Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department of Computer Science
Markus Püschel, David Steurer
Gleb Novikov, Tommaso d'Orsi, Ulysse Schaller, Rajai Nasser

22. November 2021

Algorithms & Data Structures Exercise sheet 9 HS 21

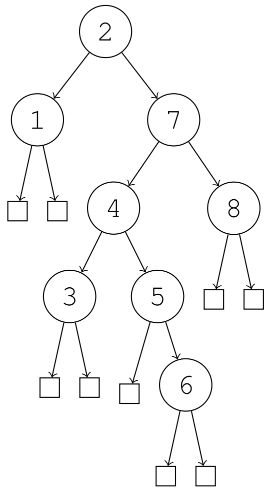
Exercise Class (Room & TA):	
Submitted by:	
Peer Feedback by:	
Points:	

Submission: On Monday, 29 November 2021, hand in your solution to your TA *before* the exercise class starts. Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

Exercise 9.1 *Search Trees* **(2 points)**.

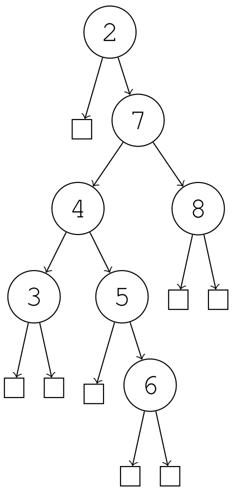
a) Draw the resulting tree when the keys 2, 7, 8, 4, 5, 6, 3, 1 in this order are inserted into an initially empty binary (natural) search tree.

Solution:

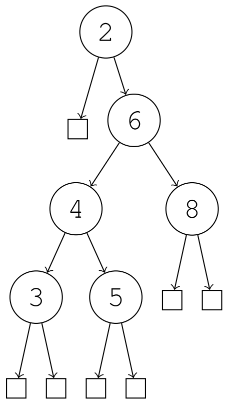


b) Delete key 1 in the above tree, and afterwards delete key 7 in the resulting tree.

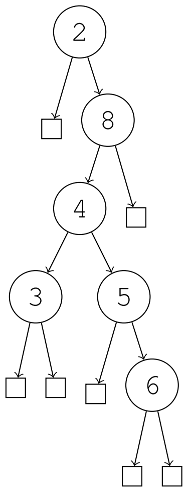
Solution: Key 1 is a leaf, so we can simply delete it without need for replacement:



Key 7 must either be replaced by its predecessor key, 6, or its successor key, 8. If key 7 is replaced by its predecessor:



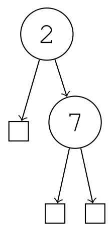
If key 7 is instead replaced by its successor:



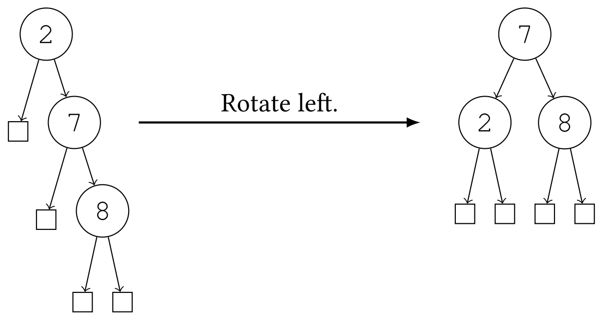
c) Draw the resulting tree when the above keys are inserted into an initially empty AVL tree. Give also the intermediate states before and after each rotation that is performed during the process.

