A binary search tree is a binary tree whose elements are kept in order. In this lab you will practice the core concepts of this data structure. First solving a couple of problems on paper and then adding methods to the provided code.

Download the provided files: the .zip contains a pdf file and a java program.

1. On paper

Read the provided slides from Georgia Tech, which describes Binary Search Trees and complete the exercises on slides

- 12,
- 15 and
- 18

Finally think about the Analysis described from slide 23 til the end. You should understand the importance of the BST to be **balanced**.

Ask us if you have any difficulties. We are here to help you.

2. Code

In this lab you have to implement the following four methods for BSTs of (unique) integers. Use the definition of a binary search tree provided in the provided file.

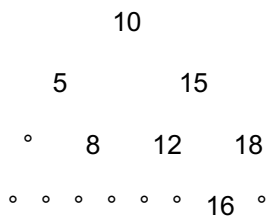
Note that a non-balancing insert() method, which keeps track of the tree's size, has already been implemented for you. This method will not re-insert a key (if a key is already present, it returns false). A traversal method has also been implemented for you, which you can use to help you debug.

int height()

Return the height of the tree: the number of edges on the longest downward path between the root and a leaf.

String printByLevel()

Return a string that contains the 2D output of the BST: a level-by-level traversal of the BST. The output needs to be centered and to use place holder for null nodes, such that the relations between level are easily readable. You can assume that key are usually a digit. For example, inserting in order the keys 10, 5, 15, 12, 8, 18 and 16 should produce an output similar to the following



Help

In class we studied three different types of *depth-first traversals*, preorder, inorder, and postorder.

printByLevel() is a *breadth-first traversal*--the level order traversal

- read the algorithm description
- read and watch the different traversal animations

Start implementing a simpler version such as printing first the following

10

5 15

° 8 12 18

° ° ° ° ° ° 16 °

int LCA(int x, int y)

Find the *lowest common ancestor* of the keys x and y, which is the node farthest from the root having both x and y in subtrees (including itself).

Throw a NoSuchElementException if x or y is not in the tree.

Running time should be O(*h*) where *h* is the height of the tree.

I recommend drawing some BST examples and figuring out to process to find the LCA for two given keys on paper.

Iterable<Integer> findBetween(int x, int y)

Returns a collection of keys from the tree in the range between x and y, inclusive, in ascending order. The object returned can be anything that supports iteration so that the items can be printed in order, for example, a linked list, an array list, etc.

Running time should be O(*k* + *h*), where *k* is the number of keys in the range and *h* is the height of the tree.

You are not expected to submit any files. This lab is to practice for the final exam.