The Tree of Life

SD : Structures de Données

Première année

Trees are among the most fascinating, interesting and useful Data Structures in Computer Science. When well built, they have incredible features for storing, searching and manipulating data sets. In Computer Science, as in mother nature, there exist multiple tree species and variants : Binary Trees, Ternary Trees, Binary Search Tree, AVL-Trees, Heaps, B-Trees, 2-3-4 Trees, Radix-Tree, Van Emde-Boas Trees, Enfilades, . . . All have their own beauty, characteristics, features, advantages, limits and favorite application domains. All do also implement the basic operations on dynamic sets: insert, delete, search, minimum, maximum, . . .

The exercises defined below, followed by a lab have been designed to deepen your knowledge on Tree Data Structures and their processing. It complements the main lecture by :

— manipulating concrete data sets with trees,
— writing a set of operations on those trees (insert, delete, traversal, . . .),
— gradually writing advanced operations to maintain given properties on trees,
— implement trees in the C language to get a glimpse of their internal representations,
— empirically approximate major features on BST and AVL trees.

★ Exercice 1. *"Promenons nous dans les bois . . . "*

Lets start by growing trees. In this exercise, we will work on Binary Search Trees, i.e. trees that are . . . binary :-) and which comply to the constrain given in the definition below.

Définition 1. For every node x in the BST, all the values of its left subtree are smaller or equal to the value hold by x and all the values in its right subtree are greater or equal to the value hold by node x.

▷ **Question 1.** For the following set of keys :

16	1	5	21	4	10	32	65

, draw binary search trees of height 7,6,5,4,3 and 2.

▷ **Question 2.** For the following set of keys :

16	1	5	21	4	32	65

, build the corresponding BST assuming entries are processed from the left to the right.

▷ **Question 3.** Define a data structure that represents a node of (and by transitivity) a binary tree.

▷ **Question 4.** Write a function that inserts a value in a BST.

▷ **Question 5.** Write a function that computes the height of a binary tree.

▷ **Question 6.** Write in pseudo-code a function that prints the sorted values of a BST in an increasing order. The function shall have the following signature `PRINT_ORDERED_BST(Tree Node)`.

▷ **Question 7.** What is the complexity of the above defined function. Provide a proof.

▷ **Question 8.** Write a function that searches the minimum value of a given node in a BST. What is its complexity ?

▷ **Question 9.** Write a function that searches the successor value of a given node in a BST.

▷ **Question 10.** Write a function that counts the number of nodes in a BST.

▷ **Question 11.** Write a function that returns the sum of all values in a binary tree.

★ Exercice 2. Lets get balanced

As we saw in the previous exercise, BSTs can take multiple forms that for a given set of input data can seriously impact the structure of the tree and thus its height. The height itself will impact the complexity of the different operations on the BST and can in the worst case, completely anihilate the gain (i.e. degrade the complexity from $O(log(n))$ to $O(n)$. To avoid this degradation, several extensions to BSTs provide algorithmic solutions to keep the tree balanced after every operation.

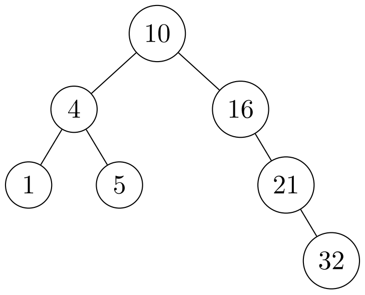
One of these solutions called AVL trees was developed by Adelson-Velsky and Landis in 1962. An AVL tree is a BST tree that complies to the following property:

for each node x of an AVL tree, the height of the left and right subtrees of x differ by at most 1.

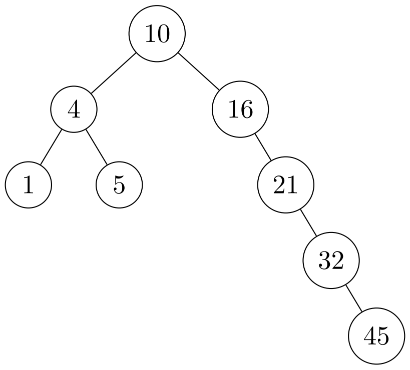
The exciting part of AVL trees is to define how we can maintain this property dynamically. This is what we will do in this second exercise.

▷ **Question 12.** Compute the balance of a BST (each of his node).

▷ **Question 13.** Compute the balance attribute for each of the nodes below. Is the tree balanced ?

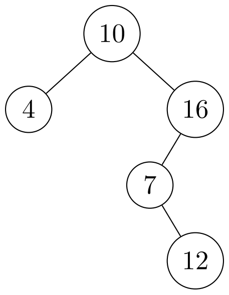


▷ **Question 14.** Given the following BST, what can be done to get it balanced ?



▷ **Question 15.** Based on the above experiment, write a `ROTATE_LEFT(x)` function that, given the root of the subtree to rotate, (in the example above, node 21), changes the tree. For convenience, we do DS augmentation by adding a "parent" field which allows a node to access its parent directly. You can then write the symetric function `ROTATE_RIGHT(x)`.

▷ **Question 16.** Given the following BST, what can be done to get it balanced ?



▷ **Question 17.** Starting with the table given in question 2 of exercise 1, rebuild the AVL tree by systematically applying the balance operation after each node addition.

▷ **Question 18.** Generalize your previous experiment to write a node addition function that also maintains the AVL property.

★ **Exercice 3.** The Lab (Part 1)

The lab is straight-forward for those who did complete part 1 as part of their devotion to algorithms and investment in homework. It basically consists in implementing in C a BST and validating the algorithms that were defined in the first exercise of this sheet. This is our quest.

▷ **Question 19.** Implement in C a BST structure and fill it with 8 integers of your choice.

▷ **Question 20.** Implement the functions defined above on BSTs

★ **Exercice 4.** The Lab (Part 2)

▷ **Question 21.** Write a function that generates 1000 distinct random positive integer values and fills them in a table. To do the generation, please look and use the `srand()` C function.

▷ **Question 22.** Following a FIFO policy, generate the corresponding BST and compute its height.

▷ **Question 23.** Repeat the above experiment with 50+ experiments. How does the average height relate to the number of items?

▷ **Question 24.** For those who are still in a good mood and shape, extend your code so as to make your node insertion function AVL friendly. Replay the experiments with the new algorithm and estimate the link between the average height and the number of nodes.

▷ **Question 25.** For the super-top scientists in the room, compute the ratio between the theoretical tree size (in number of nodes related to its height) and the real number of node in built trees.