COMP333 Assignment 2 Part 1

Macquarie University

October 29, 2019

1. Objectives

The first part of the assignment is designed to test your ability in implementing the concepts and algorithms that you have learned so far to solve practical (and not-so-practical) problems. You are going to be assessed on your ability to:

- recognise the nature of the problem and devising an algorithm to solve it,
- implementing your solution efficiently using the Java programming language, and
- writing an informative report discussing your approach in solving the problem.

2. Structure

The first part of Assignment 2 has two questions, each worth 10 marks, for a total of 20 marks. For each question, you will be given a starting Java template and a JUnit test file that you can use to check the correctness of your code.

The second part of the assignment is independent to the first part and is released separately. Both parts of the assignments have equal weighting, so the 20 marks from this part of the assignment makes 50% of the total marks for the whole assignment. There is no group work component for the first part of Assignment 2.

3. List of Problems

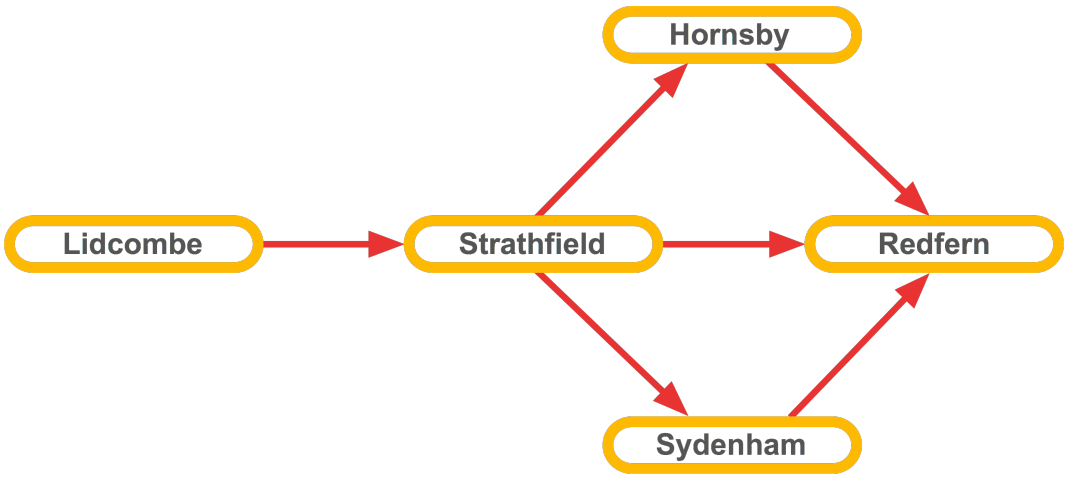
3.1 Problem A: Finding the total number of routes

From the first assignment, you may have noticed that there are many possible routes between two different stations. For example, to travel from Blacktown to Central, the most direct route

would be to use the T1 Line, passing through Parramatta and Strathfield. However, if we were to block one of these stations, you would still be able to find another route between Blacktown and Central, e.g. by going through Hornsby, or Sydenham. It is useful to know the number of different routes between two stations so that we can better plan for an emergency response in case there is a disruption on the rail network.

For this question, we are going to use a simplified model of the Sydney Trains network: instead of considering all 178 stations in the network, we can limit ourselves to only some of the stations in the network. For example, we can consider only stations that sit on two or more rail lines (such as Blacktown or Parramatta), i.e. the major transport hubs in Sydney — after all, it is not very interesting to ask how many routes there are between Emu Plains and Doonside. Another simplification we make is that we consider any *direct route* between two stations to be one-directional. A direct route is a route between two stations in our simplified network that does not have any other station in between.

For example, if we consider only five stations: Lidcombe, Strathfield, Hornsby, Sydenham, and Redfern, then we would have the following simplified network:

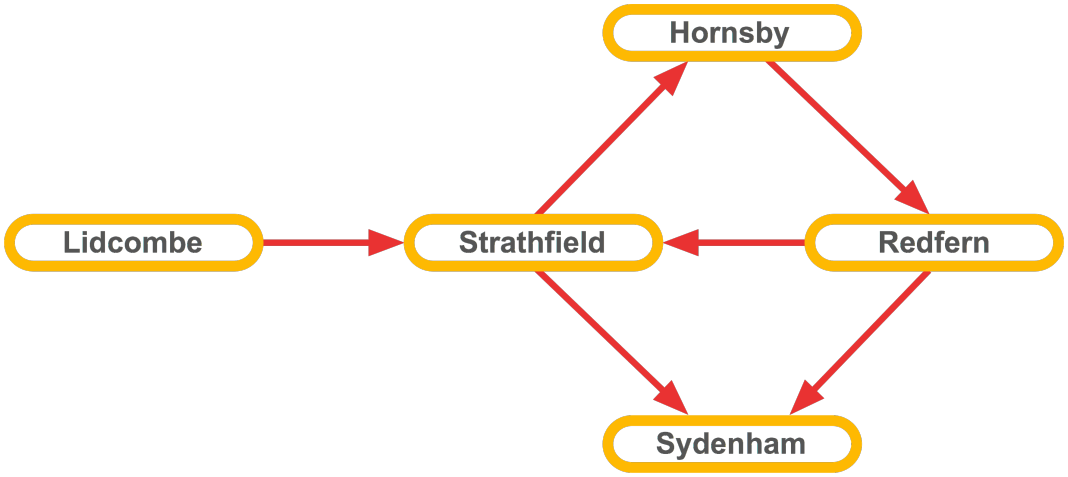


In the above network there are three possible routes from Strathfield to Redfern: the direct route Strathfield→Redfern, Strathfield→Hornsby→Redfern, and Strathfield→Sydenham→Redfern.

We assume that all direct routes in our network are one-directional because otherwise we may end up with an infinite number of routes between two stations. For example, if we were to add a route going from Hornsby to Strathfield in the above network, then there is now an infinite number of routes from Strathfield to Redfern:

- Strathfield→Hornsby→Redfern
- Strathfield→Hornsby→Strathfield→Redfern
- Strathfield→Hornsby→Strathfield→Hornsby→Redfern
- and so on.

Unfortunately, even with this assumption, it is possible that there is an infinite number of routes between two stations. For example, consider the number of routes between Strathfield and Sydenham in the following network:



3.1.1 The Input

For this problem, you will be given a text file containing the direct routes between two stations that forms a simplified network. Each line of the input file will contain two station names separated by one or more empty spaces, denoting that there is a direct route from the first station to the second station. For example, for the first graph, the input CSV would contain

Lidcombe Strathfield
Strathfield Redfern
Hornsby Redfern
Sydenham Redfern
Strathfield Hornsby
Strathfield Sydenham

Note that the station names are arbitrary for this problem, that is, they are not limited to the 178 stations names in the Sydney Trains network. A station name can be any string, although you can assume that it will only contain alphabetical characters and of reasonable length (no station names will have more than 20 characters). You may also assume that the station names are case-insensitive.

3.1.2 The Task

You are tasked to write an algorithm to determine the total number of routes that are possible between any two stations in a simplified network. If there is an infinite number of routes between two stations as described earlier, then you should set the total number of routes to -1.

The starter template for this task is given in the class ProblemA. Note that the constructor for this class accepts a String argument which should be the full path and name of the input file (as was done in Assignment 1). You should write your algorithm in the method

```
// compute the number of routes between any two stations in the network
public HashMap<String,HashMap<String,Integer>> findNumberOfRoutes()
```

returning a HashMap<String,HashMap<String,Integer>> which contains the number of routes between all possible pairs of stations in the network. For example, if a and b are two strings

representing the names of two stations in the network, then $\mathrm{findNumberOfRoutes().get(a).get(b)}$ should return the number of routes from *a* to *b*.

You may, and in fact are encouraged, to write another method to process the input file to create the underlying graph for this problem, similar to how it was done in Assignment 1 (with the function $\mathrm{readStationData()}$). You may also add helper methods if you find the need. The JUnit test will only call the constructor and the method $\mathrm{findNumberOfRoutes()}$.

3.2 Problem B: Electrical connection

The electrical engineers who previously performed the exhaustive scans have a new task at hand and they are requesting your help again, so prepare yourself for another highly practical real-world problem!

