CSE 247 Data Structures and Algorithms Spring Semester 2019

Lab 7 Post-Lab Writeup

Assigned: 3/6/2019 *Due Date: 3/22/2019*

After you have successfully completed your implementation of the hash table, answer the following questions.

First, for each of the five methods you implemented, describe briefly (in three sentences or less) how it works. Did it use any instance variables of the class, and if so, which ones? Did you create any helper methods, and if so, what do they do? In particular, describe your computation to map hashcodes to indices in the table.

The methods that you should describe are:

1. `public StringTable(int nBuckets);`
2. `public boolean insert(Record r);`
3. `public Record find(String key);`
4. `public void remove(String key);`
5. `private int toIndex(int hashcode);`

Now answer the following additional questions:

6. Java linked lists, which you used to implement each hash bucket, have both head and tail pointers. Hence, it does not matter to the cost of insertion whether you use `add()` or `addFirst()` to insert a new item.

   If the person using your hash table frequently accesses the *most* recently inserted item(s), which of these two insertion methods is likely to yield better performance, and why? What if the user frequently accesses the *least* recently inserted item(s)?

7. Our hash table is allocated with a fixed number of buckets. If we insert a bunch of values, the *load factor* of the table (number of items over number of buckets) can grow without bound. We'd instead like to maintain the table's load factor $\leq$ some constant L, no matter how big it grows.

   Why would maintaining a fixed maximum load factor help the performance of the table?

8. One way to decrease the table's load factor is to make it bigger; that is, we create a new table with more buckets and then transfer the items from the old table to the new one.

   Sketch pseudocode for this operation. Assume we have an existing array B of m buckets, and we are transferring its contents to a new array B' of $m' > m$ buckets. Please specify which value, m or m', is used by your `toIndex()` function.

9. How much time does it take (asymptotically, on average) to transfer n elements from the old to the new table, assuming simple uniform hashing?

10. Describe a strategy for deciding when to allocate a new table, and what size the new table should be, so as to keep the maximum load factor $\leq L$ while maintaining an amortized average-case cost of insertion $\Theta(1)$. (*Hint*: remember your first couple of studios!)