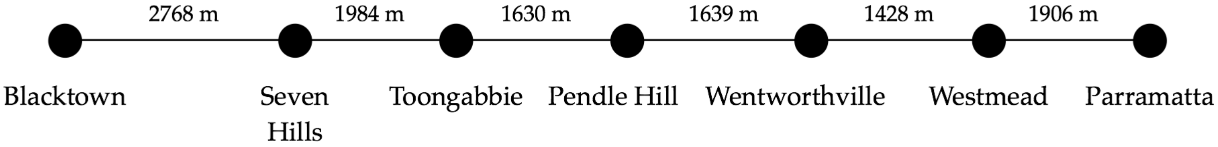
```
// given a route, returns the total distance of the route from the
// starting station to the ending station
public int findTotalDistance(ArrayList<String> path)
```

4.3 Stage 2

For Stage 2 of the assignment, you are required to devise an algorithm to help with the following task:

To test the integrity of a route between two stations, the electrical engineers have to perform an *exhaustive scan* on the length of rail between the two stations. A segment of rail between two stations does not need to be scanned if there are no other stations between them. However, if there are multiple stations in the route, then the engineer have to further divide the segment into smaller segments and perform additional scans on them (until the segment contains no station in between).

For example, suppose you are given the route between Blacktown and Parramatta as illustrated in the following diagram:



The engineers do not need to scan the segment between neighbouring stations, so a scan is only needed on a segment with three or more stations. If we need to check the segment between Blacktown and Toongabbie, then we need to do one scan on the length of the rail between these two stations, and we are then finished because the segments Blacktown-to-Seven Hills and Seven Hill-to-Toongabbie do not contain any other station.

However, if we have to test the segment between Blacktown and Pendle Hill, then after our first scan, we have two possibilities for our next step:

- test the segments Blacktown-to-Seven Hills and Seven Hills-to-Pendle Hill, or
- test the segments Blacktown-to-Toongabbie and Toongabbie-to-Pendle Hill.

Note that the two options above are mutually exclusive, that is, there is no need to test overlapping segments. For example, if you choose to test the segments Blacktown-to-Seven Hills and Seven Hills-to-Pendle Hill, then you do not need to test the segment between Blacktown and Toongabbie.

The cost of performing a scan on a rail segment is linear to the length of the segment, so to simplify the calculation, let us assume that it costs 1 time unit to scan 1 metre of the rail segment, i.e. the cost of scanning the rail segment between two stations is x time units where x is the distance between the two stations.

The total cost of performing an exhaustive scan changes depending on how we choose our segments. For example, given the segment between Blacktown and Pendle Hill, our first scan costs (2768+1984+1630) = 6382 time units, then

- If we test the segments Blacktown-to-Seven Hills and Seven Hills-to-Pendle Hill, we need to further scan the second segment at a cost of (1984+1630) = 3614 time units.
- If we test the segments Blacktown-to-Toongabbie and Toongabbie-to-Pendle Hill, we need to further scan the first segment at a cost of (2768+1984) = 4752 time units.

Therefore the first option is more efficient since it costs a total of (6382+3614) = 9996 time units instead of (6382+4752) = 11134 time units.

Your task is to devise an algorithm to determine the minimum cost of performing an exhaustive scan on a given route, and also to give the list of stations that were chosen to further segment the route in order to obtain this minimum cost. For this stage, you need to complete two methods in RailNetwork.java:

```
// returns the minimum cost of performing an exhaustive scan
public int optimalScanCost(ArrayList<String> route)


// returns the list of stations that were chosen to obtain the minimum cost
// of performing an exhaustive scan
public ArrayList<String> optimalScanSolution(ArrayList<String> route)
```

To give another example, the minimum cost of exhaustive scan on the route Blacktown-to-Parramatta is 29313 time units where the segmentation is given by the stations Toongabbie (i.e. we choose the segments Blacktown-to-Toongabbie and Toongabbie-to-Parramatta), followed by Wentworthville (i.e. we choose the segments Toongabbie-to-Wenworthville and Wentworthville-to-Parramatta).

