Computer Science 112
Data Structures

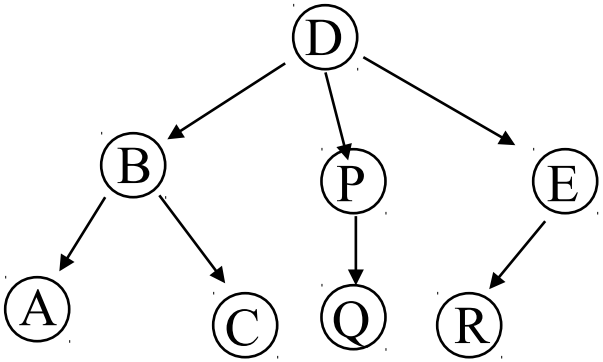
Lecture 12:
AVL Trees

Midterm Exam

- **Was canceled due to snow/ice**
- **Reschedule? To be announced??**

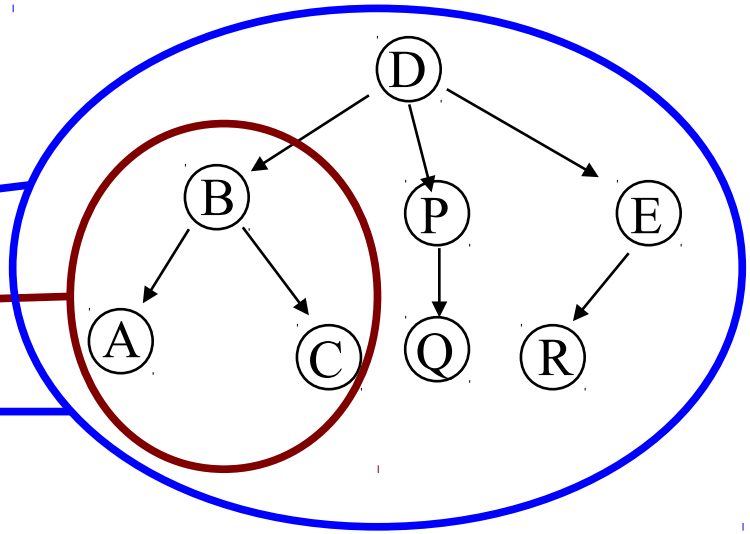
New: Tree Terminology

- **Nodes (vertices) and arcs (edges)**
- **Relationships:**
  - **Parent and Child**
    - **E is a child of D**
    - **D is the parent of E**



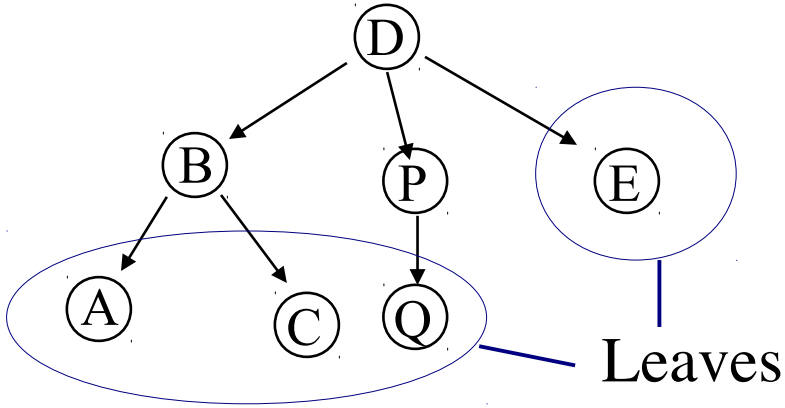
Review: Tree Terminology

- Nodes / vertices and arcs / edges
- Relationships:
  - Parent and Child
    - E is a child of D
    - D is the parent of E
  - Root and Subtree
    - D is the root of this tree
    - This is a subtree of this one.
    - The root of the subtree is B



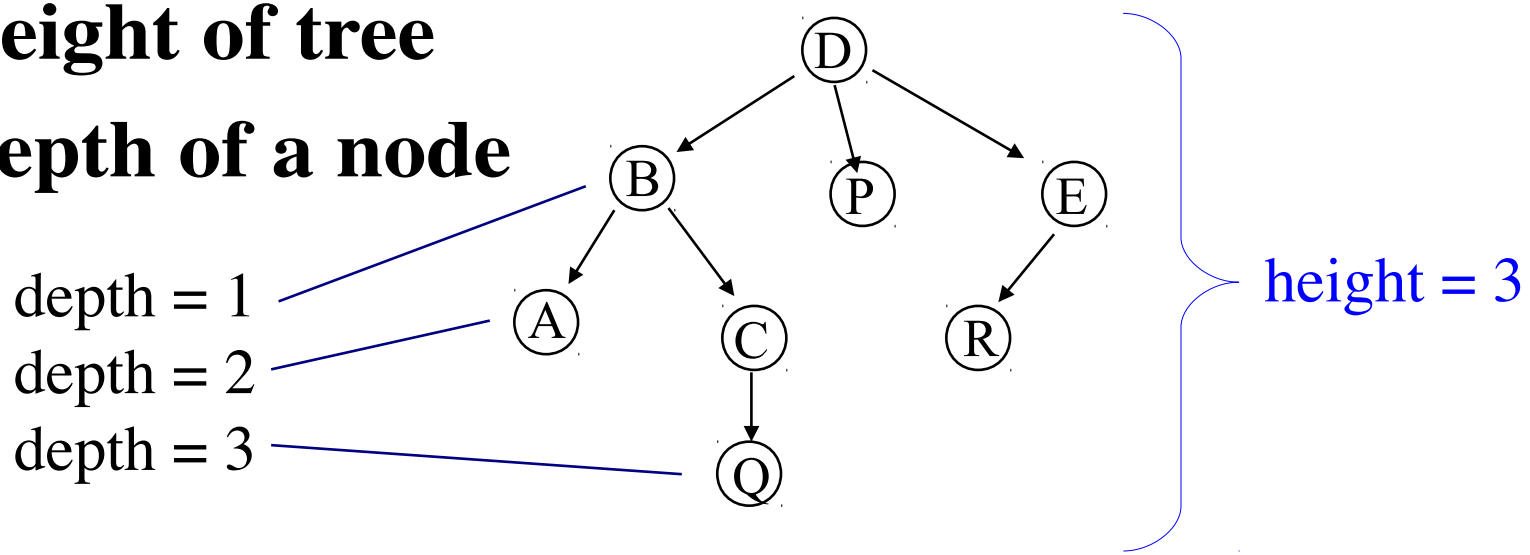
Trees

- The root has no parents
- All nodes except the root have a single parent
- Leaf nodes have no children
- There is exactly one path from root to any node



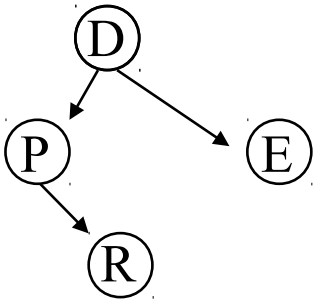
Trees

- **Height of tree**
- **Depth of a node**

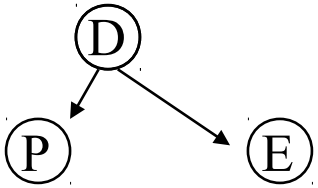


Trees

- **height = 2**



- **height = 1**



- **height = 0**

- **height = -1**

null

Binary tree

- each node has at most 2 subtrees
  - left and right subtree

  root of left subtree is called the left child

  root of right subtree is called the right child

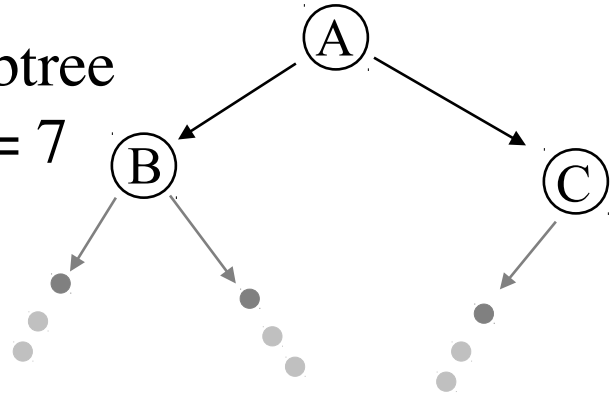
Recursive Data Structures

- **Recursive definition of a binary tree**
  - **empty (i.e. null)**
  - **not empty**
    - **data at the root**
    - **a left subtree, which is a binary tree**
    - **a right subtree, which is a binary tree**

height

What is the height
of the whole tree?

