

# DATA STRUCTURE AND ALGORITHMS

## LECTURE 5

### Trees

---

# Reference links:

<https://cs.nyu.edu/courses/fall17/CSCI-UA.0102-007/notes.php>

<https://www.comp.nus.edu.sg/~stevenha/cs2040.html>

[M.Goodrich, chapter 8]

---

# Lecture outline

- ❑ Tree ADT
  - Definitions
  - Applications
- ❑ Binary Tree
  - Definitions
  - Implementations
    - Linked Structure
    - Array based
  - Binary Tree Traversal
- ❑ Binary Tree Application

# Tree ADT

---

## Introduction

---

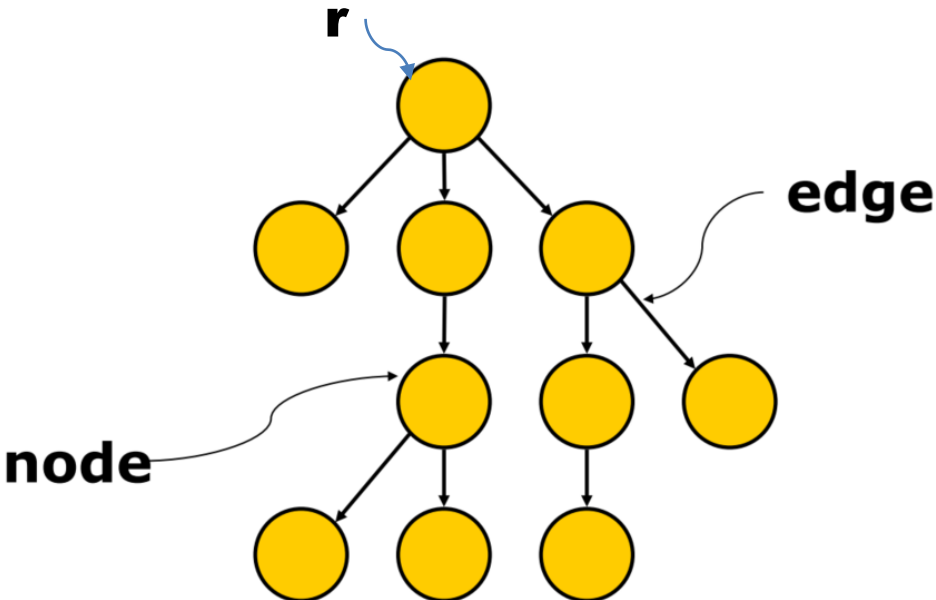
---

# Why trees?

- ❑ Tree structures are a breakthrough in data organization, nonlinear way. – Cách tổ chức dữ liệu phi tuyến
  - ❑ Allow implementing algorithms much faster than using linear data structures, such as arrays or linked lists.
  - ❑ Trees are ubiquitous in CS, covering operating systems, computer graphics, databases, etc.
-

# Trees: definitions

- A tree is a collection of nodes. Non-empty trees have a distinguished **node**  $r$ , called root, and zero or more nonempty (sub)trees  $T_1, T_2, \dots, T_k$ , each of whose roots are connected by a directed **edge** from  $r$



**node:** data object in tree

**edge:** links between nodes

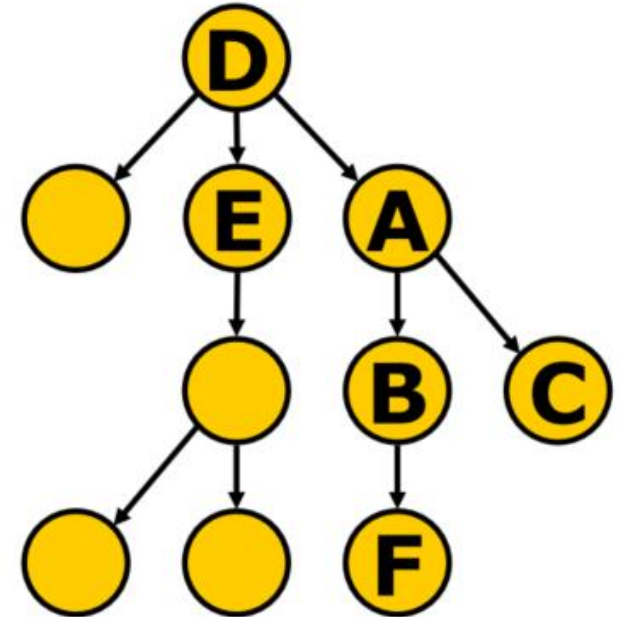
# Trees: definitions

## □ Relationship

- **A** is a **parent** of **B** and **C**
- **B** and **C** are **children** of **A**
- **B** and **C** are **siblings**  
(with the same parent **A**)
- **D** is an **ancestor** of **B**
- **B** is a **descendant** of **A** and **D**

□ **D** is **root** (has no parent) – nút gốc

□ **F** is **leaf** (has no children) – nút lá



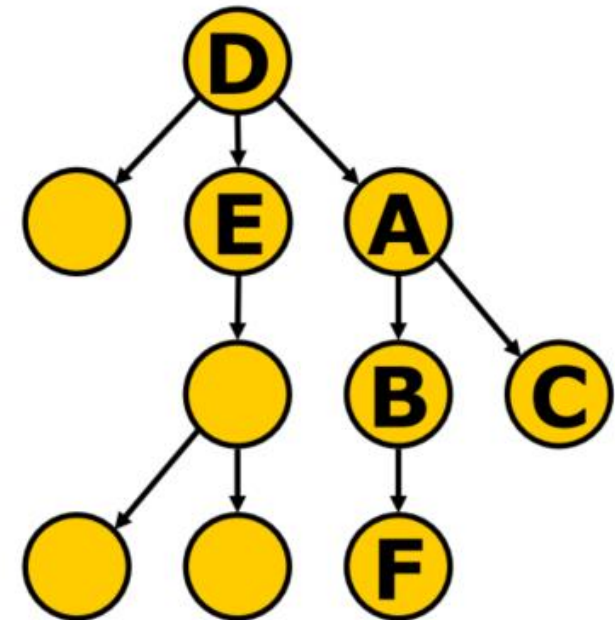
# Trees: definitions

□ **External** node – node ngoài

- Has no children (also leaf)
  - F, C are external nodes

□ **Internal** node – node trong

- Has one or more children
  - E, A, B are internal nodes





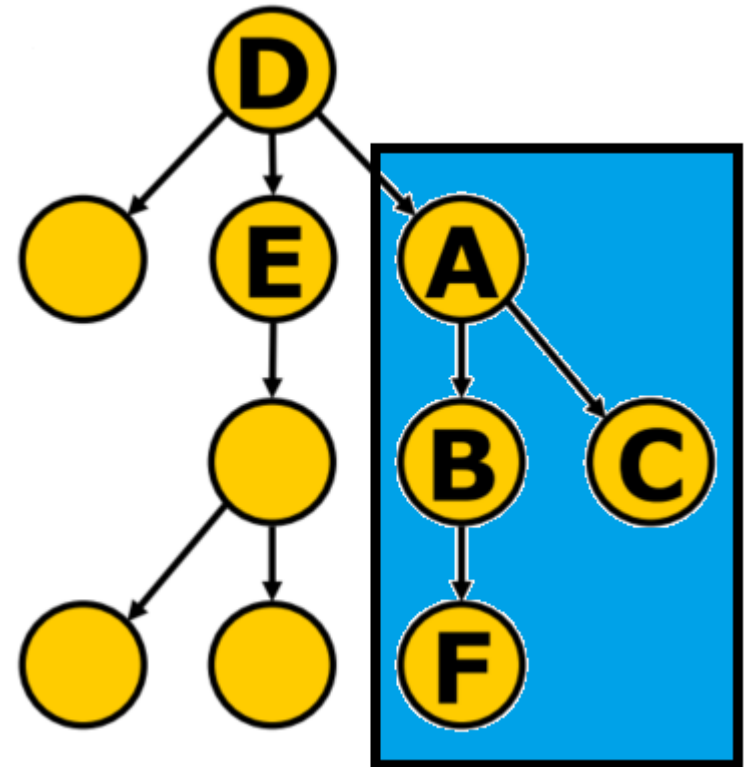
# Trees: definitions

## □ Subtree – cây con

- A node and all of its descendants

Ex: Subtree with root **A**

- Q: can a leaf be a subtree?



