ENSF 593/594
Data Structures – AVL Trees

Mohammad Moshirpour

Outline

- AVL Tree
- Insertion to AVL Trees

Goal

- In this lecture we will learn about AVL trees which are self balancing trees and make searches and insertions more efficient.

The Need for Balanced Trees

- Searches and insertions are most efficient when a binary tree is well balanced.
- Binary trees may become unbalanced after insertions and deletions.
  - In the worst case, the tree degenerates into a linked list.
- There are several variants of ordered binary trees that remain well balanced after insertions and deletions.
  - AVL trees are one example.

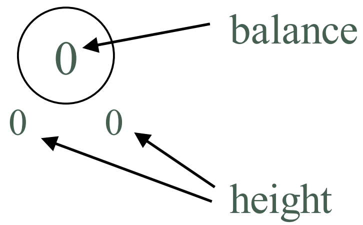
Balance of a Node

- **Definition:** is the height of the right subtree minus the height of the left subtree:

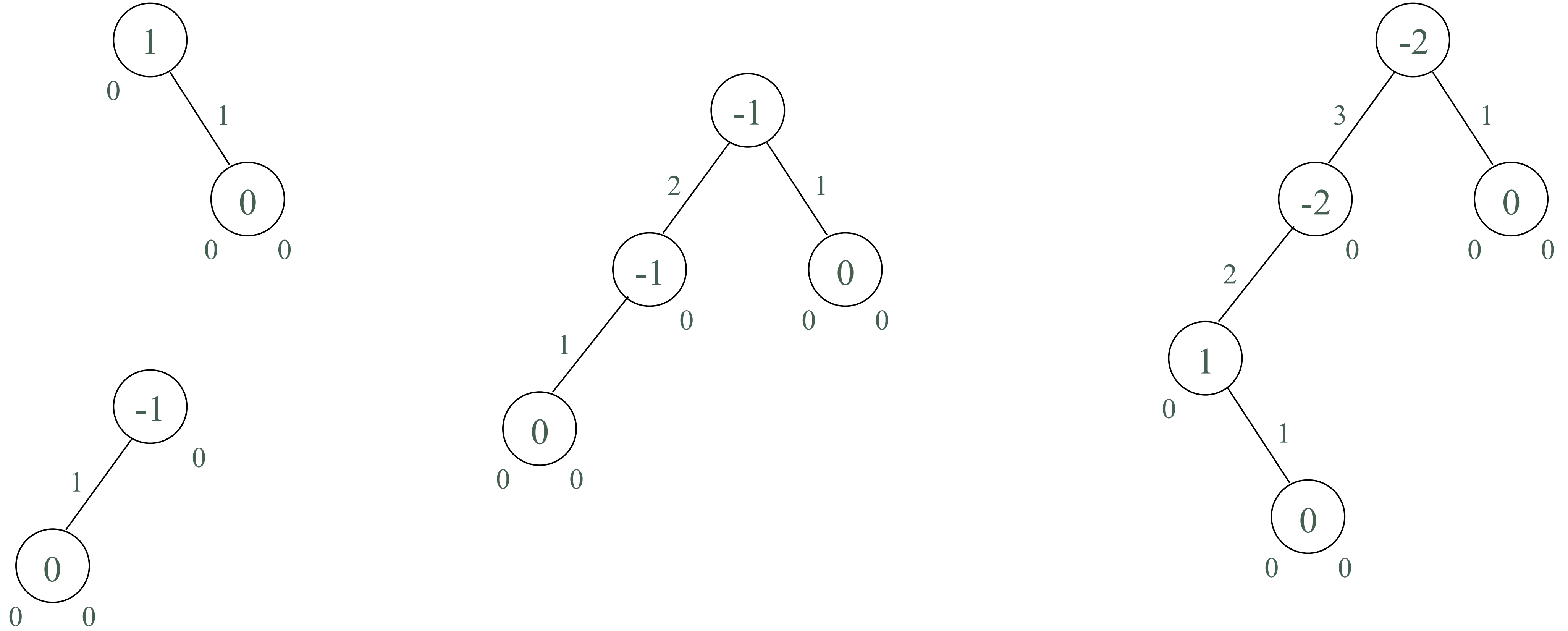
  balance(n) = rightHeight(n) - leftHeight(n)

  where n is some node in the tree

  - E.g.

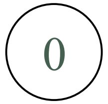


Balance of a Node (cont'd)

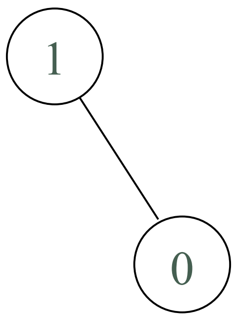


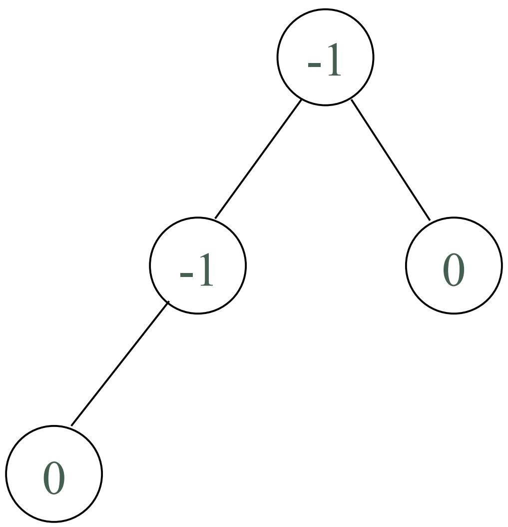
AVL Trees

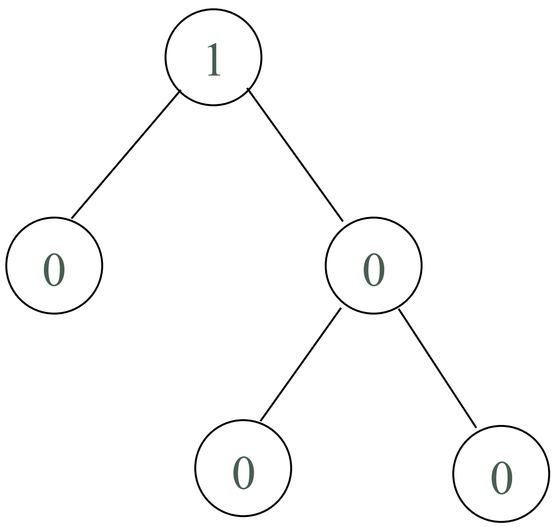
- Named after their inventors: G. M. Adel’son-Vel’skii and E. M. Landis
  - Developed in 1962
- Definition: is an ordered binary tree where every node has a balance of -1, or 0, or +1
  - E.g.



AVL Trees (cont’d)

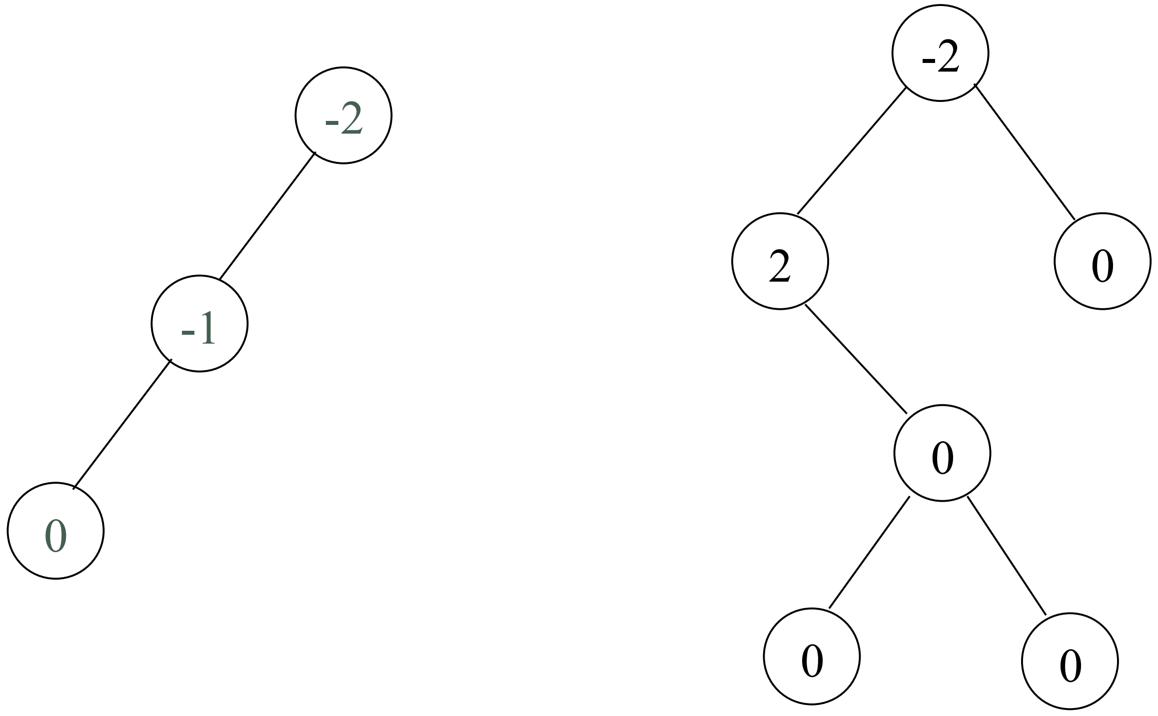






AVL Trees (cont'd)

- Note that the difference between the subtrees can never exceed 1
- Examples of non-AVL trees



Node Structure

- Must add a *balance* field to the Node class used for binary search trees:

```
public class Node {
   private int data;
   private Node parent, left, right;
   private int balance;

   . . .
}
```

Insertion into an AVL Tree

- General procedure:
  1) Insert the node into the tree, following the rules for a regular binary search tree
  2) If necessary, adjust the shape of the tree so that it conforms to the rules of an AVL tree
     - Involves doing a single or double *rotation*
  3) Update the balance fields for all nodes affected by the steps above

Insertion (cont'd)

- *Pivot Node*
  - **Definition:** is the ancestor node closest to the inserted node that is *not* in balance
    - i.e. is not 0
    - It is possible there may be no pivot when doing an insertion
  - One adjusts the AVL tree and updates the balances according to the nature of the pivot and where the insertion is done