Left subtree
height = 7



Right
subtree
height = 5

Recursive functions
height

```
height(tree):
   if (tree = = null) return  -1
     else  return  1 + max   (  height  (tree.left),
                                  height   (tree.right))
```

Recursive functions

- **Common form of function on a tree is recursive**

f(tree):

if (tree = = null) return ◯

else return □ **(data, f(tree.left), f(tree.right))**

Where ◯ **is a value and**

□ **is a function**

Recursive functions
nodeCount

```
nodeCount(tree):
   if (tree = = null) return 0
     else  return 1 + sum (nodeCount(tree.left),
                           nodeCount(tree.right))
```

Recursive functions
Sum

```
sum(tree):
    if (tree = = null) return  0
    else return  +  (tree.data,
                     sum( tree.left ),
                     sum( tree.right ))
```

Recursive functions
has0

```
has0(tree):
    if (tree = = null) return false
    else return  or  (tree.data = = 0,
                      has0( tree.left ),
                      has0( tree.right ))
```

Binary Search Tree

- data at a node is > any data in left subtree
- data at a node is < any data in right subtree
- Therefore, to print a BST in data order:
  - Print left subtree in data order
  - Print data
  - Print right subtree in data order

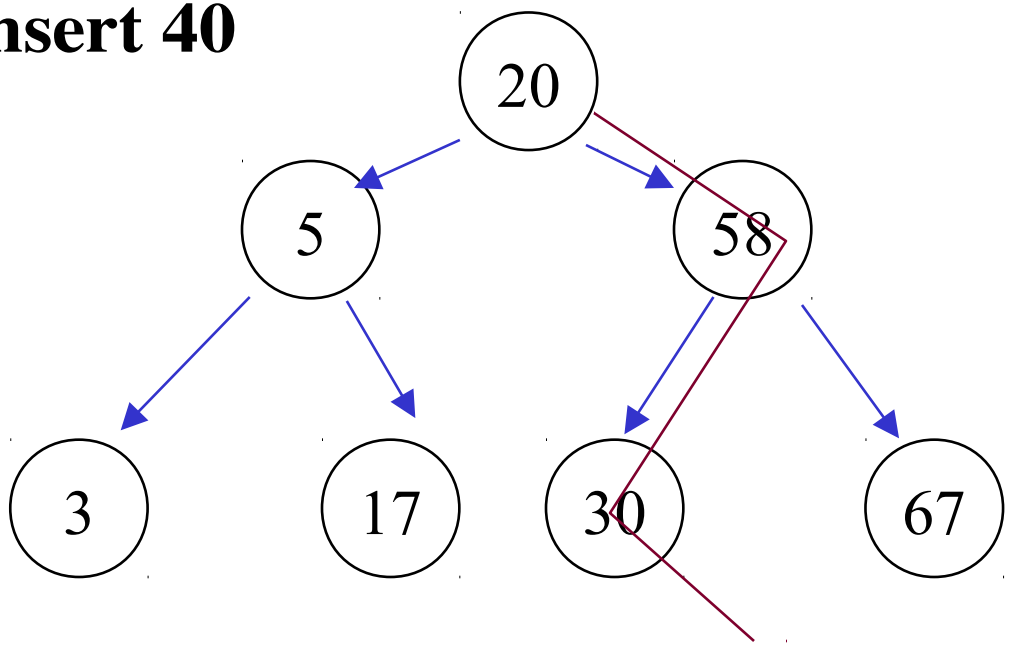
Search

- **Searching a BST is easy**
  - **if node = null, search fails**
  - **if node.data equals target, found**
  - **if target < node.data, search on left subtree**
  - **else search on right subtree**

- **See BSTL11 > BST.java**

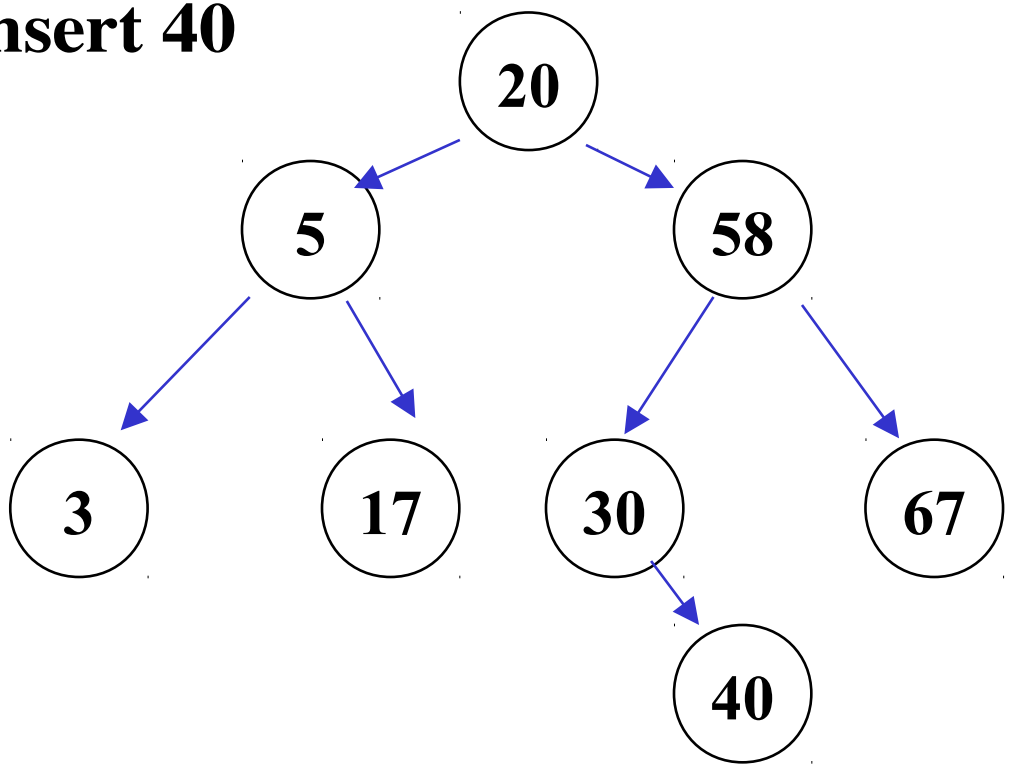
Insert

- **Search, fail, insert where failed**
  - **Insert 40**



Insert

- **Search, fail, insert where failed**
  - **Insert 40**



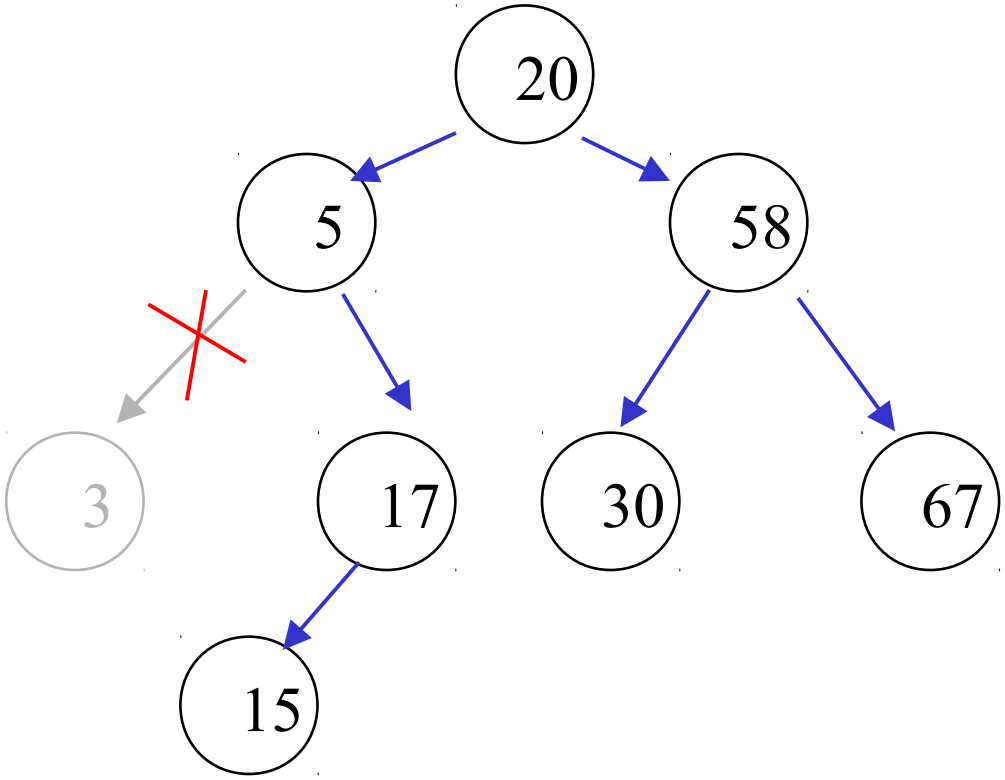
See java > BSTL11 > BST.java

Delete

- **Three cases**
  - **node to delete has no children => delete it**
  - **node to delete has 1 child => link to node becomes link to that child**
  - **node to delete has 2 children**

