We will now consider data structures which are used to store and lookup data in an efficient manner.

We may want these data structures to be dynamic, that is, we should be able to insert and delete elements as needed.

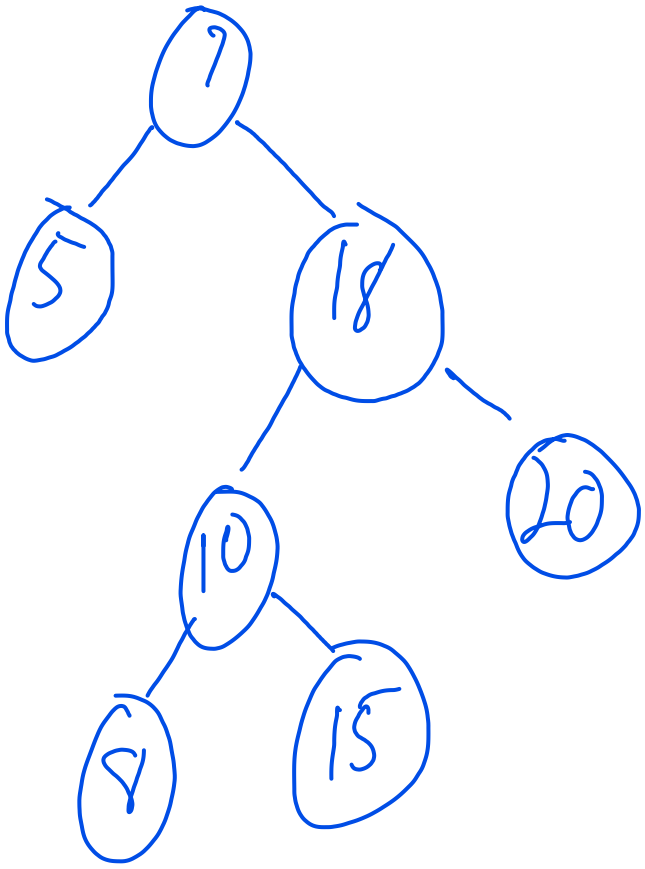
We want to be able to perform various operations on the data structure. For example:

- Search
- Max
- Min
- Insert
- Delete

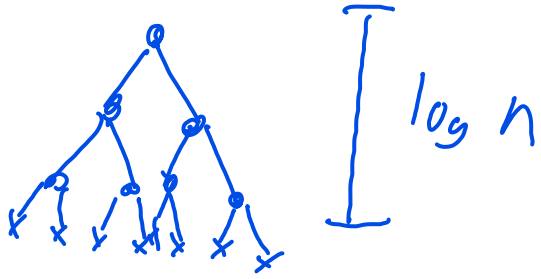
We want these operations to be computed as quickly as possible.

A **binary search tree** is a special type of binary tree in which each node stores a value such that the following properties hold for a node x:

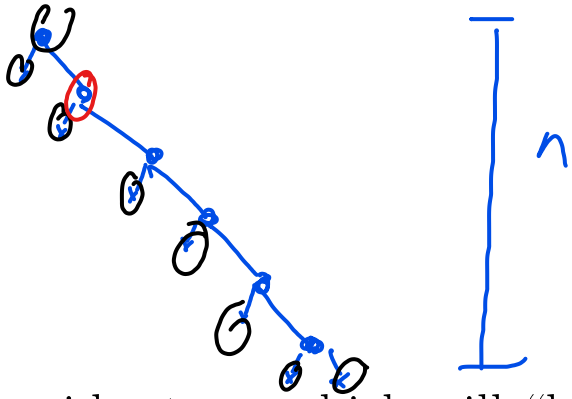
1. For each node y in the left subtree of x, we have $y \leq x$.
2. For each node y in the right subtree of x, we have $x < y$.



If a binary tree with n nodes has height $O(\log n)$, then we can find any node in the tree (or determine that it isn't there) in time $O(\log n)$.



Since we want the tree to be dynamic, a "bad case" sequence of inserts can leave us with a tree of height n.