# Trees: definitions

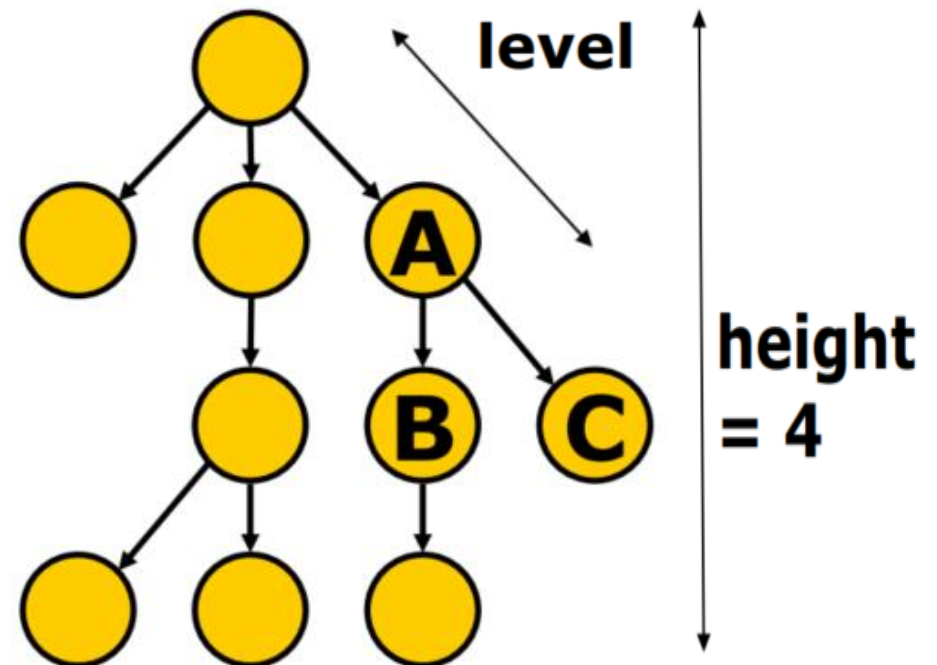
## □ Level of a node - mức của node

- Number of nodes on the path from the root to the node
  - level of **root** is 1
  - level of **A** is 2

## □ Height of a tree

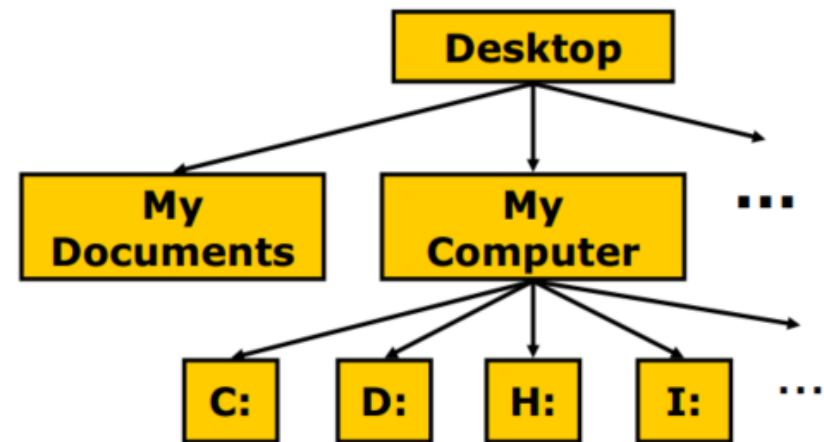
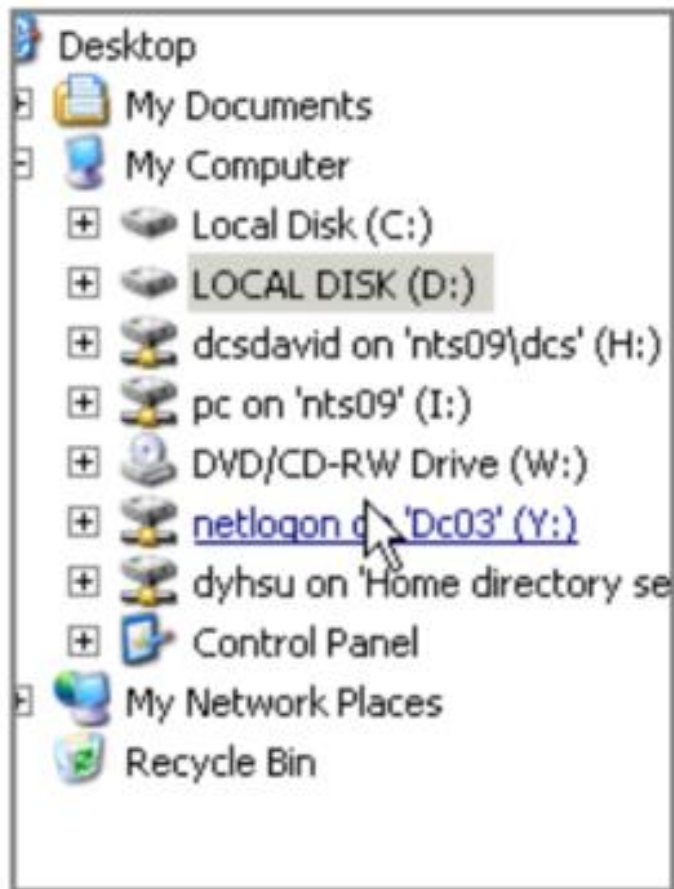
– chiều cao của cây

- Maximum level of the nodes in the tree



# Trees: applications

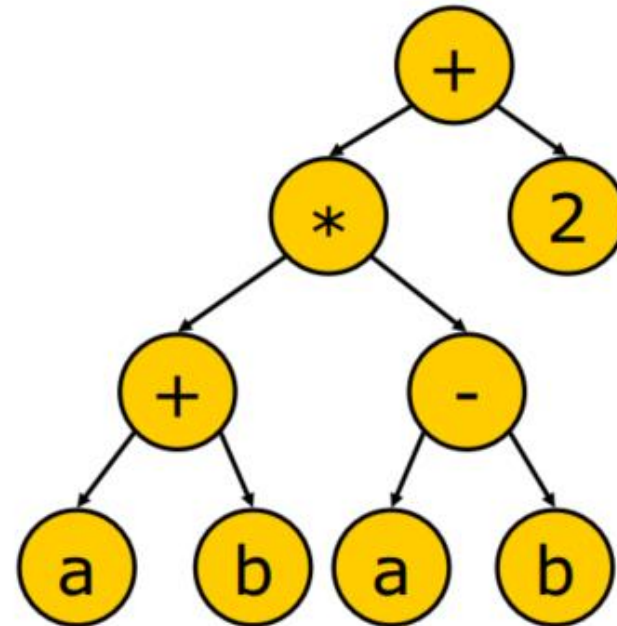
## □ File systems



# Trees: applications

## □ Arithmetic Expressions

$$(a+b) * (a-b) + 2$$

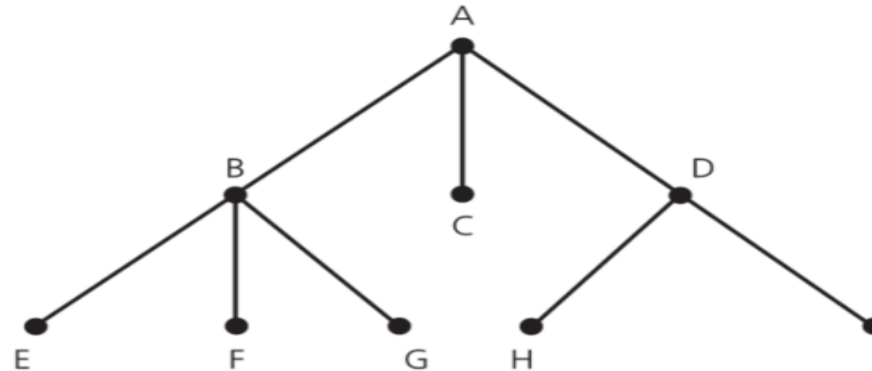


Q: How do you construct such a tree from a given arithmetic expression?