Insertion (cont'd)

- There are 3 possible cases when doing an insertion:
  1) There is no pivot
  2) The pivot exists, and you add to the shorter subtree
  3) The pivot exists, and you add to the longer subtree

Insertion (cont’d)

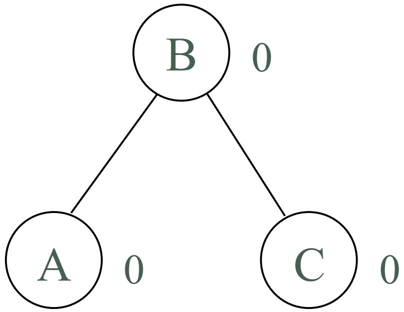
- **Case 1: There is no pivot**
  - Essentially, you are adding to a subtree with all 0 balances
    - You change the balances for all ancestor nodes by ± 1
  - The shape of the tree is *not* adjusted after the insertion
    - But the balances must be updated

Insertion (cont’d)

- **Procedure:**
  - Insert the node into its proper place in the tree
  - Adjust the balances for all nodes from the inserted node up to the root node (i.e. all nodes on the *search path*)
    - The inserted node is given a balance of 0
    - For the other nodes:
      - If inserted node < node value, decrement balance
      - If inserted node > node value, increment balance

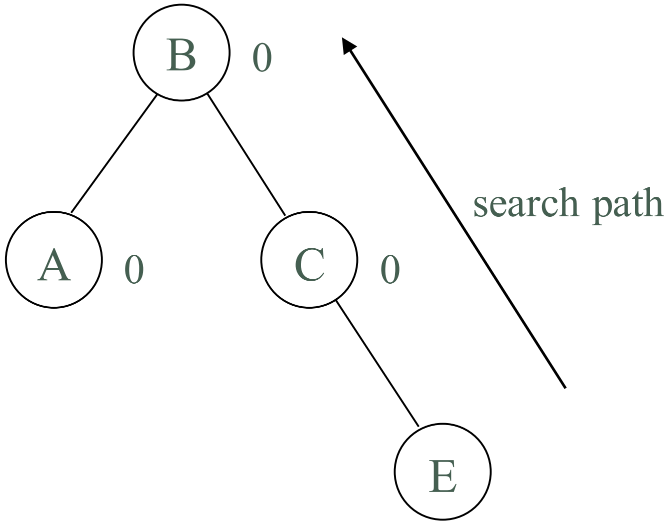
Insertion (cont’d)

- E.g.
  - Original tree



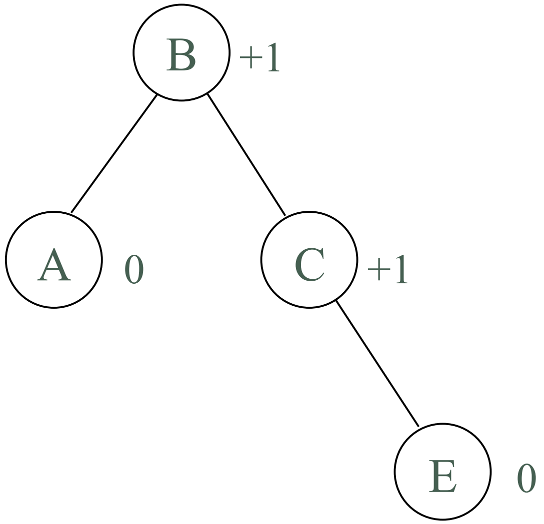
Insertion (cont'd)

- Insert E:



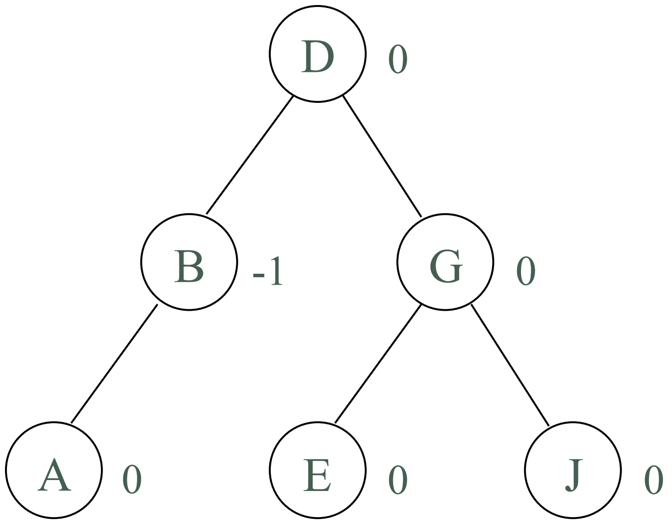
Insertion (cont’d)

- Update balances:



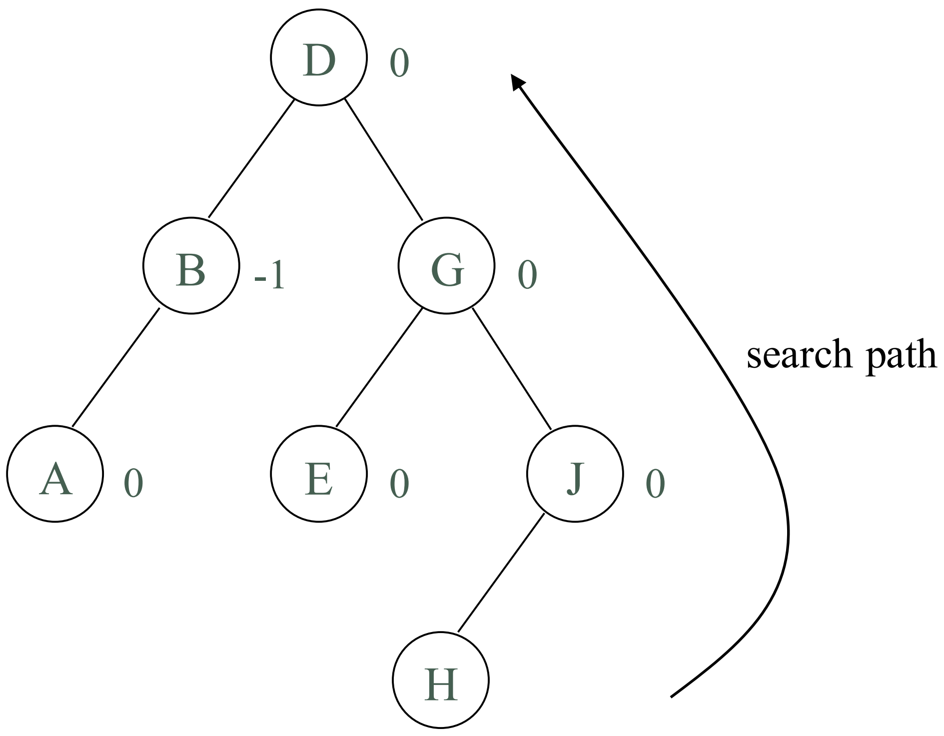
Insertion (cont'd)

- E.g.
  - Original tree



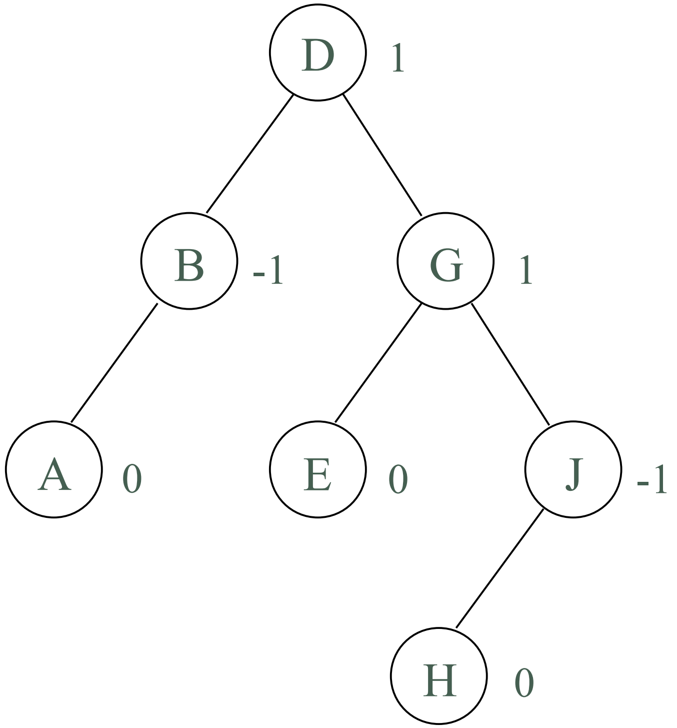
Insertion (cont'd)

- Insert H



Insertion (cont'd)

- Update balances



Insertion (cont'd)

- **Case 2: A pivot exists, and a node is added to the shorter subtree**
  - Essentially, you are adding to a shorter subtree to bring it into better balance
  - The shape of the tree is *not* adjusted after the insertion
    - But the balances must be updated

Insertion (cont'd)

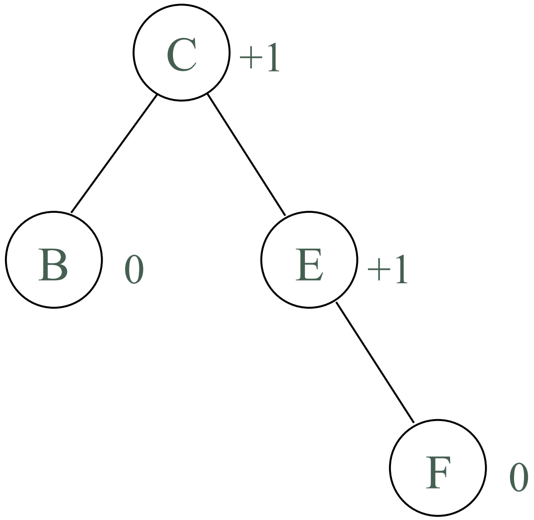
- You must be able to tell if you are adding to the shorter subtree (to distinguish from Case 3); you are if:
  - Pivot == +1 and inserted node < pivot node, or
  - Pivot == –1 and inserted node > pivot node

