Algorithms & Data Structures
SI3 - Polytech Nice-Sophia - Edition 2018

# Lab #3: Dictionary ADT, Binary Trees

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

## Part 1: introduction to binary trees

In this part you have to write the basic functions manipulating objects of the `Tree` class. Check carefully this class and look at the provided height function. You are to complete the following functions:

- `lowness`: returns the lowness of the tree (i.e. the length of a *shortest* path from the root to a leaf)
- `size`: returns the number of nodes in the tree
- `leaves`: returns the number of leaves of the tree (remember that a leaf has no sons : both are `None`)
- `isomorphic`: two binary trees are said to be isomorphic if they have exactly the same structure, no matter the data they hold. Same structure means same number of nodes, and all nodes at the same place in both trees.

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

**Supporting file:**

- Lab3.py

## Part 2: more functions on binary trees

In this part you have to implement more algorithms on binary trees.

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

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:

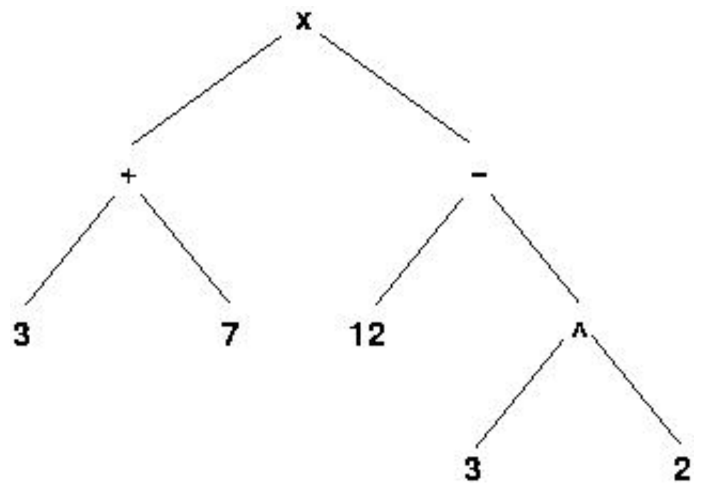
- `shapely1`: check if a binary search tree is shapely. To write this method, you must use the `height` and the `lowness` methods

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

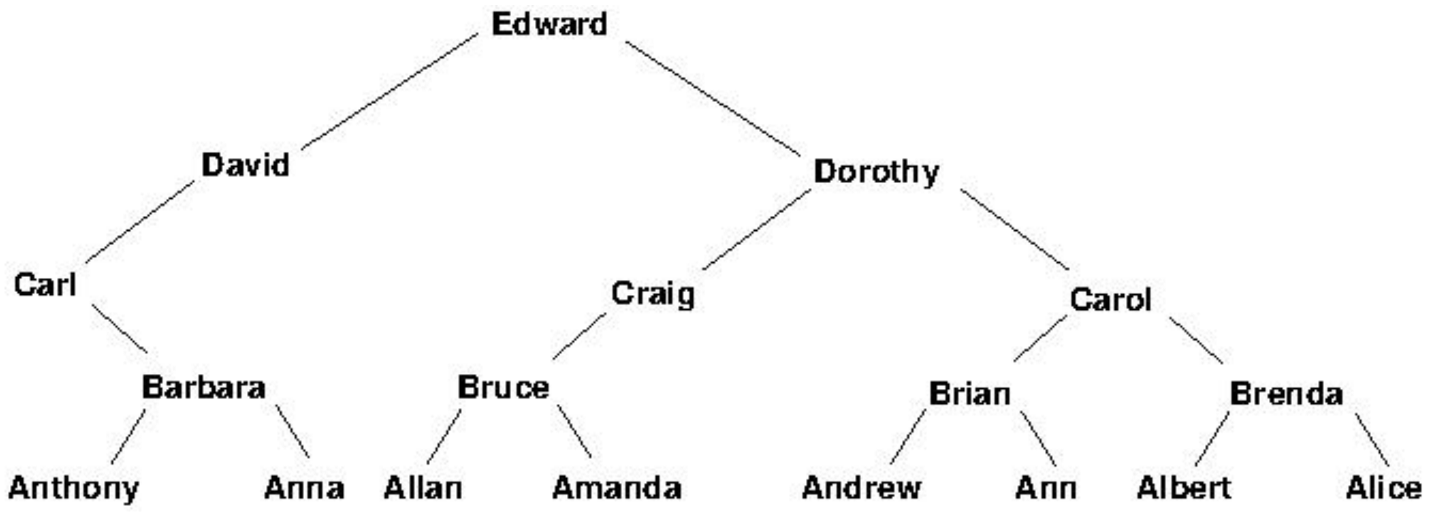
A mathematical expression can be represented as a binary tree whose internal nodes (non leaf nodes) are operators and leaves are values. For example, the expression (3 + 7) x (12 - 3^2) can be implemented as the following binary tree:



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.

## Part 4: genealogy tree

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 grand-fathers of Edward but for some reason Edward only knows about his grand-mother from his mother side, Carol. Using the class `GenealogyTree,` you are to complete the following methods:

- `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.
- `ancestors(male=True):` 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.