ECE 242: Data Structures and Algorithms- Fall 2017

Project 4: FunWithTrees (BST and Applications)

Due Date: **Deadline: see website and moodle for submission**

Description

This project is intended to familiarize you with operations on binary search trees (BST). The outcome is very similar to project 1 since it is also using the same dictionary database and operation procedures.

The words of the dictionary will be read (at first) from an input file. The project comes with 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 "sample.txt" for testing the spellchecker.

The project also includes multiple 'source' files:

1. `App1.java` to `App6.java`: application files used for testing (provided). **You are not allowed to modify them**.
2. `DictionaryBST.java` (to complete).
3. `CollectionBST.java` (to complete)
4. `NodeWord.java` (provided).
5. `Word.java` (to complete).
6. `ArrayWord.java` (to complete).
7. `EasyIn.java` file containing the class used to read user input from keyboard (provided)
8. `StdDraw.java` for graphics, library developed by R. Sedgewick

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.java`

The application asks the user to load a series of dictionary (which are non-sorted by default). The code is 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 'short' (the code will here load 'short.txt' internally), you will obtain the following result:

```
>java App1

Welcome to the Tree Dictionary App 1
====================================

Enter dictionary name (english,french,short,tiny,etc.) or return to end:
short
Enter dictionary name (english,french,short,tiny,etc.) or return to end:


The size of the dictionary is 20 using 6 BST levels

Display in-order 20 words
act; 3; short
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
file; 4; short
morning; 7; short
of; 2; short
school; 6; short
screen; 6; short
simon; 5; short
sorting; 7; short
tea; 3; short
this; 4; short


The tree using type (word) looks like:
               this
          tea
               sorting
     simon
                    screen
               school
          of
 morning
          file
     electrical
                    eat
               dictionary
                              computer
                         code
                    class
                              cat
                         cases
          ate
               animal
                    act

The tree using type (id) looks like:
```

```
               short
          short
               short
     short
                    short
               short
          short
short
          short
     short
                    short
               short
                              short
                         short
                    short
                              short
                         short
          short
               short
                    short

The tree using type (index) looks like:
               14
          6
               13
     2
                    26
               12
          5
 0
          4
     1
                    18
               8
                              74
                         36
                    17
                              72
                         35
          3
               7
                    15
```

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, its size (number of letters) and the name ID of the dictionary it belongs too (the name ID is 'short' here). The same structure is shown (using a 90 degree angle) considering the three different attributes (word, size, name ID).

Here an example to understand better the name ID:

```
Welcome to the Tree Dictionary App 1
====================================

Enter dictionary name (english,french,short,tiny,etc.) or return to end:
short
```

```
Enter dictionary name (english,french,short,tiny,etc.) or return to end:
tiny
Enter dictionary name (english,french,short,tiny,etc.) or return to end:


The size of the dictionary is 23 using 6 BST levels

Display in-order 23 words
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 tree 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 tree 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 tree using type (index) looks like:
                14
           6
                13
      2
                     26
                12
           5
                11
 0
           4
                9
      1
                     18
                8
                                74
                           36
                     17
                                72
                           35
           3
                7
                           32
                     15
```

Last example of execution:

```
>java App1

Welcome to the Tree Dictionary App 1
```

```
==================================

Enter dictionary name (english,french,short,tiny,etc.) or return to end:
english
Enter dictionary name (english,french,short,tiny,etc.) or return to end:
french
Enter dictionary name (english,french,short,tiny,etc.) or return to end:
spanish
Enter dictionary name (english,french,short,tiny,etc.) or return to end:


The size of the dictionary is 324907 using 30 BST levels
```

What you need to implement:

1. All the `Word` class including variables, constructor and methods. The methods include all the useful `getWord`, etc. methods, as well as the `toString` method that returns a formatted String object used for the `System.out.println` method (see Examples).

2. In the class `DictionaryBST`, you need private variables, basic constructor, and many methods including: the `loadDictionary` method that uses the `insert` method, the `info` method (see example), the `displayInOrder` method and the `show` method that accepts an argument flag with three different values (so three different outputs as shown in example). **Remark:** you can implement the method `getMaxNbLevel` to return the tree height. Few solution possible here. One of them, is to obtain the maxIndex first and do some math.

