Roleen Ferrer 1731261

Professor Maxwell Dune

CSE 13S - Computer Systems and C Programming

Assignment 6 Write-Up

For this assignment, I was able to implement the Abstract Data Type of a Bloom Filter, a Hash Table, and a Linked List. I learned more about these ADT's, as while implementing each function to create them and after extensive testing to assure that they work properly, I was able to gain a better understanding of the ins and outs of the data types.

While implementing these data types, I gained a lot of memory leak errors when running Valgrind, and occasional NULL dereference errors when running make infer. I was able to fix these by slowing down and contemplating which functions were not properly constructed to create the ADT. Most of the memory leaks stemmed from not freeing memory correctly, especially when freeing all of the heads in a Hash Table and the proper data from the Linked List. This took a lot of time.

After testing and making sure that everything worked, I noticed some differences in output, especially with statistical analysis with the ADT's.

Testing Conditions:

```
./hatterspeak -f 600000 -h 50000 -s
snack snout annoyed annoy big philomel phoebe abishua abishur stunk
```

What happens if you vary the size of the hash table?

Using the given hash table size (50000), I obtain the results:

```
Seeks: 14588
Average seek length: 0.146559
Average Linked List Length: 0.291280
Hash table load: 25.286000%
Bloom filter load: 7.011667%
```

With a hash table size of 40000, I obtain the results:

Seeks: 14588

Average seek length: 0.187551

Average Linked List Length: 0.364100

Hash table load: 30.427500%

Bloom filter load: 7.011667%

It appears that the Average Seek Length, Average Linked List Length, and Hash Table Load all increased when decreasing the size of the Hash Table. This makes sense as there would be more calls to ll_lookup() when decreasing the Hash Table size. The Average Linked List Length would have gone up since the denominator to find the result of the average length is dependent on the hash table length, and the same goes for the Hash Table Load. These also correlate backwards, with a decrease in these statistics if the Hash Table length is increased.

What happens when you vary the Bloom filter size?

Using the given Bloom Filter size (600000), I obtain the results:

Seeks: 14588

Average seek length: 0.146559

Traversed Variable: 2138.000000

Average Linked List Length: 0.291280

Hash table load: 25.286000%

Bloom filter load: 7.011667%

With a Bloom Filter size of 50000, I obtain the results:

Seeks: 14588

Average seek length: 0.146559

Traversed Variable: 2138.000000

Average Linked List Length: 0.291280

Hash table load: 25.286000%

Bloom filter load: 8.351600%

It appears that the Bloom Filter Load will increase if the Bloom Filter Size is decreased, and the same works vice-versa (Bloom Filter Load will decrease if Bloom Filter Size is increased). The Bloom Filter Load is calculated by the number of non set bits in the Bloom Filter divided by the Bloom Filter length. There are going to be fewer non-set bits inside a Bloom Filter with a smaller size, but the denominator (bloom length) is another factor as to why there is more Bloom Filter load when using smaller bloom filter lengths.

Do you really need the move to front rule?

Here are the results when running the program without the move to front rule enabled:

```
Seeks: 14588
Average seek length: 0.146559
Average Linked List Length: 0.291280
Hash table load: 25.286000%
Bloom filter load: 7.011667%
```

And here are the results when running the program with the move to front rule:

```
Seeks: 14588
Average seek length: 0.146490
Average Linked List Length: 0.291280
Hash table load: 25.286000%
Bloom filter load: 7.011667%
```

It appears that the move-to-front rule's average seek length is less than the usage without move-to-front, but it is very little. This makes sense, as using the move-to-front rule will reduce the number of traversals in ll_lookup(), but is this number substantially worth it to continue using the move-to-front rule? With this example, I deduce that there is no reason to use the move-to-front rule as the difference between statistics is very miniscule. Though, if there were a much bigger testing condition, such as parsing through many many more words, we may see a substantial difference in average seek length.