(Nearly) Balanced BST tree

(nearly) balanced: no leaf is much farther away from the root than any other leaf.

Different balancing schemes allow different definitions of "much farther" and different amounts of work to keep them balanced.

Self-balancing binary search tree :

- a binary search tree
- & keep it balanced

Popular self-balancing BSTree

- red-black tree
- AVL tree

no leaf is more than a certain amount farther from the root than any other

(Height-)Balanced tree

Height-balanced tree :

A tree whose subtrees differ in height by no more than one
and the subtrees are height-balanced, too.
An empty tree is height-balanced.

`http://xlinux.nist.gov/dads/`

Height-balanced tree

- AVL tree

BST in Java.util and C++ STL

Java.util

- TreeMap

  a Red-Black tree based implementation

- TreeSet

  implementation based on a TreeMap

C++ STL

- map, multimap

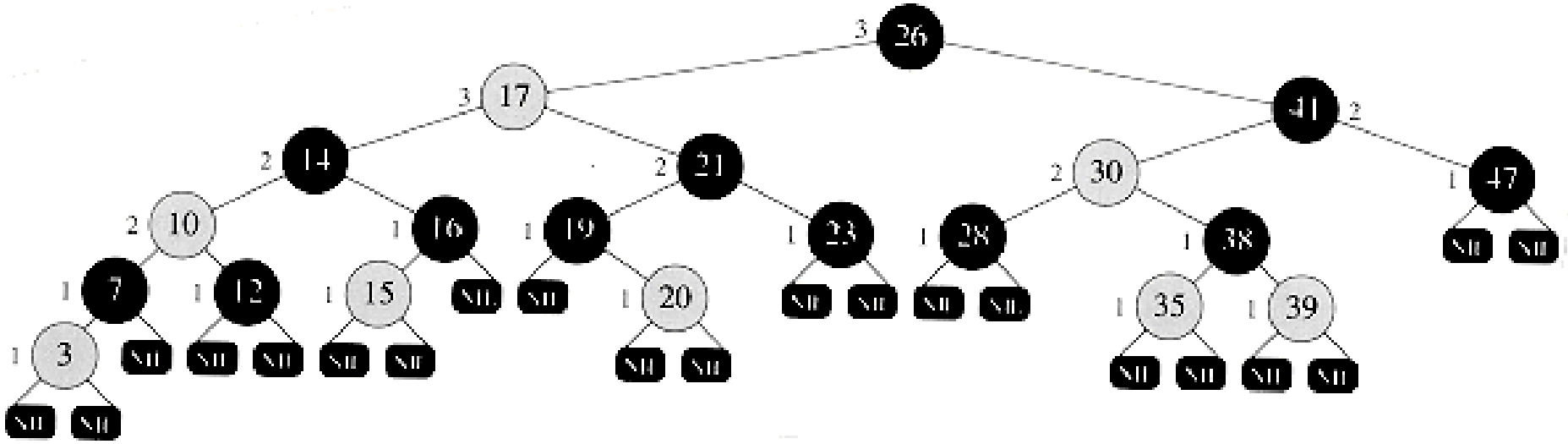
  are typically implemented as binary search trees

  `www.cplusplus.com`

  maps are usually implemented as red-black trees

  `en.cppreference.com/w/cpp/container/map`

Red-black tree



Red-Black tree

A red-black tree is a binary search tree which satisfies:

1. Every node is either red or black.
2. The root is black.
3. Every leaf NIL is considered black.
4. A red node have two black children.
5. For each node:

   all paths from the node to descendant leaves contain the same number of black nodes.

This rule is sometimes omitted, since the root can always be changed from red to black

Red-Black tree

- one extra information per node:

  its ***color***, which can be either RED or BLACK.

- **black-height** of a node x: bh(*x*)

  the number of black nodes on any path from *x* to a leaf node

- **black-height of a red-black tree**: the black-height of its root.

