Aurora Havens
Emely Villeda-Principe

Data Structures Report

For the search engine project, we were required to implement two data structures to contain the index data: an AVL Tree structure and a Hash Table structure. Theoretically, a hash table is more efficient than an AVL Tree due to its constant time, O(1), while the AVL Tree has a Big-O of O(logn). Though we suspected that a hash table would better suit our purposes for this project, we tested both structures in order to determine which of the two would be the most efficient.

In order to test the efficiency of the two data structures, we needed to put them under certain program restraints and ensure that each structure received the same amount of attention during the program. We therefore implemented both data structures as derived classes of the Index interface, allowing for easy access to both in the index handler. Though the two data structures handled data differently, they both contained the same type of data (namely and Index Entry object) and had the same basic functions for adding, searching, printing, and emptying. Either data structure could be used to load/store index data using the same functions (thanks to the interface), with the decision on which data structure to use determined during Interactive Mode. When we started testing, we decided to start with AVL Tree as we figured it would the easiest to load from the index into an AVL tree structure instead of trying to load a Hash Table organized index into an AVL Tree. After we decided which structure to load back and forth, we implemented Stress Test Mode and recorded the outputted time of how long it took to load the index into each structure.

Indexing Time – Hash Table vs. AVL Tree		
Number of Documents	Time for Hash Table (seconds)	Time for AVL Tree (seconds)
10	1	2
100	9	13
500	62	213
1000	94	604
2000	130	1820
Full File	20995	293930

****Note**: The jump from a testing size of 2000 documents to the full 171,000 documents is due to the fact that both data structures were taking increasingly longer time to index and we needed to avoid spending too much time on gathering report data

Aurora Havens
Emely Villeda-Principe

Throughout all tests we performed on both structures we found that Hash Table was faster and more efficient. After learning how to create both data structures, we now understand that the reason behind the increasing difference in the indexing times has to do with their methods of adding and organizing data. The AVL Tree relies on node traversing in order to figure out where the value needs to go, and after each addition it must rebalance the tree to keep the data organized. On the other hand, the hash table only needs to assign each key to a unique bucket through a generated a hash and work to make sure that there are no collisions in the table (either through an efficient hash algorithm, linear probing, or chaining).

Size does not seem to matter when comparing hash table to AVL tree. Even with small sets of data, the Hash Table is faster and more efficient than the AVL Tree, though the difference is negligible. However, as the amount of data increased, it became more evident that the underlying recursion involved in AVL Tree indexing made it inefficient. Overall the Hash Table is the better choice, as it is better at storing and searching than the AVL Tree.

Below is a link to our GitHub repository:
https://github.com/SMUCSE2341/EvilAHavens.git