CS 321-002 Data Structures (Fall 2019)

Program Assignment #3 (100 points), Due on 10/28/2019, Monday (11PM)

- **Introduction:**

  Suppose we are inserting n keys into a hash table of size m. Then the load factor α is defined to be n/m. For open addressing $n \leq m$, which implies that $0 \leq \alpha \leq 1$. In this assignment we will study how the load factor affects the average number of probes required by open addressing while using linear probing and double hashing.

- **Design:**

  Set up the hash table to be an array of HashObject. A HashObject contains a generic object, a duplicate count and a probe count. The HashObject needs to override both the `equals` and the `toString` methods and should also have a `getKey` method.

  Also we will use linear probing as well as double hashing. So design the `HashTable` class by passing in an indicator via constructor so that the appropriate kind of probing will be performed.

  Choose a value of the table size m to be a prime in the range $[95500 \ldots 96000]$. A good value is to use a prime that is 2 away from another prime. That is, both m and $m-2$ are primes. Two primes (differ by two) are called "twin primes". Please find the table size using the smallest twin primes in the given grange $[95500 \ldots 96000]$. Vary the load factor α as 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.98, 0.99 by setting the value of n appropriately, that is, $n = \alpha m$. Keep track of the average number of probes required for each value of α for linear probing and for double hashing.

  For the double hashing, the primary hash function is $h_1(k) = k \bmod m$ and the secondary hash function is $h_2(k) = 1 + (k \bmod (m-2))$.

  To test whether a number p is a prime, the following method should be used:

$$\left\{ \begin{array}{ll} \text{if } a^{p-1} \bmod p \neq 1, & \text{then } p \text{ is not a prime} \\ \text{if } a^{p-1} \bmod p = 1, & \text{then } p \text{ is most likely a prime with} \\ & \text{a false positive chance of } \frac{1}{10^{13}} \end{array} \right\}$$

  where a is a random integer with $1 < a < p$. To increase the certainty of the test, please perform the test twice using different random numbers.

  There are three sources of data for this experiment as described in the next section.

  Note that the data can contain duplicates. If a duplicate is detected, then update the frequency for the object rather than inserting it again. Keep inserting elements until you have reached the desired load factor. Count the number of probes only for new insertions and not when you found a duplicate.

- **Experiment:**

  For the experiment we will consider three different sources of data as follows. You will need to insert HashObjects until the pre-specified α is reached, where

  - Data Source 1: each HashObject contains an `Integer` object with a random int value generated by the method `nextInt()` in `java.util.Random` class. The key for each such HashObject is the Integer object inside.
  - Data Source 2: each HashObject contains a Long object with a long value generated by the method `System.currentTimeMillis()`. The key for each such HashObject is the Long object inside.
  - Data Source 3: each HashObject contains a word from the file `word-list` in the directory
    `/home/JHyeh/cs321/labs/lab3/files`
    The file contains 3,037,798 words (one per line) out of which 101,233 are unique. The key for each such HashObject is the word inside.

  Note that, for fair comparison, the data inserted into both linear and double tables must be the same.

  When you hash a HashObject into a table index, you will need to

  - Compute the `hashCode()` of the key of the HashObject.
  - Use the `hashCode()` to perform the linear probing or double hashing calculation. Note that `hashCode()` can return negative integers. You need to ensure the mod operation in the probing calculation always returns positive integers. For example, the computation of the primary hash value (i.e., $h_1(key)$) should be

    ```
    primaryHashValue = key.hashCode() % tablesize;
    if (primaryHashValue < 0)
       primaryHashValue += tablesize;
    ```

    We need to do the same while calculating the secondary hash value ($h_2(key)$). Note that the modulus in $h_2(key)$ is `tablesize - 2`.

  Note that two different objects (key objects) may have the same `hashCode()` value, though the probability is small. Thus, you must compare the actual key objects to check if the HashObject to be inserted is a duplicate.

  Don't copy the `word-list` file to your directory since its is large. Instead set up a symbolic link as follows:
  `ln -s /home/JHyeh/cs321/labs/lab3/files/word-list`

- **Required file/class names and output:**

  The source code for the project. The driver program should be named as `HashTest`, it should have three (the third one is optional) command-line arguments as follows:

  `java HashTest <input type> <load factor> [<debug level>]`

  The `<input type>` should be 1, 2, or 3 depending on whether the data is generated using `java.util.Random, System.currentTimeMillis()` or from the file `word-list`. The program should print out the input source type, total number of keys inserted into the hash table and the average number of probes required for *linear probing* and *double hashing*. The optional argument specifies a debug level with the following meaning:

– debug = 0 ⟶ print summary of experiment on the console

– debug = 1 ⟶ print summary of experiment on the console and also print the hash tables with number of duplicates and number of probes into two files `linear-dump` and `double-dump`. The two sample dump files generated by executing

  [jhyeh@onyx sol]$ java HashTest 3 0.5 1

  are given in the following directory

  `/home/JHyeh/cs321/labs/lab3/files/linear-dump` and

  `/home/JHyeh/cs321/labs/lab3/files/double-dump`

  Please make sure your dump files have the same format as the provided sample dump files so that you can use linux command `diff` *to compare them for correctness checking.*

For debug level of 0, the output to the console is a summary. An example is shown below.

```
[jhyeh@onyx sol]$ java HashTest 3 0.5
A good table size is found: 95791
Data source type: word-list


Using Linear Hashing....
Input 1305930 elements, of which 1258034 duplicates
load factor = 0.5, Avg. no. of probes 1.5969183230332387

Using Double Hashing....
Input 1305930 elements, of which 1258034 duplicates
load factor = 0.5, Avg. no. of probes 1.3904918991147486
```

- **Submission**

  A **readme** file that contains tables showing the average number of probes versus load factors. There should be three tables for the three different sources of data. Each table should have eight rows (for different α) and two columns (for linear probing and double hashing). A sample result containing three tables can be seen in the file below

  `/home/JHyeh/cs321/labs/lab3/files/sample_result.txt`

  Please do not submit executable since I'll be recompiling your programs.

  Before submission, you need to make sure that your program can be compiled and run in `onyx`. Submit your program(s) from `onyx` by copying all of your files to an empty directory (with no subdirectories) and typing the following FROM WITHIN this directory:

  ```
  submit jhyeh cs321 p3
  ```