CSCD 427/527 Project #2 (50 points)

PROJECT DESCRIPTION

The purpose of this project is to implement one of the most commonly used index structures in database systems, B+ Tree. While a real B+ tree is usually implemented on database files, this project requires you to implement an in-memory B+ tree to simulate a real system. The project is to parse a web page and build a B+ tree to help you calculate and store the frequencies of words found in a given web page.

GETTING STARTED

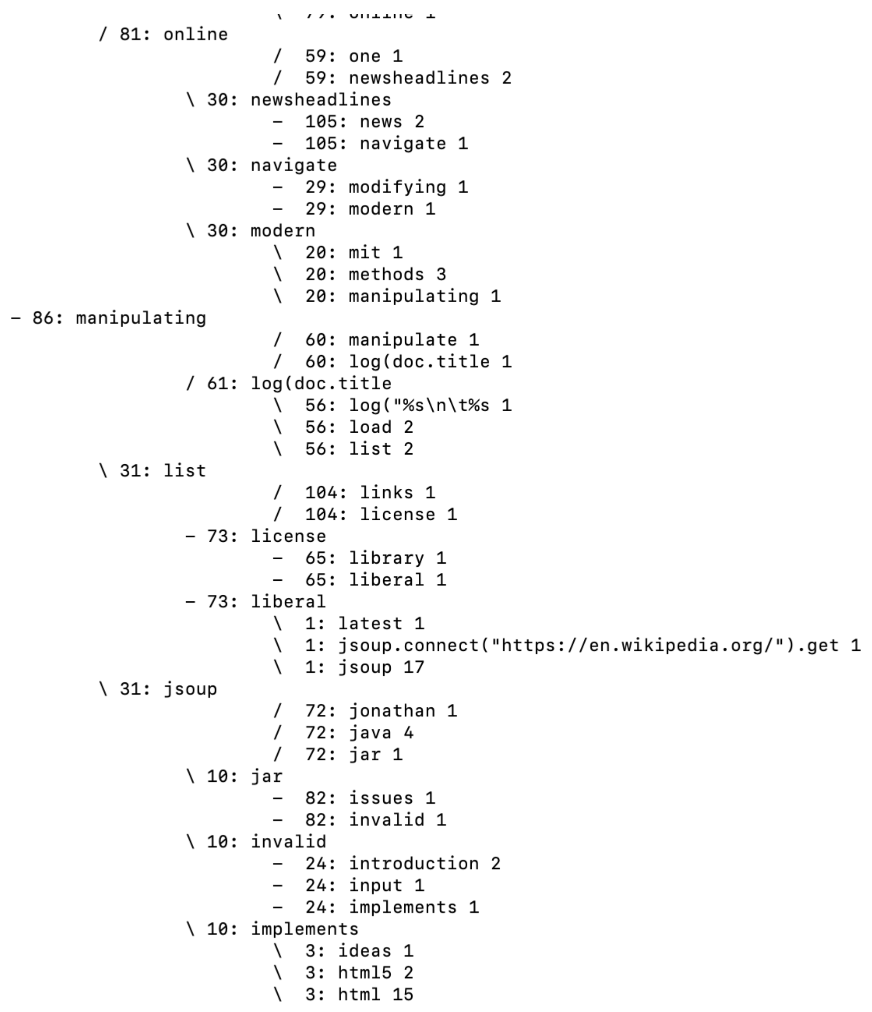
The following classes are defined:

- **`WordFrequency`** class which holds the **`main`** method.
- **`BPlusTree`** class which represents a B+ index tree and stores all the words from a given url.
- **`BTNode`** class which defines a single node in a B+ tree. A **`BTNode`** object stores a variable number of keywords (**Let's set the maximum number of keywords to 3**). If a **`BTNode`** object represents a leaf node, it should also record the frequency count of each word stored in the node.
- **`BTNodeInternal`** class which extends from **`BTNode`** and represents an internal node object.
- **`BTNodeLeaf`** class which extends from **`BTNode`** and represents a leaf node object.

Basically, you don't need to make changes to **`WordFrequency`**, and you only need to have a few changes to **`BPlusTree`** and **`BTNode`** depending on how you are going to pass parameters. The major work is within **`BTNodeInternal`** and **`BTNodeLeaf`**.