# General Trees

## □ An $n$ -ary tree (cây $n$ phân)

- A tree whose nodes each have no more than  $n$  children



## ■ The Tree ADT

- Specification, Implementation
- Read and handle by yourself

[M. Goodrich, subsection 8.1.2, 8.1.3]

# General Trees

## □ Tree ADT Methods

`element(v)`     *// returns the content of node v*  
`root()`         *// returns the root node of the tree.*  
`parent(v)`      *// returns the parent of node v*  
`children(v)`    *// returns an iterable container holding the children of node v*  
`isInternal(v)`   *// returns true if node v is internal*  
`isExternal(v)`   *// returns true if node v is external*  
`isRoot(v)`      *// returns true if node v is the root*  
`size()`          *// returns the number of nodes in the tree*  
`isEmpty()`       *// returns true if the tree is empty*  
`positions()`    *// returns an iterable container of all nodes in the tree*  
`elements()`      *// returns an iterable container of the content of all node in the tree*  
`replace(v,e)`    *// replace the contents of node v with e. return the old contents of v*

For more details on the Tree Interface [M. Goodrich, p313]

# Tree ADT

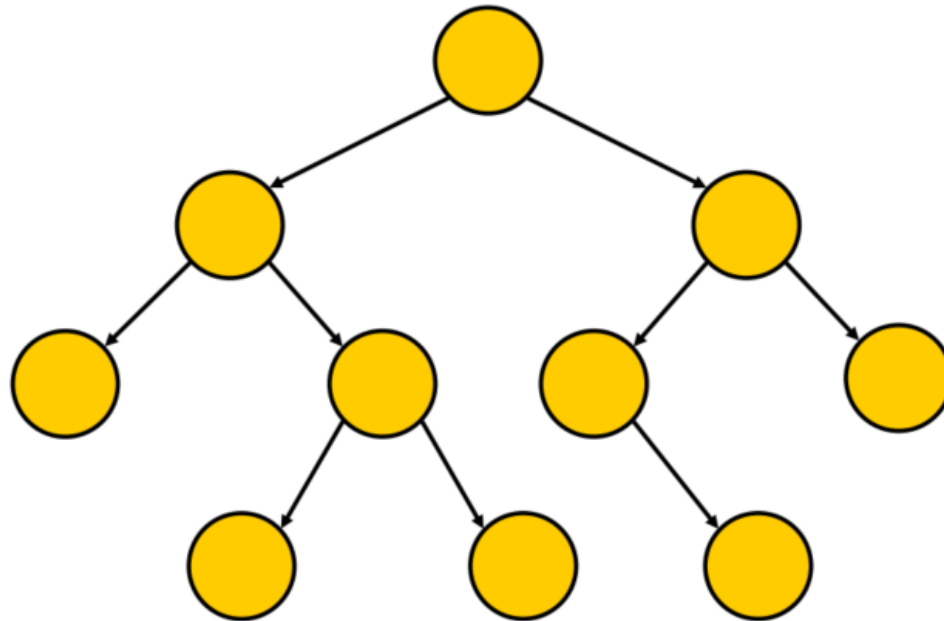
---

## Binary Tree

---

# Binary Tree: Definitions

- ❑ Binary Tree = 2-ary tree (cây nhị phân)
  - Each node has **at most 2 “ordered”** children
  - Two children: left child, right child

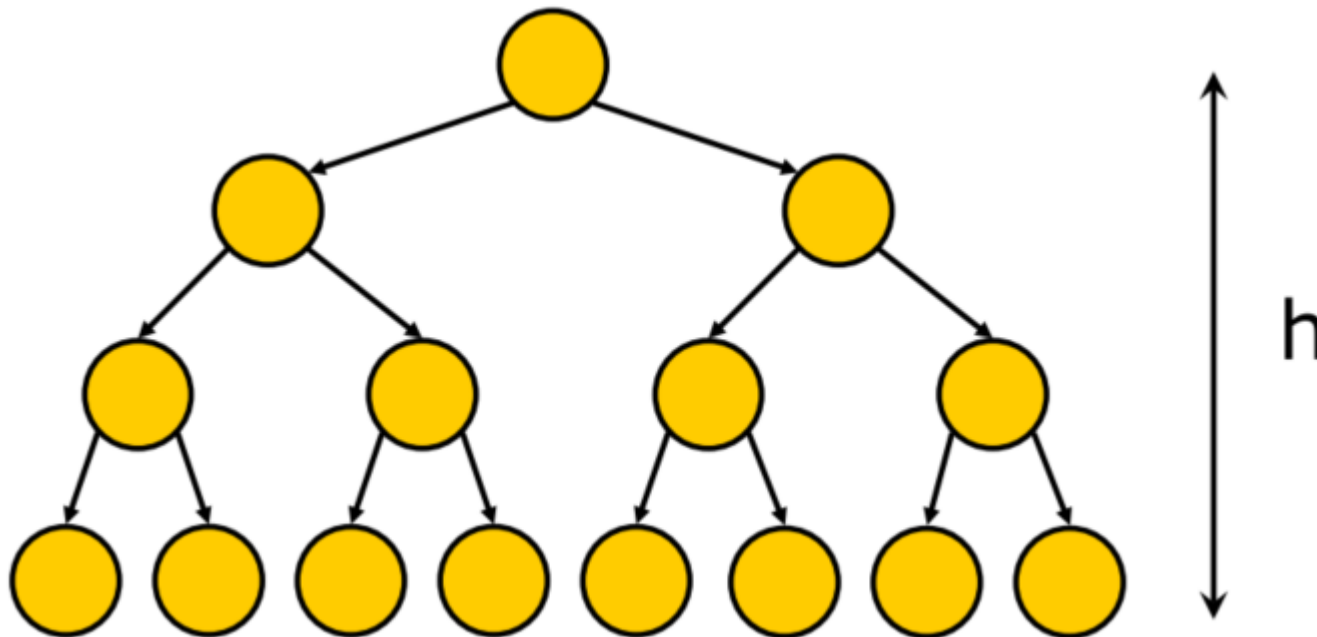




# Binary Tree: Definitions

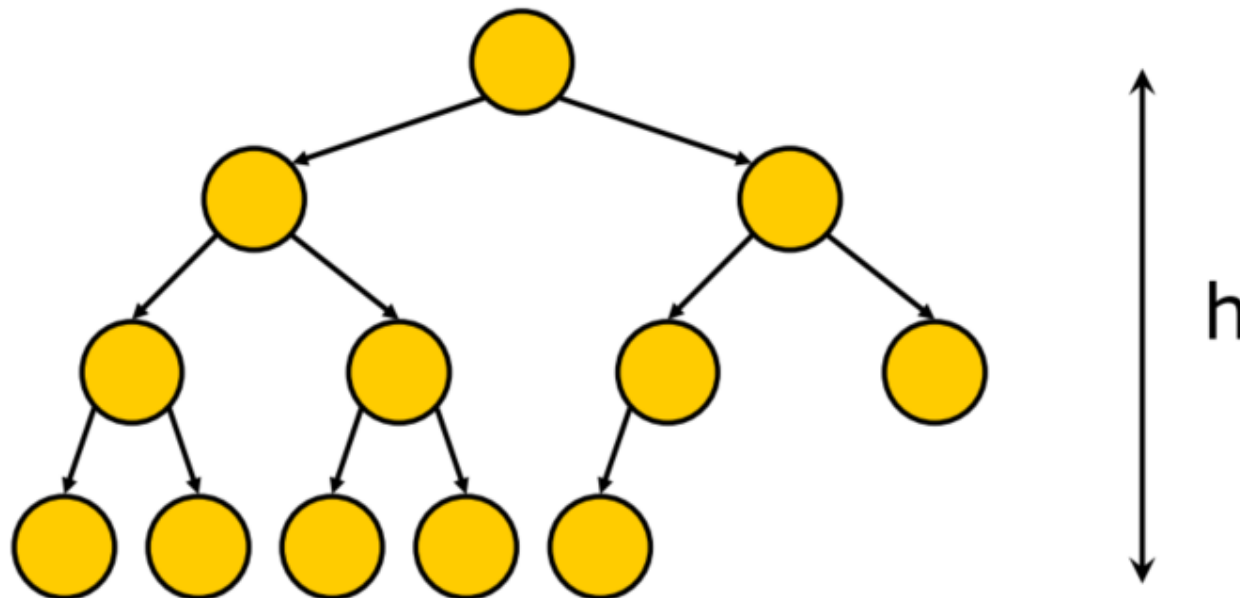
## □ Full Binary Tree – Cây nhị phân đầy đủ

- All nodes at a level  $< h$  have two children (where  $h$  is the height of the tree)



# Binary Tree: Definitions

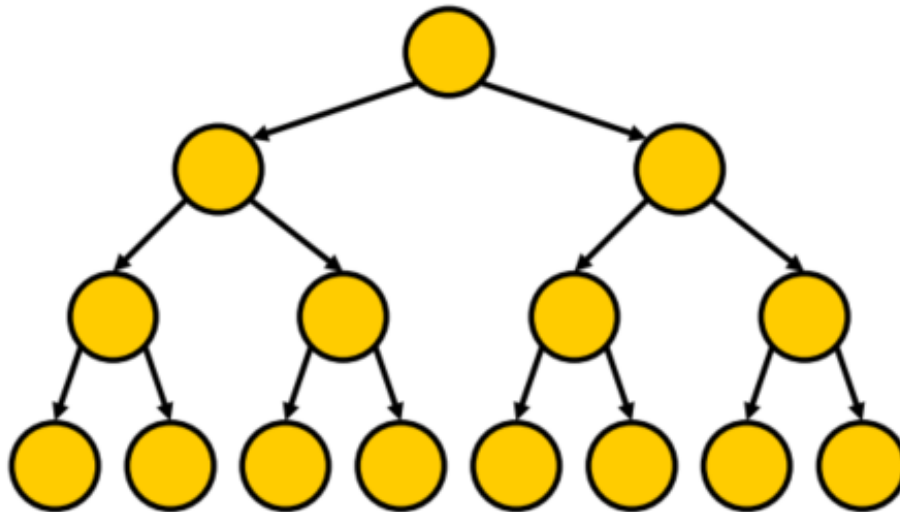
- ❑ Complete Binary Tree – Cây nhị phân hoàn chỉnh
  - Full down to level  $h-1$
  - Level  $h$  filled in from left to right.



