CS1027

ASSIGNMENT 4

Computer Science Fundamentals II

This assignment is due on Friday, July 23, 2021 by 11:55pm. See the bottom for submission details.

Learning Outcomes

To gain experience with

- Working with trees and their traversals
- Loading data from external files
- Algorithm design

Introduction

The NHL (National Hockey League) is the largest hockey league in the world with 31 (soon to be 32!) teams across Canada and the United States. Each year, the NHL has a regular season in which each team plays an equal number of games (normally 82, but with the pandemic this year, they had it shortened to 56 games). After the regular season, there are playoffs in which the top 16 teams compete in a structure bracket over 4 rounds in the hopes of winning the Stanley Cup! Note that the structure was slightly changed this year due to border closures, but they still did complete the full 4 rounds of the playoffs. In each round, the teams play in best-of-7 series which means that a team must defeat their opponent 4 times to move on to the next round, and the losing team is eliminated.

In this assignment, we are building a binary tree to represent the playoff structure from this year's NHL playoffs. The root of the tree will contain the team who won the Stanley Cup, and the other series from the playoffs will be represented in the various nodes below, each rounds being represented as a level of the tree. As you might begin to envision, we have to build the tree from the bottom up rather than the top down. We will still have a tree with a root at the top and the leaf (pun not intended) nodes at the bottom, but we have to leave the upper nodes empty initially and then later fill them in once the series from the lower layers are completed.

Figures 1-4 show an example of an 8-team playoff structure (half the number of teams as the actual playoffs). Examine these figures and the captions below each one to better understand this concept. The main idea is that we start with the bottom row nodes as round 1 of the playoffs, and then we can backfill each level of nodes higher in the tree as the previous round ends.

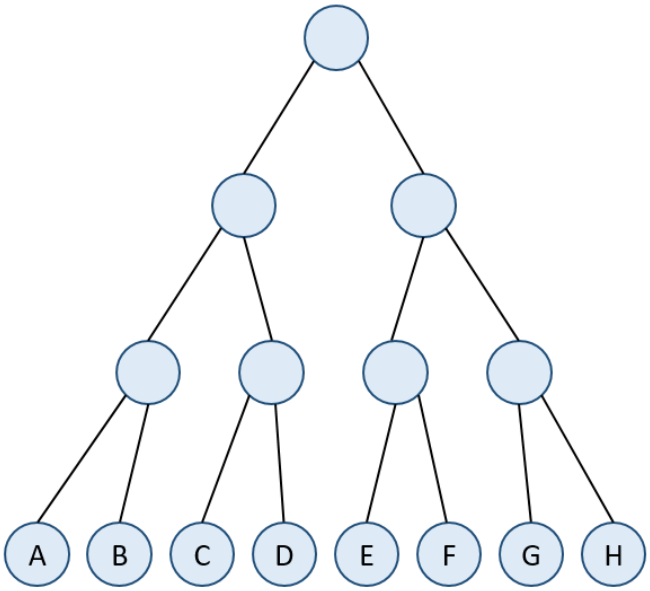

Figure 1. A tree showing an example of an 8-team playoff structure. The teams are labelled as A, B, C, …, H. Team A plays against Team B; C plays D; E plays F; and G plays H.

