NE PAS ÉCRIRE DANS CE CADRE

NOM : LUZZATTO
Prénom : JULIEN
Promotion : BX2021
COMPOSITION d CSE 101 – COMPUTER SCIENCE
en date du 30/01/2019

Signature de l'élève

Groupe de PC

Remplissez l'en-tête de la copie sans AUCUN renseignement supplémentaire

* Question 2.4: 5

Again, before even reading, we can say that print_odd_lines_2 is much clearer in its approach than print_odd_lines_1, where it is confusing to keep track of all the variables.

Second, as a matter of efficiency (both in memory and time), the second function performs much better. It is extremely easy to understand how it works: it simply prints one line out of two, without taking care nor storing the other ones. As a comparison, the first function stores all the lines (and copying costs O(n)), and then iterates multiple times (each time costing O(n)). and occupies memory, and slows down everything

The second function is then clearer and much more efficient.

Ok

NOTE

N° 1/2

COMPOSITION d CSE 101
en date du 30/01/2019

PART II -

* Question 2.1: 3

We can notice a couple of errors; first, there is an important inversion of the second arguments of open: the infile should be read, the outfile should be wrote. Second, characters like '\n' have length 1, so if we do not strip them from the line, they will be counted in len(line), which is not what we want.

As a summary:

(should be)

6 • infile = open(filename, 'w') → infile = open(filename, 'r')
or infile = open(filename)
or with open(filename, 'r') as infile

7 • outfile = open(filename + '.counts', 'r') → outfile = open(filename + '.counts', 'w')
or with open(filename + '.counts', 'w') as outfile

9 • outfile.write(len(line)) + '\n') → outfile.write(len(line.strip('\n')) + '\n')

(1, 2, 3...) = mistakes

NE PAS ÉCRIRE DANS CE CADRE

* Question 2.3: 3

We might want to highlight two problems; first, an important one: seq is an iterable. Hence, removing elements from the iterable during an iteration on its elements is going to make the program crash. A solution could be to copy the iterable at the beginning, iterating over its copy and removing from the original one, or vice-versa. The second one is that not all iterables are simply made of integers. Thus, a comparison with 0 could potentially make no sense. We could therefore ask for the type of object.

As a summary:

just another name, not a copy

- (we add a line): seq_copy = seq
- (became 5) • if x < 0 → (we add another line): if type(x) == int:
- (became 7) • seq.remove(x) → seq.remove(x)
- P.S.: (became 9) • for x in seq → for x in seq_copy:

Basically:

```
def remove_negatives(seq):
    seq_copy = seq
    for x in seq_copy:
        if type(x) == int:
            if x < 0:
                seq.remove(x)
```

* Question 2.2: 3

First of all, we can say that the function is not clear (it is, objectively and with sincerity, hard to understand).

The first big problem is the index: after a certain recursion, when seq will be strictly less than 3 characters long, seq[1:-1] will be empty, so applying reverse (and the seq[0] to it) is going to make it crash. Thus, we should add to our recursive reverse function a (before line 3):

```
if len(seq) == 1:
    return seq
elif len(seq) == 0:
    return None
else:
    ...
```

The second big problem is that the function simply does not do what we want: by adding seq[0], seq[-1] etc., it adds their value, not as a list. We would then prefer something like:

return [new_first] + new_middle + [new_last] → return [new_first, new_middle, new_last]

