COMPUTER SCIENCE 21A (SPRING TERM, 2021)
DATA STRUCTURES

PROGRAMMING ASSIGNMENT 2

Chess Leaderboards with AVL trees

Background:

You have been hired by an international chess society to update their scoreboard system. The chess society is a membership organization, the members play chess against one another and the society keeps track of the outcomes. Each member has an Elo rating (a score based on their prior performance in chess matches), and the members are ranked based on their Elo ratings. Whenever there is a chess match between two society members, the winner's score increases and the loser's score decreases, and the rankings may also change. The society members are very competitive, so they frequently check their own ranking and Elo rating and the rankings and ratings of their opponents.

Your predecessor, who dropped out of data structures class after the first assignment, wrote the old scoreboard code using a linked list with insertion sort. This wasn't a problem when the society was small, but the society is growing, and members are complaining that it takes too long to look up the rankings and ratings of individual players. Of course, we know that this is because searching for a particular member in a linked list takes O(n) time. You have been instructed by the chess society that these lookups need to be worst case O(log(n)) time or you will be fired. To accomplish this task, you will need to build a self-balancing AVL tree, augmented to allow fast computation of player ranking.

Basic Idea:

Each player has a name, ID number, and Elo score. You need to allow players to look up their Elo score or their rank in the league using only their player ID number. To do this, you will need to make two trees to store players - one tree where the players are sorted based on ID, and another where they are sorted based on Elo. In order to find the ranking of the player with ID number 100, you would first search for player 100 in the ID tree. Inside the node for player 100 is the Elo score for player 100. Use this score to query the Elo tree to find the actual ranking of player 100 based on Elo score.

Computing the Rank:

The rank of a player is one more than the number of players with a higher score (the best player is ranked 1). To compute the rank, each node will have to store the number of nodes in the right (bigger) subtree. Using this property, the rank can be computed recursively. The rank of the root is one more than the number of nodes in the right subtree. If the Elo score is greater than the root, the rank is equal to the rank in the right subtree, computed recursively. If the Elo score is less than the root, the rank is the rank in the left subtree (computed recursively) plus the number of nodes in the right subtree, plus one for the root. You will need to add a line of code to the end of the left and right rotation procedures to maintain the count of nodes in the right subtree.

Writing Your AVL tree:

As mentioned before, in order to find the rank of a player using only their player ID, you will need two trees. You should not write two different tree node classes. Instead, in order to reuse code, you should write one tree node class that can do everything you need for both trees. Each node should store the value that it is sorted by in a separate field apart from where it stores its Player object. Your insert method should take as arguments a Player object to insert and a double value to sort the player by. Then, if you want to insert a new Player p with name "Bill" with id 0 and Elo 100.0, you can do **eloTree.insert(p,p.getElo())** and **idTree.insert(p,p.getID())**.

Requirements:

Your job is to fill in the methods of `AVLPlayerNode.java`, your AVL tree node class. You have been provided with the code for the `Player` class, which is described below. You have also been provided with the interactive part of the code, including the main method, in ScoreKeeper.java. We have commented out the parts of that file which use the `AVLPlayerNode` methods you are responsible for writing. As you fill in the methods for `AVLPlayerNode`, you can uncomment the relevant sections of `ScoreKeeper.java`, **but you should not add any new code to this file.**

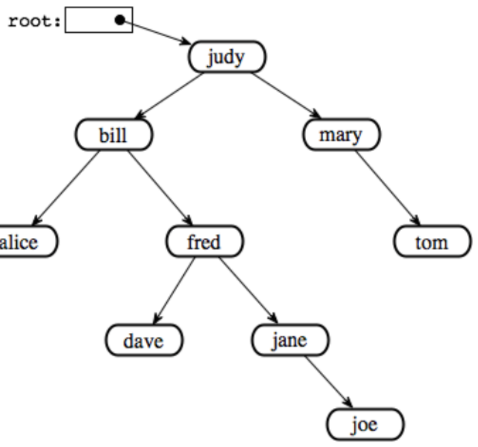
Your tree should do self-balancing AVL insertions. Your tree must also allow for deletions, but we have made self-balancing deletions extra credit. For full credit, you need only implement a delete method which returns the tree in a valid state. Be warned that these unbalanced deletions can break the AVL property of your tree, so once you delete from a tree there is no guarantee that insertions will work properly. If you are not doing the extra credit, you should not expect insertions to work properly after deletions when testing your code.

Your tree will need a method `getPlayer(double value)` which returns the Player object stored in the node with this.value=value.

As mentioned before, your tree node class will also need a `getRank(double value)` method that returns the rank of the player at the node with this.value=value.

