Programming Assignment #3

Managing and multiple users with a Binary Search Tree

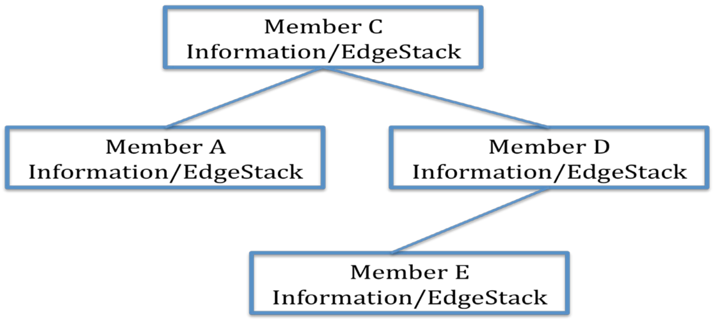
Due date: March 27, Tuesday 2:00PM
URL: http://www.cs.colostate.edu/~cs200

A. Objective

In this assignment, you will build methods to manage multiple users with a binary search tree. Your system should allow the following capabilities: inserting a new member, deleting a member, and retrieving information about a member.

B. Description of Task: maintaining a BST for the members

Members are searched and sorted using a binary search tree. The **user's ID** is the key that will be used to sort and search. Your software should sort the user IDs lexicographically. To build a BST, start with the algorithm as outlined in the textbook (Prichard).



a. Create classes **MemberNode** and **MemberTree**.

b. Implement methods of the **MemberNode** Class.

The **MemberNode** class contains *information about a member* and *references* to its children nodes, and should provide the following public methods:

```
Member getItem();
MemberNode getLeftChild();
MemberNode getRightChild();

void setItem(Member _thisItem);
void setLeftChild(MemberNode _leftChild); // _leftChild can be null
void setRightChild(MemberNode _rightChild); // _rightChild can be null
```

c. Implement methods of the **MemberTree** Class
The class **MemberTree** should provide the following public methods:

```
boolean isEmpty();
MemberNode setRoot(MemberNode _rootNode);
MemberNode getRoot();
MemberNode searchMember(String userID MemberNode startnode);
void insertMember(Member _newMember);
MemberNode removeMember(String userID MemberNode startnode);
MemberNode retrieveMemberInfo(String userID, MemberNode startnode);
LinkedList<Member> getMemberTreeInOrder(MemberNode startnode);
LinkedList<Member> getMemberTreePreOrder(MemebrNode startnode);
LinkedList<Member> getMemberTreePostOrder(MemberNode startnode);
```

The **searchMember** method searches a **MemberNode** object that has the same **userID**. The searching process will start from the **startnode**. If it cannot locate the **memberNode**, it will return null.

The **deleteMember** method deletes a **MemberNode** object that has the same **userID**. This method will start to search the item to delete from **startnode**. If your **deleteMember** method was successfully processed, your method should return the **MemberNode** you deleted. Otherwise, it should return null.

The **retriveMemberInfo**() method retrieves a **MemberNode** object that has the same **userID** .This method will start to search the item to retrieve from **startnode**. If the member does not exist, return null.

Provide a public method, **getMemberTreeInOrder**(). This method should return a LinkedList of members from your member tree following the in-order tree traversal algorithm. The traversal process starts from the **startnode**. Similarly, methods **getMemberTreePreOrder** and **getMemberTreePostOrder** return LinkedLists of members following the Pre-Order and Post-Order tree traversal algorithms. The traversal process starts from the **startnode.**

You may add one or more methods in this class.

C. Parsing Information of multiple users

Your example file contains 15 members and their edges. Modify your current **InformationParser** to read multiple users' information. Please note that some users might not have any edges yet.

```
public static MemberTree parseMultipleInfo(String input)
```

[Note] An input file contains information of multiple users.

D. Using Skeleton Files

You can modify **your classes** from PA1 and PA2. Additional skeletons are provided. Please note that using these skeleton files is **OPTIONAL**. You are encouraged to build your software based on your own design. PA3.java will be provided but please do not change its **`main`**() method. Example data will also be provided.

D. Requirements/Test Cases

The test cases listed below are provided to assist you in verifying the correctness of your software. You are also required to test your software with different test cases that you will build yourself. Actual grading will be done by test cases using the same commands; however, the values that the specified input arguments take will be different.

(1) Test case 1

Objective:
Build a tree from the provided example dataset, and print your tree using the **In-order** traversal algorithm. In this test case, print only userID and separate userIDs with a space. For the test case 1,2 and 3, use the command line syntax,

```
java PA3 [input_file] build_tree [1|2|3]
```

Here, the last argument “1” is for In-order, “2” is for Pre-order, and “3” s for Post-order.

Command: `java PA3 PA3_input.txt build_tree 1`
Output:
```
1372William ar3090 bettyfriedan dh3136 eh0721 epyle jambs lf8203
mtwain nhawth remerson sinclair theodore1 theodore789 willr
```

(2) Test case 2

Objective:
Build a tree from the provided example dataset, and print out your tree using the **Pre-order** traversal algorithm. In this test case, print only userID and separate userIDs with a space.

Command: `java PA3 PA3_input.txt build_tree 2`
Output:
```
eh0721 bettyfriedan 1372William ar3090 dh3136 jambs epyle
remerson nhawth mtwain lf8203 sinclair theodore789 theodore1 willr
```

(3) Test case 3

Objective:
Build a tree from the provided example dataset, and print out your tree using the **Post-order** traversal algorithm.

Command: `java PA3 PA3_input.txt build_tree 3`

Output: `ar3090 1372William dh3136 bettyfriedan epyle lf8203 mtwain nhawth theodore1 willr theodore789 sinclair remerson jambs eh0721`

(4) Test case 4

Objective: After you build a tree from the example file, add a member to your tree without any edge, and print out your tree using the **In-order** traversal algorithm (userID only).

Command: `java PA3 PA3_input.txt add_member be1376 Black Elk`
Output: `1372William ar3090 be1376 bettyfriedan dh3136 eh0721 epyle jambs lf8203 mtwain nhawth remerson sinclair theodore1 theodore789 willr`

(5) Test case 5

Objective: After you build a tree from the example file, delete a member from your tree and print out your tree using the In-order traversal algorithm (userID only). The deleted member is specified with its userID.

Command: `java PA3 PA3_input.txt remove_member bettyfriedan`
Output: `1372William ar3090 dh3136 eh0721 epyle jambs lf8203 mtwain nhawth remerson sinclair theodore1 theodore789 willr`

(6) Test case 6

Objective: After you build a tree from the example file, retrieve a member's information from your tree and print out the first name and last name.

Command: `java PA3 PA3_input.txt retrieve_memberInfo jambs`
Output: `James Baldwin`

PA3 file, input file, and submission instructions will be posted on the class web site along with this document. **DO NOT MODIFY the main() of the PA3.java that has been provided to you.**

E. Grading
This assignment will account for 5% of your final grade. The grading itself will be done on a 50 point scale.

G. Late Policy
Please check the late policy available from the course web page.