Solution:

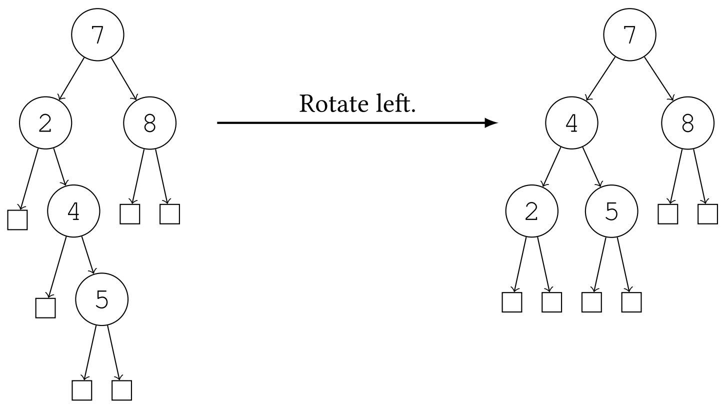
Insert 2 and then 7:



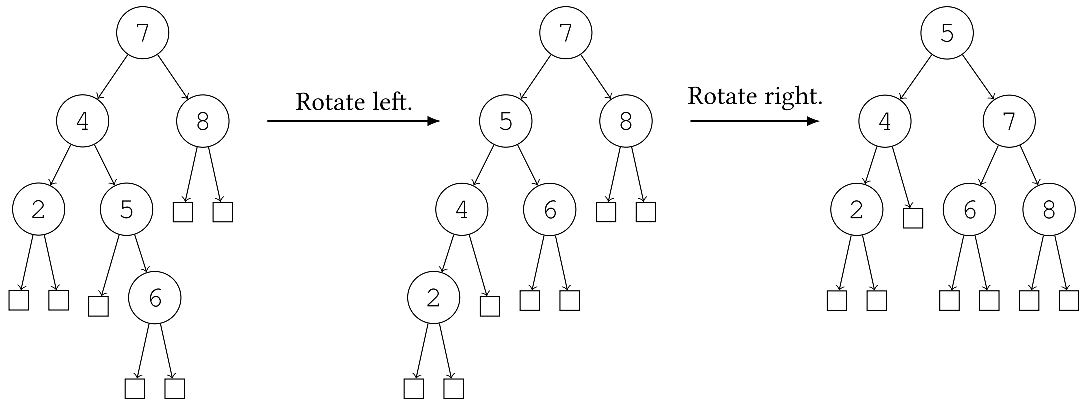
Insert 8:



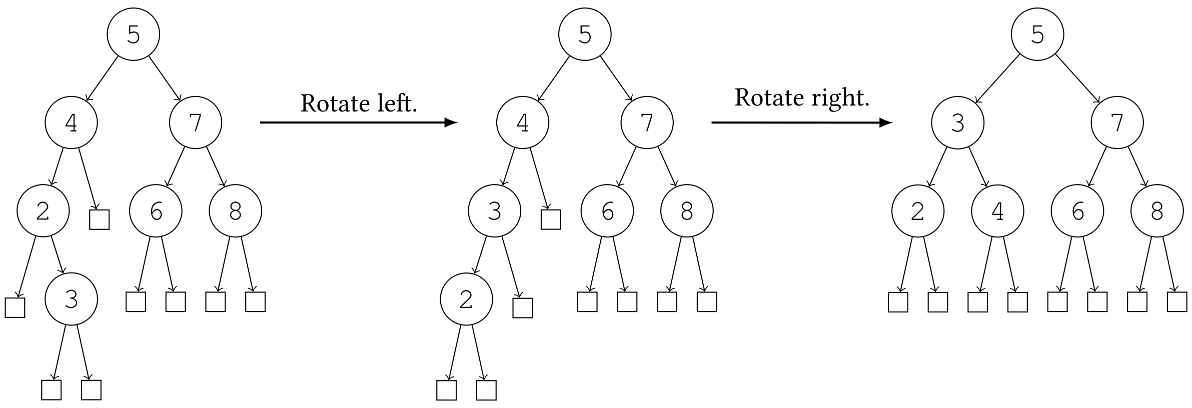
Insert 4 and then 5:



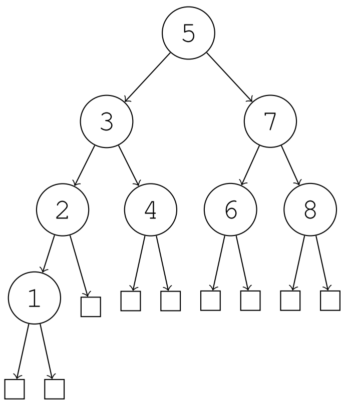
Insert 6:



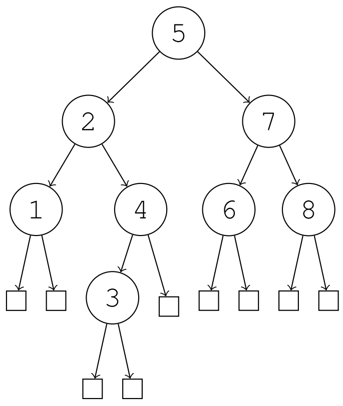
Insert 3:



Insert 1:



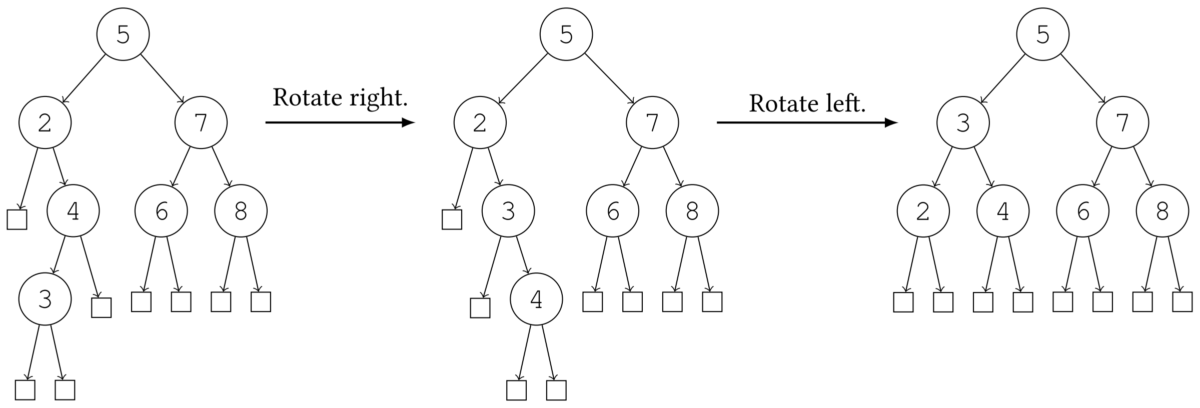
d) Consider the following AVL tree:



Delete key 1 in this tree, and afterwards delete key 7 in the resulting tree. Give also the intermediate states before and after each rotation is performed during the process.

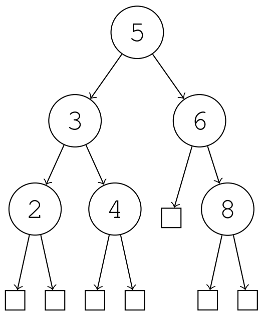
Solution:

Delete 1:

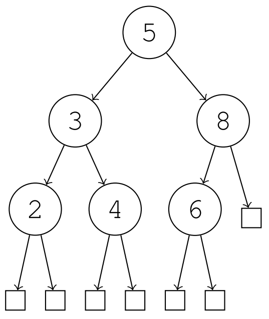


Delete 7:

Key 7 can either be replaced by its predecessor key, 6, or its successor key, 8. If key 7 is replaced by its predecessor:



If key 7 is instead replaced by its successor:



Exercise 9.2 *Exponential bounds for a sequence defined inductively.*

Consider the sequence $(a_n)_{n\in\mathbb{N}}$ defined by

$$\begin{aligned} a_0 &= 1, \\ a_1 &= 1, \\ a_2 &= 2, \\ a_i &= a_{i-1} + 2a_{i-2} + a_{i-3} \quad \forall i \geq 3. \end{aligned}$$

The goal of this exercise is to find exponential lower and upper bounds for a_n.

a) Find a constant $C > 1$ such that $a_n \leq \mathcal{O}(C^n)$ and prove your statement.

