3. SEARCHING

BINARY SEARCH TREE

Insert: assume $T = \emptyset$ & we insert elements in order: (5, 6)

Tree-Insert (T, z) // $T = \emptyset$ so-far, i.e., T.root = NIL

L1 x = T.root = NIL
L2 y = NIL
L3 WHILE not applied as x = NIL
{
L7 z.top = y = NIL
L8 IF (y = NIL) => T.root := z
Stop

z:

TOP	L	key	R
NIL	NIL	5	NIL

T.root = z = (5) = T.root!

Tree-Insert (T, z)

z:

TOP	L	key	R
NIL	NIL	6	NIL

L1 x = T.root

x:

TOP	L	key	R
NIL	NIL	5	NIL

= T.root

L2 y = NIL

y

L3 WHILE (x ≠ NIL)
L4 y = x
L5 ✗
L6 yes: x := x.right (= NIL)

x.right

TOP	L	key	R
NIL	NIL	NIL	NIL

"NIL"

x

while-loop stop as x = NIL

---

up to here!

L7 z.top = y
L8 ✗
L9 ✗
L10 z.key ≥ y.key
6 => y.left = z

y

TOP	L	key	R
NIL	NIL	5	NIL

= T.root

z

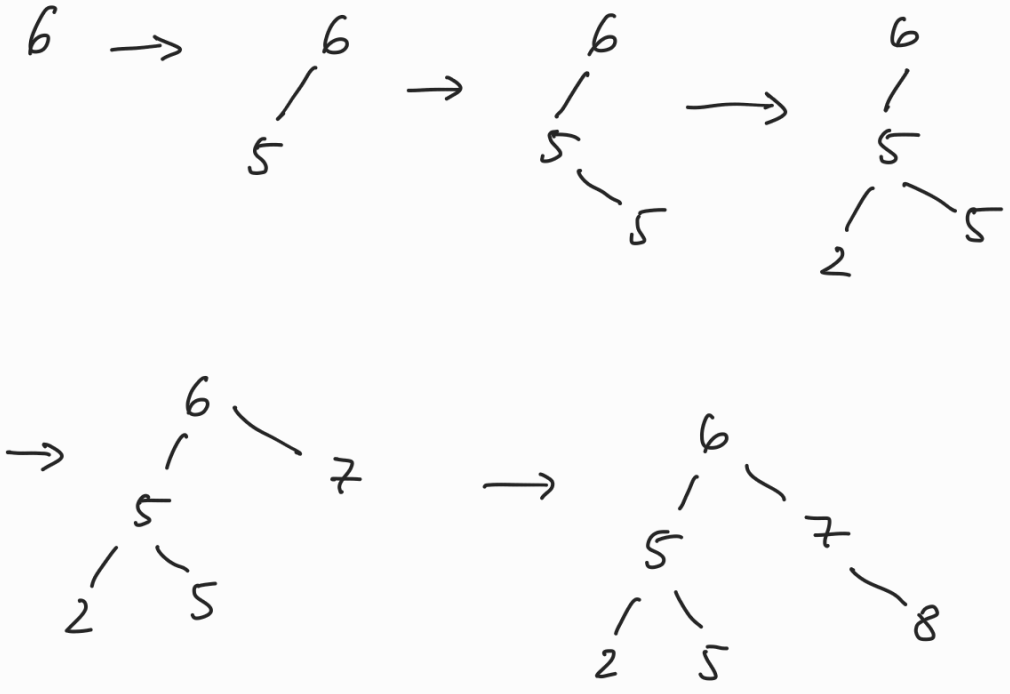
z:

