**HW12: Hashing**

**Goals**

To get a better understanding of hash functions and hash tables.

**Reminder:** This assignment is due by the end of Friday (April 23), and there is no grace period.

**Requirements**

This assignment is primarily a problem set, and should be completed individually. You are welcome to talk with others about the problems, but you must write up your assignment by yourself. If you talk with others about the problems, please note their names in your Collaborations.txt.

Please submit a single PDF of your solutions to the written exercises. Please type your answers in a word processing program and convert it to PDF. For problems that ask you to "draw" an array, you could simply create a table to represent the array:

| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| --- | --- | --- | --- | --- | --- | --- |
| 30 |  |  | 16 | 78 | 9 |  |

This assignment is worth 20 points.

**Problem Set**

**Q1**

Please modify your code (or the sample solution) from the in-class lab, and generate the following results:

For each of the two array sizes, 196613 and 200000 buckets, give results for *the number of empty buckets* (different from proportionOfBucketsWithNoWords) and *the average number of items per non-empty bucket* with each of the five hash functions from both words.txt and HoundOfTheBaskervilles.txt. Copy the results from the terminal (as below) and paste them to your file.

m1-mcs-yjiang3:HW12-Hashing yjiang3$ java HashCodeCalculationsSolution HoundOfTheBaskervilles.txt

Size = 196613, Hash function #-1

Number of empty buckets: \*\*\*\*\*\*

Average number of items per non-empty bucket: \*\*\*\*\*\*

Size = 196613, Hash function #0

Number of empty buckets: \*\*\*\*\*\*

Average number of items per non-empty bucket: \*\*\*\*\*\*

...

Then, explain which hash function you would prefer to use (and why) based on these results. You're welcome to supplement your results with anything else you measured about collisions in the lab—use your evidence-based reasoning skills!

**Q2**

Briefly, explain why hashCode2(String) from the in-class lab isn't a very good hash function for English words. Please use examples to bolster your argument.

**Q3**

Imagine I decide to have a hash table with 264 buckets. My hash code function maps to each bucket based on the first four letters of the word (e.g., aaaa has its own bucket, aaab has its own bucket, and so on). Explain why this hash function may be inefficient for storing English words.

**Q4**

I have a Pet class that has instance variables for the animal's name (a String), species (a String), year of birth (an int), preferred toy (a String), and a path to an image of the animal (a String). Two Pets are equal if they have the same name, species, image path, and year of birth. Propose a way to implement hashCode() for Pets such that two Pets that are equal will return the same hash code and Pets that are not equal will tend not to return the same hash code (e.g., just returning the same hash code for all Pets is not a good strategy). You may use any methods you want.

**Q5**

Imagine we have seven buckets in our table (i.e., our array is of length seven). We've decided to handle collision resolution via linear probing. Our hash code takes the first letter of a word and turns it into an integer based on its position in the alphabet such that a=1, b=2, c=3, ..., z=26. For compression, we just use the modulus of the array length.

**a.** Draw the array after adding the following items, in the order below:

llama

rabbit

giraffe

yak

seal

**b.** What indices in the array do I examine if I'm trying to find out if the hash table contains the word turtle? I examine an index if I access that index (e.g., if I say "array[i]" at some point, then I've examined index i).

**Q6**

Now imagine we have ten buckets in our hash table, and we do collision resolution using quadratic probing. We use the standard compression function of the modulus of the array length. We'll be storing integers in our array, and the hash code function returns "the integer mod 10"—that is, the remainder when the integer is divided by 10. Draw the array after adding the following items, in the order below; assume the hash table does not resize during these additions:

38

52

47

19

78

67

71

24