Program 6 – Search Engine:

Comparison of using an AVL Tree vs. a Hash Table for indexing

Andy Rash

CSE2341 – Spring 2015

Prof. Mark Fontenot

<u>Introduction</u>

Program 6—the Search Engine program—sets forth in its requirements to include an interface for an index based on an AVL tree and an index based on a hash table. Between storing all of the parsed data to loading in the data from a persistent index to searching the index for specific terms, one data structure consistently performs better in all tested circumstances: the hash table.

All tests were run on my 2014 MacBook Pro (2.5GHz Intel i7 4870-HQ; 16GB RAM).

<u>AVL Trees vs. Hash Tables</u>

Which one is better?

As stated earlier, the hash table consistently outperformed the AVL tree as an index. Whether it was storing or accessing data, the hash table was constantly better performance-wise than the AVL tree index.

How do you know? Can you quantify this?

I ran the search engine through Instruments (a profiling tool that comes with XCode) multiple times with different sized data sets ascending in size from 1 page to 40 pages to 25,000 pages to the entire corpus of 238,726 pages, and indexing times for the hash table were frequently a minute or more less than the times for indexing using an AVL tree. After looking at the function calls in Instruments, the main difference between the two was that the AVL tree takes quite a bit of time to perform rotates on incoming data. Additionally, when searching for the data, $O(1)$ (amortized) will almost always be better than $O(lg\ n)$ in the case of the AVL tree.

Is one better for small data sets compared to large data sets?

After using the data sets that I did, I found negligible timing differences between the hash table index and the AVL tree index for small data sets and less and less time for the hash table as compared to the AVL tree as the data sets grew larger.