TOP	L	key	R
NIL	NIL	6	NIL

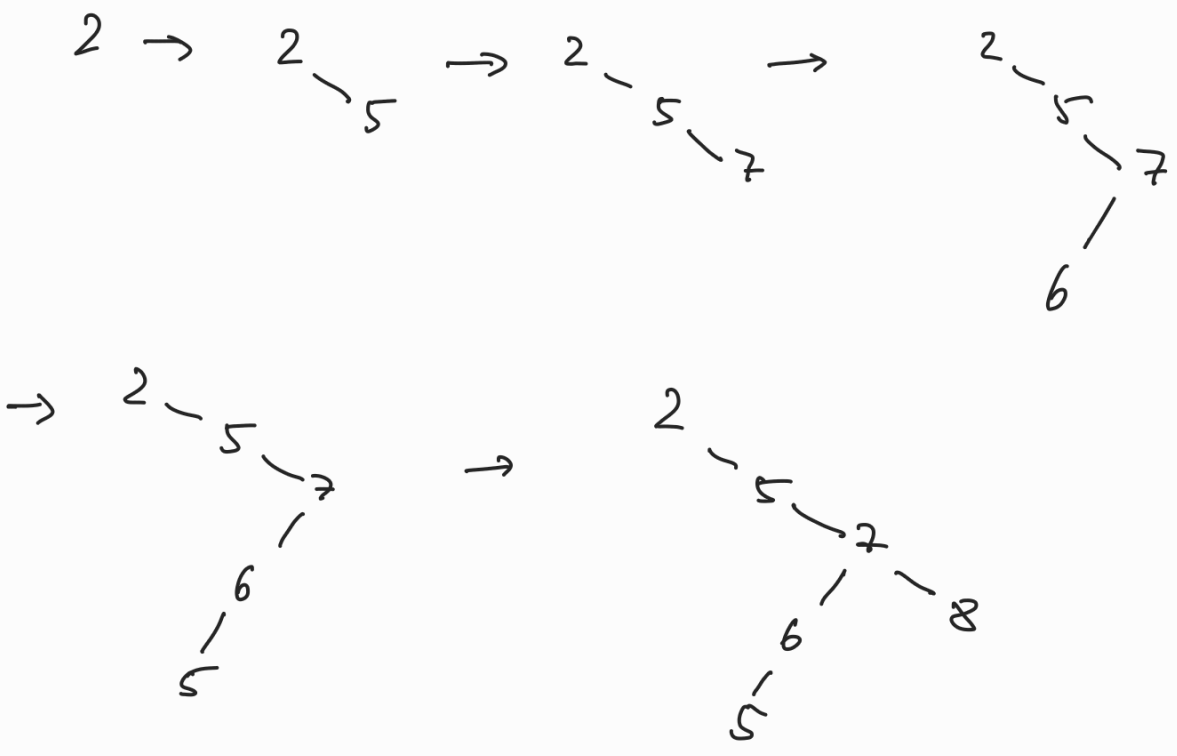
y

insert "short & simplified tree representation:

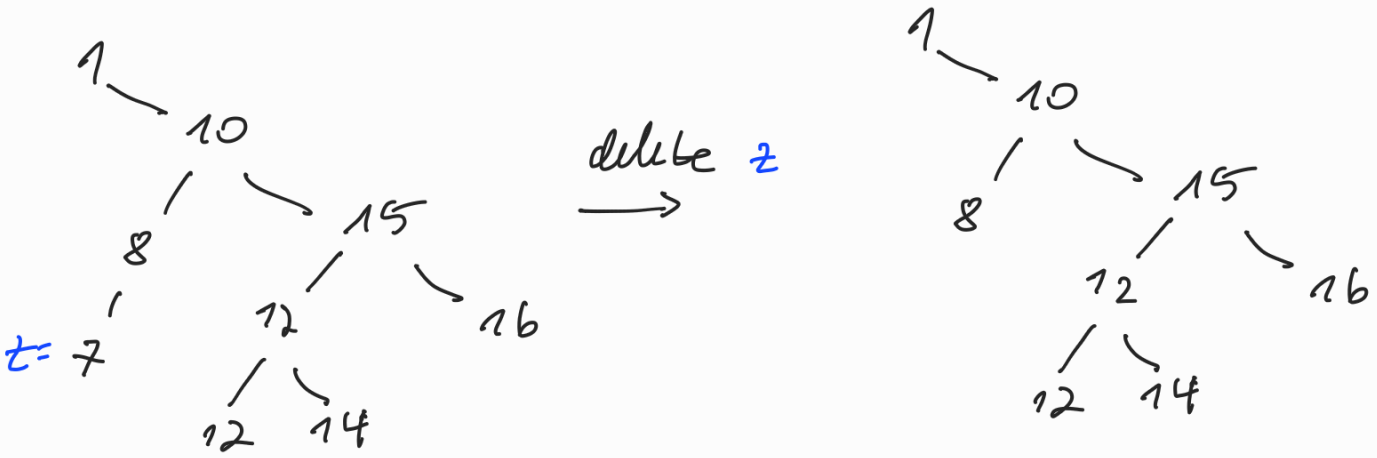
insert in order: 6, 5, 5, 2, 7, 8

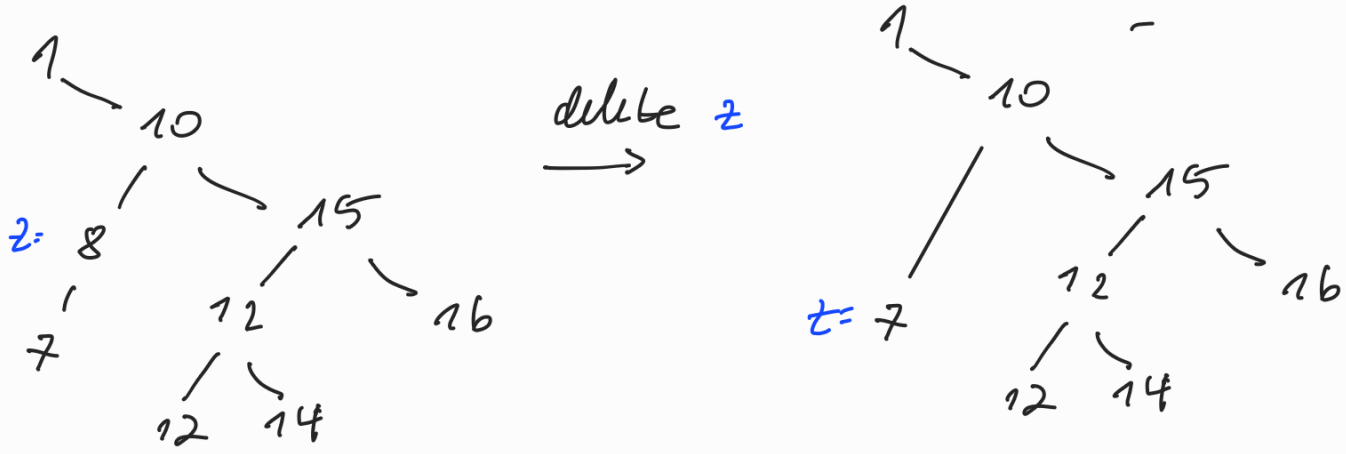


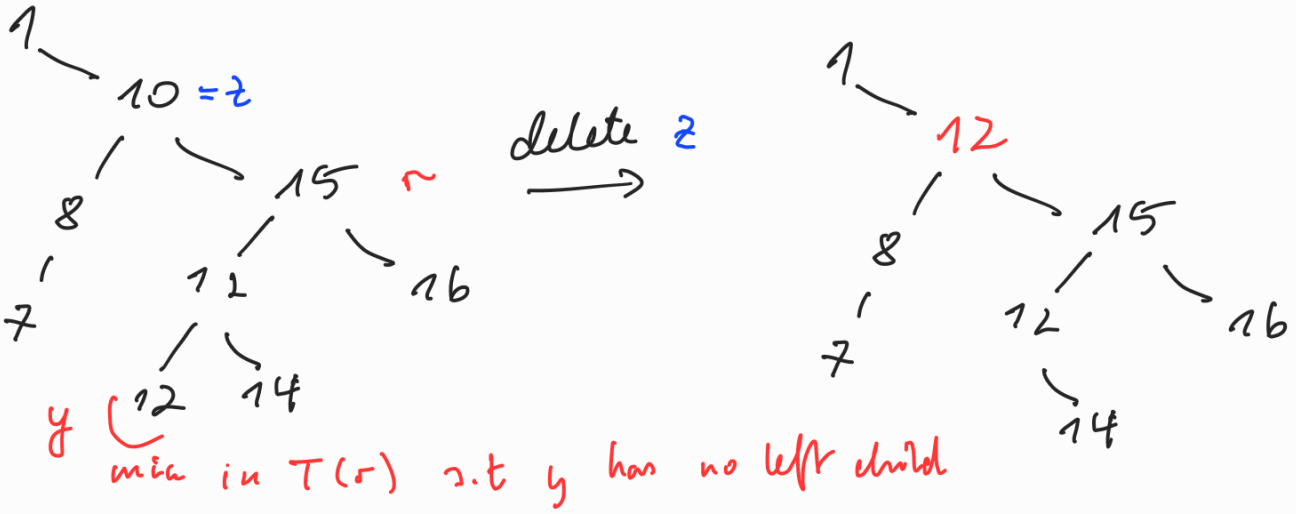
insert in order: 2, 5, 7, 6, 5, 8



delete





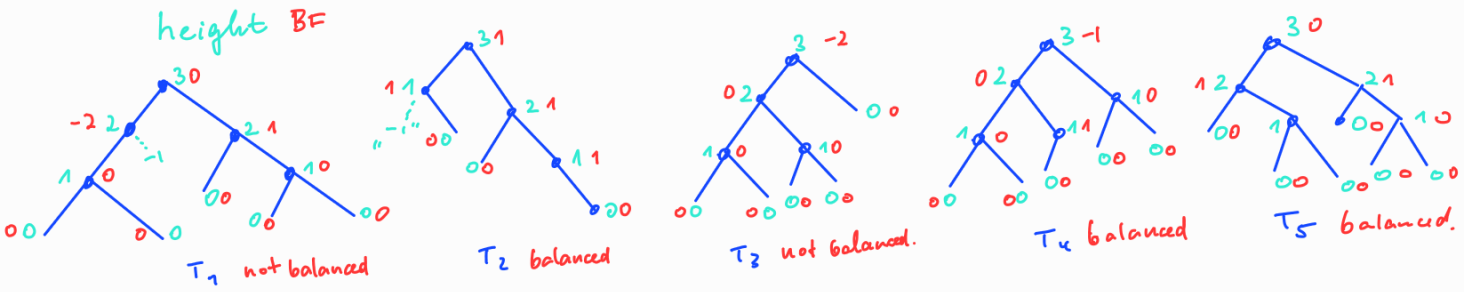


Since y min => all keys in T(r) are $\geq$ y.key

while z.key $\in$ y.key (as y $\in$ T(r))

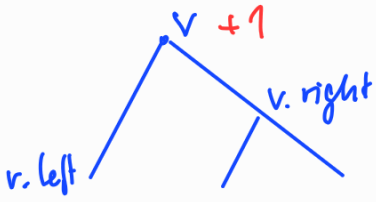
=> correct.!