Insertion (cont’d)

- **Procedure:**
  - Insert the node into its proper place in the tree
  - Adjust the balances for all nodes from the inserted node up to and including the *pivot* node
    - The inserted node is given a balance of 0
    - For the other nodes:
      - If inserted node < node value, decrement balance
      - If inserted node > node value, increment balance
    - Note that balances do not change above the pivot node

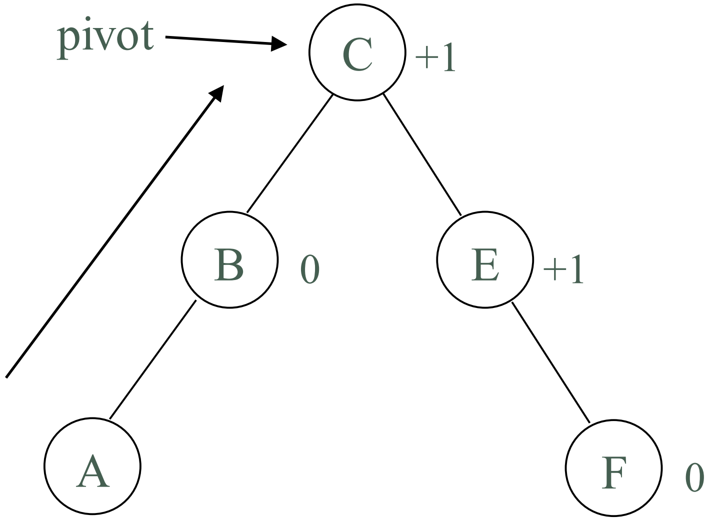
Insertion (cont'd)

- E.g.
  - Original tree



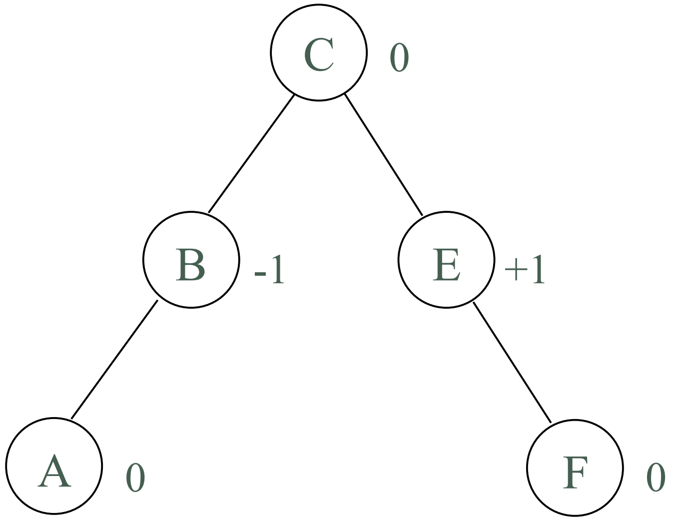
Insertion (cont'd)

- Insert A, identify pivot



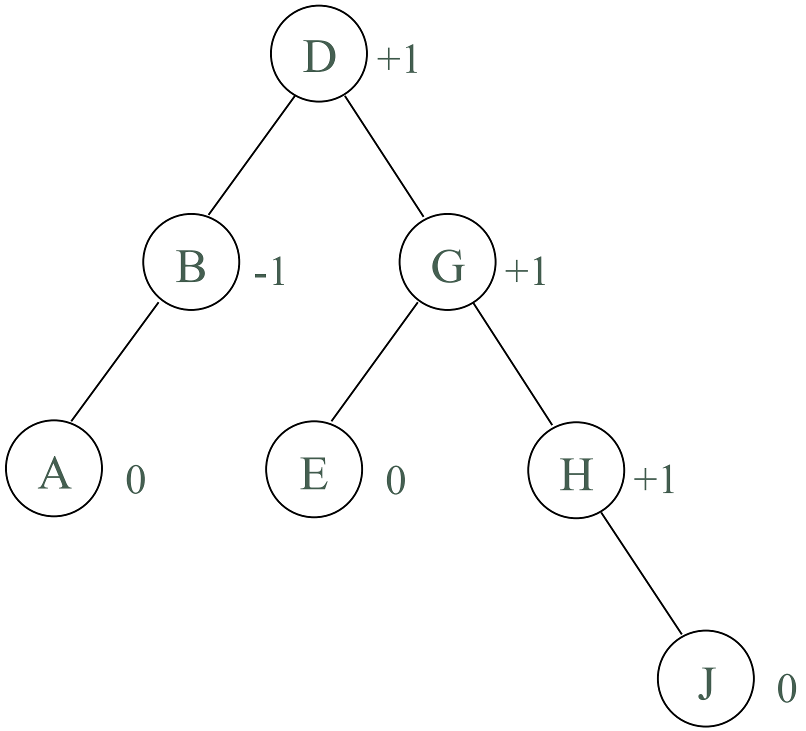
Insertion (cont’d)

- Adjust balances



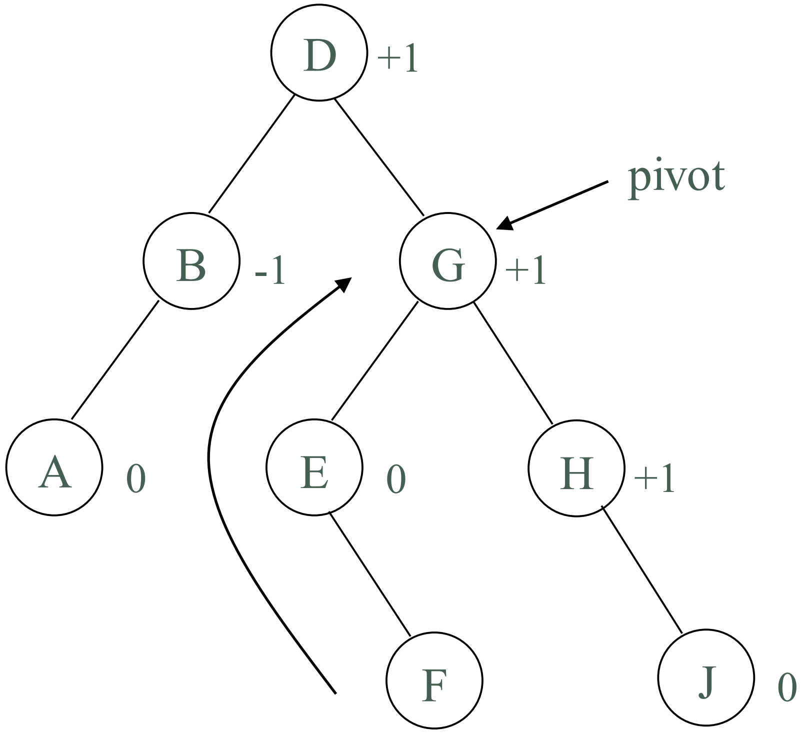
Insertion (cont'd)

- E.g.
  - Original tree



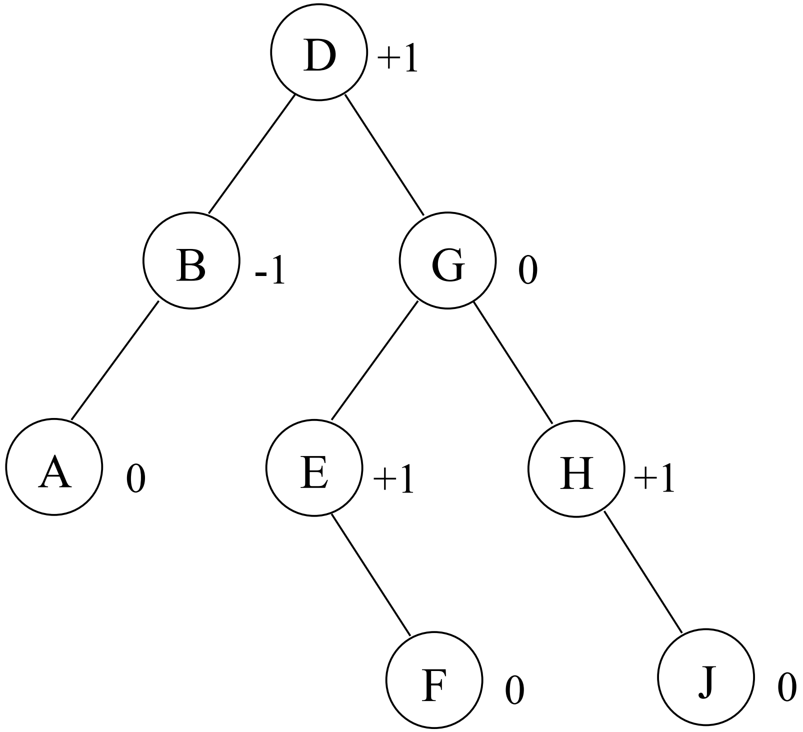
Insertion (cont'd)

- Add F, identify pivot



Insertion (cont’d)

- Adjust balances



Insertion (cont'd)

- **Case 3: A pivot exists, and you add to the longer subtree**
  - Essentially, you are putting the tree into worse balance
    - The pivot's balance changes to ± 2
  - The shape of the tree *must* be adjusted after doing the insertion

Insertion (cont'd)