4.4 Stage 3 (Group Work)

The engineers are now conducting a study on the benefit of constructing a new train line between two existing stations. One important data that they need is a comparison between the *current distance* between two stations using existing rail tracks, and the hypothetical distance between two stations if there is a direct train line between them. For any two stations, the higher the ratio between current distance and hypothetical distance is, the more benefit can be obtained by building a train line between them.

Your task is to generate the ratio between the current and hypothetical distances for all pairs of stations in the Sydney Trains network. For example, the distance between St James and Martin Place is 3,344 metres, but the distance between the two stations is just 315 metres, giving a ratio of 10.615873. The engineers want the ratios to be stored in a 2D-map data structure, which in Java is HashMap<String,HashMap<String,Double>>.

For this task you need to write two methods:

```
// returns the ratio of current and hypothetical distances between
// the origin and destination stations
public double computeRatio(String origin, String destination)
```

and

```
// compute the ratio of current and hypothetical distances between
// the origin and destination stations for all stations in the network
public HashMap<String,HashMap<String,Double>> computeAllRatio()
```

To obtain the current distance between two stations you need to call the findTotalDistance and routeMinDistance methods that you implemented in Stage 1 of the assignment. Efficiency is extremely important in this task, so you need to find the best way to compute these ratios quickly.

4.5 Stage 4 (Group Work)

For the final stage, you are again asked to find the minimum distance between two stations in terms of number of stops, but this time you are required to specify the train line(s) that is being used to travel between the two stations and noting any transfer that is required

```
// find the shortest route (in terms of the number of stops) between
// the origin station and the destination station, noting the train
// line(s) used and the required transfers
public ArrayList<String> routeMinStopWithRoutes(String origin, String
    destination)
```

For example, the shortest route between Beecroft and Chatswood is 7 stops:

- Beecroft, Cheltenham, Epping, Macquarie University, Macquarie Park, North Ryde Chatswood.

However, this time you need to include the train line information, and since this route uses two different train lines, the returned ArrayList should contain:

- T9 towards Gordon from Hornsby
- Beecroft
- Cheltenham
- Epping
- Metro towards Chatswood from Epping
- Epping
- Macquarie University
- Macquarie Park
- North Ryde
- Chatswood

The information about the train should be written in the form of

<Line Name> towards <Line End> from <Line Start>

where

- <Line Name> is the name of the train line (e.g. T1, T2, T3)
- <Line End> is the final station on the train line
- <Line Start> is the first station on the train line

The station where the transfer occurs should also be listed twice. The train line names are given to you in the lines_data.csv file. As discussed in Section 2.2, the information about the train lines can also be found in lines_data.csv, but the information about which station belong to which line is found in station_data.csv. You need to process these CSV files in the appropriate methods (the method readLinesData to process the train line CSV was added to RailNetworkAdvanced.java).

Furthermore if the route was reversed, that is, if you are asked to provide the route between Chatswood and Beecroft, then you need to return the ArrayList containing

- Metro towards Epping from Chatswood
- Chatswood
- North Ryde
- Macquarie Park
- Macquarie University
- Epping
- T9 towards Hornsby from Gordon
- Epping
- Cheltenham
- Beecroft

Notice how the train line information is now reversed.

If two (or more routes) are possible, then you need to return the route with the least number of stops (as the name of the method suggests). For example, to travel from Blacktown to Parramatta, three routes are possible:

- T1 towards Chatswood from Emu Plains
- T1 towards Chatswood from Richmond
- T5 towards Leppington from Schofields

The first T1 line requires only 4 stops: Blacktown, Seven Hills, Westmead, Parramatta, whereas the other lines require stops at Toongabbie, Pendle Hill, and Wentworthville. Therefore you should return the route with only 4 stops.

For Stage 4, you are allowed to add classes to help you write your code, but you should not modify the constructor RailNetworkAdvanced() in RailNetworkAdvanced.java (it now takes three String as input, corresponding to the three csv files). The method readLinesData was added to process the extra csv file, which you should complete.

Note: for this part of the assignment, since multiple optimal routes are possible for some combinations of stations, the test cases that we use to check your code should only include routes where there is only one optimal solution.

