Computer Science I Program #5: Word Sorting, Searching (Binary Search Trees)

Please check Webcourses for the Due Date

Read all the pages before starting to write your code

Introduction: For this assignment you have to write a c program that will utilizes binary search tree and either merge sort or quick sort algorithm.

What should you submit?

Write all the code in a single file and upload the main.c file, leak detector c file, leak detector header file, and your test strategy (yourname_test.txt), and your generated three files in1.txt, in2.txt, in3.txt and the corresponding outputs out1.txt, out2.txt, and out3.txt. You need to submit all of them to mimir.

Please include the following commented lines in the beginning of your code to declare your authorship of the code:

```
/* COP 3502C Assignment 5

This program is written by: Your Full Name */
```

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.

Caution!!!

Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report to office of student conduct and an investigation can easily trace the student who shared/posted it. Also, getting a part of code from anywhere will be considered as cheating.

Deadline:

See the deadline in Webcourses/mimir. No late submission will be accepted. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

What to do if you need clarification on the problem?

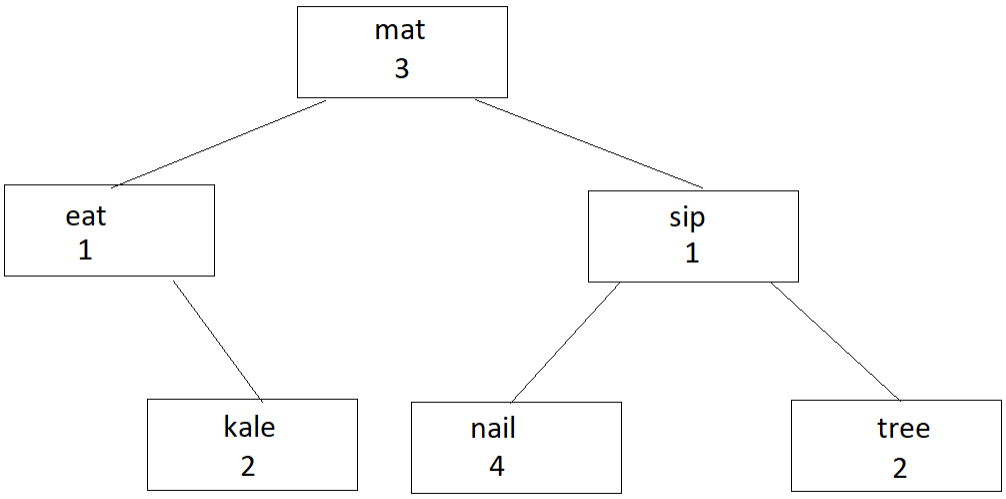
I will also create a discussion thread in webcourses, and I highly encourage you to ask your question in the discussion board. Maybe many students might have same question as you. Also, other students can reply, and you might get your answer faster.

How to get help if you are stuck?

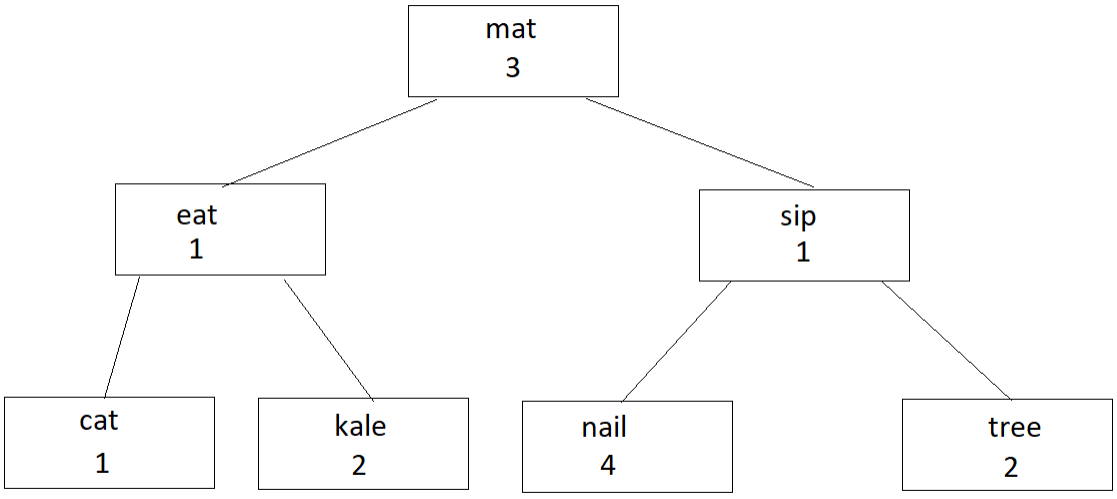
According to the course policy, all the helps should be taken during office hours. Occasionally, we might reply in email, but we cannot guarantee a timely response.