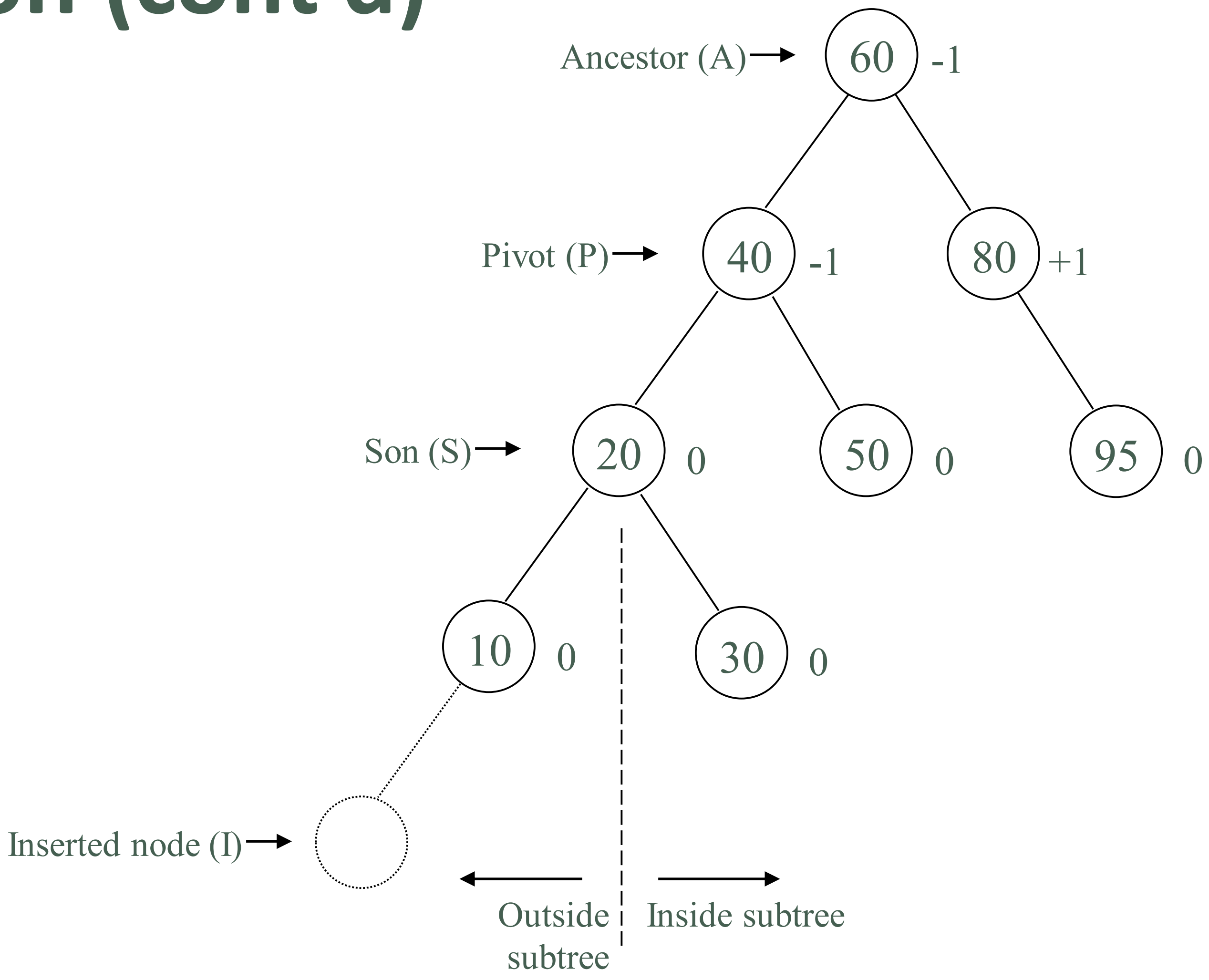
- You must be able to tell if you are adding to the longer subtree (to distinguish from Case 2); you are if:
  - Pivot == +1 and inserted node > pivot node, or
  - Pivot == –1 and inserted node < pivot node
- Case 3 breaks down into 2 subcases:
  - You add to the "outside subtree" of the "son" of the pivot on the search path
  - You add to the "inside subtree" of the "son" of the pivot on the search path

Insertion (cont'd)

- **Terminology:**
  - *Ancestor node*: the parent node of the pivot node
  - *Son node*: the child node of the pivot node, on the path from the pivot to the inserted node
  - *Outside subtree*:
    - The left subtree of the son, if the pivot is negative
    - The right subtree of the son, if the pivot is positive

Insertion (cont'd)

- E.g.



Insertion (cont'd)

- **Case 3a: adding a node to the outside subtree**
  - Procedure:
    1) Insert the node into its proper place in the tree
    2) Adjust the shape of the tree by doing a *single rotation*. 2 cases:
       a) Do a *right rotation* if the outside subtree is on the left (the pivot is negative)
       b) Do a *left rotation* if the outside subtree is on the right (the pivot is positive)

Insertion (cont'd)

3 pointers must be changed:

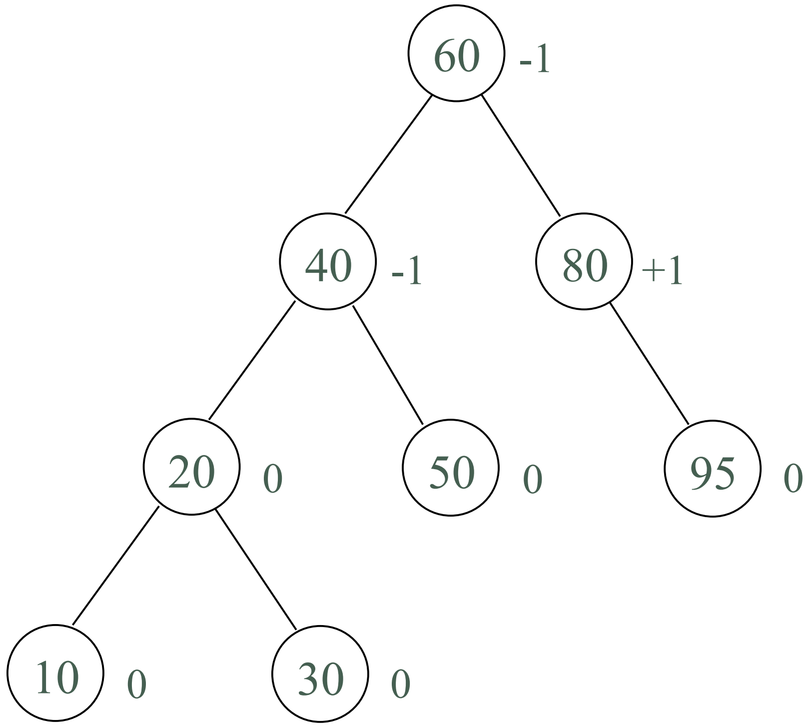
1) If pivot < ancestor, then ancestor's *left* child pointer is set to the son node, otherwise set the *right* child pointer
2) 2 cases:
   a) Right rotation: pivot's left child pointer set to the right child of the son node
   b) Left rotation: pivot's right child pointer is set to the left child of the son node
3) 2 cases:
   a) Right rotation: son's right child pointer set to the pivot node
   b) Left rotation: son's left child pointer set to the pivot node

Insertion (cont’d)

3) Adjust balances of affected nodes:
   a) Set pivot and inserted node to 0
   b) Adjust the balances for all nodes above the inserted node, up to the child of the son node
      - If inserted node < node value, decrement balance
      - If inserted node > node value, increment balance

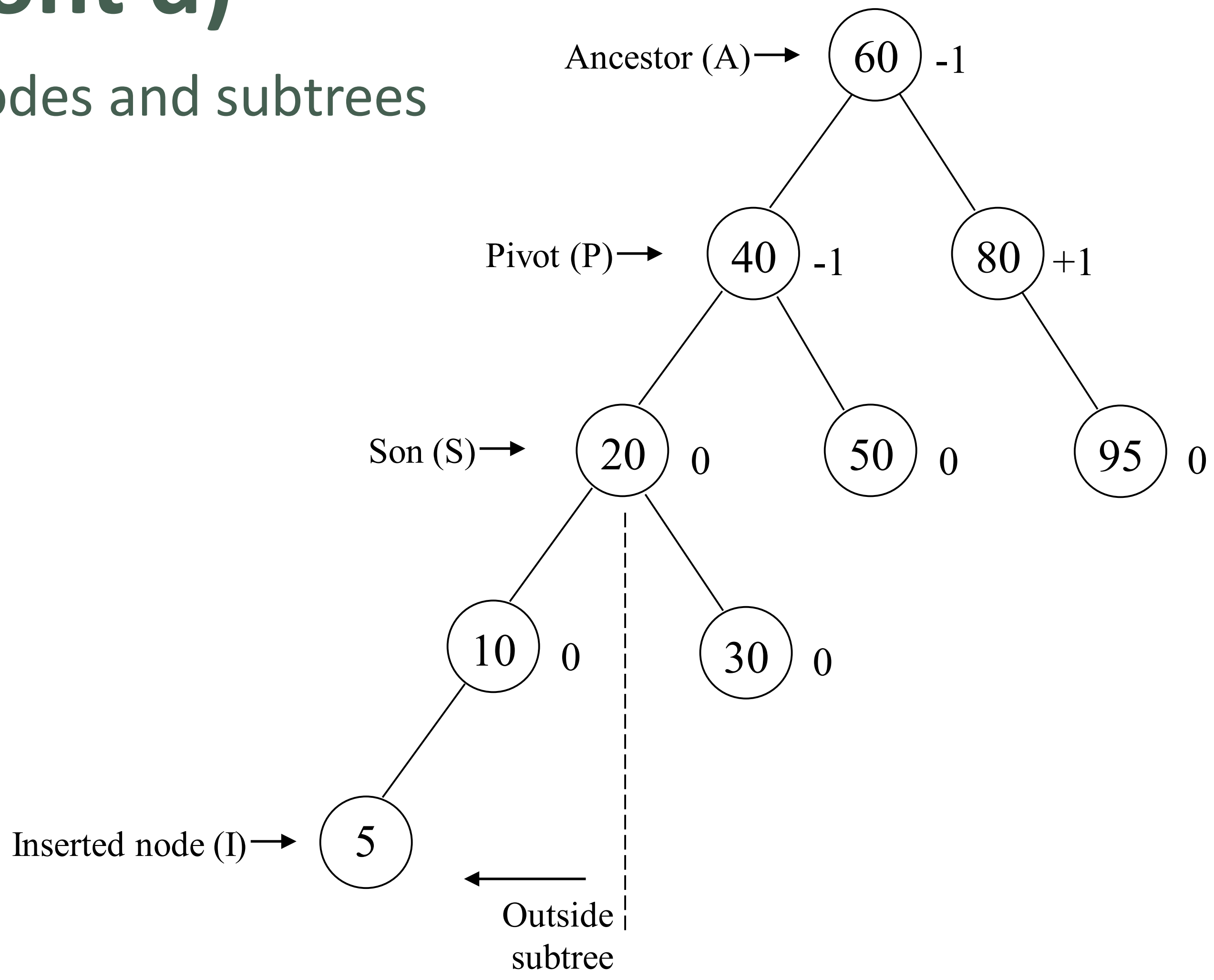
Insertion (cont'd)