AVL-trees

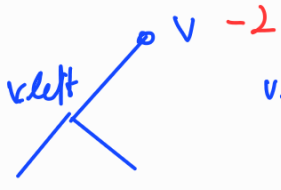


$$BF(v) = h(T(v.right)) - h(T(v.left))$$

[height of empty tree = $-1 \hat{=} T(NIL)$]



$$BF(v) = h(T(v.right)) - h(T(v.left))$$
$$= 1 - 0$$
$$= +1$$

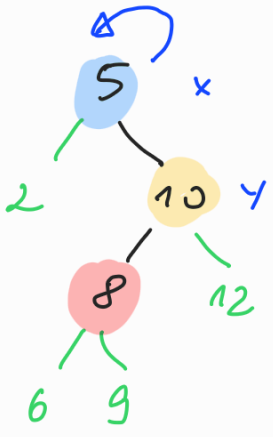


$v.right = NIL$

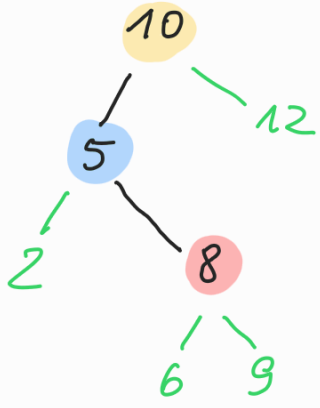
$$BF(v) = T(NIL) - T(v.left)$$
$$= -1 - 1$$
$$= -2$$

Example Rotation:

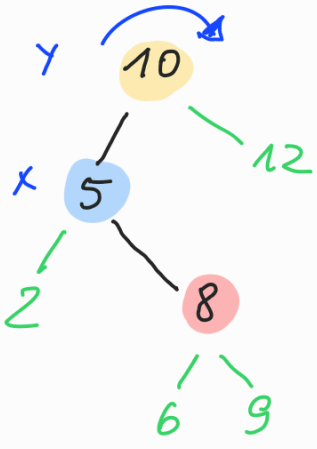
Search tree:



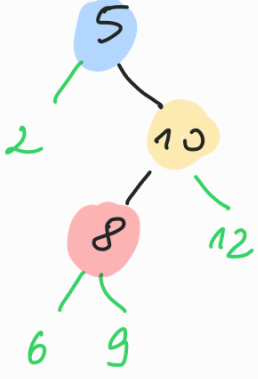
left-rotate (5)



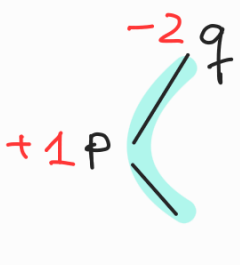
green part "remains unchanged"



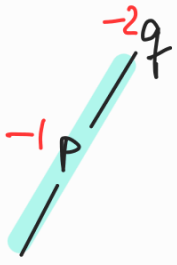
right-rotate (10)



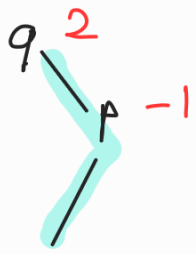
BF in T+x



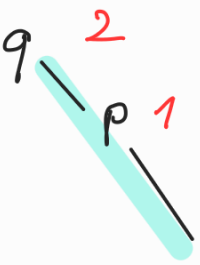
leftrot(p)
rightrot(q)



rightrot(q)



rightrot(p)
leftrot(q)

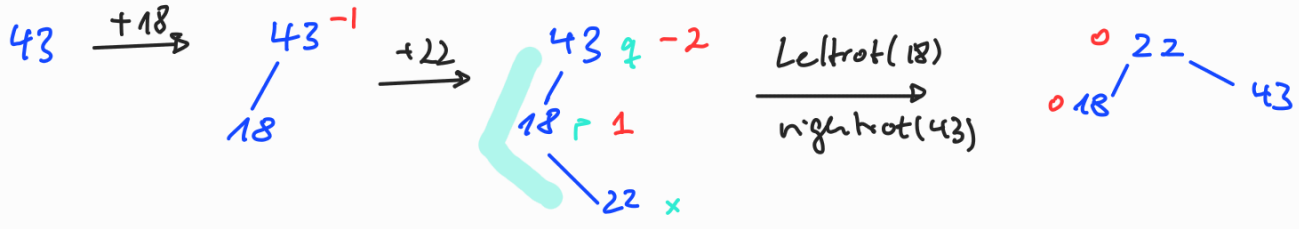


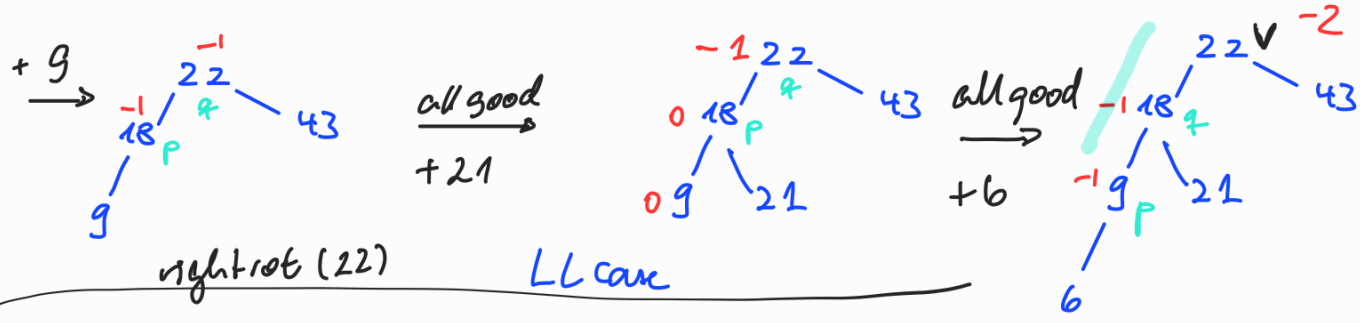
leftrot(q)