App2.java

Once the dictionaries are loaded in the BST (like in App1), the code is going to extract an array of word object using in-order traversal. It will display the list of words (that are sorted) for a small dictionary. The code is then similar to App2 in project1, it is asking the user to enter a number of random words in the array to search for using now the BST data structure. It returns the number of average search steps and the time it takes. Here an example:

```
>java App2

Welcome to the Tree Dictionary App 2
==================================

Enter dictionary name (english,french,short,tiny,etc.) or return to end:
short
Enter dictionary name (english,french,short,tiny,etc.) or return to end:


The size of the dictionary is 20 using 6 BST levels

Array extraction using in-order traversal
act; 3; short
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
file; 4; short
morning; 7; short
of; 2; short
school; 6; short
screen; 6; short
simon; 5; short
sorting; 7; short
tea; 3; short
this; 4; short

How many random words would you like to search?:
10
Ok search done in 1ms
Average number of step is 4
Goodbye!
```

Here another example:

```
>java App2

Welcome to the Tree Dictionary App 2
====================================

Enter dictionary name (english,french,short,tiny,etc.) or return to end:
english
Enter dictionary name (english,french,short,tiny,etc.) or return to end:


The size of the dictionary is 99171 using 30 BST levels

Array extraction using in-order traversal

How many random words would you like to search?:
10000
Ok search done in 9ms
Average number of step is 24
Goodbye!
```

What you need to implement:

1. For the `DictionaryBST` class, the method `extractArrayInOrder` that returns an object of type `ArrayWord`. You also need to implement the `search` method. The latter could be iterative that would make it easier to keep track of the number of steps required to find a word. The method `getStep` returns the number of steps.
2. For the `ArrayWord` class, the variables, constructor and methods such as: `insert`, `display`, `getRandomWord` that returns a word object at random (I use the seed "100", you could do the same).

Other requirements: Compare the binary search approach of project1 with this one, answer the following questions in a separate pdf document (brief answer):

- Why the number of average search steps here is bigger?
- Comments on running/execution time (which one is faster and why).

App3.java

Similarly to App4 in project 1, we propose to create a collection of dictionaries of same size words from an original input dictionary file. We then perform the search (like in App2) using the collection of the various BST dictionaries.

Here an example of execution:

```
>java App3

Welcome to the Tree Dictionary App 3
====================================

Enter dictionary name (english,french,short,tiny,etc.) or return to end:
short
Enter dictionary name (english,french,short,tiny,etc.) or return to end:


The size of the dictionary is 20 using 6 BST levels

Array extraction using in-order traversal

Subarray extraction for word of size 1 to 10 using in-order traversal
-------------------------------------------------
The  collection contains 10 BST dictionaries
dict --> size --> nblevels
1 --> 0 --> 0
2 --> 1 --> 0
3 --> 5 --> 3
4 --> 3 --> 1
5 --> 3 --> 1
6 --> 3 --> 1
7 --> 2 --> 1
8 --> 1 --> 0
9 --> 0 --> 0
10 --> 2 --> 1
-------------------------------------------------


How many random words would you like to search?:
10
Ok search done in 1ms
Average number of step is 1
Goodbye!
```

Here another example:

```
>java App3

Welcome to the Tree Dictionary App 3
====================================
```

```
Enter dictionary name (english,french,short,tiny,etc.) or return to end:
english
Enter dictionary name (english,french,short,tiny,etc.) or return to end:


The size of the dictionary is 99171 using 30 BST levels

Array extraction using in-order traversal

Subarray extraction for word of size 1 to 23 using in-order traversal
-------------------------------------------------
The  collection contains 23 BST dictionaries
dict --> size --> nblevels
1 --> 52 --> 13
2 --> 182 --> 14
3 --> 846 --> 20
4 --> 3352 --> 26
5 --> 6799 --> 28
6 --> 11302 --> 30
7 --> 14788 --> 30
8 --> 15684 --> 30
9 --> 14248 --> 30
10 --> 11529 --> 29
11 --> 8408 --> 27
12 --> 5502 --> 27
13 --> 3234 --> 22
14 --> 1676 --> 21
15 --> 890 --> 20
16 --> 381 --> 21
17 --> 176 --> 13
18 --> 72 --> 10
19 --> 31 --> 10
20 --> 10 --> 3
21 --> 3 --> 1
22 --> 5 --> 3
23 --> 1 --> 0
-------------------------------------------------


How many random words would you like to search?:
10000
Ok search done in 6ms
Average number of step is 8
Goodbye!
```

What you need to implement (this is not a step-by-step, read everything below):

1. In `CollectionBST`, variables, a constructor that initializes the collection of sub-BST, and methods including: `info` that displays the number of words and the number of levels for each dictionary (see example), the new method `search` and `getStep`. For initialing the collection, I suggest to extract in-order the subarrays `ArrayWord` first (they will be sorted), and use them as described below to create a new (balanced) subtree.

