Homework 4

Due: 4:00pm on November 29, 2022 (Tuesday)

Objectives:

- Have a deeper understanding about how HashTable works by implementing it from scratch.
- Get a chance to think about what makes a good hash function.

In homework assignments 1 and 3, you implemented `MyArray` and `SortedLinkedList` to store words parsed from a text file. This time, your goal is to make search very fast. So, your plan is to implement a new Data Structure class

```
public class MyHashTable implements MyHTInterface
```

MyHashTable must have the following **properties and capabilities.**

- This is not an exercise for recursion. Do not use recursion.
- It implements MyHTInterface that is provided to you. Examine this interface carefully.
- It uses an array as its underlying data structure and each element of the array is DataItem (private static nested class). Each DataItem has String as its data value and int value to keep track of the frequency of its data value. That means, if you add the same data value (string) twice, the frequency of the data becomes 2. You must use the following code for the underlying array in your code.

```
private DataItem[] hashArray;
```

- Do not use any other nested class than the DataItem class.
- As you learned from the lecture, it does not care about the order because our focus is to make search fast.
- It uses **linear probing** as its collision resolution mechanism.
- It has the default capacity of 10 and the default load factor of 0.5. Along with a no-arg constructor, it should also have another constructor that allows the clients to pass an initial capacity value. If the initial capacity is less than or equal to 0, it should complain by throwing a runtime exception. To make it simpler, we won't allow the user to pass a load factor but relies on the default load factor.
- Its hash function should be implemented based on the algorithm shown in the lecture 11. For example, the hash value of "cats" is: (3*27^3 + 1*27^2+20*27^1+19*27^0) = 60,337 % tableLength.
- However, it recognizes the importance of making the hash function more efficient. For that purpose, ***Horner's method*** (http://en.wikipedia.org/wiki/Horner_scheme) must be used in its hash function as follows.

var4*n^4 + var3*n^3 + var2*n^2 + var1*n^1 + var0*n^0 can be rewritten as follows using Horner's method (((var4*n + var3)*n + var2)*n + var1)*n + var0

- It should rehash the current hashtable so that the load factor stays within (less than or equal to) 0.5 at any point. The load factor is the ratio of the number of data items (not including deleted flags) in a hash table to the length of the array.

- As it rehashes, it should print how many items are now being rehashed and what the new length of the table is to users. (Check the expected results while testing your program with `HTTest.java` file.)
- Also, when rehashing, increase the underlying array by twice bigger than the current length. However, if the new length is not a prime number, then you should use the next prime number that is larger than the new length. For example, if your current array length is 2 and you need to rehash, 4 is twice bigger than the current array length but you need to use 5 instead of 4 because 4 is not a prime number and 5 is the next prime number.
- While rehashing, think about how you would deal with those deleted items. Would it make sense to keep them or not?
- Unlike Map or Set interface of the Java Collections Framework, it has a few useful (?) methods that can be helpful for a better understanding of HashTable such as getting the number of collisions and getting the hash value of any string.
- To not waste any time, the calculation of size (number of items) and number of collisions should occur while inserting a new element or rehashing. As a result, when the user calls `size()` method or `numOfCollisions()` method, its running time complexity is O(1).
- If you delete an existing string with frequency 2, the final frequency of the string becomes 0. That means, no matter what the frequency of the word is, the delete operation should mark the space as deleted (not decreasing the frequency by one). And, of course, the size (number of items) should be updated accordingly. But, to make it simpler, you do not need to worry about updating (more specifically reducint) the number of collisions when remove method is invoked.

Just like homework assignments 1 and 3, **you are NOT allowed to use any of the Collections Framework which means there should be NO imports in your source code**. To make it simpler, you can assume that your clients know that they need to convert all the words in a file to lowercase before inserting them into your hashtable. Thus, you should not convert strings to lowercases in your implementation.

A word is a sequence of letters [a..zA..Z] that does not include digits [0..9] and does not include other characters than the sequence of letters [a..zA..Z]. Even though the word definition here talks about uppercase letters (this is to make the definition be consistent with other assignments), **you can assume that we won't test your code with any uppercase letters.** Also, your implementation should not make any assumption about the length of words. Here are examples of non-words: abc123, a12bc, 1234, ab_c, abc_

In case of non-words, do not throw an exception. Instead, gently ignore any non-words in your implementation.

Testing:

There will be one test case provided to you, HTTest.java. ***Your output should match the output in the expected result comment section in the HTTest.java file.*** There are a lot more cases that you should think about and test on your own. Also, test your program on your own with other text files.

Be careful!

- When display() method is called, DataItem should show its String value and its frequency as pair, "**[Value, Frequency]**", and the empty space is represented with "**" and any space that used to have an item but deleted later should be represented as "#DEL#". Please refer to expected results of `HTTest.java` file to make sure your implementation is on the right track or not.

- It is advised to spend some time to think about how to count the number of collisions. The definition of collision you are to rely on is: "Two different keys map to **the same hash value**." Checking the code for lab 4 would be helpful!

There is `MyHTInterface` interface where you can find about the methods to be implemented and their descriptions. Read its JavaDoc comments very carefully!

Deliverables:

- Submit your `MyHashTable.java` file using Autolab (https://autolab.andrew.cmu.edu). Do not zip it! Do not use package! The test code provided to you is a starting point. Test extensively on your own!

Grading:

Autolab will grade your assignment as follows.

- Working code: 90 points
- Coding conventions: 10 points
    - We'll deduct one point for each coding convention issue detected.

Autolab will show you the results of its grading within approximately a few minutes of your submission. You may submit multiple times to correct any problems with your assignment. Autolab uses the last submission as your grade.

The Autolab score is not final. The TAs will look into your source code to check correctness and design and deduct points accordingly. The most important criterion is always correctness. Buggy code is useless (even if you may think a found bug is very minor). It is also critical that your code is efficient and follows the specifications properly. Additionally, it should be readable and well organized. This includes proper use of clear comments. Points will be deducted for poor design decisions and unreadable code, etc.

As mentioned in the syllabus, we will be using the Moss system to detect software plagiarism. Make sure to read the cheating policy and penalty in the syllabus. *Any cheating incident will be considered very seriously. The University also places a record of the incident in the student's permanent record.*

Late submissions will NOT be accepted and, if you have multiple versions of your code file, make sure you do submit the correct version.

WHEN IN DOUBT, CHECK THE QUESTIONS ON PIAZZA FIRST. CHANCES ARE THE OTHER STUDENTS ASKED THE SAME QUESTIONS ALREADY. IF YOU CANNOT FIND THEM, FEEL FREE TO POST QUESTIONS. Do not proceed with your own interpretations!