Red – Black Trees

BST operation have time complexity $O(h)$

height of a bst with n nodes may range anywhere from $O(\log n)$ to $O(n)$

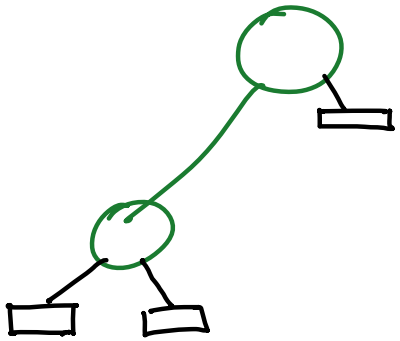
There are many schemes to maintain the height of a BST $O(\log n)$. Red-Black tree is one of them.

balanced BSTs

R-B trees are BST,

In a BST consider nil values as separate nodes.

These are called external nodes



▭ are exteral nodes / leaves

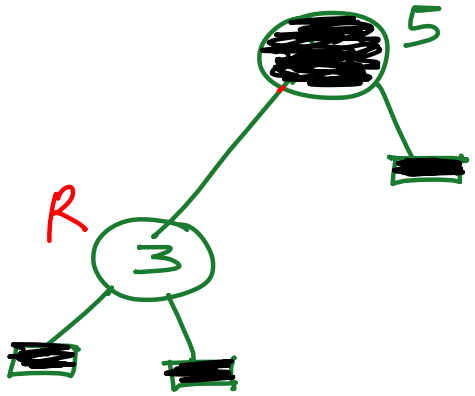
◯ internal nodes

Each node has either no child or exactly two children

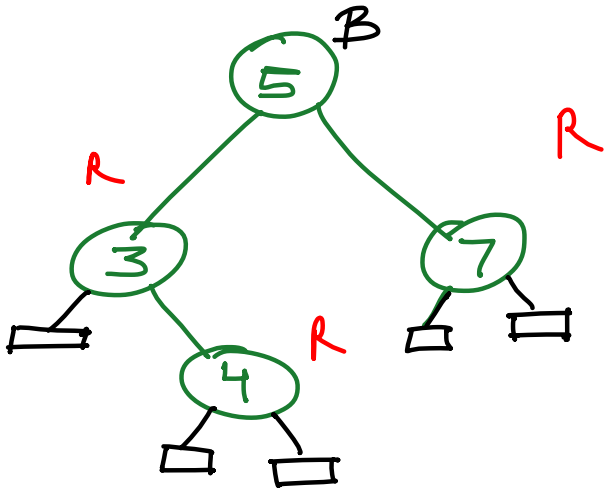
Def R-B trees are BST with the following properties

1. Each node is either red or black
2. Root node is black.
3. All the external (leaf) nodes are black
4. Any red node has both its children black
5. For any x is a R-B tree, all simple paths from x to descendent leaves contain the same no of black nodes.

