CS2230 Computer Science II:Data Structures
Homework 6
Queries on binary trees

Due October 24th & November 1st

In this assignment, you will build part of a database for hierarchical data. Specifically, you will write code that can be used to query (i.e., ask questions about) data that is stored as a binary tree. Although you'll use family tree data as a case study, you will write your database using generic types so that it can be used on any data type. You'll also use higher order functions so that the database can accept custom queries.

Learning objectives for this assignment

- Write code that uses a binary tree built from linked "TreeNodes"
- Write both recursive and iterative algorithms for trees
- Use higher order functions and generic types with a new data structure
- Produce evidence that your code is correct by writing your own JUnit tests

Submission Checklist

By **October 24, 11:59pm,** you must fill in PROGRESS_REPORT.txt on GitHub (click the file then choose the edit  button). Your progress report will contain answers to the questions in that file. Double-check that your GitHub repository has the updated PROGRESS_REPORT.txt with your answers inside.

By **November 1st, 11:59** (or +slip days), You should have changes in GitHub to the following files:

- BinaryTree.java
- BinaryTreeTest.java
- FamilyRecordQuery.java

<u>You must submit to ICON: the link to your GitHub repository.</u>

You will submit them via GitHub. Follow the directions in getting_hw5.pdf, with the changes above, on "Setup your own private repository to push your commits to". Before you are done submitting, you must check the following.

- Do the tests pass?
- Does my GitHub repository reflect the code I intend to turn in? (You must view this in your web browser, not in NetBeans).

Getting HW6

Follow **all the same** instructions for getting_hw5.pdf **with the following changes**:

GitHub Url: https://github.uiowa.edu/cs2230-assignments/BinaryTreeQueries.git

Project Name: BinaryTreeQueries

adding collaborators: only add the username **rghimire**

Part 0

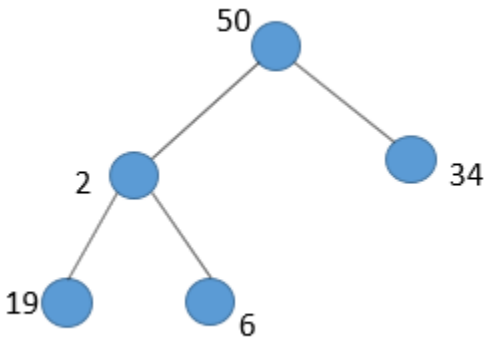
Open *familyRecord.csv* and briefly look at the data. Draw out a binary tree using this data. Start with the top record as the root of the tree. Inserting a node into the binary tree should go into the next available spot in the tree starting with the left node, an example is shown in Part 1. You will use this drawn out binary tree as reference and to check your work, in the following parts of the homework.

Part 1

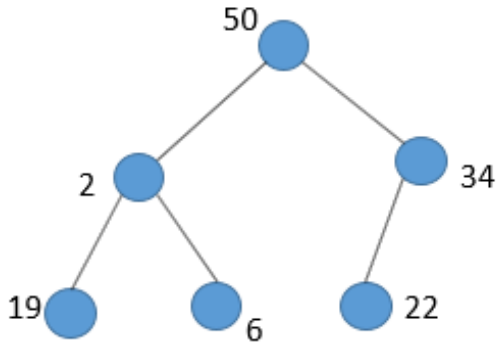
We provided a file BinaryTree.java that defines a class for a binary tree and some methods.

The method `insertNode()` inserts a node in the leftmost available free spot. That is, if the method is called in the following sequence:

`insertNode(50), insertNode(2), insertNode(34), insertNode(19), insertNode(6),` then the tree will look like:



And, if we call `insertNode(22)`, the tree will look like:



Implement this method using the fields in the constructor. Notice that there is a LinkedList called `nodesToInsertAt`. This will help you keep track of the next available spot in the binary tree. If done correctly, `testInsertionAndtoArray` test should pass.

Sanity Check:

Does `testInsertionAndtoArray` pass?

In the test file, call bt.displayTree() to print out the tree as text. Does this tree look like the example above?

Part 2

Write a method that returns the combined value of the nodes at a given depth. Here, "combined" means the same as what `ReduceFunction` defines in HW5, that is, start with `ReduceFunction.initialValue()` and combine each element to the total value using `ReduceFunction.combine()`. The depth is given as a parameter. If the depth of the tree is actually less than the given height, then return `ReduceFunction.initialValue()`.

