Search Engine Final Project – Team: *EmperorPenguins* **Data Structure**

Team Member: ***Skyler Tran***
Jaime Goicoechea
Terry Wang

Analysis Paper

Benefits of AVL Tree:
Fast data structure to:
-Insertion is O (Log(n))
-Finding is O (Log(n))
-Deleting is O (Log(n))
-Keys are sorted, which means you can print primitive types or by overloading the ostream operator you can also print objects stored in the nodes of the tree in order by using depth first traversal.

Cons of AVL Tree:
Dynamic Memory allocation costs time.
Rotations to keep the tree balance costs time.
Harder to implement compared to Hash table.
-Will be very slow with duplicate values.

Benefits of Hash Table with chaining (if collision are low):
Fastest data structure to:
-Insertion is O (1)
-Finding is O (1)
-Deleting is O (1)
-If array of containers(array of arrays, array of vectors, lists, AVL Trees and so on.) is big enough this data structure can have very few collisions which would result in perfect hashing.
-Very easy to implement.

Cons of Hash Table with chaining:
-If you would like to see the elements stored in the hash table they will not be in ordered.
-Hashing takes up time.
- Insertion, finding, deleting require hashing.
-If array is not big enough you need to resize constantly.
-If array is too big you might waste a lot of dynamic memory.
-If there are many collisions, Insertion, deleting and finding is not constant, it can be up to O(N), this is cause by not having a good hash function or small array of containers.
-O(1) is not guaranteed.

Conclusion:
Hashtable is an unordered Data structure where an Avl Tree is a sorted one. When is it better to use one over the other? The performance under different circumstances should be taken into consideration when deciding for either data structure.

Hash table is done correctly with few collisions and good hash function could have O(1) for inserting, deleting and finding. However this is not always the case. The following are scenarios where this data structure might not perform so well:

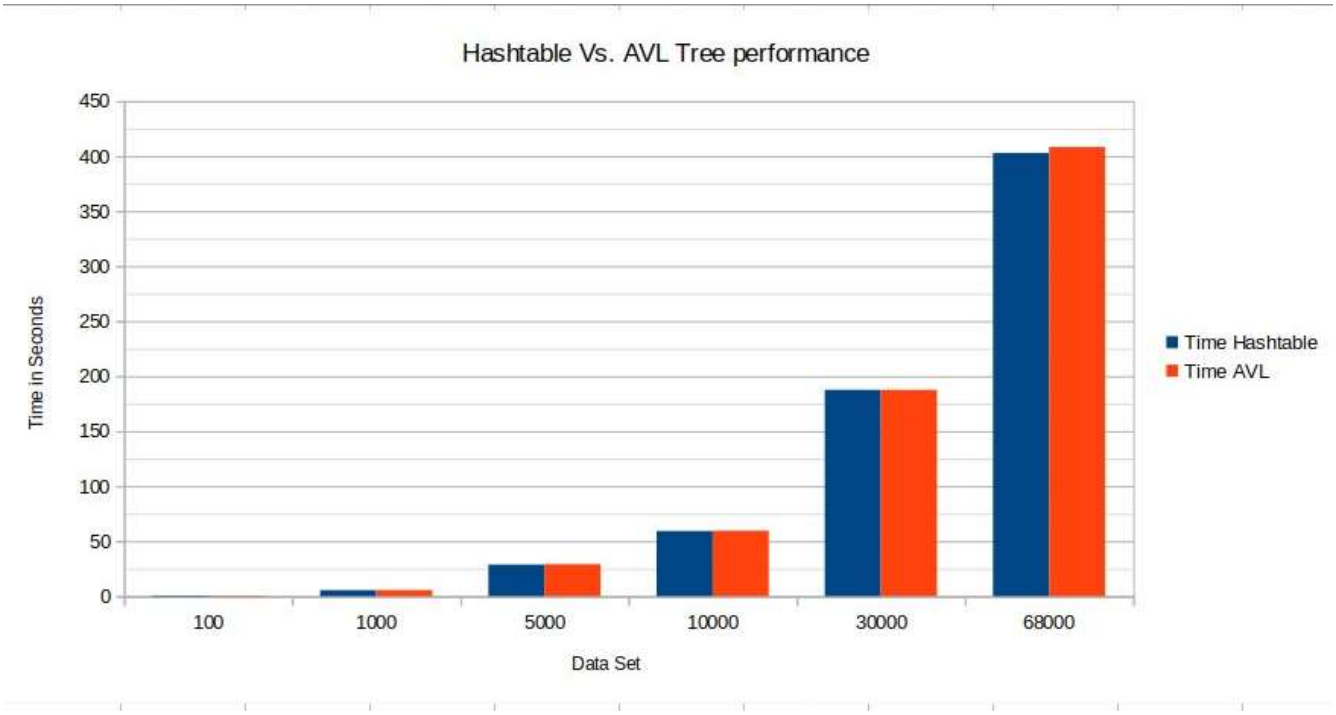
a) Large data set:
The bigger the data there is mostly a chance that there will be more than single digit collisions. If collisions happen to often this will make this data structure O(n) for inserting instead of constant time.

b) Having a small array of containers:
Resizing costs time, if allocate to much or too little memory for the array of containers this will slow down the data structure.

On the other hand AVL Tree:
Can most likely guarantee O (Log(n)) Finding, inserting and deleting and does not need resizing.

Because of the nature of Hash table if you know how much data you have and do not need it to be sorted, a hash table is a good option. However, if you would constantly need to loop up items, check the order they are in and have them in order an AVL tree is a better option.

When implementing our data structures the data set(x-axis) was the number of files in a folder directory(files are .json with .html string objects inside). This graph represents the time in seconds it took to read that number of files, parse, stem and store in each labeled data structure. The performance in both Hash table and AVL tree is very fast. The appropriate size to avoid collisions on the hash table was to allocate an array of lists of pairs the size of 100,000. Having an array this size gave an an average number of collisions to be 7. Thus, keeping the look up and insertion time relatively fast. The number of unique words was around 5 million for 68000 files.



Log(n) refers to Logarithm base 2.