Problem Description

You would like to create a tool that allows you to analyze the frequency of words in various works, while you practice your binary tree skills for COP 3502. You've decided to write a program that reads in words, one by one and stores them in a binary search tree, organized in alphabetical order. You will have one node per each unique word, and in that node, you'll store the word with its frequency. Here is an example picture of a small tree:

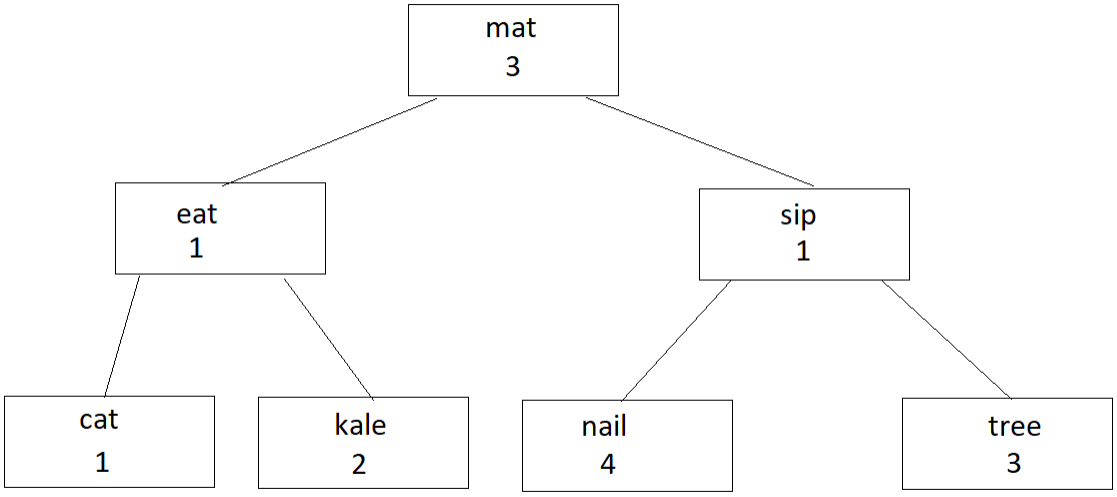


In this tree, the word "mat" is stored in the root and it has appeared 3 times, the word eat is to its left and has appeared once. "kale" which comes before "mat" and after "eat" is the right child of "eat" and has appeared twice. If we were to insert "cat" in this tree, the new picture would look like this:



Since "cat" was not previously in the tree, a new physical node is created, storing cat and showing that it has been seen once.

Subsequently, if we were to insert “tree” into this tree, the new picture would look like this:



Notice that since “tree” was a word that was already processed, no new node is created. Instead, 1 is added to the frequency value stored at the node.

Assignment Part I: Building the Tree, Answering Queries

To prove that you are building the tree correctly, in the middle of the process of reading in some words, you may have to handle some queries. A query is a word, and your program must determine how many times the word has appeared so far **AND** the depth of the node storing the word (depth of a node is how many links must be followed from the root node to get to it). You should store this value in the node structure. If a word has not appeared, you should answer -1 for both questions.

After completing reading in all the words and answering the various queries, your program will move to Part II, described below.

Assignment Part II: Building a Sorted List of Words

For statistical purposes, you want to create a sorted list of words by frequency, from most frequent to least frequent. If two words appear the same number of times, break the tie by alphabetical order, with words that come first alphabetically being listed first.

What you need to do is copy all of the words from the tree into an array of structs, each of which stores both the word and the frequency. (If you are clever about it, you’ll only have one copy of each struct that gets used in BOTH the tree and the array!)

Then, write either a Quick Sort or a Merge Sort and sort the array according to the directions given. Finally, output the list of words with frequencies in the desired order.

The Input (to be read in from in.txt file. Your fopen line should be "in.txt" while submitting)
The first line of input will contain a single positive integer, n ($n \leq 200000$). This will represent the total number of words to insert into the tree and the number of queries.

