Binary Search Trees

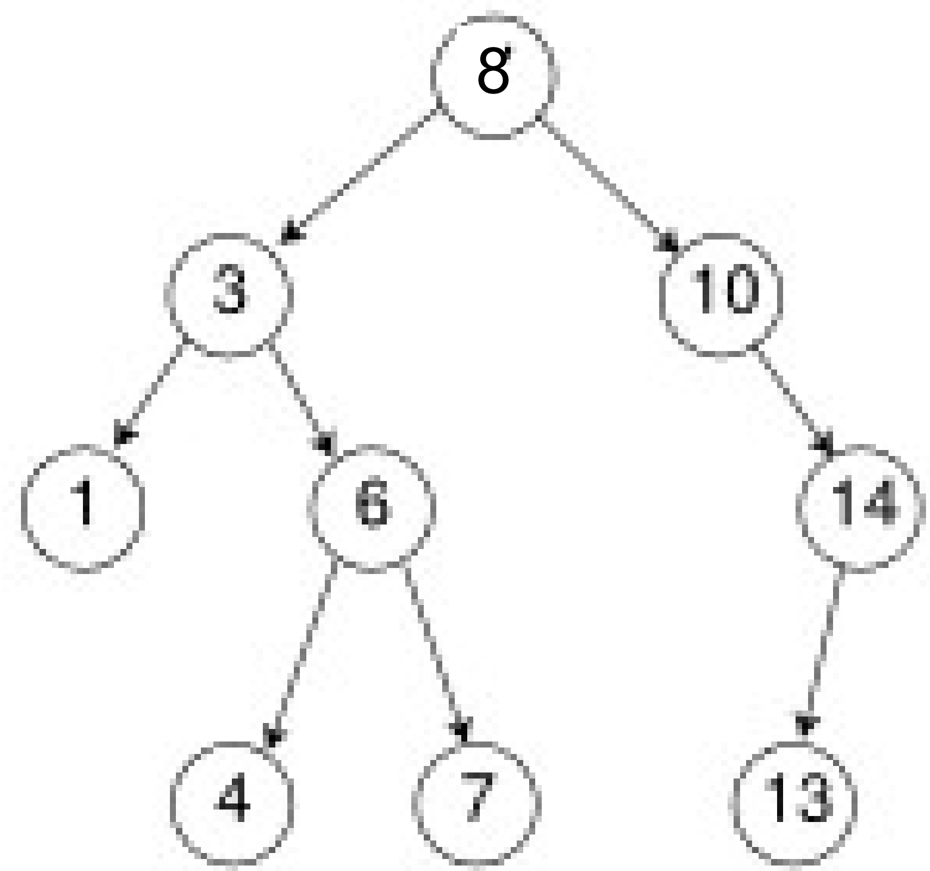
- Binary search trees.
- Balanced binary search trees.
- Application of binary search trees (next lecture).

Search Trees

- A **search tree** S is a data structure that can represent items with keys and permits the following operations:
  - access(k, S): returns item with key k.
  - insert(i, S): inserts item i into S.
  - delete(k, S): deletes item with key k from S.
  - make(): returns new empty search tree.
- Each item has a unique key.
- Items cannot be accessed directly.

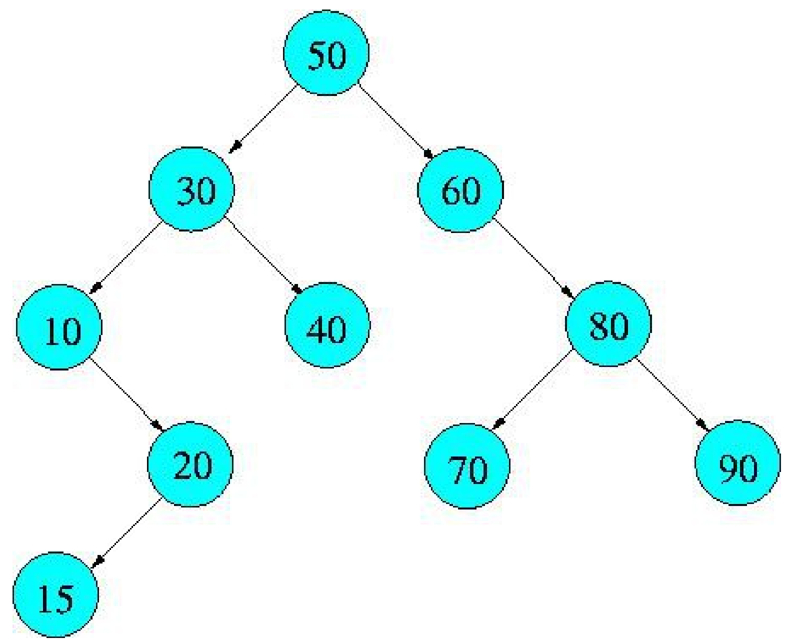
Binary Search Trees

- One node – one item.
- Nodes in the left subtree of any node have keys ≤ the key of the node itself.
- Nodes in the right subtree of any node have keys > the key of the node itself.
- Pointers: $L(x)$, $R(x)$.