Your main method will need to be able to print the entire scoreboard in order from highest Elo to lowest, so your tree node class should have a method that returns the full board as a String.

Lastly, your tree node class should have a method that returns a String of the tree of names, using parentheses to separate subtrees. For example, for the tree below, the printout would be: **(((alice)bill((dave)fred(jane(joe))))judy(mary(tom)))**



Extra Credit:

You can receive extra credit for this assignment if you implement AVL self-balancing deletions for your tree. If you choose this path, your main loop should allow the user to delete a player from the league (players should be deleted from both trees). It should also allow the user to log a match between two players in the league. In order to log the match. You should ask the user for the id numbers of the two players who are going to play a match. You should then delete the players from both trees and ask the user which player won. The Player class has a method called `logVictory(Player defeated)` which takes as a parameter the defeated player and updates the scores of both players to reflect the outcome of the game. There is a similar method called `stalemate(Player worthyAdversary)` which also updates the scores of both players. If a game is a tie, you should pick one player to call the stalemate method on the other, but do not do two calls of the stalemate method to log a single outcome. Once the players have updated their scores, you can insert them back into the trees.

Code:

`ScoreKeeper.java` works in the following way: ScoreKeeper.java - Main Method

1. First, it asks the user how many players need to be added to the tree.
2. Next, it has the user enter the player information for the players one at a time. Each player has three fields that need to be stored
    a. String name
    b. int playerID
    c. double Elo rating
3. Next, it gives the user a prompt offering them the following options (repeat until they choose exit):
    a. add/delete a player (enter A for add or D for delete, if you don't do the extra credit, make sure you always do all adds before any deletes)
    b. Check the ranking or Elo rating of a player (requires player id), (enter R for

rank or E for Elo)

c. Print the full scoreboard in decreasing order by Elo
d. Log the outcome of a match between two players (requires both players' ID's) (enter M for Match, extra credit, same as delete, give option and print "Unsupported operation" if you did not implement delete)
e. Print the Elo scoreboard tree in (in the parentheses format shown below) (P for print)
f. Exit (X for exit)

AVLPlayerNode.java

This will be the class for the nodes of your AVL tree. Each node will have a left child, a right child, and a Player object, a double (the value that the node is sorted based on), balance factor for AVL balancing, number of nodes in the right subtree. You can add other fields you think will be helpful.

You will also need methods for:

- `AVLinsert(Player p, double value)`
- `AVLPlayerNode delete(Player p, double value)`
- `Player getPlayer(double value)`
- `int getRank(double value)`
- *a method for creating the tree string in parentheses form* (as described earlier)
- *a method for generating the scoreboard in order of Elo*
- Internally, you will want to write methods for *left* and *right rotations*. Make sure you maintain the balance factor and number of nodes in the right subtree after each time you modify the tree structure.
- Feel free to add other methods you think will be helpful. All insert methods should be self-balancing, and for those doing the extra credit portion, your delete code should also be self-balancing.

Player.java

This class has been provided for you, and it has the following public methods for you to use:

1. `Player(String name, int playerID, double Elo)` - constructor
2. `String getName()`
3. `int getID()`
4. `double getElo()`
5. `void logVictory(Player defeatedOpponent)` - when a game is logged, the winning player should call this method with the losing player as the parameter, and both of their Elo scores will be updated to reflect the outcome.
6. `void stalemate(Player opponent)` - make sure you only call this on one

player, the other player gets updated automatically.

AVLUnitTesting.java

You will need to write JUnit tests for your AVLPlayerNode class. The tests you write should test all functionalities of the AVL tree thoroughly. The quality of your tests will determine 20% of your grade for the first part of the programming assignment, so test your code well! If all of your tests pass, you should feel confident that you have a flawless data structure.

