Due on: January 6, 2021

Data Structures

Lab6: Count the frequency of words in a Text file

Given a text file, you are asked to implement an algorithm that returns a list of the most frequently occurring "k" words in the file, in the order of their frequencies. The words that have the same frequency will be given in sorted order. In processing the file:

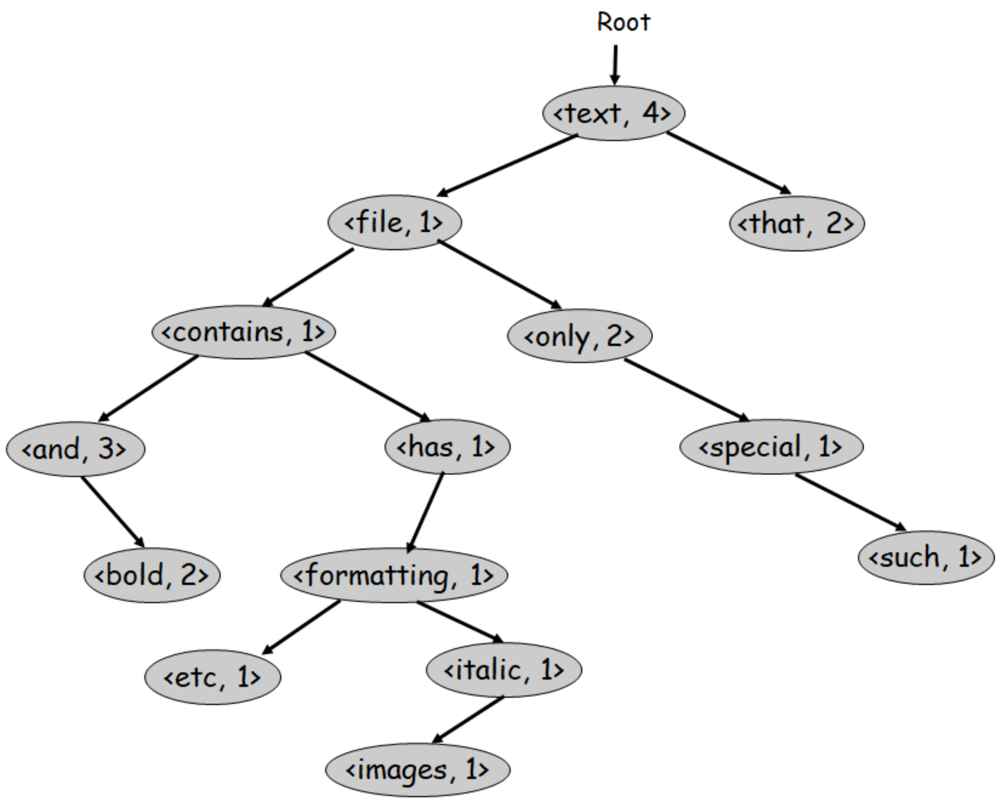
- Read the file line by line, parse each line into its words, and insert each word into a **map** containing <word, frequency> pairs. Convert all words to lower-case characters and then store them in the map. Ignore words that are less than 3 characters long. This means that words such as "a", "to", "in" etc. will be ignored.
- Ignore all non-letter characters. That it, your words will only contain letters in the alphabet 'a'-'z'. Other characters will be used to delimit the words.

As an example, consider the following text file:

```
A text file is a file that only contains
text and has no special formatting
such as bold text, italic text, images, etc.
bold, and, file, that, and.
```

When you process this file, you will get the following <word, frequency> pairs:

Word	**Frequency**
and	3
bold	2
contains	1
etc	1
file	1
formatting	1
has	1
images	1
italic	1
only	2
special	1
such	1
text	4
that	2



You will then be asked to return the most frequent "k" words. For example, if we want you to return the most frequent 7 words from this list, then the result would be given in the following order:

