Algorithmics
Final Exam #2 (P2)

Undergraduate 1^{st} year S2#
EPITA

8 January 2019 - 11 : 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 (How many? – *Final S2# – Jan. 2019*)

Write the function `nb_inter(B, a, b)` computes the number of values of the binary search tree B in $[a, b[$

Application examples with `B1` the tree in figure 1:

```
>>> nb_inter(B1, 0, 27)
12
>>> nb_inter(B1, 6, 17)
5
```

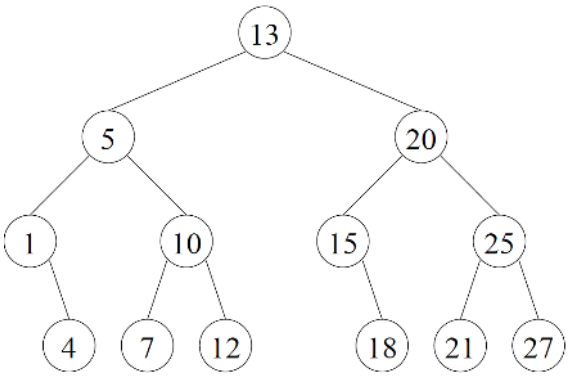

Figure 1: BST `B1`

Exercise 2 (BST → AVL – *Final 2# - 2019*)

Write a function that builds from a classic BST (a binary tree without the balance factors) an equivalent tree (containing same values at same places) but with the balance factor (the "field" *bal*) specified in each node.

Clue: You might need a recursive function that will return the copied tree as well as its height.

Exercise 3 (AVL - Add 0 – *5 points*)

We work here with AVL that contain only non zero naturals.

Write the recursive function `add0(`A`)` that inserts the value 0 in the AVL A (with balance factor updates and possible re-balancing while going up). It returns the tree after insertion 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 4 (AVL – *3 points*)

Starting with an empty tree build the AVL corresponding to the successive insertions of values 13, 20, 5, 1, 15, 10, 18, 25, 4, 21, 7, 12, 23.

- 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 5 (2.4-tree → Red-black Tree – *2 points*)

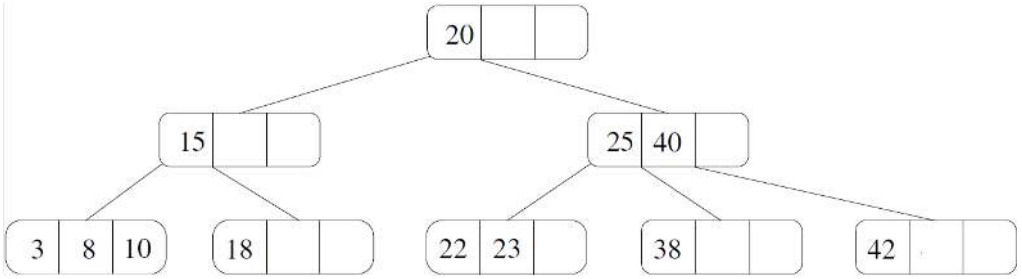

Figure 2: Arbre 2.3.4 à transformer

1. Build the red-black tree corresponding to the 2-4 tree in the figure 2. You will assume that all the 3-nodes lean to the left.
2. Is the result tree an AVL? Justify your answer (be precise and concise).

Exercise 6 (Trees and mystery – *4 points*)

```
1            def __makeTree(n, i, cur):
2                if i > n:
3                    return (None, cur)
4                else:
5                    (left, cur) = __makeTree(n, 2*i, cur)
6                    key = cur+1
7                    (right, cur) = __makeTree(n, 2*i+1, key)
8                    return (binTree.BinTree(key, left, right), cur)
9
10           def makeTree(n):
11               (B, val) = __makeTree(n, 1, 0)
12               return B
```

1. The function `makeTree(n)` builds (and returns) a binary tree. Draw the result tree when $n = 13$.
   If you do not see which values to put as keys, draw the structure anyway (the tree without values).
2. `makeTree(n)` is called with `n` a strictly positive integer.
   Give two properties of the returned tree.

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

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.