CPSC 319 Assignment 3: Binary Search Trees

Due Date: Wednesday Aug 04th, 2021 11:59pm

Collaboration

IMPORTANT

Discussing the assignment requirements with others is a reasonable thing to do, and an excellent way to learn. However, the work you hand-in must ultimately be your work. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. **Collaborative coding is strictly prohibited.** Your assignment submission must be strictly your code. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. **You can not (even with citation) use another student's code.**
2. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's console, then this code is not yours.
3. Everything that you hand in must be your original work, except for the code copied from the textbook, lecture material (i.e., slides, notes), web, or that supplied by your TA. When someone else's code is used like this, you must acknowledge the source explicitly, citing the sources in a scientific way (i.e., including author(s), title, page numbers, URLs, etc.)
4. Cite all sources of code that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:
   ```
   # the following code is from
   https://www.quackit.com/python/tutorial/python_hello_world.cfm.
   ```
   Use the complete URL so that the marker can check the source.
5. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. However, you may still get a low grade if you submit code that is not primarily developed by yourself.
6. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).
7. Copying another student's work constitutes academic misconduct, a very serious offense that will be dealt with rigorously in all cases. Please read the sections of the University Calendar under the heading "Student Misconduct". If you are in doubt whether a certain form of aid is allowed, ask your instructor!

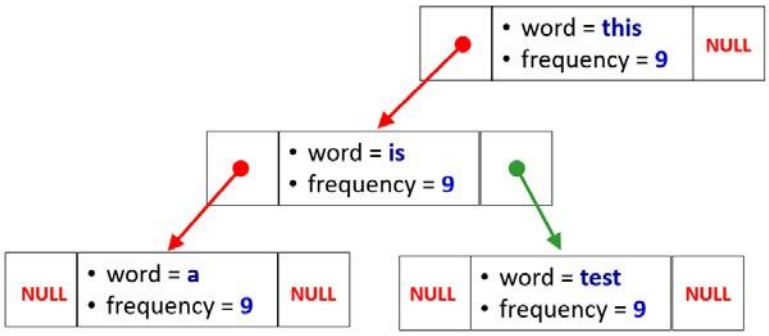
Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize.

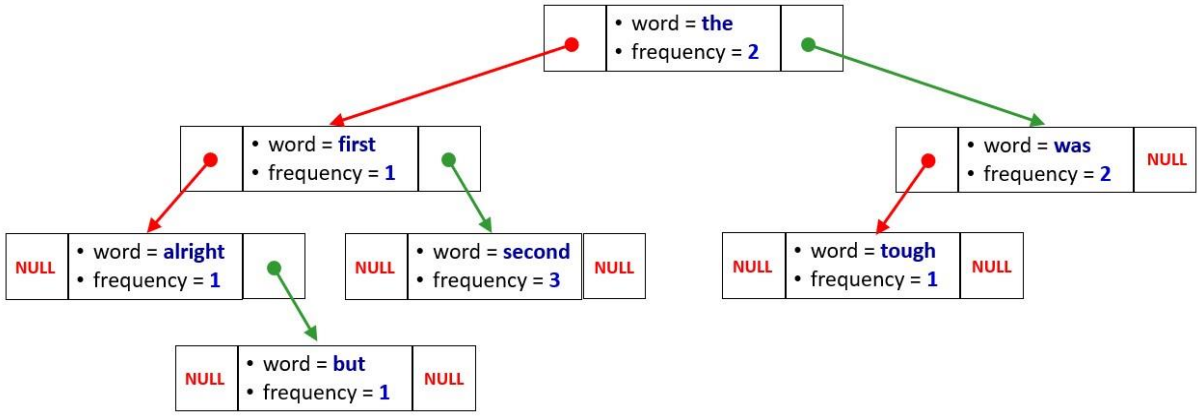
Late Penalty:

LATE ASSIGNMENTS WILL NOT BE ACCEPTED

Goal

Trees are invaluable data structures in software. In this assignment, you will write the code that will create a binary search tree (BST) of unique words from a given input text file, and also keep track of the frequency of the words in the text. For example:

Example 01: Input text file "example01.txt"	**Resulting BST**
This is a test. This is a test. This is a test. This is a test. This is a test. This is a test. This is a test. This is a test. This is a test.	

Example 02 – Input text file "example02.txt"
The first second was alright, but the second second was tough.
Resulting BST


The nodes from the BST store the word itself and its frequency in the input text file. Using this structure for a tree, it is possible to find if a word occurs in the text in $O(\log_2 n)$ (i.e., if the tree is of minimum height) as well as the number of times the word occurs in the text.

Specifications for reading and processing the input text file:

- All text will be considered to be lower-case, even if the word in the file has an upper-case character as the first character, or even if it is all upper-case, or a mixture of upper- and lower-case.
- Words are delimited with spaces.
- All punctuation will be ignored.
- Hyphenated words will be considered as two words (or stated another way, the hyphen is a delimiter just like a space).

- You should use String[] words = line.replaceAll("[^0-9a-zA-Z]", " ").toLowerCase().split("\\s+");

Instructions

Write the Java program that will do the following:
You will be requesting input from System.in, and you will be opening a file in your program to read as input as well. You will be writing output to System.out.

