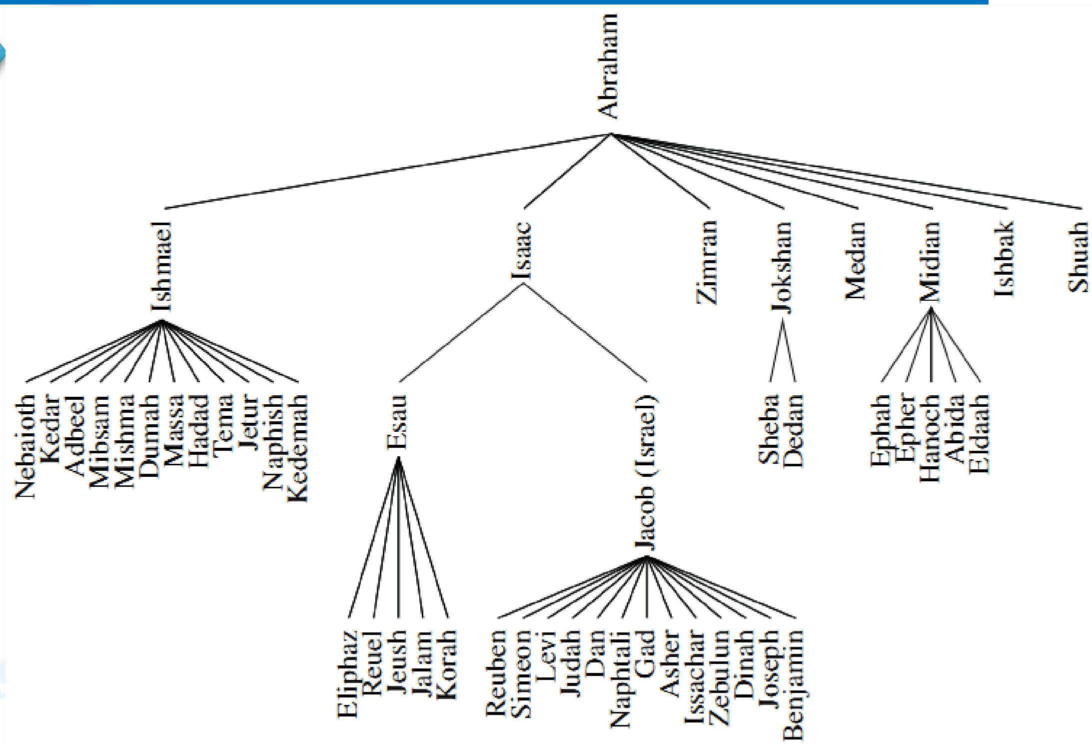
FACULTY OF INFORMATION TECHNOLOGY

DATA STRUCTURES (CTDL)

Semester 1, 2021/2022

Tree


Abraham
Ishmael
Isaac
Zimran
Jokshan
Medan
Midian
Ishbak
Shuah
Nebaioth
Kedar
Adbeel
Mibsam
Mishma
Dumah
Massa
Hadad
Tema
Jetur
Naphish
Kedemah
Esau
Jacob (Israel)
Sheba
Dedan
Ephah
Epher
Hanoch
Abida
Eldaah
Eliphaz
Reuel
Jeush
Jalam
Korah
Reuben
Simeon
Levi
Judah
Dan
Naphtali
Gad
Asher
Issachar
Zebulun
Dinah
Joseph
Benjamin

Why Trees?

- Trees are one of the fundamental data structures.
- Many real-world phenomena cannot be represented with data structures we've had so far.

- Arrays:
  - Easy to **search**, especially if ordered.
    - $O(\log_2 n)$ performance! Great. (binary search)
  - **Inserting**; **Deleting**? Horrible if ordered. Must find item or place before actions.

Why Trees?

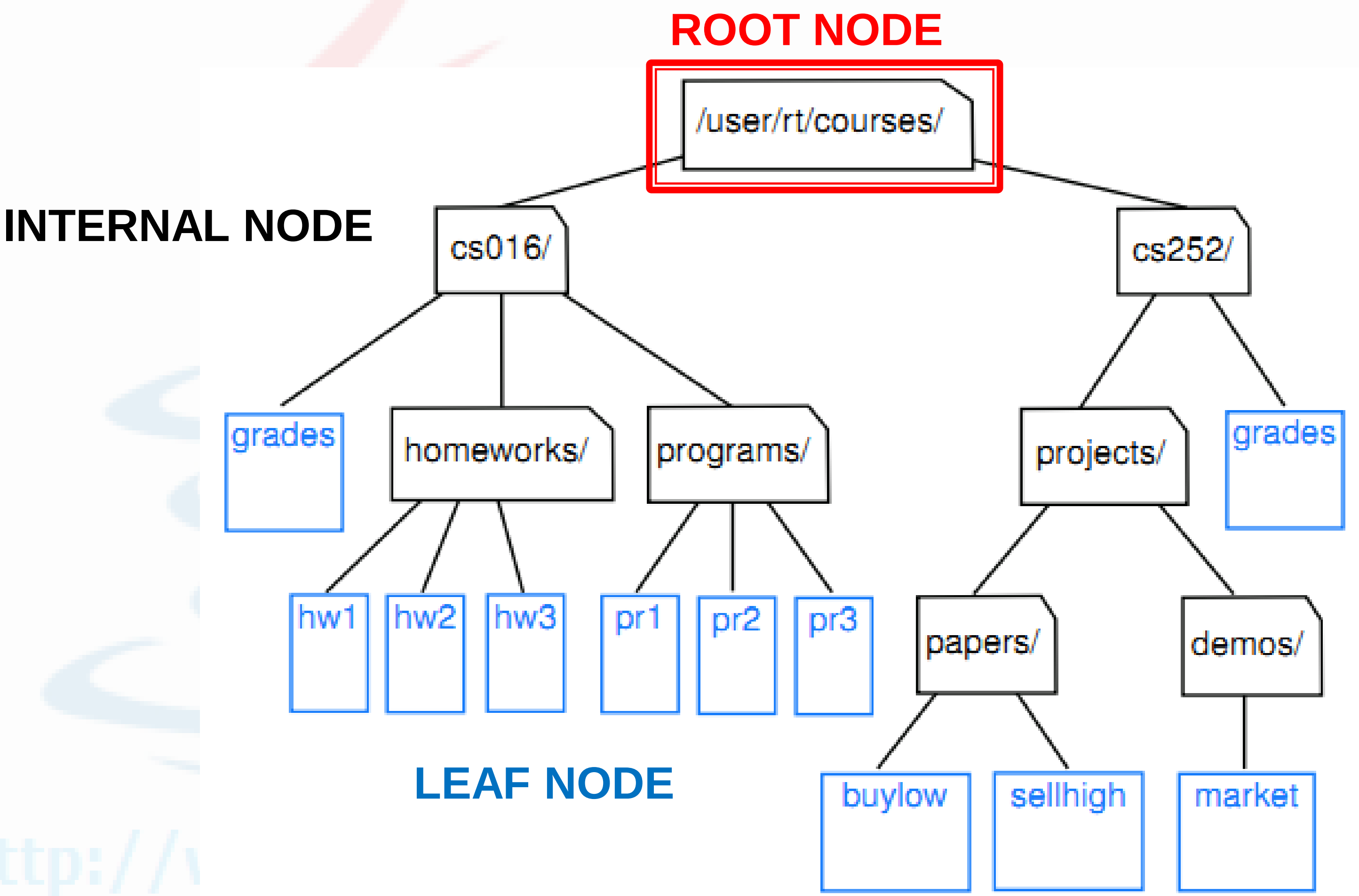
- How about Linked Lists?
  - Inserts and deletes? Great. Take O(1) time – the best you can get! (if inserting / deleting from one end)
  - Searching? Search to insert / delete/ change? Not nearly as good as O(1) or even $O(\log_2 n)$!
    - On average, must search n/2 items!
    - Process requires O(n) time.
    - Ordering the linked list may help, as we must still search to find.

Definition

- A tree T: a set of nodes storing elements such that the nodes have a parent-child relationship satisying the following properties:
  - If T is nonempty, it has a special node, called the root of T, that has no parent.
  - Each node v of T different from the root has a unique parent node w; every node with parent w is a child of w.

Tree Example

- Tree representing a portion of a file system

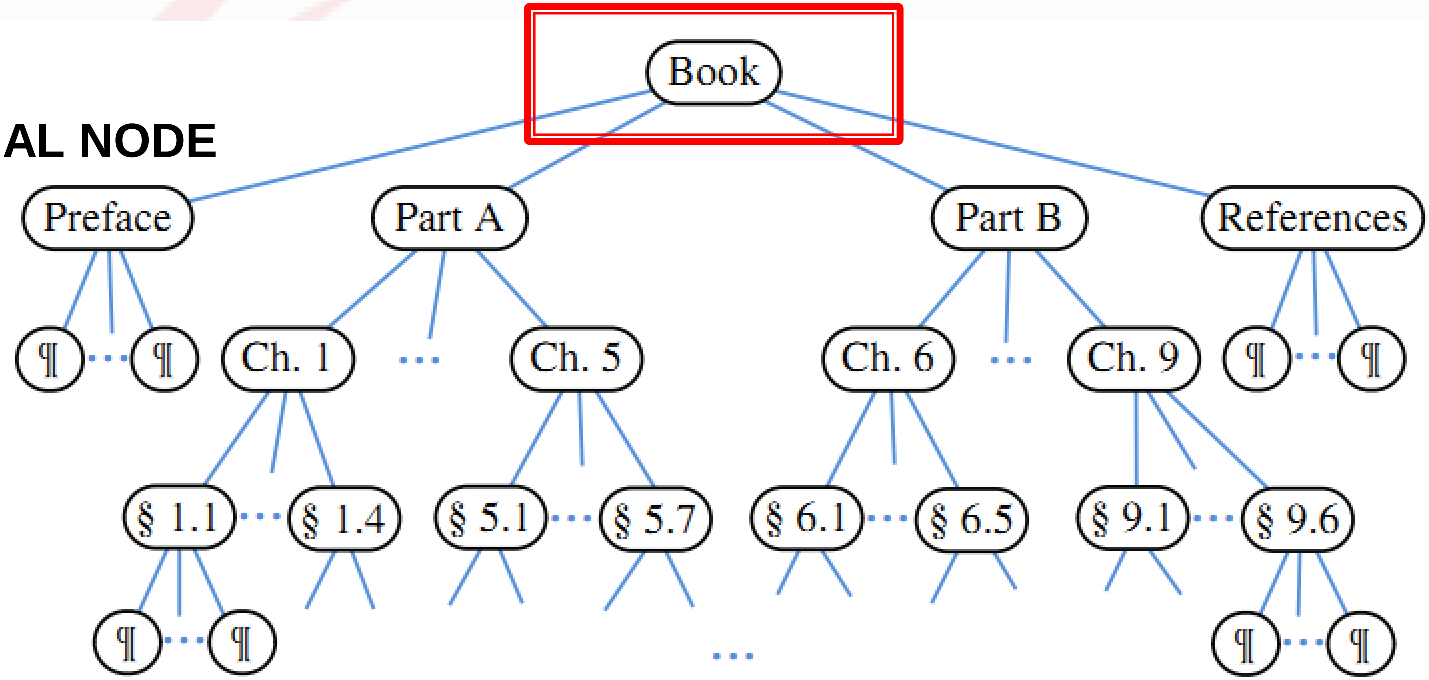


Tree example (cont.)