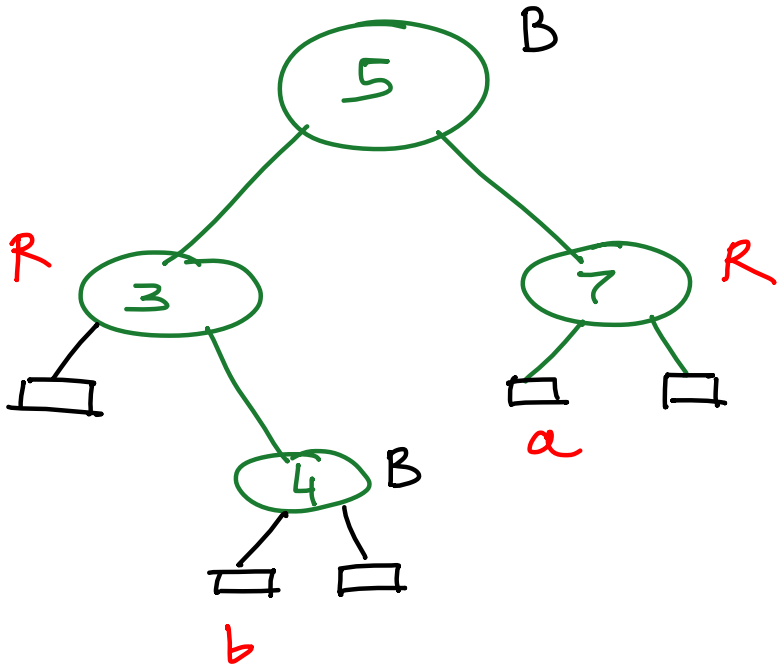
Ex 1



Red – Black Tree



Not a R–B tree because red node 3 has a red child



Not a R-B tree

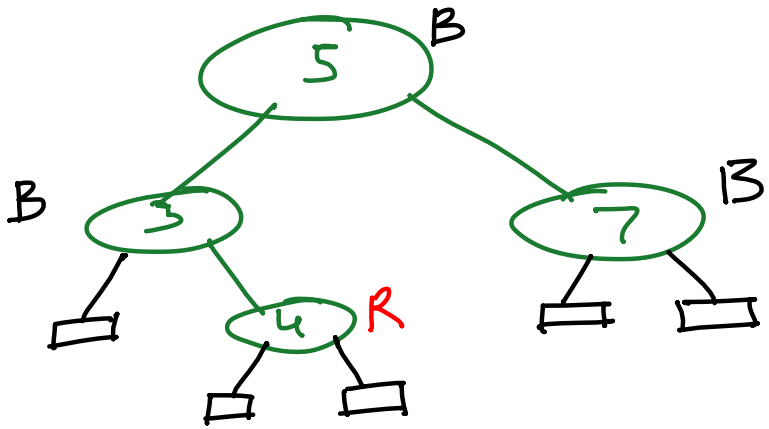
path from 5 — a

Contains 2 black nodes

path from 5 — b

Contain 3 black nodes

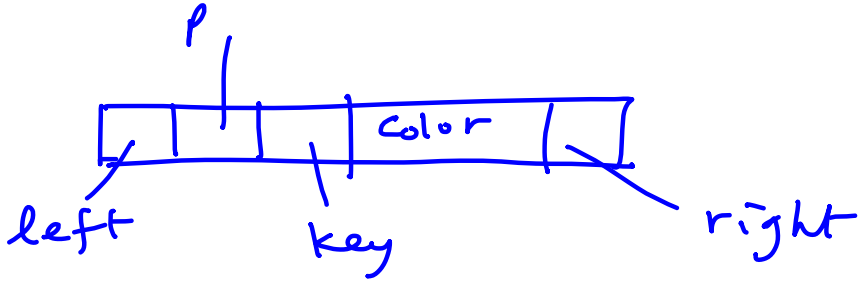
(5, 4, b)



← Red — Black Tree

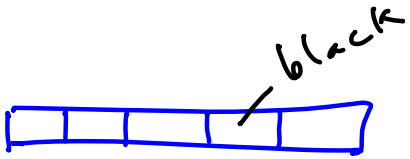
Representation of R-B trees (as data structure)

Each node

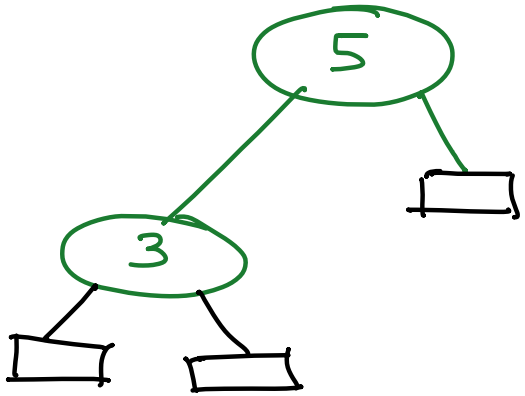


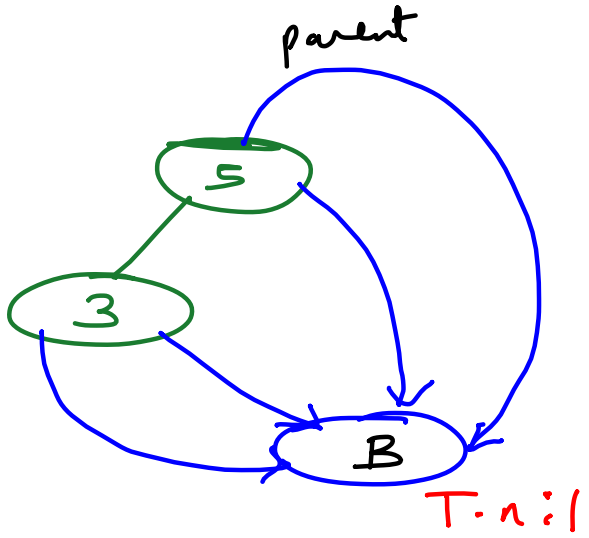
We represent each external node by a single sentinel node which is named T.nil