2. In `DictionaryBST`, implement a new `initDictionary` method that accepts an `ArrayWord` object, randomized the array, and create a new BST subtree (by inserting word object into it). Other methods include `getMaxSizeWord` (Hint: you can keep track of this number while inserting inside the main BST),

and `extractSubArrayInOrder` that can be used to extract the subarrays of different word length (size of these arrays can be overestimated).

3. In `ArrayWord`, a method `shuffle` that will randomized the list of items (I use the seed "100" again), a method `deleteLast` that can be used to return the last object `Word` (Hint: can be useful for `initDictionary` method).

App4.java

This is a dictionary application. The goal is to find anagrams of a word, and create a new "anagram dictionary". Execution example:

```
 java App4

Welcome to the Tree Dictionary App 4
====================================

Enter dictionary name (english,french,short,tiny,etc.) or return to end:
short
Enter dictionary name (english,french,short,tiny,etc.) or return to end:


The size of the dictionary is 20 using 6 BST levels

Enter word to analyze:
tea
Great! 3 anagram(s) found

The size of the dictionary is 3 using 1 BST levels

Display in-order 3 words
ate; 3; short
eat; 3; short
tea; 3; short


The tree using type (word) looks like:
      tea
 eat
      ate
```

What you need to implement:

- The `createBSTAnagram` method in the `DictionaryBST` class. Hint: you can use the following 'simple' solution (note that 'permutation' algorithms are rather involved): (i) sort the characters in your key word using any sorting algorithm (for example the word "dbca" will give "abcd"); (ii) scan in-order through all the main BST dictionary for words that have the same String length than your key word, (iii) if found, sort similarly the corresponding word by characters and compare with your sorted key word, (iv) if the words are identical, you have found an anagram that you can insert into an ArrayWord, (v) create the anagram tree using the `initDictionary` method.

Remark: You can decompose the letters of a given String 'word' and create a new array of Character 'c' using for example:

```
Character[] c = new Character[word.length()];
for (int i=0;i<len;i++) c[i] = word.charAt(i);
```

App5.java

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

```
Bonjour les étudiant!
This is a sample file similar to the one that will
be used to test your projects. It contains spelling mistakes
in English, Français, and Español.
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!
Eric Polizzi
```

The code will 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 dictionary, it will flag it as incorrectly spelled and return it into brackets '(' ')'. You may still find that there are common words that are not found in the dictionary and thus will be flagged as incorrectly spelled. Do not worry about this. 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:

```
>java App5

Welcome to the Tree Dictionary App 5
====================================


The size of the dictionary is 324907 using 30 BST levels

Enter Text filename to spellcheck:
sample.txt

Bonjour les étudiant!
[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 English, Français, and Español.
[english-english-french-english-spanish-]
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-]
Eric (Polizzi)
[english-no ID-]
```

You only need to implement the `spellCheckFile` method. Make sure to remove periods and commas from words before trying to find them in the dictionary. You will also check the decapitalize version of the

word (using lowercase) to improve the search. You can use the following command to remove the punctuation within a String "word".

```
String word1=word.replaceAll("\\p{Punct}|\\d", ""); //remove punctuation
```

and the following to get the lower case version:

```
String   word1lower=word1.toLowerCase());
```

App6.java

Similar to App1, but here the BST is converted to an array structure (using the indexes), all the non-null items of the array are displayed and the tree structure is plotted. Here an example of execution:

```
>App6

Welcome to the Tree Dictionary App 6
====================================

Enter dictionary name (english,french,short,tiny,etc.) or return to end:
short
Enter dictionary name (english,french,short,tiny,etc.) or return to end:


The size of the dictionary is 20 using 6 BST levels

 Array:
0 morning; 7; short
1 electrical; 10; short
2 simon; 5; short
3 ate; 3; short
4 file; 4; short
5 of; 2; short
6 tea; 3; short
7 animal; 6; short
8 dictionary; 10; short
12 school; 6; short
13 sorting; 7; short
14 this; 4; short
15 act; 3; short
17 class; 5; short
18 eat; 3; short
26 screen; 6; short
35 cases; 5; short
36 code; 4; short
72 cat; 3; short
74 computer; 8; short
```

The resulting Tree structure is given in Figure 1.

