CSCE 210/ 2201 Fall 2018 Dr. Amr Goneid
Assignment #5

Date: Monday November 5, Due: Saturday November 17, 2018

Problem: Text Concordance using BST

Given a text document, produce a listing of all the distinct words present in the text and their frequencies.

This problem has many applications in linguistics in general and in ***Computational Linguistics*** in particular. Text concordance is used to construct a ***Corpus***, a body of the words most commonly used in a modern standard language, e.g., Modern Standard English (***MSE***) and Modern Standard Arabic (***MSA***). It is also used in ***Morphological Analysis*** of text, in ***Text Compression*** and in author and style analysis of text.
Because we do not know in advance how many distinct words are there in the document, the straightforward way to do this is to build a ***Dictionary*** of the distinct words (as keys) and their frequencies in the text (as data). However, the number of elements in the Dictionary may grow so large that accessing a word (e.g. by using sequential search) is not efficient. The solution is to maintain the dictionary as an ***array of Binary Search Trees (BST's)***; each BST is for words beginning with a given alphabetical letter.

Objective

In this programming assignment, you will develop a program that would serve as a ***Text Analyzer***. The program would read a text file and generate its text concordance (i.e. associate each word with the frequency of its occurrence in the file). For this purpose, you will maintain a ***Dictionary*** of words and their frequencies. The dictionary is ***an array of Binary Search Trees (BST)*** with each BST storing the words in the file beginning with a given alphabetical letter (or numeric digit) together with their frequencies.
Your dictionary should be a ***growing*** one, i.e., it should accumulate text concordances from several text files processed in separate runs.

Required Implementation:

1. **The BST ADT**

   The ***BST*** ADT can be implemented with the following member functions:
   - ***Constructor:*** Construct an empty tree
   - ***Destructor:*** Destroy tree

- ***insert*****:** Insert an element into the tree
- ***empty:*** Return True if tree is empty
- ***search:*** Search for a key
- ***retrieve:*** Retrieve data for a given key
- ***traverse:*** Traverse the tree
- ***preorder:*** Pre-order traversal of the tree
- ***levelorder:*** level-order traversal of the tree
- ***remove:*** Remove an element from the tree

Design and implement a template class ***BST*** with a minimum of the above member functions. Use a node class that consists of the data, a counter (to count the number of data occurrences), a pointer to left sub-tree and a pointer to the right-sub-tree.

2. Implement a program to build and maintain a dictionary with the following functions:
   - Generate the dictionary for an input text file.
   - Update a dictionary already stored on disk with the one obtained from a new text file. The updated dictionary should give the cumulative text concordance for all text files processed so far.
   - Save an updated dictionary to disk as a text file.

3. Given a dictionary, your program should also provide the following functions:
   - Find how many words are there in the dictionary
   - Search and obtain the frequency of a given word in the dictionary.
   - Given a frequency (n), list the words in each tree having frequency $\geq$ n.
   - Produce a listing of all dictionary words and their frequencies.

Notes on Implementation:

1. Consider the words in the input text file to be **separated either by one or more blanks or by end-of-line.**
2. Words **will not be case sensitive.**
3. **What is a word?** A word in your dictionary is considered to be any string of characters **starting with an alphabetic character or a numeric character.** Ignore words starting with special characters.
4. **Saving the dictionary on disk:** Given the array of BST's in memory, save the nodes in a text file on disk using **traversal**. Be careful in choosing the traversal method in order to obtain the same BST when you download the dictionary back from disk.
5. Your program should distinguish between the following runs:
   - **Initial Run:** Assumes that there are no previous dictionaries stored on disk. Constructs the initial dictionary from the first input text file then saves the resulting dictionary to disk.
   - **Cumulative Run:** Assumes the presence of a previous dictionary on disk. It will download that dictionary from disk into memory (as BST's) and updates it using a new text file. The updated dictionary is then stored on disk.
   - **Query Run:** Assumes the presence of a previous dictionary on disk or already downloaded into memory (as BST's). It will be used in searches or listings.

Test File

The file **“Star Wars Movie.docx”** is provided for testing your program. You may use other text files of your own choice.