Binary Trees (Arbres binaires)

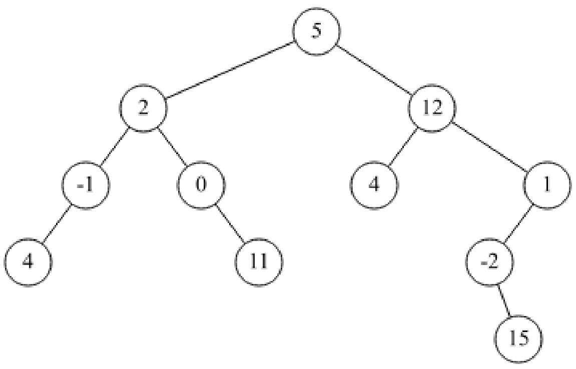

Figure 1:

1 To recurse is divine

Exercise 1.1 (Size (Taille))

1. Using the abstract data type `binary tree`, give the axioms that define the *size* operation.
2. Write a function that calculates the size of a binary tree.

Exercise 1.2 (Height (Hauteur))

1. Using the abstract data type `binary tree`, give the axioms that define the *height* operation.
2. Write a function that calculates the height of a binary tree.

Exercise 1.3 (Depth-First Search (Parcours profondeur))

1. Give the three orders induced by the depth-first search (beginning on the left side) of the tree in figure 2.
2. Give the $<r, G, D>$ form of the tree in figure 2 with _ to represent the empty tree.
3. Write a function that displays the $<r, G, D>$ form of a binary tree with _ to represent the empty tree.

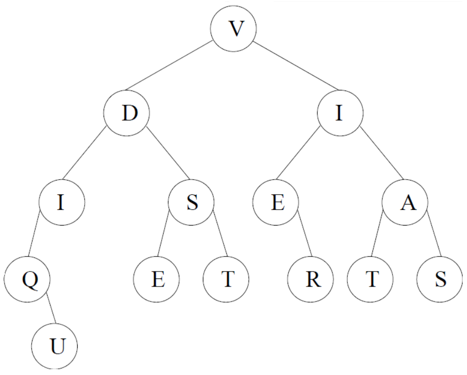

Figure 2: Binary tree for traversals

Exercise 1.4 (Serialization: to save trees in files)

In order to save binary trees in text files, a unique "linear" representation must be found. It is based on the abstract algebraic type representation.

- The empty tree is represented by `"()"`
- The non empty tree $< r, L, R >$ is represented by the string `"(rLR)"` with `L` and `R` the linear representations of L and R.

Simple example:

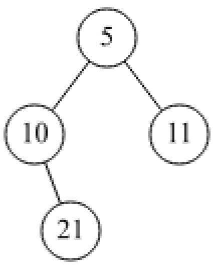

Figure 3: B_2 : `"(5(10()(21()()))(11()()))"`

1. Give the linear representations of the following trees:

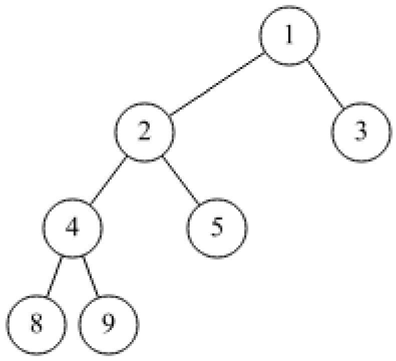

Figure 4: B_4

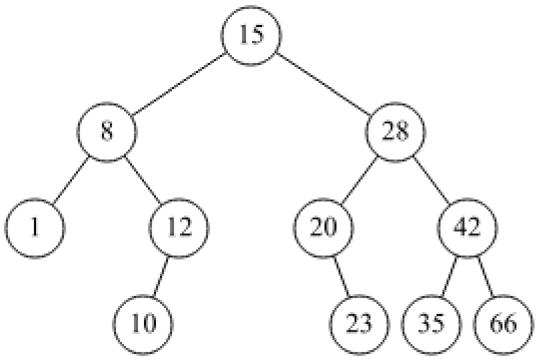

Figure 5: B_5

2. Draw the trees whose linear representations are as follows:
   (a) `"(5(4(11(0()())(2()()))())(8(13()())()))"`
   (b) `"(15(8(5()(7()()))(12(10()())()))(25(20()())(42()(66()()))))"`
3. Write the function `to_linear(`B`)` that returns the linear representation of the tree B (a string).