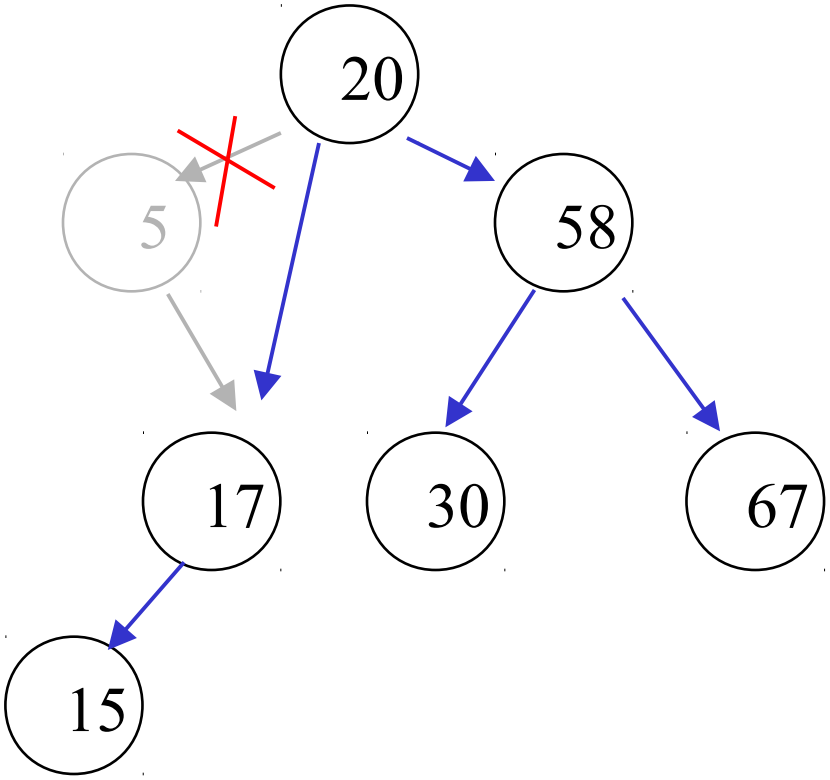
0 children

- **Delete 3**



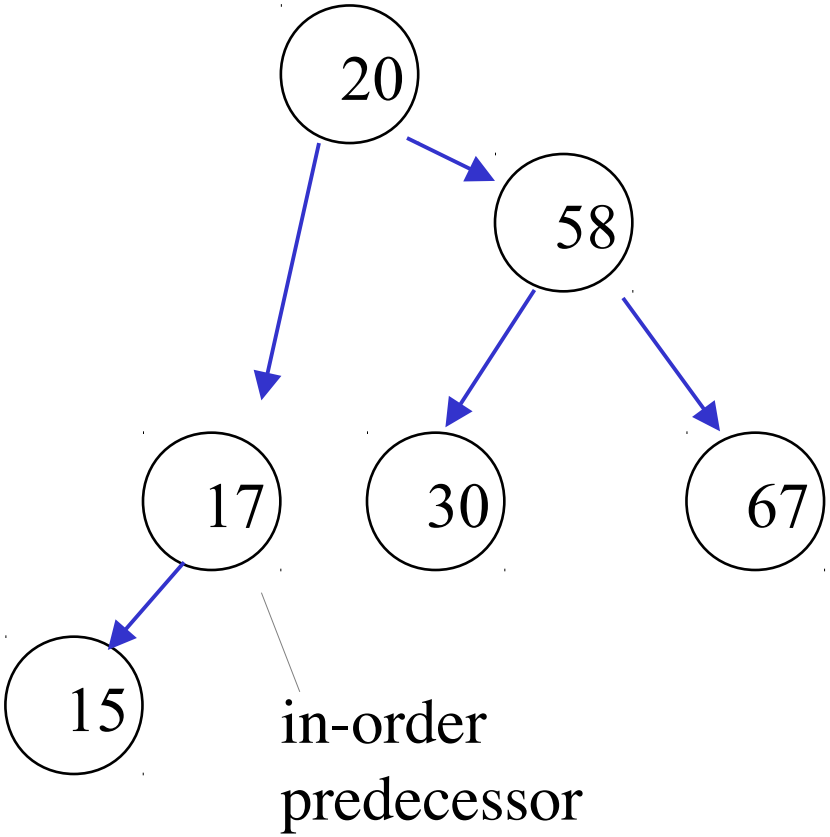
1 child

- **Delete 5**



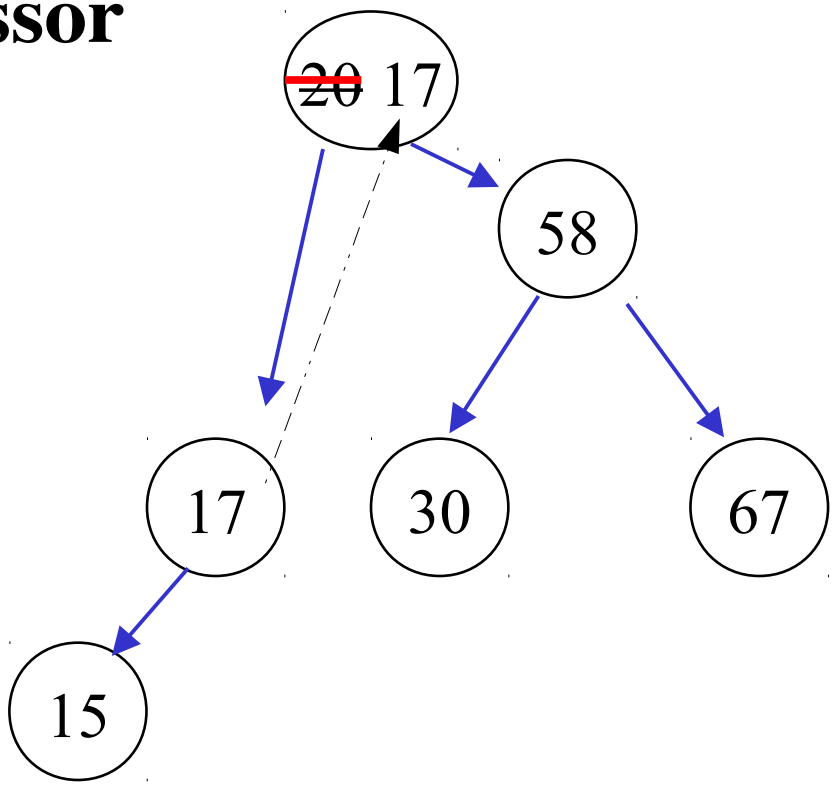
2 children

- **Delete 20**



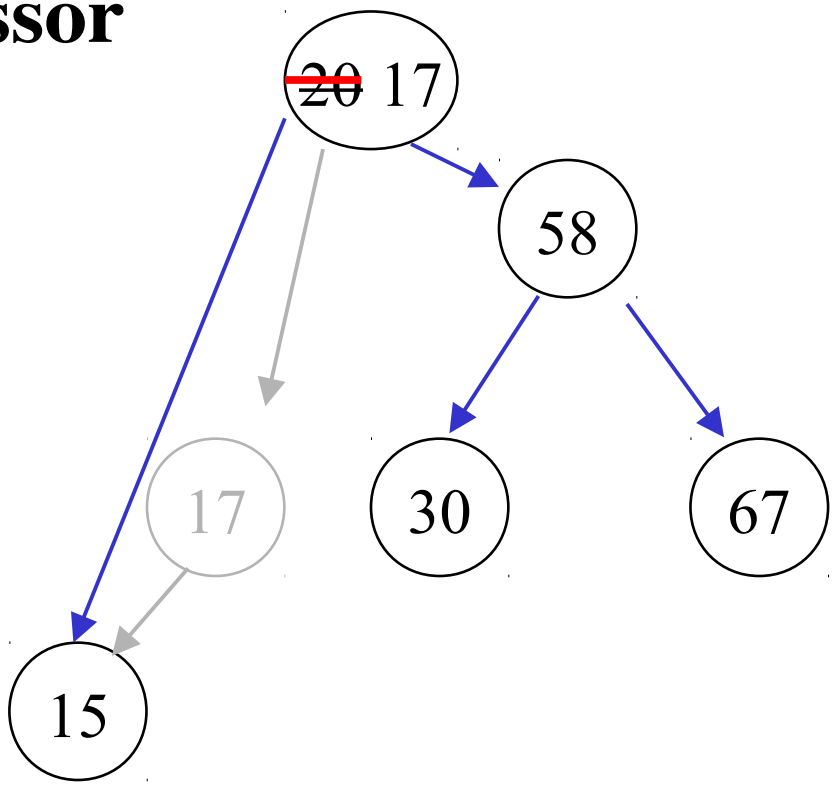
2 children

- **Delete 20: copy in data from in-order predecessor**



2 children

- **Delete 20: then delete in-order predecessor node**



Deleting node with 2 children

- **Replace data at node with data of inorder predecessor**
- **Delete inorder predecessor (which must have either 0 or 1 child)**
- **See BSTL11 > BST.java**

Repeated Keys

