CMPE 343
Fall 2020
Programming Homework #1

This assignment is due by 23:55 on Friday, 30 October 2020.

Late submissions: 20 points will be deducted for each late day.

PART I (60 points)

Recently there has been quite a bit of research into analyzing Twitter data to determine sentiments, trends, etc. of a particular region or population. Much of this research involves a combination of statistics and computational linguistics. In this part, you will simplify things greatly and only look for the most popular hashtags, i.e. strings/tokens starting with the '#' character.

In this part of the assignment, you are asked to implement a program using a hash table to store the distinct hashtags found in a given input file (twitter.txt) and keep track of how many times each appears, then report the 10 most popular (in order) (as you surely know, hashtags are words/tokens that begin with the '#' character).

This assignment assumes you have a working implementation of a hash table of strings. As an example, consider a hash table of strings of size 20, and a hash function that adds up the ASCII values of the characters in the string. For the string "dog", the hash code would be 100 + 111 + 103 = 314. Since 314 % 20 = 14, the string would be added to the linked list that is in the 14th spot (0-based, of course) of the array.

First, begin implementing a hash table class in Java. You may want to start with the Hash class given in the book. Use the **separate chaining collision resolution strategy** while implementing the hash table.

Second, create a Java application that reads the name of the input file containing the tweets as a command-line argument (i.e., do not hardcode the name and do not prompt the user to enter it). If the filename is not specified, or if the file does not exist or cannot be opened, display an error message and terminate the program.

If you're able to open the file, read the file one word at a time. Keep track of all of the hashtags that you encounter and the number of occurrences of each. Remember, the hashtable should only contain distinct hashtags, so you will need to modify the node definition so that it has a field that records the number of times the hashtag has been seen, and only add a new node when you encounter a string that haven't been seen before.

After you've read the entire file, display the 10 most common hashtags (as well as the number of occurrences of each) in descending order. Note that it is not necessary to sort the entire list! You only need to find the 10 most common (think about the other data structures we used in CMPE 221 course).

You are free to create your own simple file for testing, but your program should work for "big" files, too. Download the set of around 300,000 tweets from the Moodle site, and be sure that your program can handle it. Also, there are several applications that help you to download your live tweets to a text file; you may want to experiment with them for fun, e.g. check https://www.quora.com/How-can-I-export-all-x-user-tweets-to-excel-or-csv).

For the given test file (twitter.txt), your program should produce this output:

```
#jobs: 4251
#job: 3174
#ff: 2057
#tweetmyjobs: 1558
#getalljobs: 748
#fb: 701
#wordstodescribeme: 691
#healthcare: 628
#teamfollowback: 548
#coupon: 486
```

PART II (40 points)

In the second part, you will calculate the statistics for your hash function. The hash function given in Part 1 is a very simple form of hash function, and there are more successful alternatives. Propose two different new hash functions, and calculate the average number of probes for each hash function, using the given test file as input.

You are welcome to study the alternative hash functions for strings, on the Web and the book, but you should give proper reference to your sources.

WHAT TO HAND IN

A zip file containing:

- The Java sources for your program.
- A **maximum-2 page** PDF report that explains:
    - the two new hash functions you have used, and that compares the average number of probes for each hash function,
    - a short discussion on why the selected hash function works better than the others.

IMPORTANT

IMPORTANT NOTES: Do not start your homework before reading these notes!!!

1. You should submit your homework to course Moodle page before the deadline. No hardcopy submission is needed. You should send files and any additional files if you wrote additional classes in your solution as a single archive file (e.g., .zip, .rar).

2. The standard rules about late homework submissions apply (20 points will be deducted for each late day). Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

3. You ARE NOT ALLOWED to modify the given method names. However, if necessary, you may define additional data members and member functions.

4. For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. You ARE NOT ALLOWED to use any codes from somewhere else (e.g., from the internet, other text books, other slides ...).

5. The submissions that do not obey these rules will not be graded.

6. To increase the efficiency of the grading process as well as the readability of your code, you have to follow the following instructions about the format and general layout of your program.

7. Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your Java files. Example:

```
//-------------------------------------------------------
// Title: BST
// Author: Ibrahim ILERI
// ID: 2100000000
// Section: 0
// Assignment: 1
// Description: This class defines a BST
//-------------------------------------------------------
```

8. Since your codes 1will be checked without your observation, you should report everything about your implementation. Well comment your classes, functions, declarations etc. Make sure that you explain each function in the beginning of your function structure. Example:

```
void setVariable(char varName, int varValue)
//-------------------------------------------------------
// Summary: Assigns a value to the variable whose
// name is given.
// Precondition: varName is a char and varValue is an
// integer
// Postcondition: The value of the variable is set.
//-------------------------------------------------------
{
      // body of the function
}
```

   - Indentation, indentation, indentation...

9. This homework will be graded by your TAs, İbrahim İleri and Hamid Ahmadlouei (ibrahim.ileri@tedu.edu.tr, hamid.ahmadlouei@tedu.edu.tr). Thus, you may ask them

your homework related questions through HW forum on Moodle course page. You are also welcome to ask your course instructor Tolga Çapın and Ulaş Güleç for help.

GRADING RUBRICs

PART I						
Performance Element	**Weight**	**Master (4/4)**	**Advanced (3/4)**	**Developing (2/4)**	**Beginner (1/4)**	**Insufficient (0/4)**
Information	5%	Information (id, name, section, assignment number) given in each file.	Missing minor details.	Missing few details (e.g. section id).	Only name given.	None.
Documentation	5%	Every class and method has a detailed explanation (pre- and post-conditions, sample I/O, etc).	Most classes and methods have an explanation , but missing in some parts.	Attempted to document the classes and methods, but they are not clear.	Only few comments.	Not even an attempt.
Code Design	5%	Modular source code and code format. Complete submission of classes and methods.	Methods make sense. Includes constructor that initializes carefully.	Uses set/get methods as necessary.	Class does very little; most functions remain in one main (driver) class.	Methods not properly defined.
Abstract Data Type: Hash Table	10%	Uses hash table based data structure for implementation. And provides all functionality.	Uses hash table based data structure for implementation. And provides most of expected functionalities.	Implements hash table with major deviations from the specification.	Used different data structure from hash tables to solve this problem.	Not even an attempt.
Functionality	65%	All functions are implemented with no missing functionality. Runs without any crash.	Missing some minor features or minor output problems. Runs without any crash.	Attempted to implement all functions but some of them do not work.	Only few functions are implemented correctly. Compiles but several warnings.	No working solution or does not compile.
Testing: Test data creation & generation	10%	Provided a tester class. Able to identify key test points.	Provided a tester class. Able to identify key test points.	Provided a tester class. Able to identify key test points.	Sporadic.	No test case presented.

		Clear on what aspects of the solution are being tested with each set. Able to generate test data automaticall y when necessary.	Clear on what aspects of the solution are being tested with each set.			

PART II					
Performance Element	**Weight**	**Master (3/3)**	**Average (2/3)**	**Beginner (1/3)**	**Insufficient (0/3)**
Documentation (PDF Report)	100%	Every methodology step has a clear and compact explanation within page limits. Analyses are complete. Supported with tables and graphics.	Most steps have an explanation, but missing in some parts like complete visualization elements or exceed page limits too much.	Attempted to document but they are not clear. No visualization elements. Steps don't have explanations and conclusion.	Not even an attempt.