Université de Bordeaux Licence S&TS Mention Info et MI, S4 2018/2019

Algorithmique des structures de données arborescentes

TP Noté — 15 avril 2019 — duration 1h20

In this test, we use the following `'a tree` type:

```
type 'a tree = Empty | Bin of 'a * 'a tree * 'a tree
```

We consider *enriched* binary trees: a tree is enriched if each of its nodes is labeled by a pair (x, y), where y is the **number of nodes** of the subtree rooted at this node. Thus, an enriched tree can be represented by an object of type `('a * int) tree`.

An example is given on Figure 1. The leftmost tree is a tree whose nodes are labeled by an integer. For example, the label of the root is 42. The rightmost tree is the corresponding enriched tree. Together with the original key, each node has the **number of nodes** of its subtree (as its second component). For example, the subtree rooted at the node labeled 84 has **3** nodes (the node labeled 84 itself, the one labeled 0, and the one labeled 99).

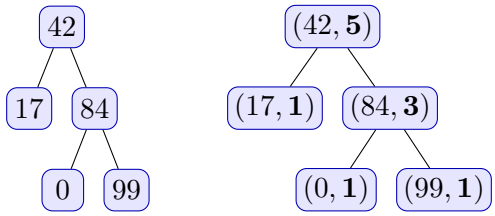


Figure 1: A tree and the corresponding enriched tree

Note that in an enriched tree, whenever a node is labeled (x, y), the first component x is arbitrary. It can be itself a pair (c, m), where (for example) m is a multiplicity. Thus, the rightmost tree of the following figure is also the enriched version of the leftmost tree.

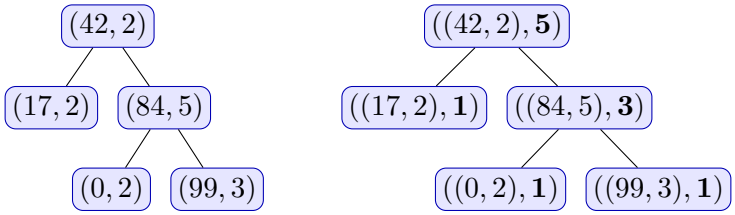


Figure 2: Another tree and its enriched version

Exercice 1 Write a function `size : ('a * int) tree -> int` outputting the number of nodes of the enriched tree given as parameter.

Important note. This function is simple and non-recursive. We ask for a complexity $O(1)$. It can then be used in the following exercises.

Exercice 2 Write a function `enrich : 'a tree -> ('a * int) tree` outputting the enriched tree corresponding to its argument. For example, if we apply the function on the leftmost tree of Figure 1, we obtain the rightmost one (the same holds for the trees of Figure 2).

We ask for a function of **linear complexity** with respect to the input tree; we suggest that your function considers each node only once.

Exercice 3 We represent multisets as enriched binary search trees. Each node contains an element of the form `((cle, mult), size)`, where `cle` is an element of the multiset, `mult` its multiplicity, and `size` the size of the subtree rooted at this node. Thus, a multiset is represented by an object of type `(('a * int) * int) tree`. This is exactly the situation described in Figure 2.

Write a function

```
insert : 'a -> (('a * int) * int) tree -> (('a * int) * int) tree
```

taking as input a key k and an enriched tree representing a multiset t and returning the enriched tree representing the multiset obtained by inserting k in t.

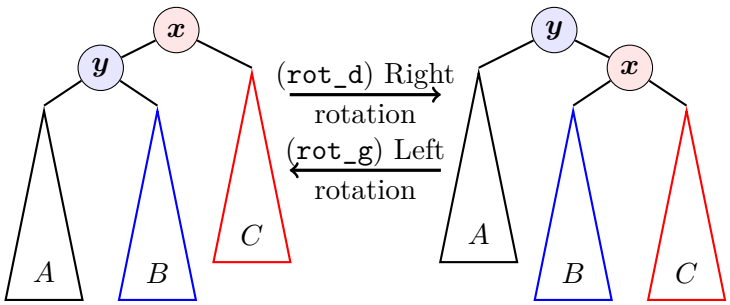
The complexity of this function has to be the same as the insertion in a regular binary search tree.

Notes. The input tree, as well as the returned one, has to be a BST. Moreover, to obtain the requested complexity, you will have to produce an enriched tree by recomputing the sizes of **only some** subtrees.

Exercice 4 Write two functions `rot_g` and `rot_d` of type

```
('a * int) tree -> ('a * int) tree
```

applying the following transformations on the enriched search tree given as input:



The result has to remain an enriched search tree. If the tree does not have the requested shape, your function will output the input without modifying it.

Note. The functions are simple and non-recursive. Do not forget to make the necessary changes on the "size" component of each node.

Exercice 5 With these two functions, write a function `remove` of type

```
'a -> (('a * int) * int) tree -> (('a * int) * int) tree
```

taking a key k and a multiset t as input, and returning the multiset obtained by removing an occurrence of k in t (if there is at least one), and using the following strategy.

To remove a node of key k whose subtrees G and D are non empty, you will use a rotation to move the node **down**, towards the leaves, while preserving the enriched search tree structure. You will use a right rotation when G has at least as many nodes as D, and a left one otherwise.

To remove a node having at least one empty subtree, use the same algorithm as the one for usual binary search trees – while preserving the enriched tree structure.