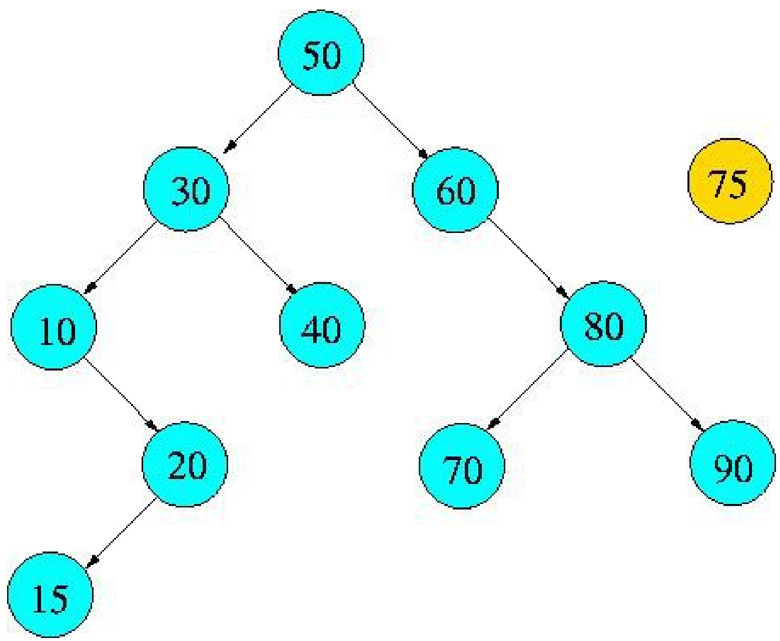


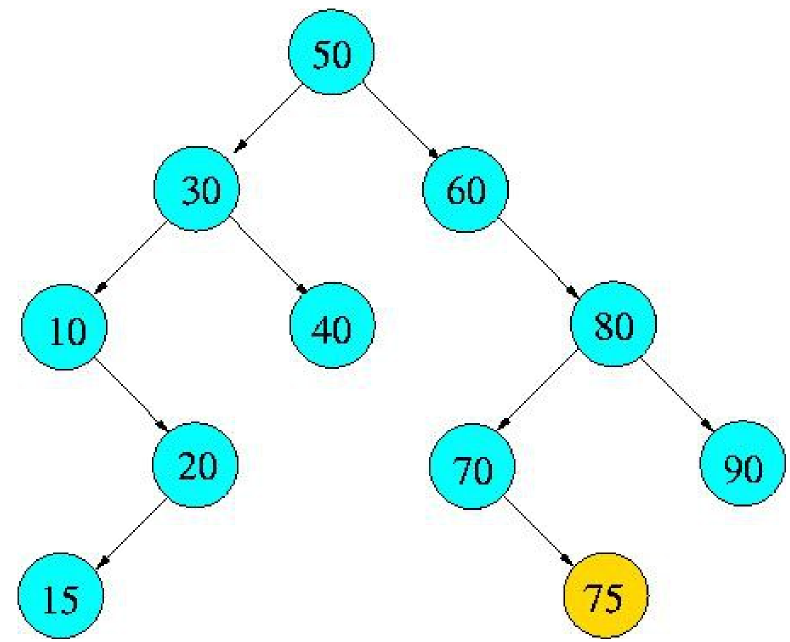
access(k,S)

- x = root of S;
- while key(x) ≠ k
  - if key(x)<k then x=r(x)
    else x=l(x)
- return x
- Accessing an item takes time proportional to its depth. O(n) in the worst case.



