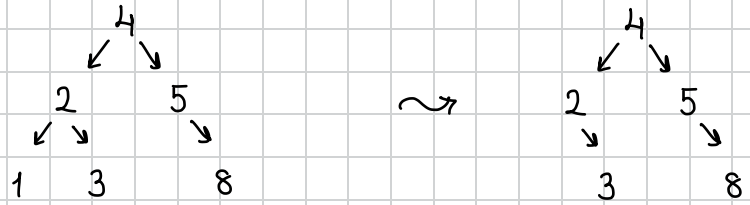
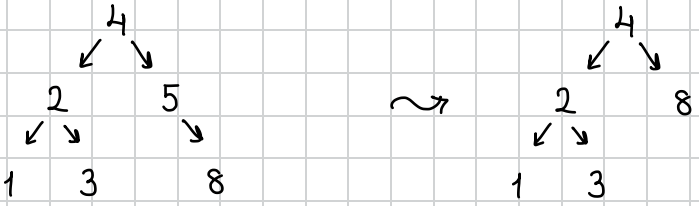
BST - remove (x)

Case 1: remove (1)

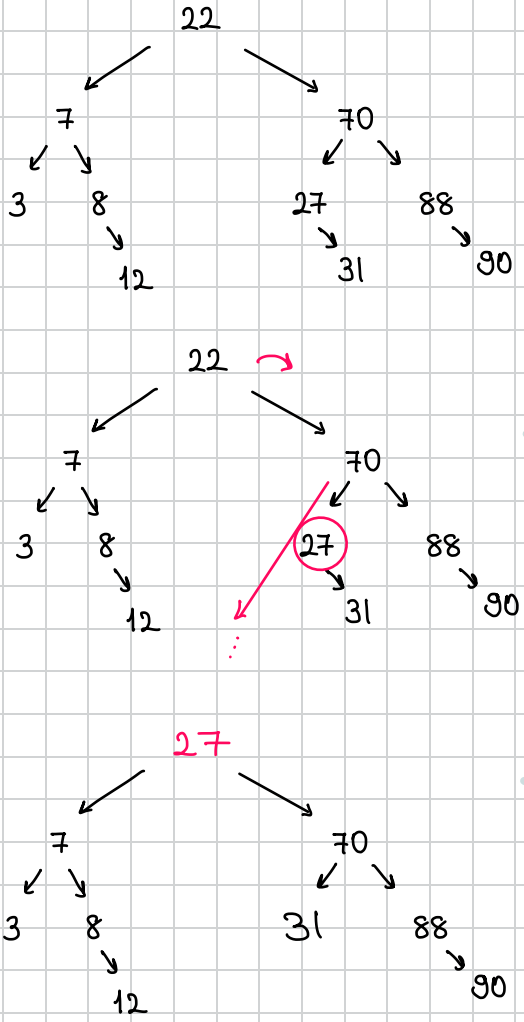


Case 2: remove (5)



Case 3: remove (2) / bigger BST

remove (22):



★ Find the next biggest element !

How?

walk once to the right and all the way to the left

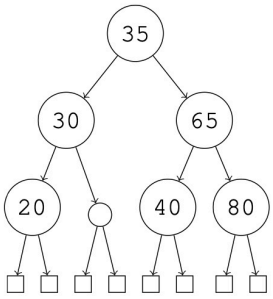
Or: find the previous largest element walk once to the left and all the way to the right

Note: will be case 1 or 2

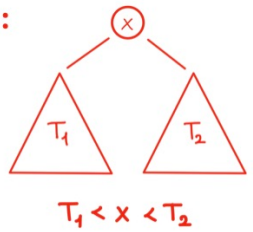
We "remove (27)" and put it in the place of 22.