- **What do you do if you can have two nodes with the same data?**

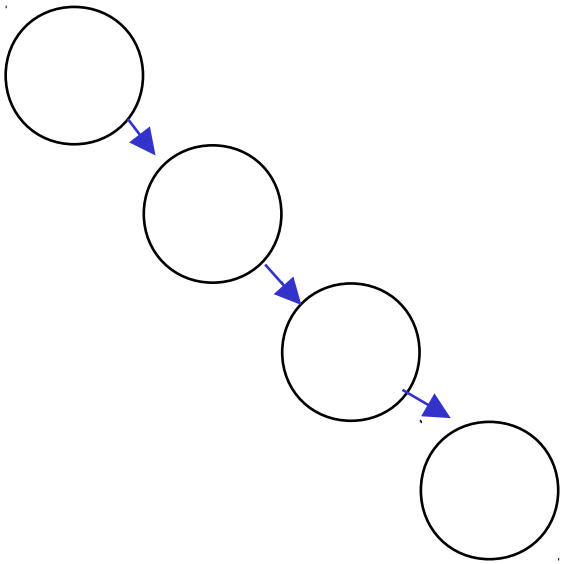
Cost of using BST

Search, insert delete: O(height)

- **What is depth of tree with n nodes?**
  - **best depth is log n**
  - **but worst depth is n**

Binary Search Trees

- **Problem: insertion & deletion can give tree of any shape - even**

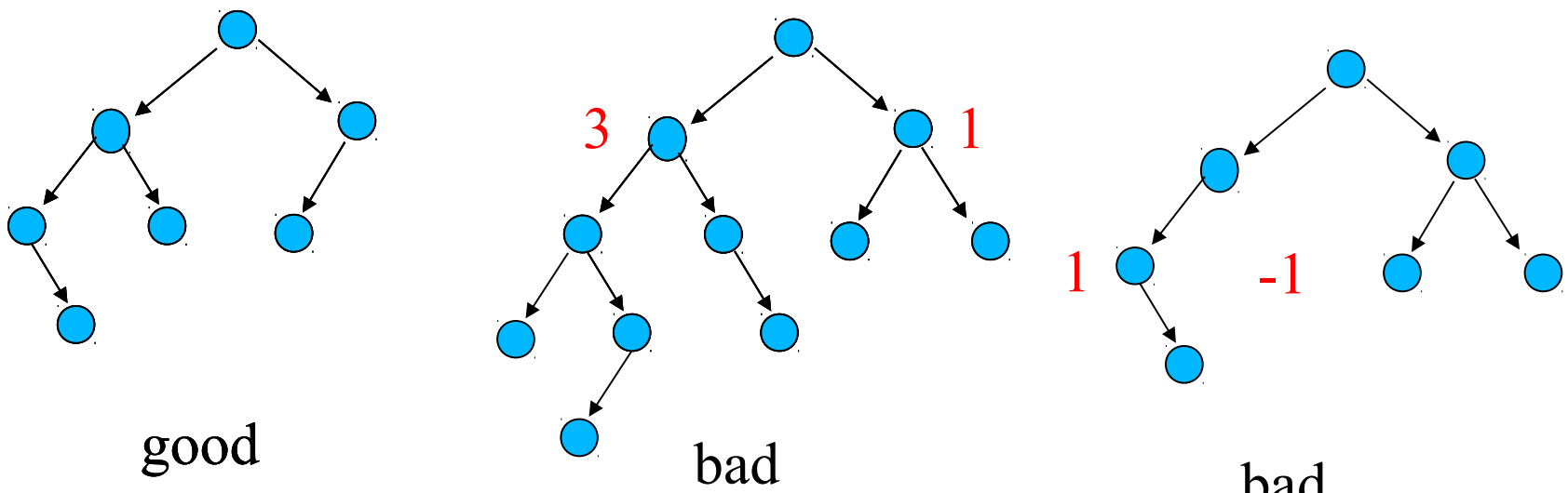


Binary Search Trees

- **Problem: insertion & deletion can give tree of any shape**
- **Solution: AVL trees**

New: AVL Trees

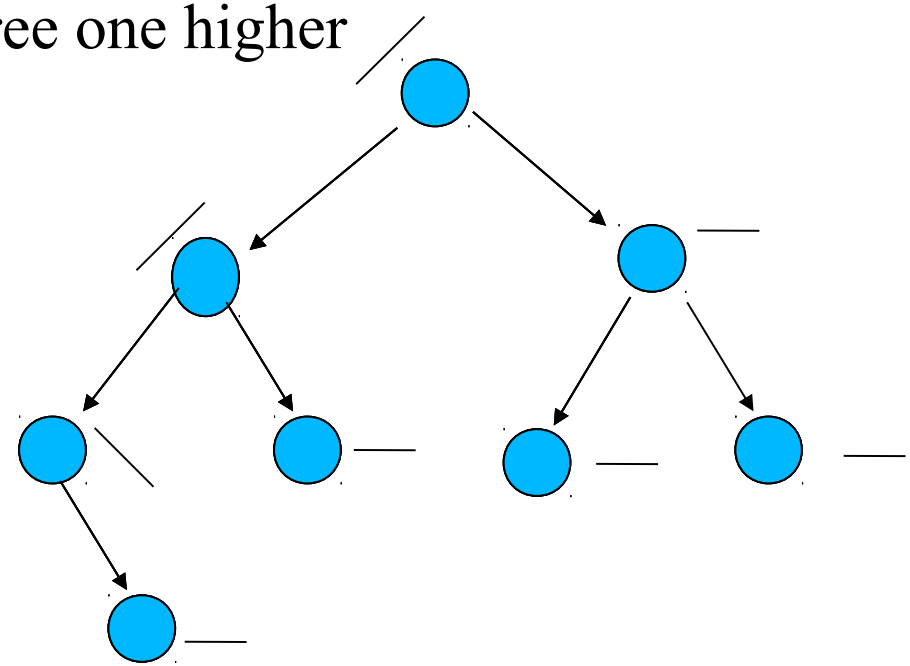
- **Binary Search Tree**
- **Almost balanced**
  - **At every node, subtree heights same +/- 1**