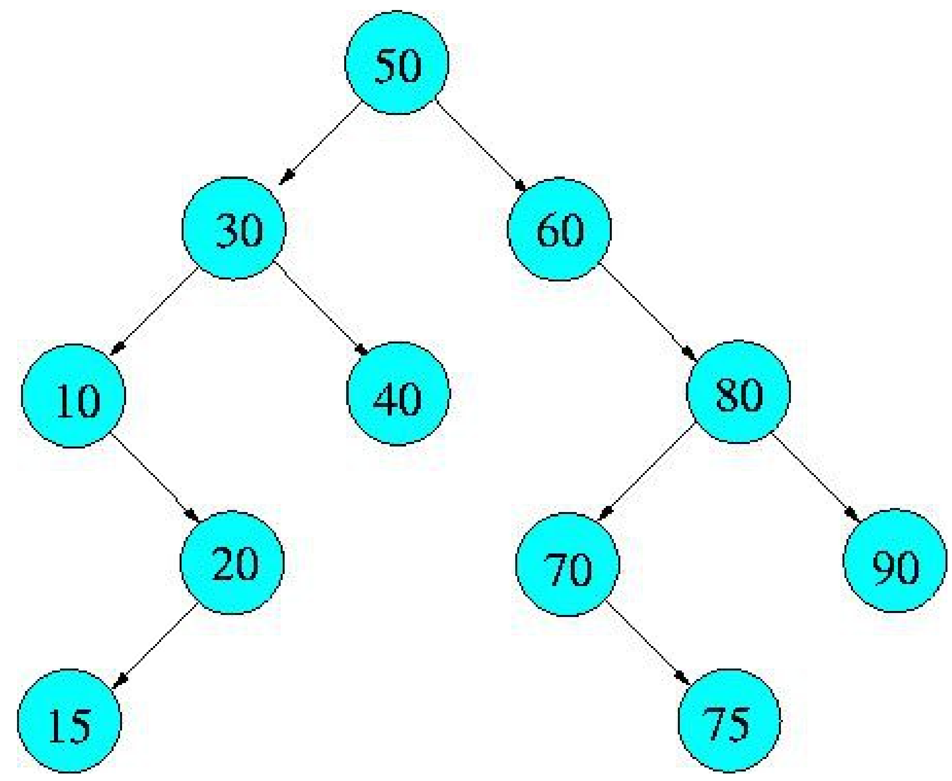
insert(i,S)



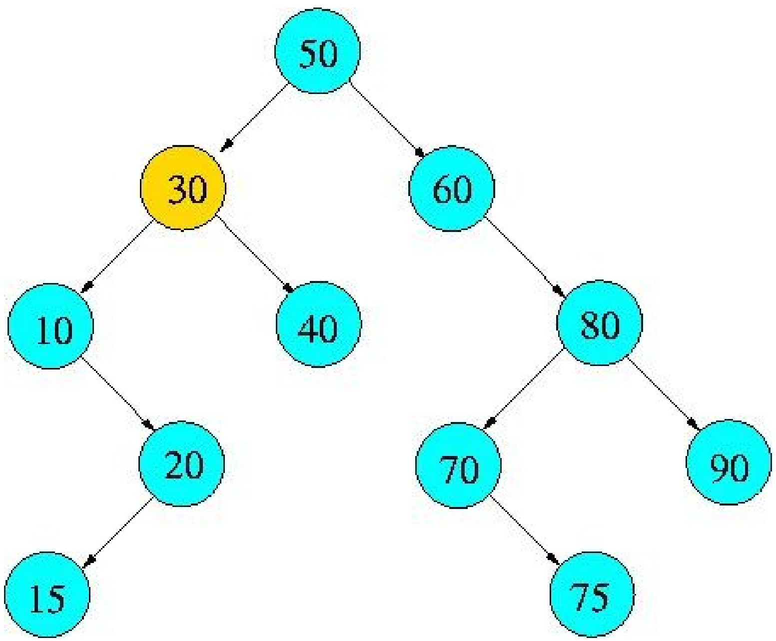


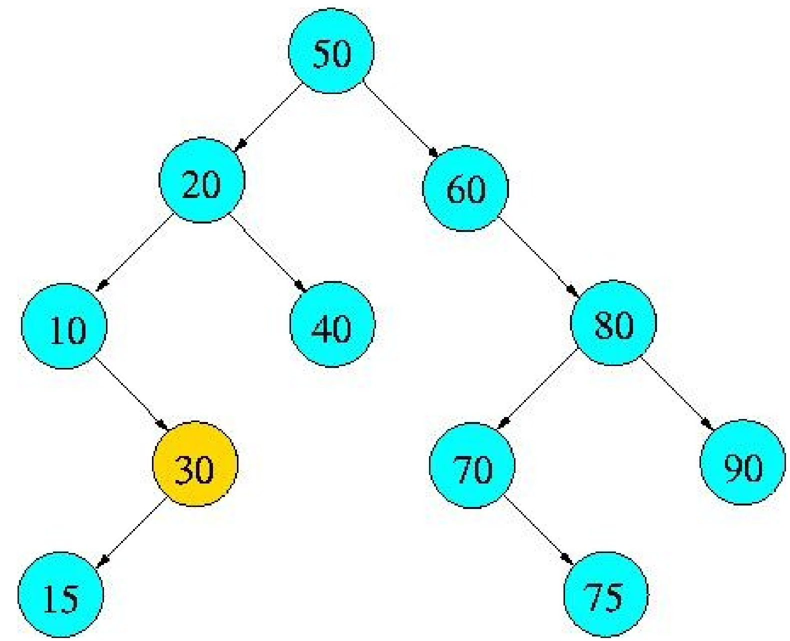
delete(k,S)

