Read Me

Part One ----- How I completed the implementation (the structure of my implementation)

In this lab, my implementation is consisted of four big parts in summary.

The first part of my implementation is building an empty table. I used an array of “Record” as the table. And in order to store at least a given “maxSize” of keys, I made the size of the table to be a power of 2. And make sure it is larger than the given “maxSize”.

The second part starts with the implementation of multiplicative hashing. With the return value of “toHashKey” I built two methods “baseHash” and “stepHash”, which return the values of two different hash functions respectively. Combining these two methods, I built another method called “finalHash”. By given a string, this method can return a particular slot index where the string could be stored.

The third part is about implementing the dictionary operation to the hash table. That is to complete the “insert”, “remove” and “find” methods. The most important method out of the three, I think, should be the “find” method. I started from the base slot and check the probable slots step by step with the “stepHash” function. As for “remove” method, I recalled the “find” method and marked the slot as “Deleted” if it could be found by the “find” method. One thing I want to say about the “insert” method is when the load factor is equal to or greater than 0.25, I double sized the table by calling the “doubleSize” method. This comes to the forth part of my implementation.

In the forth part, I made a new array of “Record”, whose size is two times of the original one. In order to rehash the elements in the original table, I build another new “Record” array called “broker”, whose size is the same as the original table. And copy every elements from the original table to “broker”. Then change “this.size” and “this.patternRecord” to the double-sized new empty table. Finally rehash the element in “broker” to the new table. In this way, I think, I can ignore the given “maxSize”.

Part two ------ How to Improve the table

(1) Different strings may have the same value derived from “toHashKey” function. In order to speed up searches by comparing values of “toHashKey”, I added an instant variable in type of integer in “Record” class to store the returned value, and built the relevant setters and getters. Also, I modified the “find” method by firstly checking the “toHashKey” values of the given string and the one stored in the slot before fully comparing the two strings.

(2) The improvement required about doubling the size of the table in this section is illustrated in Part One as the forth part of my implementation.

Part Three ------ Mistakes I made and Acknowledgement

(1) "NullPointerException" I used ".equals()", and ".getKey()" to imply "null".

Thanks very much for the help from Tyler, the TA on Friday and a classmate whose name I carelessly forgot to ask. They gave me a lot of help to fix this mistake.

(2) "StackOverFlow" stupid mistake made by forgetting mod "this.size" in "finalHash" method which also takes me some time to debug.

(3) Thanks Alan, a TA on Saturday, very much. He gave me some very useful advice on improving the "stepHash" function.