Programming Assignment 1
Points: 500
Due: Mar 8, 11:59PM
Late Submission Due: Mar 9, 11:59PM (25% penalty)

Description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. You are encouraged to consult instructor/Teaching Assistants for any questions/clarifications regarding the assignment. Your programs must be in Java, preferably Java 8.1.

For this PA, you **may work in teams of 1, 2 or 3**. It is your responsibility to find team member(s). If you can not find team member(s), then you must work on your own. **Only one** submission per team please.

For this PA, you are **not** allowed to use any of Java's in-built hash-based data structures as `hashSet`, `hashMap` etc. In addition, you are not allowed to use any external libraries.

In this programming assignment you will

- Implement a hash table that can store $\langle key, value \rangle$ pairs.
- Use hashing and hash tables to estimate document similarity.

You will design following classes:

- Tuple
- HashTable
- BruteForceSimilarity
- HashStringSimilarity
- HashCodeSimilarity

All your classes must be in the **default package** (even though it is not a good programming practice). Ie., do not specify package name.

1 Tuple

Design a class name `Tuple` to represent tuples of form $\langle key, value \rangle$, where *key* is of type `int` and `value` is of type `String`. This class will have following constructor and public methods

`Tuple(int keyP, String valueP)` Creates `Tuple` object with *keyP* as *key* and *valueP* as *value*.

`getKey()` Returns *key*

`getValue()` Returns *value*

`equals(Tuple t)` returns *true* if *this* tuple equals t; otherwise returns false. I.e returns true if `this.key` equals`t.key` and `this.value` equals `t.value`.

2 HashFunction

This class will represent a *random hash function* that maps integers to non-negative integers, that can be used in constructing a hash table. This class is provided to you. Please do not change this class. This class has following public constructors and methods.
`HashFunction(int range)`. Picks the first (positive) prime integer p whose value is at least `range`. Then picks two random integers x and y from $\{0, 1, \cdots, p-1\}$ and uses $(ax+b)\%p$ as hash function.

`hash(int x)` Returns the value of the hash function on x; i.e, returns $(ax + b)\%p$. This method assumes that x is non-negative.

3 HashTable

This class will implement a hash table. The hash table will hold a multi-set of data items whose type is `tuple`. Recall that in a multi-set, an element can appear multiple times. This class will have following public methods and constructor.

`HashTable(int size)` Finds the smallest prime integer p whose value is at least `size`. Creates a hash table of size p where each cell initially is NULL. It will determine the hash function to be used in the hash table by creating the object `new HashFunction(p)`.

`maxLoad()` Returns the maximum load of the hash table. Return type is `int`.

`averageLoad()` Returns the average load of the hash table. Return type is `float`.

`size()` returns the current size of the hash table. Return type is `int`.

`numElements()` returns the number of *distinct* `Tuple`s that are currently stored in the hash table. Return type is `int`.

`loadFactor()` return the load factor which is numElements()/size(). Return type is `float`.

`add(Tuple t)` Adds the tuple t to the hash table; places t in the list pointed by the cell $hash(t.getKey())$ where *hash* is the hash function method from the class `HashFunction`. When the load factors becomes bigger than 0.7, then it (approximately) doubles the size of the hash table and rehashes all

the elements (tuples) to the new hash table. The size of the new hash table must be: Smallest prime integer whose value is at least twice the current size. Return type is `void`.

`search(int k)` returns an array list of Tuples (in the hash table) whose key equals k. If no such Tuples exist, returns an empty list. Note that the type of this method must be `ArrayList<Tuple>`.

`search(Tuple t)` returns the number of times Tuple `t` appears in the hash table. Return type is `int`.

`remove(Tuple t)` Removes one occurrence Tuple `t` from the hash table. Return type is `void`.

4 Set Similarity

Given two sets how similar are they? This is one of the problems that is of great interest in the current age of big data. In this homework you will learn about a method to determine the similarity of sets.

We will define a notion of *similarity* between *multi-sets*. Recall that in a multi-set, an item can appear multiple times. For example, $\{2, 4, 6, 1, 2\}$ is a multi-set. Given a multi-set S, we define a notion called *VectorLength* of S. Let $f(S, i)$ denote the number of times i appears in the multi-set S. Now

$$VectorLength(S) = \sqrt{\sum_{i \in S} [f(S, i)]^2}$$

Let S_1 and S_2 be two multi sets and let U be the union of S_1 and S_2 after removing the duplicates. I.e, in U every element appears only once. Now the similarity between S_1 and S_2 is defined as

$$Similarity(S1, S2) = \frac{\sum_{i \in U} [f(S_1, i) \times f(S_2, i)]}{VectorLength(S_1) \times VectorLength(S_2)}$$

