COMP3506/7505 Homework Task 4

Due Fri 27 Sep 2019, 5:00pm

10 marks total

Overview

The goal of this problem set is to understand how to apply the algorithms and data structures that have been taught in this course.

You will be required to write a series of algorithms to search through and analyse the data of a social media feed. The following searches will need to be implemented:

- A search to find all posts made by a user between two dates
- A search to find the first post made by a specific user after a specific date
- A search to find the post with the nth highest upvotes
- A search to find all posts containing a specific text pattern

The intention is for most of the preprocessing to be performed in the constructor of the class, allowing calls to these searches to be as fast as possible. Most of these searches are trivial to implement using a brute-force algorithm. The majority of this task's marks will be awarded for choosing (and justifying the use of) efficient algorithms and data structures. Simply implementing a brute force solution will result in very few marks, so you are encouraged to think about how to maximise the efficiency of your solution before you begin writing code.

You have been supplied with a Java file, `FeedAnalyser.java`, which is responsible for loading the data file and performing the searches. Stubs and Javadoc comments for each of the search methods have been provided. Your implementation should strictly adhere to the documentation.

Other files you have been provided with include:

- `FeedItem.java` - a data class for storing information about feed items
- `Util.java` - containing utility methods for performing various functions, in particular for file parsing (the methods in this class have already been used in the base code - you are not required to use them in your solution)

You are permitted to use any programming constructs from the Java Collections Framework (or any other standard Java library) for this task. You should however understand the underlying implementation of any data structures/algorithms you use from the JCF.

Input File

The social media data that you are analysing will be initially stored in a `csv` file. The constructor of `FeedAnalyser.java` is already partially-implemented to load this data from the file. You are required to modify the constructor so this data is loaded into appropriate data structures.
Each line in the file represents a single post in the feed, and is formatted as follows:

```
id,username,date,upvotes,content
```

In this format:

- `id` is a unique integer identifier
- `username` is a string representing the user who posted this data
- `date` is the time the item was posted at (formatted in 24-hour time as `dd/mm/yyyy hh:mm:ss`)
- `upvotes` is the number of upvotes this post received (downvotes are possible and are represented as negative integers)
- `content` is a (possibly very long) string containing the content of the post

For simplicity, you may assume the `username` and `content` fields contain only printable ASCII characters (i.e. characters with ASCII values from 32 to 126 inclusive) and do not contain the comma character. You can also assume that the data file is always formatted correctly so there is no need to implement format validation.

Your Task

1. (4 marks) Implement the four searches as per their Javadoc specifications. You will likely need to modify the constructor and class' member variables to achieve an efficient implementation.
2. (6 marks) Describe any design choices that were made while implementing these searches. In particular you should:
    - Identify the underlying theoretical algorithms and data structures used by your code and any of the classes from the JCF you have used
    - State and briefly explain the worst-case time complexity of the constructor and each of your searches in big-O notation
    - If the worst-case time complexity differs from the expected-case time complexity, also provide the expected-case complexity in big-O notation and explain why these cases differ
    - Justify why the chosen algorithms and data structures were optimal for efficiently implementing each of the searches (you should discuss both the runtime and memory usage of your implementation) - keep in mind that there may be no "perfect" solution and certain implementations may have certain tradeoffs (which you should identify)
    - Compare and contrast your design against other potential implementations (including, but not limited to, brute force implementations or implementations with other tradeoffs)

Constraints

- You must write your solution in `FeedAnalyser.java` - do not modify any other files or introduce new packages
- You may only use standard Java libraries
- You may introduce new member variables or helper methods but these should have the strictest access modifiers possible
- Your answer to question 2 should be no longer than 4 pages (at size 12 font and standard line spacing/margins)

Submission and Marking

Submit two files as a part of your submission. Your solution to question 1 should be in the file `FeedAnalyser.java`. Your answer to question 2 should be in a PDF file named `README.pdf`. Do not submit any other files or directories. To preserve anonymity, please do not include your name in any submitted files (it is okay to include your student number).

Question 1 will be partially marked by an automated test suite with timeouts present on each of the tests. A sample test suite has been provided in `FeedAnalyserTest.java`. This test suite is not comprehensive - there is no guarantee that passing these will ensure passing the tests used during marking. It is recommended, but not required, that you write your own tests for your algorithms. Passing the tests also does not guarantee an efficient solution - tutors *will* make mark deductions if your solution is inefficient. Marks may also be deducted for poor coding style.

You should submit `README.pdf` using Turnitin - this will be manually marked by a tutor. This file should be electronically processed - handwritten solutions will *not* be accepted. Any asymptotic bounds should be as tight as possible. Your analysis should be as concise as possible, while still achieving the level of required detail (marks may be deducted for overly long answers). You are encouraged to support your analysis, justification, and/or comparison with information from other sources, but you must cite these appropriately (IEEE style is recommended). Using LaTeX to write your README is once again recommended but not required.

Late Submissions and Extensions

Late submissions will *not* be accepted. It is your responsibility to ensure you have submitted your work well in advance of the deadline (taking into account the possibility of computer or internet issues). See the ECP for information about extensions.

Academic Misconduct

Students are reminded of the University's policy on student misconduct, including plagiarism. See the course profile and the School web page for more information:
`http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`.