- An ordered tree associated with a book

ROOT NODE

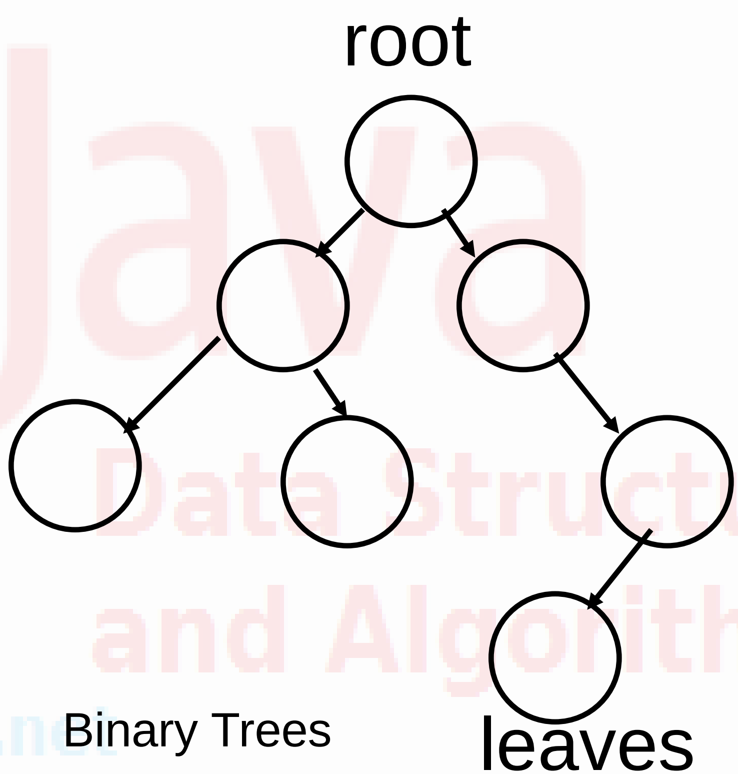
INTERNAL NODE



LEAF NODE

Properties of Trees

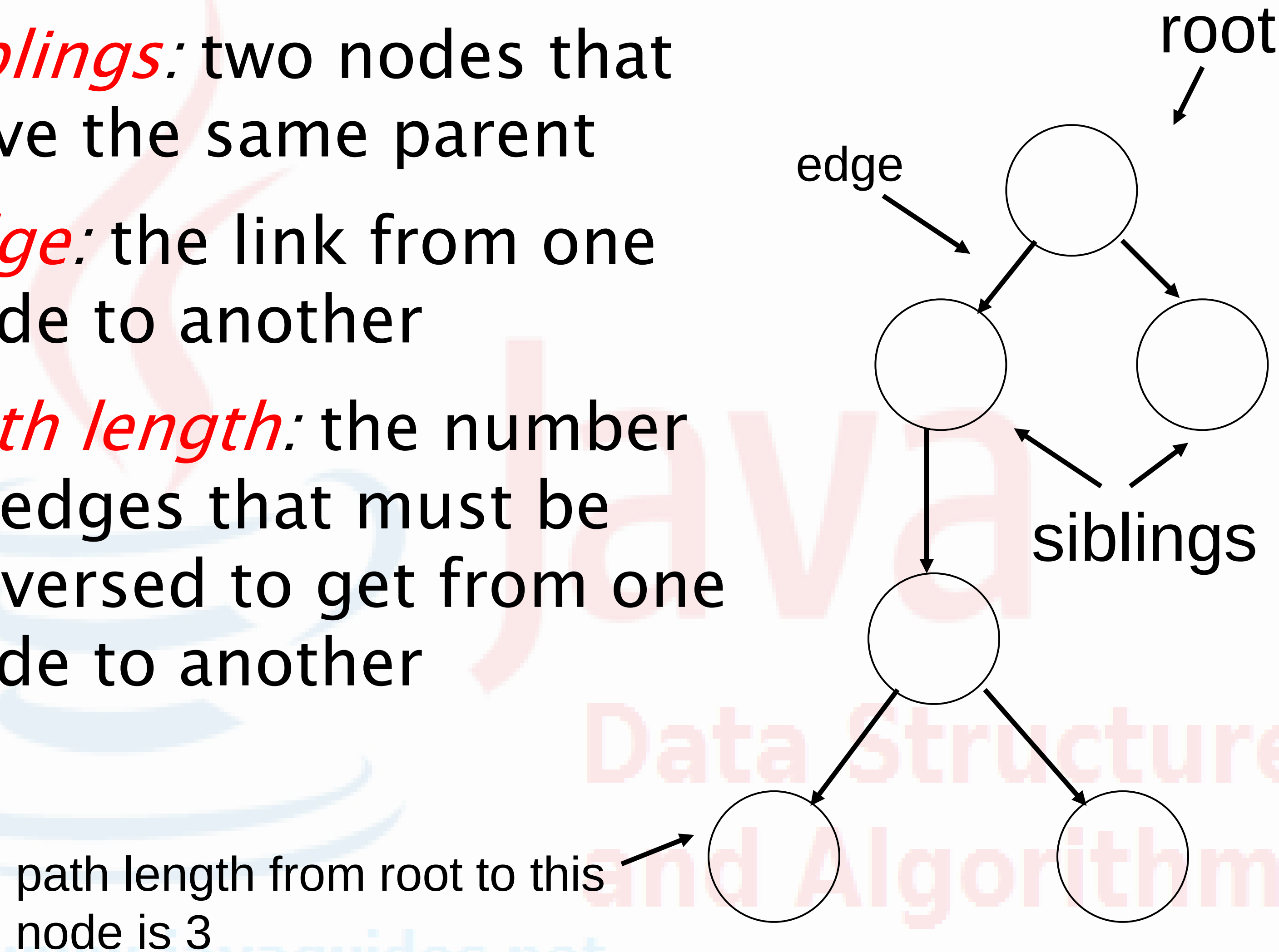
- Only access point is the root
- All nodes, except the root, have one parent
  - like the inheritance hierarchy in Java
- Traditionally trees drawn upside down



Binary Trees

Properties of Trees and Nodes

- *siblings:* two nodes that have the same parent
- *edge:* the link from one node to another
- *path length:* the number of edges that must be traversed to get from one node to another



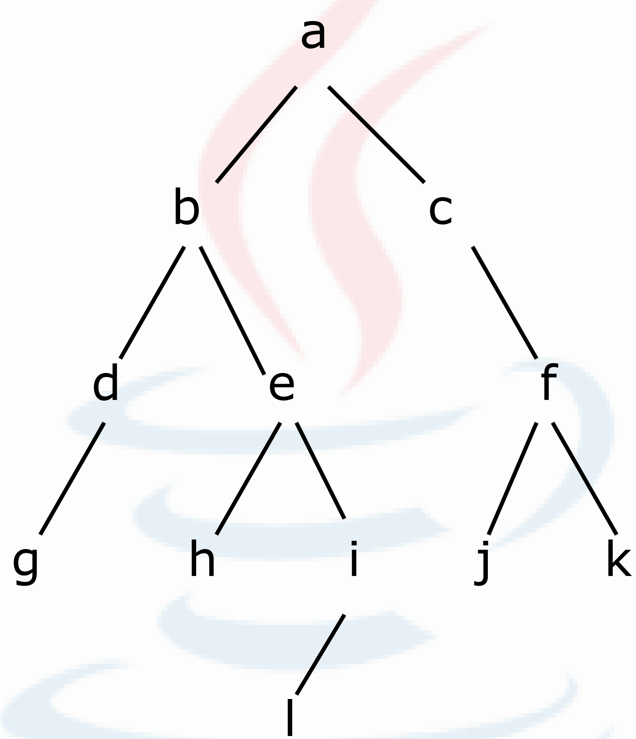
More Properties of Trees

- *depth (or Level)*: the path length from the root of the tree to this node

- *height of a node:* The maximum distance (path length) of any leaf from this node
  - a leaf has a height of 0
  - the height of a tree is the height of the root of that tree

- *size*: the number of elements that are contained in the tree.

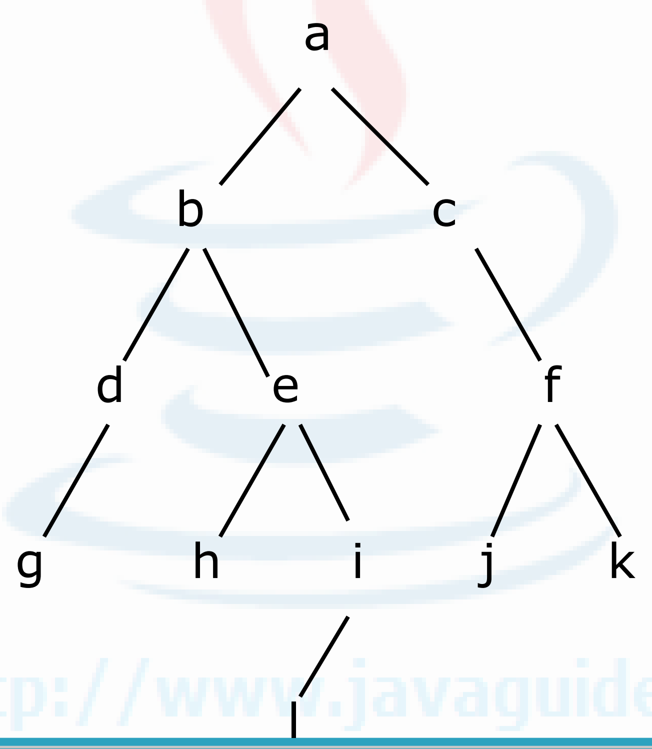
Size and Depth



- The size of a binary tree is the number of nodes in it
  - This tree has size 12
- The depth of a node is its distance from the root
  - a is at depth zero
  - e is at depth 2
- The depth of a binary tree is the depth of its deepest node
  - This tree has depth 4

More Properties of Trees

- *descendants*: any nodes that can be reached via 1 or more edges from this node
- *ancestors:* any nodes for which this node is a descendant



- *Descendants* of e: ???
- *Ancestors* of e: ???

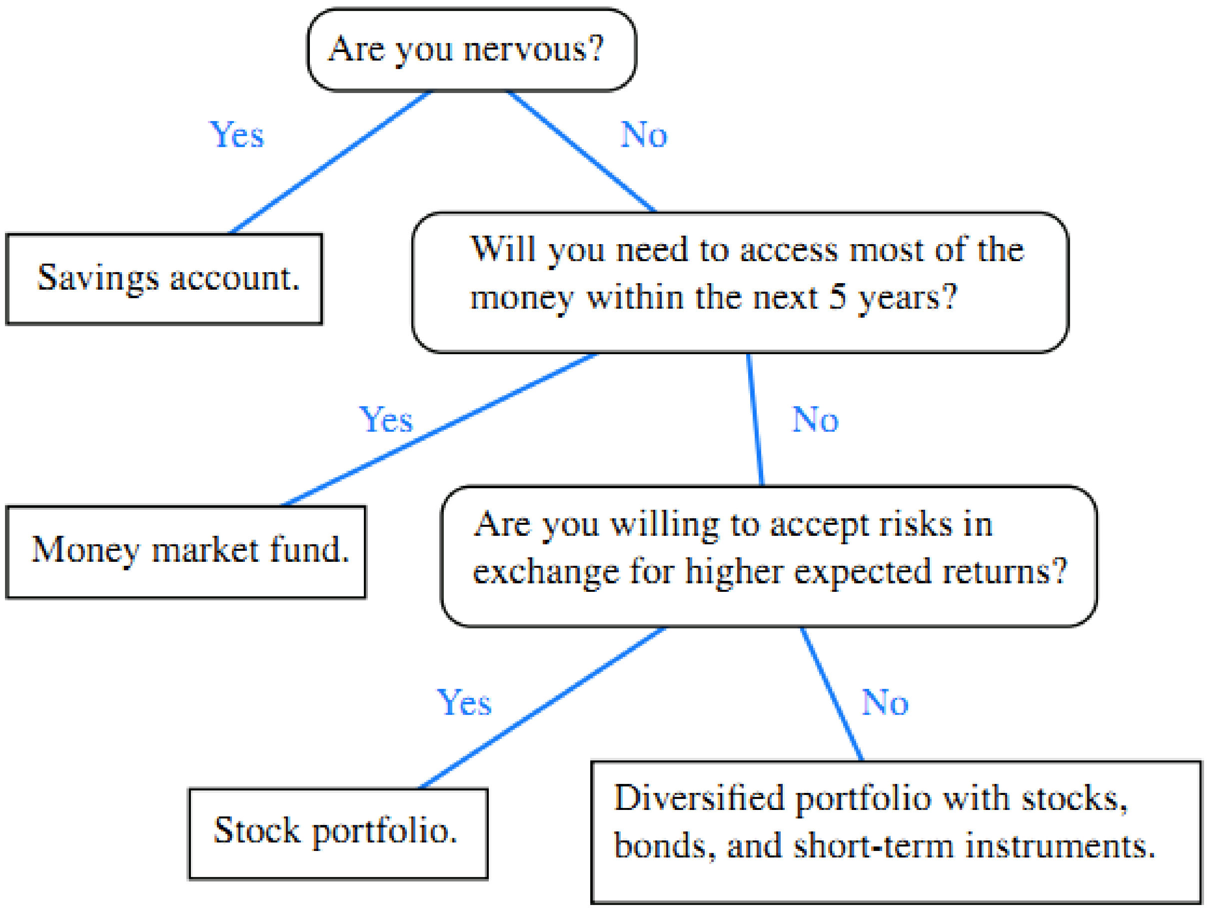
Binary Trees

Definition of Binary Tree

- A binary tree is an ordered tree with the following properties:
  - Every node has **at most two children**.
  - Each child node is labeled as being either a left child or a right child.
  - A left child precedes a right child in the order of children of a node.

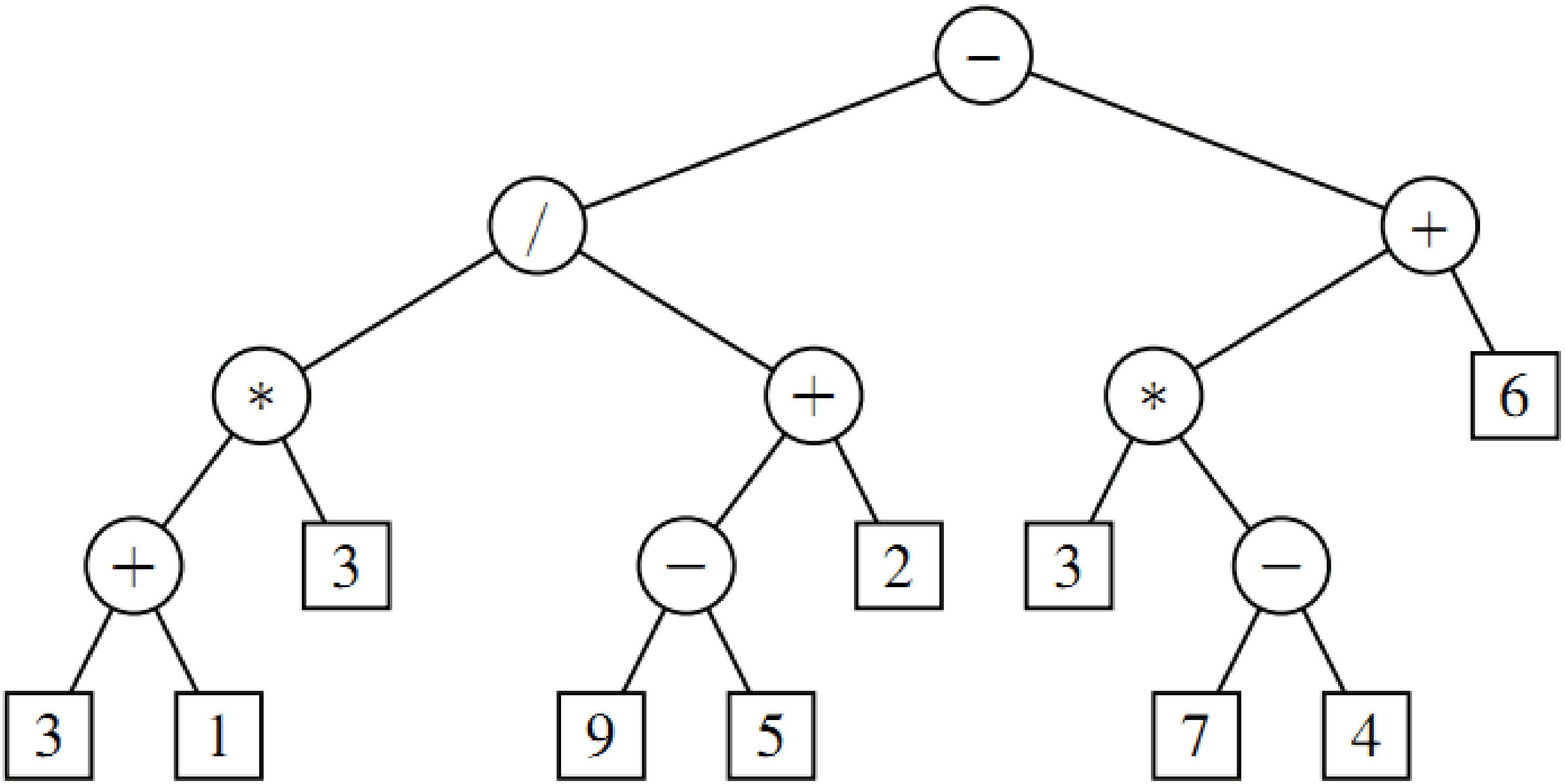
Binary Tree Example

- A decision tree providing investment advice



Binary Tree Example (cont.)

