Assignment: Compute "Keys At Same Binary Tree Level"

Avraham Leff

September 7, 2022

1 Assignment-Specific Packaging

The general packaging is unchanged from the basic "Homework Requirements" (see slides from first lecture and "Homework Policies for COM 2545" on Piazza).

This assignment's "DIR" **must be named** *BTKeysAtSameLevel*, and your application's Java class **must be named** *BTKeysAtSameLevel.java*.

2 Motivation

This work is being assigned at the beginning of the semester, and therefore doesn't involve material taught in *Introduction to Algorithms*. The purpose is for you to apply material that you already know ("binary trees") together with software engineering skills to solve a problem. It will also give you a sense of how I formulate a requirements document, and how I expect you to package your solution.

3 Background

You've already mastered "binary trees", understand the formal definition, and the associated algorithms. This assignment is based on the fact that every node in a binary tree has a *depth*, which is its distance from the root.

In this assignment:

1. You will be given a *String representation of a binary tree*

2. You will convert that String to the corresponding "actual" binary tree (implemented with your own data-structures)

3. You will process the binary tree and build a "list of lists" in which each list element holds the set of *node key values* at a **given depth**. The top-level List is ordered in increasing depth from the root. Within a given List element, the list is ordered from left-to-right in the binary tree.

Details of the String representation are specified in the skeleton class's (below) Javadoc.

4 Requirements

> Aside from using the JDK, for this assignment, you may **only use code written by yourself**!

The assignment consists only of a "programming" component.

4.1 Programming

> Especially because this is your first assignment, I urge you to review the general requirements for a programming assignment! I've tried to reduce the chances of "mistakes" occurring through the use of a "skeleton interface", but ultimately, **you are responsible** for ensuring that I can compile and test your code without incident.

1. Begin by downloading a skeleton `BTKeysAtSameLevel.java` class from this git repository. Then, implement the stubbed methods of *BTKeysAtSameLevel*.

You will be graded only on *correctness*: we may revisit the *performance* angle later in the semester. That said: an implementation that performs really poorly may timeout on my tests, so don't completely ignore the performance aspect in this assignment.

4.2 Small Example

4.2.1 Test Code

```
  @Test(enabled=true, timeOut=50)
  public void demo() {
    final String treeInStringRepresentation = "1(2)(3)";
    logger.info("Input string: {}",
       treeInStringRepresentation);
    final List<List<Integer>> actual =
      new BTKeysAtSameLevel().compute(
         treeInStringRepresentation);
    final List<List<Integer>> expected = new ArrayList<>();
    expected.add(new ArrayList<Integer>(List.of(1)));
    expected.add(new ArrayList<Integer>(List.of(2, 3)));
    logger.info("Result = {}", actual);
    logger.info("Expected = {}", expected);
    assertEquals(expected, actual,
                 "incorrect 'binary tree keys at same depth
                    '");
  }
```

```
onTestStart - Started test demo
demo - Input string: 1(2)(3)
demo - Result = [[1], [2, 3]]
demo - Expected = [[1], [2, 3]]
onTestSuccess - >> Test demo succceeded (took 16 ms)
```