Solution:

Intuitively, the sequence $(a_n)_{n\in\mathbb{N}}$ seems to be increasing. Assuming so, we would have

$$a_i = a_{i-1} + 2a_{i-2} + a_{i-3} \le a_{i-1} + 2a_{i-1} + a_{i-1} = 4a_{i-1},$$

which yields

$$a_n \le 4a_{n-1} \le \ldots \le 4^n a_0 = 4^n.$$

This only comes from an intuition, but it is a good way to guess what the upper bound could be. Now let us actually prove (by induction) that $a_n \le 4^n$ for all $n \in \mathbb{N}$.

Induction Hypothesis. We assume that for $k \ge 2$ we have

$$a_k \le 4^k, \qquad a_{k-1} \le 4^{k-1}, \qquad a_{k-2} \le 4^{k-2}. \tag{1}$$

Base case $k = 2$. Indeed we have $a_0 = 1 \le 4^0$, $a_1 = 1 \le 4^1$ and $a_2 = 2 \le 4^2$.

Inductive step ($k \to k+1$). Let $k \ge 2$ and assume that the induction hypothesis (1) holds. To show that it also holds for $k+1$, we need to check that $a_{k+1} \le 4^{k+1}$, $a_k \le 4^k$ and $a_{k-1} \le 4^{k-1}$. The two last inequalities clearly hold since they are part of the induction hypothesis, so we only need to check that $a_{k+1} \le 4^{k+1}$. Indeed,

$$a_{k+1} = a_k + 2a_{k-1} + a_{k-2} \overset{(1)}{\le} 4^k + 2\cdot 4^{k-1} + 4^{k-2} \le 4^k + 2\cdot 4^k + 4^k = 4\cdot 4^k = 4^{k+1}.$$

Thus, $a_n \le 4^n$ for all $n \in \mathbb{N}$. In particular, we have shown that $a_n \le \mathcal{O}(C^n)$ for $C = 4 > 1$.

b) Find a constant $c > 1$ such that $a_n \ge \Omega(c^n)$ and prove your statement.

Solution:

If we again assume that the sequence is increasing, we would get

$$a_i = a_{i-1} + 2a_{i-2} + a_{i-3} \ge a_{i-3} + 2a_{i-3} + a_{i-3} = 4a_{i-3},$$

which yields

$$a_n \ge 4a_{n-3} \ge \ldots \ge 4^{\lfloor n/3 \rfloor} a_0 = 4^{\lfloor n/3 \rfloor}.$$

So we will aim to prove a lower bound of the form $a_n \ge \varepsilon \cdot 4^{n/3}$ for some constant $\varepsilon > 0$. We see that taking $\varepsilon := \min\{1, 4^{-1/3}, 2\cdot 4^{-2/3}\} = 4^{-1/3}$ will make the inequality satisfied for the base case, so let's prove by induction that $a_n \ge 4^{-1/3}4^{n/3}$ for all $n \in \mathbb{N}$.

Induction Hypothesis. We assume that for $k \ge 2$ we have

$$a_k \ge 4^{-1/3}4^{k/3}, \qquad a_{k-1} \ge 4^{-1/3}4^{(k-1)/3}, \qquad a_{k-2} \ge 4^{-1/3}4^{(k-2)/3}. \tag{2}$$

Base case $k = 2$. Indeed we have $a_0 = 1 \ge 4^{-1/3}\cdot 4^0$, $a_1 = 1 \ge 4^{-1/3}4^{1/3}$ and $a_2 = 2 \ge 4^{1/3} = 4^{-1/3}4^{2/3}$.