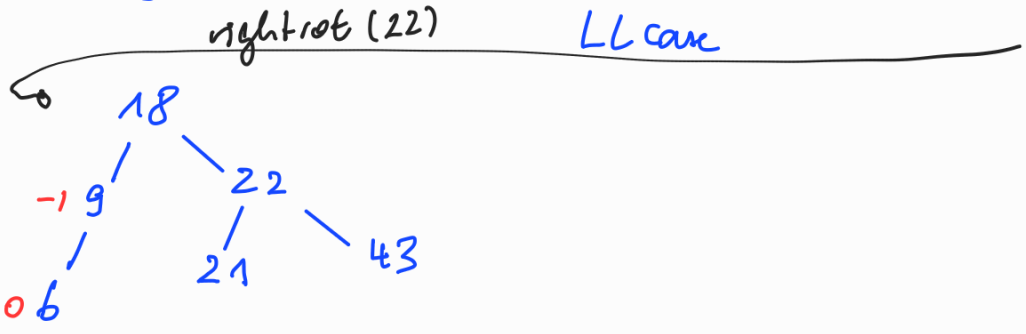
---

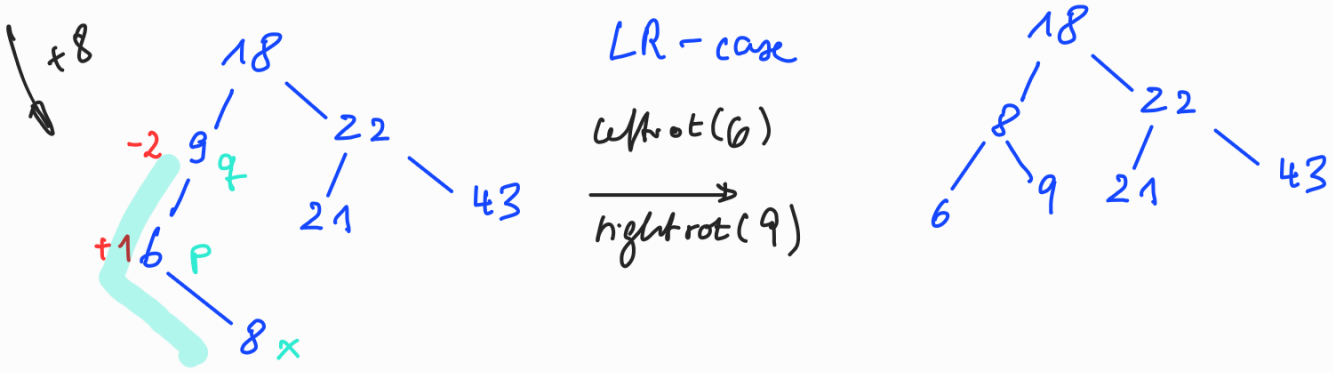
insert to AVL

Exmpl: BEI

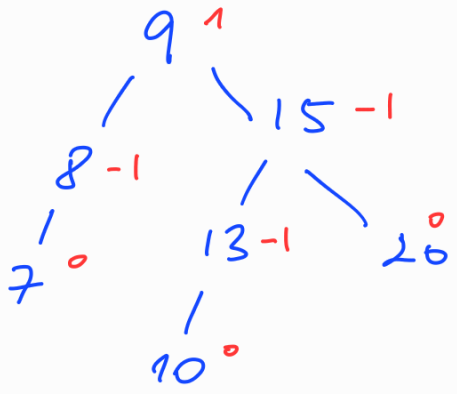






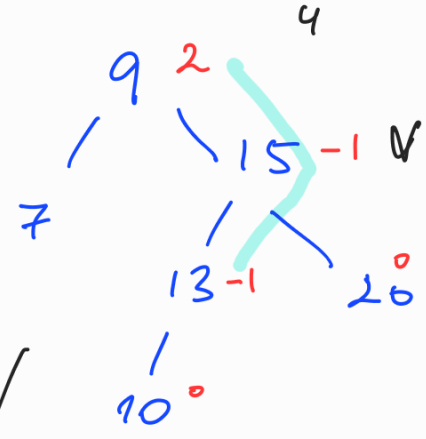


delete from AVL



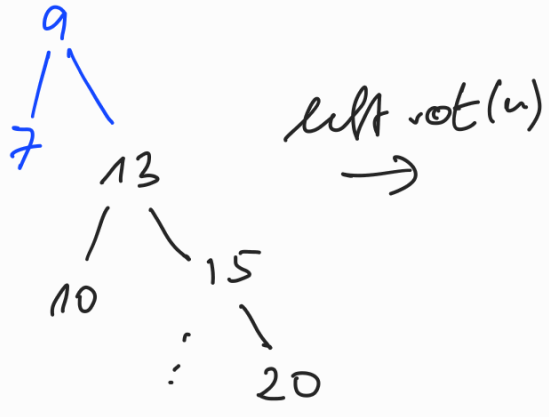
RL-case

delete 8:

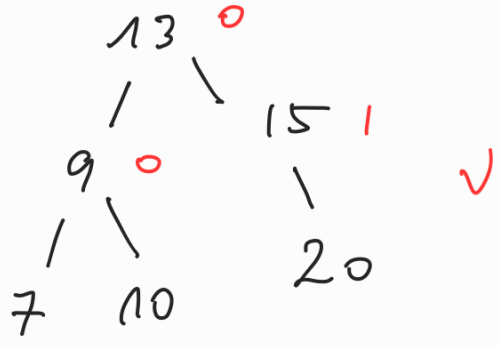


Rightrot (V)
+ leftrot (u)

rightrot(v)



leftrot(u)



