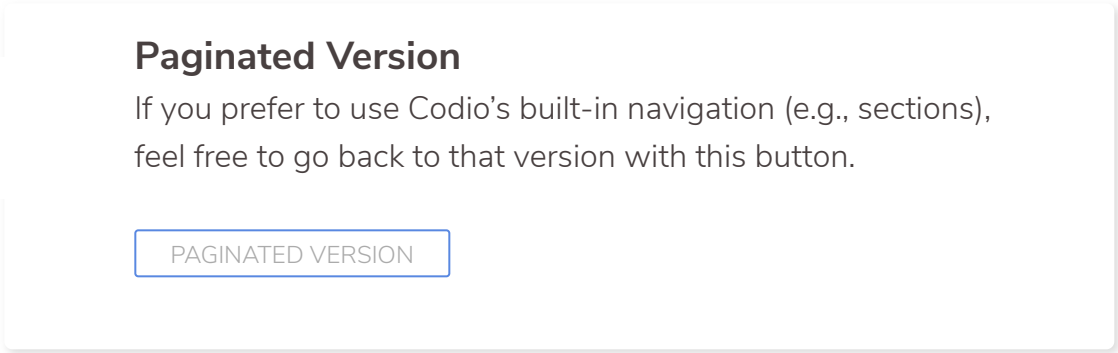
8. All in One Page



▾ Overview

This program focuses on maps implemented with arrays, binary trees, and hash tables.

Introduction

In this project, you will be investigating tweets related to the recent U.S. federal election (tweets about Donald Trump and Joe Biden).

The dataset being used was taken from Kaggle. The data is organized into two files: hashtag_donaldtrump.csv and hashtag_joebiden.csv. Your task will be to determine the top 20 twitter users in terms of how many tweets they posted about either/both candidate.

To count tweets, the most common approach is to read each result one at a time, look up what Twitter users posted them in a dictionary/map, and if the user is already in the map, add one to its tweet count. If the user is not in the map, the user is added to the map with a count of 1.

In this project, you will investigate the performance of 3 map designs:

1. A map implemented with an unsorted list. Your unsorted list should be of type `ArrayList<TweetCount>`. A `TweetCount` class is already provided. You may make `TweetCount` different or more complex as needed; what is provided is a minimum implementation. You should not maintain a

8. *All in One Page*

   `ArrayList` class, see the Java API documentation for ArrayList.

2. A map implemented with a binary search tree. Your map should be of type: `TreeMap<String, TweetCount>`. See the Java API documentation for more information.
   Aside: this is actually a Red-Black Tree, which is a type of binary search tree.

3. A map implemented with a hash table. Your map should be of type `HashMap<String, TweetCount>`. See the Java API documentation for more information.

Please note that the manual pages for the `ArrayList`, `TreeMap`, and `HashMap` classes provide a lot of information and even examples! One of the goals of this project is for you to gain experience with learning to use built-in classes from a class library and the work required to learn how to use these classes by reading available documentation.

We have provided the dataset and some code that will read and convert each file into an object that lets you iterate through data once. You will iterate through the data and use a map implementation to count the tweets. You will then output to the console the 20 users with the highest number of tweets across both candidates.

▾ Iterable & CSVRecord

Iterable

We have provided code that provides a variable, `allTweets`, which is an object of type `Iterable<CSVRecord>`. While you may think that you are unfamiliar with `Iterable`, this is something you've actually been using for many different data structures. If you look at the documentation for Iterable, you can see that it mentions `ArrayList`, `LinkedList`, `HashSet`, `TreeSet`, `LinkedHashSet`, and *many* more objects in the "All Known Implementing Classes" section. This means, to use an `Iterable` object, you can work with them in a familiar way using a **for each loop**.

8. *All in One Page*

or `String`s.

```
Iterable<String> iter = //...
for (String s : iter) {
    System.out.println(s);
}
```

On the right-hand side of the assignment (where the `// ...` is) could be any of the above-mentioned data structures (e.g., an `ArrayList`).

Some important differences about `Iterable` and `List` or `Set` interfaces you are familiar with:

- you can only iterate through the elements **one time**. Once the for each loop is done, the list is *exhausted*.
- you **cannot modify** the list in any way (there's no `set` method available).
- there's no (easy) way to do a more traditional `for` loop, so if you want an "index", you'd need to keep track of that separately.
- you only have access to the "next" element (you can't get the i^{th} element).

