Data Structures and Algorithms

Assignment v1.0

Semester 2, 2022

Department of Computing
Curtin University

1 Introduction

In practicals you have implemented and learned about a number of algorithms and ADTs and will be implementing more of these in the remaining practicals. In this assignment, you will be making use of this knowledge to implement a system to explore and compare a variety of ADT implementations. Feel free to re-use the generic ADTs from your practicals. However, remember to self-cite; if you submit work that you have already submitted for a previous assessment (in this unit or any other) you have to specifically state this. Do not use the Java/Python implementations of ADTs – if in doubt, ask.

2 The Problem

This assignment requires the extension of your graph code, or other approach(es), to determine the optimal series of moves to **enter a string into a virtual keyboard**. Movement is limited to up|down|left|right and selecting a key. Some keyboards have modes to enter upper/lowercase and numbers/punctuation. They also can wrap around the left|right|top|bottom or be limited at the edges. Your task is to build a representation of the keyboards and explore the steps for entering strings and find the "best" steps for a given string.

Images of virtual keyboards will be provided. You will need to decide on a way of representing keyboards in files to use as input to the program. You might want to start with 10-17 key **numeric** keyboard.

Your program should be called **keyMeUp.py/java**, and have three starting options:

- No command line arguments : provides usage information
- "-i" : interactive testing environment (`java|python keyMeUp[.py] -i`)
- "-s" : silent mode (`java|python keyMeUp[.py] -s keyFile strFile pathFile`)

When the program starts in interactive mode, it should show the following main menu:

(1) Load keyboard file
(2) Node operations (find, insert, delete, update) – can build/extend keyboards
(3) Edge operations (find, add, remove, update)
(4) Display graph
(5) Display graph information
(6) Enter string for finding path
(7) Generate paths
(8) Display path(s) (ranked, option to save)
(9) Save keyboard

You can structure the menu/UI differently, just make sure at least those feature options are included.

When running in silent mode, you will give the input and output files on the command line.

- keyFile – the file representing the keyboard
- strFile – file containing one or more strings to generate paths for
- pathFile - output the [ranked] paths to a file, with a score/rating.

Once you have a working program, you will showcase your program, and reflect on its performance. This investigation will be written up as The Report.

Remember: think before you code!

3 Submission

Submit electronically via Blackboard.

You should submit a single file, which should be zipped (.zip) or tarred (.tar.gz). Check that you can decompress it on the lab computers. These are also the computers on which your work will be tested, so make sure that your work runs there. The file must be named DSA_Assignment_<id> where the <id> is replaced by your student id. There should be no spaces in the file name; use underscores as shown.

The file must contain the following:

- **Your code.** This means all .java/.py files needed to run your program. Do include code provided to you as part of the assignment if that is required to run your program.
- **README** file including short descriptions of all files and dependencies, and information on how to run the program.
- Your **unit test harnesses**. One of the easiest ways for us to be sure that your code works is to make sure that you've tested it properly. A test harness for class X should be called UnitTestX.
- **Project Report** for your code, as described in Section 3.1.
- A signed and dated **cover sheet**. These are available from Blackboard with the assignment specification. You can sign a hard copy and scan it in or you can fill in a soft copy and digitally sign it.
- **Java Students:**
    - Do not include .class files or anything else that we do not need. We will recompile .java files to ensure that what we're testing is what we're reading. We will use javac *.java to compile your files and run the unit tests by their expected names.

Make sure that your file contains what is required. It is your responsibility to make sure that your submission is complete and correct.

3.1 Project Report

You need to submit documentation in docx or pdf format.

Your **Report** will be around 6-10 pages (excluding UML and Javadocs [if used]) – it can be longer. It should include the following:

- **User Guide:** how to use your simulation, and an overview of your program's purpose and features.
- **Description of Classes:** you need to let us know not only what the purpose of each 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 Decisions:** 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.
- **UML Class diagram**
- **Traceability Matrix** of features, implementation and testing of your code
- **Showcase** of your code output, including 3 scenarios:
  - **Introduction:** Explain the features you are showcasing/investigating through 3 scenarios: e.g. you can compare different keyboards on the same strings, look at the effect of strings with (or without) caps/no-caps/numbers/punctuation, impact of wrap/no-wrap…
  - **Scenario1:** Describe how you have chosen to set up and compare your code run for the showcase. Include commands, input files, outputs – anything needed to reproduce your results. Discuss your results.
  - **Scenario2 & Scenario3:** (as above)
- **Conclusion and Future Work:** Reflect on your implementation and how it might be improved. What further investigations and/or extensions could be added?

3.2 Marking

Marks will be awarded to your submission as follows:

- **[30 marks] Implementation**. We will use unit testing as well as looking at code quality; your included test harnesses/test case specifications are 15 of these marks.
- **[30 marks] Demonstration.** We'll 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.
- **[40 marks] Project Report.** As described in section 3.1
- **[Bonus Marks]** There will be bonus marks available exceptional/additional features. This could include visualisations and additional game functionality.
- Marks will be deducted for not following specifications outlined in this document, which includes incorrect submission format and content and using built-in Java/Python ADTs.

The aims of this marking breakdown is as follows:

- To make sure you can discuss and justify your coding decisions
- To promote good testing. Industry is repeatedly telling us they are really looking for students who can properly test their code.

3.3 Requirements for passing the unit

Students must submit an assignment worthy of scoring 15% to pass the unit.

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.

3.4 Late Submission

If you need extra time, and have a valid reason, you should apply for an extension through Oasis. Late submissions will incur a daily deduction.

3.5 Clarifications and Amendments

This assignment specification may be clarified and/or amended at any time. Such clarifications and amendments will be announced via Blackboard. These clarifications and amendments form part of the assignment specification and may include things that affect mark allocations or specific tasks.