- E.g.
  - Original tree



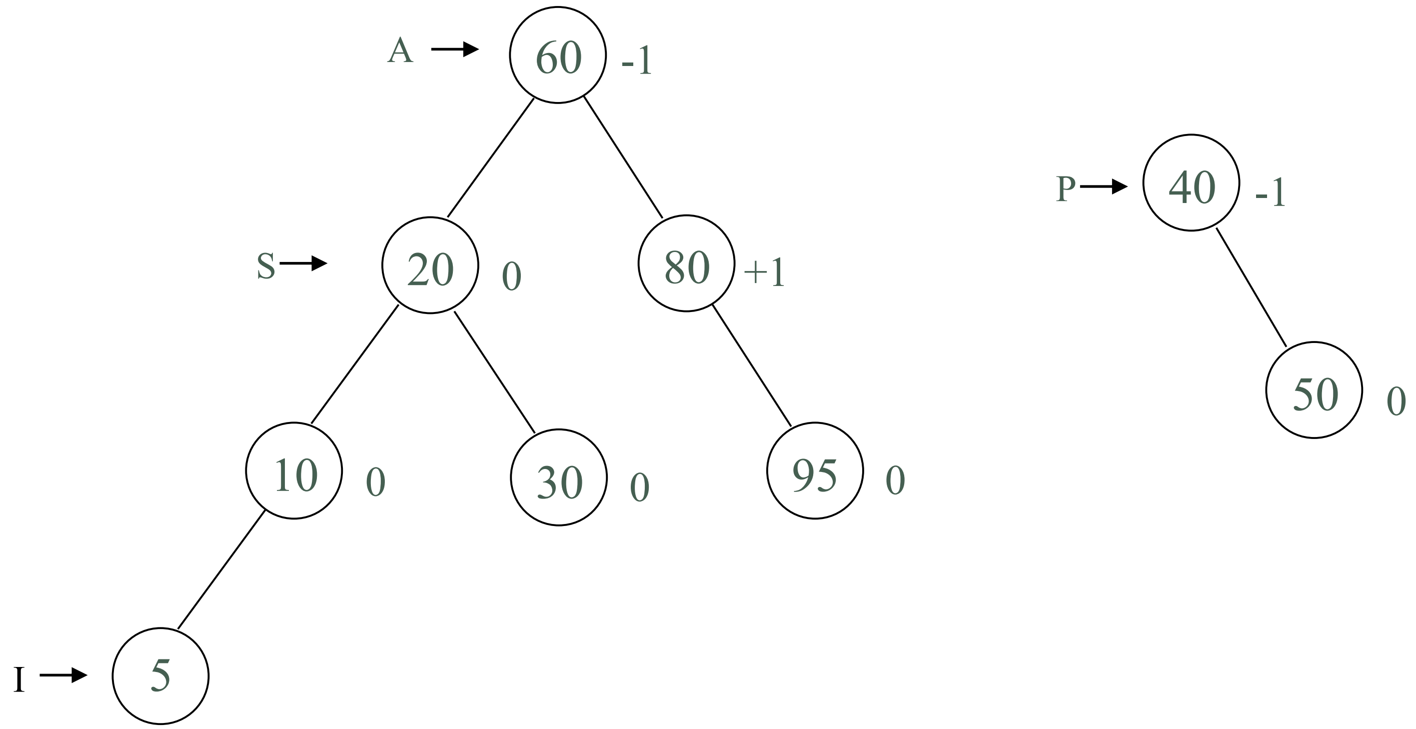
Insertion (cont'd)

- Add 5, identify nodes and subtrees

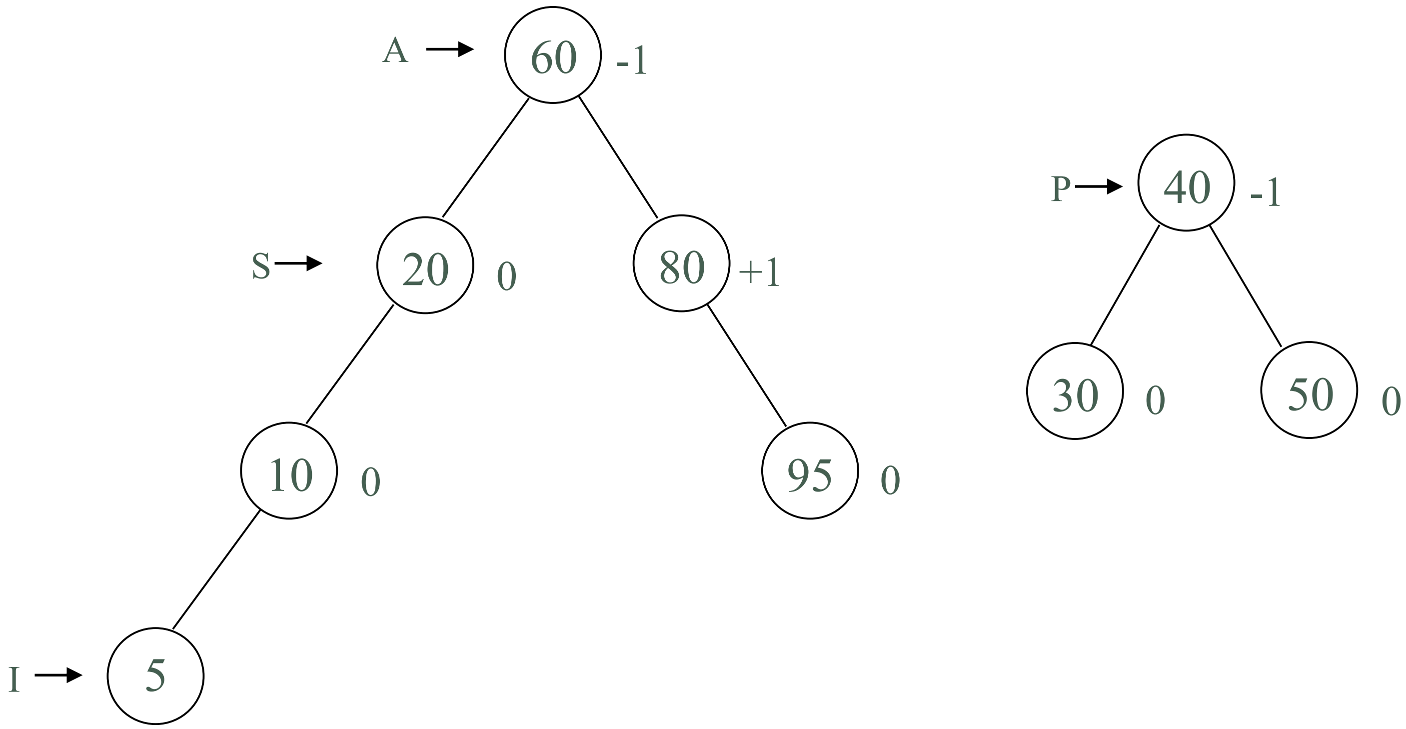


Insertion (cont'd)

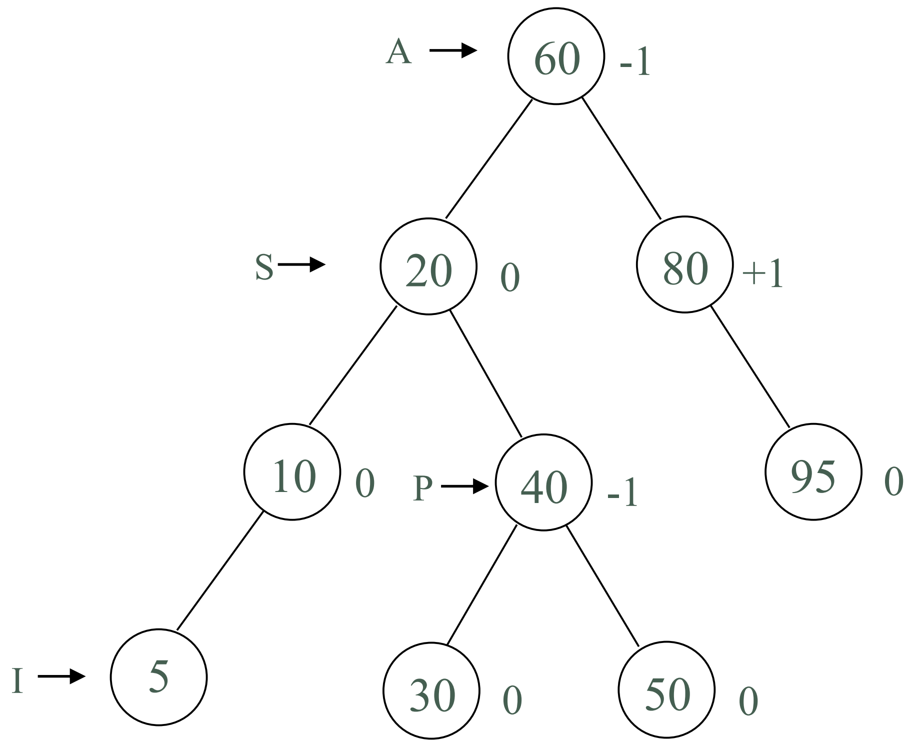
- Do right rotation (3 substeps)



Insertion (cont'd)

