UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

# Algoritmos e Estruturas de Dados

2022-2023

Projeto 4 – Tabelas de Símbolos – versão (1.1)

# Twualger


Imagem retirada de @MidnightMitch/Twitter

## Introdução

This project will be slightly different from the previous ones and will focus on using efficient data structures to solve specific problems of a computer system. In this project particular focus will be given to the use of Symbol Tables (Hash Tables and Search Trees).

The idea for this project came about with the recent acquisition of Twitter by Elon Musk, and the controversy generated regarding Twitter's efficiency. In this project we will focus on one of the features of Twitter, the construction of a timeline – where the tweets of the accounts we follow are presented in descending chronological order.

The system to be developed – Twualger – will read tweets from files. However, as there are accounts with a high number of followers, these accounts will very often appear on timelines. Therefore, a good way to make the system faster is to use a cache, keeping the tweets of celebrities with the most followers in RAM memory.

To implement the functions of the project, we recommend that you do it in the order presented here in the statement, as functions specified later may depend on functions defined earlier. Some source code that you should use for your project has been provided, including the skeleton of the main classes to implement for problems A, B and C. It is recommended that you read the source code carefully.

The tests in Mooshak are designed in such a way that you get points as you correctly implement each of the requested methods. It is not necessary to have a fully implemented problem to obtain the corresponding quotation in the project.

## Problema A: TwualgerA

In this problem we will start by implementing a less efficient version of the system, but which is already able to read tweets from files, store them in a cache, and build timelines.

Start by preparing the project (in IntelliJ or Eclipse), downloading the provided source code, as well as the files with tweets that we will use. Files with tweets must be downloaded and placed in a data folder within your main project directory. This problem should be implemented using the provided TwualgerA class.

### 1) readTweetsFromFile

This method is responsible for reading user tweets from a file. It takes as arguments a path, which indicates the directory where the .csv files with the tweets are located, and a username.

They should build the final path using the received path and the file name. Each user has a file with their tweets, and the name of the corresponding file is given by the username. For example, for Elon Musk, whose username is elonmusk, the corresponding file name is elonmusk.csv.

To read the file a BufferedReader is recommended instead of a Scanner due to Scanner inefficiency problems.

A csv file with tweets has the following organization: The 1st line has a header with 3 fields (twitter_id,date,tweet) separated by a comma. This line can be ignored. Each of the remaining lines corresponds to a tweet, organized in the same way as the header. We can see an example in the following figure.

```
1 twitter_id,date,tweet
2 1546602051022495746,2022-07-11 21:07:18+00:00,b'@BillyM2k Mimic + Dark Moon + Stars of Ruin'
3 1546598042844991489,2022-07-11 20:51:22+00:00,b'https://t.co/mvVUR4doqB'
4 1546561966331789318,2022-07-11 18:28:01+00:00,b'@greg16676935420 @MrBeast \xf0\x9f\x94\xa5\x
5 1546552413083111424,2022-07-11 17:50:03+00:00,b'@WholeMarsBlog Absolutely'
```

*Figura 1 Extrato de ficheiro csv com os tweets do utilizador elonmusk*

This method should read all tweets, and return a UserCache of tweets (here you should use the Tweet and UserCacheA classes provided).

As a date we will use the OffsetDateTime Java type, which represents a date and time with a time zone. To extract the date from a String, we must use the static method OffsetDateTime.parse, and pass the

String along with a DateTimeFormatter that can be constructed through the expression. `DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ssZZZZZ")`.

### 2) Constructor TwualgerA

Once the previous method is implemented, we can implement the constructor. The idea is that for this class we will use a cache to store each user's tweets. In this simpler version we will use an ArrayList to store each user's caches.

There is a well-known list of accounts which have the most followers. This list can be obtained using the readTopCelebs method of the Twualger abstract class. In the constructor, in addition to initializing the class's fields, you should immediately load tweets from all accounts that belong to the top into the cache, as the probability of one of these being used in any search is very high.

### 3) getUserCache

This method receives a username, and will search the cache currently in RAM memory for that user's tweets. If the tweets do not exist in the cache, it will read the tweets from the file using the method implemented in exercise 1, and will place them in its cache. Finally, the UserCache is returned with all the user's tweets.