Red-Black tree

Lemma

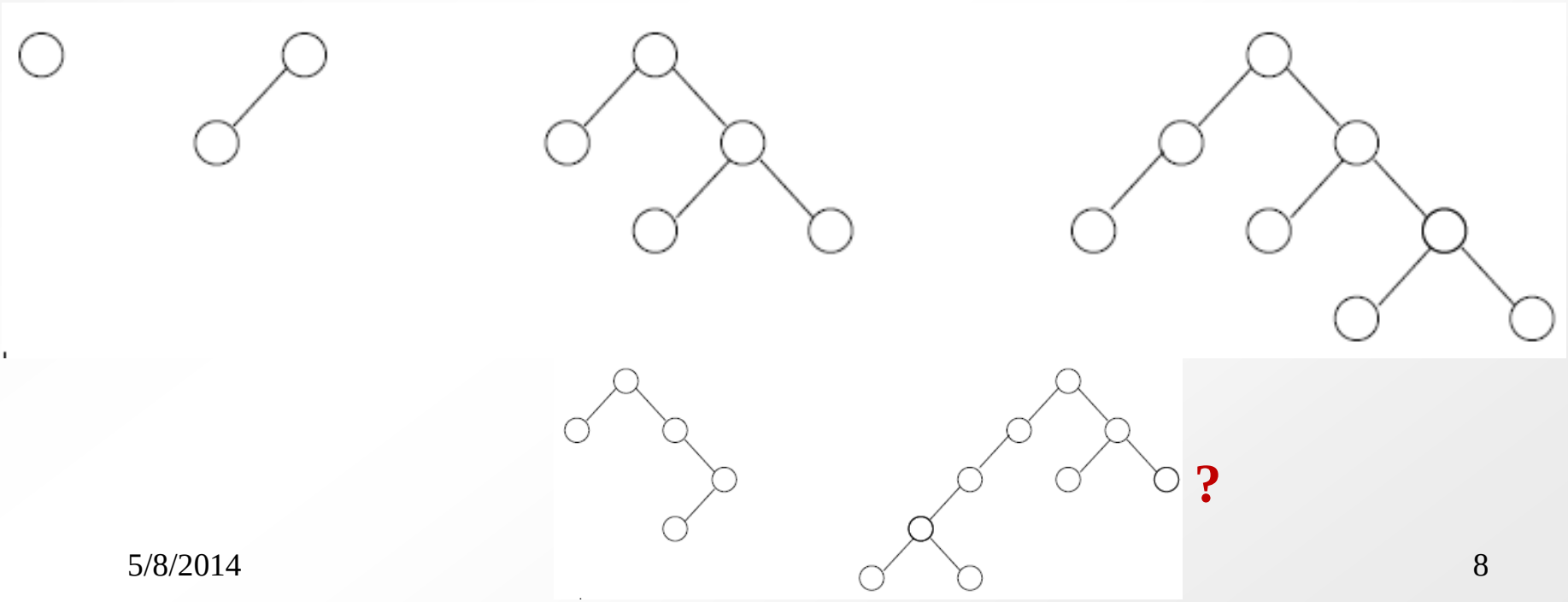
A red-black tree with n internal nodes has height at most $2*\log_2(n + 1)$.

AVL tree

An AVL tree

is a binary search tree which satisfies:

the heights of the two subtrees of any node differ by at most one



AVL tree

Suppose we have ***n*** nodes in an AVL tree of height ***h***.

$$h \sim < 1.44 * \log_2 n$$

Balanced trees

Operations

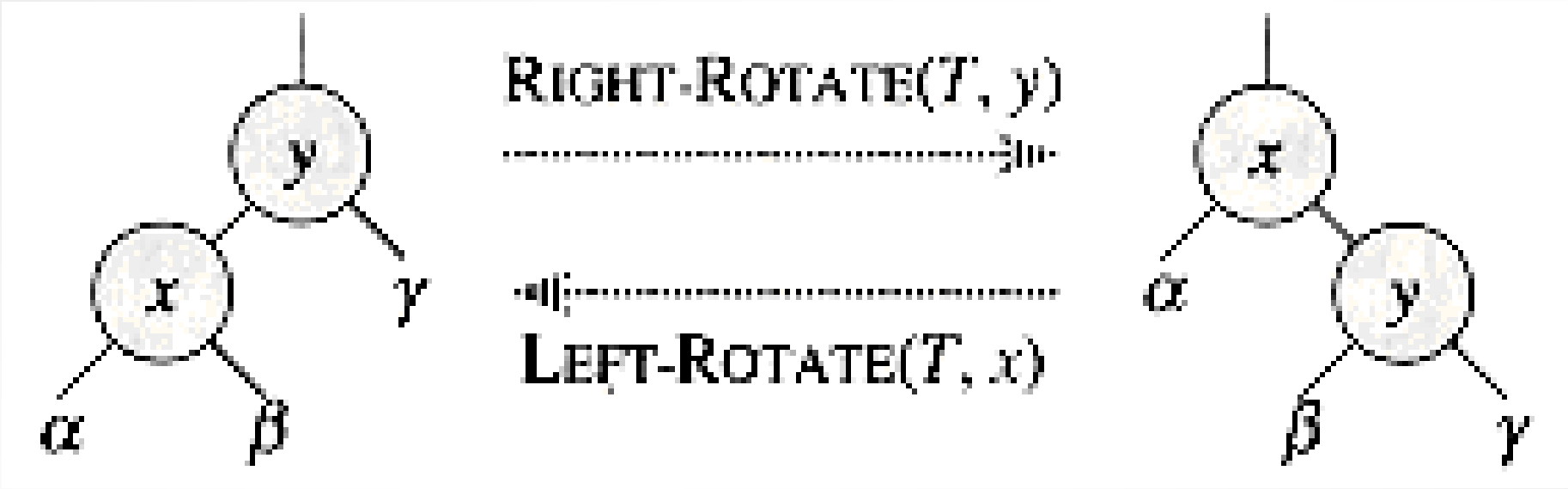
1. Search is *O(log n)* since the trees are always balanced.
2. Insertion and deletions are also *O(log n)*
3. *Balancing* adds *a constant factor* to the speed of insertion/deletion.

- Difficult to program; more space for balance factor.
- Asymptotically faster but rebalancing costs time.

Remark:

Use left-rotate / right-rotate for rebalance

Rotation



DS

TreeNode:

info: TComparable

left: ^TreeNode

right: ^TreeNode

parent:^TreeNode

end