Julian Duodu

CSE 241

Lab 2 Write-up

My hash table was implemented using an array of Records that each stored a String Key, the integer result of running that String Key through the toHashKey method in String Table.java, and the ArrayList that was implemented to handle collisions.

My StringTable constructor ignores the maxSize parameter and instantiates a Record array of size two that can then be manipulated by resize and the other methods to complete the functionality of StringTable. In my insert, find, and remove methods, I hashed the key of the provided Records/keys at the beginning of the code before carrying out the individual functions to ensure that all of the Records were correctly inserted, found, and removed from the table. This hashing was carried out by a hash() method that was designed to carry out all of the hashing functionality, so that the rest of my code could be easily debugged.

Extension 1 was implemented by creating a parameter in my Record object which stored the integer form of the String key after it had been run through the toHashKey method in StringTable. I then changed one of the conditional statements of my find method to check the HashKeys of the two compared Records, and if those two hashKeys matched, to compare the actual String keys before returning the Record with the provided key. This greatly speeds up the code because it compares integers about 99% of the time instead of comparing Strings, which turns out to be better because the comparison of Strings is a more expensive process when compared to the comparison of the primitive integers.

Extension 2 was implemented by creating a resize method that took the Records from the original table and then put them into the new array of doubled size by implementing my insert method on those Records. The insert method had to be called because the Hash functions are dependent on the size of the table. Therefore, it was incorrect to simply copy over the elements from the previous table into the new one, because the values would never be correctly found by the hash function due to the change in table size. Also, vvery time that the insert method was run, my code would first check the load factor of the hash table by counting the number of inserted elements and dividing that by the length of the table. If this number+1 was greater than ¼, then the resize method would be called before continuing with the insert process, thereby ensuring that the load factor never became greater than ¼ at any time.

One bug that were encountered was an infinite loop in the find method that kept the last two tests from passing correctly. The issue was that the table could be full without containing the desired Record with the key that was being searched for. This was fixed by creating a counter that kept track of the number of times the method looped, and forced the find method to stop and return null if the function looped a number of times equal to the length of the table. This effectively says that the value that was being searched for was not contained in the table, and thus returns null for the find method.

Another bug found was during the comparison of hashKeys in the hash method that did not take into account the fact that two strings can have the same hashKey and yet still be unequal. This was handled as explained above in the implementation of extension 1.