Algorithmics
Midterm #3 (C3)

Undergraduate 2^{nd} year - S3
Epita

9 November 2020 - 13 : 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

WOW! WHAT A PROJECT!
COLOR THESE PICTURES! CUT AND PASTE! DRAW THOSE TREES! MORE CUTTING! MORE PASTING!
WHAT A LEARNING EXPERIENCE! YES, MA'AM, YOU'VE DONE IT AGAIN!
© 1997 United Feature Syndicate, Inc.
WHEN SHE'S HAPPY, WE'RE HAPPY..

Exercise 1 (Some different results – *5 points*)

1. Draw the 11-item hash table that results from using the hash function

$$h(x) = (2x + 5)\ mod\ 11,$$

to hash the key $12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5,$ assuming collisions are handled by coalesced hashing.

2. Draw the 11-item hash table that results from the previous question assuming collisions are handled by linear probing.

3. Draw the 11-item hash table that results from the first question assuming collision are handled by double hashing using a secondary hash function

$$d(x) = 7 - (x\ mod\ 7)$$

that combined with $h(x)$ gives the following table 1 of probe sequences associated with each key.

Table 1: probe sequence values

Elt	h(x)	2	3	4	5	6	7	8	9	10	11
12	7	9	0	2	4	6	8	10	1	3	5
44	5	10	4	9	3	8	2	7	1	6	0
13	9	10	0	1	2	3	4	5	6	7	8
88	5	8	0	3	6	9	1	4	7	10	2
23	7	1	6	0	5	10	4	9	3	8	2
94	6	10	3	7	0	4	8	1	5	9	2
11	5	8	0	3	6	9	1	4	7	10	2
39	6	9	1	4	7	10	2	5	8	0	3
20	1	2	3	4	5	6	7	8	9	10	0
16	4	9	3	8	2	7	1	6	0	5	10
5	4	6	8	10	1	3	5	7	9	0	2

Exercise 2 (Find the sum – *4 points*)

Write the function `find_sum(`B`, `sum`)` that tests if there exists a branch in the tree B (`TreeAsBin`) such that the sum of its values (integers) is equal to sum.

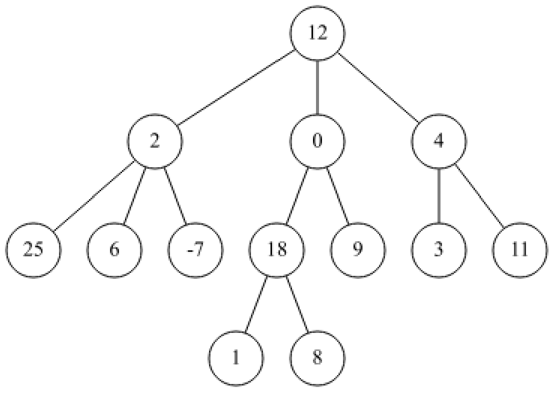
Application examples:

```
>>> find_sum(T4, 20)
True

>>> find_sum(T4, 10)
False

>>> find_sum(T4, 7)
True

>>> find_sum(T4, 17)
False
```



Tree T4

Exercise 3 (Maximum Gap – *4 points*)

For each node of a B-tree, we call *gap* the maximum difference between two successive keys. The *maximum gap of a B-tree* is the maximum *gap* of its nodes.

For instance, in the tree in figure 1, the maximum gap is 15 (35 - 20).
Write the function `maxgap` that computes the maximum gap of a B-tree. The function returns 0 for an empty tree.

Exercise 4 (What? – *4 points*)