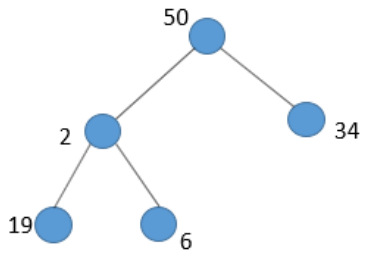
Your method must have a running time in O(N), where N is the number of nodes in the tree.

In BinaryTree.java, complete `combineValuesAtDepth()` and `combineValuesAtDepthRecurisve()`

`combineValuesAtDepthRecurisve()` must use recursion and `combineValuesAtDepth()` must be iterative (use a loop). Notice that the generic data types of `combineValuesAtDepthRecurisve()` is different than the iterative approach. This is to make the recursion easier.

Example

If the tree looks like



And the given `ReduceFunction` is plus (+), then `combineValuesAtDepth(0)` is 50, `combineValuesAtDepth(1)` is 36 and `combineValuesAtDepth(2)` is 25.

Testing

There is one test, `sumOfDepthTest`, in BinaryTreeTest.java. You must write additional test cases to ensure your code is correct. Your methods will be graded on several additional hidden tests. Some examples of things to test, an empty tree, a tree with only one node, different tree shapes etc.

Part 3

In the queries folder of the project, there is a file named FamilyRecordQuery.java. We created few methods that parses the CSV file of family data and creates a binary tree of the data. Open the CSV file in Excel or in a Text Editor to get a feel for the data. Notice that each level of the tree is a generation of the family. At depth 0 is the child Robert, depth 1 is the parents of Robert, and depth 2 the grandparents and so forth.

Write a method that returns a concatenated string of all the names in a generation **separated by spaces**.

For example, “Generation 1: Ryan Jisoon”. The String generation has the starter code. The code uses a binary tree and your combineValuesAtDepth to find the generation at a wanted depth. All you need to do is fix the class *ConcatenateNames* within FamilyRecordQuery.java.

Also create a query using the Namebt, a binary tree of names, with the *CombineValuesAtDepthRecurisve*. The String ageGroup has the starter code. For this query, please fix the *ConcatenateNamesRecursive* class in FamilyRecordQuery.java.

To check your work, use bt.displayTree() in the main method of FamilyRecordQuery.java and see if the names you printed are correct for the given generation.

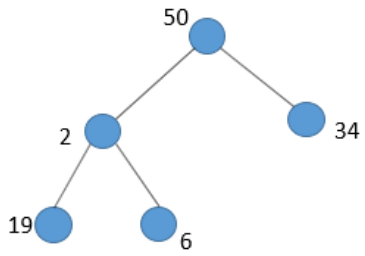
Part 4

Find the elements that are "wanted data" and return them "pre-order". Write a method to find the nodes with wanted values and return them in a list. To determine what data is "wanted", we'll use the Predicate interface from HW5. Write your method iteratively.

Note that your method must run in linear time: O(n). Implement the method `wantedNodesIterative.`

Example

If the tree contains



and our predicate is "is the element even?", then the answer will be the list [50,2,6,34]. That is, the pre-order traversal is [50,2,19,6,34] but we remove the integers where "is element even?" is false to get [50,2,6,34].

Testing

BinaryTreeTest.java contains a single test case `wantedNodesIterativeTest`. You must write additional test cases for *both* `wantedNodesIterative. Notice that wantedNodesRecursive` is already completed. You are to write test cases for both `wantedNodesRecursive` and wantedNodesIterative to make sure the code really works. Your methods will be graded on several additional hidden tests and on the quality of your tests for both methods.

Part 5

In FamilyRecordQuery.java uncomment the code required for Part 5 in the main method and the classes. As you noticed, the family has a lot of people named Robert and a lot of engineers. Use your *`wantedNodesIterative`* and *`wantedNodesRecursive`* to find these people in the family tree.

To find all the Roberts, fix the `SelectName` class and search for the **exact** string “Robert” in the name field of the FamilyRecord. This must use *`wantedNodesIterative.`*

To find all the Engineers, fix the `SelectJob` class and search for any text **containing** “Engineer” in the job field of the FamilyRecord. This must use *`wantedNodesRecursive`*. This query will output a list of many engineer types (people born in 1920 couldn’t have been Software Engineers).

Feel free to test our different names or jobs with these queries. To check your work, look at the displayed tree or the CVS file to see that the output is correct.

Part 6

Uncomment the code in the main method of FamilyRecordQuery.java. Write the code that will return and print out all the people in the family tree that are under the age of 50. Follow the same structure of

the previous queries. You must use *wantedNodesIterative* or *wantedNodesRecursive* to complete this task. Once again, check the CVS file or the displayed tree to see if your output is correct.