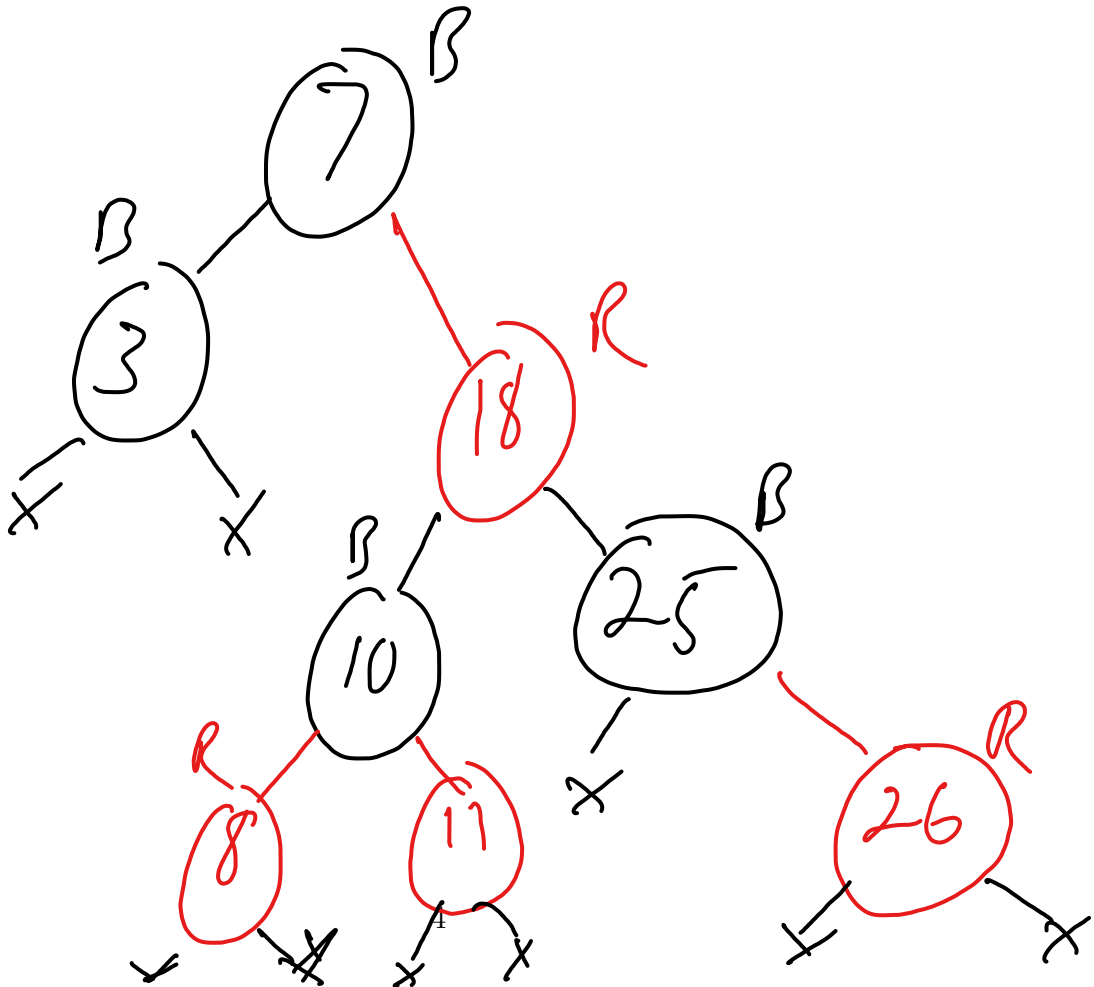


We will consider trees which will "balance themselves" in the event that we see a bad sequence of inserts (or deletions).

Red-Black Tree

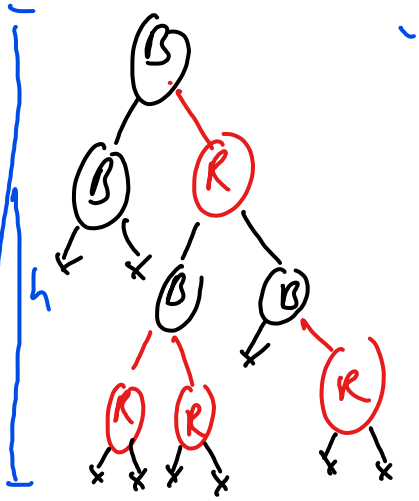
Satisfies these properties

1) Every node is red or black.
2) Root is black.
3) Leaves (nil) are black.
4) If a node is red, both children are black.
5) All simple paths from any node x, excluding x, to a descendant leaf has the same # of black nodes.
   black-height(x)

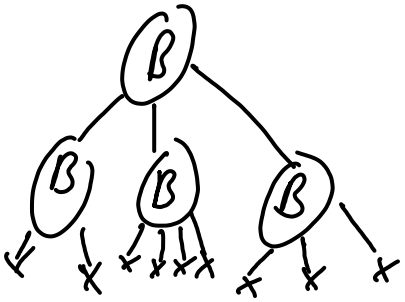


Theorem

A RB-tree with n key has height $h \le 2 \cdot \log(n+1)$



"Merge" red nodes into black parents $\Rightarrow$



perfectly balanced.

For each path of length h, there are at most $h/2$ red nodes (Property 4)
So $h' \ge \frac{h}{2}$.

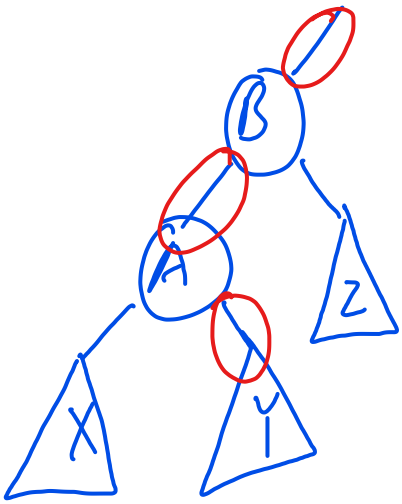
Right tree has all nil leaves are at the same level (Property 5), so it is perfectly balanced.

