Data Structures and Algorithms Assignment: Semester 1, 2018
Mapping an Alien World
Department of Computing, Curtin University
v1.0

Raymond Sheh
Raymond.Sheh@Curtin.edu.au

March 15, 2018

Abstract

You now have an open-ended opportunity to demonstrate what you have learned thus far, and in the coming weeks, about data structures and algorithms. You will need to make decisions on what to use, when and how, in order to solve a problem. You will also need to decide for yourself how to structure your solution to the problem, what objects to use, how to bring them into your program and save them out and so-on.

Feel free to use the ADTs that you have already written in the practicals however you **MUST** self-cite any work that you are re-using. You may **NOT** make use of the Java implementations of the ADTs, if in doubt please ask.

Note that the assignment specification is deliberately open ended and somewhat ambiguous. Making reasonable assumptions, stating them and then implementing them is part of your job in this assignment! If in doubt, please do ask but be prepared for the answer "Well, what do you think?".

Please start the assignment early! It has been structured in stages with most of what you need to do most of the stages already covered in the lectures so you can be doing groundwork and picking up marks right from the start. If you leave starting the assignment for when we have covered everything you will need for the whole assignment, you **will** run out of time.

GOOD LUCK! *Not that luck should have any part in this of course! ;-D*

1 Introduction

It is 2050. A team of robot soccer players has defeated the reigning human world cup champions[1]. The first elections have been held on Mars. Nuclear fusion reactors show signs of actually breaking even. And a team of planetary scientists have discovered that there is a network of tunnels underneath the surface of Phoebe, one of the more unusual moons of Saturn.

They have sent a team of identical robotic probes to Phoebe to map this network of tunnels. Each robot lands at a random point on the surface of Phoebe, drills down through the surface until it hits the tunnel network, then randomly moves through the network building a map. Each robot has sensors that detect various characteristics of the tunnels and their intersections. Oddly enough, they cannot detect each other. The robots regularly upload their maps to Earth for the scientists to process. It is your job to write the program that will process the data from this team of robots and merge it all together.

1.1 Map Details

The map in this case will be a topological map, which is a map consisting of a graph with edges corresponding to tunnels and nodes corresponding to intersections between tunnels. This kind of map

[1] http://www.robocup.org

makes sense for things like communications networks, tunnels and transportation systems. See Section 2. Do not confuse this with a *topographical* map which concerns heights.

In the robots' minds, the tunnel network consists of tunnels and intersections. The robot will only report data when it is at an intersection, thus every tunnel that it reports will have an intersection at each end[2] (even if it is a dead end, that dead end is considered an intersection with only one tunnel leading out of it). When it reaches an intersection, it will randomly pick a tunnel to travel down, thus it may report intersections with many tunnels but yet only have information about one or a few of these tunnels. Each tunnel or intersection encountered is reported once only, even if the robot visits it multiple times.

The heights of the ceilings at various points change and this is measured by sensors on the robots. Also, cameras on the robots can detect the colours of mineral deposits on the ceiling, floors and walls. The robots do not detect any other characteristics about the tunnel system (such as the exact shape of the tunnel or the ordering of tunnel entrances at an intersection).

As it moves randomly through the network, a fully operational robot records only the following information.

- Tunnels:
  - Length
  - Ceiling height
  - Floor colour
  - Wall colour
  - Ceiling colour
- Intersections:
  - Number of tunnels joining at this intersection
  - Ceiling height
  - Floor colour
  - Wall colour
  - Ceiling colour

For the purpose of this assignment, each tunnel and intersection will have a unique combination of these attributes although, as described shortly, the robot may not be able to tell them apart.

1.2 Your job

Each robot generates its own map. Sometimes, one robot will go through a part of the tunnel network that another robot has already covered. It is your job to detect when this has (probably) happened and *merge* the maps together. See Section 2 for an example.

1.3 Robot Sensors

The robot's sensors measure the attributes of the tunnel network with the following characteristics.

