COMP3506/7505 Homework Task 3

Due Fri 13 Sep 2019, 5:00pm

10 marks total

Overview

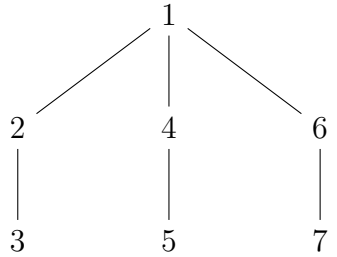
In this problem set we will working with trees! If you're feeling in*tree*gued, keep on reading :)

Support File Overview

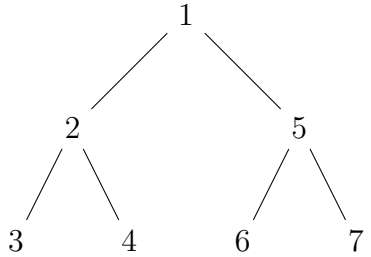
You have been supplied with a set of tree-based data structures, as follows:

- `Tree.java` contains a simple tree ADT
- `StandardTree.java` contains a non-binary tree implementation of the *Tree* ADT (that is, each node in a *StandardTree* can have as many or as few children as needed)
- `BinaryTree.java` contains a binary tree implementation of the *Tree* ADT (this is *not* an implementation of a *proper* binary tree)
- `TreeIterator.java` contains an unimplemented iterator for any tree implementing the *Tree* ADT (this is similar to an iterator over a list as described in lectures, but will instead perform a **preorder** traversal over a tree - without using recursion!)

Standard tree example



Binary tree example



Your Task

1. (3 marks) Implement the *TreeIterator* class, so that a **preorder** traversal of any tree implementing the *Tree* ADT can be performed using this iterator. At a minimum, you will need to implement the *next* and *hasNext* methods as per their specifications in the *Iterator* interface (`https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html`).

2. (2 marks) Implement the *isBST* method as per its Javadoc specification. The method should return whether a given binary tree is a binary search tree or not. You may write one or more private helper methods as a part of your solution.

3. (2 marks) In the Javadoc comments for each of the *isBST*, *hasNext*, and *next* methods, state (in big-O notation) and briefly explain the worst-case time complexity of a single call to each of these methods. If the amortised time complexity of any of these methods differ, also provide that and explain why it differs.

4. (3 marks) Trees have a variety of practical applications. For this question, you will be required to model a practical scenario that makes use of trees, as follows:

   - Write another class (in a separate Java file) that models your chosen practical scenario and implements/extends/utilises one of the given tree ADTs or CDTs. Your implementation should introduce or override at least two methods. Each of these methods should have non-trivial functionality and be documented appropriately.
   - In the Javadoc comment for this class, explain why a tree is a useful data structure to model your chosen scenario.
   - In the file `MyTreeTest.java`, write at least two realistic JUnit tests that demonstrate the usage of this data structure and its methods.

   Example scenarios include the book or file system examples described in lectures. This cannot be another theoretical tree-based data structure (eg. an AVL tree) or a theoretical algorithm that uses trees (eg. heapsort), unless you model a practical scenario where this data structure/algorithm is used. You must come up with an example that was *not* presented in lectures (these examples will receive no marks). Talk to a tutor if you are unsure about whether your implementation is classed as a practical scenario.

Constraints

- You may not modify `Tree.java` or `StandardTree.java`. You may also not modify `BinaryTree.java`, apart from implementing *isBST* and any necessary helper methods.
- Your *TreeIterator* solution should *not* use recursion (that's why it's called an iterator!)
- You may not use any constructs from the JCF, apart from those that are already imported in the base code. The only exception to this is question 4, where you are free to use any constructs from the JCF.
- Your implementation should only use basic Java programming constructs and not other libraries, apart from those that are already imported.

Failure to adhere to these constraints will result in no marks for this exercise.

Submission and Marking

Submit the following files as a part of your submission. Do not submit any other files or directories. To preserve anonymity, please do not include your name in any submitted files (it is okay to include your student number).

- `TreeIterator.java` - which should contain your solution to questions 1 and 3
- `BinaryTree.java` - which should contain your solution to questions 2 and 3
- An appropriately-named Java file containing your answer to question 4
- `MyTreeTest.java` - which should contain the JUnit tests you wrote for question 4

Questions 1 and 2 will be marked by an automated test suite with timeouts present on each of the tests. A sample test suite has been provided in `TreeTest.java`. This test suite is not comprehensive - there is no guarantee that passing these will ensure passing the tests used during marking. It is recommended, but not required, that you write your own tests for your algorithms. Marks may be deducted for poor coding style and/or inefficient algorithms.

Questions 3 and 4 will be manually marked by a tutor. Any asymptotic bounds should be as tight and simple as possible. Explanations do not have to be formal. Clearly define all variables and state any assumptions.

Late Submissions and Extensions

Late submissions will *not* be accepted. It is your responsibility to ensure you have submitted your work well in advance of the deadline (taking into account the possibility of computer or internet issues). See the ECP for information about extensions.

Academic Misconduct

Students are reminded of the University's policy on student misconduct, including plagiarism. See the course profile and the School web page:
`http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`.