TD2- Binary trees

Version originale : Marc Gaetano et l'équipe pédagogique ASD

This lab will give you practice about basic binary tree processing.

Fichiers supports à l'ensemble du TD

Part 1: introduction to binary trees

In this part you must write the basic methods inside the *BinaryTree* class.
Carefully check this class and look at the provided height method.
You are to complete the following methods:

- **size**: returns the number of non-null node in the tree
- **lowness**: returns the lowness of the tree (i.e., the length of a *shortest* path from the root to a leaf)
- **leaves**: returns the number of leaves of the tree (remember that a leaf is a non-null node with two null sub-tree)
- **isomorphic**: two binary trees are said to be isomorphic if they have the same structure, no matter the elements they hold. Same structure means same number of nodes, and all nodes at the same place in both trees

For all these methods, you must give the worst case run time complexity.

Supporting files :

- BinaryTree.java
- TestBinaryTree.java : *tests Junit*

- BinaryTreeInterface.java
- BinaryTreeTest.java : *tests interactifs*

Part 2: more methods on binary trees

In this part you must implement more algorithms on binary trees.
All the methods to complete are still in the *BinaryTree* class.

1. A binary tree is said to be *balanced* if, for each of its sub-trees, the absolute value of the difference between the height of the left sub-tree and the height of the right sub-tree is at most 1. You are to complete the following method:

- ***balanced1***: check if a binary tree is balanced. To write this method, you must use the height method

2. A binary tree is said to be *shapely* if, for each of its sub-trees, the height is less or equal than the double of the lowness. You are to complete the following method:

- **shapely1**: check if a binary search tree is shapely. To write this method, you must use the height and the lowness methods

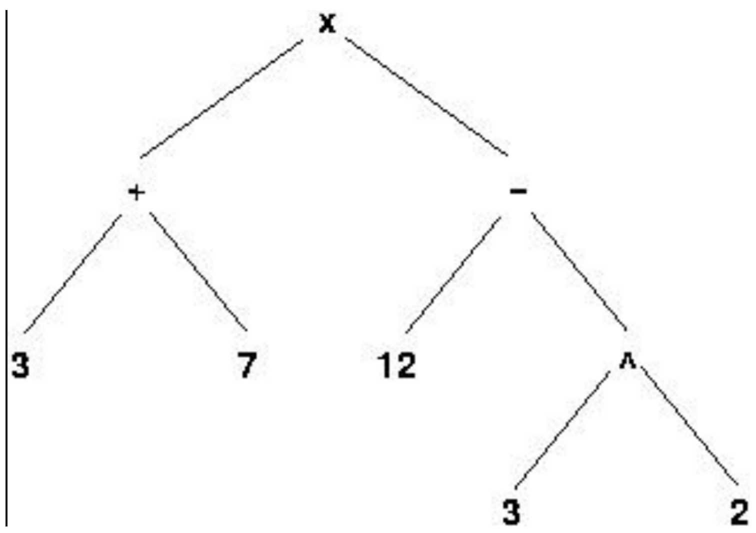
3. What is the worst-case complexity for the methods balanced1 and shapely1?
Explain why and how we can improve this complexity.
Finally, complete the following methods:

- **balanced2**: same as balanced1 but this new version does not use the method height
- **shapely2**: same as shapely1 but this new version does not use the methods height and lowness

Part 3: mathematical expression tree

A mathematical expression can be represented as a binary tree whose internal nodes (non-leaf nodes) are operators and leaves are values.

For example, the expression (3 + 7) x (12 - 3^2) can be implemented as the following binary tree:



Complete the class **ExpressionTree** such that the method eval returns the result of evaluation of an expression tree.
For example, if e is the *ExpressionTree* implementing the expression of the previous picture, e.eval() would return 30.

Supporting file:

- ExpressionTree.java

Part 4: Traversals and Function

Implement the methods of the Traversals class, which implement 3 tree traversals and apply a function on each node.
You only have to implement inOrderTraversal and postOrderTraversal and test them.