Inductive step ($k \to k+1$). Let $k \ge 2$ and assume that the induction hypothesis (2) holds. To show that it also holds for $k+1$, we need to check that $a_{k+1} \ge 4^{-1/3}4^{(k+1)/3}$, $a_k \ge 4^{-1/3}4^{k/3}$ and $a_{k-1} \ge 4^{-1/3}4^{(k-1)/3}$. The two last inequalities clearly hold since they are part of the induction hypothesis, so we only need to check that $a_{k+1} \ge 4^{-1/3}4^{(k+1)/3}$. Indeed,

$$\begin{aligned} a_{k+1} = a_k + 2a_{k-1} + a_{k-2} &\overset{(2)}{\ge} 4^{-1/3}\left(4^{k/3} + 2\cdot 4^{(k-1)/3} + 4^{(k-2)/3}\right) \\ &\ge 4^{-1/3}\left(4^{(k-2)/3} + 2\cdot 4^{(k-2)/3} + 4^{(k-2)/3}\right) = 4^{-1/3}\cdot 4\cdot 4^{(k-2)/3} = 4^{-1/3}4^{(k+1)/3}. \end{aligned}$$

Thus, $a_n \ge 4^{-1/3}4^{n/3}$ for all $n \in \mathbb{N}$. In particular, we have shown that $a_n \ge \Omega(c^n)$ for $c = 4^{1/3} > 1$.

Remark. One can actually show that $a_n = \Theta(\phi^n)$, where $\phi \approx 2.148$ is the unique positive solution of the equation $x^3 = x^2 + 2x + 1$.

Exercise 9.3 *Online supermarket.*

Assume that you work in a large online supermarket that offers different types of goods. At every moment you have to know the number of goods of each type that the supermarket currently offers. Let us denote the number of goods of type t by S_t. At any moment S_t can either be decreased (if someone has bought some goods of type t) or increased (if some goods of type t have been delivered from the manufacturer). Also your boss can ask you how many goods of type t does the supermarket currently offer. So you can receive three types of queries: to decrease S_t by $0 < x \leq S_t$, to increase S_t by $x > 0$ or to return S_t.

Assume that at each moment number of different types of goods that the supermarket offers at that moment is bounded by $n > 0$, but the number of types of goods that the supermarket can potentially offer might be much larger than n. Consider the following example: $n = 3$, at 14:00 the supermarket can offer 5 balls, 1 doll and 4 phones and at 14:15 it can offer 6 balls, 3 chairs and 12 pencils.

Provide an algorithm that can handle each query in time $\mathcal{O}(\log n)$. You may assume that initially all S_t are zero.

Solution:

We store the goods in an AVL-tree. That is, for good t we use t as the key that determines the position in the AVL-tree, and we store the value S_t in the node of t. We only store a good t in the AVL-tree if $S_t > 0$.

For all three types of queries, we first search the key t in the AVL-tree, which takes time $O(\log n)$. If t does not exist in the tree, then we give out 0 if we are asked for the number of elements; we give an error for a decrease query; and we insert the key t into the AVL-tree with value $S_t := x$ if we get an increase query. The latter operation takes time $O(\log n)$, the other two take constant time. However, since we first need to search for the key, all queries need total time $O(\log n)$.

If the key t exists, we proceed similarly. Depending on the query, we return S_t, or in-/decrease S_t by x. Moreover, if we decrease S_t to zero then we delete the key from the AVL-tree, which takes time $O(\log n)$. Again, all queries take a total time of $O(\log n)$, as required.

Exercise 9.4 *Augmented Binary Search Tree* **(1 point)**.

Consider a variation of a binary search tree, where each node has an additional member variable called SIZE. The purpose of the variable SIZE is to indicate the size of the subtree rooted at this node. An example of an augmented binary search tree (with integer data) can be seen below (Fig. 1).

a) What is the relation between the size of a node and the sizes of its children?

