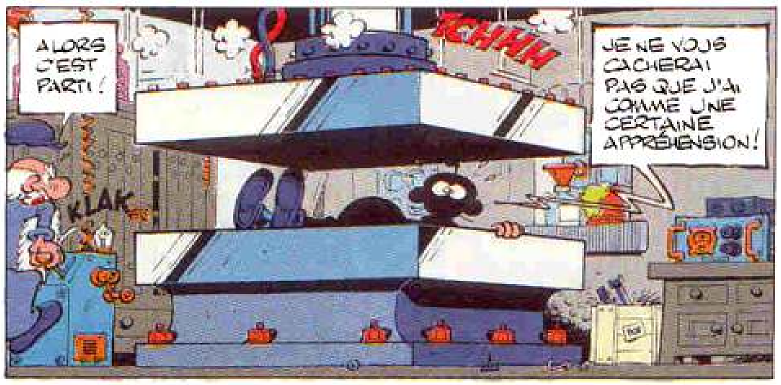
Algorithmics
Final Exam #2 (P2)

Undergraduate 1^{st} year S2
EPITA

30 May 2018 - 14 : 00

Instructions (read it) :

- ☐ You must answer on **the answer sheets provided.**
  - No other sheet will be picked up. Keep your rough drafts.
  - Answer within the provided space. **Answers outside will not be marked**: Use your drafts!
  - Do not separate the sheets unless they can be re-stapled before handing in.
  - Penciled answers will not be marked.
- ☐ The presentation is negatively marked, which means that you are marked out of 20 points and the presentation points (maximum of 2) are taken off this grade.
- ☐ **Code:**
  - All code must be written in the language `Python` (no C, CAML, ALGO or anything else).
  - **Any `Python` code not indented will not be marked.**
  - All that you need (types, routines) is indicated in the **appendix** (last page)!
  - Your functions must follow the given examples of application.
- ☐ Duration : 2h



Exercise 1 (AVL – *3 points*)

Starting with an empty tree build the AVL corresponding to the successive insertions of values 25, 60, 35, 10, 20, 5, 70, 65, 45.

- Only draw the final tree.
- Give used rotations in order (for instance if a left rotation occurred on the tree the root of which is 42, write *lr(42).*)

Exercise 2 (Leonardo trees – *3 points*)

In this exercise we will study some properties of a certain type of tree: the Fibonacci trees. These are defined recursively as follows:

$$\begin{cases} A_0 = \ EmptyTree \\ A_1 = < o, EmptyTree, EmptyTree > \\ A_n = < o, A_{n-1}, A_{n-2} > \ \ if\ n \geqslant 2 \end{cases}$$

1. Give a graphical representation of the Fibonacci tree A_5.
2. (a) Give, in terms of $n \geqslant 2$ the height h_n of the tree A_n.
   (b) Prove that the tree A_n is height-balanced.

Exercise 3 (`List` → `AVL` – *5 points*)

Using a strictly increasing list we want to build a *balanced* binary search tree (A.-V.L.). For instance, from the list bellow we want to obtain one of the trees in figure 1.

0	1	2	3	4	5	6	7	8	9	10	11
1	4	5	7	10	12	13	15	18	20	21	25

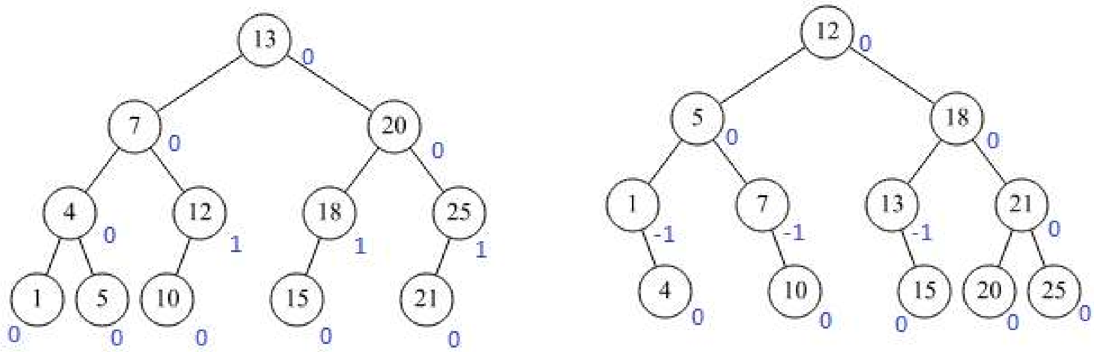


Figure 1: A.-V.L.