This method will also count the number of hits and misses in the cache. A hit occurs when searching for a user's tweets, the cache is already in memory, and a miss occurs when the cache does not yet exist in memory and must be read from the file. Additionally, whenever a UserCache is returned (regardless of whether it was read from disk, or taken from RAM memory) the number of times this cache was used (or searched) is incremented.

### 4) totalSearches e cacheHitRatio

Here we intend to implement two simple methods useful to understand how the cache works. totalSearches returns the total number of searches that were performed on the system, accounting for the total for all users. For example, to build a timeline with 10 accounts that are being followed, the number of lookups needed is 10 (regardless of whether the accounts are already in cache or on disk).

The second method, cacheHitRatio, calculates and returns a float between 0 and 1 that represents the ratio between the total hits (searches for elements already in the cache) and the total number of searches carried out.

### 5) buildTimeLine

This is the method responsible for building a timeline that will be later used to render the homepage of a given user. It takes as an argument a List of account names that are being followed, a start date (from) and an end date (to). This method will fetch tweets from each account being followed that fall between the start date and end date. At the end, a list of all grouped tweets is returned, sorted in descending chronological order, that is, from the most recent tweet to the oldest tweet.

To sort the returned list you can, for example, use the sort method of the ArrayList class, as this method uses a sorting algorithm very similar to the sorting method implemented in project 3, and which will work quite efficiently for arrays that are already partially sorted

### 6) downsizeCache

This method aims to decrease the size of the cache when it is too large. When possible, the intention is to eliminate 50% of the least used caches. However, there are users that we always want to keep in cache, they are the top users that were loaded in the constructor. For this reason, it may happen that it is not possible to delete 50% of the caches. So we eliminate the less used caches (avoiding deleting those belonging to top accounts), and when we have eliminated 50% of the caches (or when there are no more caches available) we stop.

Additionally, this method will also reset all counters related to accounting for cache operation, such as cacheHit, cacheMiss, and counters for each user's cache (which is not deleted).

Don't worry too much about the efficiency of this method, as it won't be executed as often as the others.

**Requisitos técnicos:** The requested methods must be implemented in the provided TwualgerA class (Twalger.java file). Other auxiliary classes or methods must be defined in the same file.

Submit only **file TwualgerA.java** in Problema A.

## Problema B: Twualger B

Imagine that the CTO of the fictitious company (Twualger) that hired us is as demanding or even worse than Elon Musk – let's call him Melon Tusk. During one of the daily meetings , Melon Tusk asked to see our code (Problem A), and called us several nasty names when he saw that we were saving the caches in an ArrayList. And he warned us that if the code wasn't better and more efficient by the end of the Sprint, we'd be fired ☹. The Sprint ends on the 2nd of November.

Hopefully we'll be able to use some of the things we learned in AED to not get fired.

The first thing we can do is use a HashTable to store each user's caches (indexed by username or handle which is unique). That way, each time we need to check (or find) the cache for a given user, we can do it in constant time (instead of having to go through the entire list).

The second thing we can do is make selecting tweets from the intended dates more efficient (instead of comparing one by one). One way to achieve this is to use an Ordered Symbol Table. Here we can use the natural ordering of dates (Java's OffsetDateTime class is Comparable), using the date as the key to store and organize each user's tweets. Therefore, our Ordered Symbol Table will contain all Tweets in order from smallest (oldest) to largest (newest).

Once we use a Table of Symbols Ordered by date, we have a series of efficient functionalities (O(log2 n)), such as obtaining the maximum (most recent tweet), the minimum (oldest tweet), all values after (or before) a date, or even all values within a date range.

To use an Ordered Symbol Table in Java we could use Java's TreeMap class, which implements an Ordered Symbol Table using red-black trees (just like we saw in theoretical classes), but instead (to try to impress Melon Tusk ) let's do something cooler. We'll use another implementation: Treap . A Treap is a data structure that combines characteristics of a binary search tree with a heap, by using random numbers as priorities that will be used to reorganize the Treap, so that the final tree obtained is similar to the tree that would be obtained by placing the elements in random order. Treap was one of last year's projects,

implemented by your colleagues. You won't have to implement it (the code is provided), but you'll have to know how to use it.