Or, simply:

```
def reverse(seq):
    return seq[::-1]
```

NE PAS ÉCRIRE DANS CE CADRE

NOM : LIZZATTO

Prénom : JULIEN

Promotion : BX2021

COMPOSITION d CSE101

en date du 30/01/2019

Signature de l'élève

Groupe de PC

Remplissez l'en-tête de la copie sans AUCUN renseignement supplémentaire

NOTE

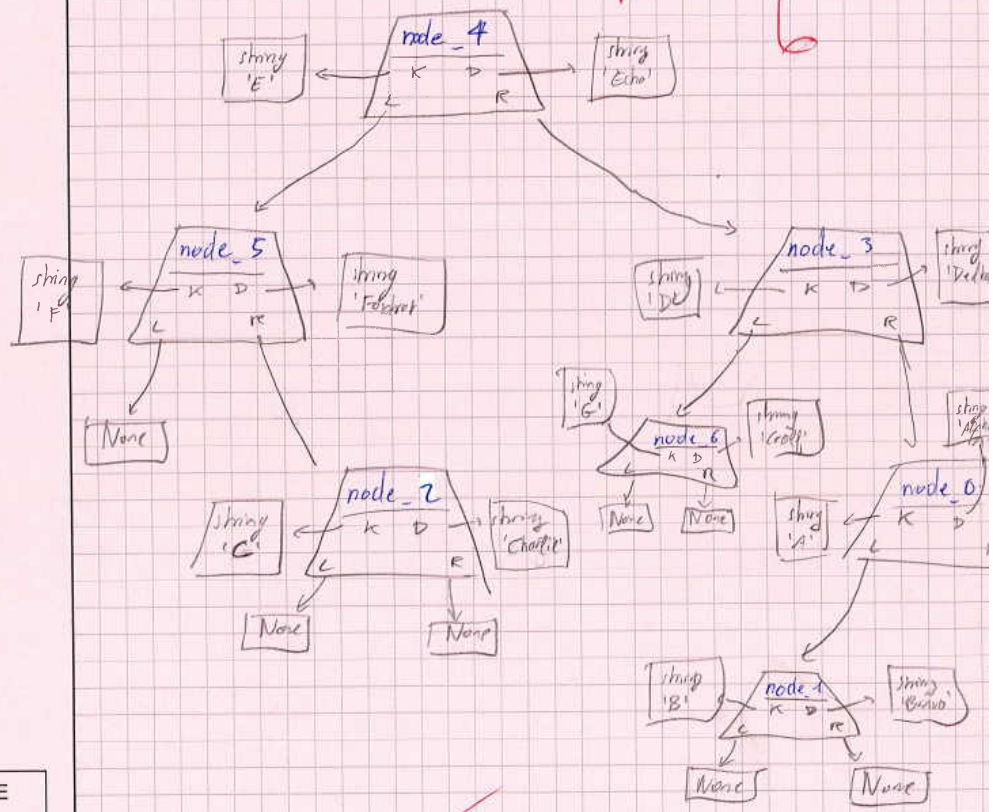
N°

COMPOSITION d CSE101

en date du 30/01/2019

PART III -

1) TreeNode diagram:



2) It is not a binary search tree, as for binary search trees, each node at the left of a node has a key smaller than the key of the node, and each node at its right a key bigger than the key of the node. (continues →)

NE PAS ÉCRIRE DANS CE CADRE

NE PAS ÉCRIRE DANS CE CADRE

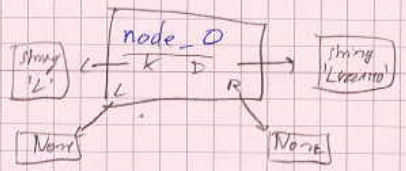
3

Here, it is the exact contrary; hence, a specular version of this tree (where we would invert each node's left and right, would be a binary search tree.

4

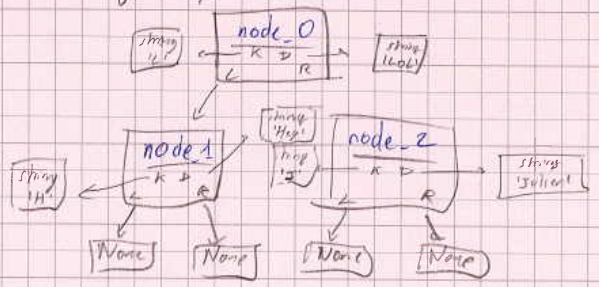
3) • First, the function does not treat the case where the key is not contained in the tree!

Take the tree:



assume we look for contains('J'). Then 'j' != 'k', but self.left == None and self.right == None. Thus, it will cause a runtime error.

• Then, we might say that in such a case, (we still look for 'J')



how is this one different?

then 'J' is contained as a key (in the right node); but the first 'elif' will go to the left, as 'J' < 'L' and self.left != None. (it will eventually reach for the nearest described before, but even if there was a 'return False' it would not find 'J').

• finally (or first) a syntax problem: else if should be elif, otherwise nothing is going to work except if self.key == k.

~~what if self.right == None?~~

~~ex:~~