COMP3506 – Assignment 4

s4529458

Due 27/09/2019 5:00 pm

Implementation Details

FeedAnalyser Constructor

This makes use of a HashMap which maps users to a tree map. HashMap is implemented by the Java Collections Framework using a hash table [1]. By default (which we use), this has a load factor of 0.75, meaning the hash-table is resized when 75% of its buckets are full. This is a compromise between time and space costs and results in (amortised) $\mathcal{O}(1)$ insertions and lookups, with linear space usage. In the worst case, it is $\mathcal{O}(n)$ if the hash table needs to be resized.

The TreeMap maps dates to an ArrayList of posts made on that day. This is implemented by the JCF as a red-black tree [2]. This has the property of $\mathcal{O}(\log n)$ insertion and lookup in all cases [3].

Finally, the ArrayList is an array-backed list with constant-time insertions [4].

Suppose there are n FeedItems and the get methods on FeedItem are $\mathcal{O}(1)$. In the constructor, the `while` loop iterates n times, each iteration taking $\mathcal{O}(\log n)$ time because of the TreeMap inseration. The other operations (ArrayList add) are $\mathcal{O}(1)$. As a whole, this loop takes $\mathcal{O}(n \log n)$ and the array sorting algorithm used by Java is bounded by $\mathcal{O}(n \log n)$ [5]. Thus, the constructor is bounded by $\mathcal{O}(n \log n)$ in the worst case.

getPostsBetweenDate

This performs a lookup on the HashMap to get one user's posts, in $\mathcal{O}(1)$ time. Then, we index the nearest index using `subMap()`, `headMap()` or `tailMap()` to get the range of posts between the given dates. This is done using lookups which are always $\mathcal{O}(\log n)$ for a red-black tree since they are balanced [6]. Note that this TreeMap only contains the posts for this user, so will often contain less than n items if there are multiple users posting.

Then, we collect the lists of across all dates in the range into an ArrayList, which takes $\mathcal{O}(k)$ time where k is the number of posts falling within the range.

The algorithm is worst-case $\mathcal{O}(\log n)$ or $\mathcal{O}(k)$, whichever is larger.

getPostAfterDate

This performs one lookup on a HashMap and one loopup on a TreeMap. These are $\mathcal{O}(1)$ and $\mathcal{O}(\log n)$ respectively (with the same caveats as above). This returns an array which is indexed in $\mathcal{O}(1)$ time. Thus, this is $\mathcal{O}(\log n)$ worst-case.

getHighestUpvote

This is just an array indexing which is $\mathcal{O}(1)$ time, using a precomputed sorted list of posts. An integer variable keeps track of which index of the array we are currently up to. This is also incremented each time the function is called.

getPostsWithText

This is an implementation of the Boyer-Moore algorithm to search the text of every post. Suppose we have a pattern of length m, n is the number of posts and k is the maximum text length.

Preprocessing for the Boyer-Moore algorithm takes $\mathcal{O}(m)$ time and this is done once for all texts. The algorithm runs in $\mathcal{O}(m+k)$ if the search pattern does not appear and $\mathcal{O}(mk)$ if it does [7].

Thus, searching every post for this pattern will take $\mathcal{O}(np(mk)+n(1-p)(m+k))$ worst-case where p is the proportion of posts which match the search pattern.

References

[1] Oracle, "HashMap (Java Platform SE 8)," Mar 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

[2] Oracle, "TreeMap (Java Platform SE 8)," Mar 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 3rd ed. MIT Press, 2009.

[4] Oracle, "ArrayList (Java Platform SE 8)," Mar 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/ArraryList.html

[5] Oracle, "Arrays.sort() (Java Platform SE 8)," Mar 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#sort-java.lang.Object:A-

[6] B. Hasti, "CS 367: Red-Black Trees," 2012. [Online]. Available: http://pages.cs.wisc.edu/~skrentny/cs367-common/readings/Red-Black-Trees/

[7] M. A. Sustik and J. S. Moore, "String searching over small alphabets," University of Texas at Austin, Tech. Rep. TR-07-62, Dec 2007.