Algorithms and Data Structures

Übungsblatt 7

Paramie Willmann pw221 Andrés Fernández Lebrón af231
Mehmet Kaan Isik mi84

21. Juni 2022

Aufgabe 1

a) Determining RBT

RBT-Properties (Syn: Rules, Conditions) [1, 2]:

(1) Nodes are either (R) or black (B).

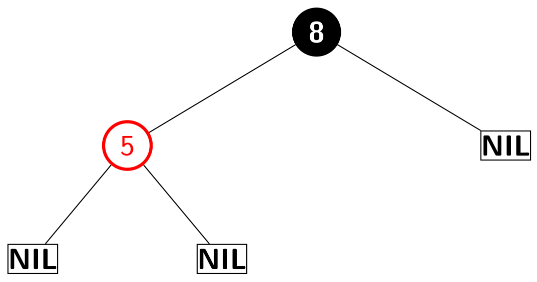
(2) The root is B.

(3) Leaves (NIL) are B.

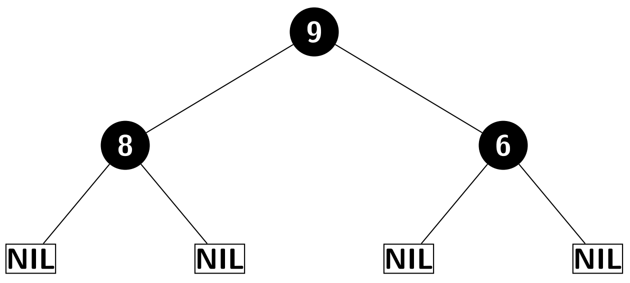
(4) Every R-Node has a B-Parent [3]

(4a) If n is a R-Node, then both children are B. <==> A red node does not have a red child.

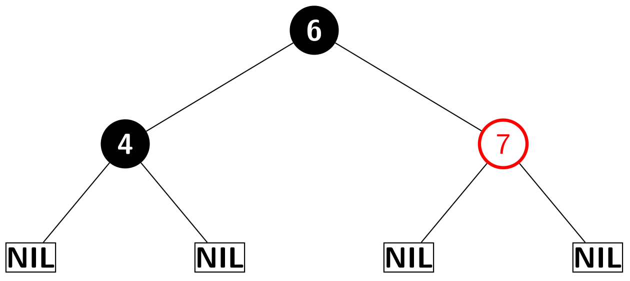
(5) All simple paths from any node n to a descendant leaf contain the same number of B-Nodes = black-height(n) –Color of n is not counted.

i. 

Yes, a.i is a RBT. Justification: All RBT-Properties are fulfilled.



ii. No, a.ii is a RBT. Justification: All RBT-Properties are fulfilled, except it is not a BST.

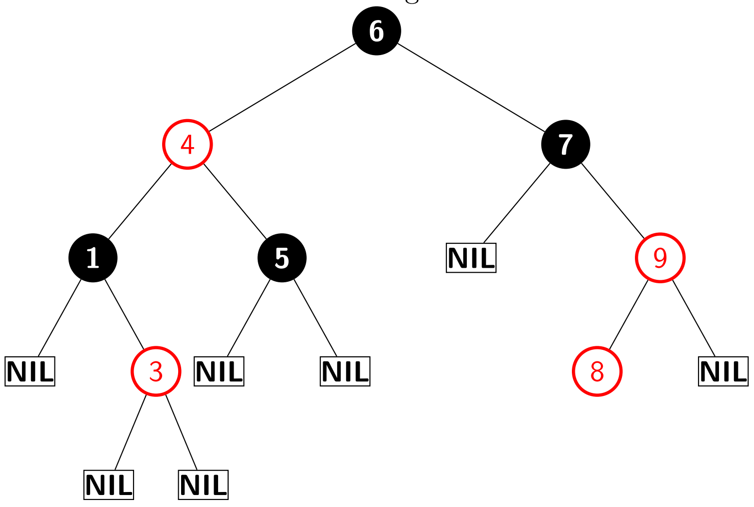


iii. No, a.3. is not a RBT. Justification: For every NIL-node the number of B-nodes is not equal. E.g. Any simple path from node (6) to any leaf passing through node (4) and any simple path from that same node to any leaf passing through node ((7)) differs in the amount of black nodes.

b) Procedure:

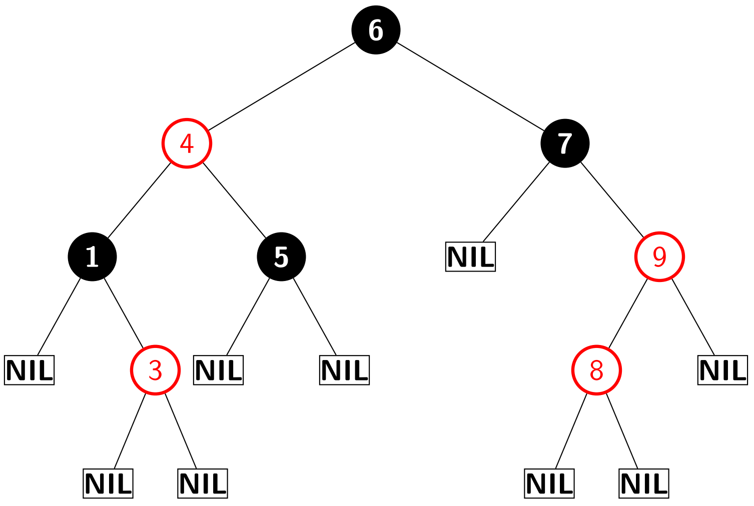
1. Using the relevant BST-Operation ((insert(), delete()))
2. Change Colors
3. Restructuring of pointers (links) via rotations
4. Check RBT and BST-Properties Fulfillment

First we start with the insertion of 8 into given Tree:

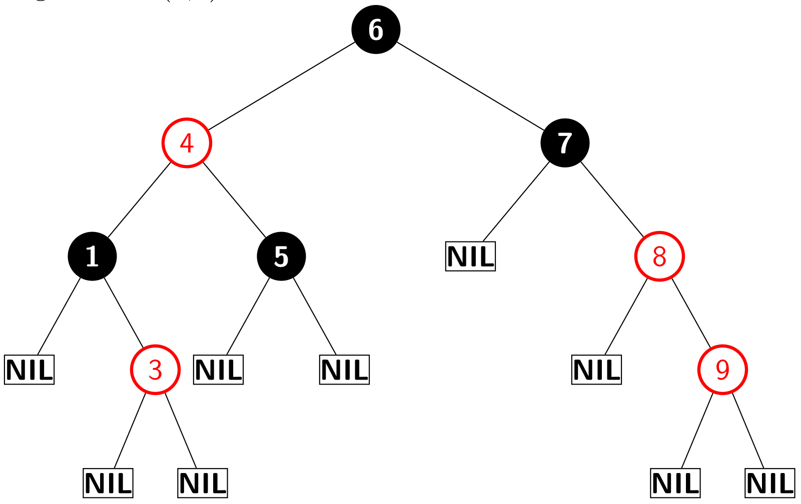


insert(8)

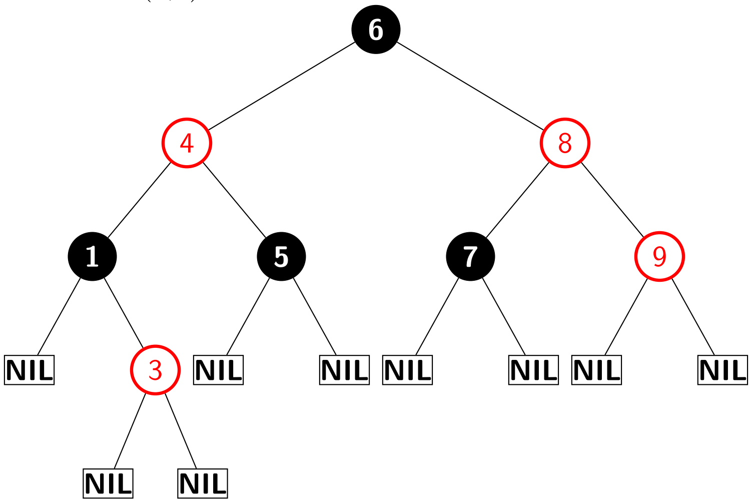
Then we will put two nill nodes after 8.



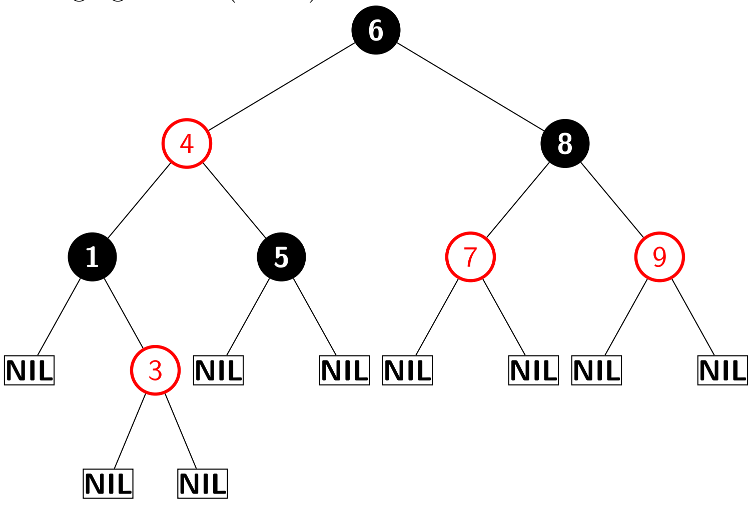
Then we will rebalance the red-blackness of the tree.
Right-rotate(9,8) :