Your program will take one or two command line arguments. The first argument is a URL that your program should process. The optional second argument is an ignore list file containing words that should be ignored during processing. If the ignore list file is given, they program will process the ignore list and use a HashSet object to store all these words (The initial ignore list is provided to you in `stopwords.txt`. Feel free to use it as a starting point to develop your own list). Next, the program should process the input .html file. In the give **`WordFrequency`** class, the .html file has been processed and tokenized using **`jsoup`** library. If you don't like how it is used, feel free to make changes. Eventually, for all tokens in the input file but not in the ignore list, you need to add them to a **`BPlusTree`** sorted alphabetically. Only unique words should be added to the tree. When a duplicate word is encountered, just update the frequency count of the word. After processing the input file, your program should enter a loop as shown below, which prompts the user to select from one of the six options:

1. (7 points) Print out all keywords in the `BPlusTree` in alphabetical order. You must implement this function by properly traversing the tree, and then print all the words in order.
2. (12 points) Display the B+ tree structure. This is the challenging part of this project. Basically, you need to implement the BFS tree traversal algorithm applied to a B+ tree. You can decide how you are going to display the tree. Here's an example for your reference:



```
                        \  77: online 1
        / 81: online
                        /  59: one 1
                        /  59: newsheadlines 2
                \ 30: newsheadlines
                        -  105: news 2
                        -  105: navigate 1
                \ 30: navigate
                        -  29: modifying 1
                        -  29: modern 1
                \ 30: modern
                        \  20: mit 1
                        \  20: methods 3
                        \  20: manipulating 1
- 86: manipulating
                        /  60: manipulate 1
                        /  60: log(doc.title 1
                / 61: log(doc.title
                        \  56: log("%s\n\t%s 1
                        \  56: load 2
                        \  56: list 2
        \ 31: list
                        /  104: links 1
                        /  104: license 1
                - 73: license
                        -  65: library 1
                        -  65: liberal 1
                - 73: liberal
                        \  1: latest 1
                        \  1: jsoup.connect("https://en.wikipedia.org/").get 1
                        \  1: jsoup 17
        \ 31: jsoup
                        /  72: jonathan 1
                        /  72: java 4
                        /  72: jar 1
                \ 10: jar
                        -  82: issues 1
                        -  82: invalid 1
                \ 10: invalid
                        -  24: introduction 2
                        -  24: input 1
                        -  24: implements 1
                \ 10: implements
                        \  3: ideas 1
                        \  3: html5 2
                        \  3: html 15
```

In this screenshot, the left-most node is the root. The key inside this node is “manipulating”. It has a left child node which includes keys “jsoup” and “list”, and a right child node which includes key “online”. You may wonder what those numbers 86, 31, 10, etc. are. They are the IDs assigned to the nodes, but they don’t present any ordering. The reason an ID is assigned to a node is to make it clear which keys are within the same node. For example, in Node 10, there are three keys including “implements”, “invalid” and “jar”. If you don’t think this is necessary, you can simply ignore `nextNodeID` and `assignNodeID()` in `BPlusTree`.

3. (12 points) Insert a word. When you insert a word to an existing tree, make sure that (1) it does not allow duplicate words to be inserted to the tree (you only increment the frequency count instead), and (2) split the tree node when needed.
4. (7 points) Search a word. If the word exists, display its frequency count. This function must be implemented by properly traversing through the tree.
5. (7 points) Range search. Ask user to enter two words as lower and upper boundaries, and list all the words in between. Again, this function must be implemented by properly traversing through the tree.
6. Quit.

SUBMISSION

You need to submit a single zip file via the Canvas system. In this zip file, you need to include:

- All the java source files needed for running your program (You must remove any packages from your java files).
- An output file which shows the test result of running your program.
- A readme file which includes all the information you would like to share with the instructor and/or the grader. If there's nothing to share, this file can be omitted.
- `jsoup-1.14.3.jar` if you choose to use it to parse the html file. It must be included in the same folder as your java files.
- `stopwords.txt` if you choose to use this file as your ignore list file. Again, it must be included in the same folder as your java files.

IDE & TESTING

You can use any Java IDE for this project, but eventually you need to make sure you program can compile and run using the following commands in terminal (I assume you use `jsoup` for html processing).

```
% javac -cp jsoup-1.14.3.jar *.java

% java -cp jsoup-1.14.3.jar:.  WordFrequency url [ignore_file_name]
```

Grading

- (45 points) Program Correctness
- (5 points) Testing (results.jpg, results.pdf, etc.)