T.nil

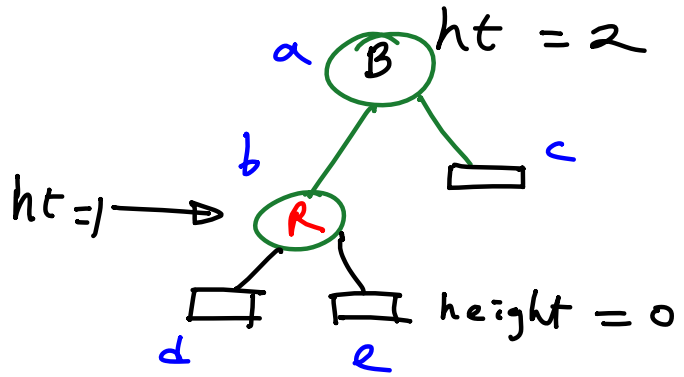


other fields of T.nil may contain any value.


5
3


parent
5
3
B
T·n:1

height of a node in a tree



Black height of a node x in R-B Tree T.

[No. of black nodes in any simple path from x to any descendent leaf (not counting x)

Well defined because of property 5 of R-B trees.

bh(d) = bh(e) = bh(c) = 0

bh(b) = 1

bh(a) = 1

bh(a)

$2^{bh(a)} - 1 = 2^1 - 1 = 1$

T_a has two internal nodes (a, b)

T_b has 1 internal node (b)

Lemma: In a R-B tree, for each node x, T_x has at-least $2^{bh(x)} - 1$ internal nodes.

Pf: By induction on height of x.

Base Case: $h(x) = 0$. x is an external node

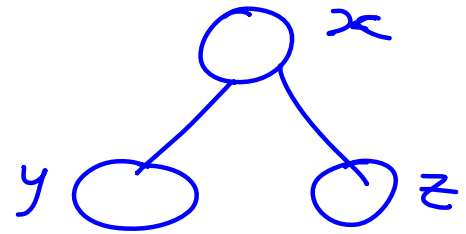
$bh(x) = 0$, $2^{bh(x)} - 1 = 2^0 - 1 = 1 - 1 = 0$

T_x has 0 internal node. ∎

Induction Step

x is internal $ht(x) = h$

x has two children y, z



If y/z have black color
then $bh(y/z) = bh(x) - 1$

If y/z have Red color
then $bh(y/z) = bh(x)$

[of course, y and z may have different colors]

No. of internal nodes in T_x

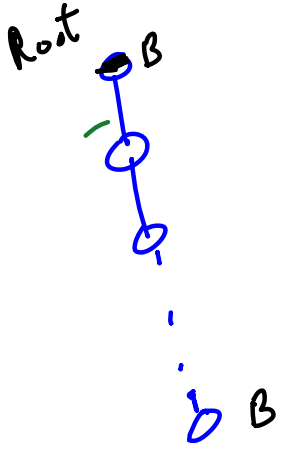
is $1 + (\text{no. of internal nodes in } T_y) + (\text{no. of internal nodes in } T_z)$

$\geq 1 + (2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) = 2^{bh(x)} - 1$ □

Lemma: If a R-B tree has n internal nodes then its height $h \leq 2\log(n+1)$

In Single R-B example
$n = 2$, $\log(n+1) > 1$
$2\log(n+1) > 2$
$h = 2 \leq 2\log(n+1)$

Proof Consider a path of length h from root to a leaf.



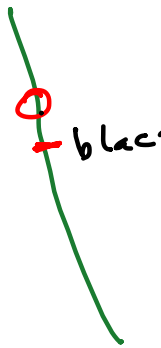
$bh(root) = m$

No. of red nodes in this path $\leq m$

Total no. of nodes in this path

$h+1 \leq 1+m+m$

$h+1 \leq 2m+1 \implies h \leq 2m$



For each red node in the path there is a black child node

Using Lemma 1,

$$n \geq 2^m - 1$$

$$2^m \leq n+1$$

$$\Rightarrow \quad m \leq \log_2(n+1)$$

In the last page $h \leq 2m$

$$\leq 2\log_2(n+1)$$

□

Corollary: In a red-black, there are $O(\log n)$ algorithms to find min, max, succ, pred.

pf We use BST algorithms for the same.

For insert/delete the bst algorithms do not work directly.

This is because insert/delete operations in a bst may not preserve the Red-black property of the original tree.