RED-BLACK-trees

Lemma: red-black with n internal nodes has height $O(\log n)$

show first:

CLAIM. subtree $T(x)$ rooted at x contains at least $2^{bh(x)} - 1$ internal nodes.

proof:

By induction on height of x

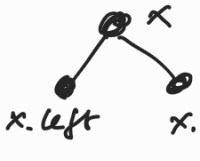
IF $h(x) = 0$ $\Rightarrow$ x leaf $(= NIL)$

$\Rightarrow$ $bh(x) = 0$

$\Rightarrow$ $2^0 - 1 = 0$ intern. nodes ✓

Now let $h(x) > 0$ $\Rightarrow$ x not leaf $(\neq NIL)$

By construction, x must have 2 ch. (Idea. [NIL possible]



if child z of x black, then z adds „+1" to $bh(x)$

red, then z adds „+0" to $bh(x)$.

$$\Rightarrow bh(z) = \begin{cases} bh(x) - 1, & \text{if } z \text{ black} \\ bh(x), & \text{if } z \text{ red} \end{cases}$$

Note $h(z) < h(x)$ $\Rightarrow$ can apply ind. assumption:

$\Rightarrow$ $T(z)$ has at least $2^{bh(z)} - 1$ nodes

as $bh(z) \geq bh(x) - 1$

$\Rightarrow$ $T(z)$ has at least $2^{bh(x)-1} - 1$ nodes.

Since x has 2 children

$\Rightarrow$ $T(x)$ has at least

$2 \cdot \left(2^{bh(x)-1} - 1\right) + 1$ internal nodes

(2: 2 children; 1: x)

$= 2^{bh(x)} - 1$ internal nodes.

——— End-proof-of CLAIM ———

To complete proof of Lemma, let h be height of the tree.

According to property 2 (each red vertex has black children)

$\Longrightarrow \geq \frac{1}{2}$ vertices on any simple path from root to a leaf (not incl. root) must be black

$\Rightarrow$ $bh(root) \geq \frac{h}{2}$

CLAIM $\Rightarrow$ $n \geq 2^{bh(root)} - 1 \geq 2^{\frac{h}{2}} - 1$

$\Rightarrow$ $\log_2(n+1) \geq \frac{h}{2} \Rightarrow h \in O(\log(n))$ □

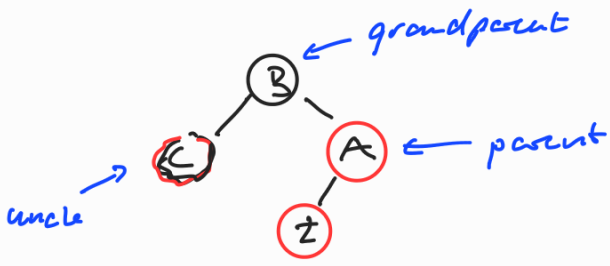
Exercise Show longest simple path from root to leaf in Rb-tree is $\leq$ 2. length of shortest path from root to leaf.

[shortest possible → all nodes black
longest → all nodes red/black alternating]
(same nr black nodes!)

RED-BLACK-TREE EXAMPLE (insert)

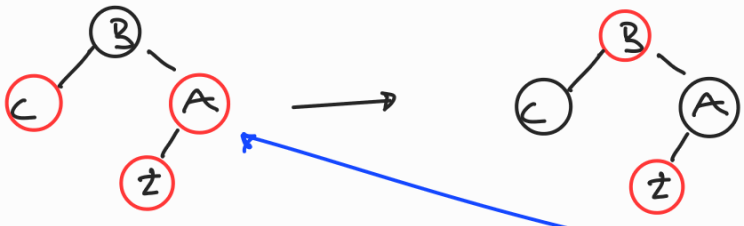
Case 1.b) z root of T ⇒ recolor z to RED

2) parent of z is RED:



A, B, C, z are names of nodes, not their values, which are A.key, B.key, C.key, z.key.

i) uncle RED: Recolor A, B, C

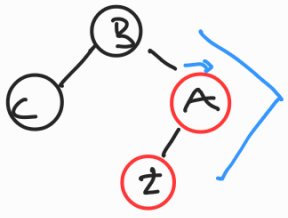


→ recheck properties for B

[Line 8 in common: z = z.p.p.]

z = B"

ii) uncle BLACK (triangle)



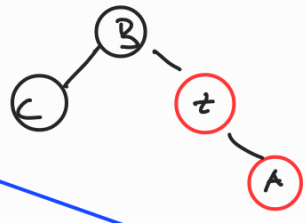
rotate A →

oppos. of z's pos. to A.

here: z left of A

⇒ right-rotate (T, A)

(or "⟨")

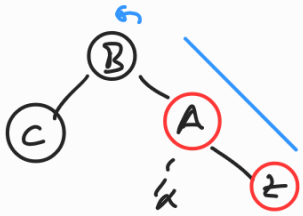


⇒ Case (iii) with A playing role of z.