1.3.1 Tunnel Length Sensor

Reports the length of the tunnel as a real number in meters (m). The tunnels may be up to 100 kilometers long. The sensor is accurate to 1%.

1.3.2 Ceiling Height Sensor

Reports the height of the ceiling as a real number in m. The height may be up to 50 m high. The sensor is accurate to 1 mm.

[2] We'll conveniently ignore the first tunnel the robot hits when it bores through the surface.

1.3.3 Number of Tunnels Sensor

Reports the number of tunnels that join at this intersection. This number will be an integer that will either be 1 (ie. actually a dead-end) or between 3 and 10 inclusive. (An intersection with 2 tunnels is simply a bend in a tunnel and for our purposes we don't care about bends.) This sensor is accurate to a single tunnel (ie. perfect).

1.3.4 Colour Sensors

3 separate sensors report the colours of floors, walls and ceilings. They all report the colour as one of eight and are always accurate:

- Black
- White
- Red
- Green
- Blue
- Yellow
- Silver
- Gold

Note that over time, robots may become damaged or dirty and thus some of these sensors may stop reporting data. Therefore, in later stages of this assignment, while the tunnels and intersections will still *actually* be unique, the robot may be unable to detect length, height and/or one or more of the colours. Thus you may encounter data from robots where it is not possible to uniquely identify a tunnel or intersection.

Also note that length and height attributes have an accuracy. This means that if two robots measure the same attribute, they may come up with slightly different numbers, each within the prescribed level of accuracy of the actual attribute. You will need to consider this when detecting if intersections reported by two different robots are actually the same or not.

1.4 Inputs and Outputs

Your program is completely file based, with no interactive input from a user. Instead, your program will take an arbitrary number of files (names, possibly with associated local or global paths) as command line arguments. Each will be a CSV file that represents a map. The file names will start with the string "robot", the string between the end of "robot" and ".csv" will be the name of the robot. These arguments appear in arbitrary order.

Your program will write one file to the working directory. It will have a name that is "combined-" followed by your student number, followed by a hyphen ("-"), followed by the date and time[3] in YYYY-MM-DD-hh-mm-ss format, followed by ".csv". For example, if my student number was 99999999 and I started the program at 3.32pm, 25 seconds on the 1st of April 2018, the filename would be `combined-99999999-2018-04-01-15-32-25.csv`.

The CSV files input to and output from your program will have columns, each with the following data.

- "t" for tunnel or "i" for intersection.
- A string representing:
  - For intersections, a string serial identifier for this intersection. See section below on serial identifiers for more details.

[3]We're not going to be too strict on the timing, just read the time of day at some point early in the execution of your program and use that in your filename.

  - For tunnels, two serial identifiers separated by a hyphen ("-") that represent the serial identifiers of the intersections at each end of this tunnel. The two serial identifiers are reported in **alphabetic**[4] order.

- A string representing a positive number that is:
  - For tunnels, the length of the tunnel in m.
  - For intersections, the number of tunnels meeting at this intersection.
- A string representing a positive number that is the ceiling height in m.
- A string that is the floor colour, which will be one of the colours mentioned above.
- A string that is the wall colour, which will be one of the colours mentioned above.
- A string that is the ceiling colour, which will be one of the colours mentioned above.

The lines coming in from the robot will be in arbitrary, random order. In particular, there is no guarantee that a line denoting an intersection will appear in the file before a line denoting a tunnel that starts or ends at that intersection.

Your program must output its lines in order. This will mean that intersections shall be reported first, in **alphabetic** order of serial identifier, followed by tunnels, also in **alphabetic** order of serial identifier. If two tunnels have the same endpoints (and, hence, same combination of serial identifiers), they shall be sorted **numerically**[5] by the length of the tunnel, if this is identical then they shall be sorted **numerically** by the height of the ceiling. If these are also the same, sorting shall proceed to floor, wall and ceiling colour **in the order shown in Section 1.3.4**.