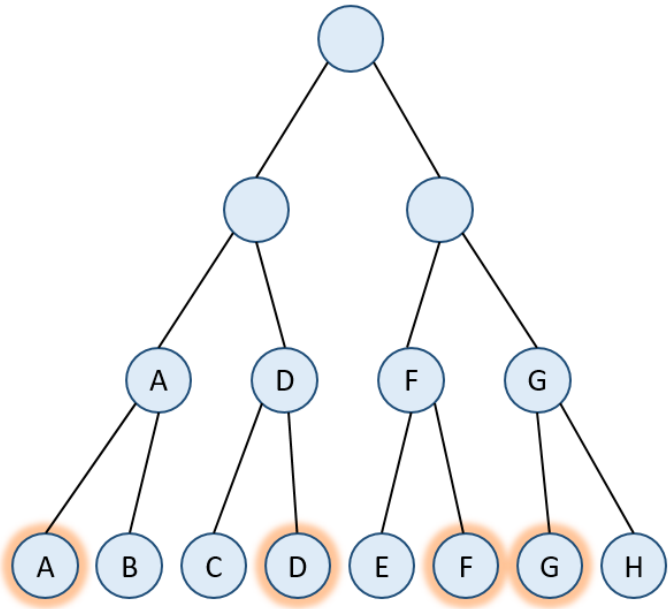

Figure 2. The tree showing the next round of the playoffs following Figure 1. The nodes with an orange glow are the teams who won their series in this example. A beat B; D beat C; F beat E; and G beat H. Thus, the teams A, D, F, and G were filled in to the corresponding nodes above the bottom row.

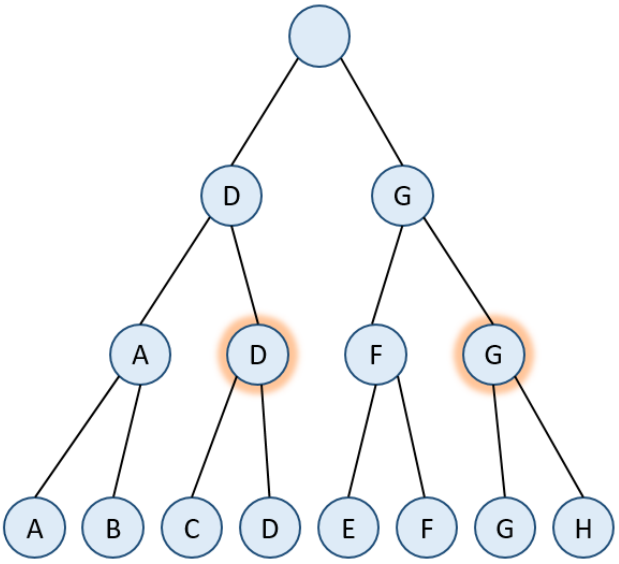

Figure 3. The tree showing the next round of the example playoffs following the previous figures. In this example, D defeated A; and G defeated F in the semi-finals.

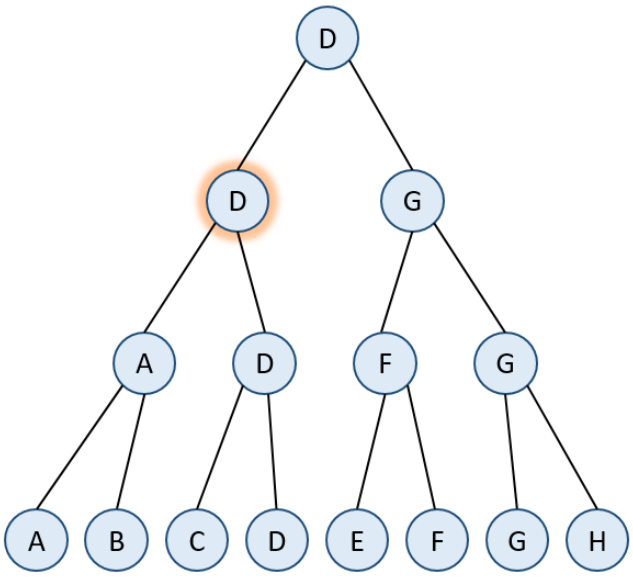

Figure 4. The final tree in this example playoff sequence. Team D conquered Team G in the finals to win the championship!

Provided files

The following is a list of Java files provided to you for this assignment. Please do not alter these files in any way.

