Lec 10 平衡搜索树

balanced search tree: search tree data structure maintaining a dynamic set of n elements, using a tree of height $O(\lg n)$ (不一定是二叉)

Examples:

- AVL trees 1962
- 2-3 trees 1970
- 2-3-4 trees
- B-trees
- red-black trees
- skip lists
- treaps (树堆) 1996

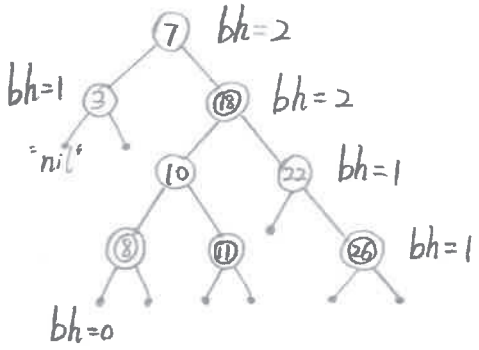
red-black trees

BST data structure with extra information in each node called the color field, and there are several properties that a tree with a color field has to satisfy in order to be called a red-black tree, 这些性质被称为红黑性 (red-black properties)

red-black properties

1. every node is either red (0) or black (0)
2. the root and the leaves (nils) are all black
3. every red node has black parent
4. all simple path (不重复任何结点), from a node x to a descended leaf of x, have the same number of black nodes on them, #black-nodes = black-height(x)
   does not count x itself

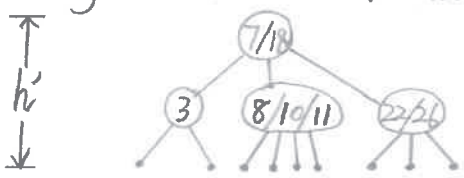
Example:



height of red-black tree

a red-black tree with n keys has height $h \le 2\lg(n+1) = O(\lg n)$

proof sketch: merge each red node into its black parent



"2-3-4 tree"

此时，所有的"nil"叶结点都有相同的深度（性质4），等于 black-height(root)。"balanced!"
每个内部结点，都有2到4个子结点。

proof:

$\#leaves = n+1$ (in either tree)

in 2-3-4 trees, $2^{h'} \le \#leaves \le 4^{h'}$

so $2^{h'} \le n+1$

$h' \le \log_2(n+1)$

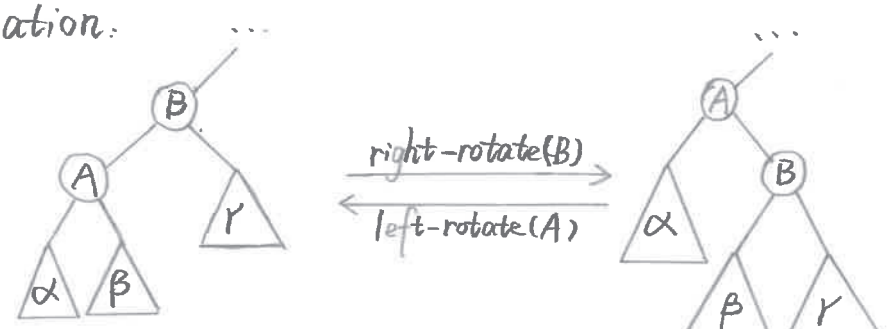
而性质3告诉我们，每个红结点只能连着黑结点，所以最多能红黑相间，故一条路径上红结点数最多是总结点数的一半，而所有路径里最长的那条，就是树的高度。

$h \le 2h' \le 2\lg(n+1)$ Q.E.D.