It is possible for one or more of length, height or colour attributes to be missing (because the sensor that provides it was dirty or broken). Its entry in the file will appear as a single underscore ("_"). For the purpose of sorting, underscores shall appear before any other character.

1.5 Serial Identifiers

The serial identifiers coming into your program from each robot will be an arbitrary integer. Each intersection in a file will have a serial identifier that is unique within that file. These identifiers should be considered random. There is no guarantee that the same intersection in the world will have the same serial identifier between multiple files. Likewise, there is no guarantee that the same serial identifier appearing in two different files will refer to the same intersection. Therefore do not attempt to use the identifier to match intersections between robots.

When you write your output file, each intersection will have a serial identifier that is the concatenation of the robot name (taken from the name of the input file as described above) and the serial identifier assigned to that intersection by that robot. If you are reporting an intersection that you think multiple robots have seen, you will assign that intersection a serial identifier that is the concatenation of these identifiers, in alphabetic order, separated by a colon (":"). For example, an intersection that was assigned serial identifier "23" by robot "alpha" and "36" by robot "beta" will be assigned the serial identifier "alpha23:beta36" by your program. You will then name the tunnels accordingly. Yes, this can result in some pretty long serial identifiers!

2 An Example

Here is a graphical example of what the robots will need to do. Figure 1 shows the tunnel network while Figure 2 shows the areas that robots "alice" and "bob" have mapped (shown in pink and blue outlines respectively).

Here are the two files that would be input into your program from these two robots.

[4] When applied to numbers, alphabetic order means comparing each character one at a time. For example, "1000, 2, 300" is considered in-order.

[5] Numbers sorted in numeric order mean that the values of the numbers are compared. For example, "2, 300, 1000" is considered in-order.

