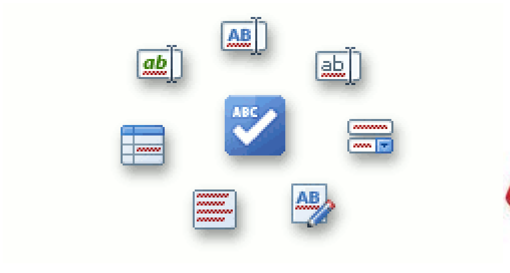
CSCE 2211 Spring 2024 Applied Data Structures
Assignment #4

Dr. Amr Goneid ***Date: Thu Mar 21, Due: Mon April 1, 2024***

The problem: Spell checker, an application of Binary Search Trees

Many applications such as word processors, text editors, and email clients, include a spell-checking feature. A direct way of performing spell checking is to store the set of all valid words in some data structure. For each word we need to check, we'll search the data structure to see if the word is stored. If it is, we will say that the word is spelled correctly. If not, we will say that the word is spelled incorrectly. Therefore, a spell checker's task is centered on searching for words in a potentially large wordlist and reports all the misspelled words (i.e. words that are not found in the wordlist). An efficient search is critical to the performance of the spell checker, since it will be searching the wordlist not only for the words in the input, but possibly suggestions that it might like to make for each misspelling.



Because we do not know in advance the size of the wordlist, the straightforward way to do this is to build a ***Dictionary*** of the distinct words (as keys). However, the number of words in the Dictionary may grow to be very large. 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

- Implement a spell checker that reads a wordlist from a file (**wordlist.txt**), stores the list of words in a ***Dictionary***, then spell-check an input file.
- Your spell checker is **case insensitive**.
- For each misspelling found in the input file, your program will report it, along with a list of 5 suggestions (i.e. 5 closest words from the wordlist).

The Dictionary

- The dictionary is to be implemented as an ***array of Binary Search Trees (BST).*** Each BST stores the words in the wordlist beginning with a given alphabetical letter (or numeric digit).
- Your dictionary should be a ***Dynamic*** one, i.e., it should allow for adding new words or deleting already existing ones.

Suggested closest words:

Definition of closest in this assignment follows the ***Hamming-Distance*** concept. Such distance between two strings is computed using the following algorithm:

For a string A of length |A| = L_A characters, and string B of length |B| = L_B characters:
A match occurs at position k if A(k) = B(k) for all k = 1 …. min (L_A, L_B)
M = # of matches
Distance = Number of mismatches D = max (L_A, L_B) – M

Notice that D = 0 if A and B are identical

For each misspelled word, starting with the same first character, find the list of **five** closest words from the wordlist using the above algorithm.

Required Implementations:

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 a word from the wordlist, a pointer to left sub-tree and a pointer to the right-sub-tree. Add to the class any required private functions.

2. **Implement a program** to build and maintain a dictionary with the following functions:
   - Generate the dictionary from an input **wordlist** file.
   - Update the dictionary by adding or removing words from it.
   - Save an updated dictionary to disk as a text file (See note below*)
   - Read a dictionary stored on disk into memory.
   - Find how many words are there in the dictionary.
   - Spell check an input file such that for each misspelled word list 5 suggestions from the wordlist that are closest to it based on the Hamming Distance measure.

     ***Note:**
     The BST's generated for the first time have to be stored on disk as a text file. This is achieved by traversal of the BST's. You should choose the traversal order that will retain the same BST's when you read them back from disk to memory.