Advanced Data Structures ans Algorithms

ESILV S7

Tutorial 4 - Binary Search Tree and AVL
2020 - 2021

Exercice 1

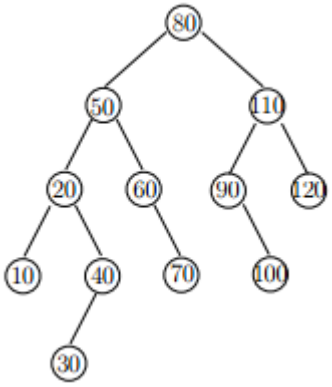
- Insert the following sequence of elements into an AVL tree : 10, 20, 15, 25, 30, 16, 18, 19.
- Delete 30 in the AVL tree that you got.

Exercice 2

Inserting elements one by one, build the AVL tree with elements : 7, 10, 15, 8, 9, 13.

Exercice 3

Give the resulting AVL-tree after inserting 65 into the following AVL-tree.



Exercice 4

Write an algorithm that checks whether a binary search tree given as an argument meets all the requirements to be an AVL tree.

Exercice 5

Write an algorithm that adds a node (whose value is given as an argument) to an AVL tree (also given as an argument). Of course, you must balance the tree if necessary to continue to meet the requirements of AVL trees. You must be careful to handle properly the potential special cases.

Exercice 6

Write an algorithm that deletes a node (whose value is given as an argument) of an AVL tree (also given as an argument). Of course, you must balance the tree if necessary to continue to meet the requirements of AVL trees. You must be careful to handle properly the potential special cases.

Exercice 7

Write an efficient algorithm (that run in linear time) that takes as argument an AVL tree and returns an array of integers sorted in ascending order that contains the elements of this tree.

Exercice 8

Write an efficient algorithm (that run in linear time) that takes as argument an array of integers sorted in ascending order and returns an AVL tree that contains the elements of this array.

Exercice 9

Write an algorithm that takes as arguments two AVL trees : A and B such that the largest element in A is less than the smallest element in B. This algorithm should return an AVL tree that contains the elements of the two inputs (beware, you must not reason about the values of these elements ; at the end there are as many elements in the final AVL tree that there are in A and B). Note that your algorithm does not have to preserve the trees given as inputs. Note Firstly, you can imagine a naive solution (with a time complexity of O(n)) or even very naive (with a time complexity of O(n log n), or worse. . .). Then, try to get an efficient one (with a time complexity of O(log n) and no extra space)