CSC263 Winter 2021 Problem Set 2

Samar Sabie & Michelle Craig

Due: March 4, 2021 at 11 am ET on MarkUs

Instructions

- You do not need to work with the same partner that you had for PS1. Both group members in a partnership are responsible for what is submitted for every question. Partners are advised to work together on every question – not to divide the questions between you.
- Solutions must be typeset electronically, and submitted to MarkUs as a PDF with the filename **ps2.pdf** Although completely handwritten submissions will receive a grade of zero, you are welcome to draw the diagram required in question 1 by hand and include the scanned picture in your PDF file.
- Your submitted file should not be larger than 9MB. Your file might exceed this limit if you use a word processor like Microsoft Word to create a PDF; if it does, you should look into PDF compression tools to make your PDF smaller, although please make sure that your PDF is still legible before submitting!
- The work you submit must be that of your group; you may not use or copy from the work of other groups, or from external sources like websites or textbooks. We recognize that although it should not be necessary to do so, students may wish to read a different textbook or reference material about course topics. This is not prohibited, but you must declare it. (See the next bullet.) All the information you need has been covered in lectures and/or in our textbook.
- On the front page of your submission, you must list **every** source of information you used to complete this homework (other than your lecture notes, our textbook, or conversations from office hours or lectures). For example, indicate clearly the **name** of every student from another group with whom you had any discussions, the **title and sections** of every textbook you consulted (excluding the course textbook), the **source** of every web document you used or website you visited (other than our own CSC263 site.)
- Questions 1 and 3 will each be worth approximately 25% of the marks. Questions 2 and 4 will each be worth approximately 20% of the marks. Question 5 is worth the remainder.
- **Throughout all questions in this assignment, your algorithms may simply call any algorithm discussed in class or in the textbook. If you would like to use a variant of one of these algorithms, do not rewrite the entire algorithm, but simply explain how your algorithm differs from that known algorithm.**

Question 1

In this question you will design a data structure to support the following ADT.

DATA A collection of Piazza posts for a course where each post has at least the following fields [1]

- `postID`: a unique integer identifying a post within the course
- `date`: the date when the post was initially created [2]
- `views`: an integer giving the total number of times this post has been viewed

OPERATIONS Each of these operations must run in worst-case $\mathcal{O}\log(n)$ time where n is the number of Piazza posts in the collection.

- `Insert(postID, date, views)` Add an item with these values to the collection. If an item with `postID` already exists in the collection, calling Insert updates the `views` but does not change the `date`.
- `Delete(postID)` Remove this item from the collection.
- `Search(postID)` Return the information in the collection about this post.
- `MaxViewedAfter(earliest_date)` Of all posts on or after `earliest_date`, return the `postID` of the one that has the most views. If multiple posts meet the date criteria and are tied for the most views, return any one of them.

(a) Describe the data structure you will use to implement this ADT. If your solution uses an augmentation of one or more standard data structures, be clear about any extra information that will be stored.

(b) Draw a diagram of your data structure with the following values already inserted (in any order you wish).

postID	date	views
202	1/15/21	30
183	1/29/21	547
464	2/14/21	49
466	2/16/21	49
95	2/17/21	20

Notice that the postIDs do not increase with increasing date in this example. It appears that Piazza may typically always have increasing postIDs as time passes, but you shouldn't assume that this will be the case in the future. Don't count on this for your solution.

(c) Describe the algorithm for Insert and justify that it runs in $\mathcal{O}(\log(n))$ time.

(d) Describe the algorithm for Delete and justify that it runs in $\mathcal{O}(\log(n))$ time.

(e) Describe the algorithm for Search and justify that it runs in $\mathcal{O}(\log(n))$ time.

(f) Provide pseudocode for MaxViewedAfter and justify that it runs in $\mathcal{O}(\log(n))$ time.

[1] There will of course be other data (like the message itself and the poster) stored in the real data structure about each post, but for the sake of this question, we will ignore all the other information.

[2] You should assume that you can compare dates with normal comparison operators. ($>$, $\leq$, etc.). Do not worry about the specific date format.

Question 2

You are working on a project modelling COVID-19 positive test counts. You have an array A of two-element tuples where the first element is a date and the second element is the number of positive tests on that date. The array is sorted in date order with earlier dates first.

You have been asked to compute a new array as follows. Each output element i corresponds to input element i. It holds the number of days from input day i until the future date in the input array that has the closest test count to the count on day i without being over the count on day i. If there is no future date with a test count lower than a given day, that element in the output array should be set to 0.

Here is an example input array:

```
[(2021/01/01, 3363), (2021/01/02, 2964), (2021/01/07, 4249),
(2021/01/11, 2903), (2021/01/15, 3056), (2021/01/16, 3422),
(2021/02/01, 1172), (2021/02/04, 1670), (2021/02/09, 1072)]
```

and the corresponding expected output:

```
[14, 9, 9, 24, 20, 19, 8, 5, 0]
```

In your algorithm you may use the statement `A[i].date - A[j].date` to determine the number of days between tuples `A[i]` and `A[j]`. This is an $\mathcal{O}(1)$ call.

(a) The obvious naive algorithm takes $\mathcal{O}(n^2)$ time. Describe this algorithm.

(b) Design an algorithm and supporting data structure to implement this task in asymptotically less time. Describe the algorithm including any data structures it uses.

(c) State and justify the worst-case complexity of your algorithm.

