CSCD 427 Project #2 (50 points)

Due: 11:59pm on May 3rd, 2017

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

You should at least implement the following three classes:

- **`BTNode`** class which defines a single node in a B+ tree. A **`BTNode`** object is used to represent 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.
- **`BPlusTree`** class which represents a B+ index tree and stores all the words in a given web page. You need to implement the **`insertWord`** method of this class. Make sure that (1) your **`insertWord`** does not allow duplicate words to be inserted in the tree, and (2) split the tree node when necessary.
- **`WordFrequency`** class which holds your **`main`** method.

Your program will take one or two command line arguments:

```
% java WordFrequency .html [ignore_file_name]
```

The first argument is a `.html` input file that your program will process. The optional second argument is an ignore list file containing words that should be ignored in the input file. If there is an ignore list command line argument, then your program should process the ignore list first and **create a data structure** for storing the list of tokens to ignore (The initial ignore list has been provided to you in `stopwords.txt`. Feel free to use it as a starting point to develop your own list). Next, your program should process the input .html file. For all tokens in the input file that are not on the ignore list, you will add it 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. Print out all words in the `BPlusTree` in alphabetical order.
2. Display the B+ tree. Rather than showing the contents in each node, your program only needs to display the layout of the tree. The display may look like this:

```
28
        34
                1
                4
```

```
          5
3
          2
          10
20
          17
          27
30
          11
          19
          38
```

Here, 28, 34, 1, 4, 5, etc. denote the node IDs. We can interpret the above B+ tree as: The root is Node 28. It has four subtrees leading by the Nodes of 34, 3, 20, and 30. Similarly, the Node 34 has three subtrees leading by the Nodes of 1, 4, and 5.

3. Select a node to display. If the selected node is a **leaf** node, your program should display all the words contained in the node along with the frequency count of each word.
4. Insert a word.
5. Search a word. If the word exists, display its frequency count.
6. Range search. Ask user to enter two words as lower and upper boundaries, and list all the words in between.

-1. Quit.

Submission

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

- All the java source files you have created.
- 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.

Grading

- (10 points) Compilable program files
- (30 points) Program Correctness
- (10 points) Testing (results.jpg, results.pdf, etc.)