Here is an example: Let $S_1 = \{1, 2, 6, 8, 2, 6, 4, 6, 1, 2\}$ and $S_2 = \{2, 5, 1, 1, 8, 8, 4, 3, 8\}$. Now $U = \{1, 2, 6, 8, 5, 3, 4\}$; $VectorLength(S_1) = \sqrt{24}$ and $VectorLength(S_2)$ is $\sqrt{17}$. The similarity between S_1 and S_2 is

$$\frac{11}{\sqrt{24}\sqrt{17}} = 0.544$$

5 Document Similarity

Given two documents how similar are they? For example, Chrome's `similar pages` plug-in finds webpages that are similar to the page that you are currently browsing. The notion of document/set similarity has quite a few applications in text/image processing, recommendation systems etc. We will use set similarity to define two notions of *document similarity*. This is done by converting each document into a `String` and then associating a multi-set with each string.

Let D_1 and D_2 be two documents whose similarity we wish to estimate. We convert both D_1 and D_2 into strings by eliminating white space/tab characters, and the punctuation symbols period, comma, colon and semi-colon. Converting each character into lower case. Now each document can be viewed as a (perhaps very long) string. We now define a notion of k-shingles of a string. A

k-shingle of a string is a substring of length k. Let S_1^k be a multi-set of all k-shingles of D_1 and S_2^k be multi-set of all k-shingles of D_2. Now

$$Similarity_k(D1, D_2) = Similarity(S_1^k, S_2^k)$$

Note that the above value depends on the value of k.

Another way to define similarity is by considering hashCodes of elements of S_1^k and S_2^k. Let HS_1^k be the multi-set of all `hashCode`s of strings from the multi-set S_1^k, and let HS_2^k be the set of all `hashCode`s of string from the multi-set S_2^k.

$$HashSimilarity_k(D1, D2) = Similarity(HS_1^k, HS_2^K)$$

Here is an example. Suppose you have two documents, Say that the contents of D_1 are

A rose is a rose is a rose.

The contents of D_2 are

A rose is a flower, which is a rose.

The the String corresponding to the first document is (by ignoring case, removing white space, period and comma)

aroseisaroseisarose

The string corresponding to the second document is

aroseisaflowerwhichisarose

Let us take $k = 4$. Now S_1^4 is

aros, rose, osei, seis, eisa, isar, saro, aros, rose, osei, seis, eisa, isar, saro, aros, rose

Below is the multi-set S_2^4.

aros, rose, osei, seis, eisa, isaf, safl, aflo, flow, lowe, ower, werw, erwh, rwhi, whic, hich, ichi, chis, hisa, isar, saro, aros, rose

Now, U is

aros, rose, osei, seis, eisa, saro, isar, isaf, safl, aflo, flow, lowe, ower, werw, erwh, rwhi, whic, hich, ichi, chis, hisa,

We calculate

$$Similarity_4(D_1, D_2) = \frac{22}{\sqrt{38}\sqrt{27}} = 0.686$$

We will calculate hash similarity as follows. The multi set HS_1^4 is

$\{3002837, 3506511, 3420552, 3526396, 3113458, 3522827, 3002837, 3506511, 3420552, 3526396, 3113458, 3241691, 3522827, 3002837, 3506511\}$

and HS_2^4 is

$\{3002837, 3506511, 3420552, 3526396, 3113458, 3241679, 3522452, 2991208, 3146030, 3327889,$
$3424405, 3645843, 3122238, 3513862, 3648427, 3202342, 3226523, 3052623, 3202831, 3241691, 3522827, 3002837, 3506511\}$

Now

$$HashSimilarity_4(D_1, D_2) = Similarity(HS_1^4, HS_2^4)$$

6 Your Task

You will implement three classes to compute similarity between two Strings.

- BruteForceSimilarity
- HashStringSimilarity
- HashCodeSimilarity

6.1 BruteForceSimilarity

For this class, the only data structures that you are allowed to use are `array` and `ArrayList`. You are not allowed to sort any of the arrays/ array lists. You are not allowed to use `HashTable` data structure that you designed earlier.

This class has following constructors and methods:

`BruteForceSimilarity(String s1, String s2, int sLength)`. `sLength` is the shingle length that should be used to compute the similarity between strings $s1$ and $s2$. You may assume that the strings are pre-processed. I.e, all punctuation symbols, white spaces are removed and all letters are in lowercase.

Let S be the multi-set of all shingles (of length `sLength`) of the string `s1`. Let T is the multi-set of all shingles (of length `sLength`) of the string `s2`.

