Algorithmics
Midterm #3 (C3)

Undergraduate 2^{nd} year - S3
EPITA

November 2019

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 (class, types, routines) is indicated in the appendix (last page).
  - You can write your own functions as long as they are documented (we have to know what they do).
    In any case, the last written function should be the one which answers the question.
- ☐ Duration : 2h

Exercise 1 (Axes and graphs... – *5 points*)

1. Give two indirect methods of hashing.
2. With which collision resolution method do secondary collisions appear?
3. What is a secondary collision?
4. What is the order of a digraph?
5. How do we name a zero degree vertex?

Let `G=<S,A>` be a directed graph defined by:

```
   S={1,2,3,4,5,6,7,8,9,10}
and A={(1,2),(1,6),(1,7),(2,3),(2,6),(3,1),(3,5),(4,3),(4,8),(4,9),(4,10),
       (5,1),(7,6),(8,5),(8,10),(10,9)}
```

6. If they exist, which vertices of `G` have an outdegree equal to 0?
7. If they exist, which vertices of `G` have an indegree equal to 1?

Exercise 2 (Average Arity of a General Tree – *5 points*)

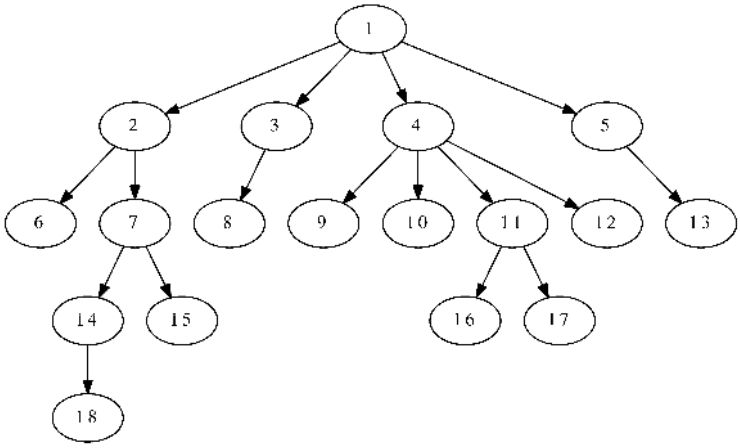

Figure 1: General tree

We now study the average arity (number of children for a node) in a general tree. We define the average arity as the sum of the number of children per node divided by the number of internal nodes.

For example, in the tree from figure 1, there are 8 internal nodes and the sum of the number of children per node is 17 (check the arrows), thus the average arity is: $17/8 = 2,125$.

Write the function `averageArity(`T`)` that returns the average arity of the a general tree T if $size(T) > 1$, otherwise 0, for *first child - right sibling* implementation.

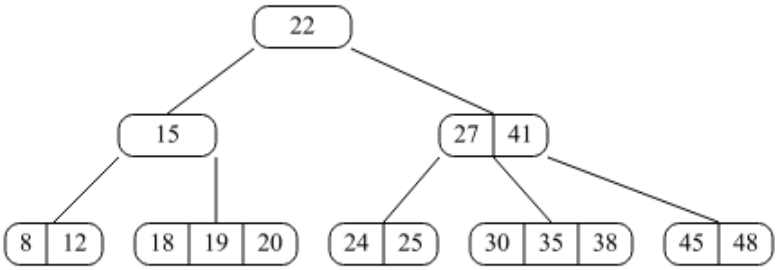

Figure 2: Btree with minimal degree 2

Exercise 3 (B-trees: Insertions – *8 points*)

1. Successively insert the keys 36 and 42 into the tree in figure 2 using the precautionary principle (in going down method). Draw **only** the tree that you get after each insertion.

2. Write the recursive function `__insert(`B`, `x`)` that inserts x in the nonempty B-tree B unless this element is already present.

   ▷ Use the precautionary principle (in going down).

   ▷ Use the split procedure with the following specifications:
   The procedure `split(`B`, `i`)` splits the child n°i of the tree B (with t the degree of the B-tree):
   - B is a nonempty tree and its root is not a $2t$-node.
   - The child i of B exists and its root is a $2t$-node.

   ▷ We assume the function `search_pos` is written. Its specifications:
   The function `search_pos(`L`, `x`)` searches for the value x in the nonempty list L. It returns x position in L if it is found, its "virtual" position (the one where it should be) otherwise.

   ▷ The function `__insert` will be called by the following function that returns the tree after the potential insertion and a boolean that tells if the insertion occurred:

```
    def insert(B, x):
        if B ==  None:
            return (BTree([x]), True)
        else:
            if B.nbkeys == 2 * B.degree - 1:     # root split
                B = BTree([], [B])
                split(B, 0)
            ok = __insert(B, x)
            return (B, ok)
```

Exercise 4 (B-Trees and Mystery – *2 points*)

```
    def build(nodes, degree):
        BTree.degree = degree
        B = BTree(nodes[0])
        end = B.nbkeys + 2
        q = Queue()
        q.enqueue((B, 1))
        while not q.isempty():
            (cur, pos) = q.dequeue()
            if pos < len(nodes):
                for i in range(pos, pos + cur.nbkeys + 1):
                    new = BTree(nodes[i])
                    cur.children.append(new)
                    q.enqueue((new, end))
                    end += new.nbkeys + 1
        return B
```

Which parameters have been passed to the function `build` to obtain the tree in figure 2?

Appendix

Trees: *First child - right sibling* implementation

The (general) trees we work on are the same as the ones in tutorials.

```
class TreeAsBin:
    def __init__(self, key, child=None, sibling=None):
        self.key = key
        self.child = child
        self.sibling = sibling
```

B-Trees

The B-trees we work on are the same as the ones in tutorials.

Reminder:

- The empty tree is `None`
- The non empty tree is an object of the class `BTree` assumed imported.
- `B.degree` is the degree (the order) of the B-Trees we deal with: it is a given constant!

```
class BTree:
    degree = t    # given constant
    def __init__(self, keys=None, children=None):
        self.keys = keys if keys else []
        self.children = children if children else []
    @property
    def nbkeys(self):
        return len(self.keys)
```

Authorised functions and methods

As usual: `len`, `range`.

The methods of the class `Queue`, assumed imported:

- `Queue()` returns a new queue ;
- q`.enqueue(`e`)` enqueues e in q ;
- q`.dequeue()` deletes and returns the first element of q ;
- q`.isempty()` tests whether q is empty.

Also, on lists:

```
>>> help(list.insert)
...    L.insert(index, object) -- insert object before index

>>> help(list.pop)
...    L.pop([index]) -> item -- remove and return item at index (default last).
    Raises IndexError if list is empty or index is out of range.

>>> help(list.append)
...    L.append(object) -> None -- append object to end
```

Your functions

You can write your own functions as long as they are documented (we have to know what they do).

In any case, the last written function should be the one which answers the question.