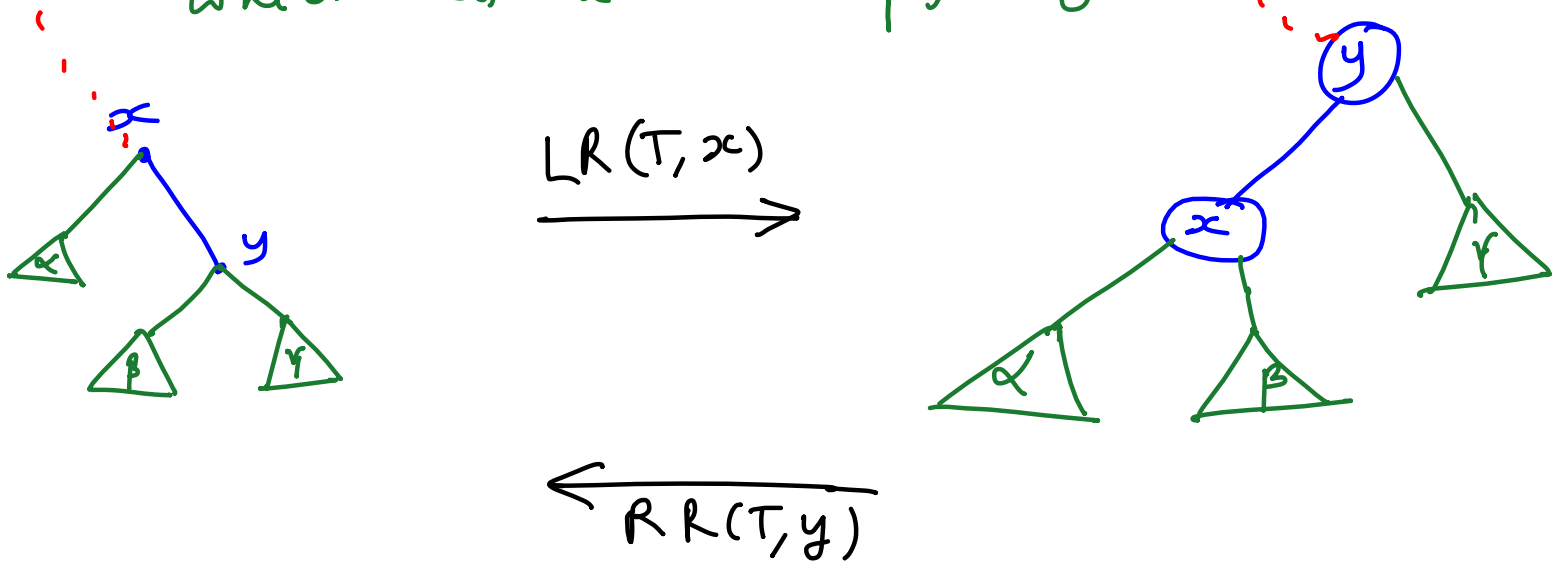
In next couple of lectures, we will see that bst algorithm can be adapted for Red-black trees also

Rotations (for bst)

Left – Rotation
Right – Rotation

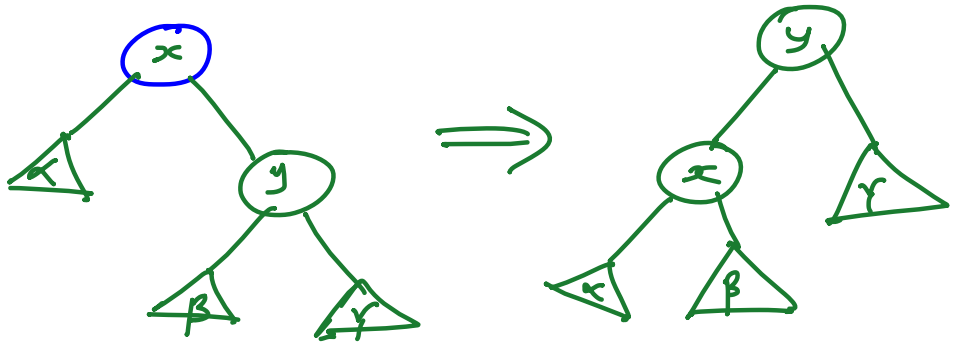
[These will be used in R-B insertion/deletion algorithms]

In a bst, a left rotation is defined for any node x which has a non-empty right child.



Verify that LR(T, x) is a bst.

Pseudo code for LR(T, x)




```
LR(T, x)
y = x.right
Transplant(T, x, y)
x.right = y.left
if (y.left) ≠ T.nil
    (y.left).p = x
y.left = x
x.p = y
```

→ uses T.nil instead of nil

→ this check is not required because of sentinel node