CSVRecord

To read in the dataset for this assignment, we are reading them from comma-separated values (CSV) files. This is a standard format that can also be read by programs like Microsoft Excel (feel free to download and open the hashtag_donaldtrump.csv and hashtag_joebiden.csv files to see).

In this project, we will be using a **library** called Apache Commons CSV to read in the records of these two CSV files. This library has several useful classes for reading in and writing to CSV files, but the only class you'll need to be familiar with is `CSVRecord` (documentation for CSVRecord). You may find it useful to review the entire documentation, but the most relevant methods from that class for this assignment are:

8. All in One Page

- `isSet(int index)` or `isSet(String name)`
  Alternatively, you can do this entire assignment with only the `toMap()` method from the `CSVRecord` class.

Summary

To iterate through the data in this assignment, you'll need a loop that looks something like this:

```
for (CSVRecord record : allTweets) {
    // use CSVRecord methods on the record object
}
```

▸ Finding the Top 20

▾ Comparing Objects

To sort all the `TweetCount` objects, we first need to copy them out of the map and into a Java Collection. We could declare such a list as follows:

```
ArrayList<TweetCount> list = new ArrayList<>();
```

where `TweetCount` is the class above that pairs users (`String`) with `TweetCount` objects. Once we have all the `TweetCount`s in a list, we need some way of sorting the list by tweet count.

If a Java Collection is a `List` of some simple type such as an `int` or `double`, we can call the `sort` method from the `Collections` class. For example:

```
ArrayList<Integer> myNumbers = new ArrayList<>();
myNumbers.add(6);
myNumbers.add(1);
// etc.
Collections.sort(myNumbers);
```

8. *All in One Page*

Unfortunately, if you try this with `TweetCount` objects, it will not work right away. This code:

```
ArrayList<TweetCount> vc = new ArrayList<>();
TweetCount uwaterloo = new TweetCount("uwaterloo", 130);
TweetCount uwhci = new TweetCount("uwhci", 231);
TweetCount engineering = new TweetCount("WaterlooENG", 4);
// ... code to set tweet counts/totals
vc.add(uwaterloo);
vc.add(uwhci);
vc.add(engineering);
Collections.sort(vc);
```

will not compile with the explanation `The method sort(List<T>) in the type Collections is not applicable for the arguments (ArrayList<TweetCount>)`.

The problem is that to sort items, we need some way to compare the items. Java knows how to compare `int`s and `double`s (every time you do `<` or `>` or `<=` or `>=` or `==` or `!=`, you're doing comparisons). If we want to sort `TweetCount` objects, we need to choose between one of two options:

1. Have `TweetCount` implement the `Comparable<TweetCount>` interface (Java API documentation for Comparable).
2. Provide a `Comparator<TweetCount>` object as a parameter to the `Collections.sort` method, that knows how to compare two `TweetCount` objects (Java API documentation for Comparator).

In both cases, the solution amounts to overloading a method called `compareTo` or `compare` that can compare two `TweetCount` objects and report whether one is less than, equal to, or greater than the other. We have not talked about overloaded methods in Java. An overloaded method is actually multiple methods that share the same name. We can tell the difference between the methods because the object it belongs to and the arguments that are passed to the methods are different. Other examples of methods that you may have overridden are the `toString` method and the `equals` method. The

8. *All in One Page*

compare two objects.

The `compareTo` function used to compare two objects, A and B, behaves according to the convention that if A is less than B, then our comparison function returns a value less than zero, if A is equal to B it returns 0, and if A is greater than B it returns a value greater than zero. Full details can be gleaned from either the Comparable or Comparator Java API documentation provided above.

We want to compare the counts of two `TweetCount` objects. We could do this by first changing the class definition for `TweetCount` as follows:

```
public class TweetCount implements Comparable<TweetCount> {
    // ...
}
```

And then writing the following method inside the TweetCount class:

```
@Override
public int compareTo(TweetCount o) {
    return Integer.compare(this.getCount(), o.getCount());
}
```

To make our life easy, since `int`s already have a built in compare method, we compare the tweet count of the current `TweetCount` object to the other `TweetCount` object passed in (`o`) as shown above. This makes the above code that wouldn't compile work and sorts the array by tweet count.

Is sorting and finding the top 20 users better than a naive priority queue implementation? I leave that to you to decide and discuss in this project.

▾ Step-by-Step

1. Follow the Assignment 0 instructions to push your code to GitHub (yes, it's okay that you haven't done anything yet!).