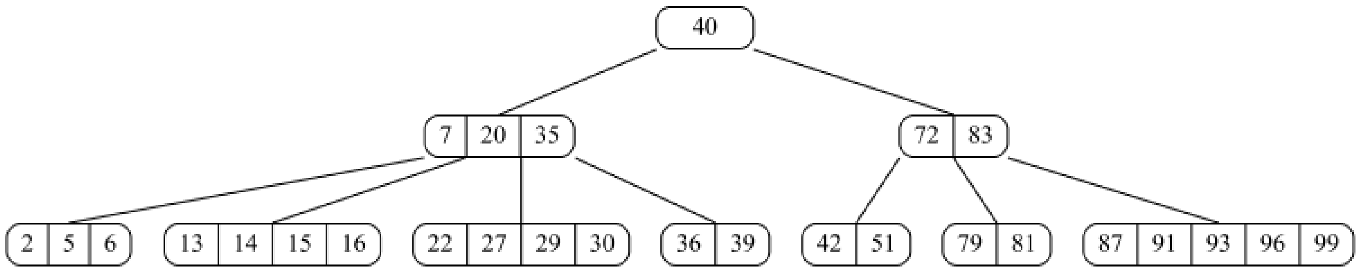


Figure 1: B-tree of degree 3

Let `what` be defined below:

```python
def __what(B, x, y):
    i = 0
    while i < B.nbkeys and B.keys[i] <= x:
        i += 1
    if i < B.nbkeys:
        y = B.keys[i]
    if B.children == []:
        return y
    else:
        return __what(B.children[i], x, y)

def what(B, x):
    if B == None:
        return None
    else:
        return __what(B, x, None)
```

1. Let B_3 be the tree in figure 1. What will be the results of the following applications?

   - what(B_3, 2)
   - what(B_3, 7)
   - what(B_3, 18)
   - what(B_3, 39)
   - what(B_3, 41)
   - what(B_3, 99)

2. Let B be a non empty B-tree and x an integer. What does what(B, x) return?

Exercise 5 (B-tree: insertion and deletion – *3 points*)

1. Using the "in going down" principle seen in tutorial (except bonus), draw the tree resulting from the insertion of the value 39 in the tree in figure 2.

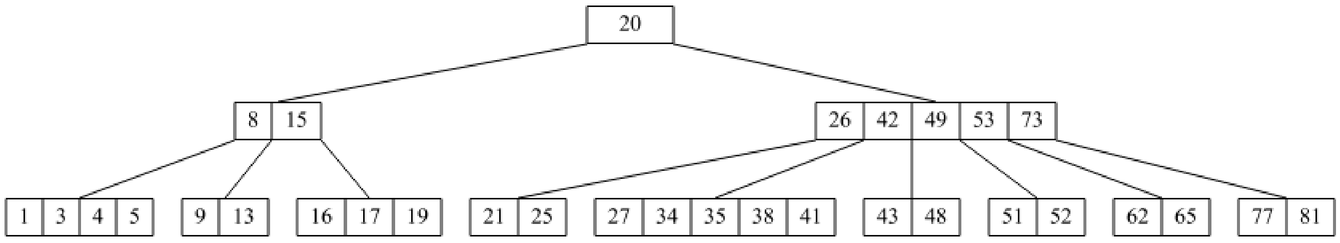


Figure 2: B-tree for insertion

2. Using the "in going down" principle seen in tutorial (except bonus), draw the tree resulting from the deletion of the value 72 in the tree in figure 3.

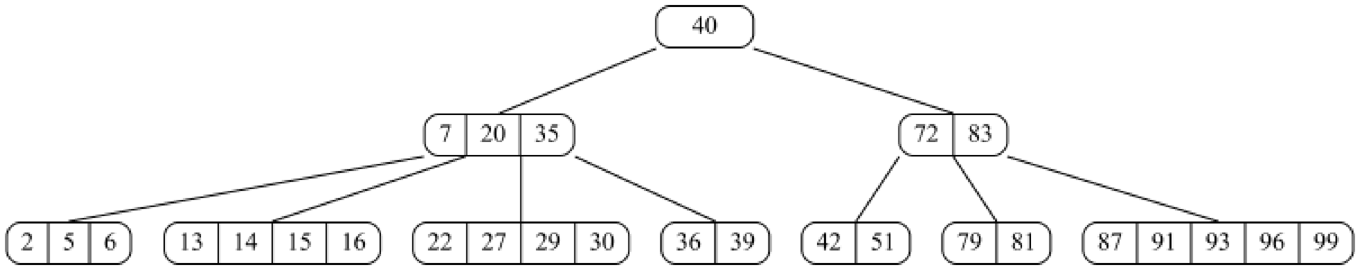


Figure 3: B-tree for deletion

Appendix

Trees: *First child - right sibling* implementation

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

```python
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!

```python
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`, `min`, `max`, `abs`.

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.