- delete(75,S)
- delete(70,S)
- delete(30,S)



delete(30,S)

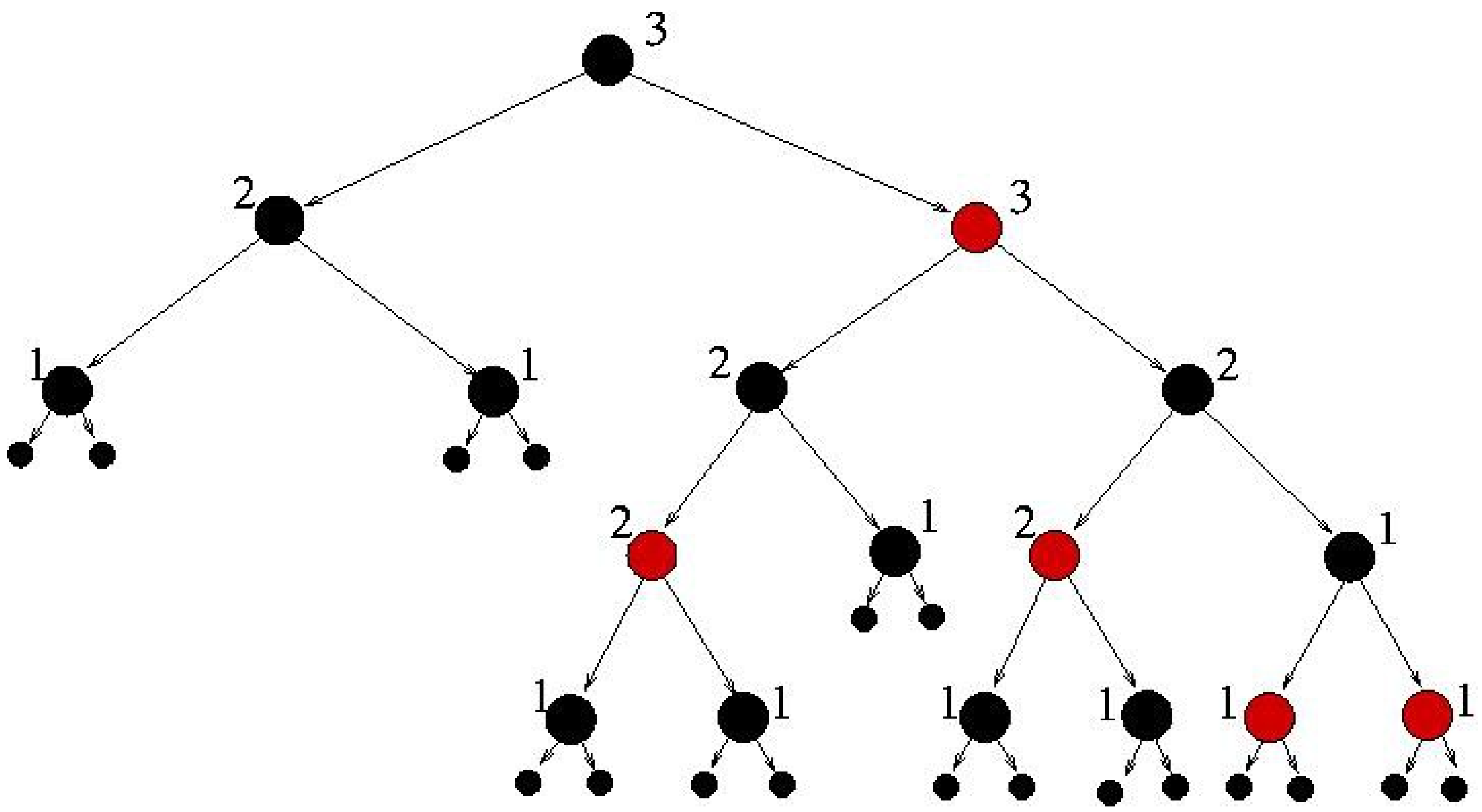




Balanced Binary Search Trees

- How to carry out BST-operations while keeping the height of the tree small?
- Is it possible to reduce the height from $O(n)$ to for example $O(\log n)$?

