ECE 241: Data Structures and Algorithms- Fall 2022

Project 3: FunWithTrees

Due Date: **Deadline: see Moodle**

Description

For this project we propose to design and operate a movie library. We are going to use both a list and a binary search tree (BST) to store the library. Using a BST allows us to store an unlimited number of movies while being able to both insert and search them in O(logN).

Once the database that contains all the movies is loaded from a file to form a list or a BST, we will be able to search a particular item, or extract a sublist/subtree containing all the movies requested by a customized user search. From this particular list or subtree, one could display the list of items, obtain the corresponding list/tree and show/plot the tree structure.

How to start

The project includes a database file `oMovies.txt` which includes an **ordered** list of 17,770 movies. The movies are listed one after another and each line of the file contains three attributes: ID number, Year, and Title. The first few lines of the file are provided below:

```
10001;2003;Dinosaur Planet
10002;2004;Isle of Man TT 2004 Review
10003;1997;Character
10004;1994;Paula Abdul's Get Up & Dance
10005;2004;The Rise and Fall of ECW
10006;1997;Sick
10007;1992;8 Man
10008;2004;What the #$*! Do We Know!?
10009;1991;Class of Nuke 'Em High 2
10010;2001;Fighter
10011;1999;Full Frame: Documentary Shorts
10012;1947;My Favorite Brunette
10013;2003;Lord of the Rings: The Return of the King: Extended Edition: Bonus Material
10014;1982;Nature: Antarctica
10015;1988;Neil Diamond: Greatest Hits Live
10016;1996;Screamers
10017;2005;7 Seconds
10018;1994;Immortal Beloved
10019;2000;By Dawn's Early Light
10020;1972;Seeta Aur Geeta
10021;2002;Strange Relations
10022;2000;Chump Change
```

```
10023;2001;Clifford: Clifford Saves the Day! / Clifford's Fluffiest Friend Cleo
10024;1981;My Bloody Valentine
10025;1997;Inspector Morse 31: Death Is Now My Neighbour
10026;2004;Never Die Alone
10027;1962;Sesame Street: Elmo's World: The Street We Live On
10028;2002;Lilo and Stitch
.
.
.
```

In addition, the project includes a file `Short.txt` which includes 20 **unsorted** movies and will come handy later on.

The project includes multiple source files:

1. `Movie.py` file containing the Movie object (provided). The attributes ID and Year are stored as `int` and Title as `str`. Each Movie item can either represents an item of the list or a node of a BST.
2. `MovieList.py` file containing the list data structure with various methods (to complete).
3. `MovieBST.py` file containing the BST data structure with various methods (to complete).
4. Multiple app files: `app1.py`, `app2.py`, etc.: used for testing (provided).

All the functionalities of the application files (presented in details 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.py

The file is already completed. Here an example of execution:

```

 Welcome to Movie Application 1
===================================

Size of database 17770
Randomly search 10000 Movies, only first 10 are displayed
<<20612; 1960; Swiss Family Robinson>> found using binarySearch and sorted List
<<14944; 1994; The Three Tenors in Concert: Dodger Stadium>> found using binarySearch and sorted List
<<22938; 1978; Scared Straight!>> found using binarySearch and sorted List
<<11583; 2004; Moog>> found using binarySearch and sorted List
<<12374; 2000; The Duel>> found using binarySearch and sorted List
<<27560; 1984; Ghostbusters>> found using binarySearch and sorted List
```

```
<<13085; 2000; Little Nicky>> found using binarySearch and sorted List
<<21983; 1961; One>> found using binarySearch and sorted List
<<11901; 1978; Cheech & Chong’s Up in Smoke>> found using binarySearch and sorted List
<<26628; 1995; The Land Before Time III: The Time of the Great Giving>> found using binarySearch and s
Time: 49.114318000000004 ms to search
Database is now random
New database is saved in Movies.txt
```

The file is creating and initializing a MovieList data structure by loading the database “oMovies.txt” (ordered by ID). It is then searching 10,000 random ID movie numbers in the Database using Binary search returning the corresponding movie. The first ten movies found are displayed on screen, as well as the total time of the entire search. Finally, the database will be shuffled and the new randomized database is saved in the file “Movies.txt”.