The input follows on the next n lines, one line for each action. Each of these lines will start with an integer, a, representing the action to be performed. This integer is guaranteed to be either 1 or 2. If this integer is equal to 1, that means the action to be performed is an insertion. If this integer is equal to 2, then that means the action to be performed is a query. In both cases, the integer is followed by a space and then followed by a string of 1 to 20 lowercase letters, representing the word for the action. It is guaranteed that at most 100,000 of these actions will be of type 1 **and that the tree that is created by following all of these actions will never exceed a height of 100.**

The Output (to be printed to standard console output as well as to out.txt file)
For each action of type 2, print a single line with two space separated integers: the number of times the queried word has already appeared and the depth of the node storing the word in the tree as previously defined.

After handling all of the queries, output the final list of words and frequencies sorted in the order designated, outputting each word, followed by a space, followed by its frequency on a line by itself.

Sample Input

```
23
1 mat
1 eat
1 sip
2 mat
2 eat
2 apple
1 mat
1 kale
1 nail
1 kale
1 tree
1 mat
2 mat
2 nail
1 nail
1 nail
1 tree
1 tree
2 tree
1 cat
2 cat
1 nail
2 nail
```

Sample Output

```
1 0
1 1
-1 -1
3 0
1 2
3 2
1 2
4 2
nail 4
mat 3
tree 3
kale 2
cat 1
eat 1
sip 1
```

Implementation Restrictions/ Run-Time/Memory Restrictions

1. You must create a binary search tree of structs, where each struct stores a string (can be statically allocated) and an integer (its frequency). Though each field of the struct will be statically allocated, the structs themselves should be dynamically allocated when needed. (Thus, the node struct should comprise of three pointers - pointers to the left and right substrees and a pointer to a struct that stores the necessary information described above.). It's also recommended to store an integer in the node that stores the depth of the node.

2. For full credit, your algorithm must perform insertions into the binary search tree in O(h) time, where h represents the height of the tree at the time of the insertion.

3. For full credit, you must have appropriately designed functions. **In particular, any correct solution which is fully coded in main will NOT receive full credit.**

4. You create an array of pointers to the structs described above for the second portion of solving the problem.

5. You must have a stand-alone sorting function that takes in the array from requirement #4 and sorts it.

6. You must have to write an auxiliary function compareTo() that takes pointers of two struct (the struct that contains the string and frequency) and returns a positive integer if the first struct should go up in the list if sorted (based on the criteria mentioned earlier). Otherwise, the function should return a negative integer.

7. The algorithm the sorting function implements must be either Quick Sort or Merge Sort. During the sorting process, you must have to use the compareTo function to decide whether the data should go up or down in the sorted list.

8. You must free all dynamically allocated memory for full credit

9. You must have to use memory leak detector like previous assignments to receive any credit.

Deliverables

You must submit the following files in mimir:

1) A source file, *main.c*. If you read from any other file in your code, there will be an automatic 10% deduction.
2) Both leak deter h and c files (The file name should be exactly same as you have it in webcourses)
3) A file describing your testing strategy, *lastname_Testing.txt*. This file discusses your strategy to create test cases to ensure that your program is working correctly. (note that I am not asking you to write how you have written your code. I am asking you to submit the

testing strategy). If you used code to create your test cases, just describe at a high level, what your code does, no need to include it.

4) At least three test files and what you got as the output after running those test files
    a. in1.txt, in2.txt, and in3.txt and their corresponding output files out1.txt, out2.txt, out3.txt (Those output must be generated by your code and they should be correct compared to their input file)

Some Hints:

1. Write use strcmp function to compare strings.
2. Consider adding an int depth parameter in the insertion function and update the depth in the recursion as you keep going deep in the tree during the insertion phase and right before insertion, update the depth in the node
3. Write a search function that lookup your word and return the node containing the word
4. Consider just insertion at the beginning of your code and write the query part later
5. After constructing the tree, consider writing an in-order function to see the words are printed in sorted order or not. If you are satisfied with the insertion, start the query part
6. For the second part, you need to transfer all the dynamically allocated structs to an array
    a. So obviously you need to malloc an array of pointers
    b. Then maybe write a function to load the tree data parts (not the tree node) into the array
        i. In this case you can use any traversal technique
        ii. However, in order to keep track of the index of your array, passing an int reference that was initialized to zero would be a good idea. Then during traversing, you can update the reference and use that as the index of the array.
7. Finally, when your array is ready, consider printing the array before sorting
8. Then call the sort function and during the sorting process, make sure to use proper data types and pointers and properly use the compareTo function
9. As always, after each step, test your code and then gradually update your code
10. Don't forget that your code will be tested with other file while grading. So, make sure you test your code properly to ensure it is working based on the specification