Algorithmics
Final Exam #2 (P2)

Undergraduate 1st year (S2)
Epita

29 May 2017 - 13h45

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 (functions, methods) is indicated in the **appendix** (last page)!
- ☐ Duration : 2h

Exercise 1 (2-4 trees ... – *6 points*)

1. Following the principle of insertion with splitting in going up, build from an empty tree the 2-4 tree corresponding to the successive insertions of values $\{Q, U, E, S, T, I, O, N, B, A, Z, Y, K\}$.(You must draw only the final tree.)
2. Build the red-black tree corresponding to the final 2-4 tree of the previous question. You will assume that all the 3-nodes lean to the left.
3. State three properties of a 2-4 tree.
4. State three properties of a red-black tree.
5. What ***simple*** method, using the red-black tree that represents it, is used to determine the size (number of nodes) of a 2-4 tree?

Exercise 2 (Trees and mystery – *3 points*)

```python
def __makeTree(n, i, cur):
    if i > n:
        return (None, cur)
    else:
        (left, cur) = __makeTree(n, 2*i, cur)
        key = cur+1
        (right, cur) = __makeTree(n, 2*i+1, key)
        return (binTree.BinTree(key, left, right), cur)

def makeTree(n):
    (B, val) = __makeTree(n, 1, 0)
    return B
```

1. The function `makeTree(n)` builds (and returns) a binary tree. Draw the result tree when $n = 13$.
2. `makeTree(n)` is called with `n` a strictly positive integer.

   Give two properties of the returned tree.

Exercise 3 (BST → AVL – *5 points*)

Write a function that builds from a classic binary tree (`BinTree`) an equivalent tree (containing same values at same places) but with the balance factor (the "field" *bal*) specified in each node (`AVL`).

Clue: The recursive function will return the tree height as well.

We do not deal with the verification of the balancing.

Exercise 4 (AA Trees – *6 points*)

AA trees (Arne Andersson trees) are a variation of red-black trees. Unlike red-black trees, red nodes on an AA tree can only be added as a right subchild. In other words, no red node can be a left sub-child.

This results in the simulation of a 2-3 tree instead of a 2-3-4 tree: there are only 2-nodes and 3-nodes with 3-nodes always leaning to the right.

For the implementation, we add an additional information (see the class `AAtree` in appendix): the `level` that represents the level from the bottom up, in the corresponding 2-3 tree (leaves at level 1). So, two keys in the same 3-node of the 2-3 trees have the same level in the AA tree (the biggest key, viewed as red, is the right child of the smallest).

For instance, the 2-3 tree in the figure 1 is represented by the AA tree in figure 2: "red" keys are those that have the same level as their parent (18, 15 and 28 here) and are always on the right.

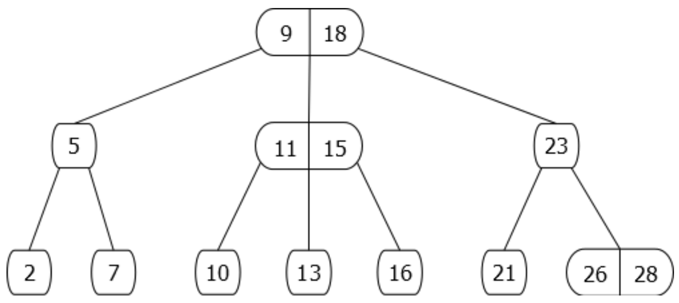

Figure 1: 2-3 tree

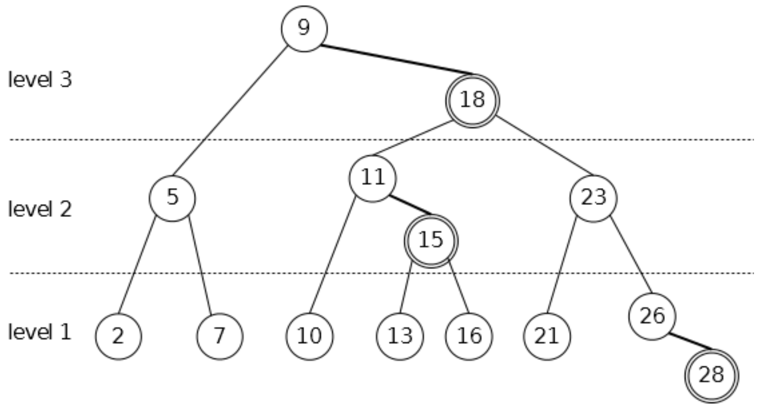

Figure 2: AA tree *A*

In the AA tree, the level represents the number of left links to follow to reach an empty child:

- Leaves are at level 1.
- Any node at an upper level must have at least a right child of same level (viewed as red) or of lower level.
- There can never be 3 nodes of same level in one path (a node can not have the same level as its grandfather).
- The level of every left child is exactly one less than that of its parent (no red child on the left).

