Coursework commentary 2018–2019

CO2226 Software engineering, algorithm design and analysis

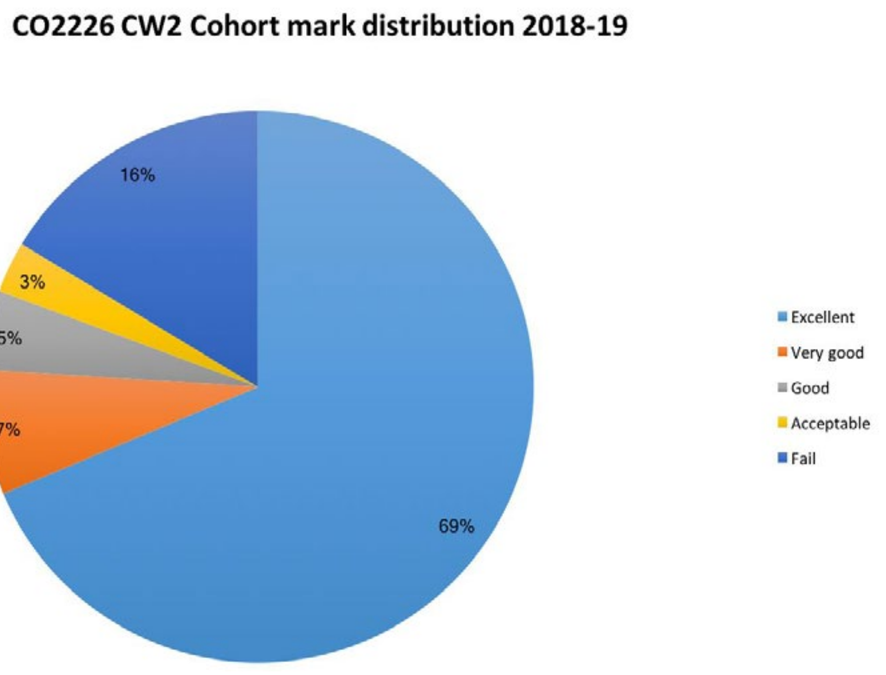
Coursework assignment 2

General remarks

This report provides a general commentary on the questions and the performance of students for the second CO2226 coursework assignment for the academic year 2018–2019. A comment on the answers expected in generic terms will be provided as well as comments on the students' performance and mistakes.

The results were very good, as over 75 per cent of students achieved an excellent or very good mark. Most marks were lost in the final two questions where more calculations were expected; the first five questions were answered well using the methods provided in the coursework assignment.

See cohort mark distribution for 2018–2019 below:



This coursework assignment was a programming exercise where students were expected to answer seven questions. The questions were based on the implementation of Dijkstra's algorithm for the shortest path problem, and fragments of code were already provided; these fragments were supposed to be used in the context of the cities distance problem. Students were expected to modify them and add their own parts of code to answer the questions. Some questions were based on previous ones and the two main criteria (apart from, obviously, correctness) were speed of execution and re-usability of data structure components, so that the same calculations do not have to be made from scratch and the existing calculations can be reused to the maximum level. The result for each question would be an all or nothing mark, depending on whether the result produced by the student's submission was correct or not.

Comments on specific questions

The list below describes common mistakes identified in the submissions; please note that students making these mistakes did not receive a zero mark, but were penalised by having marks deducted from the mark allocated for each question:

- As a result of students using IDEs for writing the code (e.g. NetBeans, Eclipse, IntelliJ, etc.) the IDE was inserting a package statement at the beginning of the code. This line had to be removed to prevent a compilation error.
- The use of IDEs resulted in a structured way of writing code (e.g. code files in a directory called src, data in a directory called data, libraries in a directory called lib and so on), but broke the assumption when running the file that all data files about countries and connection data should be in the same directory as the main file. This was easy to fix, as all that had to be done was to remove the prefix 'src' so that all files can be found in the same directory.
- The coursework assignment document clearly stated that you need to read three files from the command line without their extension. Some students put two of the three files to be read from the command line (again, an easy fix but not in line with what the coursework assignment asked for).
- Some students assumed that the data is to be read from files that will have a .csv or .txt extension – this does not necessarily have to be true, as the file names get their names from the command line and there is no reason to assume a specific format.
- Some students had hardcoded the city IDs as part of the code – the brief clearly states that different data will be used in the test. This not only means that the costs in the `randomGraph` file will be different, but the city IDs might change as well in which case the code completely breaks. The IDs of the cities are to be found dynamically based on the data (the coursework assignment was not about answering specific questions but types of questions).
- Some submissions expected additional information and arguments after providing the filenames (for example, city IDs); the brief was clear in that only the filenames should be provided as arguments and nothing else.
- Do make sure that the path for the files is correct and location independent; a few submissions had hard-coded file paths that included the local hard disk.
- Make sure that the Java statements are correct – some submissions had spelling mistakes in the Java statements (e.g. `Sytem.out.println).`
- A number of students assumed a bi-directional graph, so in the loop for reading the edges file they would have index i running from 1 to N and index j running from i+1 to N. The graph is directional, nowhere in the brief is there a statement about the graph being bi-directional. This was also made clear in the forum.
- Some students assumed that if the city file has, for example, two hundred entries, then this is the maximum for the index for the structure holding city IDs and linking them to names – this is wrong as the actual index (unless you use a more complicated structure) would use the city ID as the index, which means that you need as much space as the highest ID. One can make arguments about using extra space, which can be avoided with the correct choice of data structure, but it does not affect the correctness of results and it does not cause your code to run over the time threshold set.

- When students use arguments as variables, they should not use double quotes around them. Some students were using "args[1]" as an argument for the `FileReader` class and the program was looking for a file called args[1] rather than the file you were passing from the command line.
- Please read the instructions carefully and follow them; there were a few submissions where, although the code ran without problems, the order of command-line arguments was reversed, and the actual code had to be examined to see what was expected.
- Not a problem that will cause issues with the running of the program as such, but if students have a main class and an inner class, you would expect the main method to be in the main and not in the inner class. Some students had the main method in the `graph` class which, although not wrong technically, does not follow the instructions from the coursework brief and adds the main method for a project in an auxiliary, essentially helper, class.
- Do make sure that all classes you are using in your coding exist (are either Java built-in or you have already defined them). Some submissions had classes for which either an import package statement was missing and they could not be recognised, or they were defined in another file and your current program could not recognise them.
- In the question for cities with the minimal number of connections, some of the answers included a pair of cities with direct connections. Please read the coursework brief carefully, as it was clearly stated that shortest paths with a length of one should be discarded.
- Another common mistake that cost full marks to students was that they forgot that inner classes were required (in this case the class graph). If this class was missed out, the program would not compile as the constructor for the `graph` class is used in the program. This means that the examiners cannot run the program at all, resulting in no marks for the student.
- Some students submitted a .zip file with all the coding structure of the IDE – we are not interested in this, the brief clearly specifies that only one file is to be submitted. The examiners will run the code from the command line, so this submission will get you zero marks.
- Some submissions assumed that if a route is mentioned in the `randomGraph` file, then both cities should exist in the cities file. Although it is a reasonable assumption, it does not have to be the case (e.g. there might be a couple of old routes still mentioned, but we are only interested in the cities specified in the main file). This is easy to fix in the code, but it causes runtime errors of null pointer values.

Conclusion

In general, students should make sure that their code compiles without errors – students should pay close attention to any error messages they get, and understand why they are getting them, and what they should do to get rid of them. If students use an IDE for development, they should make sure that their program compiles without errors from the command line before they submit it (examiners will only be using the command line to run your program and will rely solely on Java's built-in classes). Also, students need to think very carefully about what data structures they should be using and why. Effort should be put into understanding what information is required to answer the questions, and how this information can be reused; a program that computes from the beginning information that it had already computed in the past is slower, ineffective, harder to debug and non-intuitive.

In terms of student performance, there were no questions which the vast majority had problems with. Some of them (e.g. Questions 2 and 5) had multiple correct answers and some students returned all of them and others one; both answers resulted in full marks as no specific instructions were given on this requirement. A number of good techniques were used both in the `shortestPaths` as well as the Dijkstra method to avoid direct links and in most cases they worked without problems (although, when you make changes to the code it is always advisable to run them through some extreme scenarios and make sure that the values returned are correct). Overall, most students achieved high marks using the pseudocode provided and submitted robust, well-structured and well-written programs.