General Trees (Arbres généraux)

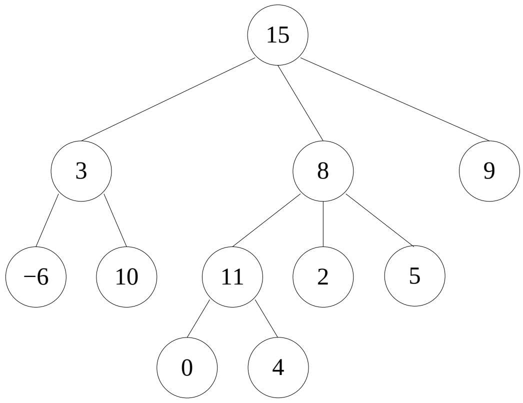


Figure 1: General Tree T_1

1 Measures

Exercise 1.1 (Size)

1. Give the definition of the size of a tree.
2. Write a function that returns the size of a tree, for both implementations:
   (a) by tuples (each node contains a tuples of children);
   (b) left child - right sibling (as a binary tree).

Exercise 1.2 (Height)

1. Give the definition of the height of a tree.
2. Write a function that returns the height of a tree, for both implementations:
   (a) by tuples;
   (b) left child - right sibling.

Exercise 1.3 (External Path Length)

1. Give the definition of external path length of a tree.
2. Write a function that returns the external path length of a tree, for both implementations:
   (a) by tuples;
   (b) left child - right sibling.

---

2 Traversals

Exercise 2.1 (Depth First Traversal)

1. Give the principle of a depth-first traversal for a general tree.
2. List elements in prefix and suffix orders for the depth-first traversal of the tree in figure 1. What other action can be done when visiting a node ?
3. Write a template depth-first traversal algorithm (insert node actions as comments) for both implementations:
   (a) by tuples;
   (b) left child - right sibling.

Exercise 2.2 (Breadth First Traversal)

1. Give the principle of a breadth-first traversal for a tree.
2. How can we detect level changes during the traversal?
3. Write an algorithm that displays every keys, with each level on its own line, for both implementations:
   (a) by tuples;
   (b) left child - right sibling.

---

3 Different representations

Exercise 3.1 (Prefix - Suffix – *C3 - Nov. 2015*)

The aim here is to fill a vector with the keys of a general tree. Each key is put **twice** in the vector: at the first encounter (prefix order) and at the second encounter (suffix order) during the depth first traversal.

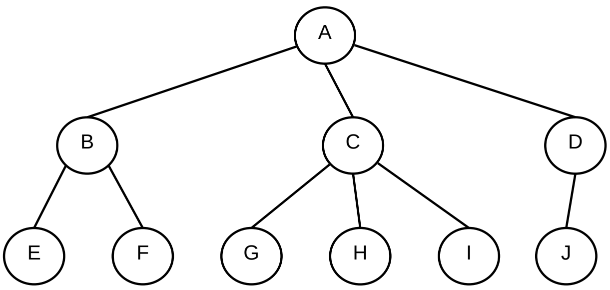


Figure 2: General Tree T_2

After the depth first traversal of the tree in figure 2, the array will be filled as follows :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	B	E	E	F	F	B	C	G	G	H	H	I	I	C	D	J	J	D	A

Write a function which, from a tree, builds the corresponding key vector (represented as a list in Python) according to the described order, for both implementations:

(a) by tuples; (b) left child - right sibling.

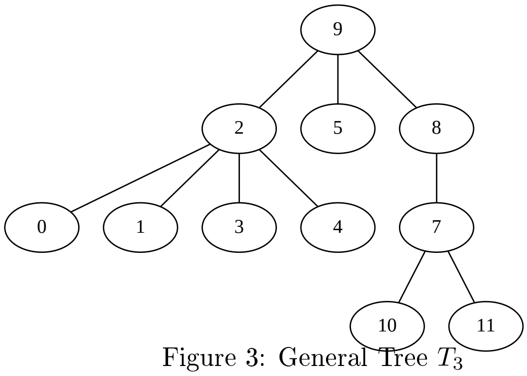


Figure 3: General Tree T_3

Exercise 3.2 (Serialization – *Nov. C3 - 2014*)

We shall now study an alternative representation for general trees: parent vectors. This implementation is linear and thus can be used to store trees in files (serialization.)

This representation is pretty simple. For each node of the tree, we associate a unique identifier: an integer in the range from 0 to the size of the tree - 1. We then build a vector where the cell i contains the identifier of the immediate parent for the node of identifier i. The parent of the root is -1.

1. Give the parent vector for the tree in figure 3.
2. Write a function which, from a tree, fills the corresponding parent vector (represented as a list in Python), for both implementations:

   (a) by tuples; (b) left child - right sibling.

Exercise 3.3 (Tuples ↔ left child - right sibling)

1. Write a function that builds, from a general tree with left child - right sibling implementation (*i.e.* a binary tree), its "by tuples" implementation.
2. Write the translation function for the other way.

Exercise 3.4 (List Representation)

Let A be the general tree $A =< o, A_1, A_2, ..., A_N >$. The following linear representation of A (o A_1 A_2 ... A_N) is called *list*.

1. (a) Give the linear representation of the tree in figure 1.

   (b) Draw the tree corresponding to the *list* `(12(2(25)(6)(-7))(0(18(1)(8))(9))(4(3)(11)))`.
2. Write the function that builds the linear representation (as a string) from a tree, for both implementations:

   (a) by tuples; (b) left child - right sibling.

   What has to be change to obtain an "abstract type" like representation ($A =< o, A_1, A_2, ..., A_N >$)?

Bonus Write the reciprocal algorithm: that builds the tree (with both implementations) from the *list*.

Exercise 3.5 (`dot` format)

A tree can be represented as a list of links (like a graph): the *dot* format.

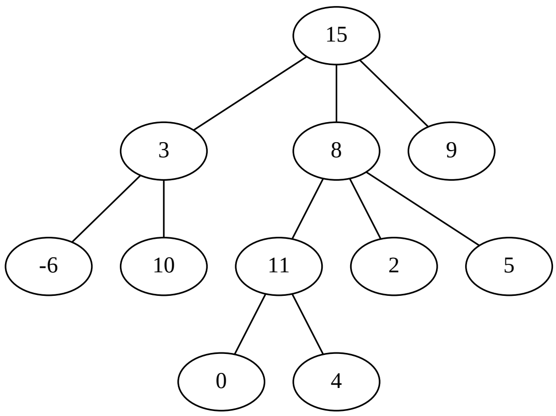
```
graph {
    15 -- 3;
    15 -- 8;
    15 -- 9;
    3 -- -6;
    3 -- 10;
    8 -- 11;
    8 -- 2;
    8 -- 5;
    11 -- 0;
    11 -- 4;
}
```

Another possibility:

```
graph {
    15 -- {3; 8; 9};
    3 -- {-6; 10};
    8 -- {11; 2; 5};
    11 -- {0; 4};
}
```

';' can be omitted.

If you want to see the graphical representation of your tree, use "Graphviz".
Warning: according to the order of links, the result will not be the same. **The order given here is the appropriate one for a tree.**



Write the functions that allow to:

- build the `.dot` file from a tree (for both implementations)
- and in return, build a tree (in both implementations) from a `.dot` file.