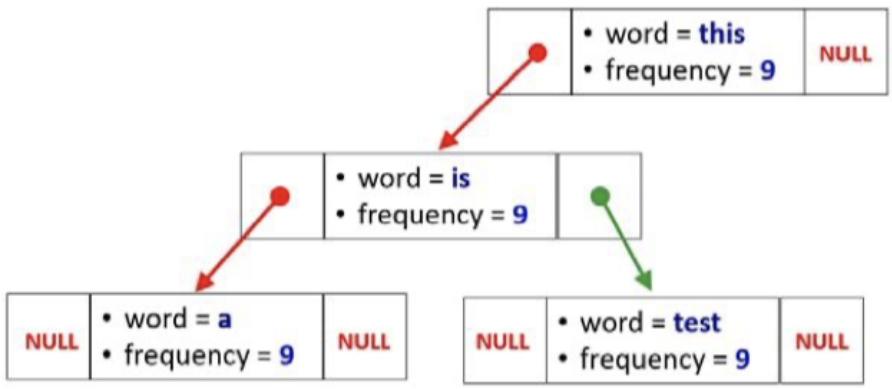
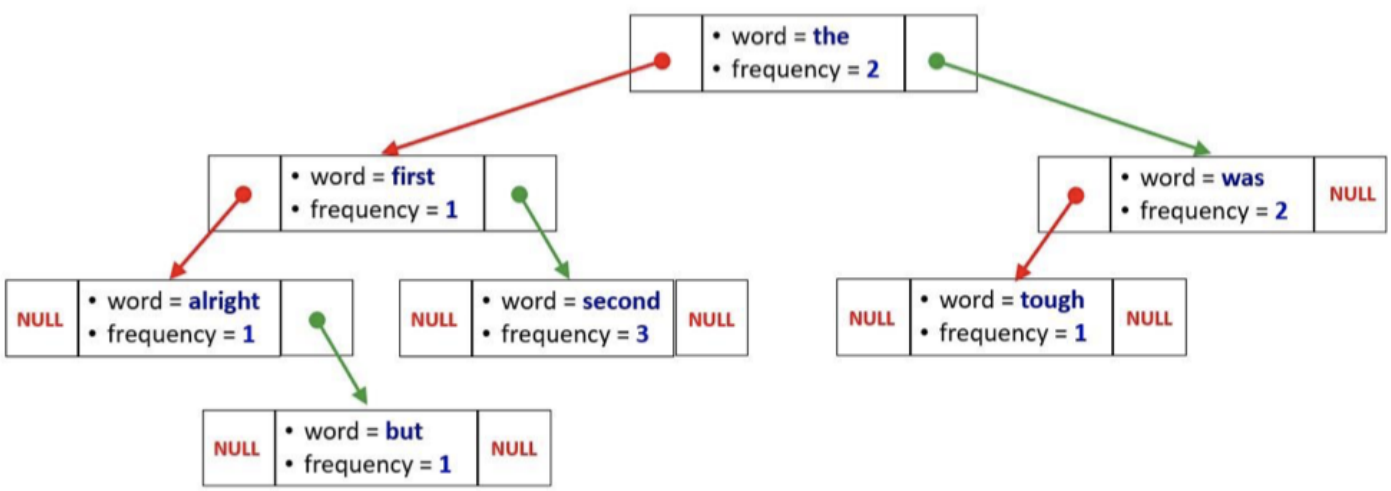
Goal

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 assign7** >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
```