[Line 10 in common: z = z.p]

z = A

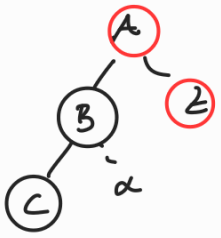
iii) uncle Black (Line)



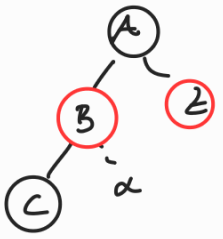
(or "/")

rotate B →

in oppos. of z's pos. to A.

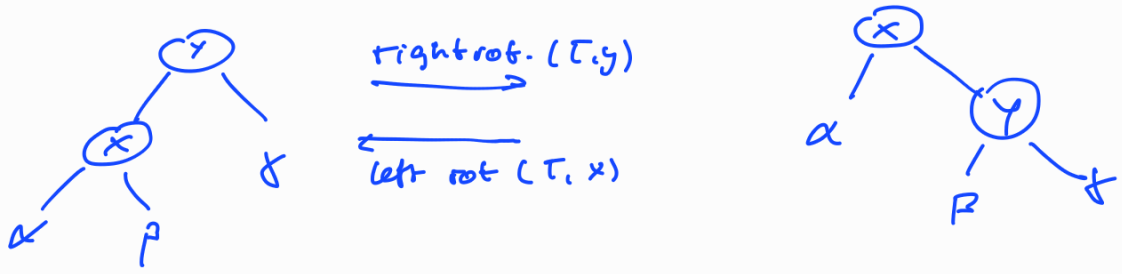


recolor A ↔ B



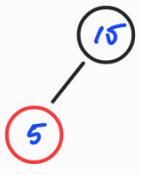
done.

Recap:



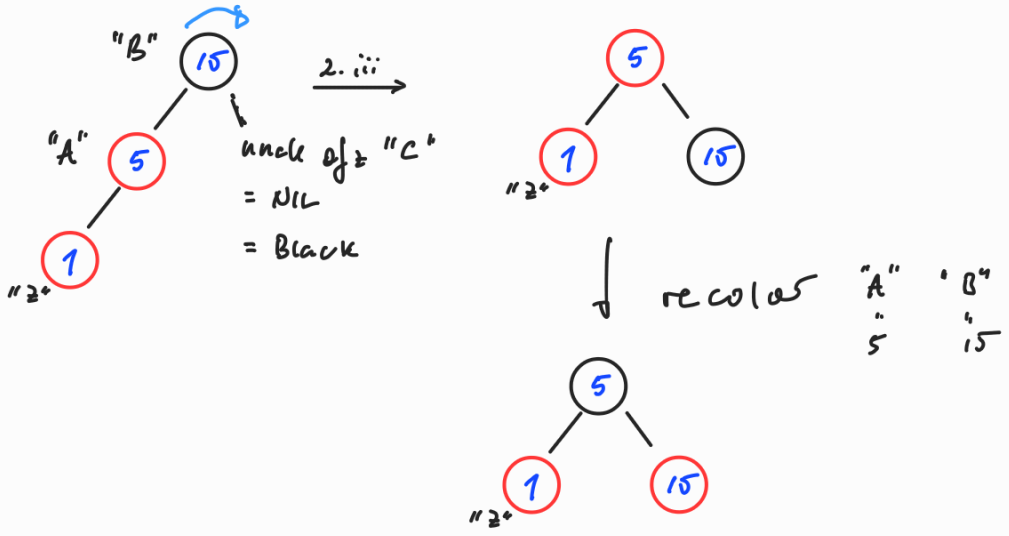
insert 15: 15 —1.b→ 15 ("+ T.NIL")

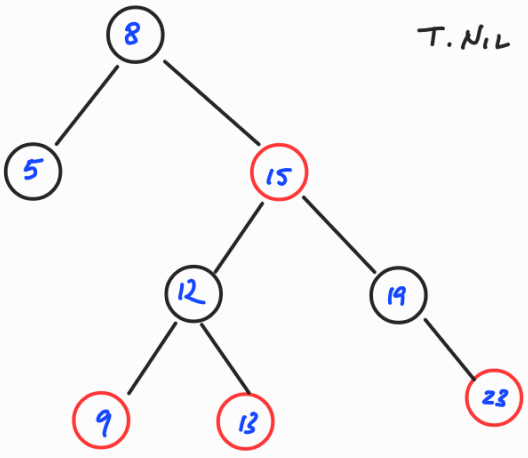
insert 5:



nothing violated!