Red-Black Binary Search Trees

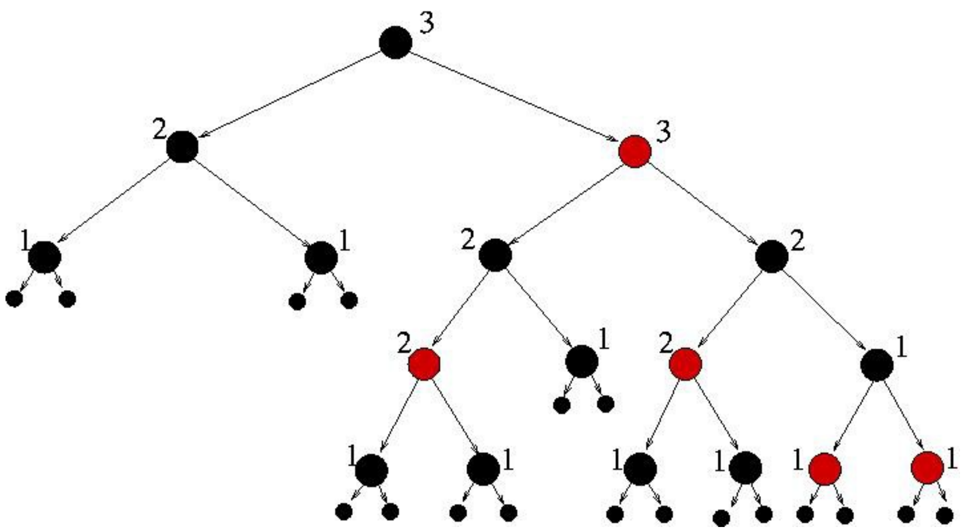


Balanced Binary Search Trees

- Internal nodes.
- External nodes are attached to internal nodes.
- Every node is either black or red.
- Root is black.
- All leaves are black.
- If a node is red, then both its children are black.
- Every path from a given node to any leaf in its subtree has the same number of black nodes.

Black Height

- Black height $bh(x)$ of an internal node x is the number of black nodes on a path from $L(x)$ or $R(x)$ to a leaf of the subtree rooted at x.
- Black height $bh(x)$ of an external node x is 0.



Height of Red-Black Trees

- **Claim**: Red-black trees with n internal nodes have height at most $2\log(n+1)$.
- **Lemma**: Let $s(x)$ denote the number of internal nodes in a subtree rooted at any node x.

$$s(x) \geq 2^{bh(x)} - 1.$$

- **Proof of the claim**: Each red node has 2 black sons. Number of red nodes on a path from root r cannot be more than $h/2$. Hence, the number of black nodes on such a path is at least $h/2$. Therefore $bh(r) \geq h/2$.
- From **Lemma** follows $n = s(r) \geq 2^{bh(r)} - 1 \geq 2^{h/2} - 1$.
- $n+1 \geq 2^{h/2} \leftrightarrow \log(n+1) \geq h/2 \leftrightarrow 2\log(n+1) \geq h$

•**Lemma**: $s(x) \geq 2^{bh(x)} - 1$

- Proof by strong induction on the height h of x.
- Basis: Valid when x has height $h = 0$:
  - x is an external leaf, and $s(x) = 0$.
  - $bh(x) = 0$, and therefore $2^{bh(x)} - 1 = 0$.
- Inequality holds.

•**Lemma**: $s(x) \geq 2^{bh(x)} - 1$

- Assume true for all nodes of height less than k.
- Let x be a node with $h(x) = k$.
- $h(L(x)) \leq k$-1 and $h(R(x)) \leq k$-1. Lemma is therefore valid for $L(x)$ and $R(x)$,

$$s(x) = s(L(x)) + s(R(\mathrm{x})) + 1 \geq$$

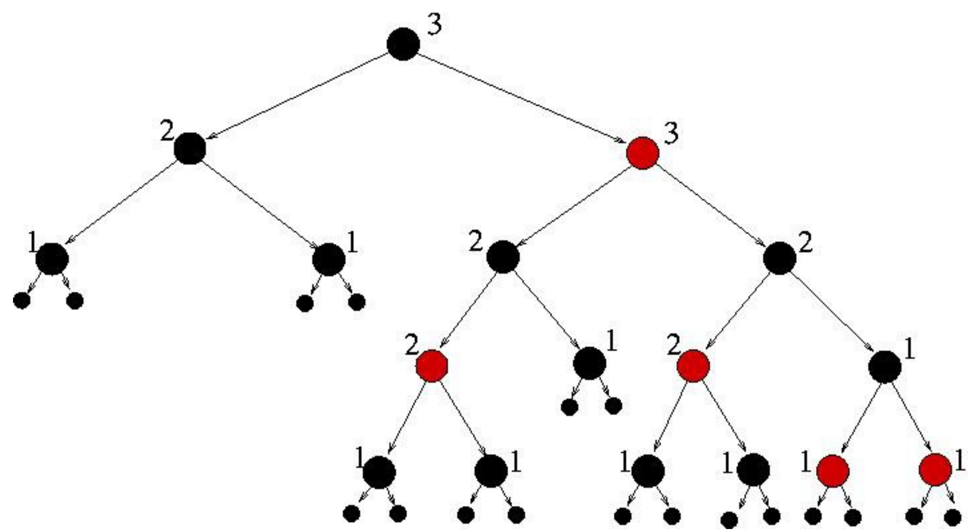
$$2^{bh(L(x))} -1 + 2^{bh(R(x))} -1 + 1 \geq$$