Labeling an AVL Tree

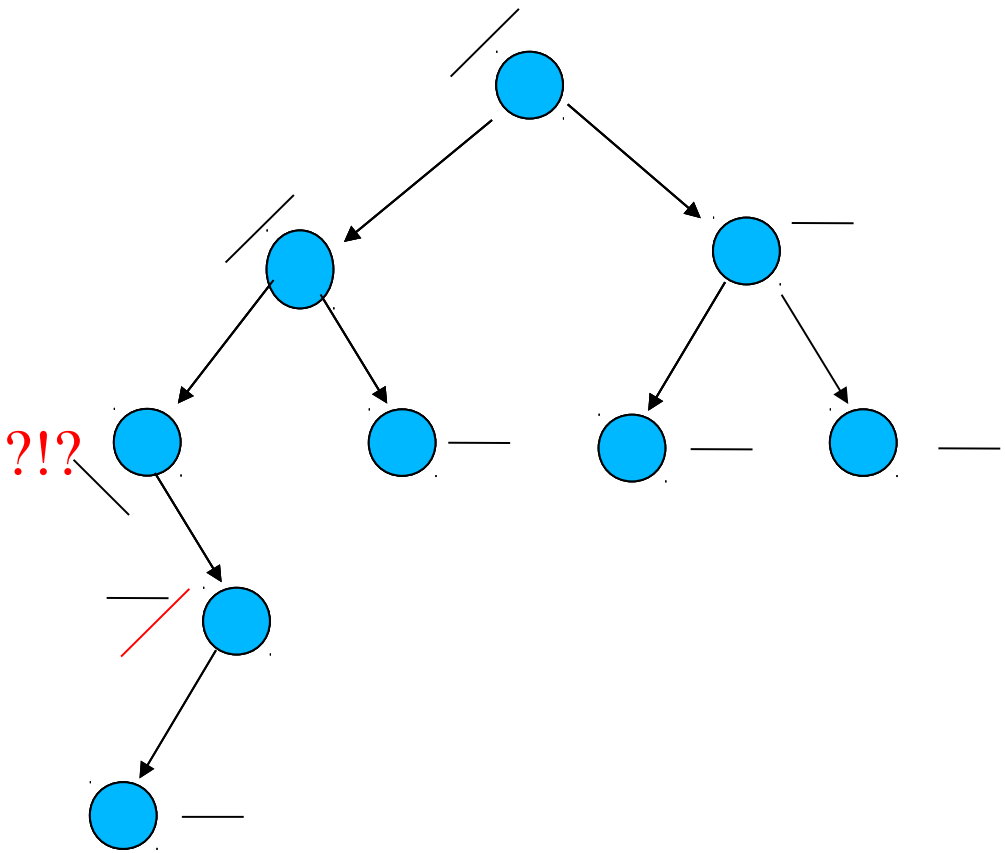
- **Label each node as**

  - left & right subtrees equally high
  \ right subtree one higher
  / left subtree one higher