The different traversals are presented in the course.

```
A
|_ B
|  |_ B1
|  |
|  |_ B2
|_ C
   |_ C1
   |  |__ C11
   |  |
   |  |__ C12
   |
   |_ C2
```

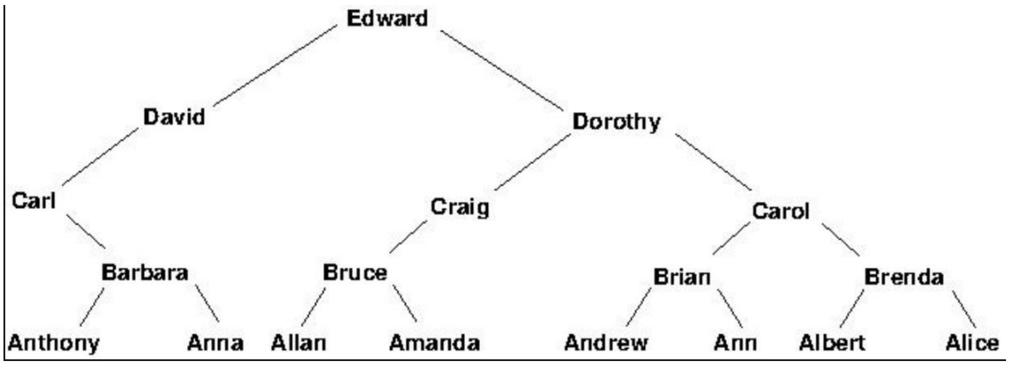
PreOrder :	[A, B, B1, B2, C, C1, C11, C12, C2]
PostOrder :	[B1, B2, B, C11, C12, C1, C2, C, A]
InOrder :	[B1, B, B2, A, C11, C1, C12, C, C2]

Supporting file:

- Traversals.java
- TraversalsTest.java

Part 5: Genealogy tree (Optional)

The *genealogy tree* of a person can be implemented by a binary tree: the root node holds the person's data (to simplify, we consider just the name of a person), the left sub-tree being the genealogy of her/his father and the right sub-tree the genealogy of her/his mother. For example, the following binary tree is the genealogy of Edward.



The father of Edward is David, and the mother of Edward is Dorothy. Carl and Craig are the two grandfathers of Edward but for some reason Edward only knows about his grandmother from his mother side, Carol. Using the class *GenealogyTree*, you are to complete the following methods:

1. **ancestors**: returns the list of ancestors of the person at the root of the genealogy tree at a given level. For example, for the previous genealogy (the genealogy of Edward), the ancestors at level 2 (the grand-parents) are Carl, Craig and Carol.
2. **maleAncestors**: same as ancestors but prints only the male ancestors. For example, for the previous genealogy (the genealogy of Edward), the male ancestors at level 2 (the grand-fathers) are Carl and Craig.
3. **displayGenerations**: prints the ancestors, line by line, each line being a generation. For example, for the previous genealogy (the genealogy of Edward), the result of *displayGenerations* should be:

```
 Edward
David Dorothy
Carl Craig Carol
Barbara Bruce Brian Brenda
Anthony Anna Allan Amanda Andrew Ann Albert Alice
```

The complexity of *displayGenerations* must be *linear in the number of nodes* of the genealogy tree!
(Hint: you must use a queue and perform a breadth-first traversal of the tree)

Supporting file

- GenealogyTree.java

Part 6: mathematical expression tree & BiFunction (facultatif)

Définissez (si vous ne l'avez pas encore fait) les opérateurs comme suit :

```
private static final Map<String, BiFunction<Double, Double, Double>>
OPERATORS =
   Map.of(
      "+", (a, b) -> a + b,
      "-", (a, b) -> a - b,
      "*", (a, b) -> a * b,
      "/", (a, b) -> a / b,
      "^", (a, b) -> Math.pow(a,b)
   );
```

1. En utilisant le *apply*, redéfinissez votre classe *ExpressionTree*
2. Nous souhaitons à présent lire des expressions Mathématiques telles que :

(1 + 2) * (3 + 4) => 21 ou 1 + 2 * 3 + 4 => 11

Pour répondre à ces spécifications vous pouvez utiliser 2 piles,

L'une qui contient les opérateurs par exemple (+ et
L'autre qui contient les opérandes par exemple 1 2.

Remarques :

Pour ceux qui demanderaient à ChatGPT de le faire pour vous, et en supposant que vous parveniez à lui faire générer, prenez le temps (1) de le tester, (2) de le comprendre, sinon ce n'est même pas la peine d'essayer.