6.046
Lecture 10

Red-Black trees, Rotations, Insertions, Deletions. (1)

Balanced search trees

Search tree data structure maintaining dynamic set of n elts using tree of height $O(\lg n)$.

Examples:
- AVL trees
- 2-3 trees
- 2-3-4 trees
- B-trees
- Red-black trees
- Skip lists
- Treaps

Red-Black trees

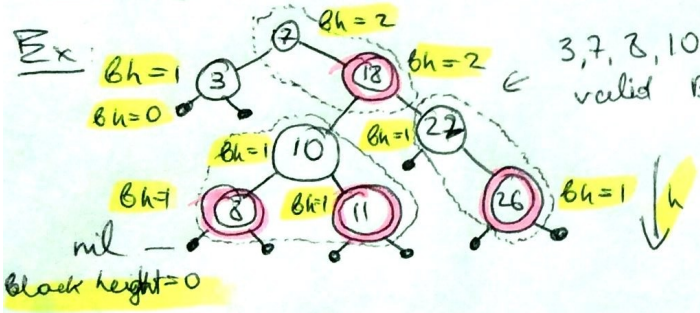
BST data structure with extra color field for each node, satisfying:

Red-Black properties

1. Every node is either red or black.
2. The root & leaves (nil's) are black
3. Every red node has a black parent
4. All simple paths from a node x to a descendant leaf of x have same # black nodes = black height (bh)

Black height does not count ⊗ if self

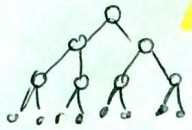
Ex:



3, 7, 8, 10, 11, 18, 22, 26 ✓ valid BST

a) These properties should force the tree to have $O(\lg n)$ height

easy to start: just build a BST with all black nodes

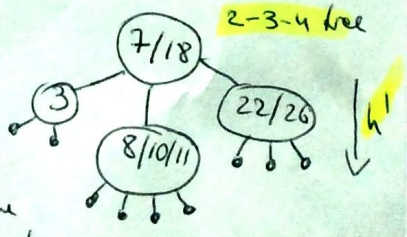


b) these properties are easy to maintain in a dynamic setting

Height of red-black tree:

Red-black tree with n keys has height $h \leq 2\lg(n+1) = O(\lg n)$

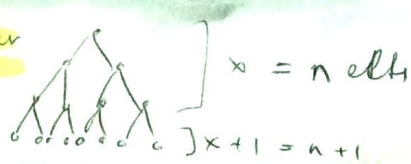
Proof sketch:
- merge each red node into its black parent node
- every internal node has 2, 3, or 4 children nodes
- every leaf has the same depth = bh of the root (by property 4)



(nil) lecture 10

leaves = n+1 in either tree

(for branching factor of 2 in the original tree.



(2)

In 2-3-4 tree:

$2^{h'} \le$ # leaves $\le 4^{h'}$ $\Rightarrow$ $2^{h'} \le n+1$

$\Rightarrow$ $h' \le \lg(n+1)$

$h \le 2h'$ $(h' \ge \frac{1}{2} h)$

↖ at most one red node for every black node on any path (property 3) from root to leaf.

$\Downarrow$

$h \le 2\lg(n+1)$

all red-black trees are balanced ✓ $\Leftarrow$

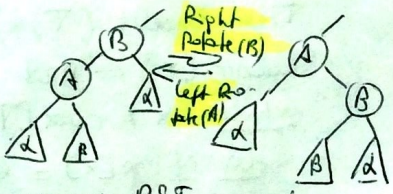
Corollary Queries (Search, Min, Max, Successor, Predecessor) run in $O(\lg n)$ time in a red-black tree. → easy to query

Updates (Insert & Delete)

must modify the tree.

- BST operation (tree insert, tree delete)
- color changes
- restructuring of links via rotations, constant time operations

Rotations:



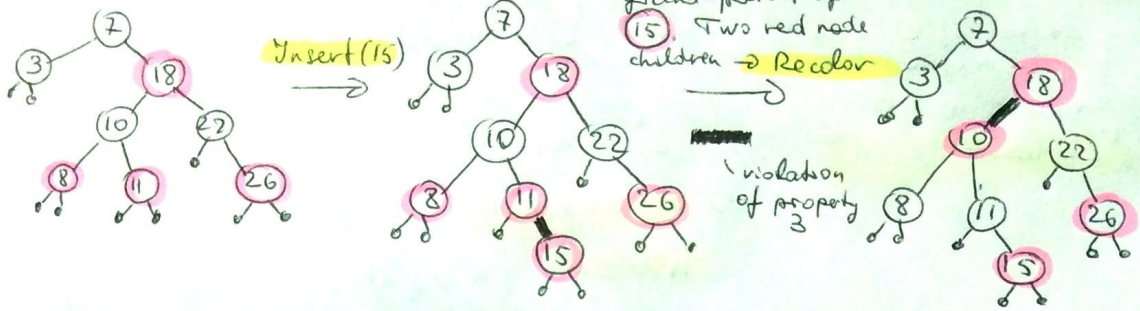
preserves BST property

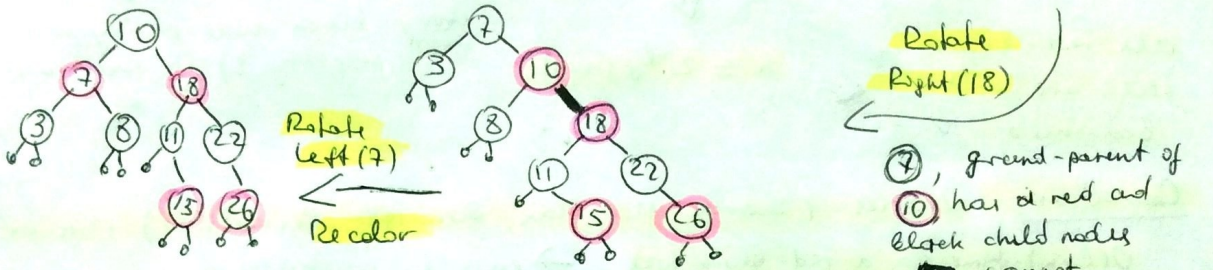
$\forall a \in \alpha, b \in \beta, c \in \gamma \quad a \le A \le b \le B \le c$

RB-Insert(x) : my comment similar to AVL

Idea:
- Tree-Insert(x) → would be a leaf in a BST, but gets two black leafs in red-black tree.
- color node red
- problem if parent is red (violate property ③)
  But property ④ still holds. [redacted]
- move violation of ③ up the tree via recoloring of nodes until we can fix violation via rotation & recoloring

Ex Insert (15)



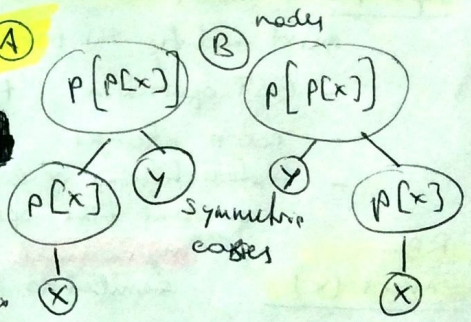


7, grand-parent of 10 has a red and Black child nodes cannot recolor 7 and its child node

RB-Insert(T, x)

```
Tree-Insert (T, x)
color[x] ← Red                                  // Red
while x ≠ root[T] and color[x] = Red
  do if p[x] = left[p[p[x]]]                    // A
    then y ← right[p[p[x]]]
         if color[y] = Red
         then <Case 1>
         else if x = right[p[x]]
              then <case 2>
              <case 3>
    else (B)
         same as (A), but reversing
              left ↔ right
color[root[T]] ← Black
```

Rotations



B is symmetric

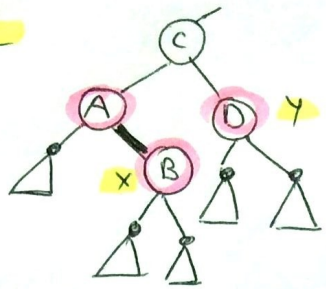
my comment

AVL tree cases also symmetric, and comparisons are two levels up

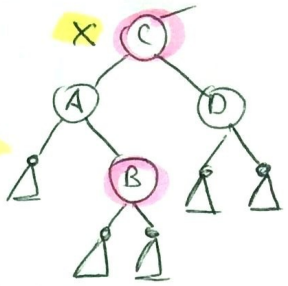
Lecture 10

3 cases of A — △ has black root & all △ have same bh (3)

Case 1



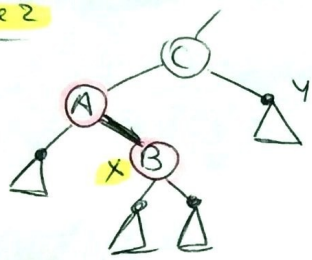
recolor →



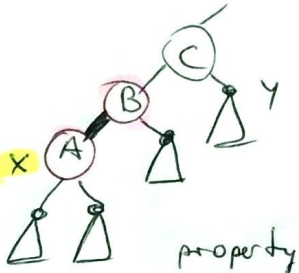
preserves property (4)
fixes property (3) locally.

in case 1 x can be right(A) or left(A)

