The University of Birmingham
School of Computer Science

Assignment 05

1 Introduction

This assignment will be marked out of 10. Following the disruptions caused by the Coronovirus outbreak, it has been decided to make this assignment **OPTIONAL**

This means that your overall mark from all 5 assignments will be calculated and compared to the mark you would get if only your first 4 assignments were totalled and scaled up to 100%. Your final continuous assessment mark would be the higher of these two values. Thus if $m_1, ..., m_5$ are the marks you achieve (each out of 10) for the 5 assignments, then your final continuous assessment mark (out of 50) for this module would be:

$$\text{Final Continuous Assessment Mark} = \max((m_1 + m_2 + m_3 + m_4 + m_5), (m_1 + m_2 + m_3 + m_4) * \frac{5}{4})$$

Deadline for submission is:

14:00 Friday 27th March

The first assignment was about linked lists and their implementation, and the second about using standard Java Collection classes for Lists, Maps, Sets and Arrays, the third about recursion and binary trees, the fourth about priority queues, external merge sorting, CSV files and file handling. This one is about manipulating graphs and implementing Dijkstra's algorithm.

2 Marking

- To get full marks, you will have to provide working implementations for the two methods where the comment: "`//WRITE YOUR CODE HERE`" appears so that it matches the specification provided in the corresponding Javadoc comments.
- There will be 1 mark if your submission compiles correctly and can be run and passes the tests provided which already pass in the initial version of the assignment files.
- While a set of tests that cover most of the requirements have been provided for you, more tests may be used as part of the marking progress. Therefore you are advised to write your own extra tests.
- If your submission is not structured correctly or does not compile, or crashes or enters an infinite loop before any tests pass, then no marks will be earned.

3 Plagiarism

Plagiarism will not be tolerated: it is unfair to the other students and prevents you from learning, and would give you a mark you don't deserve, and qualifications you don't deserve, hence being unfair to Society as a whole too. Please behave well and reward the module lecturer and TA's by submitting your own, hard work.

All submissions will be checked for copying against other submissions and against sources on the web and elsewhere. Any student found to have

When the module lecturer decides that there is evidence of plagiarism, the case will be forwarded to the Senior Tutor, who will look at the evidence and apply penalties and warnings, or, if necessary, forward this to the University.

- University regulations on plagiarism: `https://intranet.birmingham.ac.uk/as/studentservices/conduct/plagiarism/index.aspx`
- University Library sources on plagiarism: `https://intranet.birmingham.ac.uk/as/libraryservices/library/referencing/index.aspx`

4 Student Welfare

If you miss or are going to miss a deadline, or are getting behind the learning material, for reasons outside of your control, please contact Student Welfare. Occasionally students represent the University in sports, programming competitions, etc., and if this is going to affect your ability to comply with a deadline, Welfare should be informed. It is the Welfare Team, not the teaching team, who can award extensions and cancellations, and devise other measures to cope with adverse situations. It is important to contact welfare as soon as possible when such situations arise.

5 Details

There are four methods to implement for this assignment, all in `Graph.java`. The details and definitive specification are in the JavaDoc in `Graph.java`, but in summary:

- `getStudentID()` and `getStudentName()`: the usual ones to capure your information for marking purposes
- `contractNodeWithTwoEdges(int node)`: Remove the node indicated from the graph and the two edges that connected to it and replace it with a single edge, with a distance that is the sum of the distances of the removed edges, that connects the nodes that this node previously connected with.
- `dijkstra(int node1, int node2)`: Apply Dijkstra's algorithm to find the distance between 2 nodes in the graph.

To assist you there is a set of unit tests provided for you in `GraphTest.java`
You should carefully read through all the code before starting on the assignment.

5.1 Libraries

5.1.1 Hamcrest

One of the tests in `GraphTest.java` requires a Hamcrest matcher that is not available in `Hamcrest-core-1.3`. To make the initial code work, you need to use the full `Hamcrest-all-1.3`. If you did not originally install this version on your own machine, it can be downloaded from `https://www.cs.bham.ac.uk/~aps/javalibs/PreviousLibs`
On the lab machines it is already installed at

```
/bham/pd/packages/java/1.8.0/lib
```

To read more about the power and simplicity of Hamcrest, see:
`https://www.baeldung.com/java-junit-hamcrest-guide`

5.1.2 JUnitParams

The code in `GraphTest.java` requires a new library: `JUnitParams-1.1.1`, which you will have to install on your own machines
You can download that library for your own machines from `https://www.cs.bham.ac.uk/~aps/javalibs`
On the lab machines these are already installed at

```
/bham/pd/packages/java/1.8.0/lib
```

This library supports running the same test multiple different times with different parameters. While you can run parameterized tests in raw `JUnit-4.12`, as indicated in the module lecture notes on testing, it is verbose, complex and rather inelegant. `JUnitParams-1.1.1` provides a much simpler neater way of achieving the same goals.
For futher information about using `JUnitParams-1.1.1`

- `https://github.com/Pragmatists/junitparams/wiki/Quickstart`
- `https://github.com/Pragmatists/JUnitParams/blob/master/src/test/java/junitparams/usage/SamplesOfUsageTest.java`

5.2 Final Instructions

- The precise information about the behaviour required of the missing code is detailed in the Javadoc comments in the files.

- You should not `catch` any exceptions in your code. There is no need for it.
- You should modify the methods `Graph.getStudentId()` and `Graph.getStudentName()` in `Graph.java` to return your student id and name.
- You should **NOT** add any further class variables to `Graph.java`, or change the types or visibility of the existing class variables or methods nor modify any of the other files in the `main` part of the src tree.
- You can add extra **private** methods in Graph.java if you like but please note that they are not needed and my solution does not do so.
- You can (and should!) modify the `GraphTest.java` file and you can add any other test files as you please. Your test files are for your own use and should NOT be submitted.
- You should not modify the package structure or add any new classes that `Graph.java` depends upon to run. Your final submission will be purely the `Graph.java` file, so any modifications outside that file will not be considered and the `Graph.java` file that you submit must work with the other files as they currently stand in the assignment structure.
- You should not use any print statements to standard out or standard error streams: if you want to have some debug output, use the logging calls for Log4j. The logging system will be switched to disable log output when your submission is run for marking purposes.
- For marking purposes, your code will be compiled and executed against a test set in a secure sandbox environment. Any attempts to break the security of the sandbox will cause the execution of your program to fail. This includes infinite loops in your code.
- When you have completed your code changes to your satisfaction, submit your `Graph.java`