What you need to implement:

1. The method `convertToArrayInOrder` in `DictionaryBST` that returns an `ArrayWord` object. You can actually obtain the maxsize of the array using `getMaxNblevel` and some basic math. **Hint:** Use an in-order traversal to visit all the nodes, use the 'index' attribute of a given node to insert it inside the array.

2. The method `set` for the `ArrayWord` class, could also be useful. Accepting a 'new Word' object and its position 'index' as arguments in order to insert it at the right position inside the array.

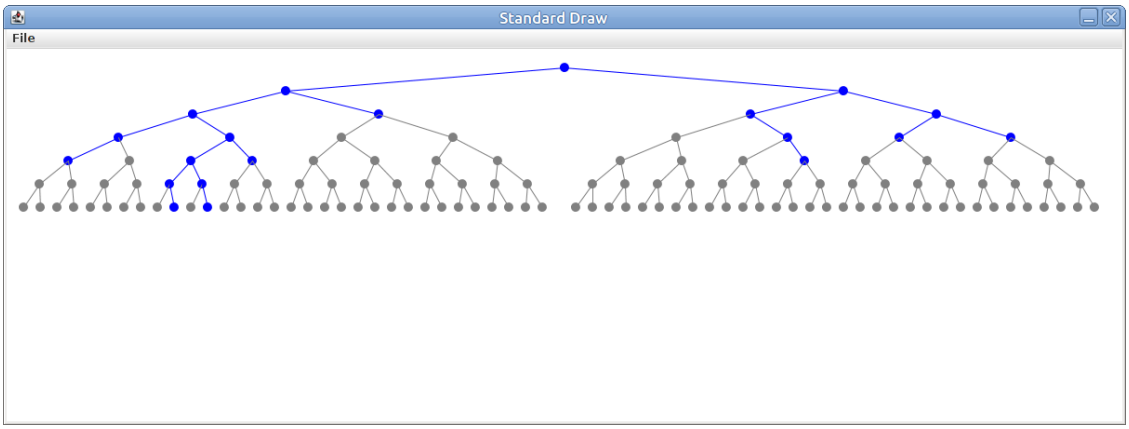

Figure 1: BST representation of "short.txt" database.

3. The method `displayAll`, it displays all the non-null items of the array (with corresponding indexes)

4. The method `plotBST`, it accepts the size of the frame, and plot the tree structure. It plots the entire CBT (Complete binary tree) but highlight only the Movies that exist in the database and their connections. The array representation can be used to know which cells are occupied and it can help defining coordinates x and y for each cell to be plotted. For this project, you may need to use only few methods in the `StdDraw.java` library.

   - `StdDraw.setCanvasSize(..,..)` to set the window size
   - `StdDraw.setXscale(..,..)` to set the x scale (default is [0,1])
   - `StdDraw.setYscale(..,..)` to set the y scale(default is [0,1])
   - `StdDraw.setPenColor(..)` to set the color of the pen (default is `StdDraw.BLACK`)
   - `StdDraw.filledCircle(x,y,r)` to plot a filled circle at coordinate x,y with radius r
   - `StdDraw.line(x1,y1,x2,y2)` to plot a line between two points 1 and 2

   You should write the method such that the BST appears as clearly as possible (see Figure).

Submissions

Submit a single zip file on moodle by the due date (no extension, no exception) composed of: All your source files (*.java) including EasyIn.java, all txt files a README (text) file. Include all information in the README file e.g.: Your name, ID, e-mail, what have you done or what does not work: Indicate which applications have been fully completed (or what has been partially completed in each application), other special features or assumptions made in your program we should know about, and everything else you want to say. Note: Your zip file must be self-contained so the code will nicely compiled without the need of extra-files or other non-standard java libraries.

Reminder: No Extension, No Exception policy. In case of any special circumstances that prevented you to complete your project on time, you need to share all valid justifications in the zip file and readme file. The TA and I will review them while grading.

Grading Proposal

This project will be graded out of 100 points:

1. Your program should implement all basic functionality and run correctly. (85 points).

2. Report and answer to questions (5 points)

3. Overall programming style: source code should have proper identification, and comments. (10 points)

Extra Credit (5pts)- (no help from TA or Instructors)

The plotting part counts has extra-credit, +5 points if it works nicely...