Write the function `list2avl(`L`)` that returns an A.-V.L. (class `AVL`) built from the list L sorted in stricly increasing order.

Exercise 4 (AVL - Minimum deletion – *6 points*)

We focus here on the deletion of the minimum value in an AVL with re-balancing.

1. Fill the given table with the rotations to execute and the possible induced height variations (Δh = 0 if the tree does not change in height after rotation, 1 otherwise) for each unbalanced case (*bal*), only after the deletion of the minimum.

2. Write the recursive function that deletes the minimum value of a non-empty AVL (with balance factor updates and possible re-balancing while going up). It returns the tree after deletion and a boolean that indicates whether the tree height has changed (a pair). You can use the functions that perform the rotations with balance factor updatings (`lr, rr, lrr, rlr`, see appendix.)

Exercise 5 (BST and mystery – *4 points*)

```python
def bstMystery(x, B):

 # first part
    P = None
    while B != None and x != B.key:
        if x < B.key:
            P = B
            B = B.left
        else:
            B = B.right
    if B == None:
        return None

 # second part
    if B.right == None:
        return P
    else:
        B = B.right
        while B.left != None:
            B = B.left
        return B
```

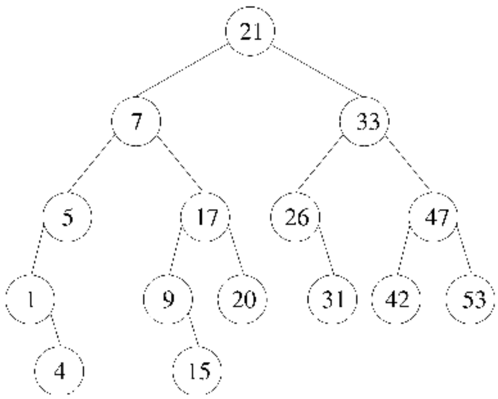


FIGURE 2 – tree B_1

```python
def call(x, B):
    p = bstMystery(x, B)
    if p == None:
        return None
    else:
        return p.key
```

1. Let B_1 be the tree given above. What are the results of each of the following calls?
   (a) `call(25,` B_1`)`
   (b) `call(21,` B_1`)`
   (c) `call(20,` B_1`)`
   (d) `call(9,` B_1`)`
   (e) `call(53,` B_1`)`

2. `bstMystery(x, B)` is called with `B` any binary search tree, where all elements are different.

   During execution, at the end of part 1:

   (a) What does `B` represent?

   (b) What does `P` represent?

3. What does the fonction `call(x, B)` do?

Appendix

Binary Trees

Usual binary trees:

```
class BinTree:
    def __init__(self, key, left, right):
        self.key = key
        self.left = left
        self.right = right
```

AVL, with balance factors:
Reminder: in an A.-V.L keys are unique.

```
class AVL:
    def __init__(self, key, left, right, bal):
        self.key = key
        self.left = left
        self.right = right
        self.bal = bal
```

Authorised functions and methods

Rotations (A:`AVL`): each of the functions bellow returns the tree A after rotation and balance-factor update.

- `lr(`A`)` : left rotation
- `rr(`A`)` : right rotation
- `lrr(`A`)` : left-right rotation
- `rlr(`A`)` : right-left rotation

On lists:

- `len`
- `append`

Others:

- `abs`
- `min` and `max`, but only with two integer values!

Your functions

You can write your own functions as long as they are documented (we have to known what they do).
In any case, the last written function should be the one which answers the question.