What you need to implement in `MovieList.py` (Hint: look at project 1):

1. A constructor that reads the file and insert all items in a List. This list must be private.
2. The `getSize` method
3. The `binarySearch` method that accepts an ID number and return the corresponding movie. **Remark:** we will not take advantage of the particular and nice ordering of the ID numbers where the info could actually be obtained in O(1), but perform here the binarySearch explicitly.
4. The **shuffle** method
5. The **save** method.

app2.py

The file is already completed. Here an example of execution:

```
 Welcome to Movie Application 2
===================================

Size of database 17770
Randomly search 10000 Movies, only first 10 are displayed
<<20612; 1960;  Swiss Family Robinson>> found using BST
<<14944; 1994;  The Three Tenors in Concert: Dodger Stadium>> found using BST
<<22938; 1978;  Scared Straight!>> found using BST
<<11583; 2004;  Moog>> found using BST
<<12374; 2000;  The Duel>> found using BST
<<27560; 1984;  Ghostbusters>> found using BST
<<13085; 2000;  Little Nicky>> found using BST
<<21983; 1961;  One>> found using BST
<<11901; 1978;  Cheech & Chong’s Up in Smoke>> found using BST
<<26628; 1995;  The Land Before Time III: The Time of the Great Giving>> found using BST
Time: 47.154139000000015 ms to search
```

Here the code makes use of the randomized list of ID “Movies.txt” obtained with the previous app and create a Binary Search Tree (BST) (the random list will guarantee that the BST is fairly balanced).

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

1. The class Node (as seen in class) followed by the class MovieBST (in the same file).
2. A constructor that reads the file and insert all items in the BST.
3. The `insert` method. **Note:** the Tree is sorted by ID number. This method should be recursive.
4. The `getSize` method
5. The `search` method that accepts an ID number and return the corresponding movie. This method can be iterative or recursive.
6. Requirement: variables such as “the number of items” should be private. Auxiliary methods used in recursion must also be private.

app3.py

The file is already completed. Here an example of execution:

```
 Welcome to Movie Application 3
===================================

Size of database 20
Display in order 20 items by ID
10181; 2004; The Last Shot
11848; 1956; Samurai Trilogy 3: Duel at Ganryu Island
12439; 2005; WWE: Summerslam 2005
13491; 2001; National Geographic: Beyond the Movie: The Lord of the Rings
14647; 1995; American Yakuza
16969; 1995; Father Ted: Series 1-2
18361; 1939; Gulliver’s Travels
18478; 2002; Earth
19416; 1957; Peyton Place
19547; 1986; Troll / Troll 2: Double Feature
20957; 1961; Come September
22522; 1975; Rancho Deluxe
22996; 1997; Operation Delta Force 2
23245; 1934; The John Wayne Collection: Vol. 4: Lawless Frontier / Randy Rides Alone
24705; 1996; Tai Chi 2
24827; 1999; For Love of the Game
25389; 1993; The Bride with White Hair
25911; 1988; War and Remembrance: Vol. 1
26244; 1977; The Late Show
26394; 1995; Prime Suspect 4
```

```
The BSTree looks like:
                  26394(14)
            26244(6)
                  25911(13)
      25389(2)
                  24827(12)
            24705(5)
                  23245(11)
22996(0)
                              22522(46)
                        20957(22)
                  19547(10)
                        19416(21)
                                          18478(178)
                                    18361(88)
                              16969(43)
            14647(4)
      13491(1)
                  12439(8)
                        11848(17)
            10181(3)
```

Here, we are using the “Short.txt” database that contains only 20 unordered items. We create a BST, display “in-order” the list, show the tree structure (90 degree angle) using both ID and index level numbers.