   Solution:

   For every node in the tree, we have

$$\text{NODE.SIZE} = \text{NODE.LEFT.SIZE} + \text{NODE.RIGHT.SIZE} + 1.$$

   Note that throughout the solution of this exercise, we adopt the convention that $\text{NULL.SIZE} = 0$.

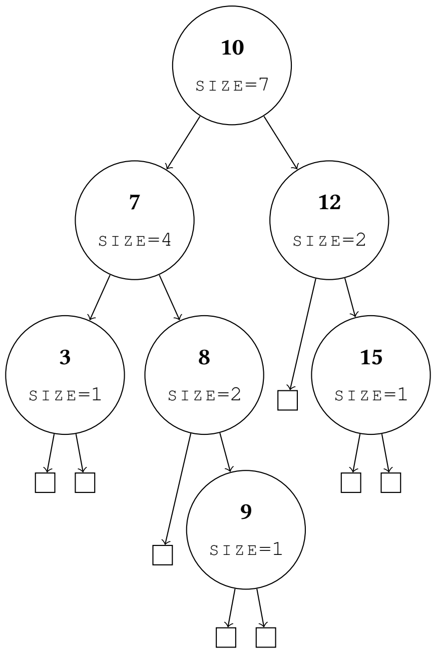


Figure 1: Augmented binary search tree

b) Describe in pseudo-code an algorithm VERIFYSIZES(ROOT) that returns TRUE if all the sizes in the tree are correct, and returns FALSE otherwise. For example, it should return TRUE given the tree in Fig. 1, but FALSE given the tree in Fig. 2.

   What is the running time of your algorithm? Justify your answer.

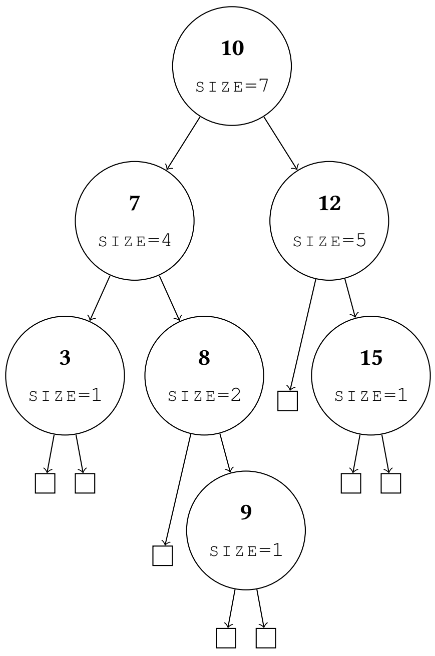


Figure 2: Augmented binary search tree with buggy size: incorrect size for node with data "12"

Solution:

Algorithm 1 Verifying the sizes of the tree

```
function VerifySizes(root)
    if root = null then
        return true
    else if VerifySizes(root.left) = false or VerifySizes(root.right) = false then
        return false
    else
        CorrectSize ← 1 + root.left.size + root.right.size
        return CorrectSize = root.size
```

The above recursive algorithm visits every node of the tree exactly once. Furthermore, it performs a constant number of operations $O(1)$ at each node. Therefore, the runtime is $O(n)$, where n is the number of nodes in the tree.

c) Suppose we have an augmented AVL tree (i.e., as above, each node has a size member variable). Describe in pseudo-code an algorithm Select(root, k) which, given an augmented AVL tree and an integer k, returns the k-th smallest element in the tree in $O(\log n)$ time.

Example: Given the tree in Fig. 1, for $k = 3$, Select returns 8; for $k = 5$, it returns 10; for $k = 1$, it returns 3; etc.

