FLORIDA SOUTHERN COLLEGE

Analysis of Algorithms
CSC 3380

Fall 2018

Program 3: *FSC Book*
Assigned: Wednesday, September 26, 2018
Due: Friday, October 5, 2018 by 11:55 PM

Purpose:

1. Learn to implement the functionality of binary search trees while also getting more and more practice with linked lists.

Read Carefully:

- This program is worth 5% of your final grade.

- **WARNING**: This is an individual assignment; you must solve it by yourself. Please review the definition of cheating in the syllabus, the FSC Honor Code, and the serious consequences of those found cheating.
    - **The FSC Honor Code pledge MUST be written as a comment at the top of your program**.

- The deadline for this project is by **Friday, October 5, 2018 by 11:55 PM.**
    - **Note:** once the clock becomes 11:55 PM, the submission will be closed! Therefore, in reality, you must submit by 11:54 and 59 seconds.

- LATE SUBMISSION: you are allowed to make a late submission according to the rules defined in the syllabus. Please see course syllabus for more information.

- **Portal Submission**:
    - This assignment must be submitted online via Portal.
    - If your file is empty or you upload wrong the file, it will be solely your responsibility, and your grade will be **zero**.
    - Your java files should all be zipped up. Your zip file should be named as follows:
        - `CSC3380_YourFSCIDnumber_Program3.zip`
        - Example: `CSC3380_2351345_Program3.zip`

Program 3: FSCbook

Objective

Learn to implement binary search trees and to combine that functionality with linked lists.

The Problem

After working hard at Program 2, you feel confident enough to do anything! You decide to create a database manager for a new FSC social networking site, www.fscbook.com. Your database should be able to store unlimited number of students (users) along with their basic personal information (ID, first name, last name, age, department, etc.). Additionally, each user should be able to maintain a list of all his FSC friends.

So how do you implement this? You know that you must search the database quickly based on their name or ID. And with your extensive knowledge of data structures and Big O running times, you know that a simple array of users or a linked list of users would be an EPIC FAIL, as this would result in an O(n) search. Therefore, you realize that the best way, based on your CSC 2281 knowledge, is to implement this as a binary search tree. Searching, insertion, and deletion should all take approximately O(logn), which is MUCH faster.

You will store users in FSCbook based on their student ID numbers, and it will be guaranteed that this ordering of students (by ID) will be the exact same as the alphabetical ordering by last name and then first name. So Bob Smith is guaranteed to have a "smaller" ID than Joe Smith, because his first name is "smaller" than Joe. What is the benefit of this? This means that you do not need to insert into the BST by names. Instead, you can just insert by ID, which is guaranteed to be also in order by name.

Your FSCbook program will read in a serious of commands from a file, such as ADD, DELETE, FRIEND, UNFRIEND, etc. You will need to process the commands and print the appropriate information to the output file.

Input/Output Specifications

The input file will be called "`FSCbook.in`". The first line of the file will be an integer, *k*, indicating the number of commands that will be run on the database. The following *k* lines will each be a single command, **with the relevant data on the same line**, separated by spaces.

The commands you will have to implement are as follows:

`ADD` – Creates a new FSCbook account (a new node). The command will be followed by the following information: ID, a non-negative int variable, which you will use to sort the accounts by, a first name and last name, each no longer than 20 characters, and the department to which they belong (CS, IS, or IT).

- Yes, we assume we have three departments in the future for this problem.

`FINDNAME` – This command will be followed by a name, first then last. You must search the database to find this name, and then print out some information about this person. It should print their name, department, and the number of Friends they have. If there is not a person by that name in the database, it should print an error message saying so.

`FINDID` – Similar to `FINDNAME`, this will find a person, based on their ID number instead. The output should be the same. It should print their name, age, and the number of Friends they have. If there is not a person by that name in the database, it should print an error message saying so.

`FRIEND` – This will be followed by two names, first and then last for both. When you see this command, this means that the two people are now Friends, and thus you should add both people's IDs to each others' Friend-lists. All connections on FSCbook are always two-way – e.g. if John is in Bob's Friend-list, then Bob is also in John's. If either person in the `FRIEND` command is not in the database, or if they are already Friends, an error message should be displayed (see sample output below).

`UNFRIEND` – This is the opposite of `FRIEND`. Like `FRIEND`, it will be followed by two names. You will have to find both names, and remove their IDs from each others' Friend-lists. Again, if either person is not in the database, or they are not actually Friends, an error message should be displayed.

`DELETE` – This command, followed by a name, means you must delete this user's account from the database. Before you do so, however, you must remove them from any other users' Friend-lists. Thus, for the user you are deleting, you will need to loop through their own Friend-list, and find all of their Friends by searching for their IDs, and run your `UNFRIEND` command to remove them from each others' lists. If the person you are trying to delete does not actually have an account already, an error message should display.

