Information and Communication Technologies — S7 — Work n°12

Trees — Bonus Part

This week, you can choose on what you want to work. I have put some bonus exercises, feel free to choose what you like inside.

Exercice 1 — Balanced binary trees

In this exercise, we use the same Tree class as before, see Listing 1.

```python
class Tree:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

    def __str__(self):
        return str(self.data)
```

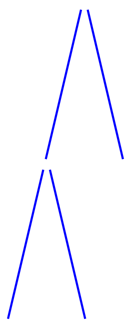
Listing 1: The Tree data structure.

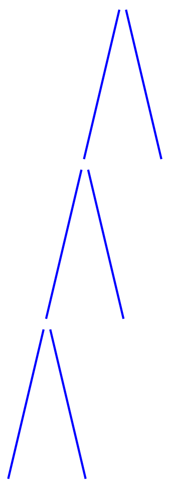
A binary tree is balanced (a) if it's an empty tree or (b) if the height of its two children differs by at most one.

Define, in Python, a function `is_balanced(tree)` that takes a tree as argument, and returns a boolean telling whether the tree is balanced or not.

Unit tests:

- `is_balanced(tree1)` should return True (where tree1 is the tree of the left image below);
- `is_balanced(tree2)` should return False (where tree2 is the tree of the right image below).





Exercice 2 — Fibonacci trees

You have maybe heard about the Fibonacci sequence. It's a sequence of integers defined recursively in the following manner:

$$\begin{cases} F_0 = F_1 = 1 \\ \forall n \geq 2, F_n = F_{n-1} + F_{n-2} \end{cases}$$

So for instance $F_2 = F_1 + F_0 = 1 + 1 = 2$, then $F_3 = F_2 + F_1 = 2 + 1 = 3$, then $F_4 = F_3 + F_2 = 3 + 2 = 5$, ...

Now let us define the Fibonacci trees as binary trees in the following manner:

- T_0 and T_1 are each the tree with only one empty node (e.g. define T_0 and T_1 as `Tree("")` with the Tree data structure of previous works, also given in the previous exercise, see Listing 1).
- $\forall n \geq 2$, T_n is the tree that has one empty node, as left child T_{n-1} and as right child T_{n-2}.

1. Draw T_0, T_1, T_2, T_3 and T_4.
2. Define, in Python, a function `Fibonacci_tree(n)` that takes a non-negative integer as argument, and returns the associated Fibonacci tree.

Exercice 3 — n-ary trees

In this exercise, we use a new `Tree_n` class to define trees with any number of children, see Listing 2.

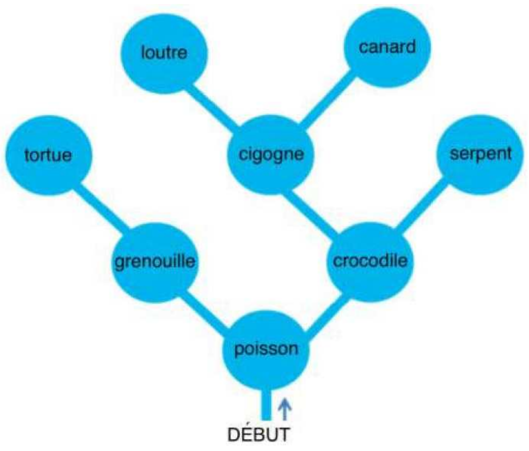
```python
class Tree_n:
    def __init__(self, data, children=[]):
        self.data = data
        self.children = children

    def __str__(self):
        return str(self.data)
```

Listing 2: The n-ary tree data structure.

An n-ary tree is either the empty tree None, as before, or is a tree with a list of children. The equivalent of a tree with two None children, as before, would be a tre with an empty list for the children (which means it is the empty list `[ ]`). In case the tree has any children, they are simply put in the children list.

Let's consider the same image as in a previous work:



1. Define, in Python, an n-ary tree corresponding to this image.
2. Write a function that takes as parameter an n-ary tree, and gives back the number of nodes.
3. Write a function that takes as parameter an n-ary tree, and gives back its height.