The # of nil leaves is $\le n+1$

$n+1 \ge 2^{h'} \Rightarrow \log_2(n+1) \ge h'$.

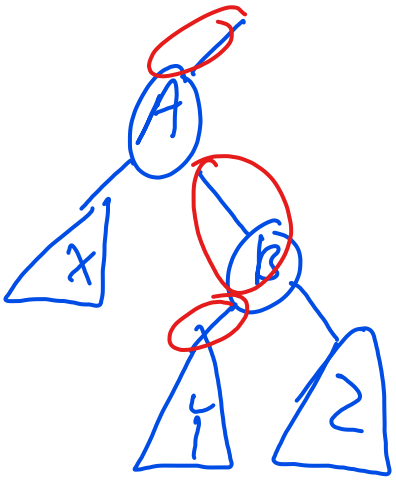
$\log_2(n+1) \ge h' \ge h/2 \Rightarrow$ $\boxed{h \le 2\log_2(n+1)}$

Rotations



Right-Rotate(B)

Left-Rotate(A)



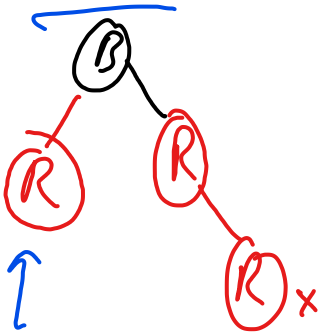
Let $x \in X$, $y \in Y$, $z \in Z$

$$x \leq A \leq y \leq B \leq z$$

Insert(x) - Insert x into tree as a leaf and color it red.

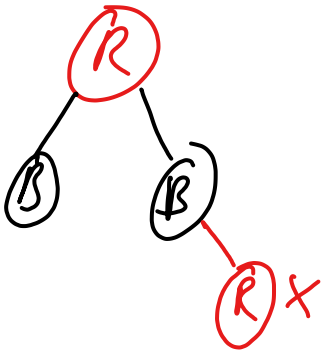
Only Property 4 can be violated or we are done. Assume a violation.

Case 1



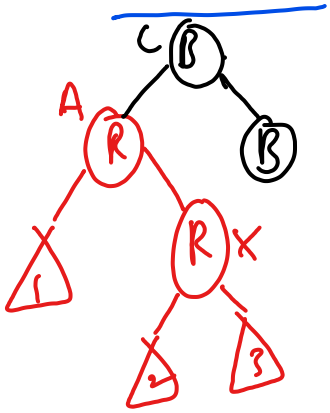
Check aunt/uncle.
If red, we are in Case 1.

Swap colors for parent, grandparent, and aunt



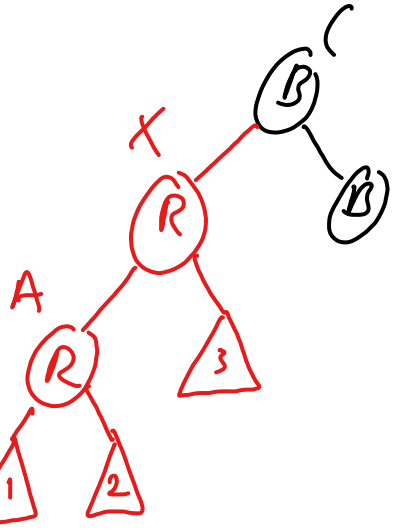
Recursively repeat with grandparent as new x.

Case 2



aunt is black & path from grandparent to X is "zig-zag"

⇒ Left-rotate(A)

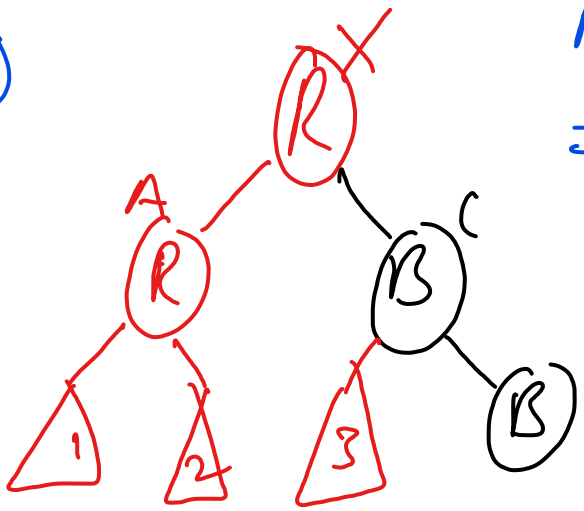


Case 3

aunt is black & path from grandparent is straight

Right-rotate(C)

⇒



Recolor X & C

⇒