- A binary tree representing an arithmetic expression

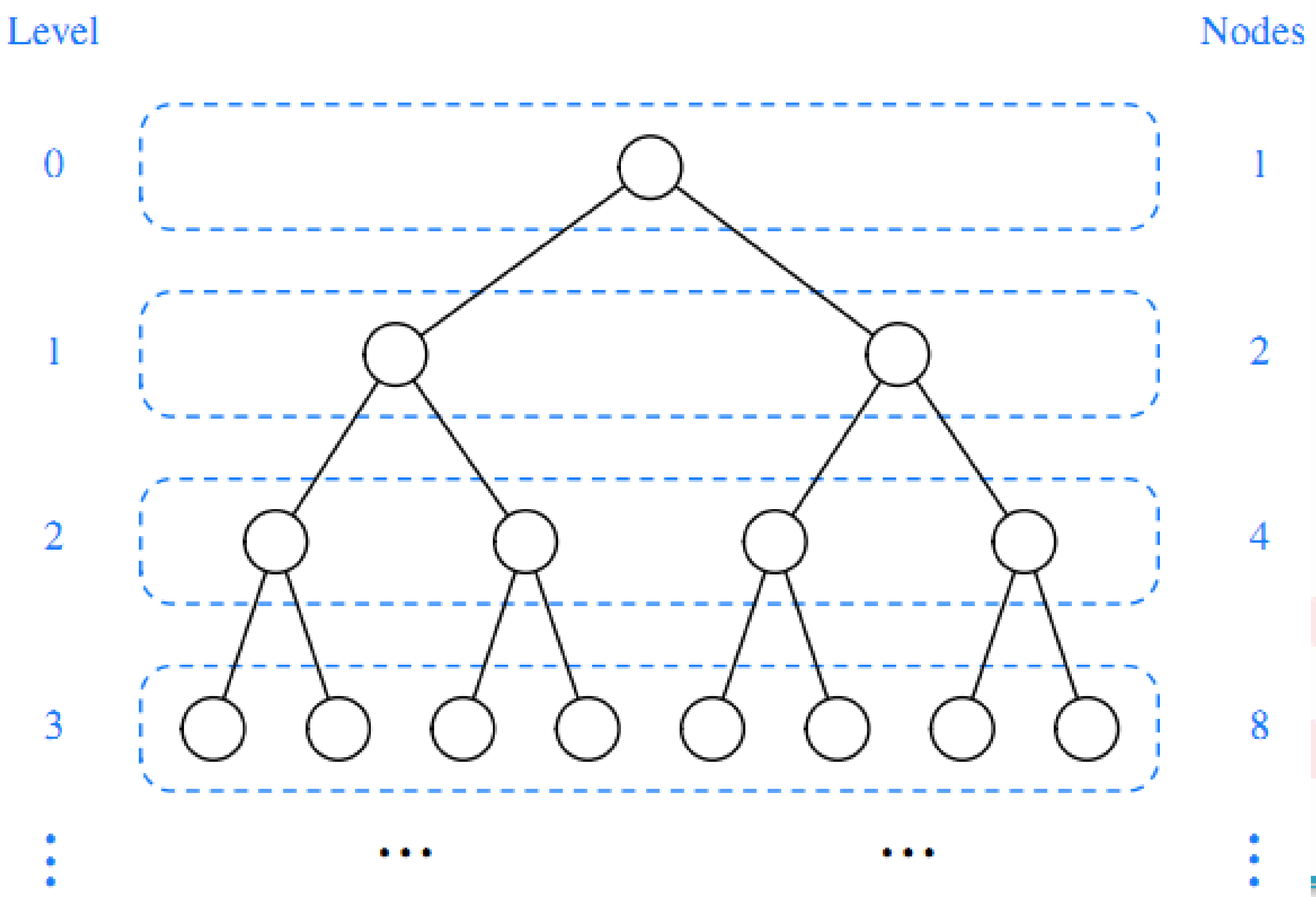


Properties of Binary Trees

- Level (depth) 0 has at most one node (the root),
- Level 1 has at most 2 nodes (the children of the root),
- Level 2 has at most 4 nodes,
- ...
- Level d has at most 2^d nodes.

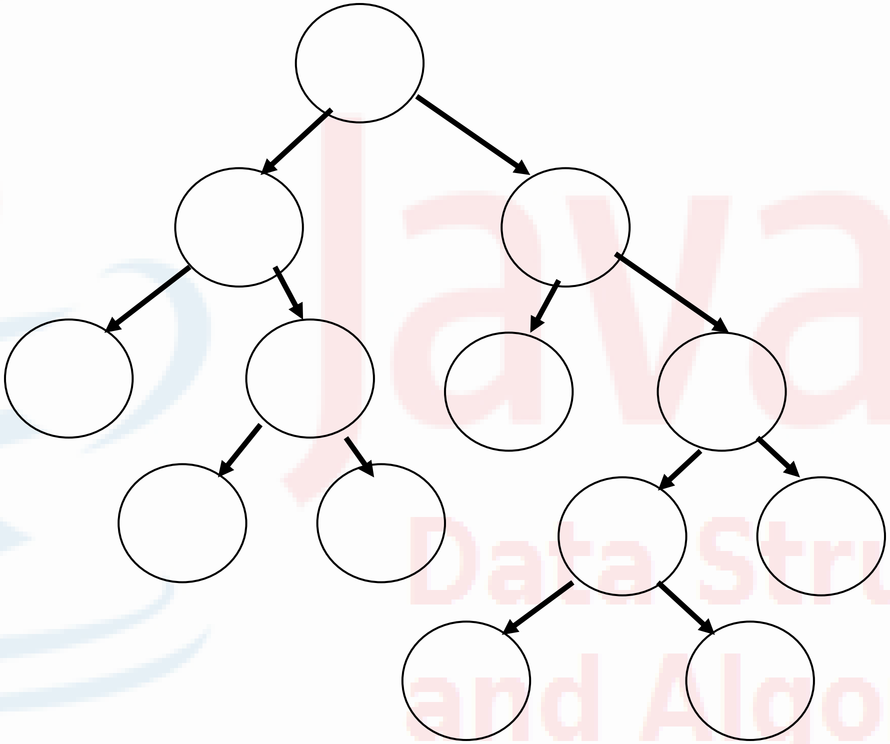
Level and Nodes example

- Maximum number of nodes in the levels of a binary tree



Full Binary Tree

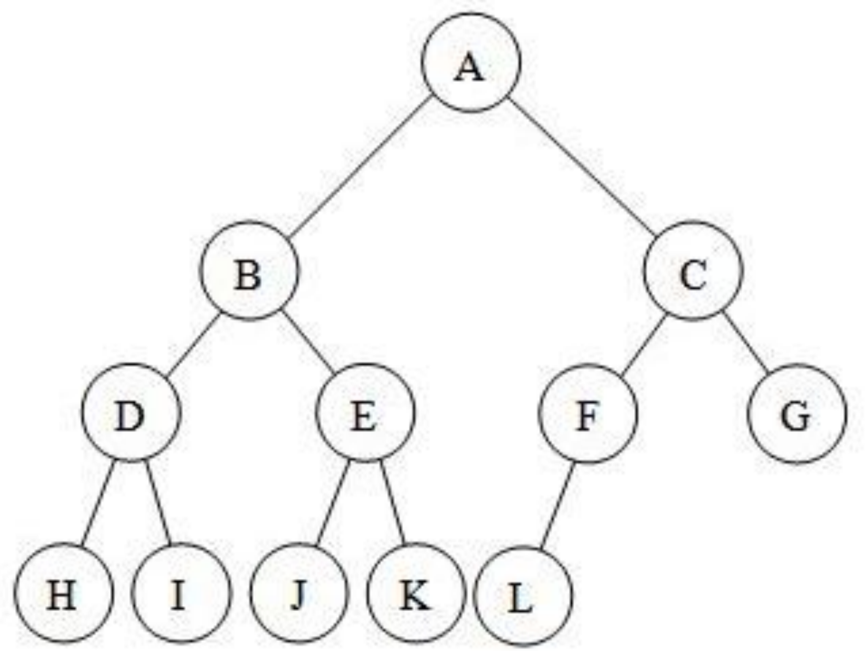
- *Full binary tree:* a binary tree is which each node has exactly 2 or 0 children



CÂY NHỊ PHÂN ĐẦY ĐỦ

Complete Binary Tree

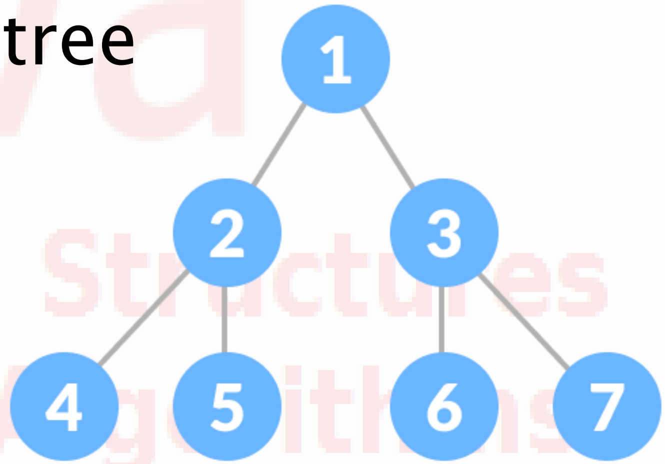
- *Complete binary tree:* all levels are completely filled *except possibly the last level*.
- The incomplete level must have all the leaf nodes left aligned.



CÂY NHỊ PHÂN HOÀN CHỈNH

Perfect Binary Tree

- *Perfect binary tree:* a binary tree with all leaf nodes at the same depth. All internal nodes have exactly two children.
- A perfect binary tree has the maximum number of nodes for a given height
- A perfect binary tree has $(2^{(n+1)} - 1)$ nodes where n is the height of the tree
  - height = 0 -> 1 node
  - height = 1 -> 3 nodes
  - height = 2 -> 7 nodes
  - height = 3 -> 15 nodes



CÂY NHỊ PHÂN HOÀN HẢO

A Binary Node class

```java
package lec10;

public class BNode<E> {
	private E data;
	private BNode<E> myLeft;
	private BNode<E> myRight;

	public BNode(E data) {
		this.data = data;
	}

	public BNode(E data, BNode<E> left, BNode<E> right) {
		this.data = data;
		this.myLeft = left;
		this.myRight = right;
	}
}
```

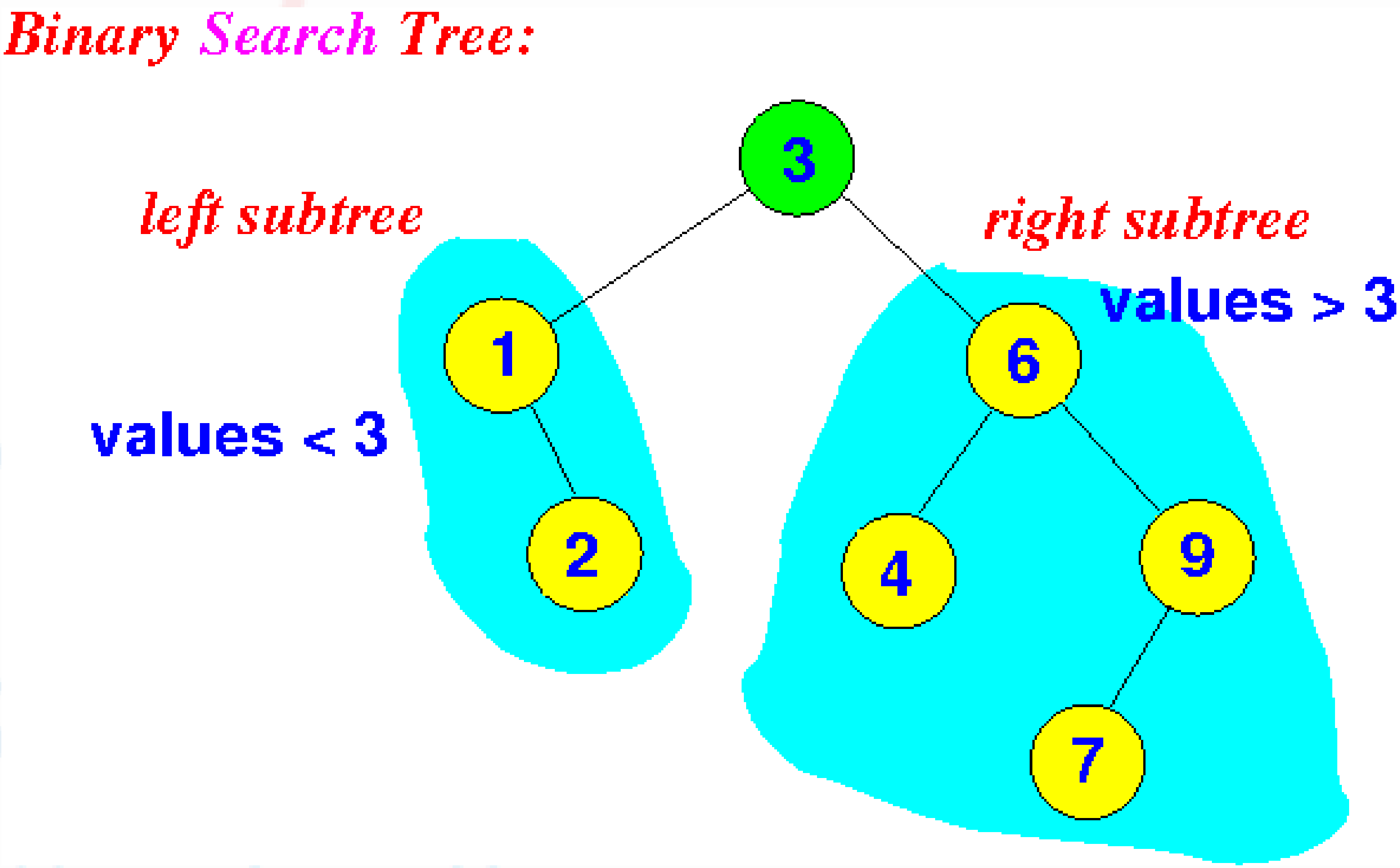
Binary *Search* Tree

Definition of Binary Search Tree

- Binary Search Tree (BST): *organizing* the nodes in a binary tree

- **BST** = a **binary tree** where:
  - The **values** in the **nodes** in the *left* **subtree** of the node *x* in the tree has a *smaller* **value** than *x*
  - The **values** in the **nodes** in the *right* **subtree** of the node *x* in the tree has a *greater* **value** than *x*