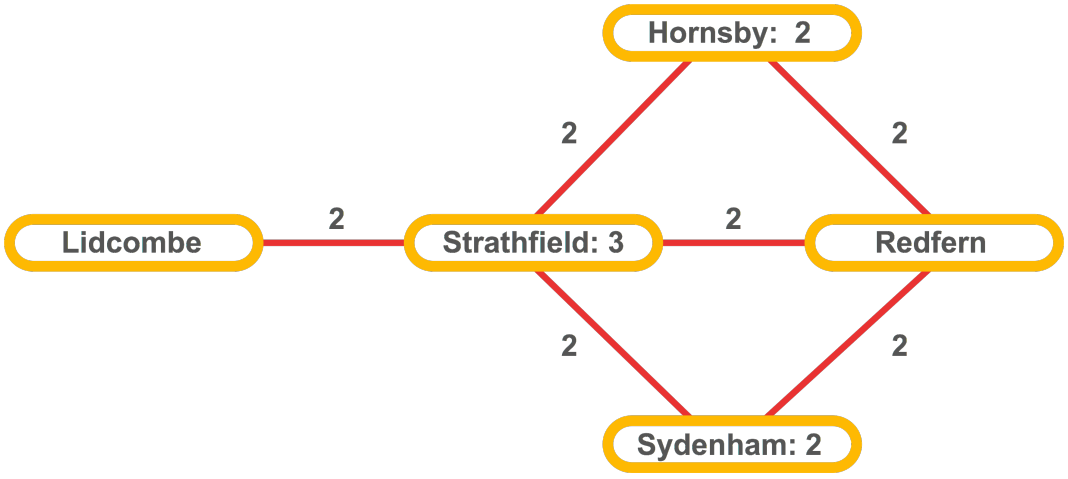
The engineers explained that the electrical power in the rail network is distributed through an electrical network that mirrors the rail network itself, i.e. the electrical power is distributed from station to station via the rail segments that connect them. At each station and rail segment, there is a special device that controls the electrical propagation, such that if this device fails, then no electricity can be transferred via that station or rail segment.

The electrical network was built with redundancy in mind, so each station and rail segment may have one or more backup devices that springs into action whenever the main device fails, ensuring that electricity can still pass via that station or rail segment. Of course if all the devices fail, then no power can be transmitted via that station or rail segment (although the network itself may still be connected, see example below).

The engineers would like you to create a model that can determine the minimum number of device failures that would cause two stations to be disconnected, i.e. such that there is no electrical connection between the two stations.

For this problem, we are also going to use the simplified model that we used earlier, that is, we can limit ourselves to only some of the stations in the network, and the connection between stations do not have to reflect the connections in real life. However, unlike the previous problem, the connection between any two stations are bidirectional.

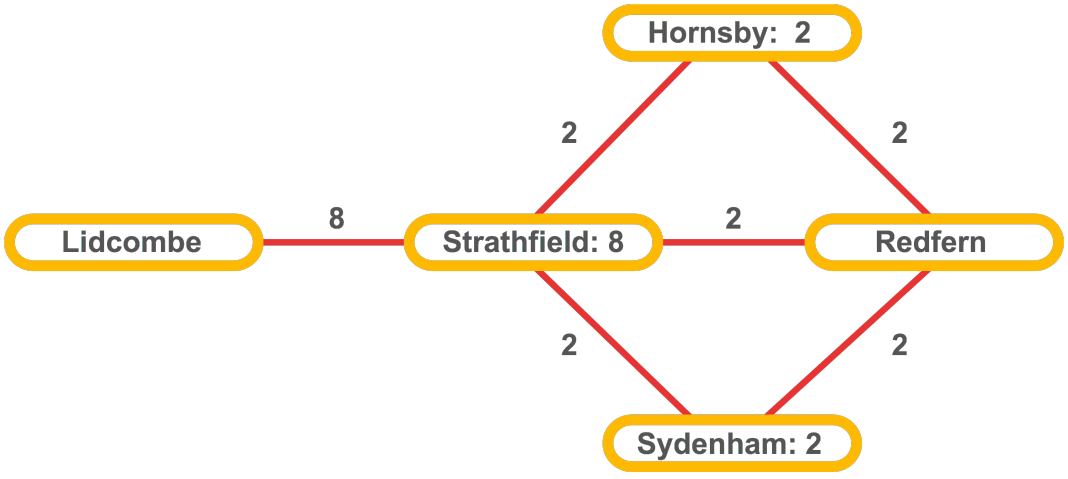
As an example, suppose that in the diagram below we are asked to find the minimum number of devices that when failing will cause Lidcombe to be disconnected from Redfern. Each station and each rail segment has 2 devices each, except for Strathfield which has 3 devices. Note that we do not need to consider the number of devices in Lidcombe and Redfern because the device is only necessary to propagate the electrical connection beyond the station with the device (and once we arrive at either Lidcombe or Redfern, we are done).