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

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

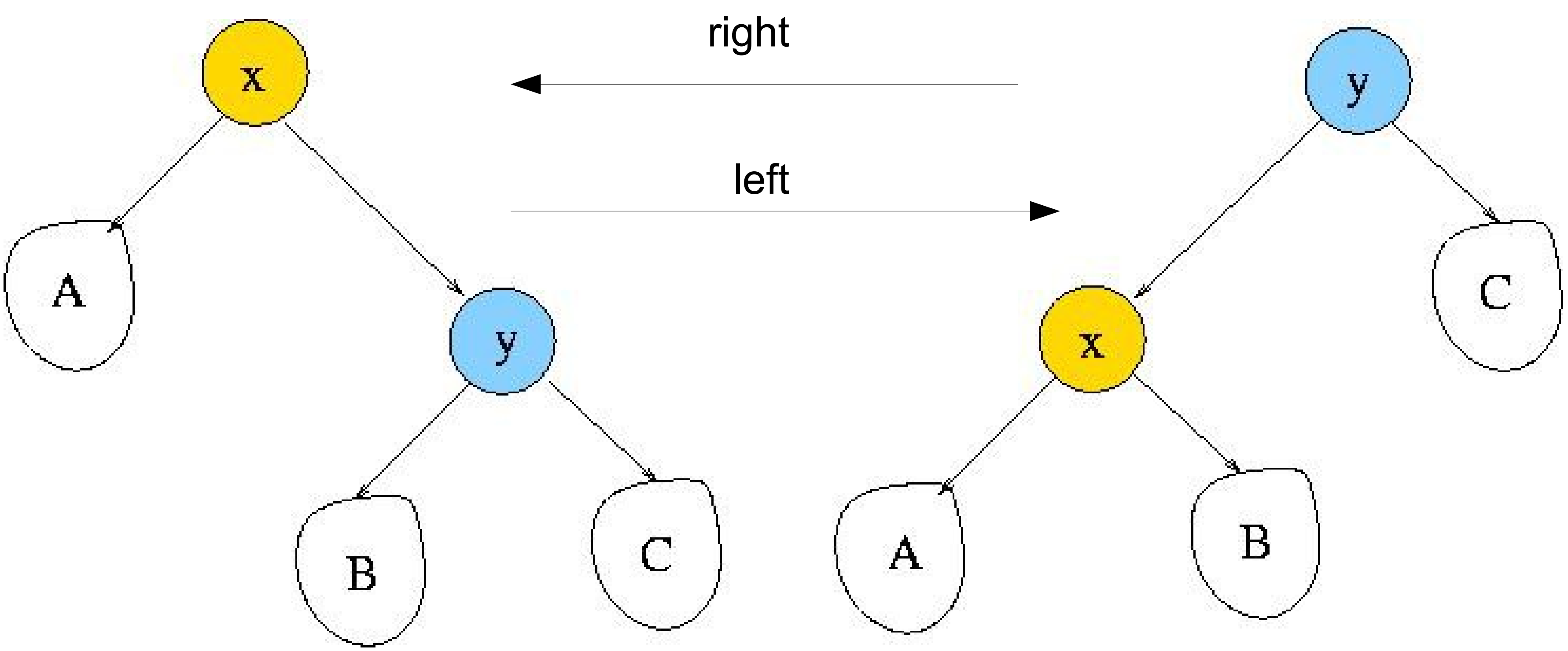
Accessing an Item in RB Trees

- Accessing an item takes $O(\log n)$ time.
- Insertion and deletion can also be done in $O(\log n)$ time but clean-up might be required.