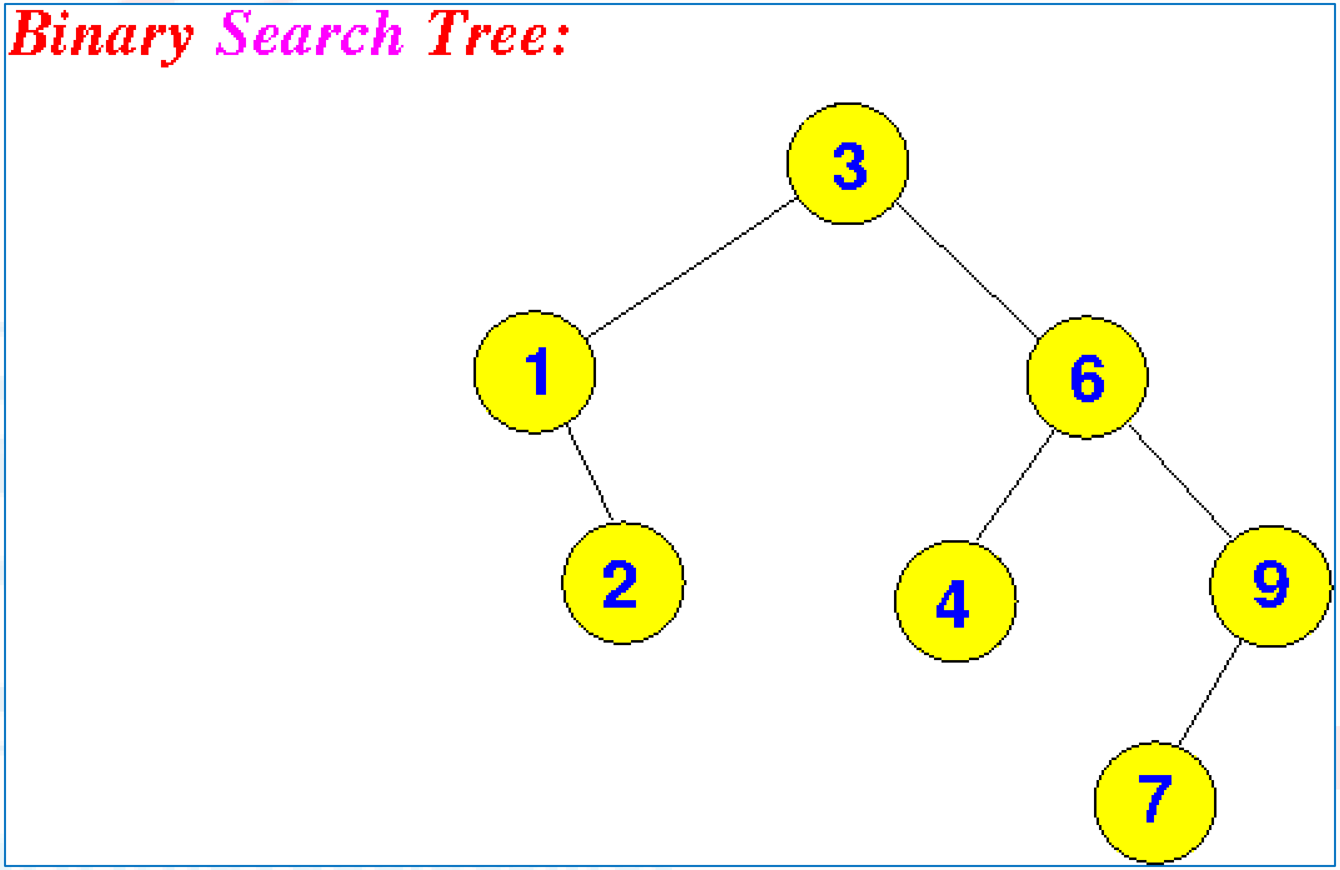
BST Example



Binary Search Tree:
left subtree
right subtree
values > 3
values < 3
3
1
2
6
4
9
7

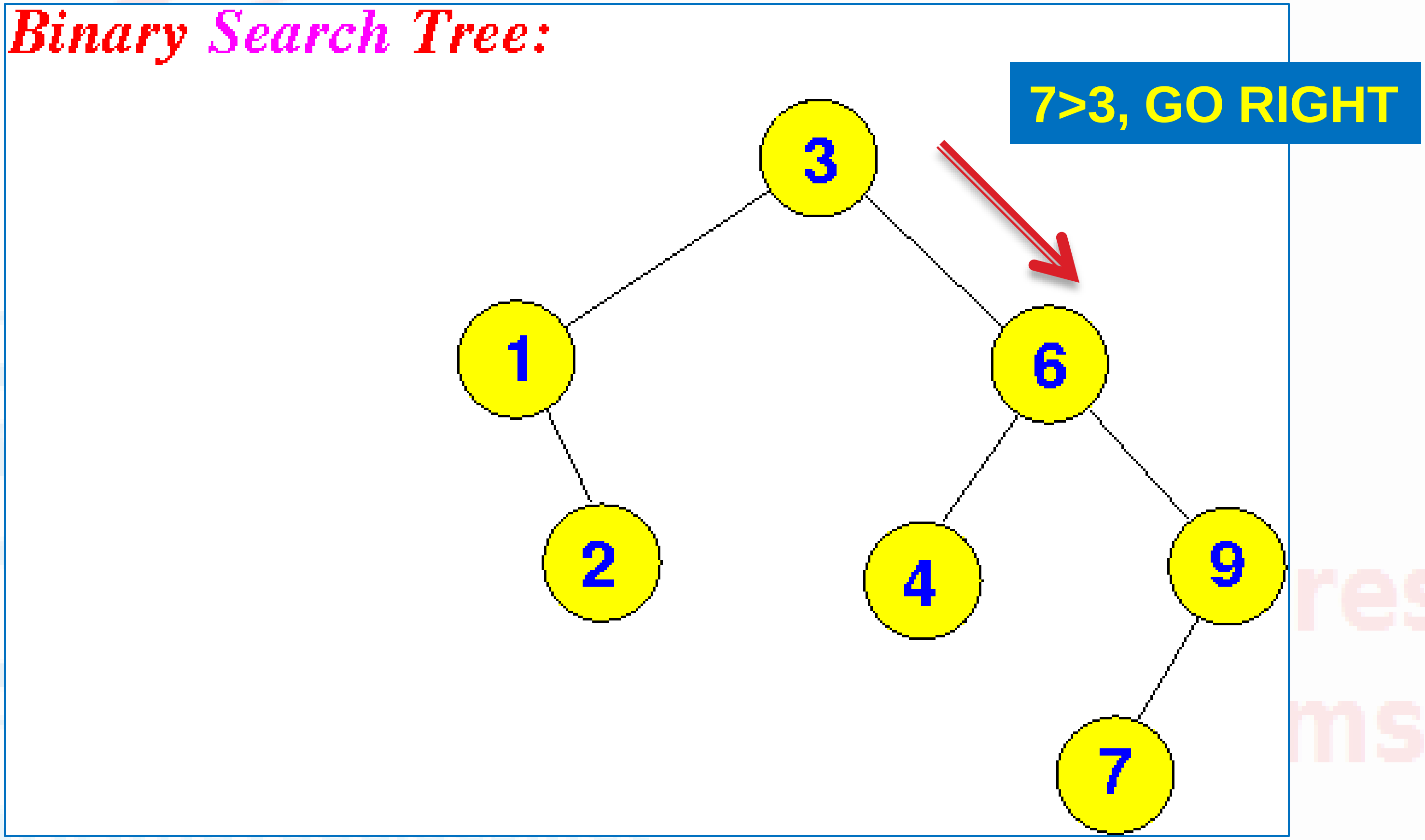
How to search a *specific* value in BST

- **Search** for the **node** with the value 7 in the following **BST**:



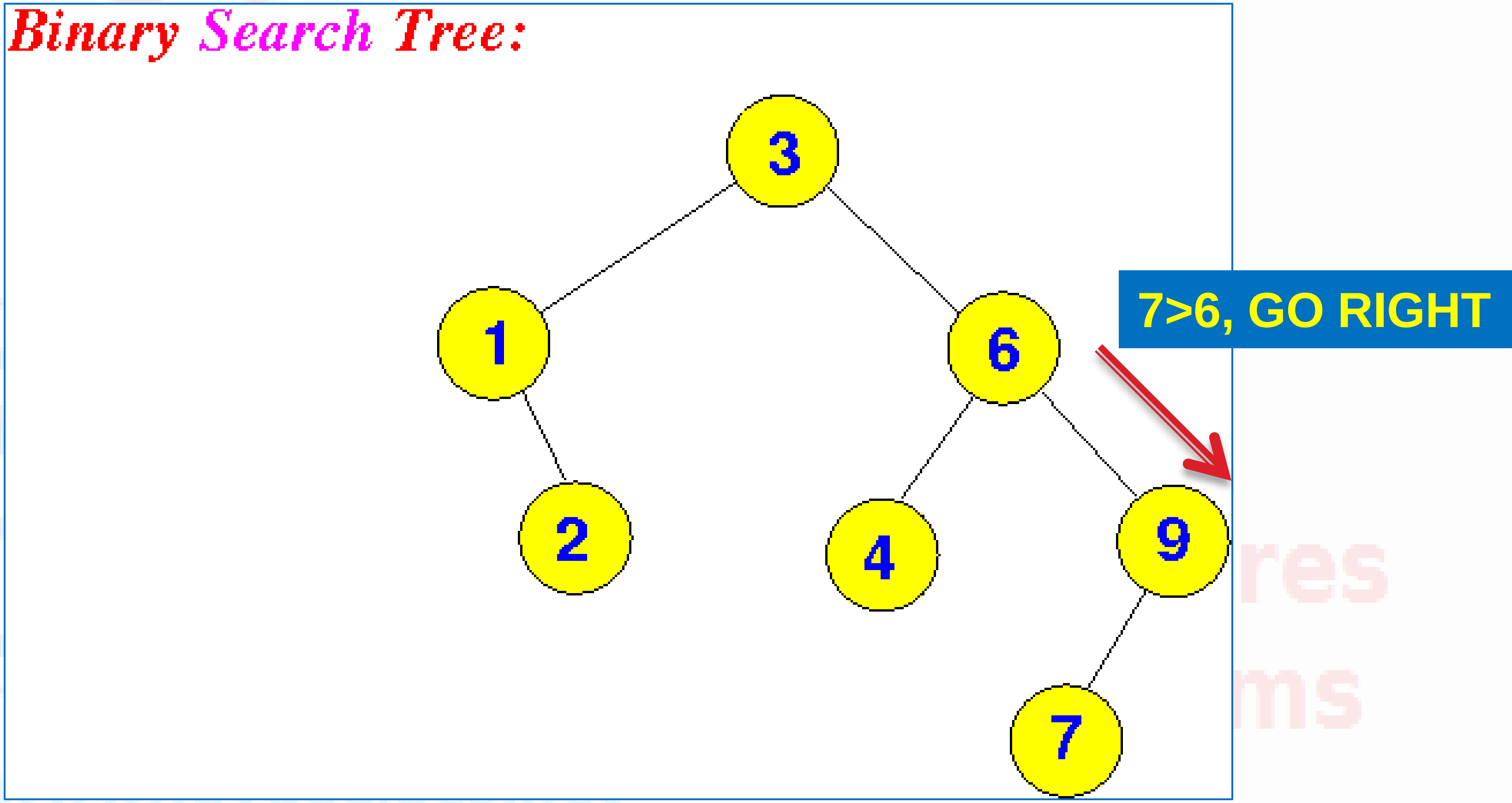
How to search a *specific* value in BST

- **Search** for the **node** with the value 7 in the following **BST**:



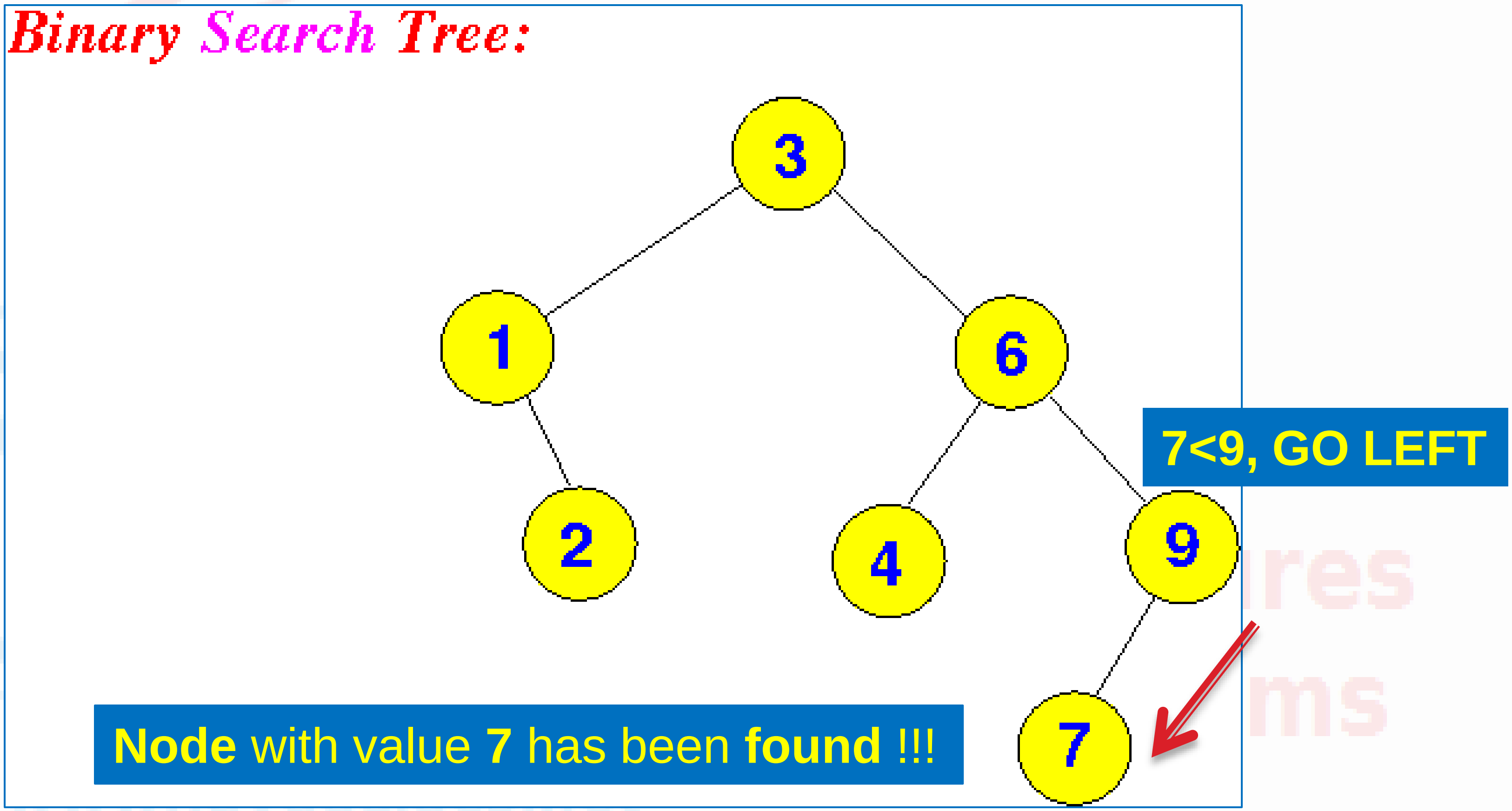
How to search a *specific* value in BST

- **Search** for the **node** with the value 7 in the following **BST**:



How to search a *specific* value in BST

- **Search** for the **node** with the value 7 in the following **BST**:



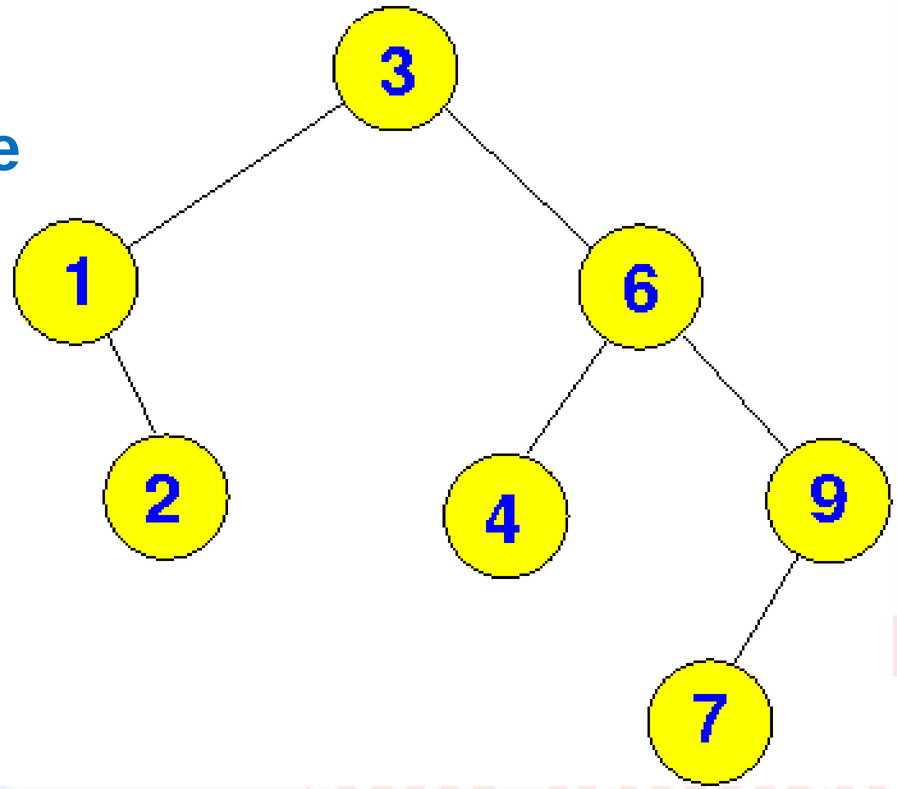
Search in BST - Pseudocode

```
if the tree is empty
        return NULL/FALSE

else if  the item in the node equals the target
        return the node value

else if  the item in the node is greater than the target
        return the result of searching the left subtree

else if  the item in the node is smaller than the target
        return the result of searching the right subtree
```

How to find the minimum value in a BST

Binary Search Tree:

Minimum value

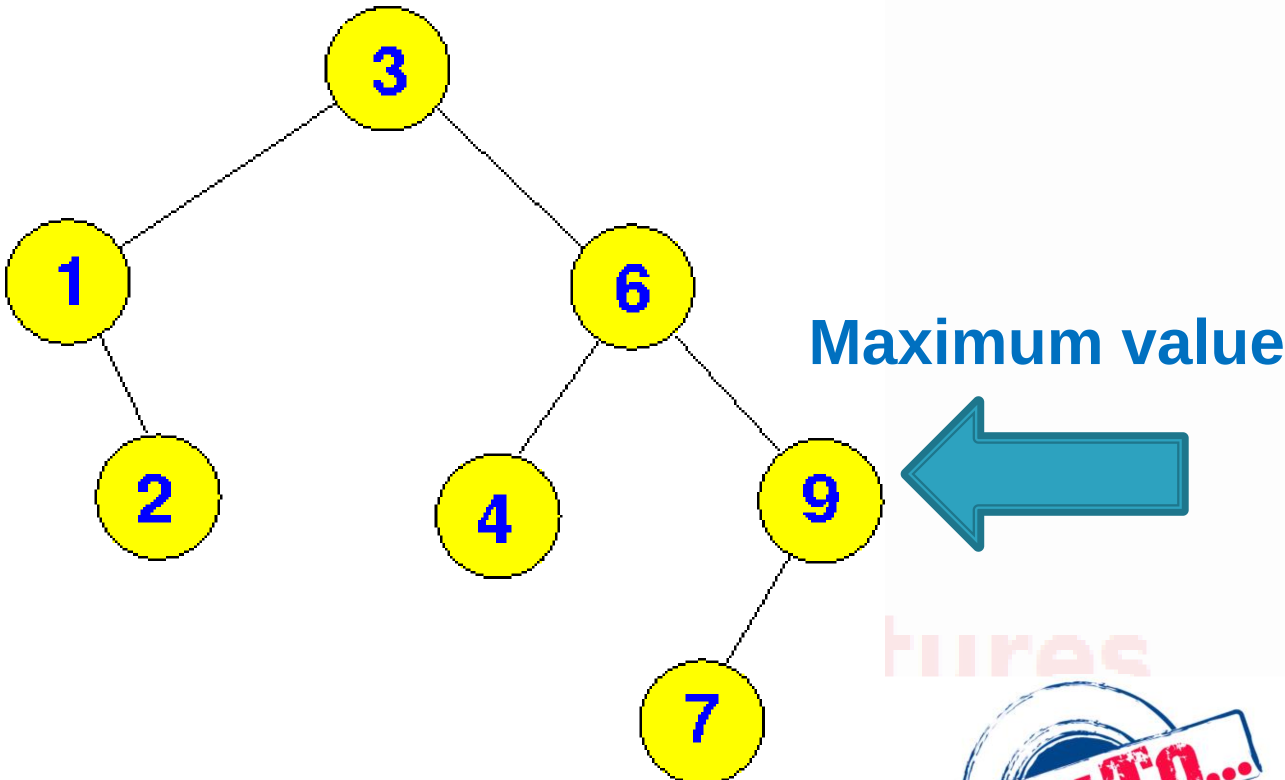


How to find the minimum value in a BST

- The *minimum* value in a Binary Search Tree can be **found** by:
  - Start at the root node
  - Follow the *left* child in *each* branch until you reach a node that does *not* have a *left* child
    - The value at *that* node is the *minimum* value in the Binary Subtree

How to find the maximum value in a BST

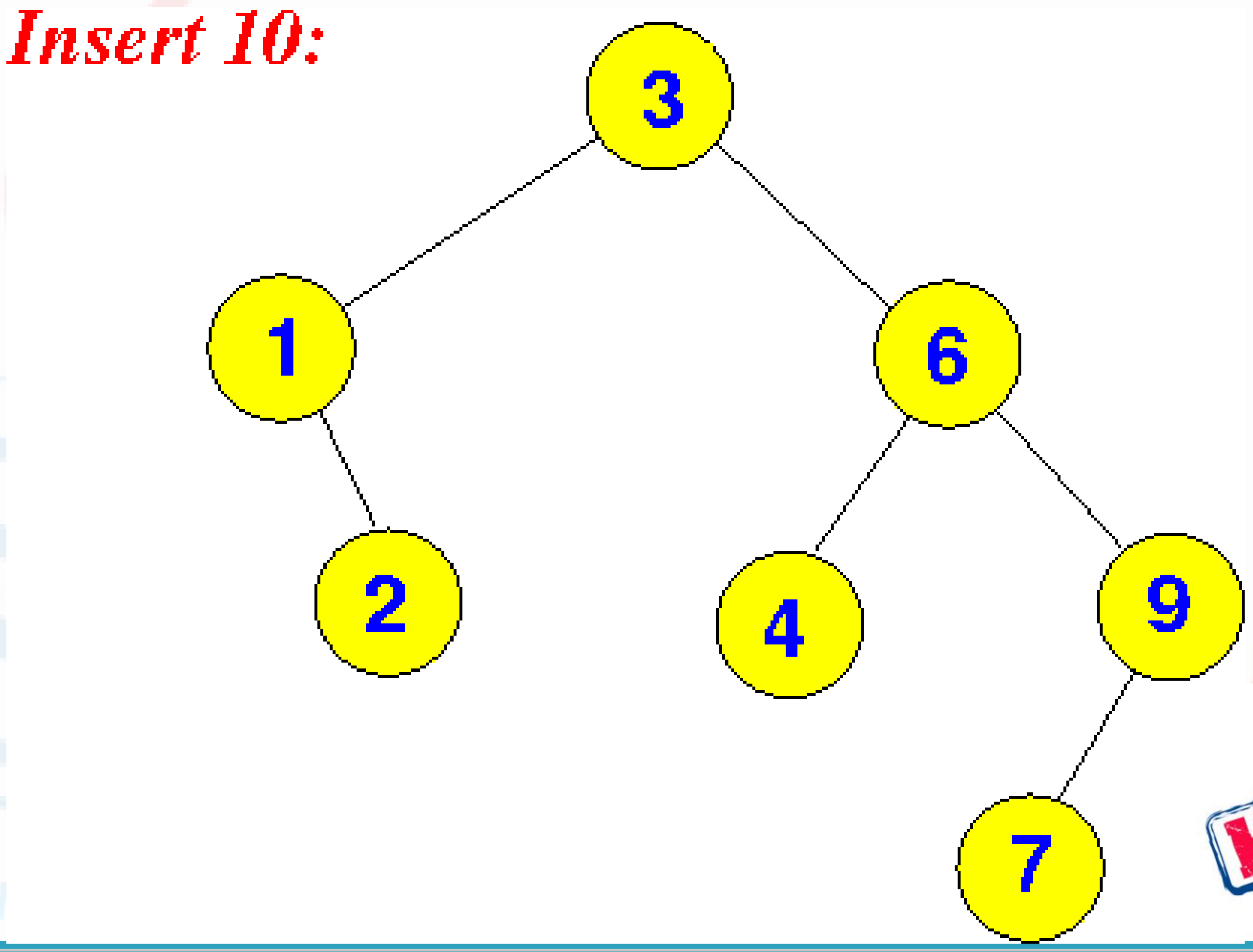
Binary Search Tree:



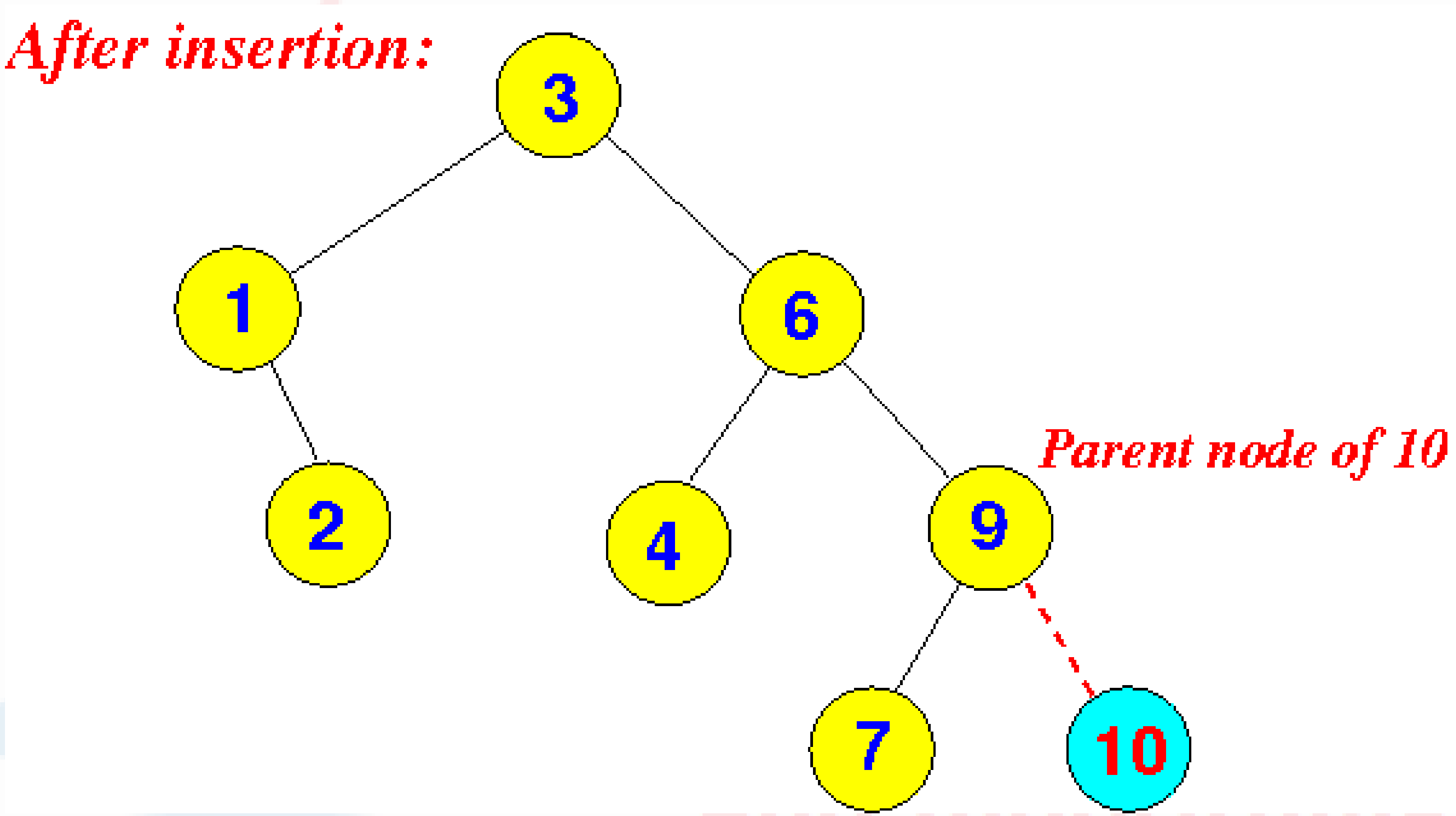
How to find the maximum value in a BST

- The *maximum* **value** in a **Binary Search Tree** can be found by:
  - **Start** at the **root** node
  - **Follow** the ***right*** **child** in ***each*** **branch** until you reach a **node** that **does** ***not*** have a ***right*** **child**
    - The **value** at ***that*** **node** is the ***maximum*** **value** in the **Binary Subtree**

How to insert a node into a BST?



How to insert a node into a BST?



Insertion in BST - Pseudocode

if tree is empty
    create a root node with the data
else
    compare data with the top node
    if **data = node data**
     replace the node with the new value
    else if **data > node data**
     compare **data** with the right subtree:
       if subtree is empty create a leaf node
       else add key in right subtree
     else **data < node data**
     compare **data** with the left subtree:
       if the subtree is empty create a leaf node
       else add data to the left subtree

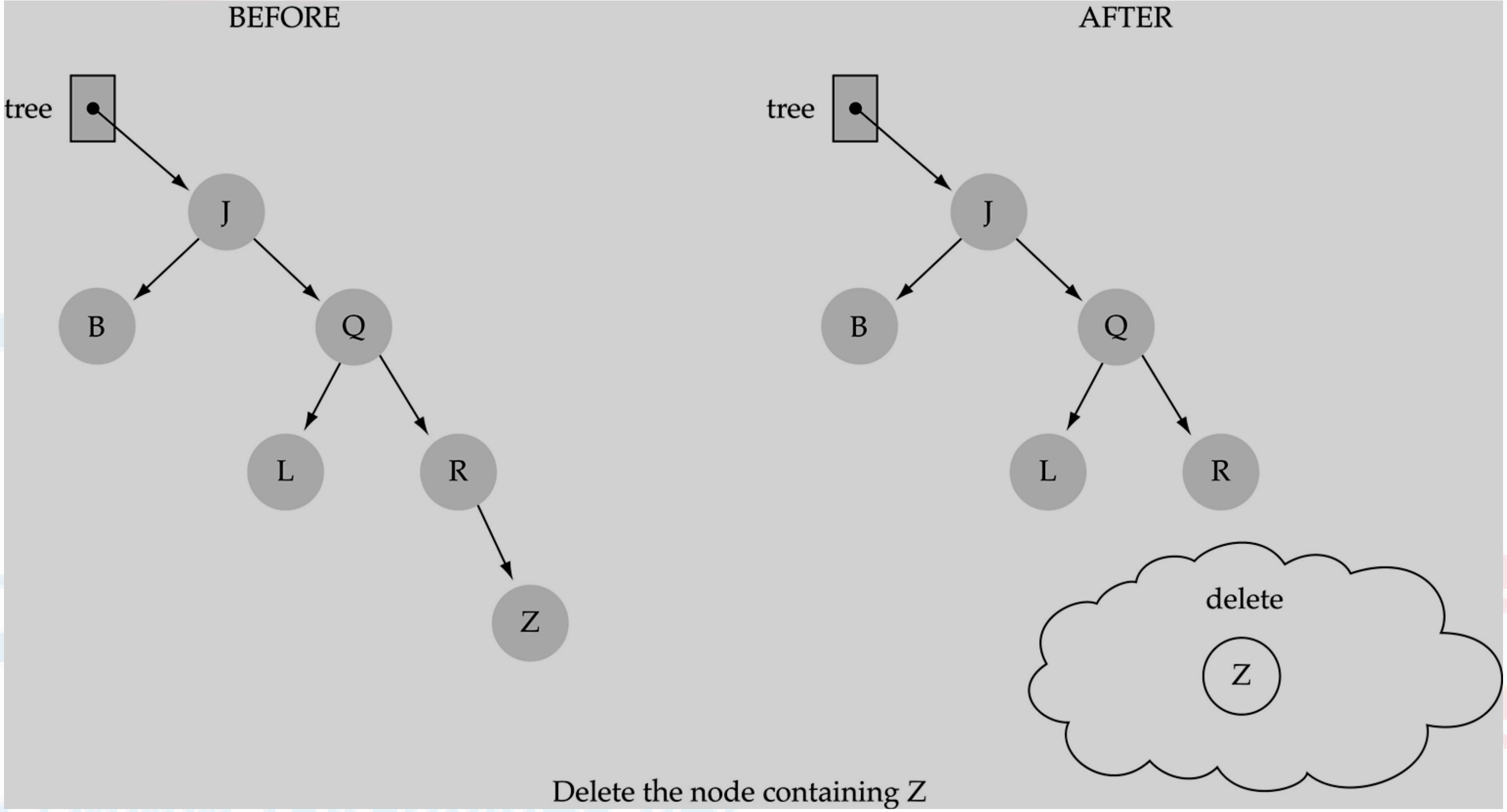
How to delete a node from a BST?

- Consider three different cases:
  - (1) Deleting a leaf
  - (2) Deleting a node with only one child
  - (3) Deleting a node with two children

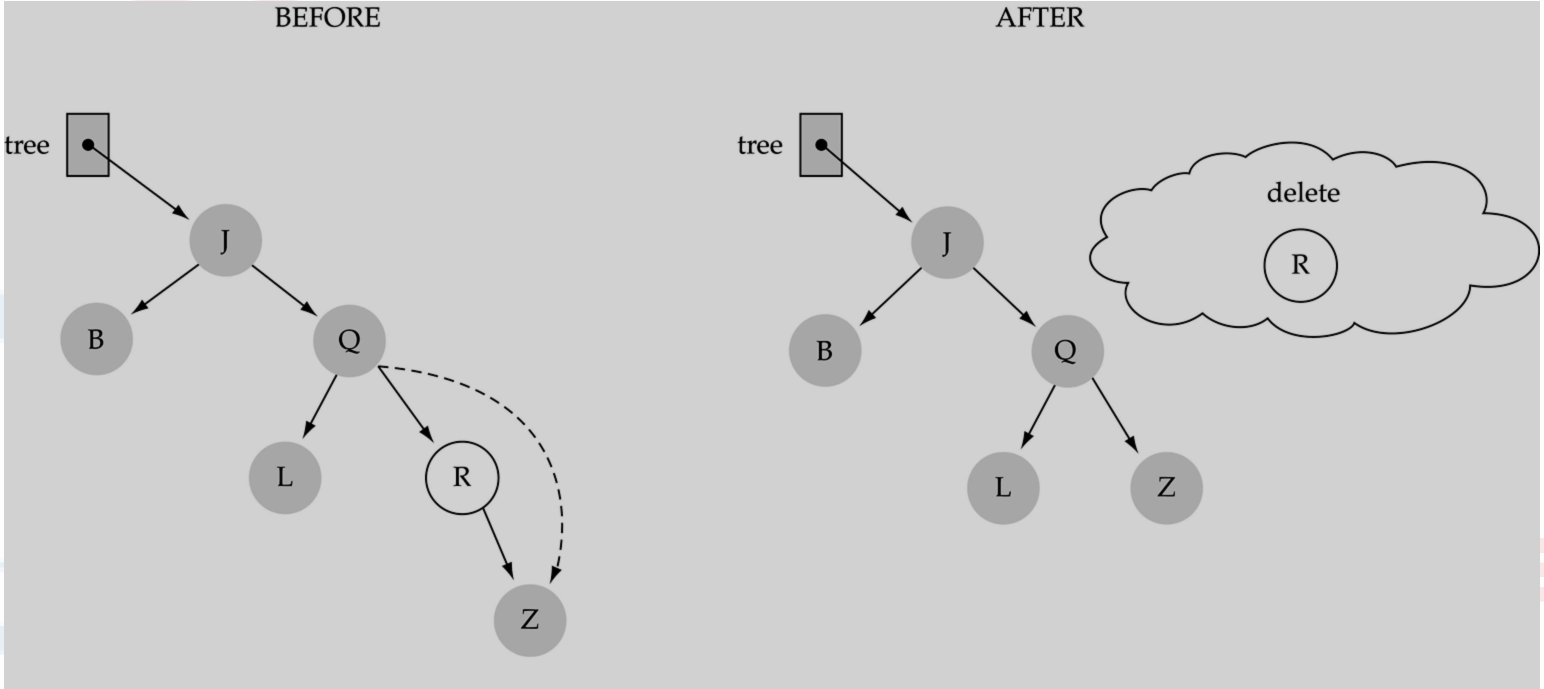
How to delete a node from a BST?

- **Deleting a leaf**:
  - Deleting a node with no children is easy, as we can simply remove it from the tree.
- Deleting a node with one child:
  - Delete it and replace it with its child.
- Deleting a node with two children:
  - Suppose the node to be deleted is called N. We replace the value of N with:
    - either its in-order successor (**the left-most child of the right subtree**)
    - or the in-order predecessor (**the right-most child of the left subtree**).