Only two distinct operations are needed for restoring balance after an insertion or a deletion:

Skew: A right rotation used when an insertion or a deletion creates a left red child (the left child has the same level as its parent). Both nodes keep the same level.

Split: A left rotation used when there are two consecutive red nodes (3 consecutive nodes of same level on the right). The root has its level increased (like a split in a 2-3 tree).

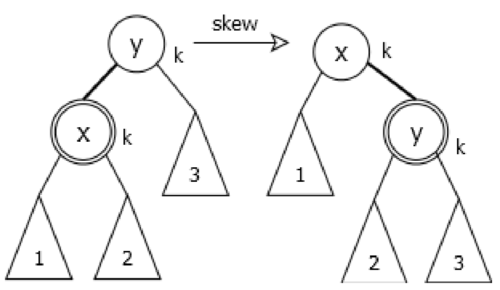

Figure 3: Skew

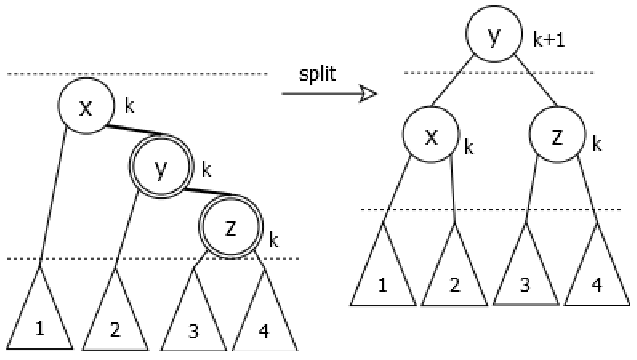

Figure 4: Split

We will assume that the function `skew` and `split` are both implemented: they take an AA tree as parameter, process the rotation, as well as the possible update of the levels and return the modified tree.

The principle of the insertion of the value x in the AA tree AA is the following:

- Going down like in a BST to insert the key at the leaf: the new key x is inserted at level 1. Unless x is already in the tree, in that case it is not inserted.
- Going up:
  - from the right: if there are 3 nodes at the same level (always on the right), perform a "split". For instance, see the result of the insertion of 31 in the tree in figure 2 (see figure 5).
  - from the left: we have to make sure that the left child has not the same level as the actual node (for instance when inserting 1 in the tree figure 2). In that event a "skew" is performed and, only in that case, we check if the right grandchild (after transformation) has not also the same level. If this is the case, a "split" is required. This can be illustrated by the insertion of 25 in the tree in figure 2 (see figure 6).

Transformations can spread to the root of the tree.

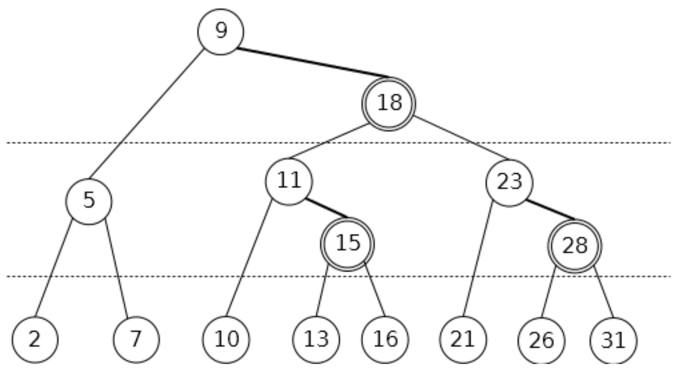

Figure 5: Tree A after insertion of 31

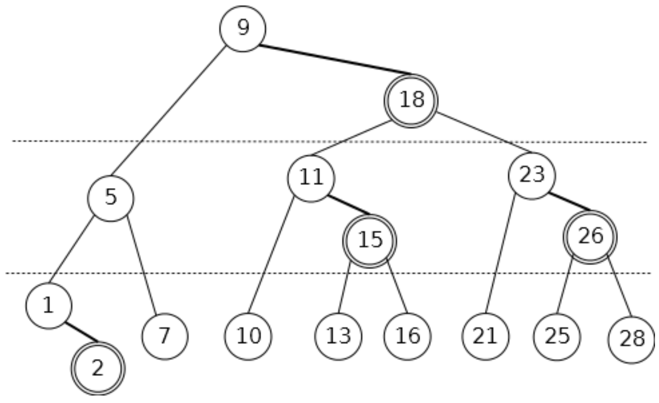

Figure 6: Tree A after insertion of 1 then 25

1. Draw the AA tree after the insertion of the value 4 in the tree in figure 6.
2. Write the function `insertAA(`x`, `A`)` that inserts the key x in the AA tree A, unless x is already in A. The function returns the resulting tree.

Annexes

Binary Trees

Usual binary trees:

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

AVL, with balance factors:

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

AA trees, with the level:

```python
class AAtree:
    def __init__(self, key, left, right, level):
        self.key = key
        self.left = left
        self.right = right
        self.level = level
```

In any case, the empty tree is `None`.

Authorised functions and methods

- `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.