Exercise 1.5 (Path Lengths and Average Depths)

1. Consider the following functions:

```
function fun_rec(binarytree B, integer h,  ref integer n) : integer
begin
    if B = emptytree then
        return 0
    else
         n ← n + 1
         return h + fun_rec(l(B), h+1, n) + fun_rec(r(B), h+1, n)
    end if
end
```

```
function fun(binarytree B) : real
variables
   integer  nb
begin
    if B = emptytree then
         /* Exception */
    else
        nb ← 0
        return (fun_rec (B, 0, nb) / nb)
    end if
end
```

   (a) What is the result of the application of the function `fun` to the tree in figure 2?
   (b) What mesure does this function calculate?
   (c) Write this function in Python.

2. What has to be modified in this function to calculate the external average depth?

2 Breadth-first Search

Exercise 2.1 (Breadth-first Search (Parcours largeur) (BFS))

1. Run through the breadth-first traversal algorithm on the tree in figure 2.
2. Write a function that displays the keys of a binary tree in hierarchical order.

Exercise 2.2 (Weighted average width – *Contrôle 2 - 2021-03*)

Write the function `get_average(B)` that takes as a parameter a binary tree `B` and that builds and returns a list `L` such that:

- `len(L)` = number of levels of the binary tree `B`
- `L[i]` = sum of the keys of the nodes at level `i` divided by the number of nodes at level `i`

Application example on the tree in figure 6:

```
>>> get_average(B)
[0.0, 3.0, 4.75, 6.666666666666667]
```

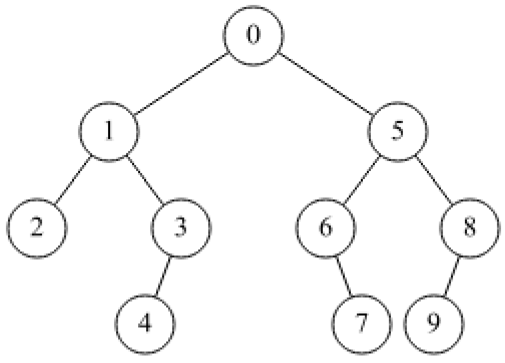

Figure 6: Binary tree *B*

3 Occurrences and Hierarchical Numbering

Hierarchical Numbering (Numérotation hiérarchique)

Exercise 3.1 (Hierarchical implementation)

How to represent / implement a binary tree with a simple vector?

1. Give the array that represents the tree in figure 1.
2. What has to be modified in the traversals (both depth and width) when the tree is given as a vector ("hierarchical" implementation)?

Exercise 3.2 (Object → List)

Write a function that builds the vector (a list in Python) containing the hierarchical numbering of a tree ("object" implementation: `BinTree`). The value `None` will be used to reference an empty tree.

Occurrences

Exercise 3.3 (Occurrence List)

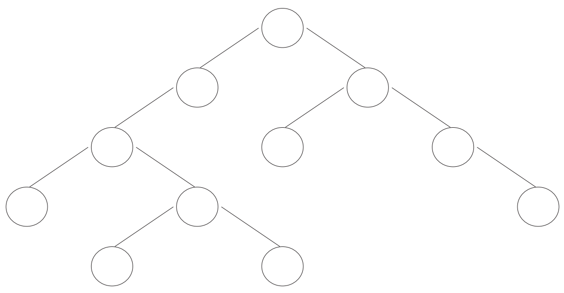

Figure 7: Binary tree for occurrences

1. Give the occurrence list representation of the tree in figure 7.
2. Write the function that returns the occurrence representation of a binary tree (a string list).