5. Report component

For each Stage, you are required to submit a report explaining your approach in designing the algorithm to solve the given problems. The discussion should include:

- A discussion on the general approach of the algorithm that you are using (exhaustive search, greedy, dynamic programming).

- An explanation on how the algorithm works, including its name if it is an existing algorithm (use an example if this helps explain your code).
- An explanation for parts of your code that may help in understanding your implementation (do not write the entirety of your code without any explanation!).
- (for Stage 4) An explanation of any classes or data structures that you added to the project and how you use them in the algorithm.

Please do not simply rewrite your code with comments and submit it as the report, since this will not be considered as a report. As a guildeline, your report should be between 1-3 pages for Stages 1 and 2, and between 2-6 pages for Stages 3 and 4.

6. Group Work for Stages 3 and 4

In order to attempt Stages 3 and 4 of the assignment, you must form a group of 2 or 3 students, and you must have already submitted a sufficient attempt for Stages 1 and 2. If you become a member of a group but did not submit a sufficient attempt for Stages 1 and 2, then you will not receive the marks for Stages 3 and 4. If you do not wish to attempt Stages 3 and 4, then you do not need to form a group.

To choose your group, a group selector tool will be made available on iLearn. Only one submission per group is required.

7. Warnings and Restrictions

- You must not use any Java libraries outside of java.io and java.util libraries.
- You can reuse code from lectures, textbook, or even the web, but you must cite your reference and explain how they work in your report.
- For Stages 1 and 2, you can discuss general approaches with other students, but you must not show algorithms and code to them (including to your group members for Stages 3 and 4). Please be aware of the university policy regarding plagiarism.

8. Marking Allocation

8.1 Stage 1

- (3 marks) Implementation of routeMinDistance and routeMinStop (to be automarked)
- (1 mark) The report explaining your approach in implementing the above two methods

- (2 marks) Implementation of routeMinDistance and routeMinStop, allowing the removal of stations from the network, and the implementation of findTotalDistance (to be automarked)
- (1 mark) The report explaining your approach in implementing the above three methods

8.2 Stage 2

- (2 marks) Implementation of optimalScanCost (to be automarked)
- (1 mark) Implementation of optimalScanSolution (to be automarked)
- (2 marks) The report explaining your approach in implementing the above two methods

8.3 Stage 3

- (2 marks) Implementation of computeAllRatio (to be automarked)
- (1 mark) The report explaining your approach in implementing the above method

8.4 Stage 4

- (3 marks) Implementation of routeMinDistance (to be automarked)
- (2 marks) A description of the algorithm used for the above method, and details of your implementation

9. Submission and Due Dates

9.1 Stage 1 and Stage 2 (due 9am Tuesday, Sept 24th)

Please zip your Java files and your report together and submit it into Assignment 1 Part 1 submission box on iLearn.

9.2 Stage 3 and Stage 4 (due 9am Tuesday, Oct 1st)

Please zip your Java files and your report together and submit it into Assignment 1 Part 2 submission box on iLearn. Only one submission per group is required.

9.3 Additional required submission

If you choose to attempt Stage 3 and/or Stage 4 as part of a group, then each individual member of the group must submit a brief note (about half a page) on the how the work was divided between the group members. Each member should list the his or her main responsibility and state if there are any reasons why another member should receive less marks (in the case of insufficient contribution). Please submit this to the Assignment 1 Part 2 Individual Report submission box.

10. Changelog:

- 26/8: Draft release
- 30/8: Added Stage 3
- 31/8: Added Stage 4 and description of the extra data for it, effectively combining two parts of the assignment into one. The changes are:
  - Added description for lines_data.csv and extra information on how to use the data in station_data.csv for Stage 4.
  - Previously I mentioned that you should not modify Station.java for 'this part of the assignment', but since we now combine the two parts together, this is no longer true. However, you should need to modify Station.java for the first three stages of the assignment, and you probably can do Stage 4 without modifying Station.java (although it might be easier if you do).
- 11/9: Fixed the wrong example in Section 4.2 (shortest route between Hornsby and Epping with Beecroft removed) - thanks to everyone who emailed me about it
- 12/9: Removed the extra stuff I had in assg1p2.Station because of my solution, thanks to *<name withheld>*.
- 12/9: Added information about submission.