- ArrayIterator.java
- ArrayList.java
- ArrayUnorderedList.java
- BinaryTreeADT.java
- BinaryTreeNode.java
- ElementNotFoundException.java
- EmptyCollectionException.java
- HockeySeries.java
- LinearNode.java
- LinkedBinaryTree.java
- LinkedQueue.java
- ListADT.java
- MyFileReader.java
- QueueADT.java
- UnorderedListADT.java
- TestTreeBuilder.java
- TestPlayoffs.java

Classes to implement

For this assignment, you must implement 2 Java classes: *TreeBuilder* and *Playoffs*. Follow the guidelines for each one below.

In both of these classes, you can implement more private (helper) methods, if you want to, but you may **not** implement more public methods. You may **not** add instance variables other than the ones specified below nor change the variable types or accessibility (i.e. making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

TreeBuilder.java

This class is used to create the binary trees for the program. They must be created using a queue-based approach, very similar to the level-order traversal that uses queues, but here the tree is being built rather than traversed. This class must use generics to build trees of any type.

The class must have the following *private* variable:

- tree (LinkedBinaryTree<T>)

The class must have the following *public* methods:

- public TreeBuilder(T[ ]) [constructor]
  - Take in a T array input parameter which contains the set of values with which to insert in the new tree's nodes. Note that some elements of the array may be null to represent that there is no tree node in that position.
  - Within this method, use two queues to build the tree using a level-order-esque approach. The summarized algorithm steps are listed below.
- public LinkedBinaryTree<T> getTree()
  - Return the tree

Algorithm for building trees

initialize dataQueue with all elements (from T[ ] parameter) to be added in the nodes in order

initialize parentQueue as empty queue

create the tree's root node with the first element of the data queue

enqueue the root node on parentQueue

while dataQueue has elements

    a = dequeue from dataQueue

    b = dequeue from dataQueue

    parent = dequeue from parentQueue

    if a is not null, add node with a as left child of parent and enqueue on parentQueue

    if b is not null, add node with b as right child of parent and enqueue on parentQueue

initialize the LinkedBinaryTree instance variable with the root node from above

Playoffs.java

This class represents the NHL Playoffs and contains several methods for updating the playoff tree and the standings table. Some methods are provided to you and some are left for you to complete.

The class must have the following *private* variables:

- tree (LinkedBinaryTree<String>)
- standings (HockeySeries[ ])

The class must have the following *public* methods:

- public Playoffs() [constructor]
  - Create a String array with 31 slots (there will be 31 nodes in the playoff tree to represent the 4 rounds of playoffs) for the data to go into the tree.
    - Since the top several layers of the tree are unknown at this time, use "TBD #" (where # is the corresponding index number) for the first 15 spots (i.e. TBD 0, TBD 1, TBD 2, …, TBD 14)
    - From index 15 to 30 (the end of the array), insert the team that is loaded in from teams.txt (hint: use the provided MyFileReader to simplify this file input step)
  - Initialize the standings table with 15 slots (there will be exactly 15 series in the playoffs – 8 in round 1, 4 in round 2, 2 in round 3, and 1 in round 4).
    - Take every 2 teams from the array you just made and group them together into a series. The order is important here so ensure you take consecutive pairs the way they were written in the teams.txt file).
    - For each pair, create a HockeySeries object with those two team names and 0 for each of their default number of wins.
  - Use the TreeBuilder to build the binary tree from the String array you created. Initialize this class' tree by using the TreeBuilder's getTree() method.
- public LinkedBinaryTree<String> getTree()
  - Return the tree
- public HockeySeries[] getStandings()
  - Return the standings
- public String updateStandings(String,String,int,int)
  - Take in 2 Strings representing team names and 2 ints representing the number of wins by which to update the standings for these teams (i.e. A,B,1,0 would mean that A beat B and thus A should get 1 more win and B should get 0 more wins in the standings).
  - Search the standings array to look for the series that corresponds to the given team names. Once you find the correct HockeySeries object for those teams, you can use the incrementWins() method on that object to update both teams' wins (typically one will be a 1 and one will be a 0).