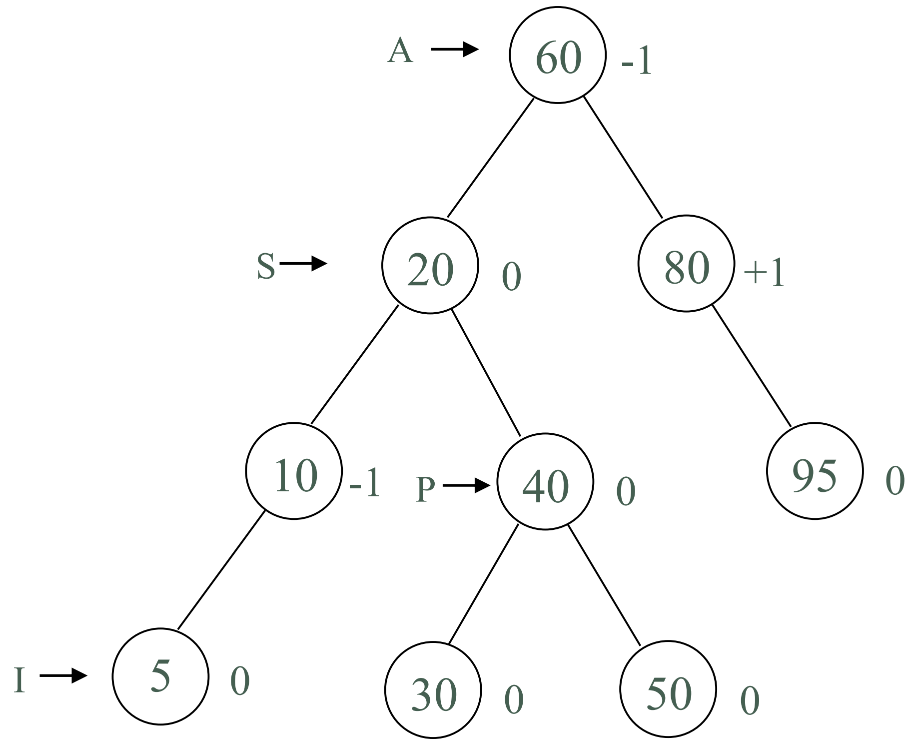


Insertion (cont'd)



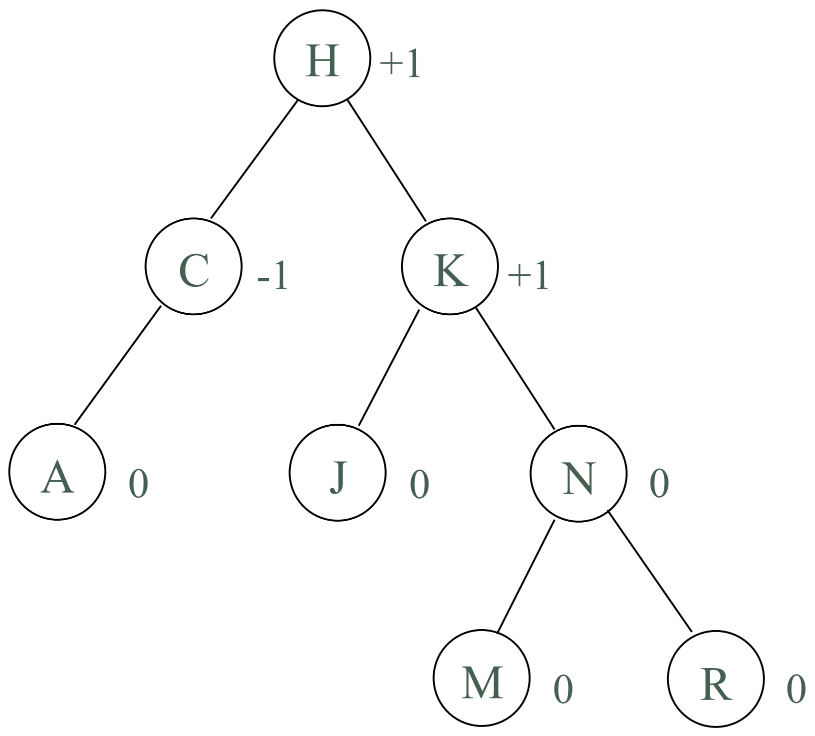
Insertion (cont’d)

- Adjust balances



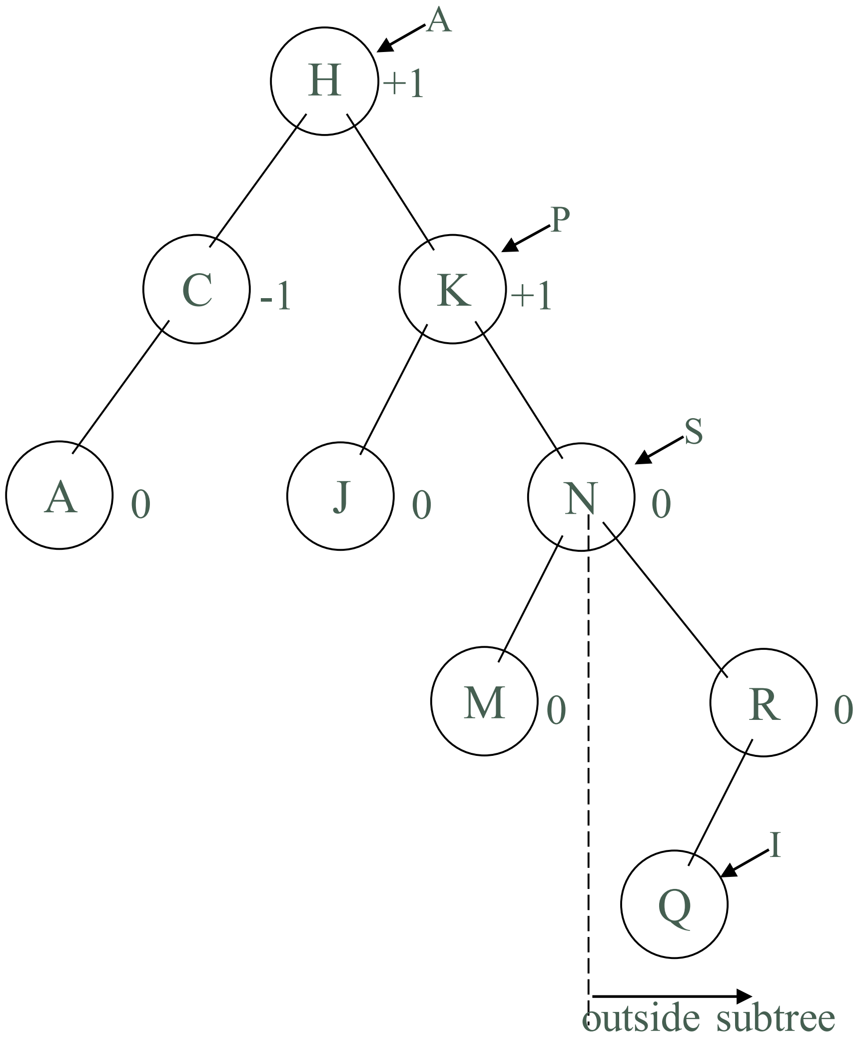
Insertion (cont'd)

- E.g.
  - Original tree



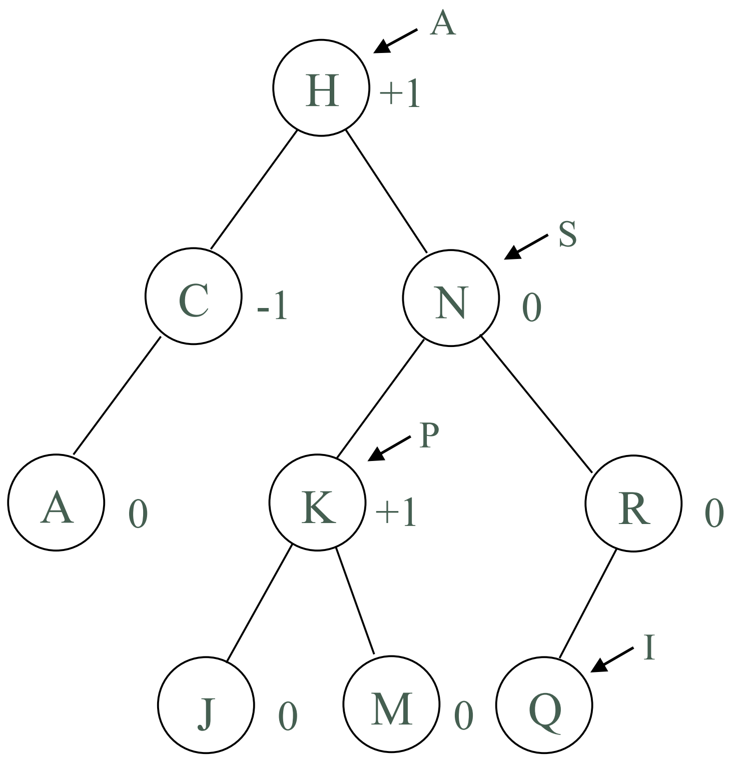
Insertion (cont'd)

- Add Q, identify nodes and subtrees

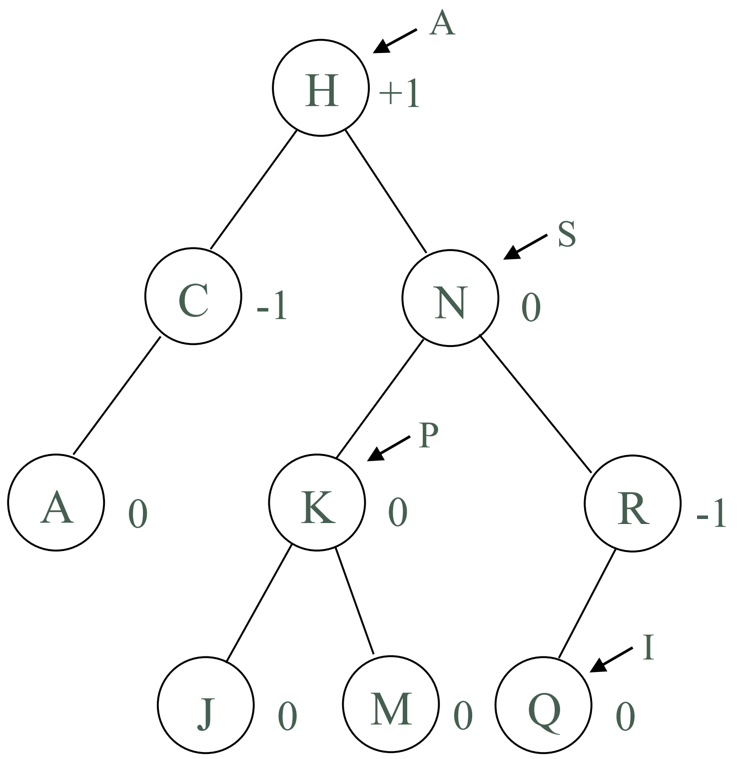


Insertion (cont'd)