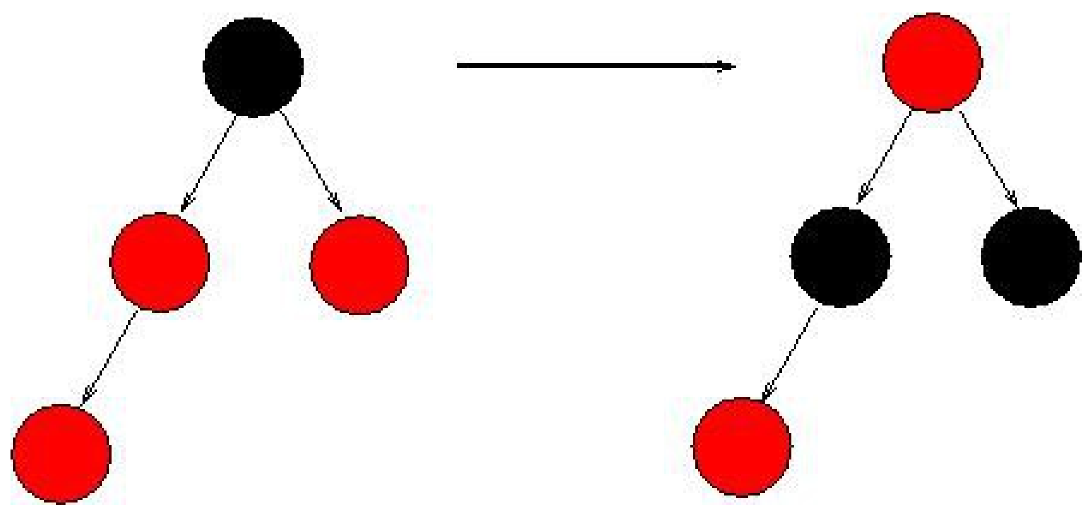


Rebalancing - Rotations

- Left and right rotations require O(1) time.

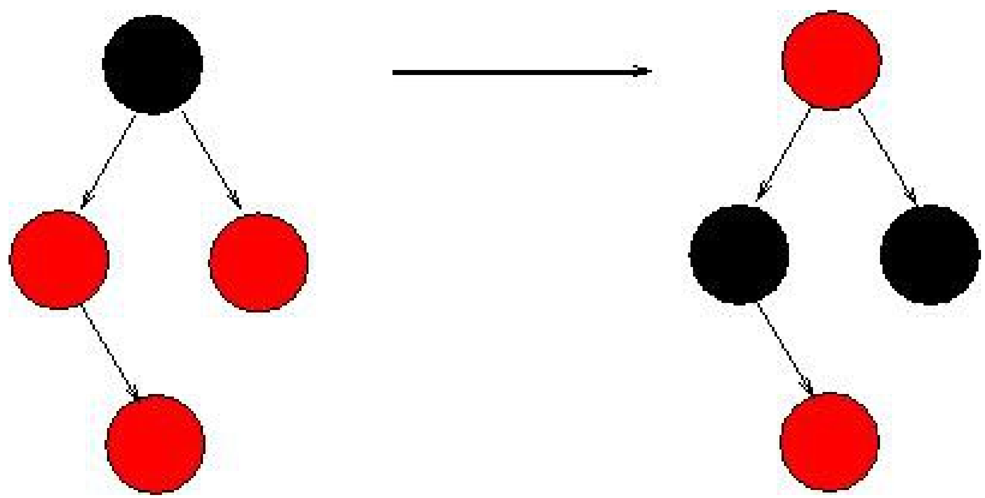


Insertion – Rebalancing – Case 1



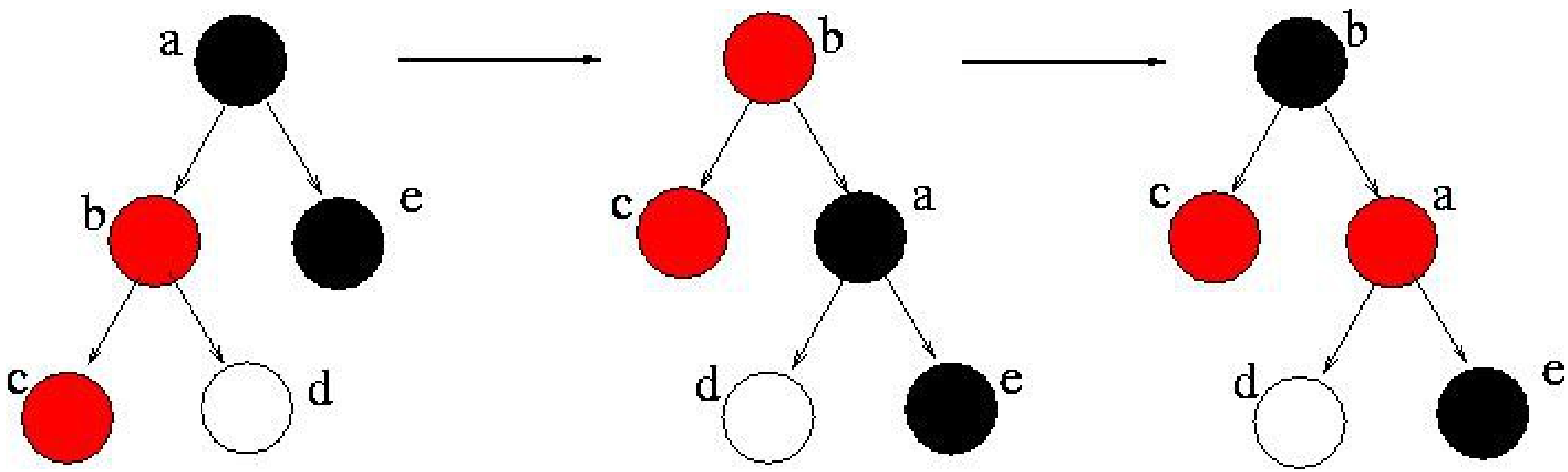
- Problem propagates up the tree.
- If the root is reached, color it black.