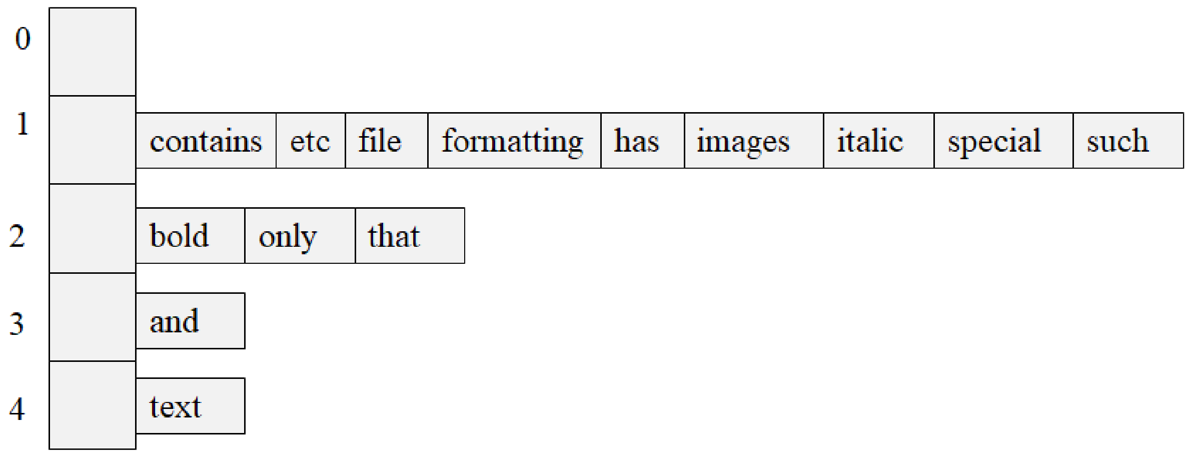
Word	**Frequency**
text	4
and	3
file	3
bold	2
that	2
contains	1
etc	1

Notice that the list is ordered with respect to the frequencies of the words. Also notice that the words that have the same frequency are given in sorted order. That is, "and" comes before "file" in the returned list. Similarly, "bold" comes before "that" both of which have a frequency of 2.

Your algorithm must take O(nlogn + k) time and O(n) space, where "n" is the number of words in the file. Here is how you would implement this algorithm:

1. Implement the map ADT using a BST as the underlying data structure. The map will store <**word, frequency**> pairs. The first time a word is inserted into the map ADT, its frequency will be set to 1. The next time the same word is to be inserted, simply increment its frequency. Thus, when all words are inserted into the map ADT, you will have the <word, frequency> pairs and you can access all words in sorted order. Assuming that we use a regular BST and insert the words in the order we encounter them in the file, the BST will look like as shown in the previous page. We expect you to modify the AVL tree code we gave you and implement this map ADT so that all insertions take at most O(logN) time. Thus, inserting "N" words would take O(NlogN) time.

2. After you process all words in the file and insert them into the map, it is now time to get the most frequent "k" words and return them to the user in the required order. To implement this efficiently, you will allocate an array with "t+1" slots, where "t" is the frequency of the most frequent word. This is 4 in our example. Each slot of the array will store the words that have that frequency. That is, words that have a frequency of "i", where 1<=i<=t will be stored in the list at slot "i". You can use the C++ vector for this purpose. After you insert all words from the map ADT into this array of lists, it should look like as follows:



You can declare this structure in C++ as follows: vector<vector<string>> words(t+1);
Here "t" is the highest frequency, which is 4 in our example.

3. After you have this array of list of words, you can simply start from the last slot, which contains the words with the highest frequency, and walk backwards filling in the result. After you copy “k” words into the result, stop and return the result. If there is less than “k” words, then simply return all the words. Notice that the words that have the same frequency are returned in sorted order. That is, the words “bold”, “only”, “that” each having a frequency of 2 are returned in sorted order.

We have already given you the base code for the project and some test code. You are asked to fill in GetMostFrequentKWords function declared in Solution.cpp. Make sure that you test your code thoroughly with larger data. We will use other tests and other data tests during grading.