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

1. The method `displayInOrder` and its “private” recursive auxiliary routine. All the movies in the BST will be listed when the latter is traversed in order.
2. The method `show` and and its “private” recursive auxiliary routine, that displays the 90 degree shifted tree structure with the ID number and index number in parenthesis. Make sure, you get the same output...

Remark: In addition to the traditional references ’left’, ’right’, your Node class should contain an ’index’ attribute that indicates the level-order position of the node in the Tree structure (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.

app4.py

The file is already completed. The goal is to operate transitions between list and BST representations. The different actions to consider are (step by step):

1. Create the BST for the main database

2. Ask user to input a Title keyword and extract from the Tree, a new `MovieList` database that contains a list of movies with the Title keyword.
3. Shuffle this new user MovieList and save it using different label for different keyword
4. Display the list
5. Create the corresponding subtree
6. Display in order the subtree
7. Show the subtree

Here an example of execution:

```
Welcome to Movie Application 4
===================================

Size of database 17770
Enter key word to search for: stars
Begin sublist extraction for word:stars using in-order traversal
Size of user list database 15
List shuffled and saved
15887; 2000;  Banner of the Stars II
25227; 1999;  A Paradise Under the Stars
27164; 1980;  Battle Beyond the Stars
19159; 2000;  Crest of the Stars
23538; 1989;  Beyond the Stars
16891; 2005;  The Man Show Boy / Household Hints from Adult Film Stars
26636; 1985;  My Lucky Stars
21869; 2004;  The Secret Lives of Adult Stars
21461; 2001;  Follow the Stars Home
11237; 1999;  The Stars of Star Wars
10235; 1996;  Unhook the Stars
23110; 2000;  The Book of Stars
18029; 2004;  Dora the Explorer: Catch the Stars
12964; 1982;  The Night of the Shooting Stars
17250; 2000;  Banner of the Stars

Size of user BST database 15
Display in order 15 items by ID
10235; 1996;   Unhook the Stars
11237; 1999;   The Stars of Star Wars
12964; 1982;   The Night of the Shooting Stars
15887; 2000;   Banner of the Stars II
16891; 2005;   The Man Show Boy / Household Hints from Adult Film Stars
17250; 2000;   Banner of the Stars
18029; 2004;   Dora the Explorer: Catch the Stars
19159; 2000;   Crest of the Stars
21461; 2001;   Follow the Stars Home
21869; 2004;   The Secret Lives of Adult Stars
```

```
23110; 2000;   The Book of Stars
23538; 1989;   Beyond the Stars
25227; 1999;   A Paradise Under the Stars
26636; 1985;   My Lucky Stars
27164; 1980;   Battle Beyond the Stars
The BSTree looks like:
            27164(6)
                  26636(13)
      25227(2)
                  23538(12)
                              23110(52)
                        21869(25)
                              21461(51)
            19159(5)
                        18029(24)
                              17250(49)
                  16891(11)
15887(0)
            12964(4)
      11237(1)
            10235(3)
```

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

1. The method `extractListInOrder`. It accepts the keyword as a String argument. It returns a `MovieList` object. **Hint:** Use an in-order traversal approach to visit all the nodes, you can use the 'split' field attribute on the movie title and search for the keyword. Lowercase and Uppercase words should not be considered different (e.g. stars, Stars, STARS, etc. are the same).

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

1. The method `display` that displays the List.

To be continued...

Submission/Grading Proposal

Only **two files** `MovieList.py` and `MovieBST.py` must be submitted on moodle. Only one submission by group of two. Write your name on top of both files. This project will be graded out of 100 points:

1. Your program should implement all basic functionality/Tasks and run correctly (90 pts).
2. Overall programming style with comments (including function header doc-string) (5 pts).
3. Pre-submission deadline for Preliminary (answering app1 to app3) (5pts). The program does not have to run correctly for pre-submission.