Solution:

Algorithm 2 Selecting the k-th smallest element

```
function Select(root, $k$)
    current ← root.left.size + 1
    if $k$ = current then
        return root.data
    else if $k$ < current then
        return Select(root.left, $k$)
    else
        return Select(root.right, $k$ − current)
```

The above algorithm follows a downward path until it finds the correct node. Furthermore, it performs a constant number of operations $O(1)$ at each visited node. Therefore, the runtime of the algorithm is $O(h)$, where h is the height of the tree. Now since the tree is an AVL tree, we have $h = O(\log n)$. We conclude that the runtime of the above algorithm is $O(\log n)$.

d)* To maintain the correct sizes for each node, we have to modify the AVL tree operations, insert and remove. For this problem, we will consider only the modifications to the AVL-insert method (i.e., you are not responsible for AVL-remove). Recall that AVL-insert first uses regular insert for binary search trees, and then balances the tree if necessary via rotations.

- How should we update insert to maintain correct sizes for nodes?

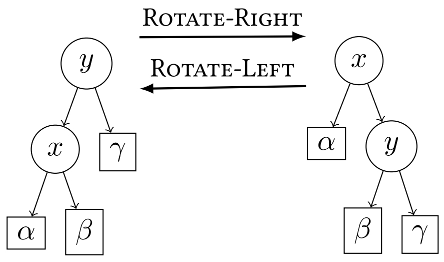
During the balancing phase, AVL-insert performs rotations. Describe what updates need to be made to the sizes of the nodes. (It is sufficient to describe the updates for left rotations, as right rotations can be treated analogously.)

Solution:

The regular INSERT function follows a downward path and then adds the new node as a leaf at the correct place. We only need to increment the variable SIZE by 1 at each visited node, and set the variable SIZE of the added leaf to 1. The runtime of the modified function remains $O(h)$, where h is the height of the tree. If the tree is an AVL tree, then the runtime is $O(\log n)$.

Regarding AVL-INSERT, after modifying the regular INSERT function as we explained, we need to modify the rotation functions LEFT-ROTATE and RIGHT-ROTATE to maintain the correct SIZE variables.

Suppose we are performing a right-rotation on the node y of the tree that is drawn on the left (or performing a left-rotation on the node x of the tree that is drawn on the right):



In the above diagrams, α, β and γ represent subtrees. As can be easily seen, only x.SIZE and y.SIZE need to be updated, and we can apply the relation in a) in the correct order:

- At the end of ROTATE-RIGHT, we apply

  $$y.\text{SIZE} \leftarrow y.\text{LEFT.SIZE} + y.\text{RIGHT.SIZE} + 1,$$

  and then

  $$x.\text{SIZE} \leftarrow x.\text{LEFT.SIZE} + x.\text{RIGHT.SIZE} + 1.$$

- At the end of ROTATE-LEFT, we apply

  $$x.\text{SIZE} \leftarrow x.\text{LEFT.SIZE} + x.\text{RIGHT.SIZE} + 1,$$

  and then

  $$y.\text{SIZE} \leftarrow y.\text{LEFT.SIZE} + y.\text{RIGHT.SIZE} + 1.$$

As we can see, the runtime of the modified RIGHT-ROTATE (resp. LEFT-ROTATE) function remains $O(1)$. Therefore, the runtime of AVL-INSERT remains $O(\log n)$.

Remark: It is also possible to modify AVL-DELETE to maintain the correctness of the SIZE variables while keeping the $O(\log n)$ runtime.

Exercise 9.5* *Maximum Depth Difference of two Leaves.*

Consider an AVL tree of height h. What is the maximum possible difference of the depths of two leaves? Describe which structure such trees need to have, and draw examples of corresponding trees for every $h \in \{2, 3, 4\}$. Derive a recursive formula (depending on h), solve it and use induction to prove the correctness of your solution. Provide a detailed explanation of your considerations.