(1) Deleting a leaf

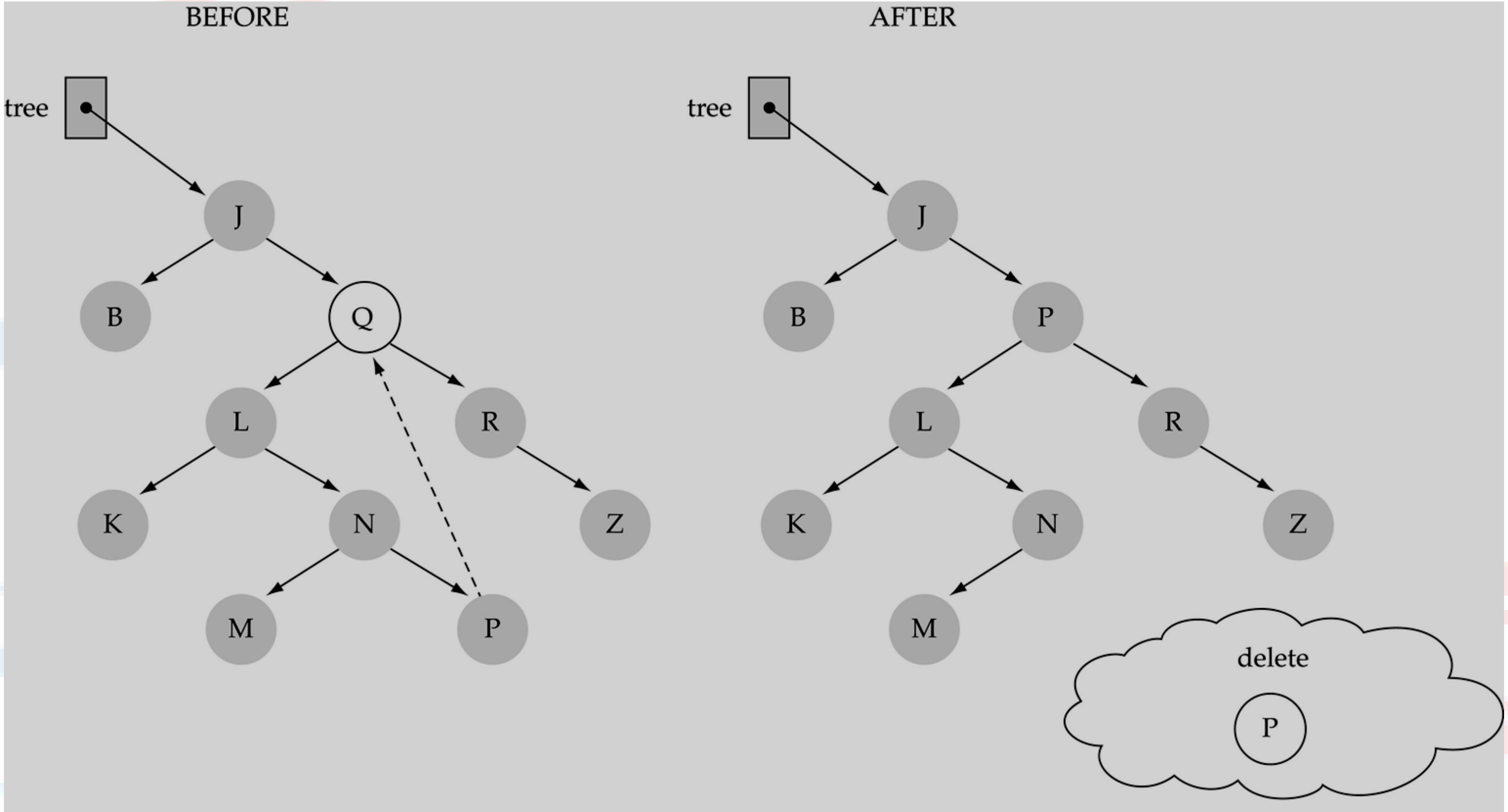


(2) Deleting a node with only one child



Delete the node containing R

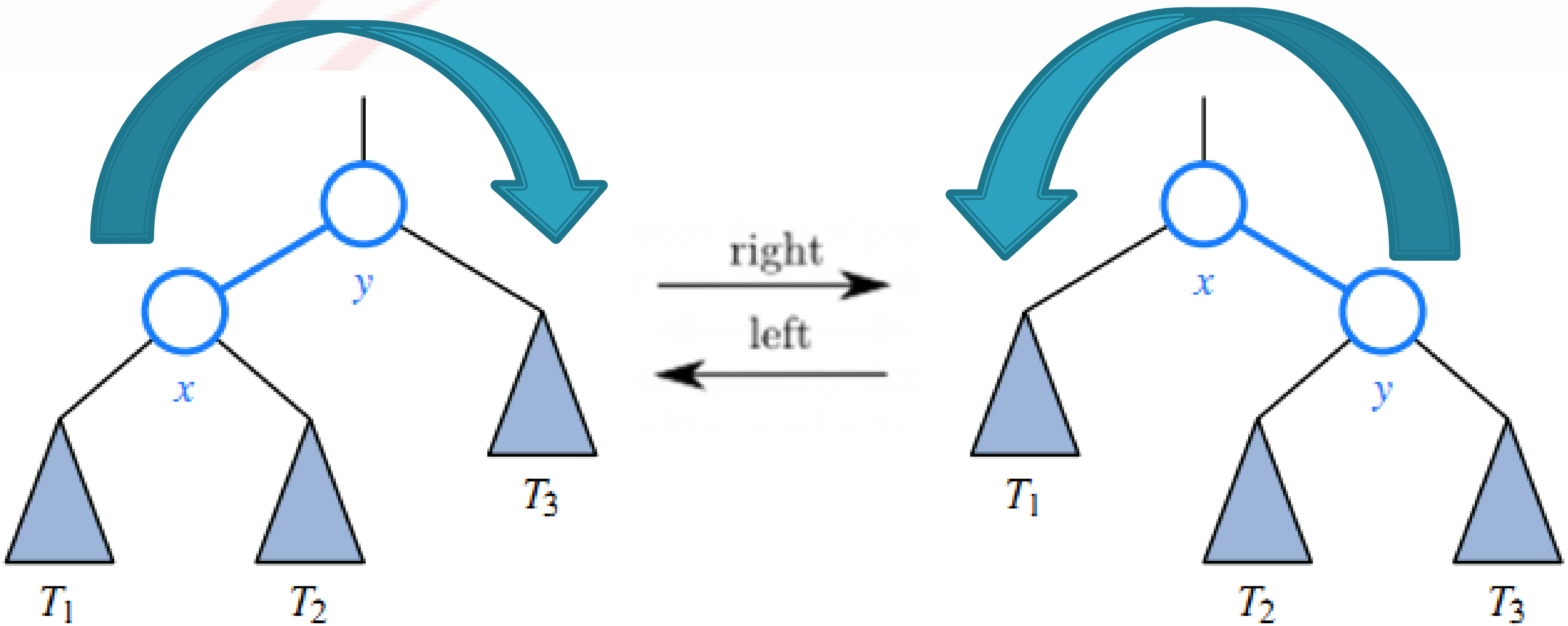
(3) Deleting a node with two children



Delete the node containing Q

Balancing Rotations in Binary Search Trees

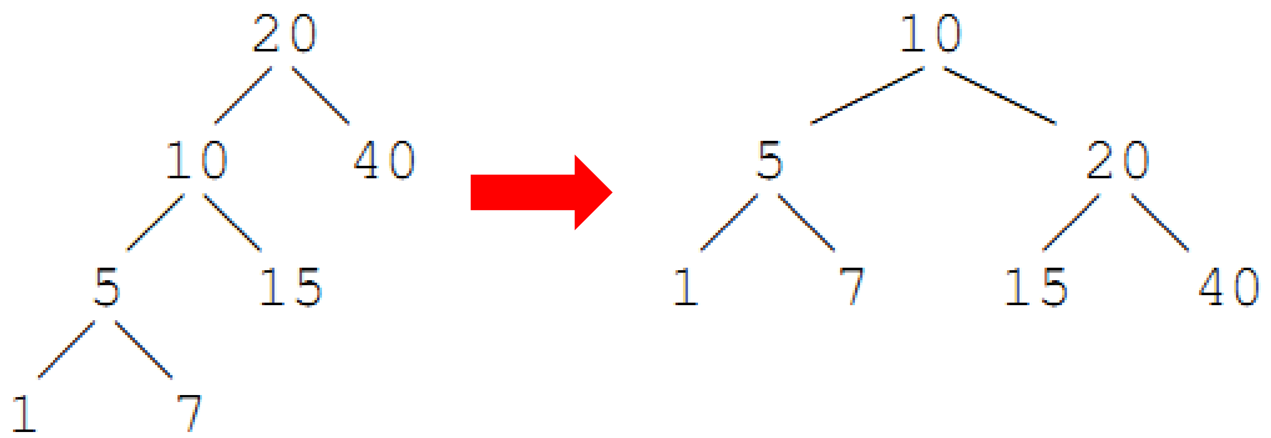
Right rotation **Left rotation**



SINGLE ROTATION

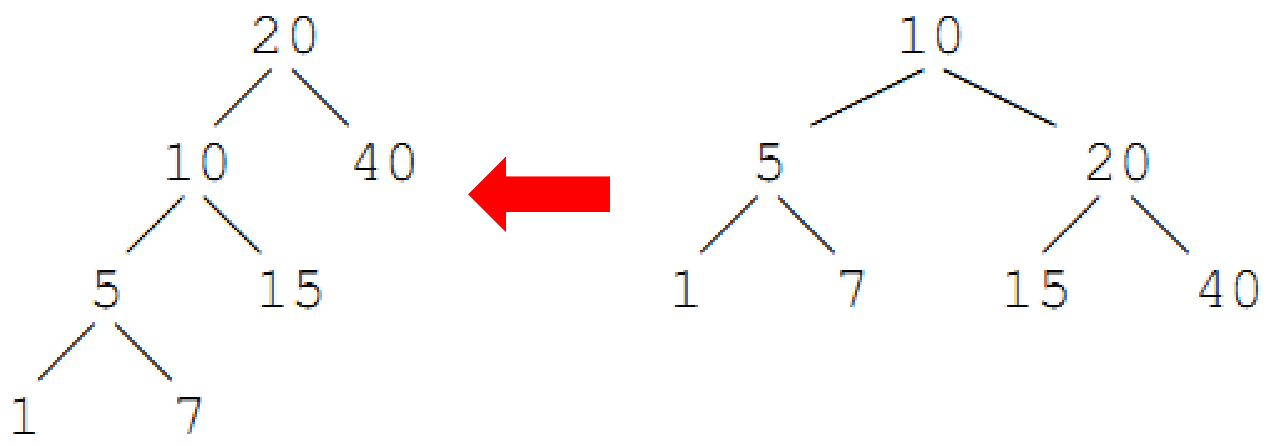
Right Rotation Example

- To balance the binary search tree, we do a right rotate around the root



Left Rotation Example

- To balance the binary search tree, we do a right rotate around the root



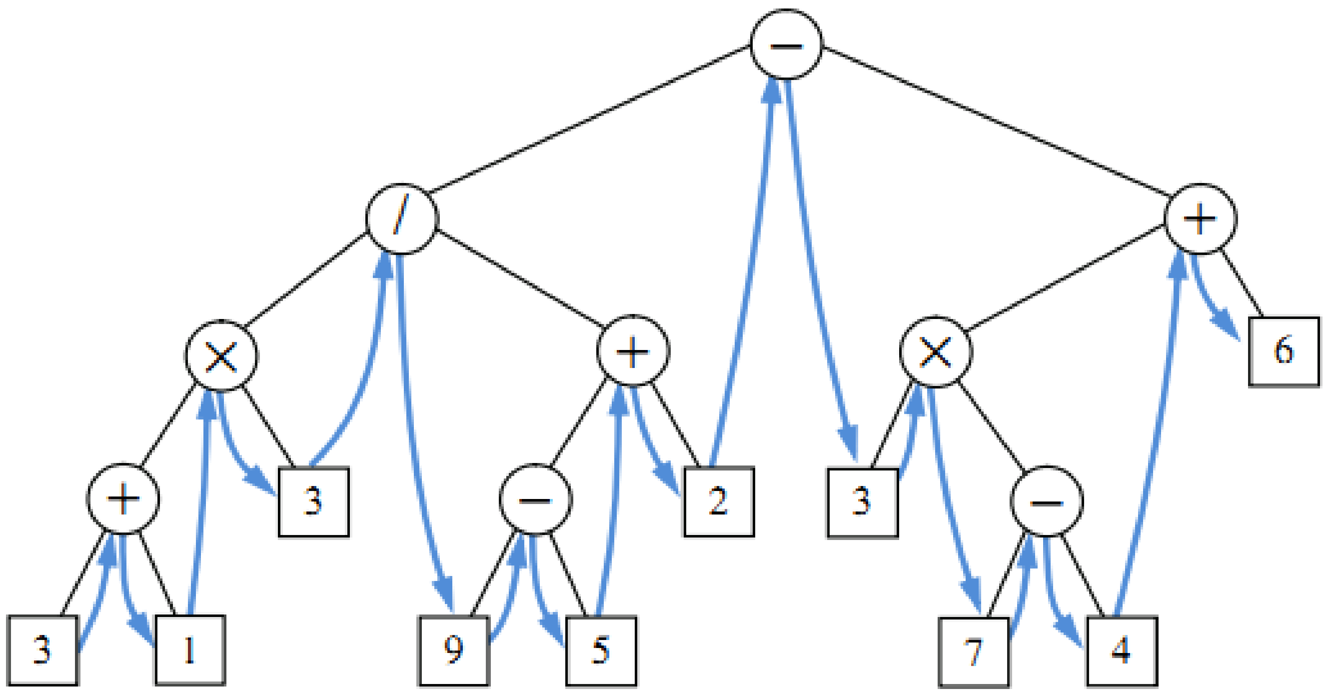
Tree Traversal Algorithms

Tree Traversal Algorithms

- Visiting each node in a specified order.
- Three simple ways to traverse a tree:
  - Inorder
  - Preorder
  - Postorder

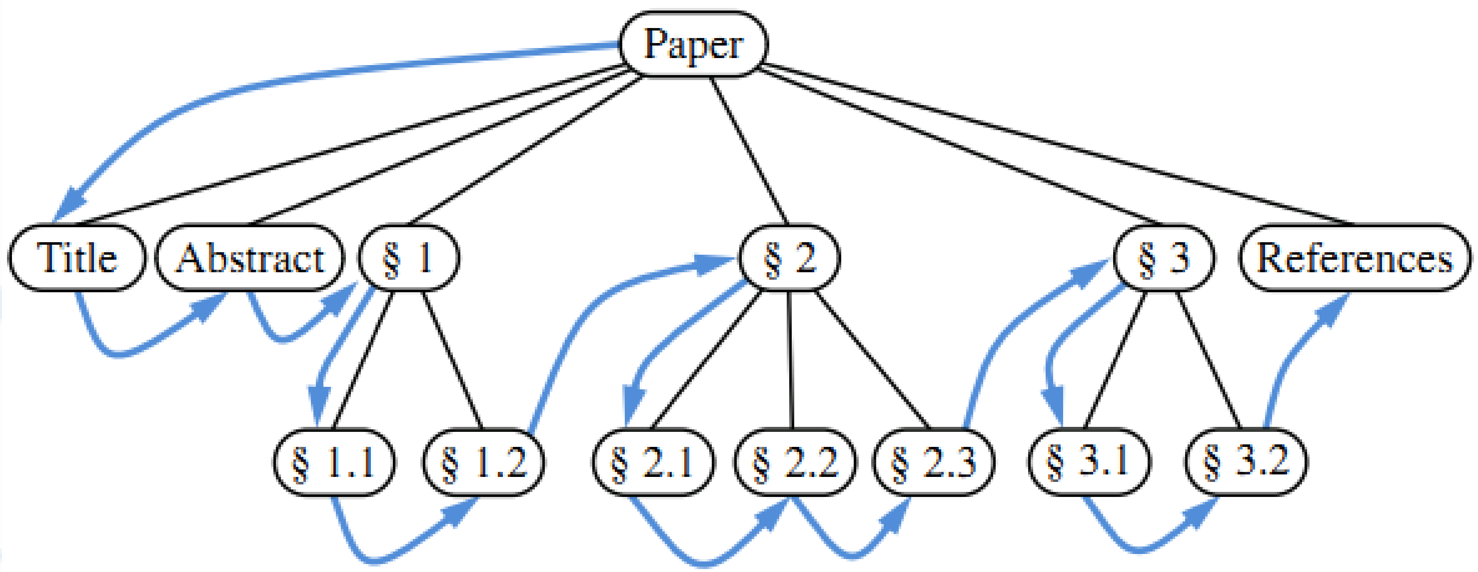
Inorder

- We visit the left subtree, then visit the node, then visit the right subtree
- Inorder traversal of a binary tree