# Binary Tree: Definitions

## □ Full Binary Tree Properties

- Number of nodes in a full binary tree of height  $h$  is  $N = 2^h - 1$ .
- Therefore, the height of a full binary tree is  $O(\log N)$



- $h = 4$
- $N = 15$

- Some other interesting properties [M. Goodrich, ss. 8.2.2, p321]

# Binary Tree ADT: Specification

## □ Binary Tree Interface

```
public interface BinaryTree<E> extends Tree<E> {  
    // Returns the Position of p's left child (or null if no child exists).  
    Position<E> left(Position<E> p) throws IllegalArgumentException;  
    // Returns the Position of p's right child (or null if no child exists)  
    Position<E> right(Position<E> p) throws IllegalArgumentException;  
    // Returns the Position of p's sibling (or null if no sibling exists)  
    Position<E> sibling(Position<E> p) throws IllegalArgumentException;  
}
```

*Position meaning for node on tree*

A BinaryTree interface [M.Goodrich, p.319]

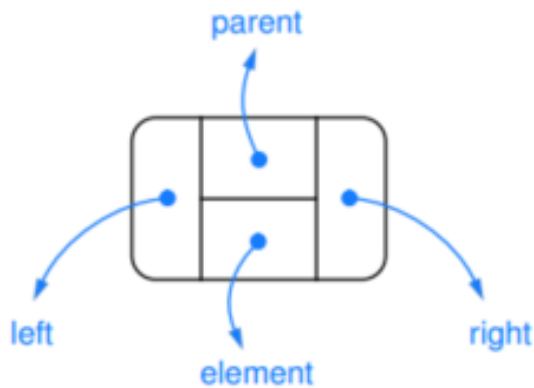
---

# Binary Tree: Implementation

- Implement Binary Tree
  - Use Linked structure
  - Array based

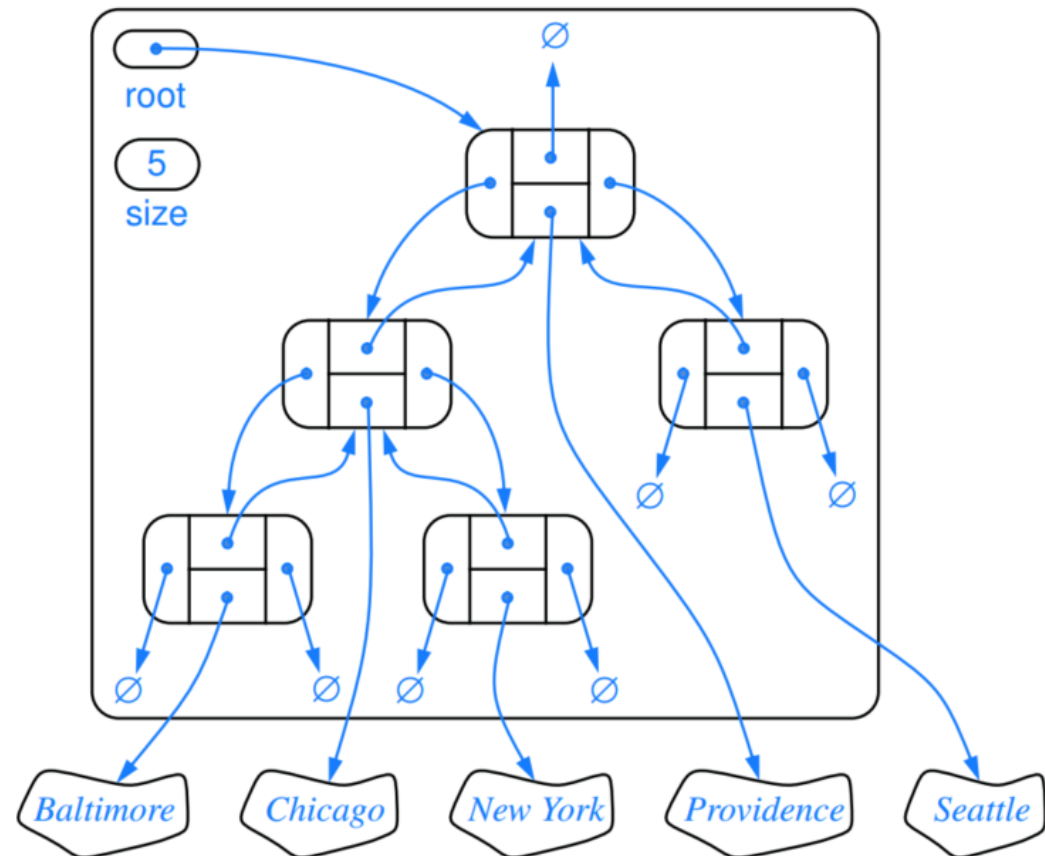
# Binary Tree: Linked Structure

- ## ❑ Implement Binary Tree using Linked Structure



## One node with:

```
parent, left, right: references
element: data
```



## Binary Tree with five nodes

Source: [M.Goodrich, p323]

---

# Binary Tree: Linked Structure

## □ Implement Binary Tree using Linked Structure

- Some methods need updated

```
class LinkBinaryTree<E> {  
    class Node<E> {  
        private E element; // an element stored at this node  
        private Node<E> parent; // a reference to the parent node (optional)  
        private Node<E> left; // a reference to the left child (if any)  
        private Node<E> right; // a reference to the right child (if any)  
        //more definitions  
    }  
    //methods  
}
```

For more detail see Code Fragment 8.9-8.11 [M.Goodrich, p326]

---

# Binary Tree: Linked Structure

## □ Implement Binary Tree using Linked Structure

### ■ Some methods need updated

`addRoot(e)` : Khởi tạo nút gốc cho cây rỗng.

`addLeft( p, e)` : Thêm nút con trái cho nút p (chưa có con trái).

`addRight( p, e)` : Thêm nút con phải cho nút p (chưa có con phải).

`set( p, e)` : Đặt phần tử e vào nút p.

`attach( p, T1,T2)` : Đính 2 cây T1, T2 thành 2 cây con trái và phải của nút lá p.

`remove( p)` : Xóa nút p, thay thế bằng con (nếu có 1 nút con).

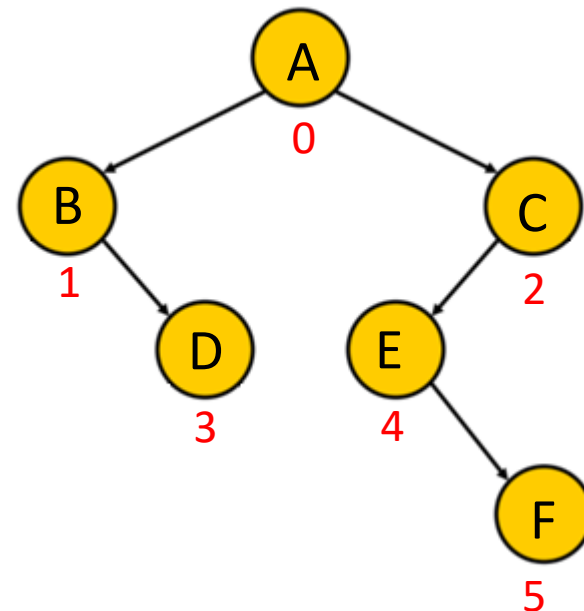
For more detail see Code Fragment 8.10, 8.11 [M.Goodrich, p328]



# Binary Tree: Array-Based

## □ Implement Binary Tree using Array

Nodes T[]	0	1	2	3	4	5
Data	A	B	C	D	E	F
Parent	-1	0	0	1	2	4
Left	1	-1	4	-1	-1	-1
Right	2	3	-1	-1	5	-1



