Algorithmics
Midterm #3 (C3)

Undergraduate 2^{nd} year - S3#
Epita

17 March 2021 - 9 : 30

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

Lecture

Exercise 1 (The final frontier – *2 points*)

Assume the following set of key E={data, kirk, neelix, odo, picard, q, quark, sisko, tuvok, worf} and the table 1 of hash values associated with each key of this set E. These values are lying between 0 and 10 ($m = 11$).

Table 1: Hash values

data	4
kirk	5
neelix	3
odo	1
picard	7
q	6
quark	2
sisko	7
tuvok	1
worf	7

Present the collision resolution for adding all the keys of the set E in the order of the table 1 (from `data` to `worf`):

1. using the linear probing principle with an offset coefficient $d = 3$;
2. using the hashing with separate chaining principle.

Exercise 2 (Representations – *3 points*)

Let G be a simple undirected graph of 9 vertices, numbered from 0 to 8, represented by the following adjacency lists:

0 : {1, 2, 6}

1 : {0, 3, 4}

2 : {0, 8}

3 : {1}

4 : {1, 6, 5, 7}

5 : {4}

6 : {0, 4, 8}

7 : {4, 8}

8 : {6, 7, 2}

1. Give the adjacency matrix of G.
2. **Properties:** is the graph G
   (a) connected?
   (b) complete?
3. Fill-in the array of the degrees of $G's$ vertices.

Tutorial

Exercise 3 (Interval – *3 points*)

Write the function `test_inter(`T`, `a`, `b`)` that checks whether the values of the a general tree T are in the interval $[a, b[$, for *first child - right sibling* implementation.

Exercise 4 (B-trees: Insertion – *7 points*)

We work here with B-trees that contain only non zero naturals.

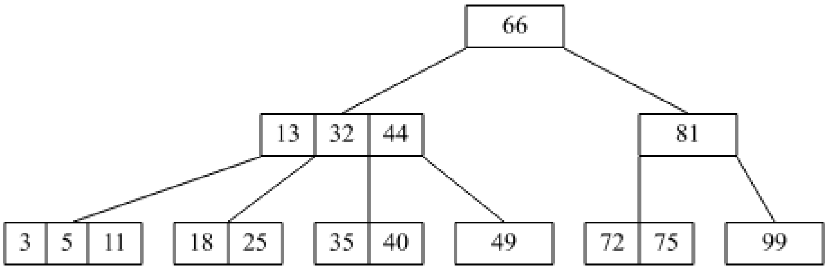

Figure 1: Btree with minimal degree 2

1. Using the "in going down" principle seen in tutorial (except bonus), draw the tree resulting from the insertion of the value 0 in the tree in figure 1.

2. Write the function `insert0(`B`)` that inserts la valeur 0 in the B-tree B. It returns the tree after insertion.

Exercise 5 (B-trees: Linear Representation – *5 points*)

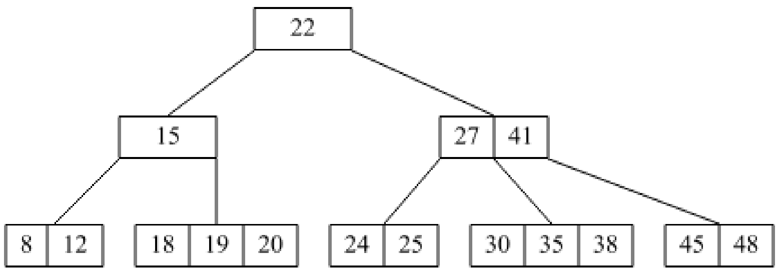

Figure 2: Btree with minimal degree 2

Remaining: A general tree $A = < o, A_1, A_2, ..., A_n >$ can be represented by $(o\ A_1\ A_2\ ...\ A_n)$.

With B-trees, a node o is represented as a list of its keys: $< x_1, ..., x_{k-1} >$.
For instance, the tree in figure 2 is represented by the string:

$$(< 22 > (< 15 > (< 8, 12 >)(< 18, 19, 20 >))(< 27, 41 > (< 24, 25 >)(< 30, 35, 38 >)(< 45, 48 >)))$$

Write the function `btree2list(`B`)` that builds the linear representation (of type `str`) of the B-tree B if not empty, the empty string otherwise.

Appendix

Trees

The (general) trees we work on are the same as the ones in tutorials.
***First child - right sibling* implementation**

- B: classe `TreeAsBin`
- `B.key`
- `B.child` : le premier fils
- `B.sibling` : le frère droit

B-Trees

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

- 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!
  - `B.keys` : liste des clés
  - `B.nbkeys` = `len(B.keys)`
  - `B.children` : liste des fils (`[]` pour les feuilles)
  - `B = BTree(key_list, child_list)` construit un nouvel arbre

Given functions

with t the degree of the B-trees

- The function `split(`B`, `i`)` splits the child n°i of the tree B:
  - 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.

Other authorised functions and methods

As usual: `len`, `range`, `min`, `max`, `abs`.

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.
```

str:

Reminder:

```
        >>> s = ""
        >>> s += str(42)
        >>> s
        '42'
```

Your functions

You can write your own functions as long as they are documented: give their specifications (we must know what they do).

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