Exercise 3.4 (Binary tree and prefix code)

Compressing text files can be done by encoding characters with binary words. We use here a variable-length code: each character can be encoded using a different number of bits. Obviously we want frequent characters to use short codes, while long codes should be reserved for infrequent characters.
Also our encoding must have the prefix property: no code should be prefix of another code. If we do not respect this property, for instance encoding 'a' with 11, and 'b' with 111, then we cannot tell if 11111 encodes "ab" or "ba".

1. Consider the following encoding:

letter	a	f	H	m	n	u
code	0	1100	111	1101	101	100

   Decode 11110011001100110101 01

2. The encoding is represented by a tree whose leaves are the letters to encode (the field `key` contains the letter). The code of a letter can be deduced from the path from the root to the leaf that contains this letter: each left branch represents a 0, and each right branch is a 1.
   What does the code of a letter correspond to?
3. Draw the tree representing the encoding of question 1.
4. The tree representing the encoding is a full (proper) tree in which each leaf is used. Write a function that returns the code of a given letter if it exists in the tree. For instance, with the encoding of question 1, if the letter is 'm', the function returns "1101".

4 Tests

Exercise 4.1 (Degenerate, complete, perfect)

1. **Degenerate tree (*dégénéré*):**
   (a) What is a degenerate tree?
   (b) How to verify if a tree is degenerate knowing its size and height?
   (c) Write a function that tests if a tree is degenerate without using `size` neither `height`.
2. **Perfect tree (*complet*) :**
   (a) Give several definitions of a perfect tree.
   (b) How to verify if a tree is perfect knowing its size and height?
   (c) Write a function that tests if a tree is complete (you can use the function `height`, only once).
   (d) Write again the test function without using `height`.
3. **Complete tree (*parfait*) :**
   (a) What is a complete tree?
   (b) **Bonus:** Write a function that tests if a tree is complete?

5 Construction

Exercise 5.1 (Parent and children)

The class `BinTreeParent` described below enables us to represent binary trees where each node has links to its children but also a link to its parent (`None` for the root).

```
class BinTreeParent:
    def __init__(self, key, parent, left, right):
        self.key = key
        self.parent = parent
        self.left = left
        self.right = right
```

Write the function `copywithparent` that builds from the "classic" binary tree B (`BinTree`) the equivalent tree (with same values at same places) leaving at each node a link to its parent (`BinTreeParent`).

Exercise 5.2 (List → Object)

Write a function that builds a binary tree ("object" implementation: `BinTree`) from the vector (a list in Python) containing its hierarchical numbering. The value `None` is used to reference an empty tree.

Exercise 5.3 (BONUS : Load binary trees)

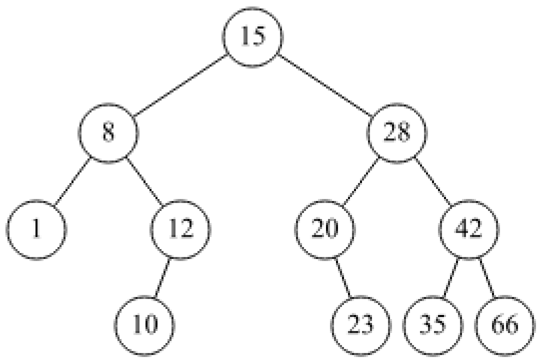

Figure 8: `files/bst.bintree`

The above tree was loaded from a text file that contains its linear representation:

```
(15(8(1()())(12(10()())()))(28(20()(23()()))(42(35()())(66()()))))
```

Write the function `load` that builds a binary tree from such a text file.

Bonus: Do not forget to test if the string in the file is "well-formed"!