- Do left rotation



- Adjust balances

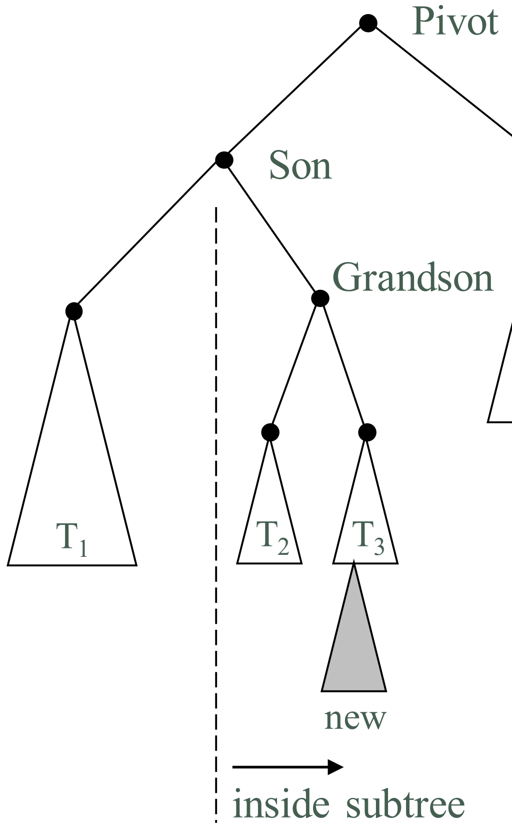


Insertion (cont'd)

- **Case 3b: adding a node to the inside subtree**
  - *Grandson node*: the child node of the son node, on the path from the pivot to the inserted node
  - To adjust the tree after an insertion, a *double rotation* is performed; consists of:
    - A right rotation at one node, followed by a left rotation at another node (RL rotation), or
    - The inverse (LR rotation)

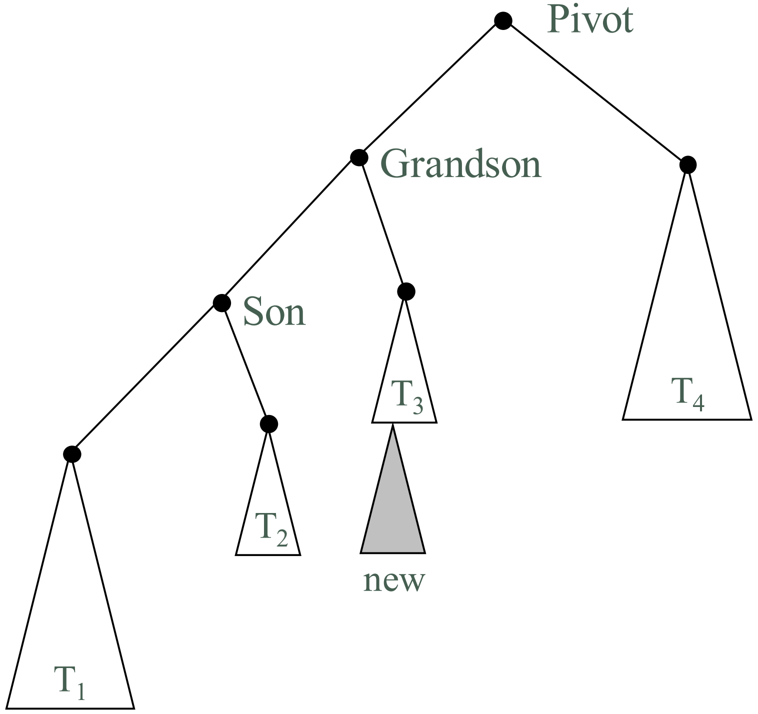
Insertion (cont'd)

- Example: LR double rotation



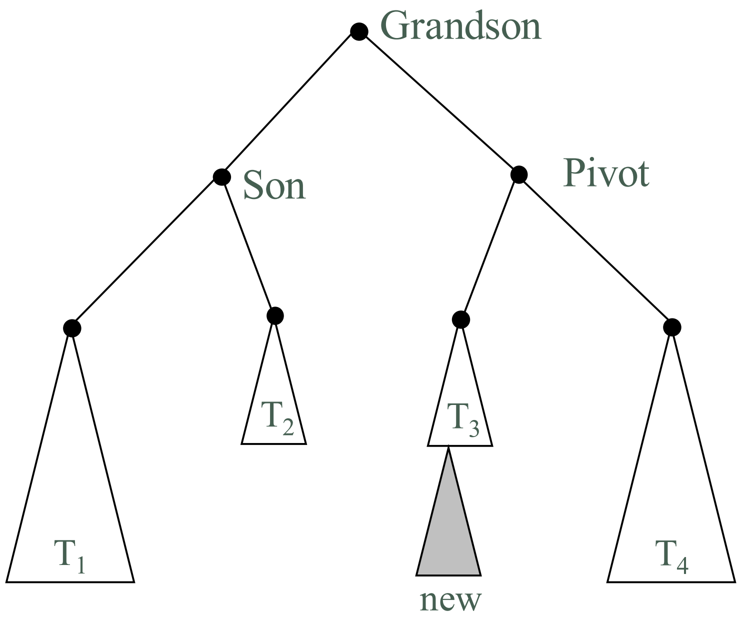
Insertion (cont’d)

- Left rotation at son



Insertion (cont’d)

- Right rotation at pivot



Insertion (cont’d)

- **Procedure:**
  1) Insert the node into its proper place in the tree
  2) Adjust the shape of the tree by doing a double rotation; 2 cases:
     a) RL rotation if the pivot is positive
     b) LR rotation if the pivot is negative

Insertion (cont'd)

- RL rotation:
  1) Right rotation through son
     a) Set pivot's right child pointer to grandson node
     b) Set son's left child pointer to grandson's right subtree (if it exists)
     c) Set grandson's right child pointer to son node
  2) Left rotation through pivot
     a) If there is no ancestor, set the root pointer to grandson; if pivot > ancestor, set the ancestor's right child pointer to the grandson, else set the left child pointer

Insertion (cont’d)

   b) Set pivot’s right child pointer to grandon’s left subtree (if it exists)
   c) Set grandson’s left child pointer to pivot

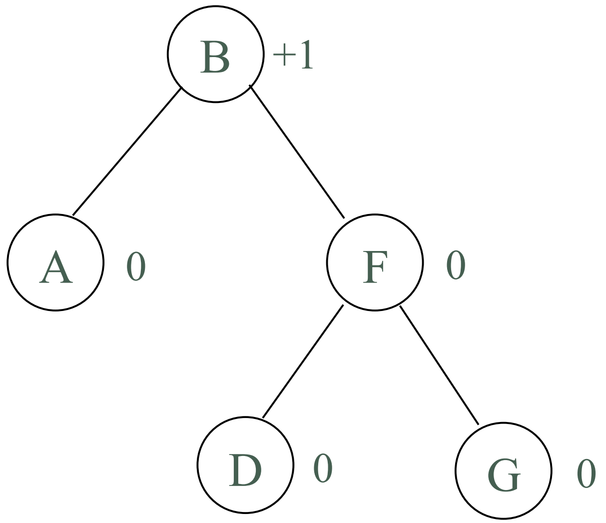
- LR rotation is symmetrical to the RL rotation

Insertion (cont'd)

3) Adjust balances of affected nodes
    a) Set inserted node to 0
    b) RL rotation:
        - If inserted node > grandson, set pivot to -1
        - Else set pivot to 0, son to +1
    c) LR rotation is symmetrical to the above
    d) Adjust balances for all nodes above the inserted node up to the child of the son or pivot
        - If inserted node < node value, decrement balance
        - If inserted node > node value, increment balance

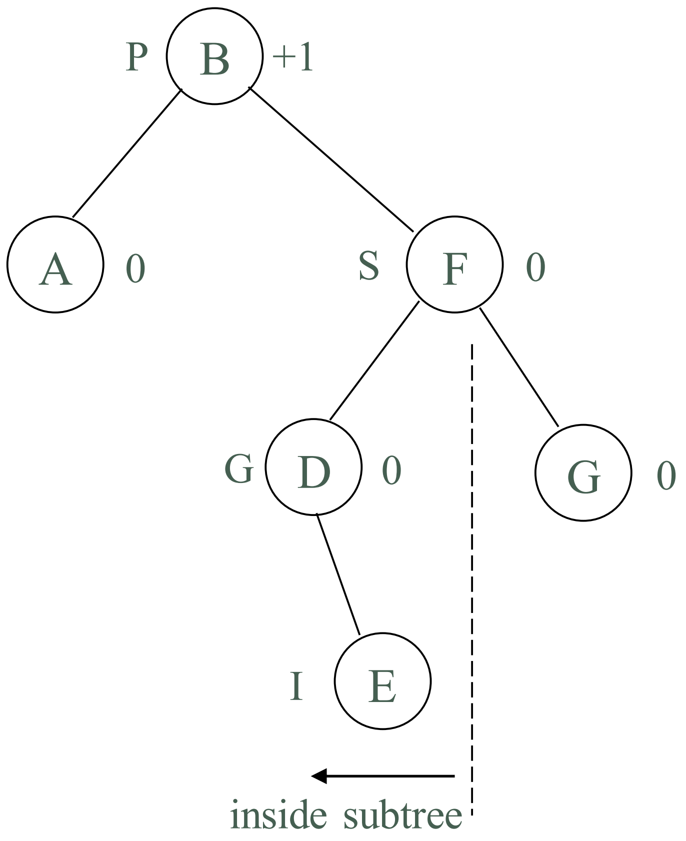
Insertion (cont'd)