- Pros: Easy to handle each node
- Cons: Waste memory

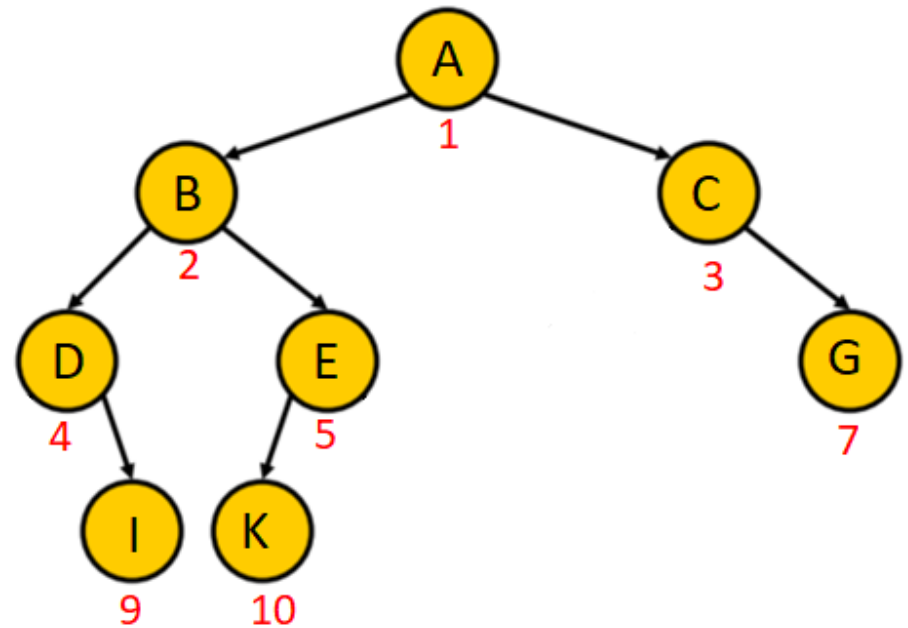
# Binary Tree: Array-Based

- Implement Binary Tree using Array
  - Effective array for Complete Binary Tree

Nodes T[]

A	B	C	D	E	∅	G	∅	I	K
1	2	3	4	5	6	7	8	9	10

Given that a node is stored in index position  $i$ , what is the position of its **parent**? **left child**? **right child**?



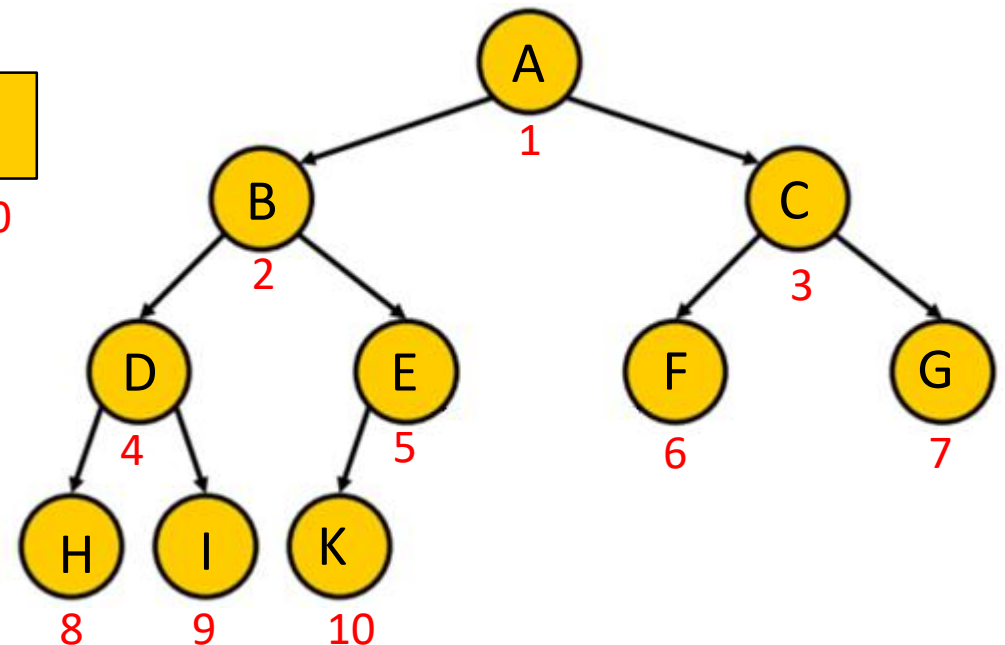
# Binary Tree: Array-Based

- ❑ Implement Binary Tree using Array
  - Effective array for Complete Binary Tree

Nodes T[]

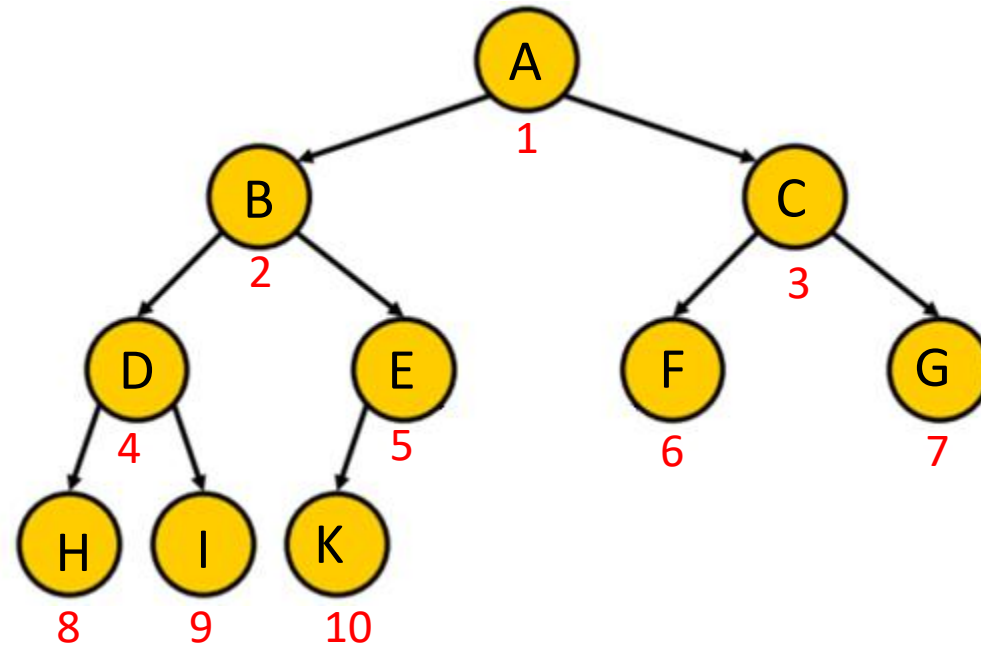
A	B	C	D	E	F	G	H	I	K
1	2	3	4	5	6	7	8	9	10

Given that a node is stored in index position  $i$ , what is the position of its **parent**? **left child**? **right child**?



# Binary Tree: Short review

## □ Binary Tree Definitions



# Binary Tree: Short review

## □ Binary Tree Specification

```
public interface BinaryTreeInterface<T> {  
    T root();  
    int size();           // number of node in tree  
    boolean isEmpty();  
    int numChildren(T p); // nmber of children of element p;  
  
    T parent(T p); //return parent of p  
    T left(T p);   //return left child of p  
    T right(T p);  //return right child of p  
    T sibling(T p); //return sibling of p  
}
```

# Binary Tree: Short review

## □ Binary Tree Implementation: using array

```
public class ArrayBinaryTree<E,T> implements BinaryTreeNodeInterface<T> {  
    private E [] array;  
    private int n = 0;  
    private int defaultsize = 100;  
  
    public ArrayBinaryTree(){  
        array = (E[]) new Object[defaultsize];  
    }  
    //update methods  
    public void setRoot(E element) {  
        //Set element to root of an empty tree (at index 1)  
    }  
    public void setLeft(int p, E element) {  
        //Set left child of p (at index 2p)  
    }  
    public void setRight(int p, E element) {  
        //Set right child of p (at index 2p+1)  
    }  
}
```