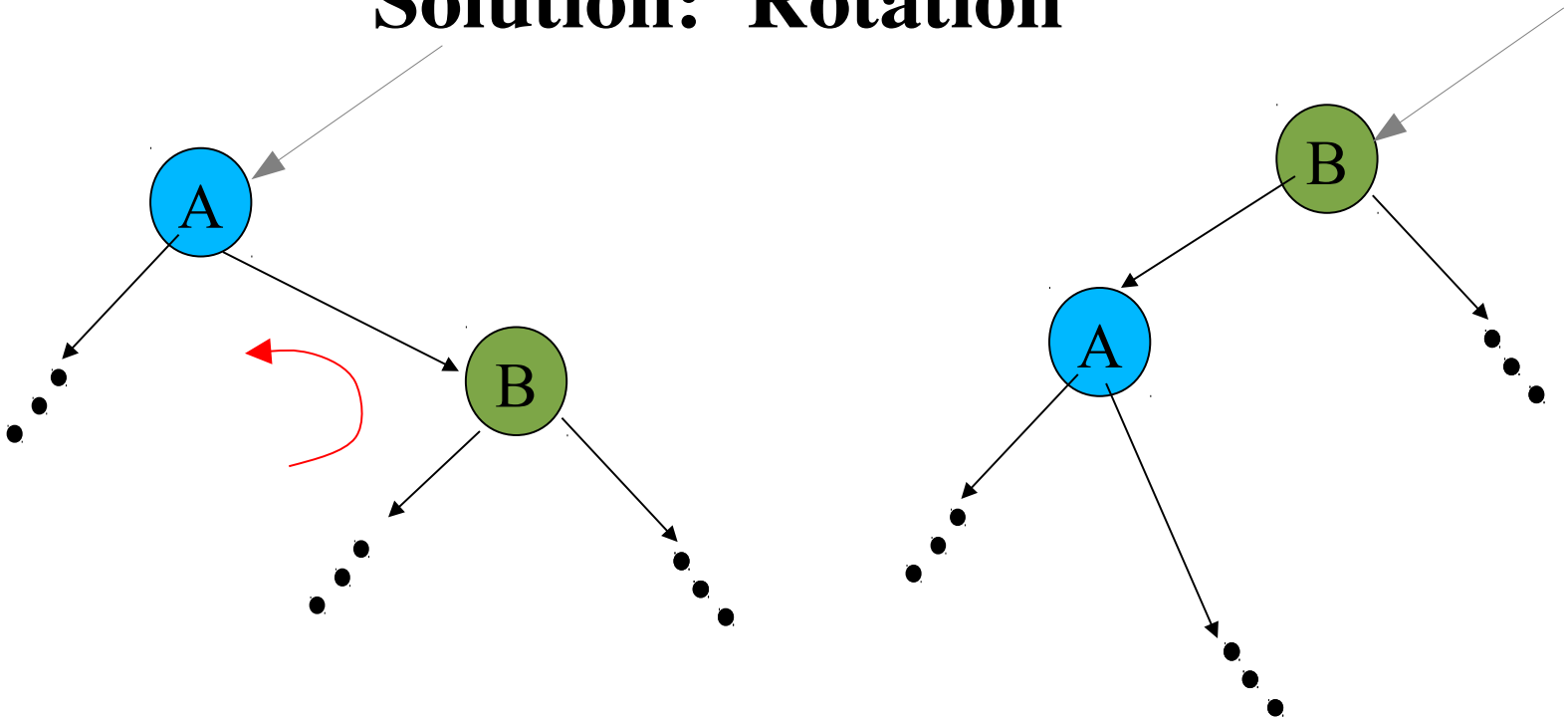
Rebalancing

- **Problem: insert/delete -> not balanced**



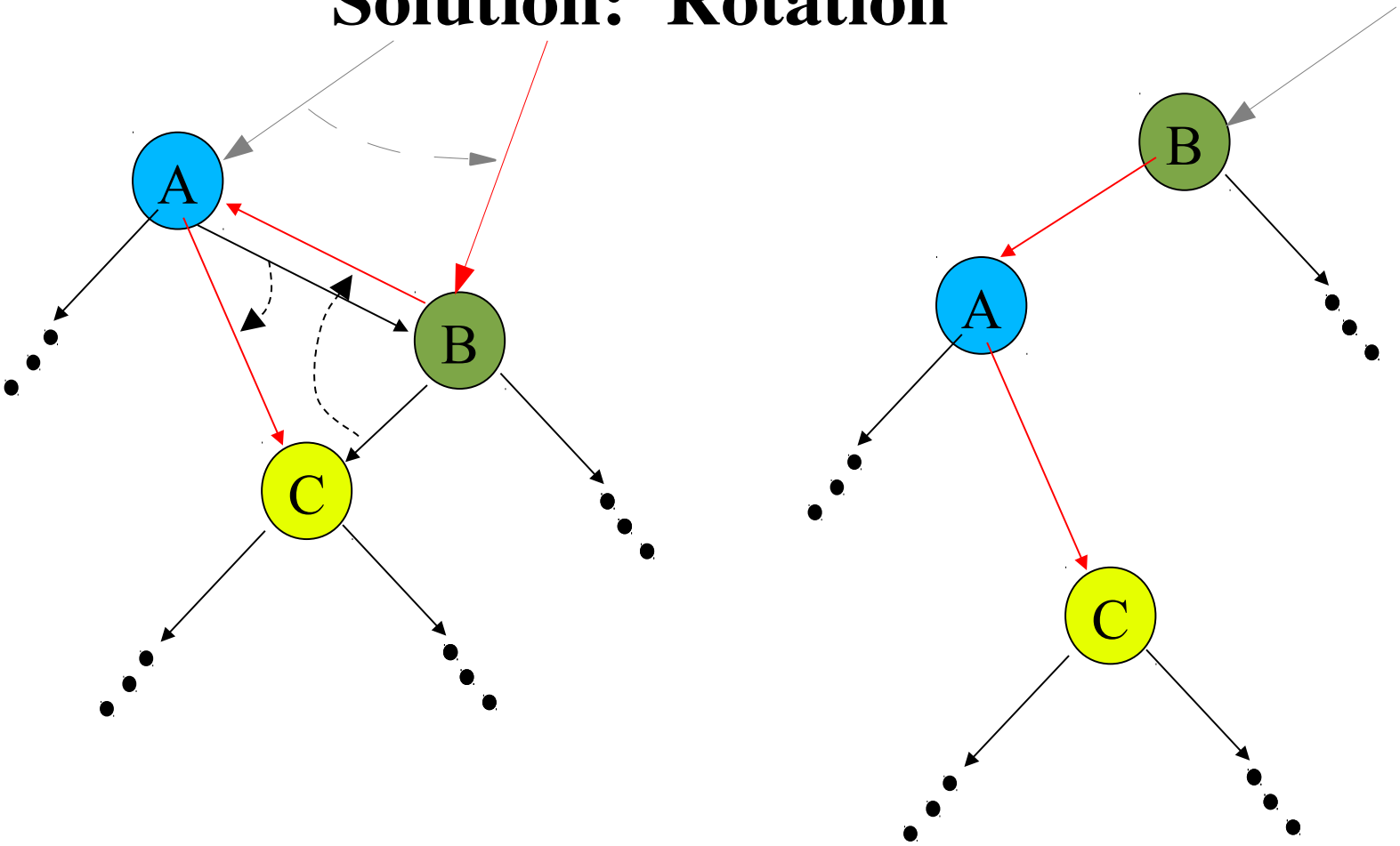
Rebalancing

- **Solution: Rotation**



Rebalancing

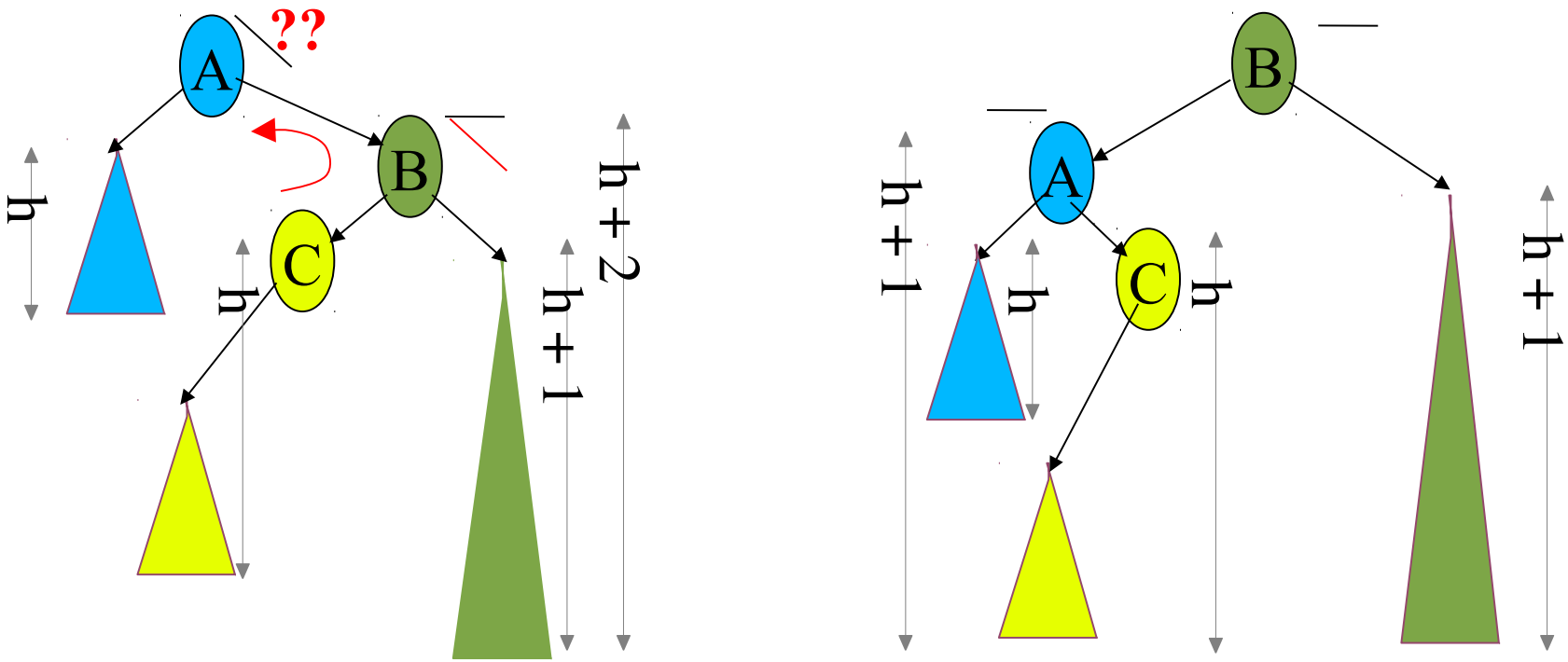
- **Solution: Rotation**



Rebalancing

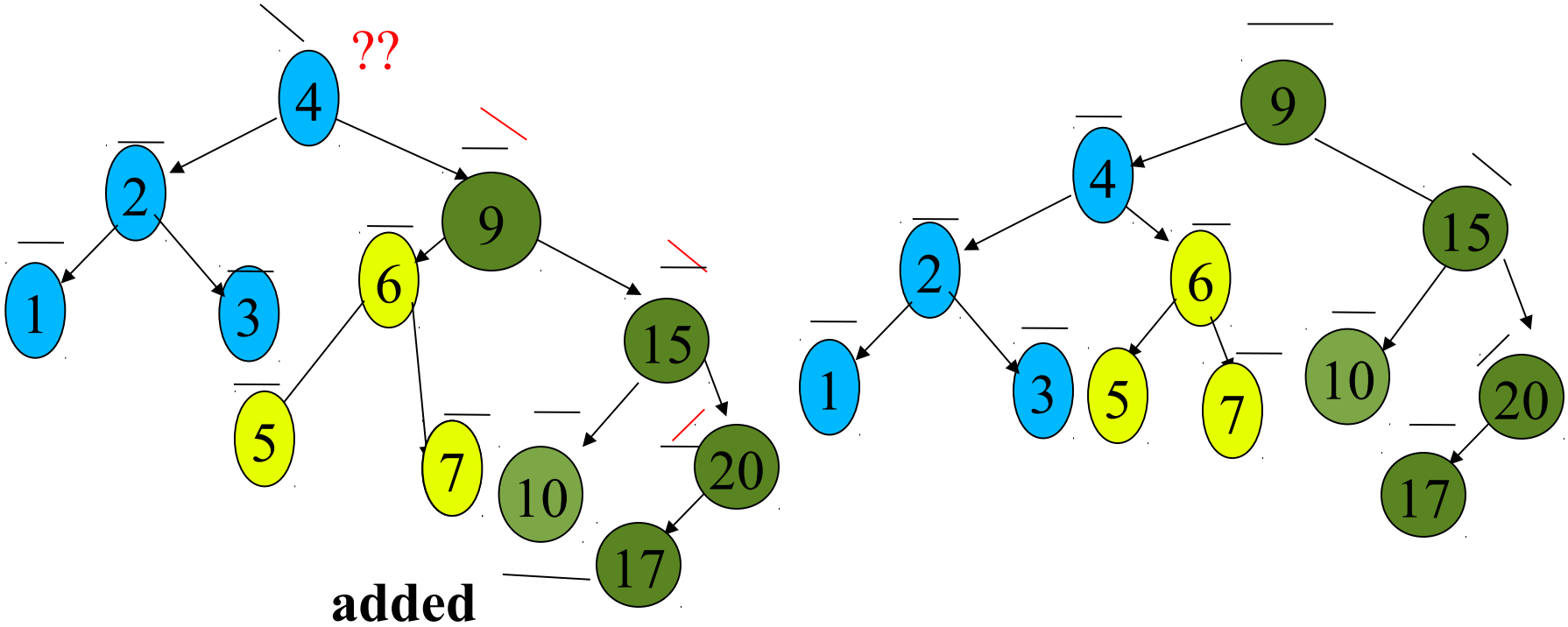