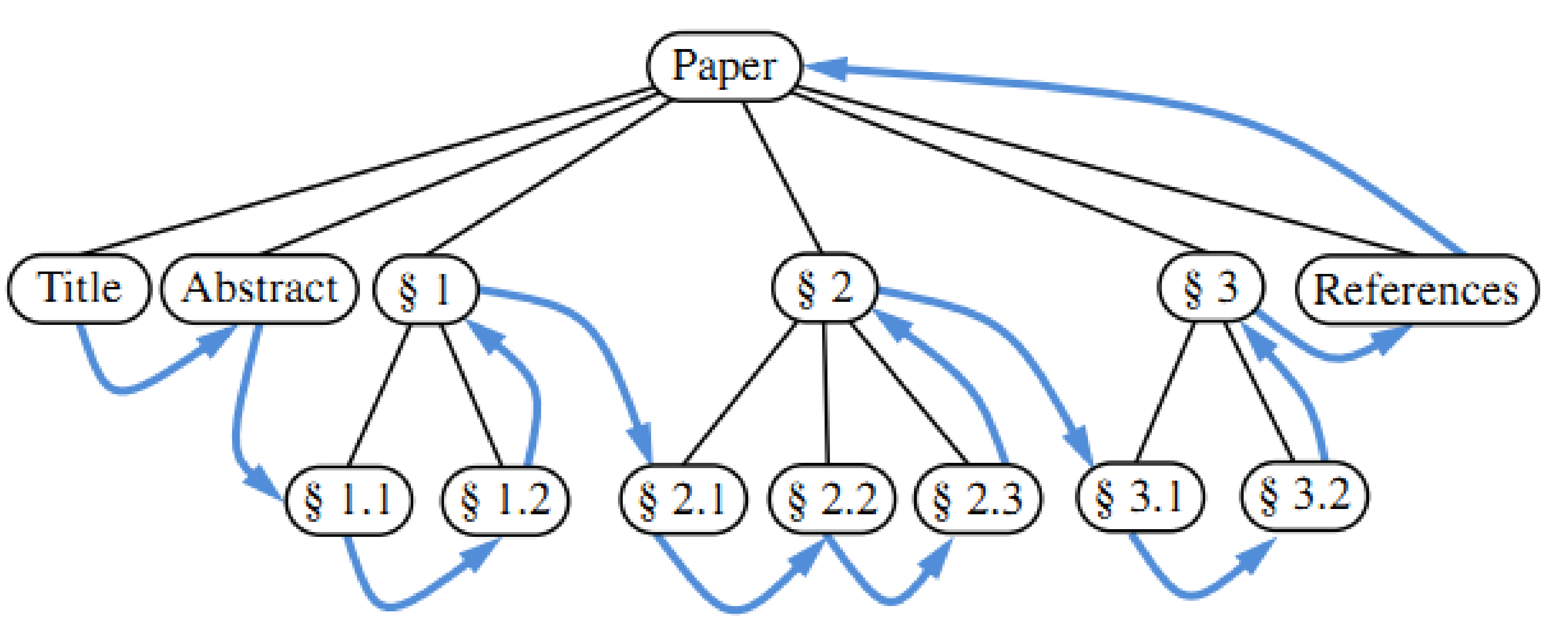
Preorder

- We visit the node, then visit the left and right subtrees



Postorder

- We visit the left and right subtrees, then visit the node



Tree Traversals: another example

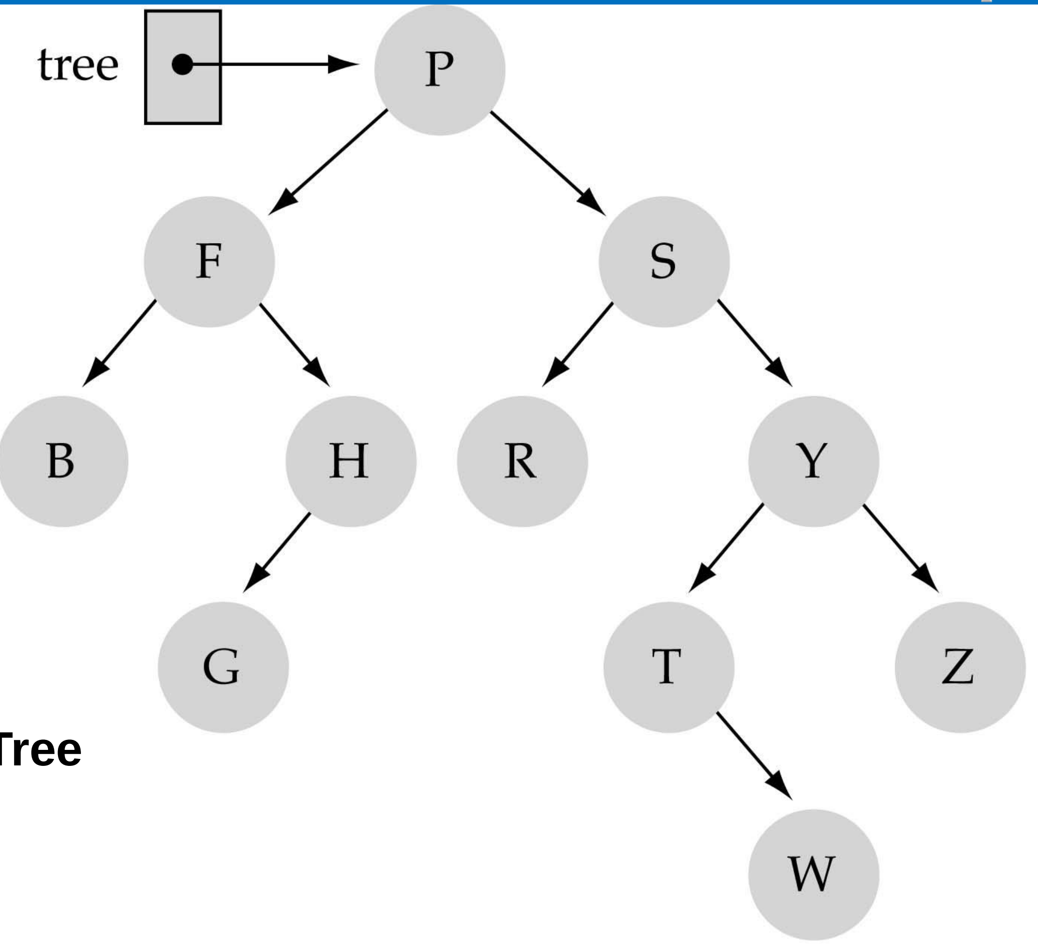


Fig. Binary Search Tree

Inorder:
Preorder:
Postorder:

Tree Traversals

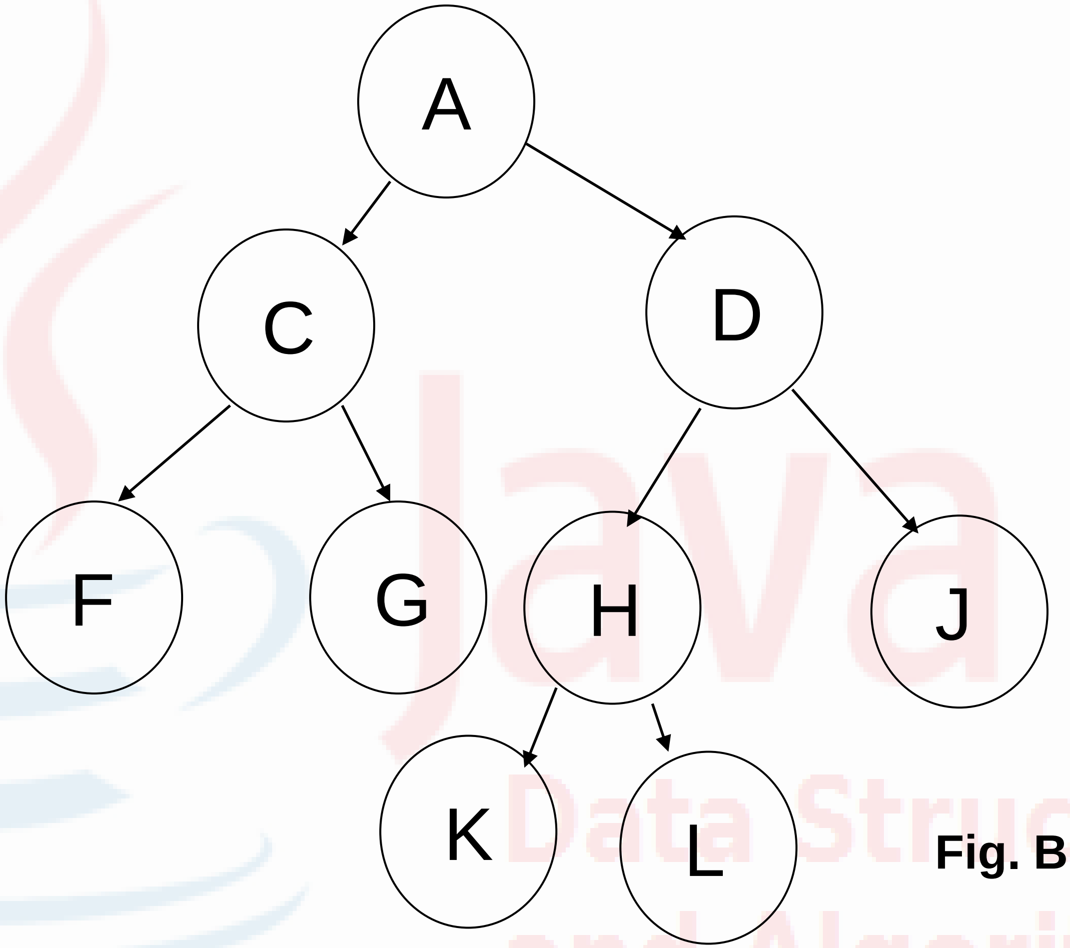


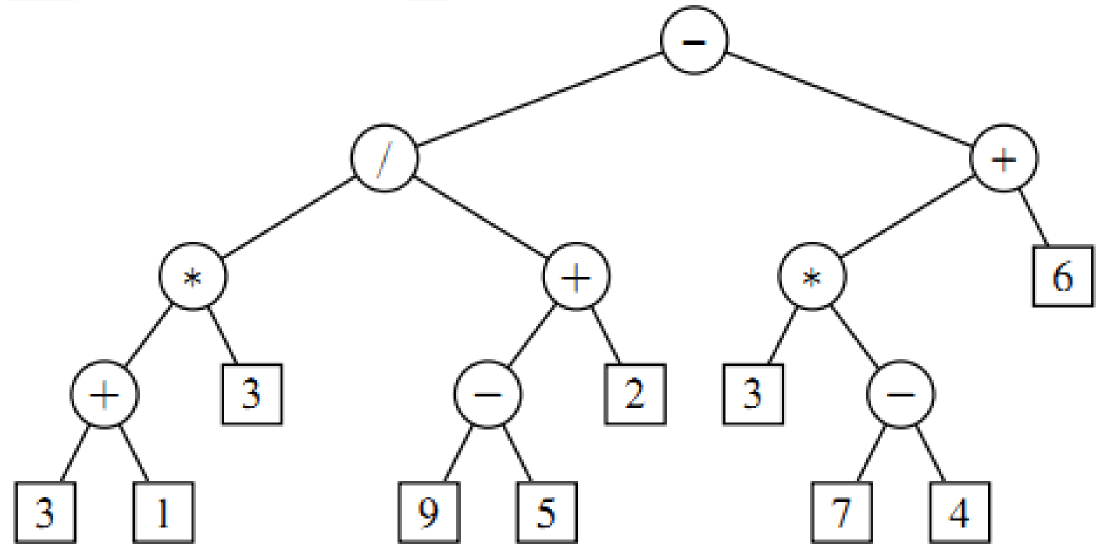
Fig. Binary Tree

What is a the result of a **PostOrder/PreOder/InOrder**?

Expression Tree

Expression Tree

- An arithmetic expression can be represented by a binary tree.
  - Leaves: are variables or constants
  - internal nodes: are one of the operators +, −, ∗, and /



[{(3+1) ∗ 3} / {(9−5)+2}] − [{3 ∗ (7−4)}+6]

Expression Tree Implementation

```java
package lec10_tree;

public class ExpressionTree {
    private String value;
    private ExpressionTree left;
    private ExpressionTree right;

    public ExpressionTree(String value,
            ExpressionTree left,
            ExpressionTree right) {
        this.value = value;
        this.left = left;
        this.right = right;
    }
}
```

How to display expression tree?

- Display Expression Tree likes Traversal Tree
- In order:
  - Print a left parenthesis; and then
  - traverse the left subtree; and then
  - print the root; and then
  - traverse the right subtree; and then
  - print a right parenthesis.
- Post order?
- Pre order?

How to evaluate expression value from Expression Tree?

```
public double total() {
    if (this.left == null && this.right == null) {
        // TODO
    } else if (this.value.equals("+")) {
        // TODO
    } else if (this.value.equals("-")) {
        // TODO
    } else if (this.value.equals("*")) {
        // TODO
    }
    return 0.0;
}
```

FACULTY OF INFORMATION TECHNOLOGY