- E.g.
  - Original tree



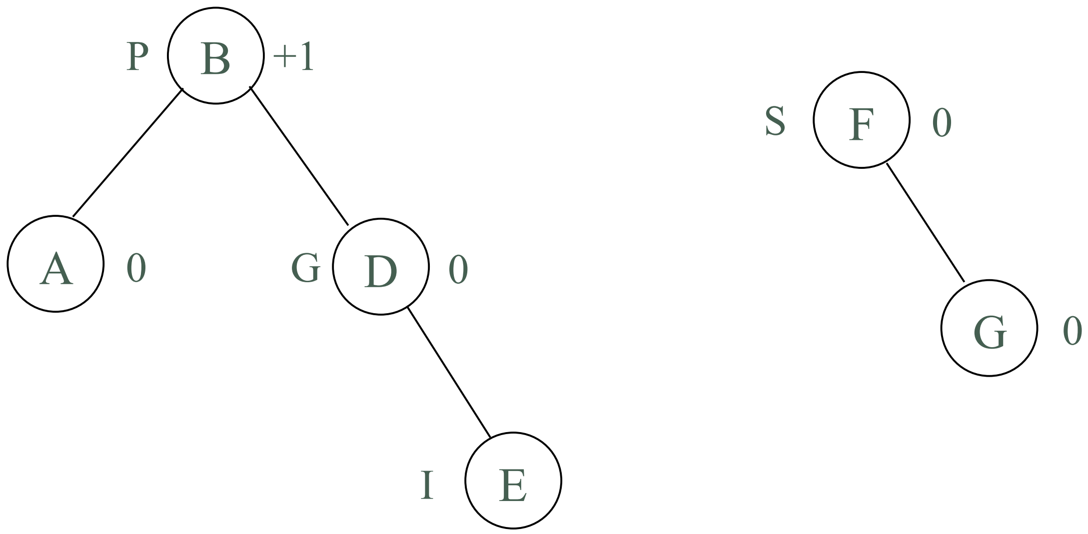
Insertion (cont'd)

- Add E, identify nodes and subtree

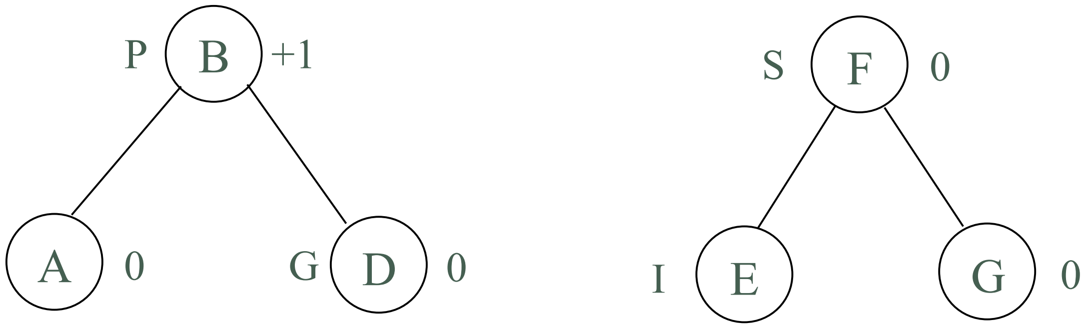


Insertion (cont’d)

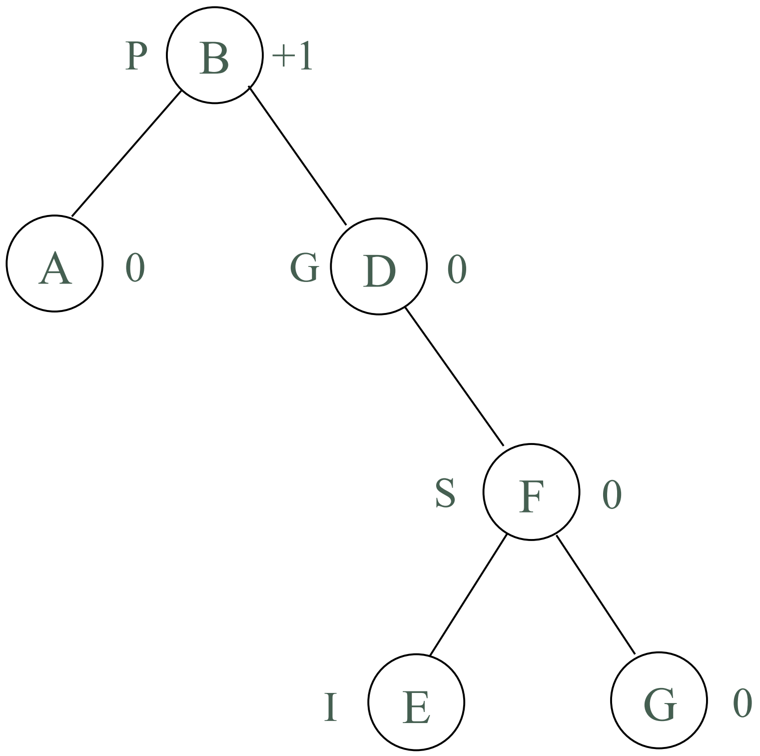
- Right rotation at son (3 substeps)



Insertion (cont'd)

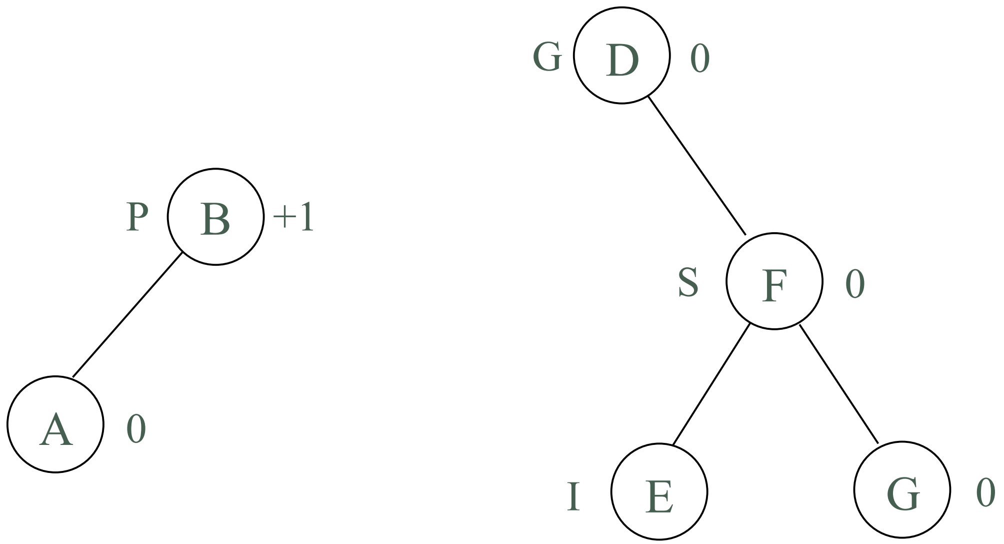


Insertion (cont'd)

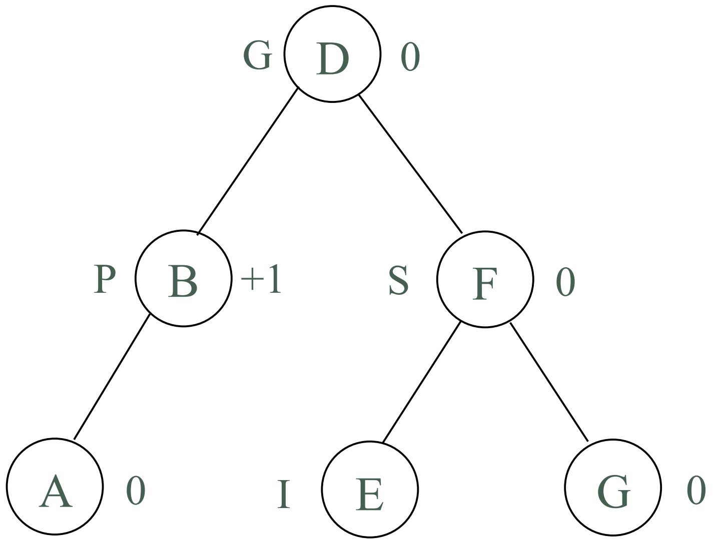


Insertion (cont'd)

- Left rotation at pivot (3 substeps)

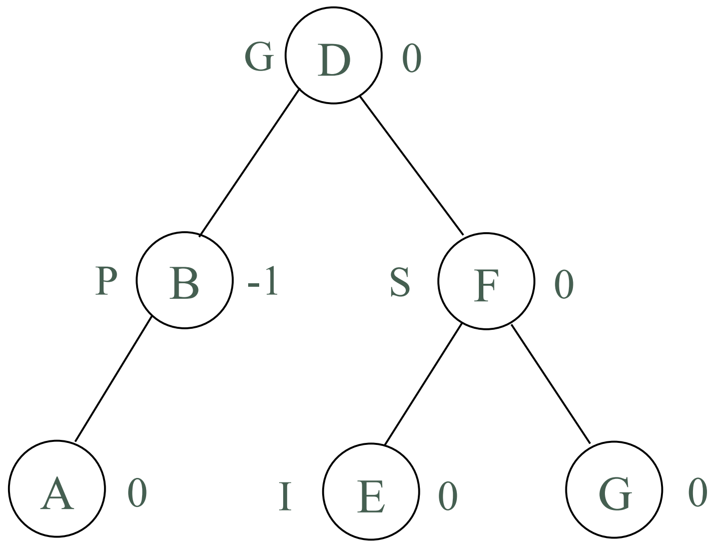


Insertion (cont’d)



Insertion (cont’d)

- Adjust balances



Summary

- AVL Trees: the difference between the subtrees can never exceed 1.
- Insertion:
  - There is no pivot
  - The pivot exists, and add to the shorter subtree
  - The pivot exists, and add to the longer subtree
    - adding a node to the outside subtree
    - adding a node to the inside subtree

Review Questions

- How can you calculate the Balance of a Node?
- What is an AVL tree?
- What is the difference between nodes in a binary and AVL trees?
- What is the general procedure of inserting nodes to an AVL tree?
- What is a Pivot Node?
- What should you do when there is no pivot while adding a node to an AVL?
- What should you do when a pivot exists, and a node is added to the shorter subtree while adding a node to an AVL?
- What should you do when A pivot exists, and you add to the longer subtree while adding a node to an AVL?

Any questions?