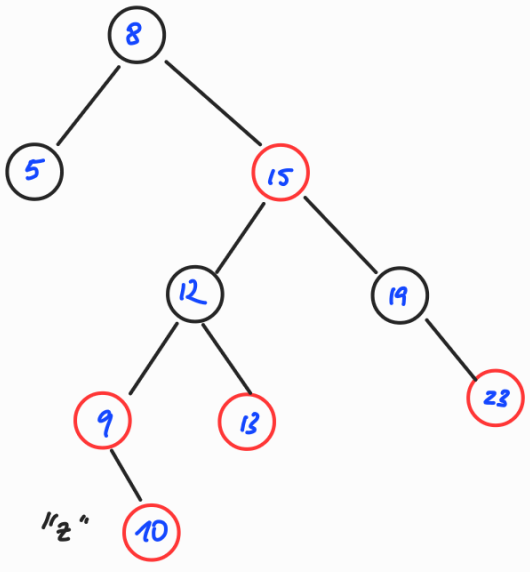
insert 1:



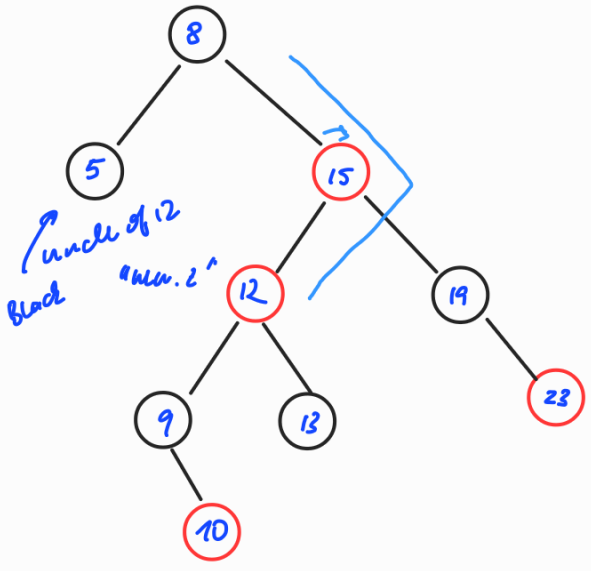


T. NIL omitted

insert 10:

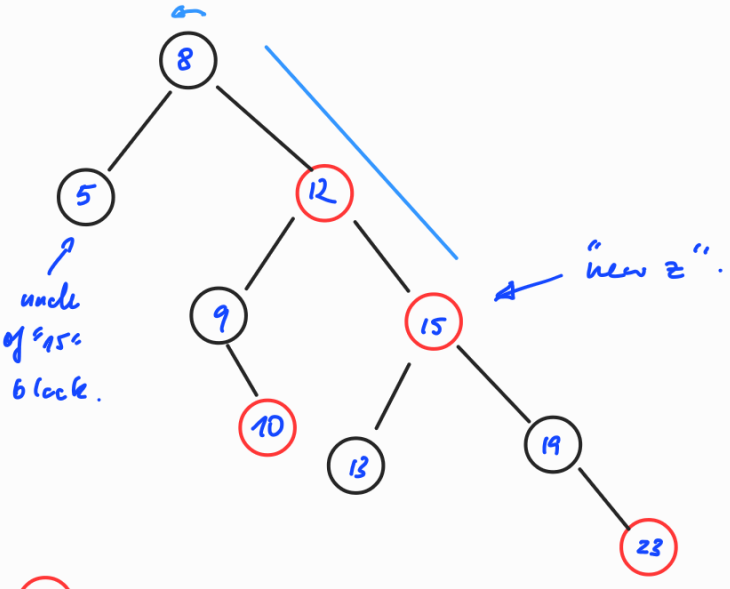


⇒ case 2.i (recolor)

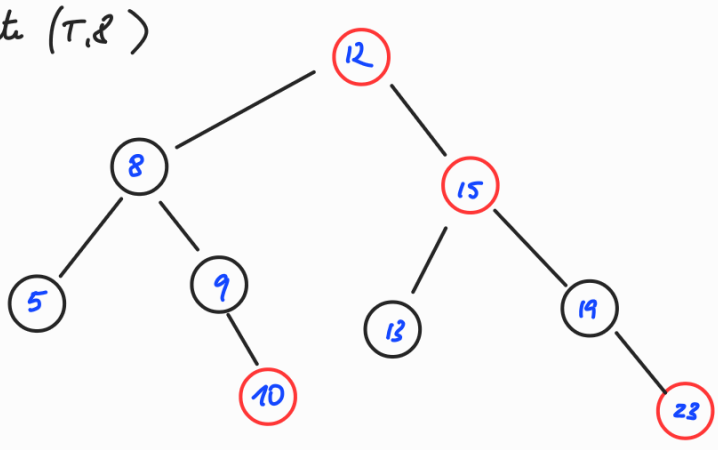


recheck for "new.z = "12"

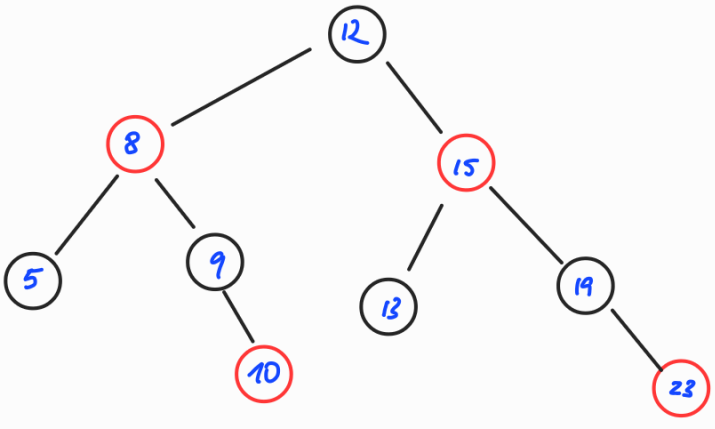
⇒ right rotate (T, 15):



⇒ left rotate (T, 8)



+ recolors:



done.