CSCI203:Algorithms and Data Structures

Assignment 1

Ivana Ozakovic, 4790339
Username: io447

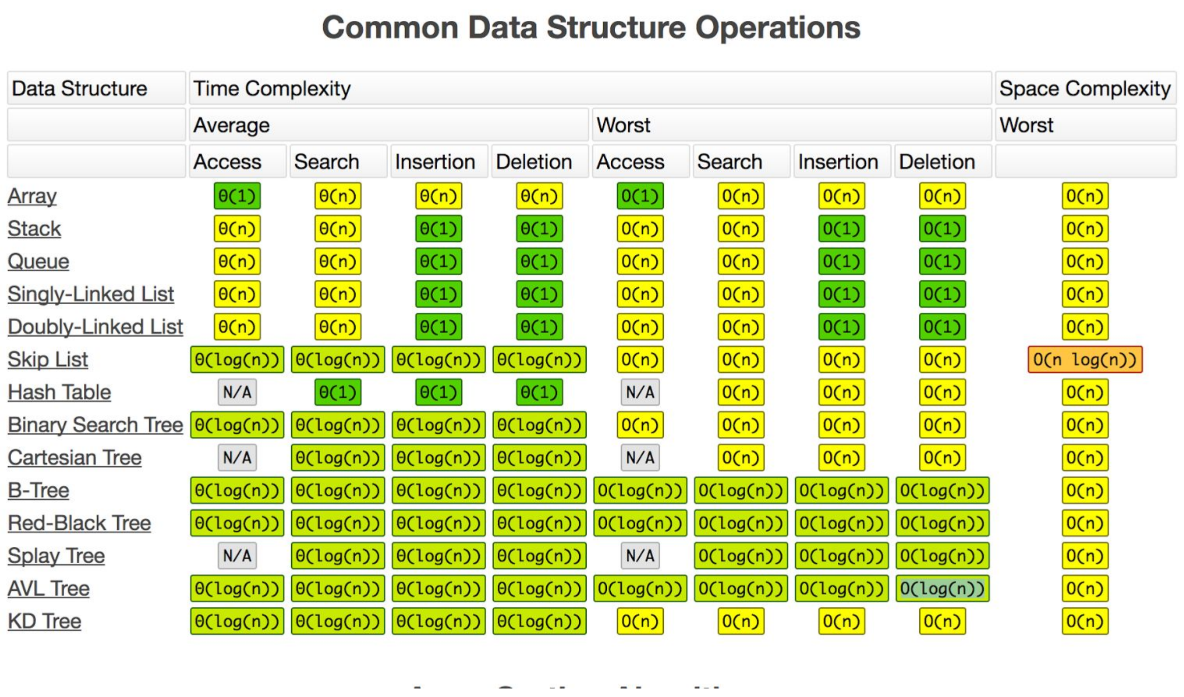
1. A high-level description of the overall solution strategy

Firstly, I have decided to use cstrings (character arrays) and functions such as sscanf() to read in the file as it works much faster than using strings, as strings are stored in a more complex object.

In order to decide which data structure to use in order to solve this assignment problem, I have looked up (refer to image below for information) the efficiency first insertion and searching of the data structures. Later on, I had to consider update and deletion of the nodes.

As it can see from the image below, the AVL tree is the most efficient data structure, as it's average time complexity for each operation is O(log(n)).

Since BST may run into issues when it comes to searching and it can become inefficient since it does not have a balancing factor as in AVL trees, and the structure will have to be sorted first based on occurrences of the word first (decreasing count order), then if there are words with the same occurrence, they have to be sorted alphabetically, which can be accessed by in-order traversal of AVL tree.



Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Image reference: http://bigocheatsheet.com/

The standard insert function had to be modified in order to store data for purposes of this assignment, first by key, then in alphabetical order. After I implemented insertion of the nodes, I ran into problems when trying to update/delete nodes with words that are already stored in the tree, and just need their counts updated and rebalance the tree accordingly.

Since this was quite complex to implement in this particular case (we did not cover it in lectures) and my rebalancing of the nodes did not work as expected, I have tried deleting the node already in the tree, and inserting it again with the new count. Since balancing was hard to achieve in this case in the first place, again I did not manage to implement it properly (I was very close, I just ran out of time, I included attempted functions in the commented section at the end of the program).

I ran out of time and I had to finish the assignment so I decided to first read in the words and store their occurrences, then insert it all into an AVL tree based on occurrence (alphabetically if occurrences equal). This will make tree already sorted, then we can print it using in-order traversal which will give us our solution.

2. A list of all of the data structures used, where they are used and the reasons for their choice.

Reason for the choice is explained in previous section.

1. Array - for reading in the words from file and storing their occurrences.
    - Initially wasn't even going to use array if I managed to finish implementing delete/update functions in AVL tree (which was my purpose first of using AVL tree) but it was difficult for me to apply it in this specific case without being covered in lectures.
2. AVL tree - for inserting words in decreasing occurrence order, and sorting alphabetically words with the same occurrence.

3. A list of any standard algorithms used, where they are used and why they are used.

1. AVL tree insertion and balancing algorithms to store words and their occurrences and sort it.
2. strcpy() - standard c++ function used to copy string into another string

3. strcmp() - standard c++ function used to compare strings in order to insert them into AVL tree in correct alphabetical order
4. tolower() - standard c++ function used to store word in lowercase after parsing it
5. ispunct() - standard c++ function used to check if a string contains punctuation characters in order to remove them