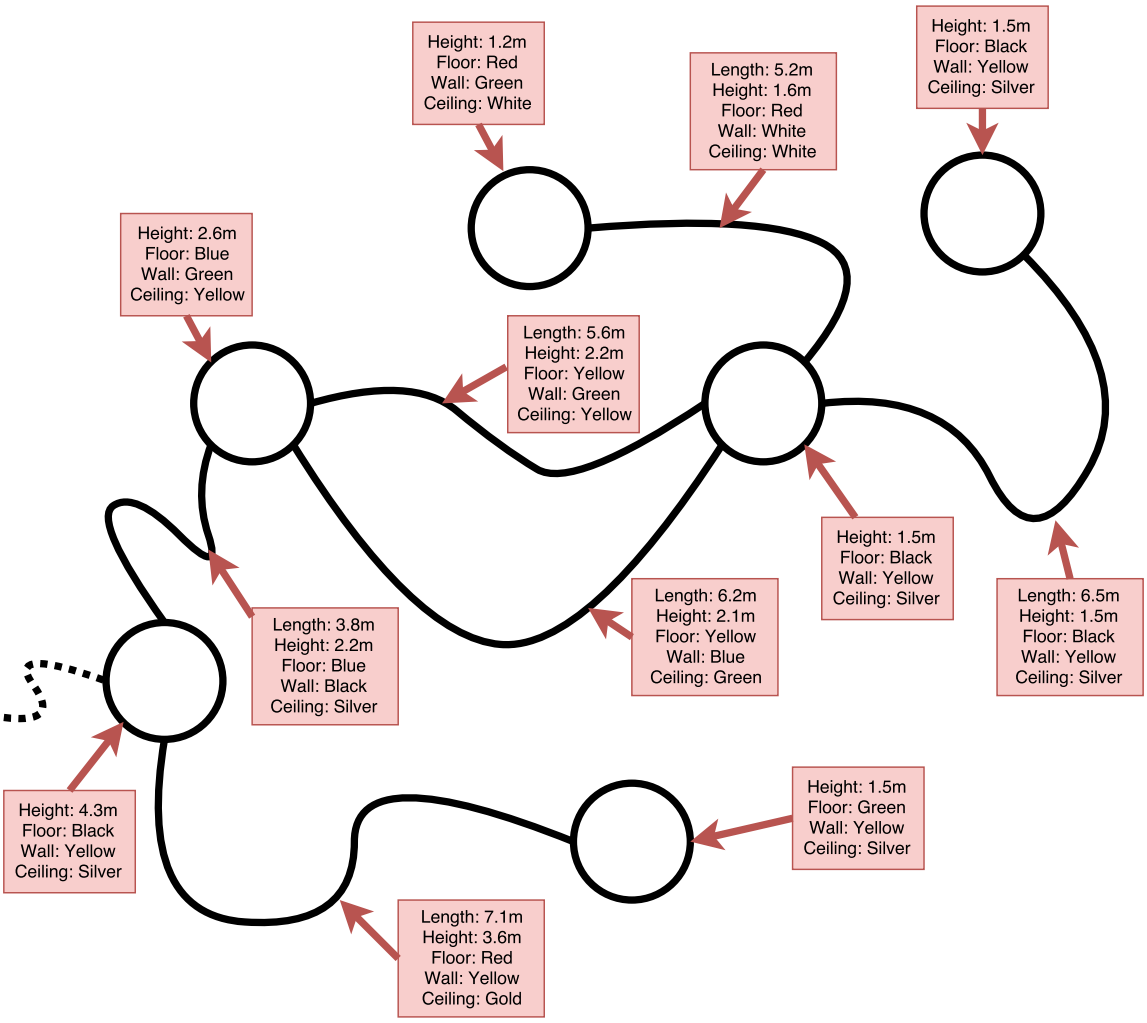
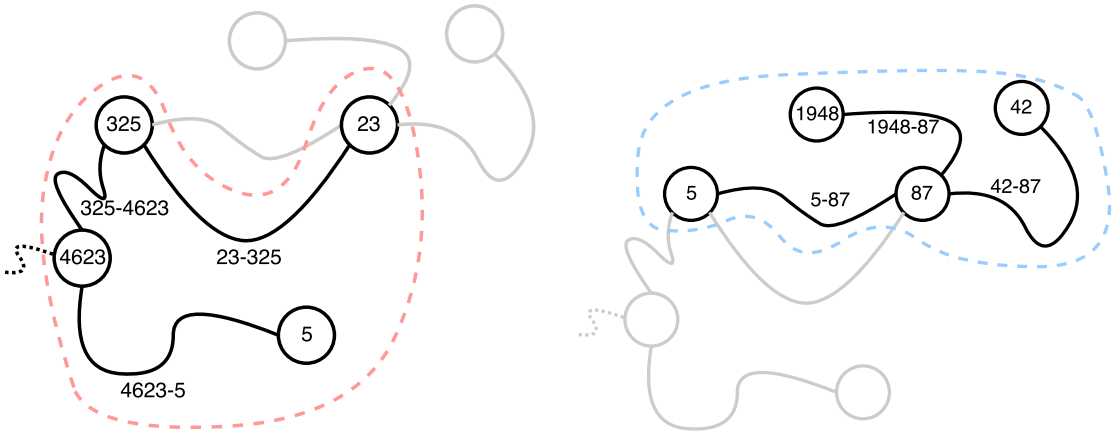


Figure 1: A schematic map of an example tunnel network with intersections represented by circles and tunnels by lines. Dotted line represents a tunnel going off to another part of the tunnel network that won't be explored in this example. Call-out boxes indicate the properties of each intersection and tunnel. Note that this is just an example, the files that you will get in the course of this assignment may not necessarily correspond to this map.



