**Comparison of Data Structure Implementations:**

The two types of Data Structures implemented were an AVL tree. The AVL tree implementation was standard using a class called IndexAVLNode as the nodes of the tree. The Hash table is a separate chaining implementation where it is essentially a vector with each index a pointer to an AVL tree, so each word would be hashed and then inserted into the AVL tree corresponding to that index. Theoretically the hash table should be better for the speed of execution because if the size of the hash table is much larger than the size of the buckets (which are AVL trees in this case) then the look up time should be constant (0(1)) vs. a lookup time for the AVL trees of Log2(N). We used the stress test mode to test the speed of the AVL tree vs. the Hash table in handling commands. Particularly, the first half of the stress test mode script would build the AVL tree as the data structure of the index, record the start time, run a number of commands, record the start time and then print the time difference to a file. The second half would be the exact same as the first half except the Hash table structure would be built (but all the same functions would be run and time recorded and output). The following were the results.

Using one simple hash function as shown below the time for execution of the AVL tree vs. Hash table as a function of the size of the Hash table is shown below:

int myhash( HashedObj & x )

{

int hashVal = hash( x );

hashVal %= theLists.size( );

if( hashVal < 0 )

hashVal += theLists.size( );

return hashVal;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of bins in HashTable | 5 | 500 | 5000 | 50000 |
| AVL tree | 9.01752 | 9.07652 | 9.08252 | 9.30453 |
| HashTable | 8.36448 | 9.04352 | 9.12452 | 9.18753 |

As seen from the results, the hash table was only slightly faster in the execution of the commands (and was actually slower in one case). This could be resultant from the fact that the hashing function was not dispersing the elements very well, and with a lot of collisions plus the overhead of the added class then the runtimes were about the same.

Next we implement the hashing function below and compared the runtimes on our large index (of 180,000 documents) and a smaller index:

int hashVal = 2166136261;

      for (const char\*s = key.c\_str(); \*s; s++)

          hashVal = (16777619\*hashVal)^(\*s);

      return hashVal;

|  |  |  |
| --- | --- | --- |
| Size of Data | Large Index | Small Index |
| Avl Tree | 7.9432 | 2.21421 |
| Hash Table | 4.8127 | 0.894919 |
| Percentage Speed ( | 165.0466 | 247.4202 |

As observed from the result the hash table implementation speed was faster than the AVL implementation so it was concluded as better for our search engine which should have the fastest speed on the operations as possible. Technically, the Avl tree would be better if the data set was so small that the overhead of the added Hash table class would be more significant than the increased look up speed but for large data sets as shown in the table above, the Hash table is the better implementation.