COSC-211: DATA STRUCTURES
HW6: HASH TABLES

Due Friday, March 23, 9:00am

Reminder regarding intellectual responsibility: This is an individual assignment, and the work you submit should be your own. Do not look at anyone else's code, and do not show anyone your code (except for me and the course TAs).

1 Introduction

In this assignment, you will write and test out several different hash functions for Strings. In general, our goal is to design a hash function that evenly distributes possible keys among the different bins in our hash table. That is, we would like it to be the case that each bin contains about the same number of keys.

The goals of this assignment are for you to observe that some hash functions work better than others and for you to think about how to design a good hash function. There is a programming component and a written component.

2 Your Tasks

Begin by downloading some starter code:

```
$ wget -nv -i https://goo.gl/UTiQmt
```

You should now have two Java files:

- `Node.java` contains code for a Linked List node. You should not modify this class.
- `HashTable.java` contains starter code for a hash table, implemented using chaining. I have provided a constructor that takes a table size as an input parameter; an `add` method that takes a `String` as an input parameter and adds that `String` to the hash table; and a `printCounts` method that prints the number of elements stored in each bin of the hash table. You will also see a `hash` method that currently returns 0; this is there so the code will compile as written. In this assignment you will write four alternative hash functions.

You should also have a text file called `wordsEn.txt`, which contains a very long list of English words (there are 109,582 words in this list).

Now do the following:

1. Write a method (in `HashTable.java`) called `hash1` that takes a `String` as its input parameter and computes the following hash function:

$$h_1(k) = \left(\sum_{i=0}^{k.length} k_i \right) \% m,$$

where k is the key (i.e., the `String` parameter), $k.length$ denotes the number of characters in k, k_i is the ith character in k (recall that you can cast a `char` to an `int`), and m is the number of bins in the hash table. Your method should return the computed value $h_1(k)$.

2. Test your hash function to see how well it performs. Specifically, write some code that adds all of the words in `wordsEn.txt` to a hash table with 2053 bins (usually the number of bins in a hash table is chosen as a prime number), then use the `printCounts` method in `HashTable.java` to print out the number of elements in each bin in the table. **In your writeup:** Plot this data in a bar graph; the x-axis should be the bin number and the y-axis should be the number of elements hashed to that bin. Is `hash1` a good hash function? Why or why not?

3. Write a second method called `hash2` that takes a `String` as its input parameter and computes the following hash function:

$$h_2(k) = \left(A \sum_{i=0}^{k.length} k_i - \left\lfloor A \sum_{i=0}^{k.length} k_i \right\rfloor \right) m,$$

where $A = \frac{\sqrt{5}-1}{2}$. This is the *multiplication method* of computing a hash function. The parenthesized term is the decimal portion of the result when a numerical version of the key is multiplied by a constant A, and then this is scaled up by the number of bins in the hash table (the final result should be cast to an `int`). The idea behind this method is to use an irrational multiplier so that any patterns in the computed hash functions (like the patterns you might have observed with `hash1`) disappear. Usually $A = \frac{\sqrt{5}-1}{2}$ is a fairly good choice for this multiplier.

4. Again, test your hash function to see how well it performs. **In your writeup:** Make another bar graph showing the results when you add all of the words in `wordsEn.txt` to a hash table using `hash2` as your hash function. Is `hash2` a better hash function than `hash1`? That is, does it do a better job getting about the same number of elements into each bin?

5. Write a third method called `hash3` that takes a `String` as its input parameter and computes the following hash function:

$$h_3(k) = \left(\sum_{i=0}^{k.length} (i+1) k_i \right) \% m$$

6. Test out `hash3` as above. **In your writeup:** Make a third bar graph showing your results. Is `hash3` better than `hash1` and `hash2`? Why did I multiply each character in the key by its position? What does this do that `hash1` and `hash2` do not accomplish?

7. Write a fourth method called `hash4` that takes a `String` as its input parameter and computes the following hash function:

$$h_4(k) = \left(\sum_{i=0}^{k.length} 31^{k.length-i-1} k_i \right) \% m.$$

This is the built-in hash function that Java uses for `Strings`. When implementing this function, you will need to handle the possibility of `int` overflow (in Java, if you store a value that exceeds $2^31 - 1$ into an `int` variable, the value will wrap around to a negative number). In this assignment you can handle this by simply negating a negative result so that it becomes positive. You may also find that a straightforward implementation of this function, in which you repeatedly call `Math.pow` to compute powers of 31, runs very slowly. To alleviate this, consider the following reinterpretation of $h_4(k)$:

$$h_4(k) = ((31(31(31k_1 + k_2) + k_3) + k_4)...)\,\%m$$

This is called Horner's method for polynomial evaluation, and can be used to write a more efficient method.

8. Test out `hash4` as above. **In your writeup:** Make a fourth bar graph showing your results. How does `hash4` compare to the first three hash functions? What is it about the function that allows it to achieve this performance?

3 Submit your work

Make sure you **test your code thoroughly** before submitting it. Code that does not compile will not receive credit.

Submit your `HashTable.java` using either the submission web site or (from `remus/romulus`) the command line:

```
$ cssubmit HashTable.java
```

Print out a hard copy of the written component of the assignment to submit in class.

This assignment is due on Friday, March 23, 9:00am.