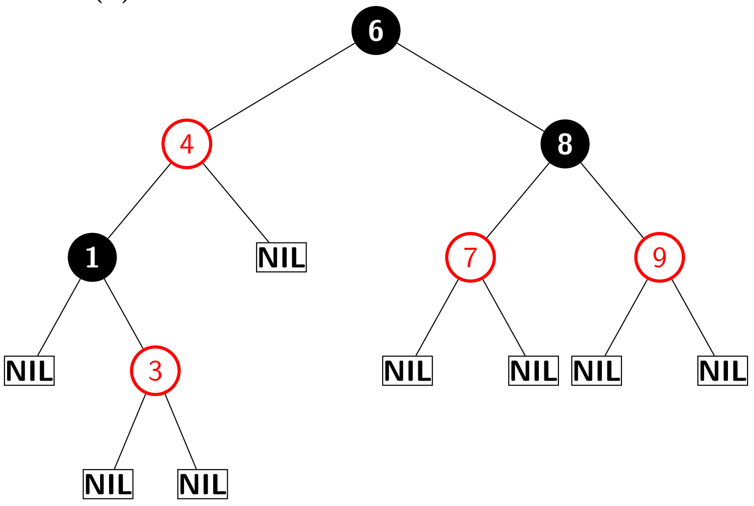
Left-rotate(8,7)



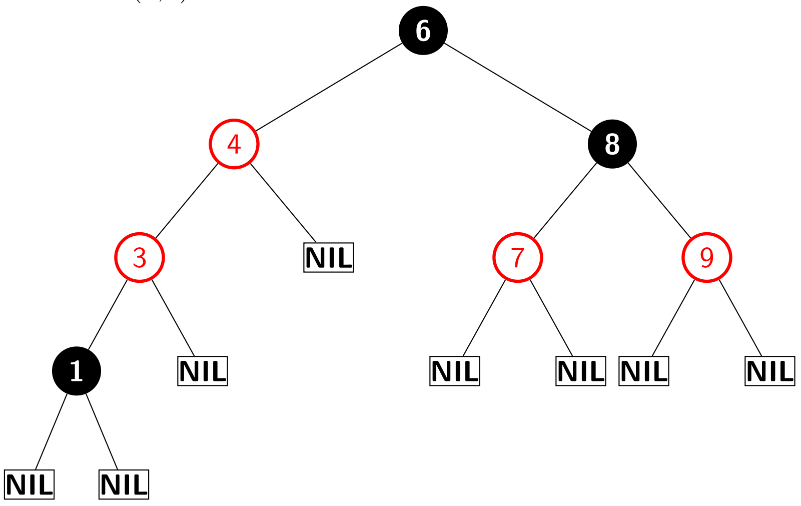
Changing Colors (7 & 8)



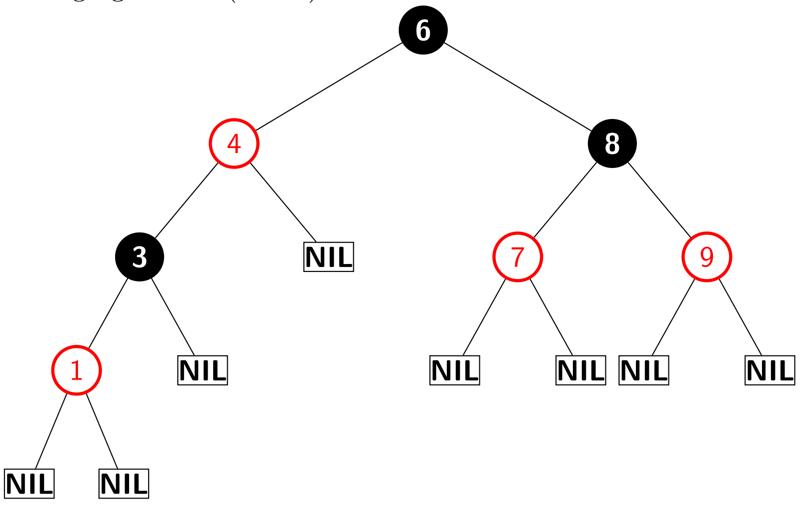
delete(5):