Implement again the methods implemented in problem A, but this time considering the mentioned optimizations.

**Requisitos técnicos:** The requested methods must be implemented in the TwualgerB class. You should also use the UserCacheB class, and the Treap class for organizing the cache. If you need other auxiliary classes or methods, these must be defined in the same file.

Submit only file ***TwualgerB*.java** in Problema B.

## Problema C: Twualger C

The changes made to Problem B already made Melon Tusk a little happier, but he thinks the system is still too slow. After analyzing the most common search requests, we came to the conclusion that 80% to 90% of the times the searches for building a timeline correspond to the last 72 hours

That being the case, it doesn't make much sense to be loading all existing tweets into the cache stored in RAM memory. Let's do a new optimization based on this idea. We will need and make the following changes.

### 1) readUserTweetsFromFile

The tweet reading method will now receive one more argument, which is the earliest date for which we want to read the tweets. As we read the various tweets, we will check if the tweet date is before the oldest date. If it is, it means that we no longer want to store that tweet in the cache, and we can stop reading. However, before returning it is important to save a flag in the cache saying that there are more tweets to be read in the file.

### 2)updateUserCacheFromFile

We will also need an additional read method, used to update a cache to read older tweets up to an older date. When we are updating a cache, we are not interested in putting tweets that are already there in the treap, and therefore we only add tweets for dates that are between the previous oldest date from the Treap, and the new oldest date received as an argument . For example, if the treap has the oldest date 72 hours before (-72H), and we need to read it up to a week ago (-7D), when we are reading from the file we can ignore all the most recent dates, and just put in the treap the dates between -7D and -72H. When we reach a date that is older than the new oldest date, that tweet should no longer be registered and we can stop reading.

Additionally, we also have to re-determine the correct value for the flag that indicates if there are more tweets to be read.

### 3)Construtor TwualgerC e getUserCache

Now when we try to get an account's tweets for the first time, and that account is not cached, we should only read tweets up to the previous 72 hours (from the most recent date for which we have data). This also applies to reading the top accounts in the builder.

### 4) buildTimeLine

When we build a timeline, we have to check if we want to include tweets from a date older than the oldest tweet saved in the treap. If this happens, and if the cache is marked as having more tweets to be read, then we will have to update the cache so that it will fetch all tweets until the new intended date. But if the cache is marked as having no more tweets to read, there's no need to do anything, because even if we tried we wouldn't be able to read anything else.

Once the cache is updated, we can proceed as we did in problem B.

### 5) downsizeCache

The downsize method, in addition to removing the least searched accounts from the cache, will also remove tweets older than the previous 72 hours for the accounts that remain in the cache.

### 6) Comparação entre implementações efetuadas

Once the optimizations are implemented, you need to be able to convince Melon Tusk that your code is much more efficient. Compare the average execution time of many searches (i.e. building timelines) for the 3 implemented variants, using the following conditions: 80% of searches are made to top accounts (remaining 20% to other accounts), 80% of searches start at most recent date (remaining 20% can start up to 10 months ago) and 80% of searches have a span of 72 hours (remaining 20% can span up to 90 days).

Write a brief summary of the tests and results, and a brief analysis of the results as comments in the Java file to be submitted. Indicate the time efficiency gain obtained when comparing the final solution with the initial one.

The main method must invoke the tests used. Although this method is not automatically validated by Mooshak, it will be taken into account in the validation of the project, and will count towards the final grade.

## Condições de realização

The project must be carried out individually. Equal or very similar projects will lead to failure in the course. The faculty of the discipline will be the sole judge of what is or is not considered copying in a project.

The project code must be submitted electronically, through the Mooshak system, by 23:59 on the 2nd of December. Students will have to validate the code together with the professor during the laboratory hours corresponding to the shift in which they are enrolled. The evaluation and corresponding project grade will only take effect after the validation of the code by the teacher.