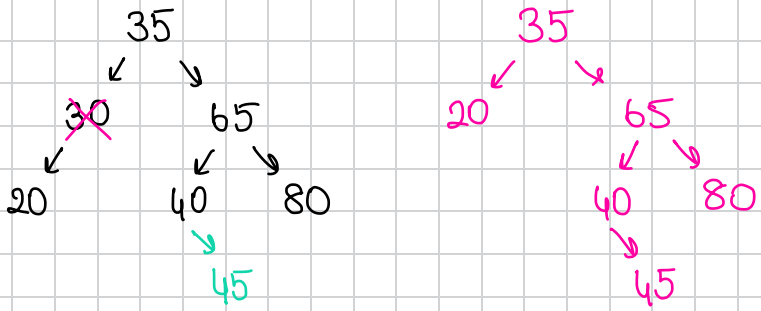
BST + AVL - exam questions

i) Draw the **binary search tree** obtained from the following tree by performing the two operations `INSERT(45)` and `DELETE(30)`, in that order.



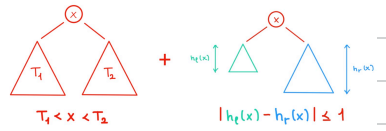
BST Condition :

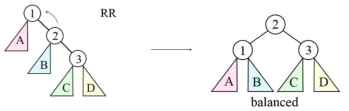


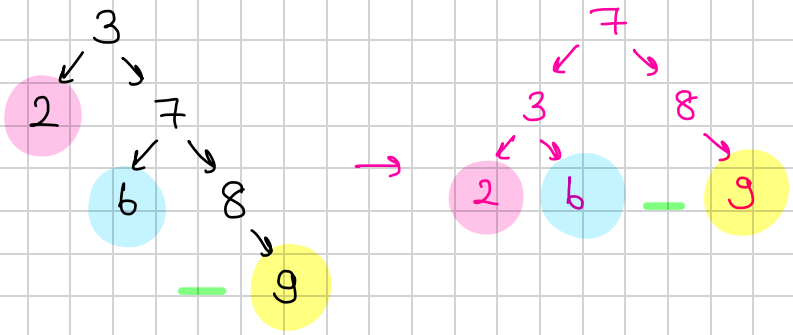


ii) Draw the **AVL tree** that is obtained when inserting the keys $3, 2, 7, 6, 8, 9$ in this order into an empty tree (it suffices to draw only the final tree).

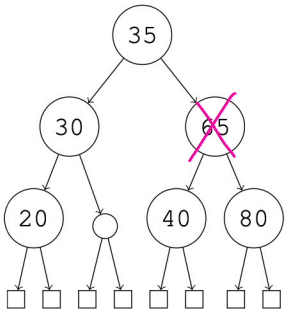
AVL-Tree Condition :



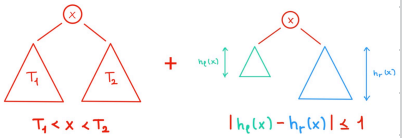




iii) Draw the **AVL tree** that is obtained by deleting key 65 from the tree below.



AVL-Tree Condition :



DP - Introduction

Consider the recurrence

$$A_1 = 1$$
$$A_2 = 2$$
$$A_3 = 3$$
$$A_4 = 4$$
$$A_n = A_{n-1} + A_{n-3} + 2A_{n-4} \text{ for } n \geq 5.$$

Compute A_n using bottom-up dynamic programming and state the run time of your algorithm. Address the following aspects in your solution:

(1) *Definition of the DP table:* What are the dimensions of the table $DP[\ldots]$? What is the meaning of each entry?

(2) *Computation of an entry:* How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.

(3) *Calculation order:* In which order can entries be computed so that values needed for each entry have been determined in previous steps?

(4) *Extracting the solution:* How can the final solution be extracted once the table has been filled?

(5) *Run time:* What is the run time of your solution?

0) Dimensions of the DP table: $DP[1 \ldots n]$

1) Definition of the DP table: $DP[i] = A_i$ for $1 \leq i \leq n$

2) Computation of an entry:

Initialization: $DP[1] = 1$, $DP[2] = 2$, $DP[3] = 3$, $DP[4] = 4$

Recursion: $DP[k] = DP[k-1] + DP[k-3] + 2 \cdot DP[k-4]$ for $k \geq 5$

3) Calculation Order:

We calculate with increasing i

4) Extracting Solution: The result is at: $DP[n]$

5) Runtime:

Each entry can be computed in: $\Theta(1)$

We're filling a DP table of size: n

So the runtime is $\Theta(n)$