`PRINTFRIENDS` – This command, followed by a name, first then last, prints the member's name, the number of friends they have, and then it prints an alphabetical list of their friends and which department they are in.

`PRINTMEMBERS` – This command prints all members in alphabetical order (last name, and then first name), along with printing basic information about these members (see output below).

Your program must output to a file, called "**`FSCbook.out`**". **You must follow the program specifications exactly**. You will lose points for formatting errors and spelling.

See sample input and output files for examples.

*****WARNING*****

Your program MUST adhere to this EXACT format (spacing capitalization, use of colons, periods, punctuation, etc). Your program will be graded with very large input files, resulting in very large output files. As such, we will use text comparison programs to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, we will will be forced to go to text editing of your program in order to give you an accurate grade.

Helpful Suggestions / Comments

The binary search tree code and the linked list code on Portal use integers throughout the examples (int data). My intention is for you to basically, with only minor changes, reuse that code…meaning, we want this to be easy for you. So for just about ALL of the commands, whenever you read in a first and last name, the FIRST thing you should do is call a method (that you will make) called getID. This method sends over the root of the tree, the first and last name in question, and then returns the ID of that user. So now, at that point, you have an int ID and can basically use the code on Portal without much modification. Of course, making the method mentioned above, getID, will require you to do a binary search on the tree based on string compare of last and first names. So you will need to write that yourself (by modifying the "findNode" method in the code on the website). But again, once done, you have an int ID and can reuse a lot of the code on the website.

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications listed on this write-up.
2) Your algorithmic design.
3) Correctness.
4) **Use of Binary Search Trees with Linked Lists. Since the purpose of this assignment is to teach you to Binary Search Trees, you will not receive credit if you don't use them.**
5) The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly

commented and spaced and works perfectly, you could earn as low as 85-90% on it.)

6) Compatibility to the newest version of NetBeans and Java. (If your program does not compile in NetBeans, you will get a sizable deduction from your grade.)
7) Your output MUST follow to the EXACT output format shown in the sample output file.
8) Your program should include a header comment with the following information: your name, course number, section number, assignment title, date, and FSC honor code.
9) You should include comments throughout your code, when appropriate.

Deliverables

You should submit a zip file with **FIVE** files inside:

1. `FSCstudent.java`
   Class to create object of type FSCstudent
2. `FSCbookBST.java`
   Class to create objects of type FSCbookBST. You will only make ONE object of this class. This is the class for the BST. The code should be very similar to the code on the website.
3. `FSCfriends.java`
   Class to create objects of type FSCfriends. This is the Linked-List class. Each student will have a reference to their list of Friends. And this reference will point to a linked-list object (an object of type FSCfriends).
4. `FSCfriend.java`
   Class to create objects of type FSCfriend. This class is to make nodes of the linked list mentioned above (FSCfriends).
5. `FSCbook.java`
   This is your main program.

NOTE: your name, ID, section and EMAIL should be included as comments in all files!

Here are the UML Diagrams for the required classes:

FSCstudent
Data Members private int ID private String firstName private String lastName private String department private int numFriends private FSCfriends myFriends private FSCstudent right private FSCstudent left
Operations/Methods CONSTRUCTOR (one or more constructors as needed) ALL obvious getter/setter methods

DETAILS:

- `FSCstudent` is used to make objects of the BST.
- `ID, firstName, lastName, and department` should be obvious
- `numFriends` is the number of friends that this student has
- `myFriends` is a reference to an object of type FSCfriends (a linked-list). This linked-list will store all the friends of this student.
- left and right are the left and right pointers of this node (for the BST).

FSCfriends
Data Members private FSCfriend head;
Operations/Methods CONSTRUCTOR (one or more constructors as needed) ALL obvious getter/setter methods All required methods for a Linked-List class

DETAILS:

- This is the Linked-List class. It should be very similar to the linked-list class on the website or used for Program 2.
- All nodes of this class are of type FSCfriend, which is shown below.

FSCfriend
Data Members private int ID; private FSCfriend next;
Operations/Methods CONSTRUCTOR (one or more constructors as needed) ALL obvious getter/setter methods

DETAILS:

- This is the class to make nodes of the linked-list of friends.
- Each FSCstudent has a variable called myFriends. This variable is

FSCbookBST
Data Members private FSCstudent root
Operations/Methods CONSTRUCTOR (one or more constructors as needed) ALL obvious getter/setter methods All required methods for a BST

DETAILS:

- This is the BST class used to create one object of type FSCbookBST
- This class should be VERY similar to the BST class shown on the course website.