Corollary:

- Queries (search, min, max, successor, predecessor) can in $O(\lg n)$ time in a red-black tree
- Updates (insert, delete) modify the tree
  - BST operation
  - color changes
  - restructuring of links via rotations, $O(1)$ time

Rotation:



preserves BST property

$\forall a \in \alpha, b \in \beta, c \in \gamma \;.\; a \le A \le b \le B \le c$ ✓

RB-Insert(T, x):

// idea: Tree-Insert(x), color the node as red ← 如果设为黑色，则会扰乱 black-height

// problem: parent might be red ⇒ violate property 3

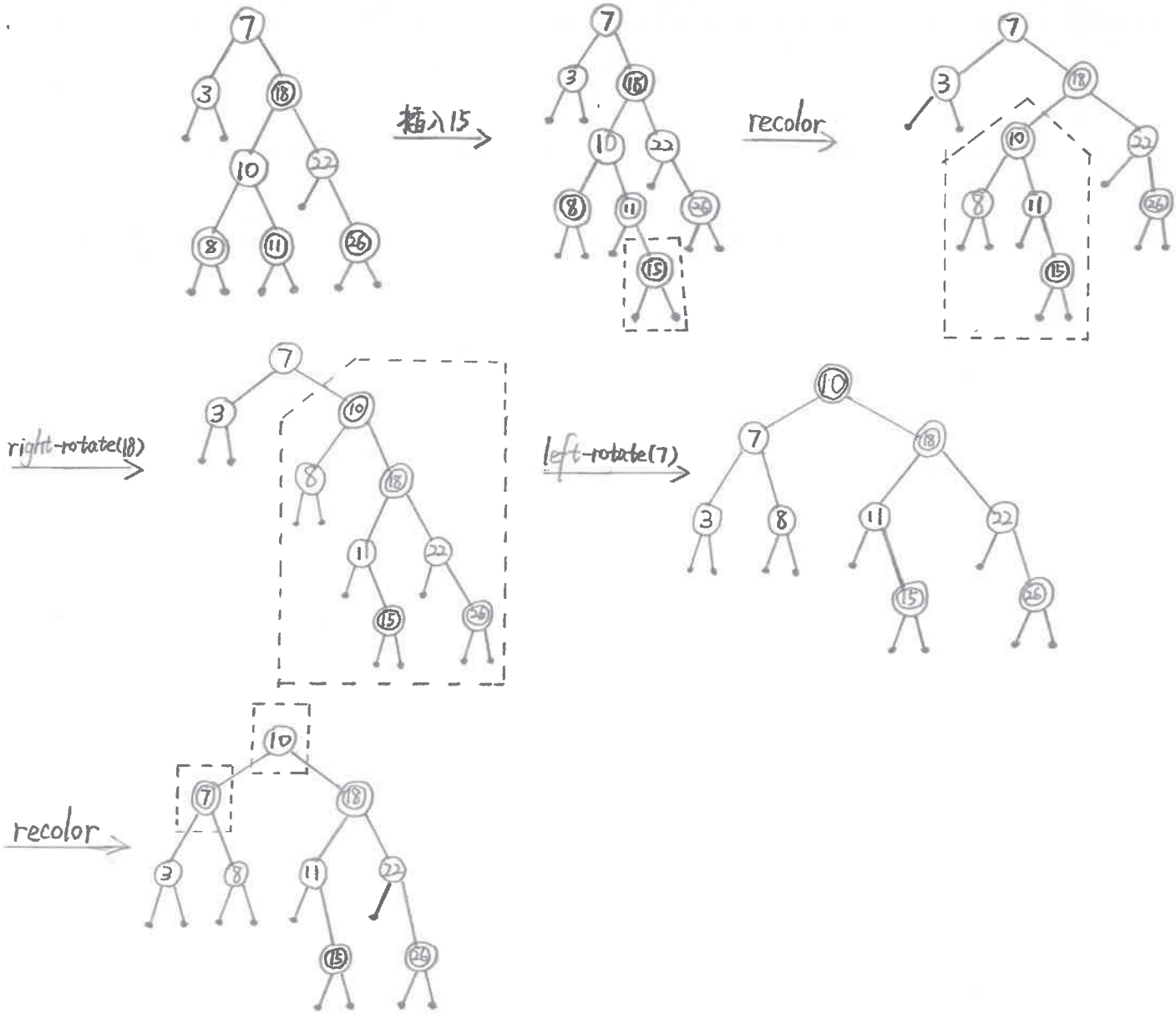
// so we need to move the violation of property 3 up the tree