1. Request the user for the name of a text file. This should be an ASCII text file. Example are given with the assignment on the course website.	Example (display screen) for example01.txt: >java CPSC319W20A3 >Enter the input filename: **example01.txt**
2. Use the words from the input files to create the BST specified in the previous page. You will obviously have to do the conversions to lower-case and take care of all of the other characters that do not make up the word as defined above.	

3. When the tree has been created, it should supply the following information as a display output:

(3.1) The total number of words in the file. (total nodes)

Use either IN-ORDER, PRE-ORDER, or POST-ORDER traversal., counting the number of times you visit each node in the BST.

(3.2) The number of unique words in the file. (total nodes with freq=1)

Use either IN-ORDER, PRE-ORDER, or POST-ORDER traversal., counting the number of nodes visited containing the word with frequency = 1

(3.3) The word which occurs most often and the number of times that it occurs.

Use either IN-ORDER, PRE-ORDER, or POST-ORDER traversal., looking for the word(s) with the largest frequency.

(3.4) The maximum height of the tree.

Implement the algorithm **`maximumDepth()`** as given below:

```
int maxDepth(Node node)
  {
    if (node == null)
      return 0;
    else
    {
      /* compute the depth of each subtree */
      int lDepth = maxDepth(node.left);
      int rDepth = maxDepth(node.right);

      /* use the larger one */
      if (lDepth > rDepth)
        return (lDepth + 1);
       else
        return (rDepth + 1);
    }
  }
```

Example (display screen for question #3) for example01 using POST-ORDER traversal (i.e., left, right, node):

> Total number of words in example01 = 4

> Number of unique words in example01 = 0

> The word(s) which occur(s) most often and the number of times that it/they occur(s) =
a = 9 times
test = 9 times
is = 9 times
this = 9 times

> The maximum height of the tree = 3

Example (display screen for question #3) for example02 using POST-ORDER traversal (i.e., left, right, node):

> Total number of words in example02 = 7

> Number of unique words in example02 = 4

> The word(s) which occur(s) most often and the number of times that it/they occur(s) =
second = 3 times

> The maximum height of the tree = 4

4. The user should also be able to request information about any word that is input when requested. The result should be whether the word exists, and the number of times it appears (i.e., its frequency). Use binary search.	Example (display screen) for example01: > Enter the word you are looking for in example01 ? hello > Word not found! > Enter the word you are looking for in example01 ? is > Found! It appears 9 times in the input text file

5. The user should also be able to request to display the entire tree using any of the 3 traversal methods.

The result should be each word in the tree separated by a single space

Example(display screen for question #5) for example01:

```
> Enter the BST traversal method (1 = IN-ORDER, 2 = PRE-ORDER, 3 = POST-ORDER) for example01 ? 1
> IN-ORDER output: a is test this

> Enter the BST traversal method (1 = IN-ORDER, 2 = PRE-ORDER, 3 = POST-ORDER) for example01 ? 2
> PRE-ORDER output: this is a test

> Enter the BST traversal method (1 = IN-ORDER, 2 = PRE-ORDER, 3 = POST-ORDER) for example01 ? 3
> POST-ORDER output: a test is this
```

Example (display screen for question #5) for example02:

```
> Enter the BST traversal method (1 = IN-ORDER, 2 = PRE-ORDER, 3 = POST-ORDER) for example02 ? 1
> IN-ORDER output: alright but first second the tough was

> Enter the BST traversal method (1 = IN-ORDER, 2 = PRE-ORDER, 3 = POST-ORDER) for example02 ? 2
> PRE-ORDER output: the first alright but second was tough

> Enter the BST traversal method (1 = IN-ORDER, 2 = PRE-ORDER, 3 = POST-ORDER) for example02 ? 3
> POST-ORDER output: but alright second first tough was the
```

Hand-In (digitally to D2L dropbox)

1. Your source code file that generates the output in the form requested (in specified format [Java8]). The README.md file should include description of which class begins execution of your submitted code. For example, “CPSC319S21A3.java” is my main class.
2. Include all of the above items in a zipped folder (standard ZIP) titled CPSC319S21A3-LASTNAME-ID.zip
   before submission to the **D2L dropbox** for assignment 3.

Grading

- Assignment grades will be base equally on the two submitted components as described on grading summary

- Source code that does not compile or produces run-time errors will receive a grade of 0%.

- Code that produces incorrect output (Ex. wrong format of output) will be penalized appropriately.

Assignment grading specification is as follows.

GRADING SUMMARY

Name: _____

ID: _____

PROGRAM:		/20
TOTAL		**/20**

PROGRAM

Program builds correctly	Yes	No
Runs: completes with zero run-time errors	Yes	No

IMPORTANT: Source code that does not compile or produces run-time errors will receive 0%

Criteria	TOTAL
Functionality: produces correct output Q1 and Q2: (3 marks) Q3: (15 marks) Q4: (2 marks) Traversal (inorder, preorder, postorder) – (12 marks)	
Readability: code is clear and easy to read (3 marks)	
Generality: handle variety of input and is reasonable error proof (2 marks)	
Documented: code classes/functions/inline commented (3 marks)	
TOTAL:	**/40**