  - If a team reaches 4 wins, that means they won the series, so return that winning team's name from this method. If neither has reached 4 wins yet, or the series could not be found in the array, return null instead.
- public void updateRound(int)
  - Load in the scores text file that corresponds to round i (1, 2, 3, or 4). Each line in the files represents one game and is formatted as: teamA,teamB,scoreA,scoreB
  - You may want to use the String's split() method to convert the String of one line into an array of 4 Strings based on the comma delimiter.
  - Call the updateStandings() method (described above) to update the number of wins in the series. Note that you have to send in both team names and both standings updates, so one team (the one that wins this game) will have a 1 sent in to updateStandings() and the other (losing) team will have 0 sent in.
  - If a team reaches 4 wins, the updateStandings() method will return that team's name. In this method, check if the returned String is a valid team name and if so, add it in to the parent node of that series (refer back to Figures 1-4 near the top to refresh yourself on what this means). For example, when Avalanche beats Blues for the 4th time, the node that is the parent of the Avalanche node and the Blues node should now be updated to the String "Avalanche" to show that they won the series and are continuing on in the playoffs. Use the findParent() method (described below) to help with this step.
- public BinaryTreeNode<String> findParent(String, String)
  - Take in two Strings representing team names and this method must find the parent node from which the children node contain those given Strings, if it exists.
  - Use a queue-based level-order traversal to go through the nodes. Upon "visiting" a node, you must check if it has two children and if they match the given String parameters. If so, return that parent node; otherwise return null.
- public void addNewStandings(int,int,int) [provided to you!]
- public void printStandings() [provided to you!]
- public static void main (String[]) [provided to you!]

Text Files Provided

- You are given several text files that are used by the program to provide the list of teams playing in the playoffs as well as the games' scores.
  - teams.txt provides the list of teams playing
  - scores1.txt provides the scores of all games in round 1 of the playoffs
  - scores2.txt provides the scores of all games in round 2 of the playoffs
  - scores3.txt provides the scores of all games in round 3 of the playoffs
  - scores4.txt provides the scores of all games in round 4 of the playoffs
- In Eclipse put all these files inside your project root folder (not inside bin or src)

- If your program shows an error/exception that a file could not be loaded, you might need to move these files to another folder, depending on how your installation of Eclipse has been configured or the project was created.

Marking notes

Marking categories

- Functional specifications
  - Does the program behave according to specifications?
  - Does it produce the correct output and pass all tests?
  - Are the class implemented properly?
  - Are you using appropriate data structures (if applicable)?
  - Does the code run properly on Gradescope (even if it runs on Eclipse, it is up to you to ensure it works on Gradescope to get the test marks)
- Non-functional specifications
  - Are there comments throughout the code (Javadocs or other comments)?
  - Are the variables and methods given appropriate, meaningful names?
  - Is the code clean and readable with proper indenting and white-space?
  - Is the code consistent regarding formatting and naming conventions?
- Penalties
  - Lateness: 10% per day
  - Submission error (i.e. missing files, too many files, etc.): 5%
  - "package" line at the top of a file: 5%
  - Compile or run-time error on Gradescope
  - Failure to follow instructions, (i.e. changing variable types, etc.)

Remember you must do all the work on your own. Do not copy or even look at the work of another student. All submitted code will be run through cheating-detection software.

Submission (due Friday, July 23, 2021 at 11:55pm ET)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

CS1027

ASSIGNMENT 4

Computer Science Fundamentals II

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope and the test marks come directly from there. If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular assignment deadline will receive a penalty.

Files to submit

- TreeBuilder.java
- Playoffs.java

Grading Criteria

- Total Marks: [20]
- Functional Specifications:
  - [3] TreeBuilder.java
  - [3] Playoffs.java
  - [10] Passing Tests
- Non-Functional Specifications:
  - [1.5] Meaningful variable names, private instance variables
  - [0.5] Code readability and indentation
  - [2] Code comments