Exercise 1: Red-Black Trees

(a) Decide for each of the following trees if it is a red-black tree and if not, which property is violated:

(b) On the following red-black tree, first execute the operation \text{insert}(8) and afterwards \text{delete}(5). Draw the resulting tree and document intermediate steps.

Sample Solution

(a) From left to right:

1) Red-black-tree
2) No red-black-tree, because it is no binary search tree (the root’s right child has a smaller key).
3) No red-black-tree, because the number of black nodes on a path from the root to a leaf is larger if you go through the left subtree.

(b) We insert a red node with key 8 according to the rule of inserting into binary search trees.
We are in case 1b from the lecture. We do a right-rotate(9,8),

\[
\begin{array}{c}
\begin{array}{c}
6
\end{array}\\
\begin{array}{c}
4
\end{array}\\
\begin{array}{c}
1
\end{array} & \begin{array}{c}
5
\end{array} & \begin{array}{c}
7
\end{array} & \begin{array}{c}
9
\end{array}
\end{array}
\ \begin{array}{c}
\begin{array}{c}
\text{NIL}
\end{array}\\
\begin{array}{c}
\text{NIL}
\end{array}\\
\begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array}
\end{array}
\ \begin{array}{c}
\begin{array}{c}
\text{NIL}
\end{array}\\
\begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array}
\end{array}
\end{array}
\]

a left-rotate(8,7)

\[
\begin{array}{c}
\begin{array}{c}
6
\end{array}\\
\begin{array}{c}
4
\end{array}\\
\begin{array}{c}
1
\end{array} & \begin{array}{c}
5
\end{array} & \begin{array}{c}
7
\end{array} & \begin{array}{c}
8
\end{array} & \begin{array}{c}
9
\end{array}
\end{array}
\ \begin{array}{c}
\begin{array}{c}
\text{NIL}
\end{array}\\
\begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array}
\end{array}
\ \begin{array}{c}
\begin{array}{c}
\text{NIL}
\end{array}\\
\begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array} & \begin{array}{c}
\text{NIL}
\end{array}
\end{array}
\end{array}
\]

and recolor nodes 7 and 8.
Now we execute delete(5). We are in case 2b from the lecture (deleting a black node with two NIL-children). First we remove node 5 from the tree and color the right NIL-child of node 4 double black to correct the black height.

We are in case A.2 from the lecture. We do a left-rotate(3,1)

and recolor nodes 1 and 3.
Now we are in case A.1. We do a right-rotate(3,4) and recolor. Finally, the tree looks like this.

Exercise 2: AVL-Trees

An AVL-tree is a binary search tree with the additional property that for each node $v$, the depth of its left and its right subtree differ by at most 1.

(a) Show via induction that an AVL-tree of height $d$ is filled completely up to depth $\lfloor \frac{d}{2} \rfloor$.

A binary tree is filled completely up to depth $d'$ if it contains for all $x \leq d'$ exactly $2^x$ nodes of depth $x$.

(b) Give a recursion relation that describes the minimum number of nodes of an AVL-tree as a function of $d$. 
(c) Show that an AVL-tree with \( n \) nodes has depth \( O(\log n) \).

You can either use part (a) or part (b).

Sample Solution

(a) **Induction start:** Each non-empty tree has a root and is hence completely filled up to depth 0. Hence the statement is true for \( d = 0 \) and \( d = 1 \) (as \( \lfloor d/2 \rfloor = 0 \) for \( d = 0 \) and \( d = 1 \)).

**Induction step:** Assume the statement holds for all AVL-trees up to depth \( d \). We show that it also holds for AVL-trees of depth \( d + 1 \).

Let \( T \) be an AVL-tree of depth \( d + 1 \) with \( r \) as root and \( T_L \) and \( T_R \) as left and right subtree. One of these subtrees must have depth \( d \) (let’s say \( T_L \)). As \( T \) is an AVL-tree, it follows that \( T_R \) has depth at least \( d - 1 \). By the induction hypothesis, \( T_L \) is completely filled up to depth \( \lfloor d/2 \rfloor \) and \( T_R \) is completely filled up to depth \( \lfloor (d-1)/2 \rfloor \). So both subtrees are completely filled up to depth \( \lfloor (d+1)/2 \rfloor = \lfloor d/2 \rfloor - 1 \) and hence \( T \) is filled completely up to depth \( \lfloor (d+1)/2 \rfloor \).

(b) Let \( n_d \) be the minimum number of nodes in an AVL-tree of depth \( d \). As every tree of depth \( d \) has at least \( d + 1 \) nodes (as it contains a path of length \( d \)), we obtain as base cases \( n_0 = 1 \) and \( n_1 = 2 \). Now let \( d \geq 2 \). An AVL-tree \( T \) of depth \( d \) consists of a root \( r \), a left subtree \( T_L \) and a right subtree \( T_R \). One of them, let’s say \( T_L \), has depth \( d - 1 \) and hence at least \( n_{d-1} \) nodes. As \( T \) is an AVL-tree, it follows that \( T_R \) has depth at least \( d - 2 \) and hence at least \( n_{d-2} \) nodes. Hence \( T \) has at least \( n_d = n_{d-1} + n_{d-2} + 1 \) nodes.

(c) **Using (a):** And AVL-tree of depth \( d \) is filled completely up to depth \( \lfloor d/2 \rfloor \), so \( T \) has \( n \geq 2^{\lfloor d/2 \rfloor} \) nodes. We obtain

\[
2^{\lfloor d/2 \rfloor} \leq n \\
\iff \lfloor d/2 \rfloor \leq \log(n) \\
\implies d/2 - 1/2 \leq \lfloor d/2 \rfloor \leq \log(n) \\
\implies d \leq 2\log n + 1 \\
\implies d \in O(\log(n)).
\]

**Using (b):** Similar to the Fibonacci-series we have \( n_d = n_{d-1} + n_{d-2} + 1 = 2n_{d-2} + n_{d-3} + 2 \geq 2n_{d-2} \). This means that increasing the depth by 2 doubles the number of nodes, so the number of nodes grows exponentially in the depth, or the depth grows logarithmically in the number of nodes. More formally, we have \( n_d \geq 2n_{d-2} \geq 2^2n_{d-4} \geq \cdots \geq 2^{\lfloor d/2 \rfloor}n_{d-2\lfloor d/2 \rfloor} \geq 2^{\lfloor d/2 \rfloor}n_0 = 2^{\lfloor d/2 \rfloor} \). The rest follows as above.