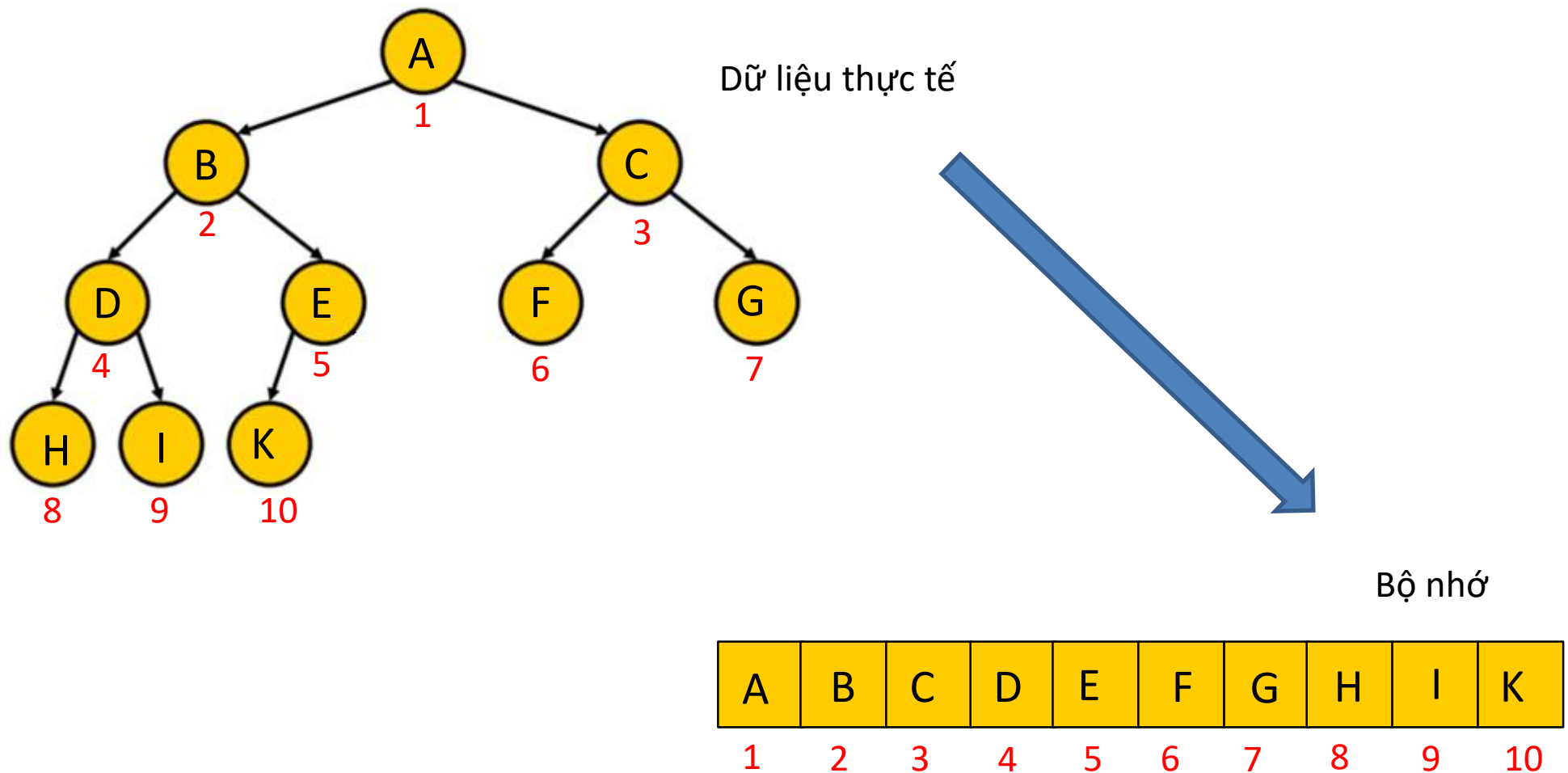
# Binary Tree: Short review

## ❑ Binary Tree Implementation: using linked structure

```
public class LinkedBinaryTree<E,T> implements BinaryTreeInterface<T> {  
  
    protected static class Node<E> {  
        private E element;           // an element stored at this node  
        private Node<E> parent;       // a reference to the parent node  
        private Node<E> left;         // a reference to the left child  
        private Node<E> right;        // a reference to the right child  
  
        // Constructs a node with the given element and neighbors. */  
        public Node(E e, Node<E> above, Node<E> leftChild, Node<E> rightChild){  
            // Sinh viên hoàn thiện phương thức  
        }  
    }  
  
    //update methods  
    public Node<E> addRoot(E element) {} // Add element to root of an empty tree  
    public Node<E> addLeft(Node p, E element) {} // Add element to left child node of p  
    public Node<E> addRight(Node p, E element) {} // Add element to right child node of p  
    public void set(Node p, E element) {} // set element to nnode p  
}
```

# Binary Tree: Short review

## □ Binary Tree Implementation





# Tree ADT

---

## Binary Tree Traversal

---

---

# Binary Tree Traversal: Orders

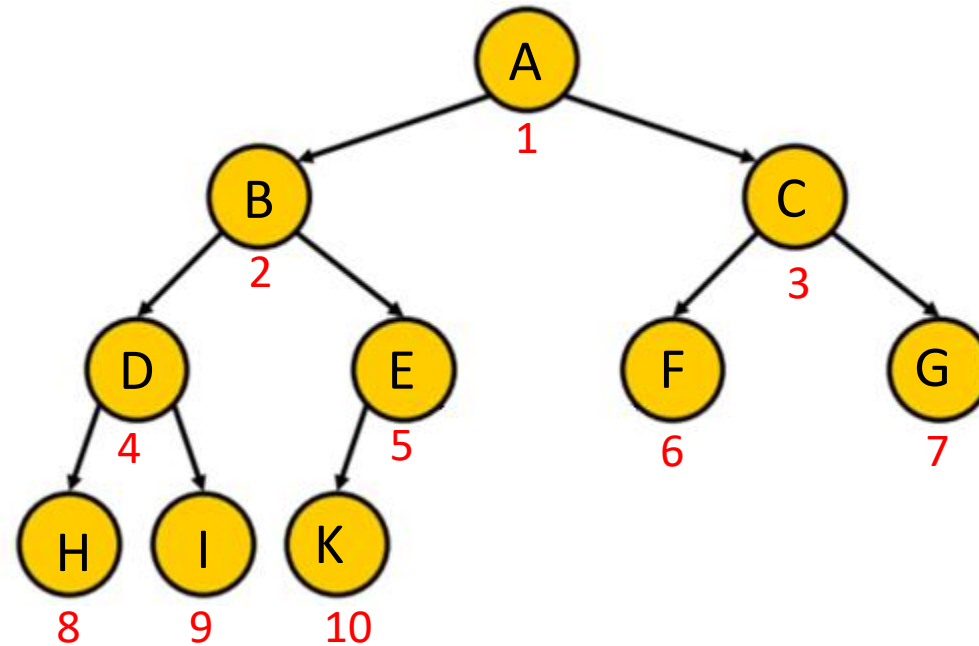
- ❑ A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree
  - ❑ To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once
  - ❑ Tree traversals are naturally recursive
  - ❑ Since a binary tree has three “parts,” there are six possible ways to traverse the binary tree:
    - root, left, right
    - left, root, right
    - left, right, root
    - root, right, left
    - right, root, left
    - right, left, root
-

---

# Binary Tree Traversal: Orders

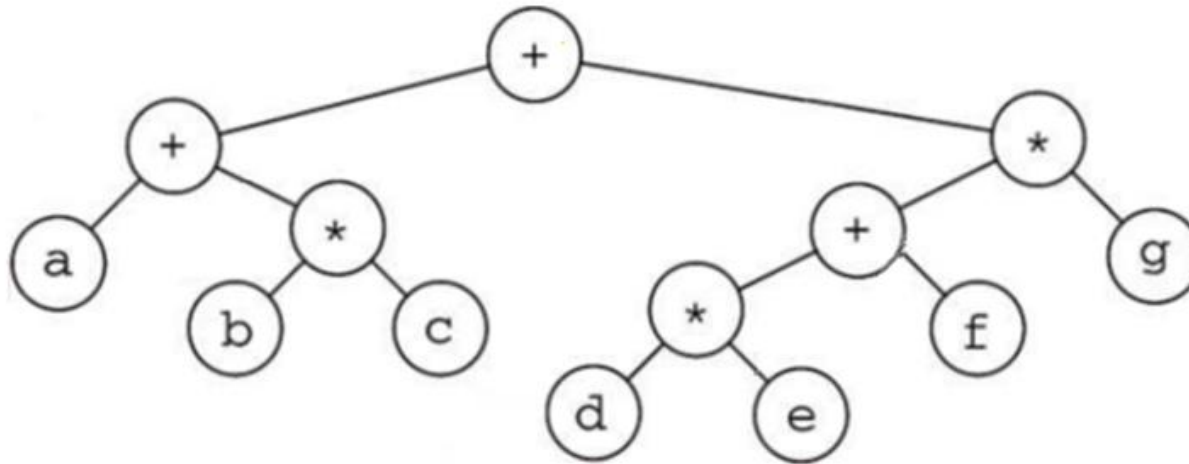
- A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree
  - To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once
  - Tree traversals are naturally recursive
  - Since a binary tree has three “parts,” there are six possible ways to traverse the binary tree:
    - root, left, right ➡ Pre-order Traversal
    - left, root, right ➡ In-order Traversal
    - left, right, root ➡ Post-order Traversal
-

# Binary Tree Traversal: Orders



- **Pre-order** Traversal (root, left, right) ➡ ?
- **In-order** Traversal (left, root, right) ➡ ?
- **Post-order** Traversal (left, right, root) ➡ ?

# Binary Tree Traversal: Orders



- **Pre-order** Traversal (root, left, right) ➡ ?
- **In-order** Traversal (left, root, right) ➡ ?
- **Post-order** Traversal (left, right, root) ➡ ?

