Lab 4: B-Trees

Kimberly Arenas

Introduction:

B-Trees are self-balancing trees that preform operations such as: search, insert, delete, max, min, ..etc. Lab 4 will have the implementations of lab 3 AVL Tree and Red and Black Tree, however the goal here is to now create the implementation of a B tree. By having completed the B Tree implementation the user will now have three options of what data structure they would like their text file of words to be stored in (1. AVL 2. Red and Black 3.BTree).

Solution Design and Implementation:

Again, the code is mostly derived from lab 3 since this lab requires to simply implement a btree into the counting anagram problem. It would be quite redundant to explain the AVL and Red and Black tree implementation so we can jump right to the BTree implementation.

With that being said, the grand focus was to include the btrees basic functions mentioned in the introduction in order to be capable of using the functions the two other trees use as well. Then of course a separate method to count the anagrams and search anagrams would need to be created for the Btree implementation.

Finally the option to select a Btree other than an AVL tree and Red and Black Tree will be inserted. Therefore instead of having only three files this lab will contain four files and that new one being the BTree implementation and then import it to the main execute file as well.

Experimental Results:

The source code for the Btree was provided by our professor who got it from Professor Fuentes. The methods inside the Btree were kept such as they helped contruct the tree and helped with searching for the anagrams. However such as the other two tree implementation there was no need for the removal portion in the btree implementation. Then a key list for searching for the quantity of anagrams was created for the search anagram algorithm.

Lastly, the user input section was modified to select the third option of Btree when asked what data structure they would like their text file words to be stored in.

A screenshot of a cell phone

Description automatically generatedWith that being said, recall that all the trees were tested with text files short as it would be quicker to demonstrate their implementations. First text file had all the anagrams of “stop” and second text file had all the possible anagrams for “alerts”.

Binary Search Tree Comparisons:

|  |  |
| --- | --- |
| AVL | RB |
| 1. .0027 seconds | 1. .0510 seconds |
| 1. .0036 seconds | 2. .0611 seconds |
| 1. .0026 seconds | 3. .0515 seconds |

B-Tree:

|  |  |  |
| --- | --- | --- |
| 5 keys | 10keys | 15keys |
| 3.60 seconds | 2.22 seconds | 1.86 seconds |
| 3.71 seconds | 2.31 seconds | 1.88 seconds |
| 3.66 seconds | 2.27 seconds | 2.01 seconds |

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated Conclusion: the times of all three trees demonstrate how the avl is the fastest implementation of the three with these text files. If I were to improve this program it would be by showing both second options of avl and rb compared with each other since my program seems like it does at the end of both but in reality it is not doing the function.

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provide inappropriate assistance to any student in the class.