Question 3

Envision an AI algorithm which tracks student engagement minute by minute during Zoom lectures. This is hypothetical of course, though it is not too far fetched to have such feature in the near future. Engagement is measured by a complicated function that takes into account the activity, the relevance of chat messages to course content, the instructor's voice tone, poll response rate, number of attendees, and the facial expressions of those with video on. The function finds the engagement score over a time period (not at a single point in time) and returns a value between 0 and 100 for each time period.

Each time period is represented as an integer t rather than an interval or tuple. For example, t_1 refers to the first, i.e. earliest, time period measured in that lecture (which could be minute 1, the first 120 seconds, from 00:00:00 to 00:02:59, or the first half of the lecture). Analogously, t_2 would be the second time period measured in that lecture: minute 2, the second 120 seconds, from 00:03:00 to 00:04:59, or the second half of the lecture. Since a lecture is made up of a bunch of time periods, we add the subscript i to indicate an ordering relationship between periods. A relationship between t_i and t_j where $i < j$ indicates that time period t_i is temporally earlier than t_j and there might be other periods between them. You do not need to worry about the specific units of the time periods, just that the periods are consecutive, are represented as integers, do not overlap, and are measured in a consistent way within a single instance of the ADT.

We wish to design the implementation of a data-structure that supports the following operations:

- `Engagement(`L`,`t`)`: Given a single time period t in L, return the engagement score for that time period. L is the ADT storing the time periods (and any other necessary data) for a single lecture.
- `AverageEngagement(`L`,`t_i`,`t_j`)`: Given two time periods t_i and t_j from L where $i \leq j$, return the *average* engagement score (per time period) for that interval. This average should include

both endpoint time periods in the interval if $i \neq j$. For example, if $t1$ has engagement score 50, $t2$ has engagement score 67, $t3$ has 80, and $t4$ has 40, then the average engagement over the interval $t2$ to $t4$ is 62.33.

- `Update(L,t,e)`: Update the engagement score for time period t to e. You can assume that t is already in L.

Your implementation will be based on an augmented AVL tree and you should make the following assumptions:

1. n is the number of time periods in a zoom lecture. A lecture is broken up into a collection of time periods $t_1, t_2, ..., t_n$, all stored in L. Your algorithm must work for any size n. Recall that what t means as a time period will always be consistent within a single ADT.
2. A tree L already exists for each lecture. ($L.root$ is the root of the AVL tree.)
3. A node already exists for **every** time period t_i in L where $1 \leq i \leq n$. Each node has the associated engagement score so you do not need to implement or use Insert.
4. The nodes in L have the in-order traversal $t_1, t_2, ..., t_n$.

Here is what you need to do.

(a) Describe any additional information that you will need to store in your AVL tree to implement the ADT operations in worst-case running time of $\mathcal{O}\log(n)$. To get full marks, your data structure must not waste space by storing unncessary data.

(b) - (d) For each of the three operations (Engagement, AverageEngagement, and Update), explain the algorithm used to implement the operation and justify that the operation runs in the required worst-case time. For `AverageEngagement`, provide pseudocode as part of your explanation.

(e) Assume that we drop assumption 3. That means L could have some of the t_i for $1 \leq i \leq n$ but not necessarily all. For example, if L only has the following time periods: t_2 with engagement score 67, t_4 with engagement score 40, t_5 with engagement score 50, and t_7 with engagement score 80, then the average engagement for the interval t_2 to t_7 is 59.25. The average here is the sum of period scores that we have in the interval / the number of periods (again, that we have in the interval). There is no need for interpolation or estimating the missing data. Describe briefly what you would have to do and what extra information you have to store so that `AverageEngagement` will calculate the average. Describe any changes you would make to the algorithm for `AverageEngagement`. You do not need to completely rewrite the pseudocode.

Question 4

Suppose we have an array of the gold medal winners in an Olympic sport over many years. The elements in the array start with the most recent Olympic games and are sorted going back in time. For men's 100m freestyle swimming we would have an array like this:

```
[(2016, "Kyle Chambers", "AUS"), (2012, "Nathan Adrian", "USA"), (2008, "Alain
Bernard", "FRA"), ..., (1904, "Zoltan Halmay", "HUN"), (1896, "Alfred Hajos",
"HUN")]
```

We are interested in finding the country with the largest number of years between their first gold medal and their most recent one (in this sport). If there is a tie for the largest number of years, then returning any one of the tied countries is fine. One small complication is that sometimes (in some sports) there was a tie for gold.(See 2018 bobsled results for an example.) In that case, the input array contains both gold-medal winners listed in either order as individual array elements.

(a) Give a naive $\mathcal{O}(n^2)$ algorithm for solving this problem.

(b) Devise an algorithm that uses a data-structure we have learned about in class and has asymptotically smaller expected running time. Explain that algorithm.

(c) State and justify the average-case running time of your algorithm.

Question 5

Quadratic probing is one strategy we saw in class for handling collisions in hash tables. To find the next empty bucket in table T, we used $h(k, i) = [h'(k) + c_1 i + c_2 i^2] \bmod m$ where $h'(k) = k \bmod m$ and $i = 0, 1, 2, ..., m - 1$. To simplify things, suppose that we drop the linear term from the probe sequence by setting c_1 to 0. Our function for finding open buckets becomes $[h'(k) + c_2 i^2] \bmod m$ for $i = 0, 1, 2, ..., m - 1$.

Show that this simplification is problematic when m is odd because the probe sequence will check at most $(m + 1)/2$ buckets as i ranges from 0 to $m - 1$.