// via recoloring or rotation until we can fix violation

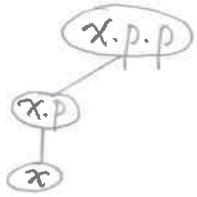
// 先看一个例子

// 伪代码附其后

// 目的：将 x 加入动态集中，同时维持并保留其红黑性

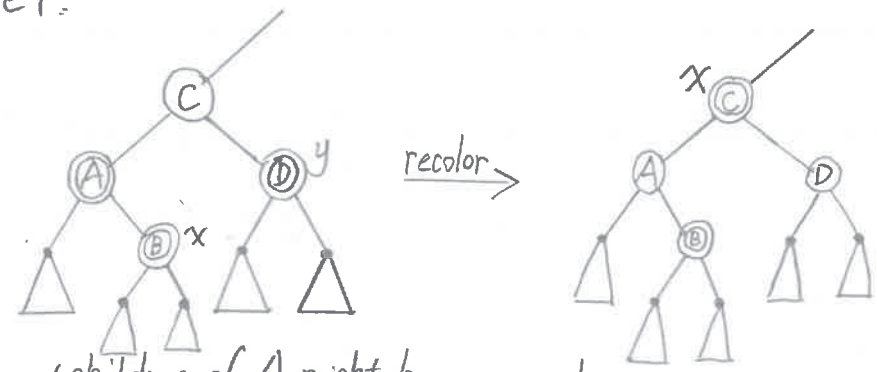



```
RB-Insert(T, x):
    Tree-Insert(T, x)
    x.color ← RED
    while x ≠ T.root and x.color = RED
        do if x.parent = x.parent.parent.left
              then y ← x.parent.parent.right
              if y.color = RED
                    then <case 1>
              else if x = x.parent.right
                    then <case 2>
                    <case 3>
        else
              <similar with if-condition, but reverse the notions of right and left>
    T.rpot.color ← Black
```



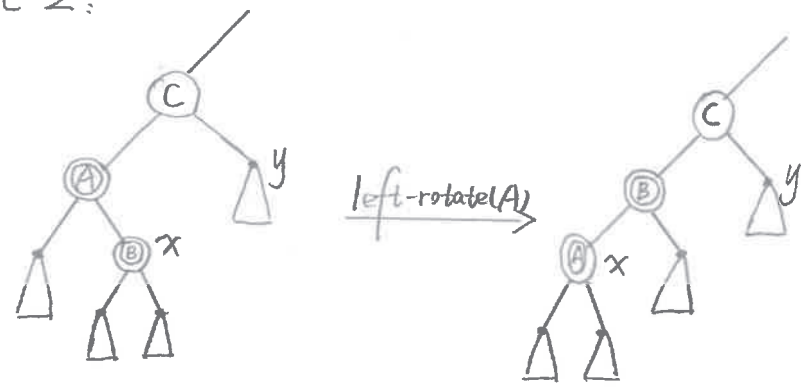
△ : tree with black root, all △ have same black height

case 1.

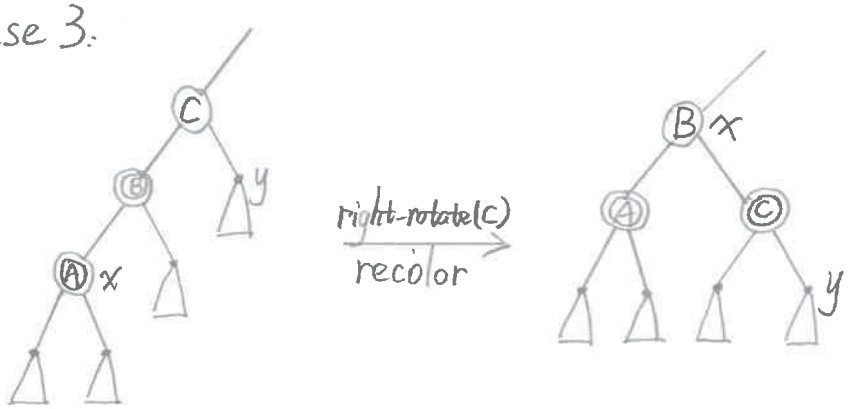


(children of A might be reversed)
recurrence on C

case 2.



case 3.



$T(n) = O(\lg n)$