8. All in One Page

using Eclipse for Assignment 3, as the code can get quite slow on Codio for such large datasets.

3. Compile & Run the program. It should output:

```
Data available:
        0 = created_at
        1 = tweet_id
        2 = tweet
        3 = likes
        4 = retweet_count
        5 = source
        6 = user_id
        7 = user_name
        8 = user_screen_name
        9 = user_description
        10 = user_join_date
        11 = user_followers_count
        12 = user_location
        13 = lat
        14 = long
        15 = city
        16 = country
        17 = continent
        18 = state
        19 = state_code
        20 = collected_at
There are 1,748,161 tweets in total.
```

4. If (and only if) you're working on your computer's IDE (e.g., Eclipse), change the `READ_INTO_MEMORY` constant to `true` (change this back to `false` when you test on Codio). Try compiling and running again, and it should output (with different times):

```
Finished reading 971,088 Trump tweets in 8.040622 seconds.
Finished reading 777,073 Biden tweets in 7.483819 seconds.
... (same output as before follows here)
```

5. Record the header information (or just use the list in step 3) and change the `SHOW_HEADERS` constant to `false` to make the output easier to read.

8. All in One Page

following:

a. For the appropriate method (`findTopUsingArrayList` , `findTopUsingTreeMap` , or `findTopUsingHashMap`), add code that uses the map to count the tweets for each Twitter user.

b. To count the tweets, the best approach (for all map implementations) is to iterate through the `Iterable` object that is provided (`allTweets`) and add/update the `TweetCount` object in the map for each `CSVRecord` object in the `Iterable` . E.g.,

```
for (CSVRecord record : allTweets) {
    // create/update TweetCount object for this user
}
```

c. You should only create one list/map of `TweetCount` objects within each method.

d. Using the code and good practices learned in lecture/lab this term, time how long it takes you to determine tweet counts for each user using this map implementation. You should only measure the time in within the method, and not the time it takes to call `readData` .

e. Collect the following information for this map and output it to the console:

- how long it took to determine the tweet counts with the map
- the total number of tweets that were counted over all candidates
- The top 20 users sorted by tweet count in descending order. For example, output of one map's solution on a subset of the data is as follows (**your output should match this format exactly**, but with different times/numbers):

```
Finished reading 1,000 Trump tweets in 0.169417 seconds.
Finished reading 1,000 Biden tweets in 0.019566 seconds.
To count 2,000 tweets with a HashMap took 0.014263 seconds
The top 20 users by number of tweets are:
snarke had 35 tweets
CupofJoeintheD2 had 23 tweets
```

8. All in One Page

```
InfamousOne13 had 14 tweets
kleensamsonite had 11 tweets
InconvenientTr5 had 10 tweets
Vote4Jonty had 9 tweets
SpeaksAthena had 9 tweets
brunolp30 had 9 tweets
elpoliticonews had 9 tweets
Cheryl84037124 had 8 tweets
greeneyes3470 had 8 tweets
sunbeanz had 7 tweets
W_K_Martin_III had 7 tweets
pepp32948008 had 7 tweets
DavidDurandMD had 7 tweets
Kegan05 had 6 tweets
whipit_Studio had 6 tweets
william7424 had 6 tweets
```

7. Throughout the assignment, continue to push to GitHub **regularly** from Eclipse (or from Codio if you make changes there). This will save your work and let you pull it into Codio (or Eclipse) whenever you want (e.g., to ask a Campuswire question).
8. Write a report. See the next page for information about the report.

Compile & Run

As mentioned above, I **strongly** recommend that you use Eclipse (or another IDE) to complete this assignment by first pushing to GitHub and then cloning the repository locally. You'll be able to run your code more quickly and get better timing data for your experiments. You can always push back to GitHub and pull into Codio to see it here again.

If you want to do quick tests in Codio, that is still possible. You can compile your program as you've done before with this button:

COMPILE

8. All in One Page

inline.

Reminder: You always need to set the `READ_INTO_MEMORY` constant to `false` when you run in Codio, or it will run out of memory. You may also want to change the `MAX_TWEETS` constant to a smaller number.

You can open a terminal yourself, or use this convenient button to open the terminal, compile, and run your code all at once:

COMPILE & RUN

You can also step through your code in Codio still using the debugger:

DEBUG

You can also always use the menus at the top to compile, run, and debug your code.

▸ Report & Submission Instructions

Mark as Completed Back to dashboard