TCSS 342 - Data Structures
Assignment 8 - Hash Table

Version 1.1.3 (May 17, 2022)

Time Estimate: 6-10 hours

Learning Outcomes

The purpose of this assignment is to:

- Build a static hash table using probing to handle collisions.
- Apply the hash table to a problem well suited to it.
- Gain experience testing and debugging data structures.
- Build data structures exactly to an API's specifications.
- Read a text file and extract the unique words.

Overview

Hash tables are the most efficient data structure for storing data for fast retrieval. The only major drawback of a hash table is that it does not keep its keys in order. If order is not necessary for your purpose then the hash table is the preferred data structure. We will build our own hash table using linear probing with step size 1 as our collision handling heuristic.

While you should test your data structure yourself, we will put the hash table to the task from Assignment 2 - Unique Words. So for this assignment we will upgrade the UniqueWords class.

To complete this you will upgrade one class and implement one public class:

- MyHashTable<Key,Value>
- UniqueWords

Formal Specifications

MyHashTable<Key extends Comparable<Key>,Value>

`-capacity : Integer`
`-keyBuckets : Key[]`
`-valueBuckets : Value[]`
`-size : Integer`
`+keys: MyArrayList<Key>`
`+comparisons : Integer`
`+maxProbe : Integer`

`+MyHashTable(capacity : Integer)`

```
-hash(key : Key) : Integer
+get(key : Key) : Value
+put(key : Key, value : Value)
+size() : Integer
+toString() : String
```

Field summary

- `capacity` - This is the capacity of the hash table.
  - This is set in the constructor and for testing with War and Peace please pass in a capacity of 2^{15} = 32768.
- `keyBuckets` - This array stores the keys according to their hash value.
- `valueBuckets` - This array stores the values according to the hash value of the associated key.
- `keys` - The list of unique keys stored in the hash table.
- `size` - The number of key-value pairs stored in the table.
- `comparisons` - The total number of probes made when putting a new key-value pair.
- `maxProbe` - The maximum number of probes made to put any key-value pair.

Method summary

- `MyHashTable` - Takes the capacity of the hash table as an argument and initializes the bucket arrays.
- `hash` - This method will use the hashcode method of the key to produce an index in the range 0-`capacity`. This method should run in O(1) time.
- `get` - This method returns the value associated with the key.
  a. Use hash to determine the index of the key.
  b. Find the key in the hash table using probing if necessary. Use a step size of 1 for probing.
  c. Using the index of the found key, retrieve the value and return it.
  d. If no key is found, return null.

  This method should run in O(1) time under uniform hashing assumptions.
- `put` - This method updates the value stored with the key.
  a. Use hash to determine the index of the key.
  b. Find the key, or the first null location, in the hash table using probing if necessary. Use a step size of 1 for probing.
  c. Using the index of the found location, write the value into the table.

  This method should run in O(1) time under uniform hashing assumptions.
- `size` - Returns the size of the hash table. This method should run in O(1) time.

- `toString` - Outputs the contents of the hash table in this format:

  [key1:value1, key2:value2, …]

  This method should run in O(n) time where n is the capacity of the hash table.

UniqueWords
…
… +addUniqueWordsToHashTable()

Method summary

- `addUniqueWordsToHashTable` - Adds the unique words to a MyHashTable.
    a. For each word in the list stored in book:
        - Check to see if the word is stored in the hash table of unique words.
            - If it is not, add it to the hash table.
            - Otherwise, skip this word.
    b. Calls toString from the hash table to extract the words.

  This method should time both steps and output each runtime to the console as well as the number of unique words, the total number of probes and the maximum number of probes. This method should run in time O(n) where n is the number of words in the book.

Testing

It is important that you test your code to ensure it works on all potential inputs. Please do this in a separate Main class, and without using additional tools (like JUnit). You will not submit your test code. Instead, your data structures will be tested by code developed by the instructor and/or grader. This code will not be provided to you, as you should test your code before submitting it. If your code fails our tests we will let you know which tests it fails so you can find and correct your errors.

Here is the output from my solution:

```
Reading input file "./WarAndPeace.txt"... 3291642 characters in 78 milliseconds.

Finding words and adding them to a linked list...  in 562 milliseconds.
The linked list has a length of 570240.

Adding unique words to a hash table...  in 61 milliseconds.
20228 unique words
40430 comparisons
191 max probe
Extracting the key-value pairs...  in 7 milliseconds.
```

Submission

You will submit a .zip file containing:

- MyHashTable.java
- MyLinkedList.java
- MyArrayList.java
- BookReader.java
- UniqueWords.java

Grading Rubric

In order to count as complete your submission must:

1. Implement the binary search tree efficiently and without errors.
2. Follow the API exactly so that your classes will work with any testing code.
3. Pass all tests in the testing code. If you fail any tests, you will be told which test failed.
4. Get the numbers right:
    a. Find all 570240 words in the book.
    b. Find the 20228 unique words in the book.
    c. Make 40430 comparisons to put the unique words in the hash table.
    d. Have 191 maximum probes.

Reminder: Incomplete assignments can always be corrected and resubmitted. If they are completed within 7 days of the due date they will count as late and after that period they will count as missed. Please review the grading matrix for the number of permitted late and missed assignments.