Insertion – Rebalancing – Case 2



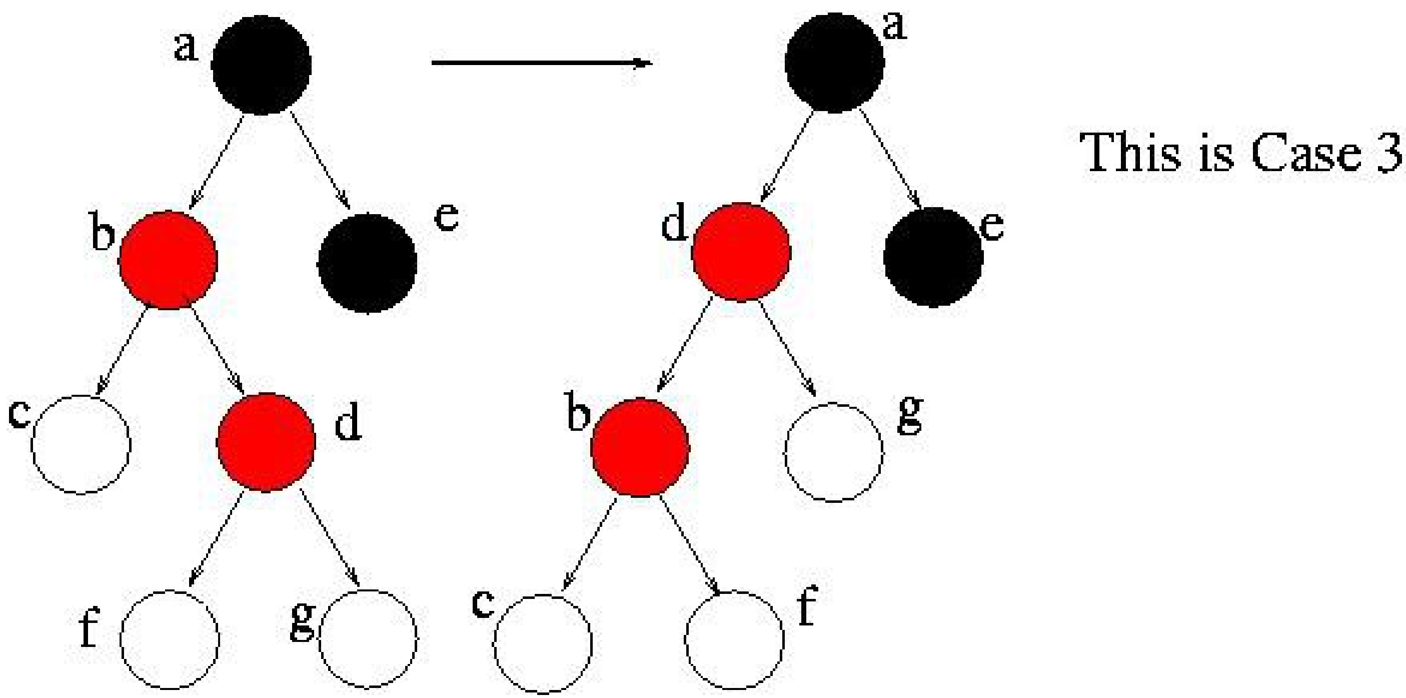
- Problem propagates up the tree.
- If the root is reached, color it black.

Insertion – Rebalancing – Case 3



- Right rotation on a-b
- Recoloring
- Problem solved.

Insertion – Rebalancing – Case 4



- Left rotation on b-d
- Case 3

Deletion - Rebalancing

- If the removed node *b* was red, no problem.
- If the removed node *b* was black while L(*b*) was red, make L(*b*) black.
- If the removed node *b* was black while L(*b*) was black? Make L(*b*) fat and propagate.