Alice Bob

Figure 2: The portions of the map that robots Alice (left) and Bob (right) have seen. The numbers represent the Serial Identifiers that each robot has assigned to the intersections and tunnels.

robotalice.csv

```
i,325,3,2.6,blue,green,yellow
t,23-325,6.2,2.1,yellow,blue,green
i,23,4,1.5,black,yellow,silver
t,325-4623,3.8,2.2,blue,black,silver
i,4623,3,4.3,black,yellow,silver
t,4623-5,7.1,3.6,red,yellow,gold
i,5,1,1.5,green,yellow,silver
```

robotbob.csv

```
i,1948,1,1.2,red,green,white
t,1948-87,5.2,1.6,red,white,white
i,42,1,1.5,black,yellow,silver
t,42-87,6.5,1.5,black.yellow.silver
i,5,3,2.6,blue,green,yellow
t,5-87,5.6,2.2,yellow,green,yellow
i,87,4,1.5,black,yellow,silver
```

We expect the output of your program to be as follows (with your filename as per the specifications above). Note that because of the sorting requirements, there is only one complete, correct output for any input[6].

combined-99999999-2018-04-01-15-32-25.csv

```
i,alice23:bob87,4,1.5,black,yellow,silver
i,alice325:bob5,3,2.6,blue,green,yellow
```

[6]Having said that, in some of the earlier tasks in Section 3 we will accept intermediate solutions that do not implement the full program. These are clearly indicated as such in the task specification.

```
i,alice4623,3,4.3,black,yellow,silver
i,alice5,1,1.5,green,yellow,silver
i,bob1948,1,1.2,red,green,white
i,bob42,1,1.5,black,yellow,silver
t,alice23:bob87-alice325:bob5,5.6,2.2,yellow,green,yellow
t,alice23:bob87-alice325:bob5,6.2,2.1,yellow,blue,green
t,alice23:bob87-bob1948,5.2,1.6,red,white,white
t,alice23:bob87-bob42,6.5,1.5,black.yellow.silver
t,alice325:bob5-alice4623,3.8,2.2,blue,black,silver
t,alice4623-alice5,7.1,3.6,red,yellow,gold
```

---

3 Your Tasks

In this section, we will take you through the implementation of your assignment to write **one** program that performs this map merging. We will start with very basic tasks and each stage is worth additional marks. **You do not need to complete all of the tasks to pass** although the more tasks you do, the better your marks will be.

A program that correctly satisfies a task below automatically satisfies all the previous tasks. If a task is in error, unless it creates knock-on problems you will only lose marks for that task. For example, if your program completes all the tasks but does not use the correct filename, you will only lose marks once, for task 3.1. However, if your program is unable to properly write files out, you will lose the "my program works!" marks for all sections where you are expected to write out files (all of them!).

For all tasks, your submission must satisfy the following for full marks. If you are in doubt as to what any of these constitutes, bring us some samples (eg. your tutorial submissions) and ask us to check.

- Be written in Java or Python to run on the latest versions available on `saeshell01p.curtin.edu.au`.
- Be structured into separate classes in a sensible manner.
- Be documented according to Section 4.1.
- Follow the Curtin Department of Computing coding standard as taught in OOPD (for Java) or PEP 8 (for Python).
- Include appropriate comments.
- Be accompanied by appropriate pseudocode, as comments in your code (up to you if you want to make this a block before each method/function or inlined into your code).
- Be accompanied by appropriate test harnesses for each class.
- Compile and run without errors or warnings (using `javac *.java` if your submission is in Java) on `saeshell01p.curtin.edu.au`. You may develop on your laptop, home computer, etc. but it **must** satisfy this requirement on `saeshell01p.curtin.edu.au`. Test early, we will not be granting extensions because at the last minute you find that your program, which worked fine at home, doesn't work on `saeshell01p.curtin.edu.au`.
- Be appropriately efficient for the task at hand in both space and time and in terms of choice of data structures/algorithms as well as implementation. The tunnel network might span the entire moon of Phoebe and you might have data from thousands of robots.

3.1 Reading and writing files with the correct name: 10%

Your first task is to write a program that just reads in the input file from one fully functioning robot and writes it straight out to disk with the correct filename. You will receive the 10% for this task if the file is just written out unchanged or if the file also satisfies one or more of the following sections.