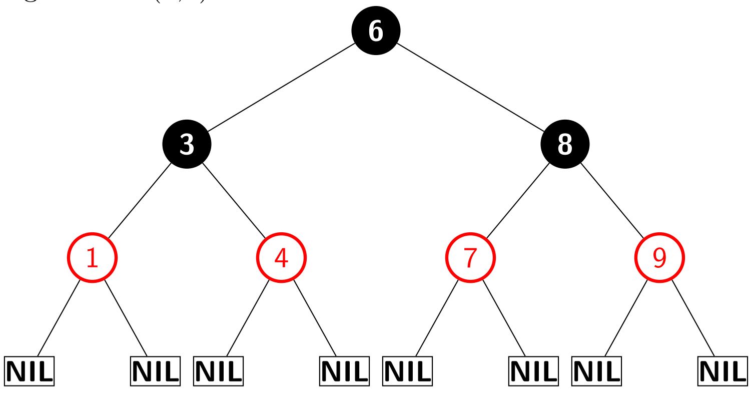
left-rotate(3,1):



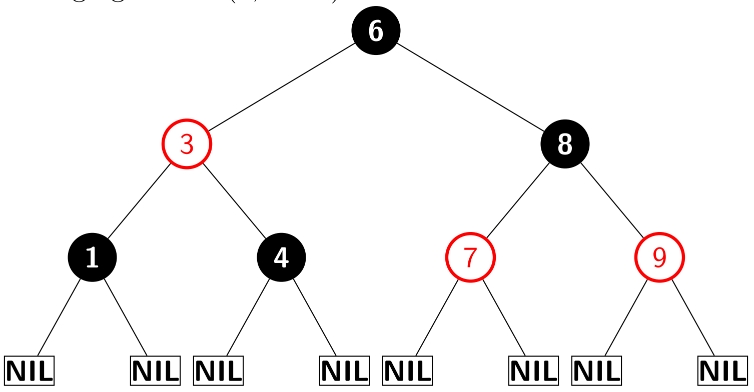
Changing Colors (1 & 3)



right-rotate(3,4)



Changing Colors (1,3 & 4)



Aufgabe 2

a)

- Proposition: A AVL-Tree of depth d is filled completely up to depth d/2. (A binary tree is filled completely up to depth d´ if it contains for all x ¡= d´ exactly 2**x nodes of depth x. depth x.)
- Proof: we define d as depth of AVL and d' as d/2. a condition muss be fulfilled as proposition mentioned.
- Base Case:

  If T is a non-empty AVL-Tree, then T is completely filled up to depth d/2 in the following cases:
  If h = 0, then D(h) = 0, so that —$_{d}$/2$_{|}$ = 0
  If h = 1, then D(h) = 1, so that —$_{d}$/2$_{|}$ = 0

  Inductive Step:

  If both base cases (h =0 or h = 1) are completely filled trees, then this should hold for a AVL-Tree of depth d + 1
  Let T be a AVL-Tree with depth d + 1, and Tl as well as Tr its left and right subtrees.
  Since T is a AVL-Tree, then either Tl or Tr has a depth of d.
  Let be Tl the subtree with depth of d.
  Then Tr has a dept of d-1.

  Inductive Hypothesis:

  Whereas Tl is would be completely filled up to depth d/2,
  Tr would be completely filled up to depth
  (d-1)/2 .

  Conclusion:

  Therefore, both Tl and Tr would be completely filled up to depth
  (d-1)/2 = ((d-1) / 2) - 1 = ((d + 1) / 2) - 1.
  Thus, T is completely filled up to depth (d + 1)/2

b) Equation expressing a recursive function for finding N(h)
(minimum number of nodes possible in an AVL-Tree with height h):

N(h) = N(h-1) + N(h-2) + 1

N(h)-Base Cases:
N(0) = 1
N(1) = 2

c) for the proof:
we define ours depth of tree as i.

$$F_i = F_{i-1} + F_{i-2} + 1$$

if we keep on iterating ours F$_i weget$

$$F_i = F_{i-1} + F_{i-2} + 1 + F_{i-3} + F_{i-4} + 1 + 1$$

then we can reform it

$$F_i = F_{i-2} + 2 * Fi - 3 + Fi - 4 + 1 + 2$$

so we can define ours Function
we keep iterating ours Function the function grow

$$2 * F\frac{i}{2}$$
$$\text{F}_i > 2 * Fi - 2$$
$$\text{F}_{i-2} > 2 * Fi - 4$$

we can redefine function by logarithm function.

$$F_i > 2 * F\frac{i}{2}$$

$$log(F_i > log(2 * F\frac{i}{2})$$

–applied logarithm rule's

$$log(F_i) > (i/2)$$

$$2 * log(F_i) > i$$

we can prove that depth of the tree is equal to i.
and time complexity is $O(log(i)$.