---

# Pre-order Traversal

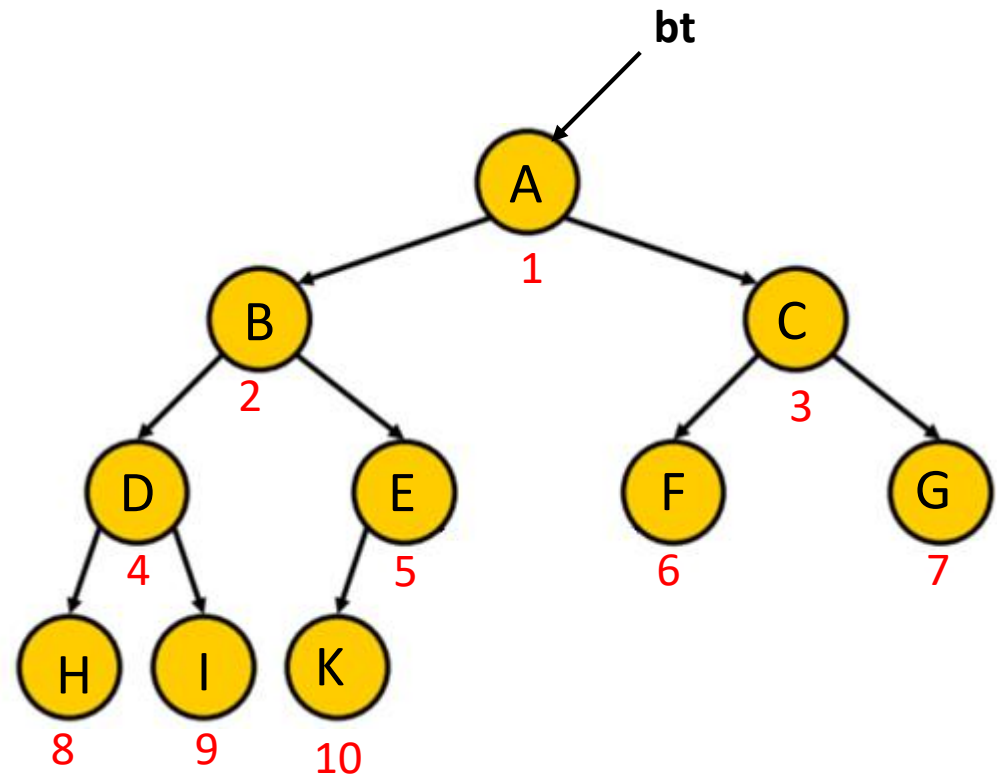
- ❑ In Pre-order Traversal, the root is visited **first**
  - Order: Root - Left - Right
- ❑ Algorithm for preorder traversal to print out all the elements in a binary tree (bt):

```
public void preorderPrint(BinaryTree bt) {  
    if (bt == null) return;  
    System.out.println(bt.element);  
    preorderPrint(bt.left);  
    preorderPrint(bt.right);  
}
```

# Pre-order Traversal

## Example

```
public void preorderPrint(BinaryTree bt) {  
    if (bt == null) return;  
    System.out.println(bt.element);  
    preorderPrint(bt.left);  
    preorderPrint(bt.right);  
}
```



Pre-order traversal result: **A B D H I E K C F G**

# Post-order Traversal

- ❑ In Post-order Traversal, the root is visited **last**
  - Order: Left - Right - Root
- ❑ Algorithm for postorder traversal to print out all the elements in a binary tree (bt):

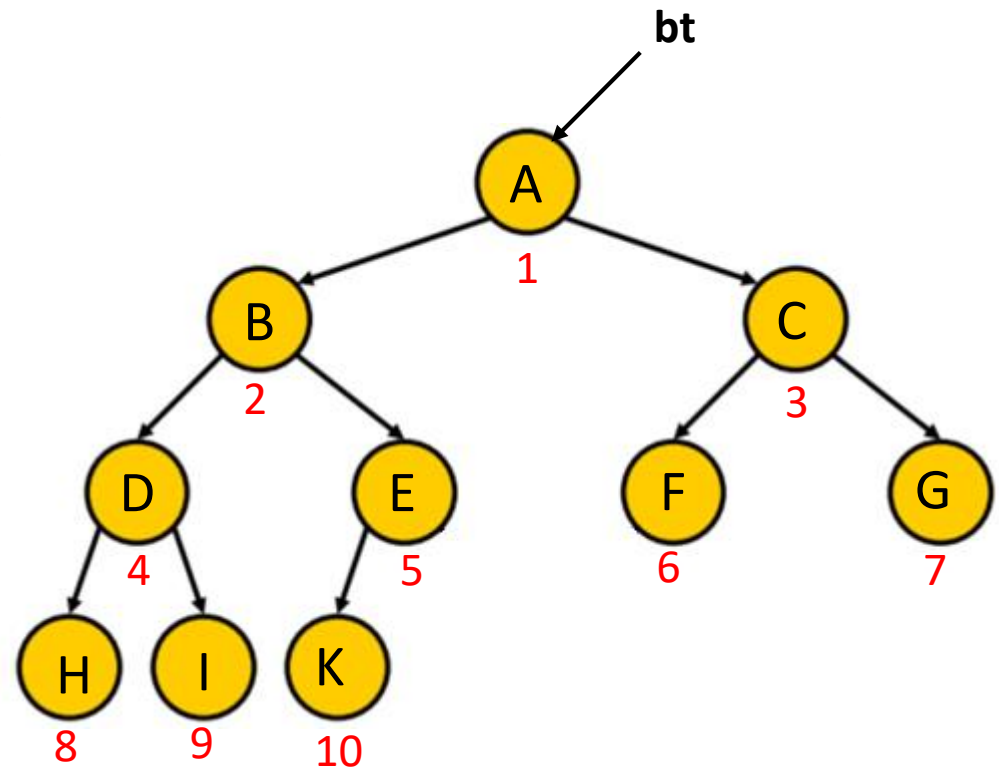
```
public void postorderPrint(BinaryTree bt) {  
    if (bt == null) return;  
    preorderPrint(bt.left);  
    preorderPrint(bt.right);  
    System.out.println(bt.element);  
}
```



# Post-order Traversal

## □ Example

```
public void postorderPrint(BinaryTree bt) {  
    if (bt == null) return;  
    postorderPrint(bt.left);  
    postorderPrint(bt.right);  
    System.out.println(bt.element);  
}
```



Post-order traversal result: **H I D K E B F G C A**

---

# In-order Traversal

- ❑ In In-order Traversal, the root is visited **in the middle**
  - Order: Left - Root - Right

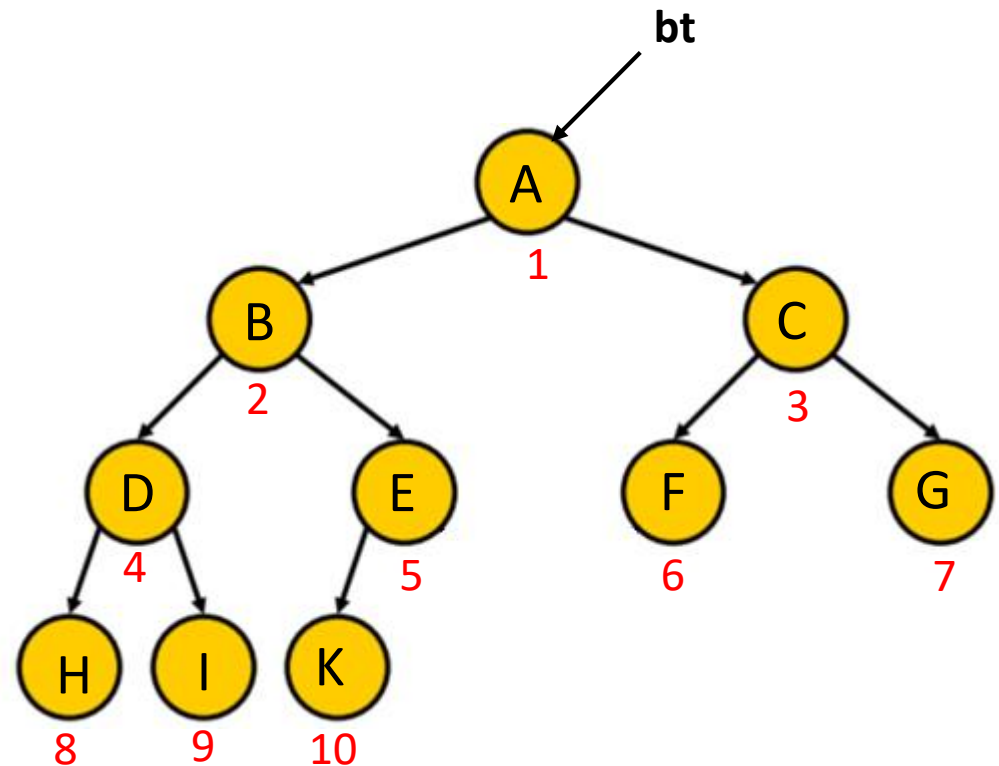
- ❑ Algorithm for In-order traversal to print out all the elements in a binary tree (bt):

```
public void inorderPrint(BinaryTree bt) {  
    if (bt == null) return;  
    inorderPrint(bt.left);  
    System.out.println(bt.element);  
    inorderPrint(bt.right);  
}
```