Case 2

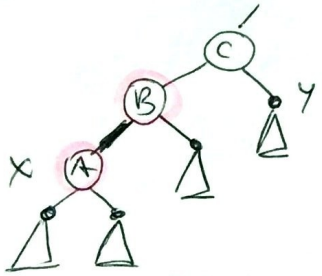


Left-Rotate(A) →

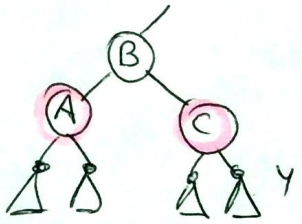


property (4) preserved

Case 3



Right-Rotate(C) →
Recolor



property (3) is fixed
property (4) preserved

RB-Insert: adds x to set and preserves the properties of RBTs

Terminates after case 2 and 3, only case 1 can continue upward traversal.

case 1, does not change tree structure by rotations, only recoloring of nodes and moves x up by 2 levels → O(lg n) O(1) in AVLs

Rotations are more compute-intensive (AVL after insert, but not delete)

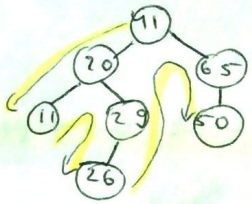
O(1) both insert & delete → const # in RBTs

my comment: but AVLs h < 1.440 lg n + log_φ √5

6.006

BSTs

- rooted binary tree
- each node has [redacted]
  - key
  - left pointer
  - right pointer
  - parent pointer
- BST property



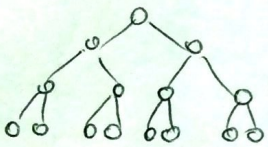
sorted order: in-order traversal

11, 20, 26, 29, 41, 50, 65

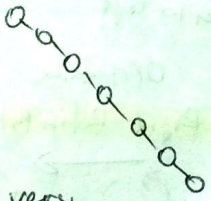
BST ops

insert, delete, min, max,
next larger / smaller (successor / predecessor) in $O(h)$ time

Balanced or not



balanced if $h = \Theta(\lg n)$



very unbalanced

h = length of longest path from root to leaf (down)

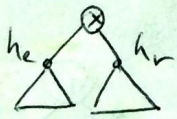
height of a node: longest path from the node [redacted] down to a leaf

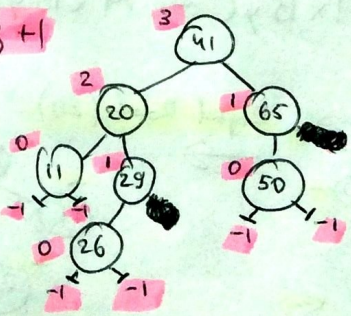
$= \max\{(\text{height left child}), (\text{height right child})\} + 1$

AVL trees

require heights of left and right children of every node to differ by at most ± 1

$|h_l - h_r| \leq 1$





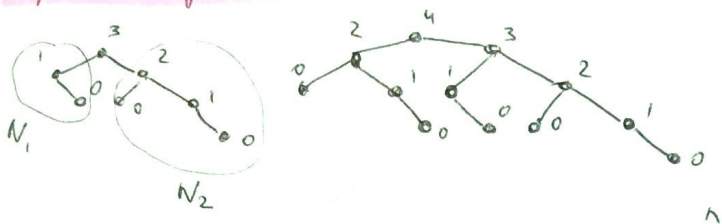
−1 from terminals/leaves → $\max\{-1, -1\} + 1 = 0$

AVL trees are balanced

worst case is when right subtree has height 1 more than left for every node.

N_h = min. # nodes in an AVL tree of height h.

my drawing: AVL worst case



$N_1 = O(1)$

$N_h = 1 + N_{h-1} + N_{h-2}$

right left

$n = N_h > F_h = \frac{\varphi^h}{\sqrt{5}}$ 1.618 $\varphi > 1$

Fibonacci

Alternative analysis (less tight)

$N_h = 1 + N_{h-1} + N_{h-2}$

$> 1 + 2N_{h-2}$

$> 2N_{h-2}$

$= \Theta(2^{h/2})$ $h < 2\lg n$

$\frac{\varphi^h}{\sqrt{5}} < n$ $\log_\varphi\left(\frac{\varphi^h}{\sqrt{5}}\right) < \log_\varphi(n)$