- **Solution: Rotation**

Highside child of A has same label as A



Rebalancing

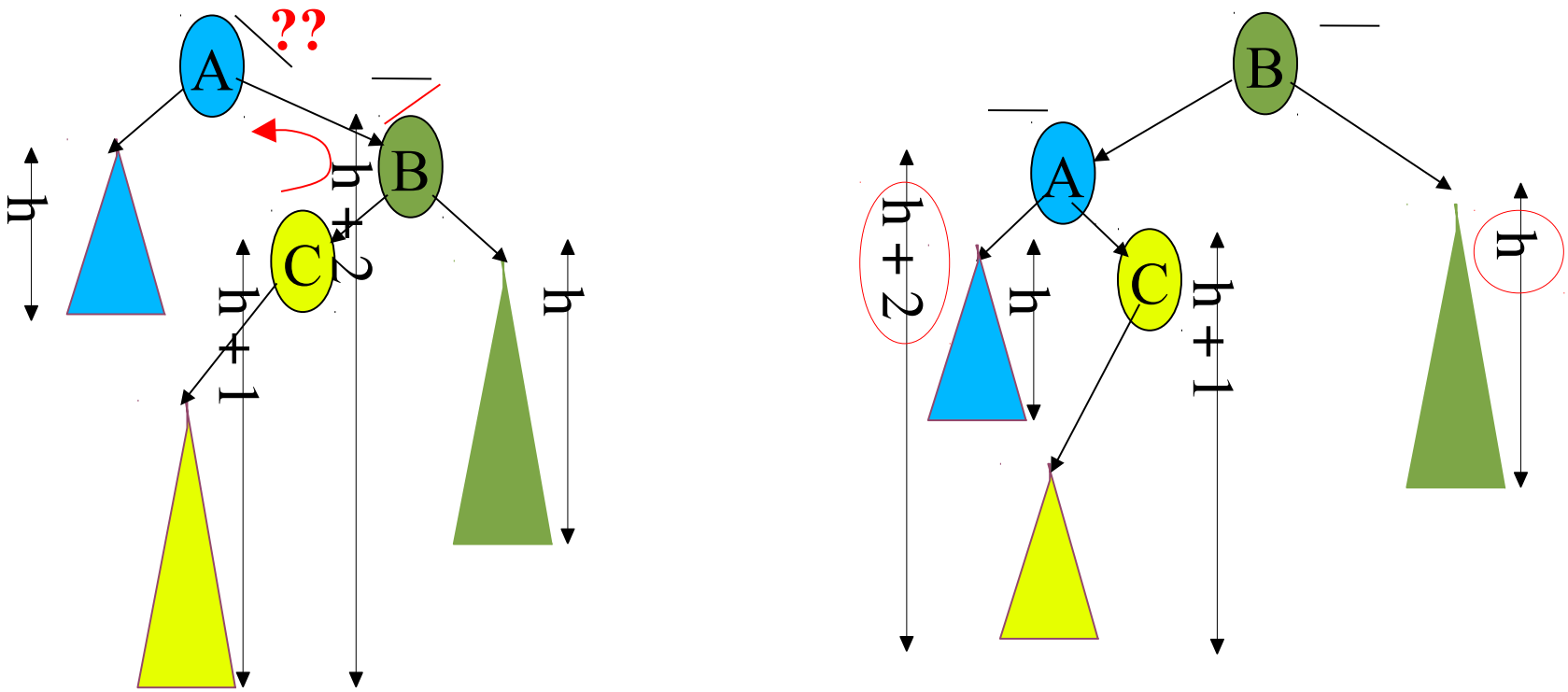
- **Solution: Rotation**
  - **Highside child of A has same label as A**



Rebalancing

- **Solution: Rotation**

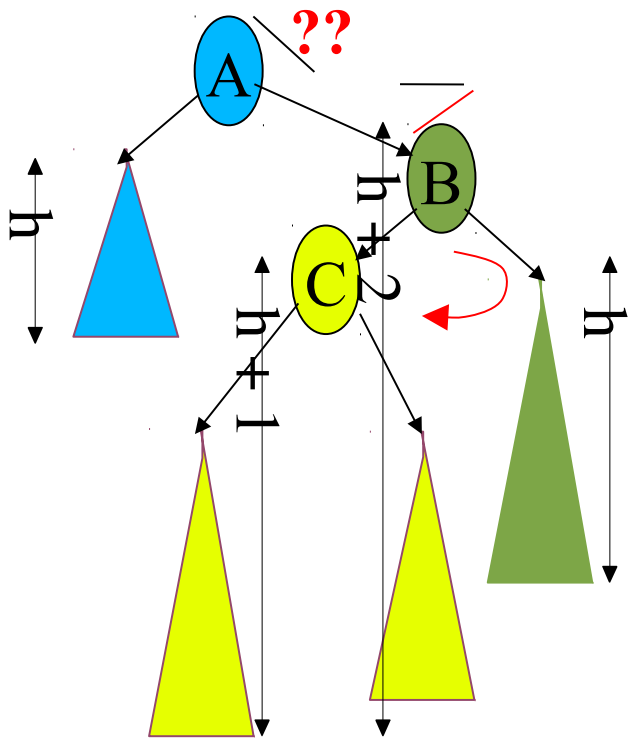
Highside child of A has opposite label from A