For example, given the input file:

robotalice.csv

```
i,325,3,2.6,blue,green,yellow
t,23-325,6.2,2.1,yellow,blue,green
i,23,4,1.5,black,yellow,silver
t,325-4623,3.8,2.2,blue,black,silver
i,4623,3,4.3,black,yellow,silver
t,4623-5,7.1,3.6,red,yellow,gold
i,5,1,1.5,green,yellow,silver
```

We expect either:

combined-99999999-2018-04-01-15-32-25.csv

```
i,325,3,2.6,blue,green,yellow
t,23-325,6.2,2.1,yellow,blue,green
i,23,4,1.5,black,yellow,silver
t,325-4623,3.8,2.2,blue,black,silver
i,4623,3,4.3,black,yellow,silver
t,4623-5,7.1,3.6,red,yellow,gold
i,5,1,1.5,green,yellow,silver
```

Or a file satisfying one of the following tasks.

3.2 Writing files with the lines correctly sorted: 15%

As above but the lines should be sorted according to the specifications. Note that you are not just sorting lines purely alphabetically (doing **that** will only get you 2% of these marks).

For example, given the input file:

robotalice.csv

```
i,325,3,2.6,blue,green,yellow
t,23-325,6.2,2.1,yellow,blue,green
i,23,4,1.5,black,yellow,silver
t,325-4623,3.8,2.2,blue,black,silver
i,4623,3,4.3,black,yellow,silver
t,4623-5,7.1,3.6,red,yellow,gold
i,5,1,1.5,green,yellow,silver
```

We expect either:

combined-99999999-2018-04-01-15-32-25.csv

```
i,23,4,1.5,black,yellow,silver
i,325,3,2.6,blue,green,yellow
i,4623,3,4.3,black,yellow,silver
i,5,1,1.5,green,yellow,silver
t,23-325,6.2,2.1,yellow,blue,green
t,325-4623,3.8,2.2,blue,black,silver
t,4623-5,7.1,3.6,red,yellow,gold
```

Or a file satisfying one of the following tasks.

3.3 Writing files with the correct intersection names: 5%

As above but the lines should have the correct intersection names. Note that this also means you will need to replace the names of the tunnels (because they are named for the intersections at both ends). You may implement this using text only manipulation.

For example, given the input file:

robotalice.csv

```
i,325,3,2.6,blue,green,yellow
t,23-325,6.2,2.1,yellow,blue,green
i,23,4,1.5,black,yellow,silver
t,325-4623,3.8,2.2,blue,black,silver
i,4623,3,4.3,black,yellow,silver
t,4623-5,7.1,3.6,red,yellow,gold
i,5,1,1.5,green,yellow,silver
```

We expect either:

combined-99999999-2018-04-01-15-32-25.csv

```
i,alice23,4,1.5,black,yellow,silver
i,alice325,3,2.6,blue,green,yellow
i,alice4623,3,4.3,black,yellow,silver
i,alice5,1,1.5,green,yellow,silver
t,alice23-alice325,6.2,2.1,yellow,blue,green
t,alice325-alice4623,3.8,2.2,blue,black,silver
t,alice4623-alice5,7.1,3.6,red,yellow,gold
```

Or a file satisfying one of the following tasks.

3.4 Using node data structures: 20%

As above but your implementation must read the input file into a (set of) appropriate data structures and then write them out from those data structures. The data structures must represent the links between tunnels and intersections in a sensible manner. The inputs and outputs will be as in Section 3.3.

3.5 Matching data from two robots: 10%

Your program must now be able to take multiple input files, from an arbitrary number of robots. **Unlike in the specification, for this task you can assume that if more than one robot sees the same tunnel or intersection, they will report identical values for those tunnels/intersections.** Your program must use sensible data structures to store the information, find matching intersections and tunnels and output a file with the correctly merged entries. Section 2 illustrates this task.

3.6 Dealing with inaccuracies: 10%

As above but now robots have potentially inaccurate sensors and thus to determine if two robots have seen the same tunnel or intersection, you will need to perform the matching with awareness of the inaccuracies in the sensors.