Sample IO for a successful implementation:

```
Hello, I'm ScoreKeeper. Let's start a new scoreboard, shall we? How
many players do you need to add to the scoreboard? (must be at least 3)
3
Thank you. You may now enter in 3 players, one at a time
Please enter the name of a player you wish to add
Mike
Please enter the id number for Mike
1
Please enter the current ELO rating for Mike
2400
Please enter the name of a player you wish to add
Sandra
Please enter the id number for Sandra
2
Please enter the current ELO rating for Sandra
2300
Please enter the name of a player you wish to add
Eric
Please enter the id number for Eric
3
Please enter the current ELO rating for Eric
2200
What would you like to do next?
A/D to Add/Delete a player to the scoreboard
R/E to check the Rank or Elo rating of a player (requires player ID)
L to list the entire leader board in order of Elo (decreasing order)
M to log the outcome of a chess Match between two players
P to Print the elo tree in parentheses format
X to eXit
A
Please enter the name of a player you wish to add
Fred
Please enter the id number for Fred
4
Please enter the current ELO rating for Fred
2100
What would you like to do next?
A/D to Add/Delete a player to the scoreboard
R/E to check the Rank or Elo rating of a player (requires player ID)
L to list the entire leader board in order of Elo (decreasing order)
```

```
M to log the outcome of a chess Match between two players
P to Print the elo tree in parentheses format
X to eXit
P
ELO tree: (((Fred)Eric)Sandra(Mike))
ID tree: ((Mike)Sandra(Eric(Fred)))
What would you like to do next?
A/D to Add/Delete a player to the scoreboard
R/E to check the Rank or Elo rating of a player (requires player ID)
L to list the entire leader board in order of Elo (decreasing order)
M to log the outcome of a chess Match between two players
P to Print the elo tree in parentheses format
X to eXit
R
Please enter the ID number of the player whose rank you wish to check
1
ID: 1 NAME: Mike RANK: 1
What would you like to do next?
A/D to Add/Delete a player to the scoreboard
R/E to check the Rank or Elo rating of a player (requires player ID)
L to list the entire leader board in order of Elo (decreasing order)
M to log the outcome of a chess Match between two players
P to Print the elo tree in parentheses format
X to eXit
E
Please enter the ID number of the player whose ELO you wish to check
2
ID: 2 NAME: Sandra ELO: 2300.000000
What would you like to do next?
A/D to Add/Delete a player to the scoreboard
R/E to check the Rank or Elo rating of a player (requires player ID)
L to list the entire leader board in order of Elo (decreasing order)
M to log the outcome of a chess Match between two players
P to Print the elo tree in parentheses format
X to eXit
L
NAME          ID  SCORE
Mike           1 2400.00
Sandra         2 2300.00
Eric           3 2200.00
Fred           4 2100.00
```


---------------------------EXTRA-CREDIT---------------------------------
----

```
What would you like to do next?
A/D to Add/Delete a player to the scoreboard
R/E to check the Rank or Elo rating of a player (requires player ID)
L to list the entire leader board in order of Elo (decreasing order)
M to log the outcome of a chess Match between two players
P to Print the elo tree in parentheses format
X to eXit
```

```
D
Please enter the ID number of the player you wish to remove from the
system
3
What would you like to do next?
A/D to Add/Delete a player to the scoreboard
R/E to check the Rank or Elo rating of a player (requires player ID)
L to list the entire leader board in order of Elo (decreasing order)
M to log the outcome of a chess Match between two players
P to Print the elo tree in parentheses format
X to eXit
M
Please enter the ID number of the first player in the match
1
Please enter the ID number of the second player in the match
4
Please enter the outcome of the match
1 if the first player (Mike) was the winner
2 if the second player (Fred) was the winner
0 if the match was a draw
1
What would you like to do next?
A/D to Add/Delete a player to the scoreboard
R/E to check the Rank or Elo rating of a player (requires player ID)
L to list the entire leader board in order of Elo (decreasing order)
M to log the outcome of a chess Match between two players
P to Print the elo tree in parentheses format
X to eXit
L
NAME          ID  SCORE
Mike           1 2404.83
Sandra         2 2300.00
Fred           4 2095.17

What would you like to do next?
A/D to Add/Delete a player to the scoreboard
R/E to check the Rank or Elo rating of a player (requires player ID)
L to list the entire leader board in order of Elo (decreasing order)
M to log the outcome of a chess Match between two players
P to Print the elo tree in parentheses format
X to eXit
X
```

Submission Details

- For every non-test method you write, you must provide a **proper JavaDoc** comment using @param, @return, etc.
- In addition to regular JavaDoc information, please include each **non-test method's runtime.**
- At the top of **every file** that you submit please leave a comment with the following format. Make sure to include **each of these 6 items.**

```
/**
 * <A description of the class>
 * Known Bugs: <Explanation of known bugs or "None">
 *
 * @author Firstname Lastname
 * <your Brandeis email>
 * <Month Date, Year>
 * COSI 21A PA2
 */

>Start of your class here<
```

- Use the provided skeleton Eclipse project.
- Submit your assignment via Latte as a .zip file by the due date. It should contain your solution in an Eclipse project.
- Your Eclipse project should be named **LastnameFirstname-PA2**. To rename a project in Eclipse, right-click on it and choose Refactor > Rename.
- Name your .zip file **LastnameFirstname-PA2.zip**
- Late submissions will not receive credit.