ECE 241: Data Structures and Algorithms- Winter 2024

Project 2: FunWithTrees

Description

This project is intended to familiarize you with operations on binary search trees (BST). The words of a dictionary will be read (at first) from an input file. The project includes various 'unsorted/random' dictionaries and one text file:

1. a tiny dictionary "tiny.txt" of only 3 words (good for testing/debugging)
2. a short dictionary "short.txt" of only 20 words (good for testing/debugging)
3. a large dictionary "english.txt" that contains 99,171 english words (for production).
4. a large dictionary "french.txt" that contains 139,719 french words (for testing).
5. a large dictionary "spanish.txt" that contains 86,017 spanish words (for testing).
6. a sample text file "letter.txt" for testing the spellchecker.

The project also includes multiple 'source' files:

1. `app1.py` to `app3.py`: application files used for testing (provided). **You are not allowed to modify them**.
2. `DictionaryBST.py` (to complete).
3. `Word.py` (provided)

All the functionality of the application files (presented in detail below) should be successfully implemented to obtain full credit. You need to proceed step-by-step, application by application. A list of methods to implement is described at the end of each section.

App1

The application asks the user to load a series of dictionary. The code keeps filling up the BST with new each dictionary entry. General information about the Tree is returned as well as three various tree structures (show method) for a small size dictionary only. For example if you enter the three words dictionary 'tiny' – you can look the file content – (the code should load 'tiny.txt'– just press 'Enter' if you do not want to load additional dictionaries), you will obtain the following result:

```
 Welcome to Dictionary Application 1
====================================

Enter dictionary name (english,french,spanish,short,tiny,etc.): tiny
Enter dictionary name (english,french,spanish,short,tiny,etc.):
```

```
The size of the BST is 3 with 1 level(s)

Display in order:
algorithm; 9; tiny
feast; 5; tiny
numerical; 9; tiny

The BSTree using type -word- looks like:
      numerical
feast
      algorithm

The BSTree using type -id- looks like:
      tiny
tiny
      tiny

The BSTree using type -index- looks like:
      2
0
      1
```

We note that the code is first returning the number of nodes in the BST as well as the Tree height (max number of levels). Then, the nodes are displayed **in order**. Each node contains the object Word which is defined by the word itself, and the name ID of the dictionary it belongs to (the name ID is 'tiny' here). The same structure is shown (using a 90 degree angle) considering the three different options (word, name ID, and the CBT index of the node).

Here an example to understand better the name ID by loading first 'short' then 'tiny':

```
 Welcome to Dictionary Application 1
====================================

Enter dictionary name (english,french,spanish,short,tiny,etc.): short
Enter dictionary name (english,french,spanish,short,tiny,etc.): tiny
Enter dictionary name (english,french,spanish,short,tiny,etc.):

The size of the BST is 23 with 6 level(s)

Display in order:
act; 3; short
algorithm; 9; tiny
animal; 6; short
ate; 3; short
cases; 5; short
cat; 3; short
class; 5; short
code; 4; short
computer; 8; short
dictionary; 10; short
eat; 3; short
```

```
electrical; 10; short
feast; 5; tiny
file; 4; short
morning; 7; short
numerical; 9; tiny
of; 2; short
school; 6; short
screen; 6; short
simon; 5; short
sorting; 7; short
tea; 3; short
this; 4; short

The BSTree using type -word- looks like:
            this
      tea
            sorting
simon
                              screen
                        school
                  of
                        numerical
            morning
                  file
                        feast
      electrical
                        eat
                  dictionary
                                    computer
                              code
                        class
                                    cat
                              cases
            ate
                  animal
                              algorithm
                        act

The BSTree using type -id- looks like:
            short
      short
            short
short
                              short
                        short
                  short
                        tiny
            short
                  short
                        tiny
      short
                        short
                  short
```

```
                                    short
                              short
                        short
                                    short
                              short
            short
                  short
                              tiny
                        short

The BSTree using type -index- looks like:
            6
      2
            5
0
                              46
                        22
                  10
                        21
            4
                  9
                        19
      1
                        18
                  8
                                    74
                              36
                        17
                                    72
                              35
            3
                  7
                              32
                        15
```

Last example of execution:

```
 Welcome to Dictionary Application 1
=====================================

Enter dictionary name (english,french,spanish,short,tiny,etc.): english
Enter dictionary name (english,french,spanish,short,tiny,etc.): french
Enter dictionary name (english,french,spanish,short,tiny,etc.): spanish
Enter dictionary name (english,french,spanish,short,tiny,etc.):

The size of the BST is 324907 with 56 level(s)
```

First, look and understand the file `Word.py` provided to you. Here is then what you need to implement in `DictionaryBST.py`:

1. The class Node (as seen in class) followed by the class DictionaryBST (in the same file). For the class Node you may want to add the additional attribute "index" that will be used to store the CBT index (initialized at zero).

2. A basic constructor for the DictionaryBST class. Hint: you may define additional **private** attributes that will be used to keep track of the number of nodes in the Tree as well as the number of levels (both set at zero).

3. a `load` method that reads a given file and insert all items (Word objects) into the BST. It means that you need to implement the `insert` method that should have its own private auxiliary recursive method (as seen in class).