3.7 Dealing with missing data (simple): 10%

As above but now robots may be damaged or dirty and thus some elements may be "_". In this task we will make life a bit easier and only consider the case that the remaining data in the tunnel or intersection entry is still enough to unambiguously match to a corresponding entry from another robot.

3.8 Dealing with missing data (difficult): 20%

As above but now some of the entries will actually be ambiguous. For example, one robot may report two intersections where there is enough data missing that it is impossible to directly match them with corresponding intersections from another robot.

Instead of just looking at a single intersection or tunnel and finding a matching one in another robot's data, you will now need to also consider the tunnels and intersections connected to it.

4 Submission

Submit electronically via Blackboard. Please test on `saeshell01p.curtin.edu.au` and submit early and often. Only your last submission will be marked. Do not submit your last version late. Read the submission instructions very carefully.

You should submit a single file, in the zip (`.zip`) or gzipped-tar (`.tar.gz`) formats. The file must be named `DSA_Assignment_<id>.zip` or `DSA_Assignment_<id>.tar.gz` where `<id>` is your student ID. You should use underscores, not spaces, in your filename.

Your file must contain the following and only the following. Make sure that your file contains what is required. **Anything not included in your latest uploaded submission by the due date and time will not be marked.** This includes anything provided later *or anything provided in any earlier submission that was since replaced.* It is your responsibility to ensure and double check that your submission is complete and correct.

- Your source code. This means all .java or .py files needed to run your program.
- Your test harnesses. This is particularly important if we find your code does not work properly and need to figure out which bits are working so we can give you partial marks. A test harness for class `X` should be called `UnitTestX`.
- Documentation for your code, as described below.
- A signed and dated cover sheet. These are available from the "Computing Colloquium" Blackboard unit or from the Computing reception (building 314, 3rd floor). You can sign a hard copy and scan it in or you can fill in a soft copy and digitally sign it.

Do not include any .class files, Makefiles (or similar) or files from your IDE/editor/operating system. Be aware that some operating systems hide files, this is not an excuse for including them. Do not put anything into directories/folders, your submission should decompress into the current working directory, ready to compile (Java) or run (Python).

As including un-necessary files or directories demonstrates either a lack of understanding of how your computer works, or careless reading/following of the assignment specifications, there is a 10% penalty for un-necessary files or directories in your submission.

4.1 Documentation

You will need to submit documentation in PDF format that includes the following.

- An overview of your code, explaining any questions that the marker may have. This is supplemented by the comments in your code. In general, if the marker is looking at your code and isnt sure why you have done something in that particular way, they will check your documentation to see if they can find an explanation. Using an automated documentation system like Javadocs may be very helpful. It is not required, though.

- UML diagrams of your classes.
- A description of any classes you have, you need to let us know not only what the purpose of that class is but why you chose to create it. As part of this, also identify and justify any places where it was possibly useful to create a new class but you chose not to, especially when it comes to inheritance.
- Justification of all major decisions made. In particular, when you choose an ADT, underlying data structure or an algorithm, you need to justify why you chose that one and not one of the alternatives. These decisions are going to be of extreme importance in this assignment.

4.2 Marking

Within each of the tasks in Section 3, marks will be allocated as follows:

40% Decisions made. This is mainly things like choosing the right class breakdown, ADT, underlying data structure and algorithm. Note that you will only get these marks if you adequately justify your decisions and properly implement them. So, for example, if you choose an algorithm and adequately justify it but arent actually able to implement it youll only get part of the marks. Any justification should be presented in the documentation, which means that your documentation is worth 40% of your marks for this assignment. Start documenting early.

30% Code testing. Well have a number of tests to run, and you get marks proportional to how many tests your code passes. If your code passes all of the tests, then you will get all of these 30 marks.

30% Implementation of the classes. It also includes implementing all required methods associated with these classes. We will use unit testing as well as looking at code quality; your test harnesses will have a big impact on these marks.

- Marks will be deducted for not following specifications outlined in this document, which includes incorrect submission format and content and using built-in Java ADTs.
- If the cover sheet isnt provided with your submission, your submission will not be marked and you will be awarded zero (0). If you forget to submit the cover sheet you will be allowed to submit it separately to the unit coordinator (by e-mail or in person) but will lose 5%.