The minimum number of device failures that can cause a disconnection between Lidcombe and Redfern is 2, namely the 2 devices on the rail segment between Lidcombe and Strathfield.

For another example, suppose there are now 8 devices between Lidcombe and Strathfield, and also 8 devices in Strathfield. The minimum number of device failures for Lidcombe and Redfern to be disconnected is now 6:

- 2 devices on the rail segment between Strathfield and Redfern
- either 2 devices on the rail segment between Strathfield and Hornsby, or on the rail segment between Hornsby and Redfern, or in Hornsby station,
- either 2 devices on the rail segment between Strathfield and Sydenham, or on the rail segment between Sydenham and Redfern, or in Sydenham station.



If less than 6 devices fail, then the whole network will still be connected. For example, with 5 devices failing, say, 2 at Hornsby, 2 at Sydenham, and 1 on the rail segment between Strathfield and Redfern, there is still a connection between Lidcombe and Redfern.

3.2.1 The Input

For this problem, you must process a text file with the following structure:

- 1st line: two strings m and n separated by a space character, denoting the two stations we want to disconnect
- 2nd line: two integers v and e, respectively denoting the number of stations (excepting the two stations above) and the number of rail segments,
- v lines, where each line has a station name, and the number of devices in that station separated by whitespace (no line should include m or n),
- e lines, where each line has two station names, and the number of devices in that rail segment (separated by whitespace).

For example, in the first diagram above, the input text file should be

```
Lidcombe Redfern
3 6
Hornsby 2
Strathfield 3
Sydenham 2
Lidcombe Strathfield 2
Strathfield Redfern 2
Hornsby Redfern 2
Sydenham Redfern 2
Strathfield Hornsby 2
Strathfield Sydenham 2
```

and for the second diagram, it is

```
Lidcombe Redfern
3 6
Hornsby 2
Strathfield 8
Sydenham 2
Lidcombe Strathfield 8
Strathfield Redfern 2
Hornsby Redfern 2
Sydenham Redfern 2
Strathfield Hornsby 2
Strathfield Sydenham 2
```

3.2.2 The Task

Your task is to write an algorithm to determine the minimum number of devices that when failing will cause the stations m and n in the input to be disconnected. You do not need to specify which devices are failing: you only need to determine the number. Indeed there are many possible combinations as you can see in the above two example.

The starter template for this task is given in the class Problem B, and as in Problem A, the constructor for this class accepts one String argument which is the full path and name of the input file. You should implement your algorithm in the method

```
// compute the minimum number of devices that can fail to cause
// a disconnection of two stations specified in the input file
public Integer computeMinDevice()
```

As before, you may write other methods or class attributes as you see fit. The JUnit test will only class the constructor and the computeMinDevice() method.

4. Java Templates

The starter template for the assignment can be found in

https://github.com/sutantyo/comp333_2019/tree/master/codes/src/assg2p1

The setup is the same as in Assignment 1. You are not required to use Eclipse while developing your solution, but please make sure that the code will compile and run under Eclipse.

5. Report component

For each problem listed above, you are required to submit a report explaining your approach, describing the algorithm that you use to solve the problem. You should begin by outlining the general approach of your algorithm, (e.g. is it using brute-force, dynamic programming, or greedy approach), before giving a more detailed explanation.

If you are using an existing algorithm that we have discussed in class, then you do not need to explain how the algorithm works, but you need to explain how you use it to solve the problem (e.g. how did you modify it for a particular problem). I strongly recommend that you use examples when explaining your algorithm, since this should make it much easier to understand.

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 guideline, you should write between 1-3 pages for each problem, although there is no page limit.

6. 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 reference your source and explain how they work in your report.
- You may discuss the problems with other students, but you should not share algorithms or code with one another. Please be aware of the university policy regarding plagiarism.

7. Marking Allocation

For this part of the assignment, each problem is worth 10 marks for a total of 20 marks. For each problem, you will receive 6 marks from the automarking of your code and 4 marks from the report.

8. Submission and Due Dates

The due date is to be determined later. At the earliest, it should be at the end of Week 13.

9. Changelog:

- 17/10: Draft release
- 30/10: Added Problem B, reduced number of problems to 2