Hint: *For the proof the principle of complete induction can be used. Let $\mathcal{A}(n)$ be a statement for a number $n \in \mathbb{N}$. If, for every $n \in \mathbb{N}$, the validity of all statements $\mathcal{A}(m)$ for $m \in \{1, \ldots, n-1\}$ implies the* *validity of $\mathcal{A}(n)$, then $\mathcal{A}(n)$ is true for every $n \in \mathbb{N}$. Thus, complete induction allows multiple base cases and inductive hypotheses.*

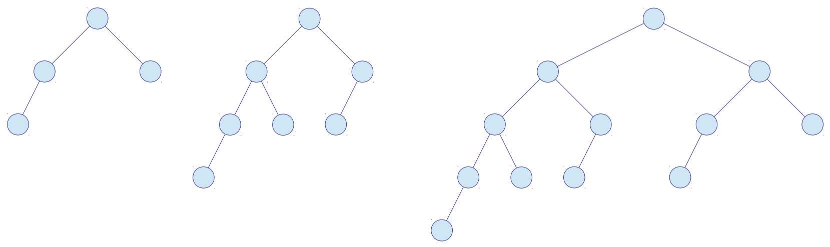
Solution:

For an AVL-tree T with a root node v and height h, we can distinguish the following 3 cases:

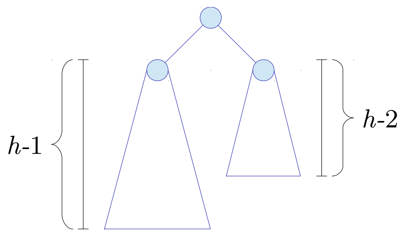
- Both sub-trees $T_l(v)$ and $T_r(v)$ have height $h-1$
- $T_l(v)$ has height $h-1$ and $T_r(v)$ has height $h-2$, or
- $T_l(v)$ has height $h-2$ and $T_r(v)$ has height $h-1$

As we are interested in the *maximum* depth difference of two leaves, we can disregard the first case, and focus on sub-trees that have heights that differ by 1. Without loss of generality, we can take the second case, assuming that the left sub-tree will have height of $h-1$, while the right sub-tree will have height of $h-2$. If the left sub-tree is an AVL-tree of height $h-1$, then the right tree must be an AVL-tree of height $h-2$. This comes from the properties of an AVL tree, because if at any time they differ by more than one, rebalancing is done to restore this property. As a result of this, the entire tree T will have a height of h and as such there will be a leaf on the left-subtree with this depth.

The figure below illustrates the AVL trees of height $h \in \{1, 2, 3\}$:



In general, we consider trees with the following structure:



The left subtree $T_l(v)$ contains a leaf of depth h (while $T_l(v)$ has height of $h-1$), the right subtree $T_r(v)$ contains a leaf of depth $h-1$ (while $T_r(v)$ has height $h-2$). The maximum possible difference of the depths of two leaves in the tree (with height h) is therefore 1 greater than the maximum difference of the depths of two leaves in the right subtree (with height $h-2$). For $h=2$ and $h=3$, the maximum depth difference is exactly 1.

As a result, we have the following recursive formula for the maximum difference of the depths of two leaves in a tree of height h:

$$D(2) = 1,\ D(3) = 1,\ D(h) = 1 + D(h-2) \text{ for all } h \geq 4. \tag{3}$$

From the above, we can assume that $D(h) = \lfloor h/2 \rfloor$. We prove this assumption using induction over h.

Base case I ($h=2$): $D(2) = 1 = \lfloor 2/2 \rfloor$.

Base case II $(h = 3)$: $D(3) = 1 = \lfloor 3/2 \rfloor$.

Induction hypothesis: Assume that the property holds for some h: $D(h-2) = \lfloor (h-2)/2 \rfloor$.

Inductive step: $((h-2) \to h)$: From the recursive definition of $D(h)$, we have:

$$D(h) = 1 + D(h-2) = 1 + \lfloor (h-2)/2 \rfloor = 1 + \lfloor h/2 - 1 \rfloor = 1 + \lfloor h/2 \rfloor - 1 = \lfloor h/2 \rfloor. \tag{4}$$