↑ monotonically increasing

$h - \log_\varphi \sqrt{5} < \log_\varphi(n) \approx 1.440 \lg n$

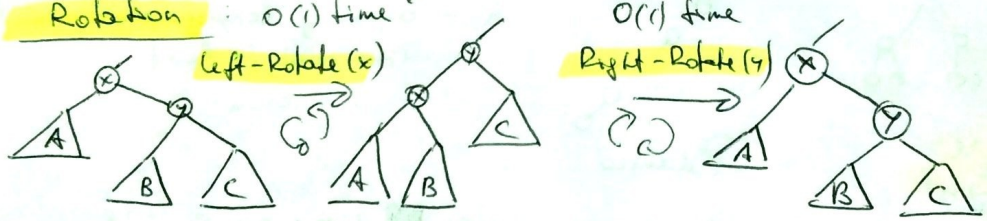
↑ close to $\lg n$

AVL insert

① simple BST insert [redacted] always (leaf insertion)

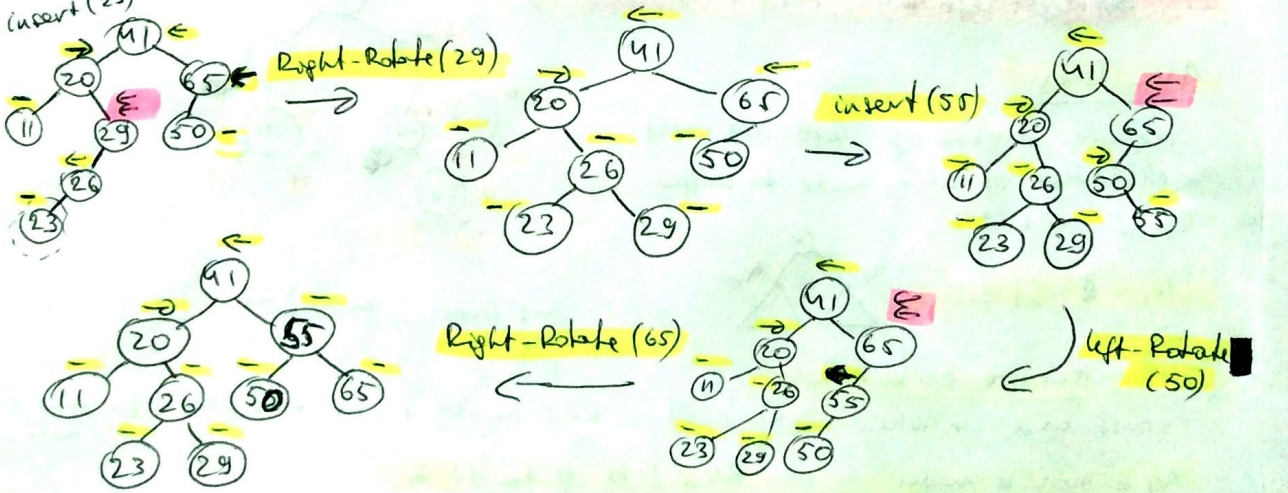
② fix AVL property from the changed node up

Rotation : $O(1)$ time ← root x goes to left



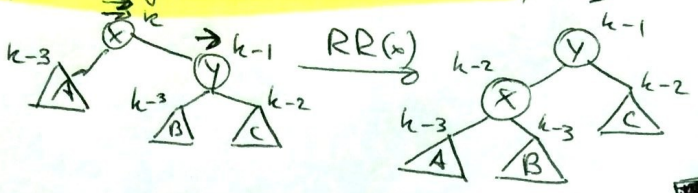
in-order traversal: $AxByC = AxByC = AxByC$

insert(23)



Lecture 6 6.006

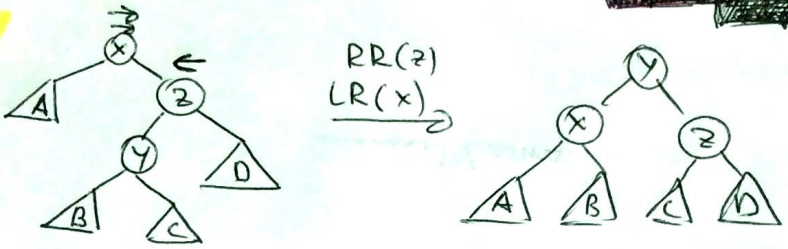
- suppose x of lowest node violating AVL property from inserted node (leaf)
- assume right(x) higher (if left(x) higher symmetric) see above example
- if x's right child is right-heavy →



my comment
1 or 2 rotations for insertion needed

may need to go up and fix y's parents

- else:



AVL Sort

- insert n items - $\Theta(n h) \overset{\text{AVL prop.}}{=} \Theta(n \lg n)$
- in-order traversal - $\Theta(n)$
  in contrast to heaps, also get successor/predecessor

Abstract Data Type

- insert/delete
- min/max
- successor/pred.

} priority queue } heap, AVL

Data Structure

balanced BST