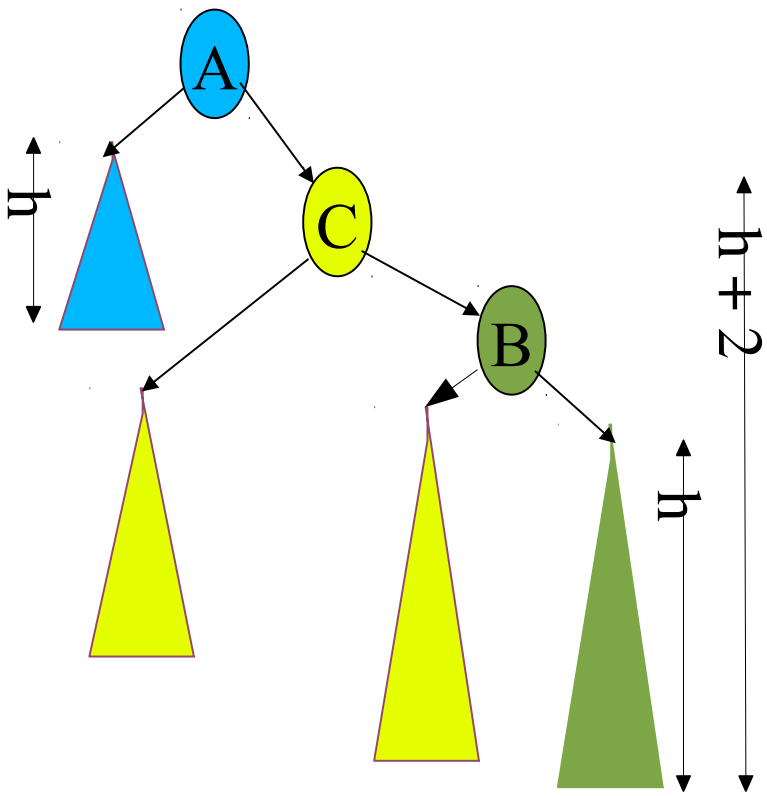


Still bad

Rebalancing

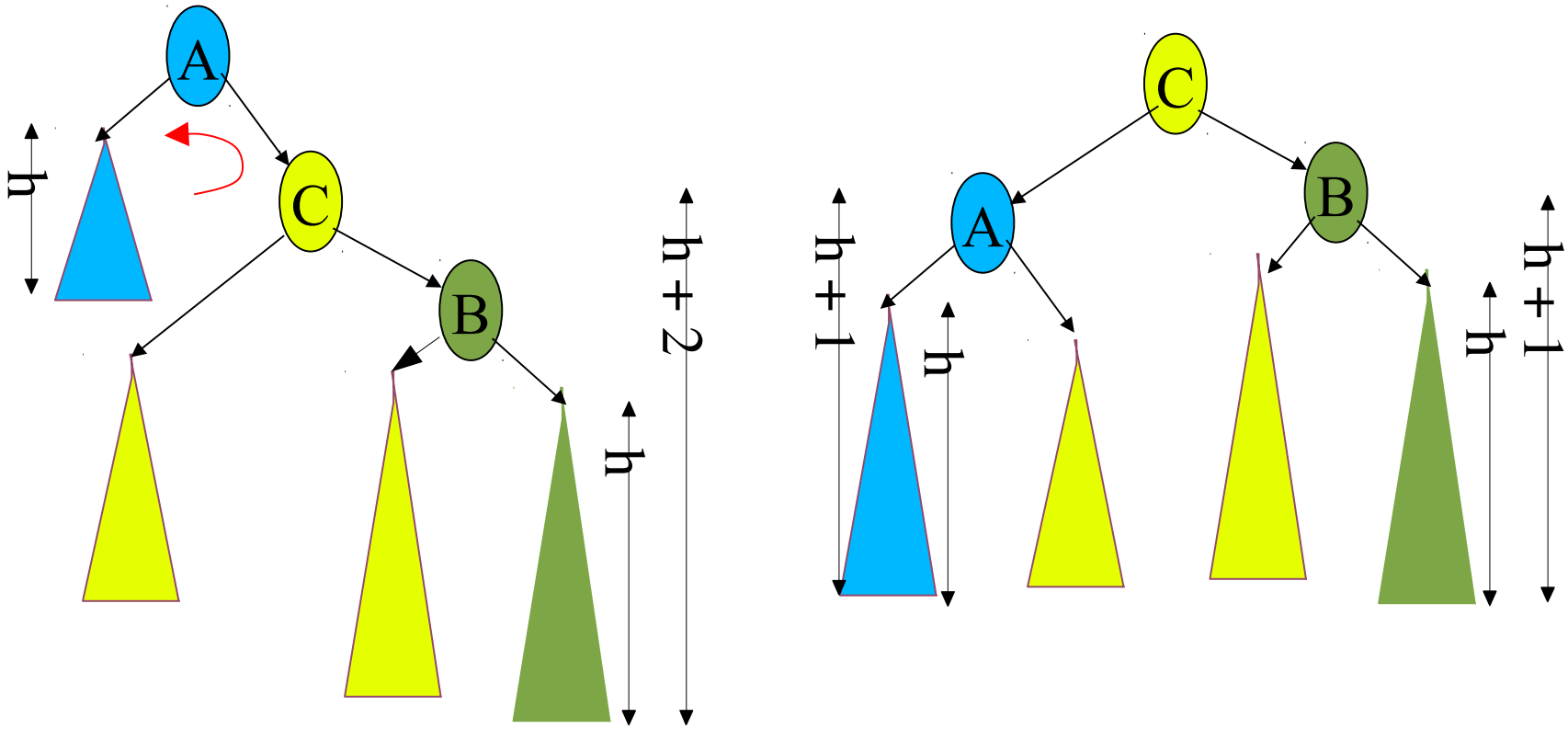
- **Solution: Rotate BC First**





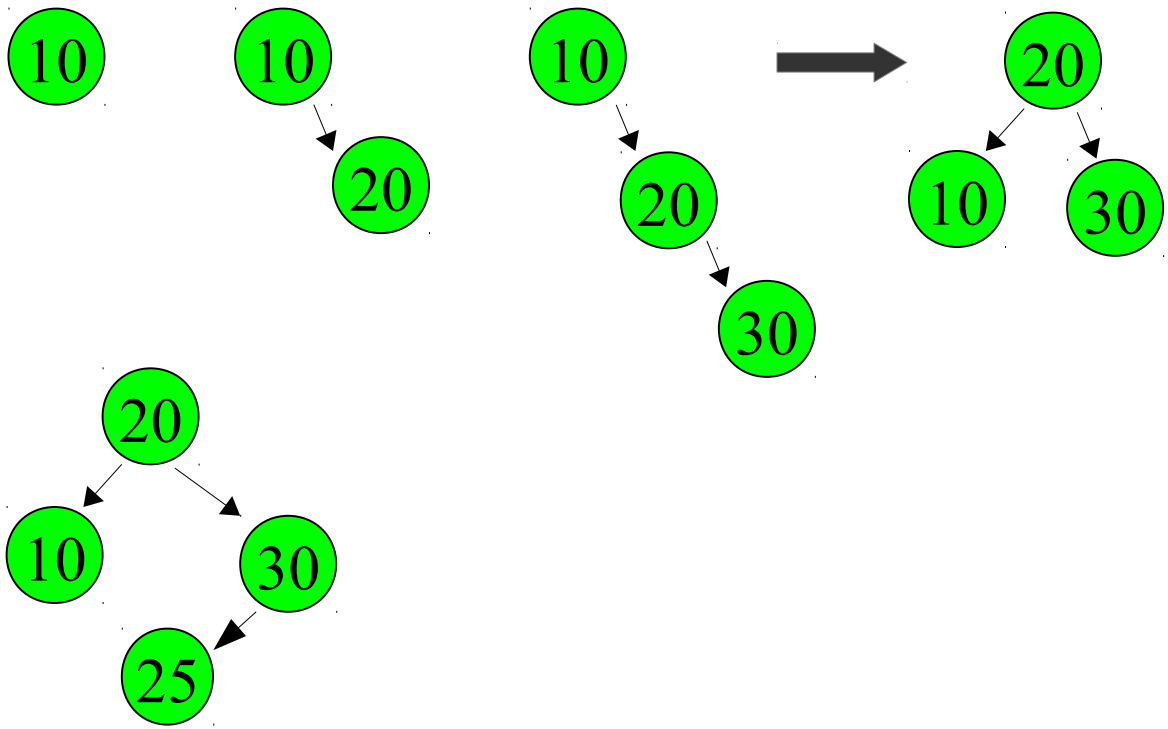
Rebalancing

- **Solution: Then Rotate AC**



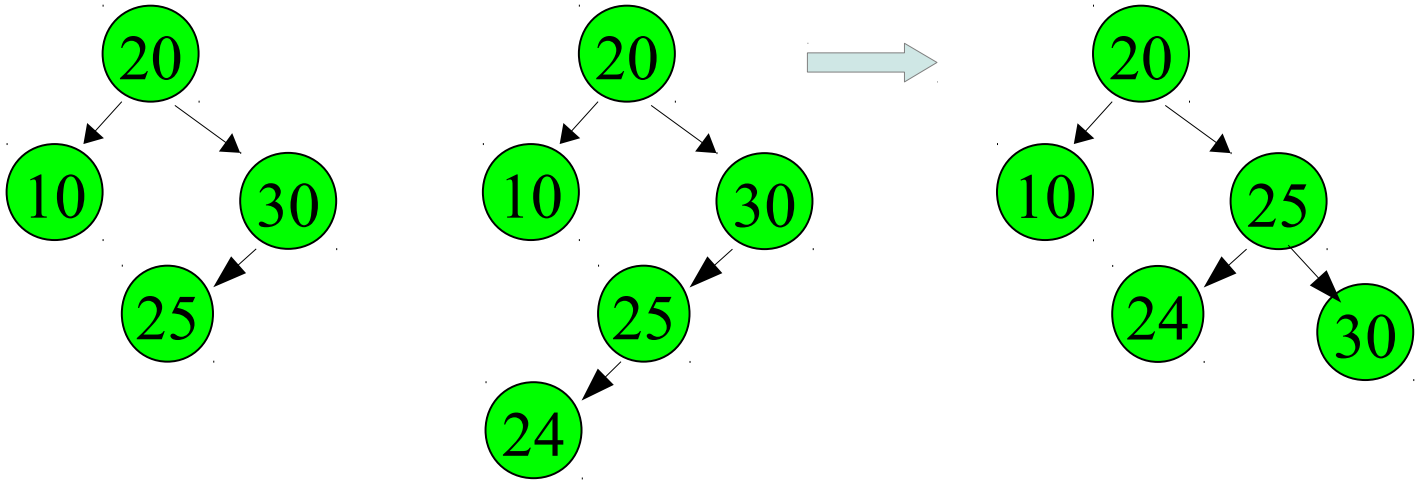
Example

- **insert 10, 20, 30, 25, ...**



Example

- **insert …, 24, …**



Example

- **insert …, 23, …**