4. Hint: the `insert` method and its auxiliary `recInsert` method, can be used to both:

   - increment the number of nodes
   - increment the max number of levels (at each time you enter the recursive method you can increment by one the current insertion level, and update the max level if the current level is greater).
   - set the CBT 'index' (level-order position) of the inserted node (see Lecture 18). Rule: The index of root is 0, if a current node has index "index", the index of its left child is "2*index+1" and right child is "2*index+2". The index numbers of a given node can easily be set up during insertion.

5. The `getSize` method

6. The `getMaxLevel` method.

7. The `extractInOrder` method. This is similar than `displayInOrder` seen in class, but here you will create and return a list of Word Objects that are visited in Order.

8. The method `show` and and its "private" recursive auxiliary routine, that shows the 90 degree shifted tree structure with an input option for displaying either the words, the dictionary ID, or the node index (see examples above).

App2

The start of app2 is the same as app1, but then app2 is asking for a number of random words to search in the BST. Timing of the search process is also provided. Example:

```
 Welcome to Dictionary Application 2
=====================================

Enter dictionary name (english,french,spanish,short,tiny,etc.): english
Enter dictionary name (english,french,spanish,short,tiny,etc.):

The size of the BST is 99171 with 37 level(s)
```

```
How many random words would you like to search?: 10000

<<extrapolation>> found in BST: extrapolation; 13; english
<<arable>> found in BST: arable; 6; english
<<hunker>> found in BST: hunker; 6; english
<<spiritualist's>> found in BST: spiritualist's; 14; english
<<Hanover>> found in BST: Hanover; 7; english
<<Margarita>> found in BST: Margarita; 9; english
<<pitifully>> found in BST: pitifully; 9; english
<<Puzo's>> found in BST: Puzo's; 6; english
<<governesses>> found in BST: governesses; 11; english
<<reinstatement>> found in BST: reinstatement; 13; english
Time: 62.521760999999955 ms to search 10000 words
```

What you need to implement in `DictionaryBST.py`:

- The method `search` (method is iterative) that accepts a single "word" as input and returns the corresponding Word object (node) if found (or None if not found).

For this app, in addition to the coding you will create a single pdf document (that will need to be uploaded in Moodle), with the following results:

- For the english dictionary, plot Time vs M where M is the number of random words you want to search (M going from 10000 to 99171). Comments briefly on the results (in term of Big-O).
- Plot another graph of Time vs N (size of dictionary) where the number of random words M is fixed at 50000. By combining english, french, spanish (1, 2 or 3 combinations), you could obtain few data points that can be used to plot (interpolate) your graph. Comments briefly on the results (in term of Big-O).

App3

The goal is to create a spellchecker. Once all the english/french/spanish dictionaries are loaded, this app allows the user to check the spelling of all the words in a text file of his/her choice. The following `letter.txt` file is provided for testing.

```
Bonjour les étudiants!
This is a sample file similar to the one that will
be used to test your projects. It contains spelling mistakes
in  varoius languages.
It will help for testing and developping your software,
un petit peu plus d'efort et se sera bientôt la fin du semestre,
estudiar mucho, la perseverencia es la clave del éxito.
Have fun!
Professor Polizzi
```

Your code must check the spelling of all the words (line by line and along the lines of the file). If a word is not found in the BST dictionary database, it will flag it as incorrectly spelled and return it into brackets '(' ')'. In addition, the code will return for each word the name-ID of the dictionary, and this information will appear every other lines. For example:

```
 Welcome to Dictionary Application 3
=====================================

letter
Enter file name to spell check:
Bonjour les étudiants!
[-french-french-french-]
This is a sample file similar to the one that will
[-english-english-english-english-english-english-english-english-english-english-english-]
be used to test your projects. It contains spelling mistakes
[-english-english-english-english-english-english-english-english-english-english-]
in (varoius) languages.
[-english-no ID-english-]
It will help for testing and (developping) your software,
[-english-english-english-english-english-english-no ID-english-english-]
un petit peu plus (d'efort) et se sera bientôt la fin du semestre,
[-french-french-french-english-no ID-french-french-english-french-english-english-french-french-]
estudiar mucho, la (perseverencia) es la clave del éxito.
[-spanish-spanish-english-no ID-english-english-spanish-spanish-spanish-]
Have fun!
[-english-english-]
Professor (Polizzi)
[-english-no ID-]
```

You only need to implement the `spell-check` method. It is going to load the file scan the text line by line, then word by word which you would need to spell check before printing on screen. Hint: before searching the dictionary, each word would have to be transformed into lowercase (to help the search), and the punctuation at the beginning and the end of this word will have to be removed. For my punctuation string I am using:

```
punc="'!()-[]{};:'"\,<>./?@#$%^&*_~'"
```

you can take advantage of the `lstrip` and `rstrip` methods for string to remove the punctuation. Depending on the result of your search you will have to display on screen the original word with or without parenthesis around. When the word you are searching for is found, the search method should return its Word object that you can use to extract its ID. IDs can be appended into a list that will be displayed every other lines (if you scan the text line by line).

Submissions

Submit a single zip file composed of: All your source files (*.py), and the pdf document including the plots for App2.

Grading Proposal

This project will be graded out of 100 points: App1, App2, App3 are 90 points. Overall programming style: source code should have proper identification, and comments. (10 points)