# In-order Traversal

## Example

```
public void inorderPrint(BinaryTree bt) {  
    if (bt == null) return;  
    inorderPrint(bt.left);  
    System.out.println(bt.element);  
    inorderPrint(bt.right);  
}
```



Post-order traversal result: **H D I B K E A F C G**

# Tree ADT

---

## Binary Tree Application

---

---

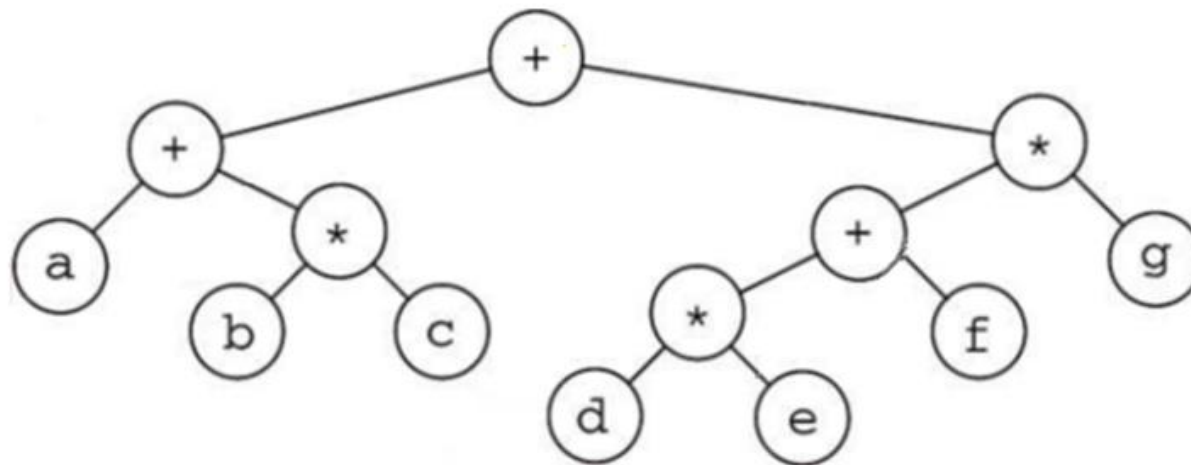
# Binary Tree: Applications

- ❑ Tree ADT has a variety of applications.
  - ❑ Same with binary tree. One important and well-known is **search trees** (more detail in lecture 7).
  - ❑ First, we discuss about Arithmetic Expression Trees –  
Cây biểu thức
-

# Binary Tree: Expression Tree

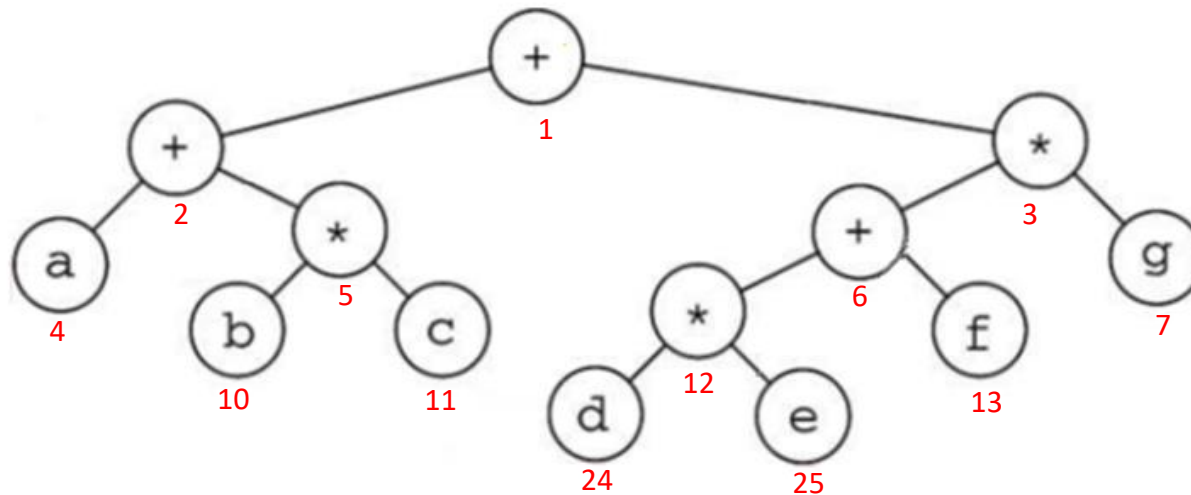
- ❑ Binary Tree for an Arithmetic Expression Tree
  - Internal nodes: operators (toán tử ở các nút trong)
  - Leaves: operands (toán hạng ở các nút lá)
- ❑ Example:

$(a + (b * c) + (d * e) + f) * g$



# Binary Tree: Expression Tree

- Using array representing the tree



+	+	*	a	*	+	g			b	c	*	f											d	e
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

- Using linked structure is better

# Binary Tree: Expression Tree

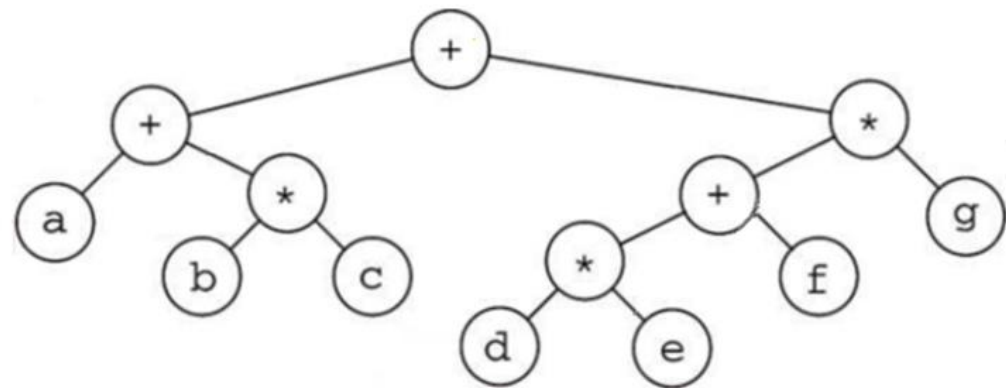
## □ Print Arithmetic Expressions

### ■ Using In-order traversal algorithm:

- print "(" before traversing left subtree
- print operand or operator when visiting node
- print ")" after traversing right subtree

### //Print Arithmetic Expression

```
void printTree(t) {  
    if (t.left != null)  
        print("(");  
        printTree (t.left);  
    print(t.element );  
    if (t.right != null)  
        printTree (t.right);  
    print (")");  
}
```



Result: ((a+(b\*c)) + ((d\*e)+f) \*g)



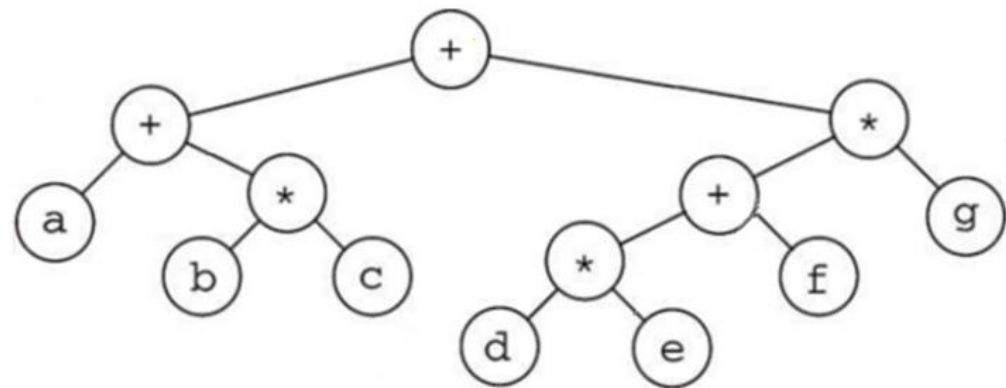
# Binary Tree: Expression Tree

## ❑ Evaluate Arithmetic Expressions

- Using Post-order traversal algorithm:
  - Recursively evaluate subtrees
  - Apply the operator after subtrees are evaluated

### //Evaluate Aithmetic Expression

```
float evaluate(t) {  
    if (t.left == null) //external node  
        return t.element;  
    else //internal node  
        x = evaluate (t.left);  
        y = evaluate (t.right);  
        let o be the operator t.element  
        z = apply o to x and y  
        return z;  
}
```



Result:  $z = ((a + (b * c)) + ((d * e) + f) * g)$

# Summary