`lengthOfS1()` This method returns the $VectorLength$ of S. Type of this method must be `float`.

`lengthOfS2()` This method returns the $VectorLength$ of T. Type of this method must be `float`.

`similarity()` This method returns $Similarity(S, T)$. Type of this method must be `float`.

6.2 HashStringSimilarity

For this class, you are allowed to use the class `HashTable` that you designed. In addition, you can only use `array` and `ArrayList`.

This class has following constructors and methods:

`HashStringSimilarity(String s1, String s2, int sLength)`. `sLength` is the shingle length that should be used to compute the similarity between strings $s1$ and $s2$. You may assume that the strings are pre-processed. I.e, all punctuation symbols, white spaces are removed and all letters are in lowercase.

Let S be the multi-set of all shingles (of length `sLength`) of the string `s1`. Let T is the multi-set of all shingles (of length `sLength`) of the string `s2`.

`lengthOfS1()` This method returns the $VectorLength$ of S. Type of this method must be `float`.

`lengthOfS2()` This method returns the $VectorLength$ of T. Type of this method must be `float`.

`similarity()` This method returns $Similarity(S, T)$. Type of this method must be `float`.

6.3 HashCodeSimilarity

For this class, you are allowed to use the class `HashTable` that you designed. In addition, you can only use `array` and `ArrayList`.

This class has following constructors and methods:

`HashCodeSimilarity(String s1, String s2, int sLength)`. `sLength` is the shingle length that should be used to compute the similarity between strings $s1$ and $s2$. You may assume that the strings are pre-processed. I.e, all punctuation symbols, white spaces are removed and all letters are in lowercase.

Let S be the multi-set of `hashcode`s of all shingles (of length `sLength`) of the string `s1`. Let T is the multi-set of `hashcode`s of all shingles (of length `sLength`) of the string `s2`.

`lengthOfS1()` This method returns the $VectorLength$ of S. Type of this method must be `float`.

`lengthOfS2()` This method returns the $VectorLength$ of T. Type of this method must be `float`.

`similarity()` This method returns $Similarity(S, T)$. Type of this method must be `float`.

7 Report

In addition to the program, submit a report with following:

1. Pseudo codes for methods `add`, and `search(Tuple t)` methods from `HashTable`.
2. Derive and state asymptotic run-times of above methods. Express run-time as a function of n, and k, where n is the number of elements on the Hash Table and k is the length of the String.
3. For each of the classes `BruteForceSimilarity`, `HashStringSimilarity`, and `HashCodeSimilarity`
   - Describe the data structures used.
   - Pseudo code for all three methods
   - Derive and state asymptotic run-time of all three methods. Express run-times as functions of n, m and k. Where n and m are lengths of the strings and k is the shingle length parameter.

4. You are provided two test files. Convert the files into strings and run the method `similarity` from all three classes. Use 8 as shingle length. Report the similarities returned and the run-times.

5. Compare all three run times. Which one is smallest? Which one is largest? Does one run time equal (or very close to) another run time? Explain why one run-time equals (or very close) /smaller/larger than the other run times.

6. Do all three methods return the same value for similarity? If not, explain the reason.

8 Data

Attached are two text files on which you will report runtimes and similarities.

9 Guidelines

You are not allowed to use any external libraries. You are not allowed use Java inbuilt classes such as `HashMap`, `HashTable` etc.

Your code **must strictly adhere to the specifications**. The names of methods and classes must be exactly as specified. The return types of the methods must be exactly as specified. For each method/constructor the types and order of parameters must be exactly as specified. Otherwise, you will lose a significant portion points (even if your program is correct). You may design additional helper classes and methods.

Your grade will depend on *adherence to specifications*, *correctness of the methods/programs*, *efficiency of the methods/programs*, and `report`. If your programs do not compile, you will receive **zero** credit. Judicious use of *roll-over hashing* will reduce run-time for methods from classes `HashStringSimilarity`, and `HashCodeSimilarity`.

10 What to Submit

Your submission must have following files.

- Tuple
- HashFunction (Please do not change the code).
- HashTable
- BruteForceSimilarity
- HashStringSimilarity
- HashCodeSimilarity
- report.pdf (must be in pdf format)
- ReadMe.txt (list the name(s) of team member(s)).
- Any Other Helper classes you created.

You must include any additional helper classed that you designed. Please include only source .java files, do not include any .class files. Please remember to use default package for all the java classes. Place all the files that need to be submitted in a folder (without any sub-folders) and `zip` that folder. Submit the `.zip` file. Only `zip` files please. Please include all team members names as a JavaDoc comment in each of the Java files. Only **one** submission per team please.