The aims of this marks breakdown is as follows:

- To reward you for good design decisions that you are able to demonstrate by implementing them.
- To let you score some marks if you get the program working but make with poor decisions. If all of your decisions are based only on ease of implementation you can still score the 30 marks from the implementation category and will score most of the marks from testing, meaning you can pass if all of your code works perfectly and is of sufficient quality. Its taking a risk, though.
- To promote good testing. Industry is repeatedly telling us they are really looking for students who can properly test their code.
- To teach you to follow specifications carefully.
- To reward students who have been serious about doing their practical work.

4.3 Requirements for passing the unit

Please note: As specified in the unit outline, it is necessary to have attempted the assignment in order to pass the unit. As a guide, you should score at least 15% to be considered to have attempted this assignment. We have given you the exact mark breakdown above. Note that the marks indicated in this section represent maximums, achieved only if you completely satisfy the requirements of the relevant section.

Plagiarism is a serious offence. This assignment has many correct solutions so plagiarism will be easy for us to detect (and we will).

For information about plagiarism, please refer to `http://academicintegrity.curtin.edu.au`.

In the case of doubt, you may be asked to explain your code and the reason for choices that you have made as part of coding to the unit coordinator. A failure to adequately display knowledge required to have produced the code will most likely result in being formally accused of cheating.

Finally, be sure to secure your code. If someone else gets access to your code for any reason (including because you left it on a lab machine, lost a USB drive containing the code or put it on a public repository) you will be held partially responsible for any plagiarism that results.

4.4 Late Submission

I know for the vast majority of you, I dont need to tell you what is in this section. But every semester, someone always tries to see what they can get away with here. For the benefit of fairness to the vast majority of you who do the right thing, we need to make the rules surrounding late submission clear

As specified in the unit outline, you must submit the assignment on the due date. Acceptance of late submissions is not automatic and will require supporting documentation proving that the late submission was due to unexpected factors outside your control. See the unit outline for details as to the procedure for requesting that an assessment be accepted after the due date.

Note that external pre-scheduled commitments including, but not limited to, work, travel, scheduled medical, sporting, family or community engagements are not considered unexpected factors outside your control. If you know you have, or are likely to have, such engagements and that they may affect your ability to complete the assignment, you will be expected to have planned your work accordingly. This may mean that you need to start and/or complete your assignment early to make sure that you are able to hand it in on time.

Also note that IT related issues are almost never a valid excuse. These include computer crashes, hard disk corruption, viruses, losing computers/storage media, networks going down (even if it is Curtins network - outages of the entire Curtin network of more than 24 hours may be considered depending on circumstances) or the like. As IT professionals in training, you are expected to have suitable backups and alternative ways of getting your assignment completed in the event that any IT problems are encountered. You are also expected to submit your assignment ahead of time to allow for unforeseen issues.

In the event that you submit your assignment late and are deemed to have a valid excuse, you will be penalised 10% (that is, 10% out of 100%, not out of what you would have received) per calendar day that you are late, up to a maximum of seven (7) calendar days. Any work submitted after this time will not be marked and you will automatically fail the unit. Note that if you are granted an extension you will be able to submit your work up to the extended time without penalty this is different from submitting late.

Note that the requirements for passing this unit are applied after penalties. An assignment normally scoring 20% that is submitted one day late, even with a valid excuse, will score only 10% and thus not satisfy the requirements for passing this unit.

4.5 Clarifications and Amendments

This assignment specification may be clarified and/or amended at any time. Such clarifications and amendments will be announced in the lecture and on the units Blackboard page (not necessarily at the same time and not necessarily in that order). These clarifications and amendments form part of the assignment specification and may include things that affect mark allocations or specific tasks. It is your responsibility to be aware of these, either by attending the lectures, watching the iLecture and monitoring the Blackboard page.

Good Luck!