Jasque Saydyk

Professor Patrick Kelley

CS 249 Section 3140

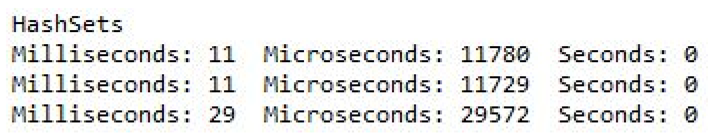
30 April 2017

Project 5 - Hash Tables

HashSet

The implementation of HashSet was relatively straightforward, the difficult part was trying to get the class to work with a large number of different hashes. I achieved this in the add method by making it add one whenever the hash values got stuck repeating for a while, however, I was not able to achieve the same effect on the has method. The issue is that the hash will get stuck in the loop on two hashes, alternating between the two, causing the program to get stuck. When I attempted to add a number to it when it get stuck repeating, it would merely do the same with a different value. For the internal data structure, I used an ArrayList, as it fully supports generics, unlike arrays, can perform all of the operations that an array can, and can be converted into an array with little effort. Aside from that hiccup, the rest of the class was simple enough.

As for the profile, on average, the 100,000 and 200,000 hashset did the same, with the results and the code showing that it is O(1). This is because, rather than sorting the list to find the item, or dividing the list to find the item, the hashset determines the items location using the item itself to generate a number so it can be placed in the data structure, normally an array, at constant time. In the last case, we see the time rise to over double the previous two, as the program is having to go through the array multiple times to locate an open spot for it to store the data. Whilst it can be difficult to determine its efficiency with one data point, looking at the code in a worst case scenario it will be at least O(n), as it goes to every index before finding an empty index. However, it can revisit already checked indexes, so the absolute worst case scenario is that it gets stuck check the same several indexes over and over again, which is a problem that I have ran into with no good solution to.

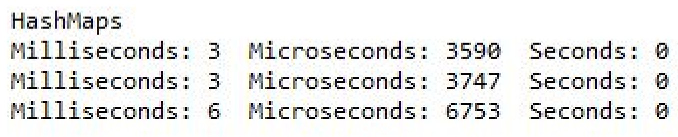


```
HashSets
Milliseconds: 11  Microseconds: 11780  Seconds: 0
Milliseconds: 11  Microseconds: 11729  Seconds: 0
Milliseconds: 29  Microseconds: 29572  Seconds: 0
```

HashMap

The implementation of HashMap was both easy and difficult, as many of the methods where similar to the HashSet class, and I just copied pasted them over with minor edits to work with the IMapPair class used to store the element and key of the data. The only other difficult part of this class was getting the LinkedList to work correctly, which was easier than getting the Double Hash to work in HashSet. Aside from that hiccup, the rest of the class was simple enough.

As for the profile, same as HashSet, on average the 100,000 and 200,000 hashset did the same, with the results and the code showing that it is O(1). This is because none of the LinkedLists are long enough to impact the performance of the data structure. As for the last data point, we see the time increase, which is due to the lengthening of the LinkedLists in the HashMap, thus displaying the weakness of this method, whose worse case is O(n) if all the hashes point to the same spot on the internal array.



```
HashMaps
Milliseconds: 3  Microseconds: 3590  Seconds: 0
Milliseconds: 3  Microseconds: 3747  Seconds: 0
Milliseconds: 6  Microseconds: 6753  Seconds: 0
```