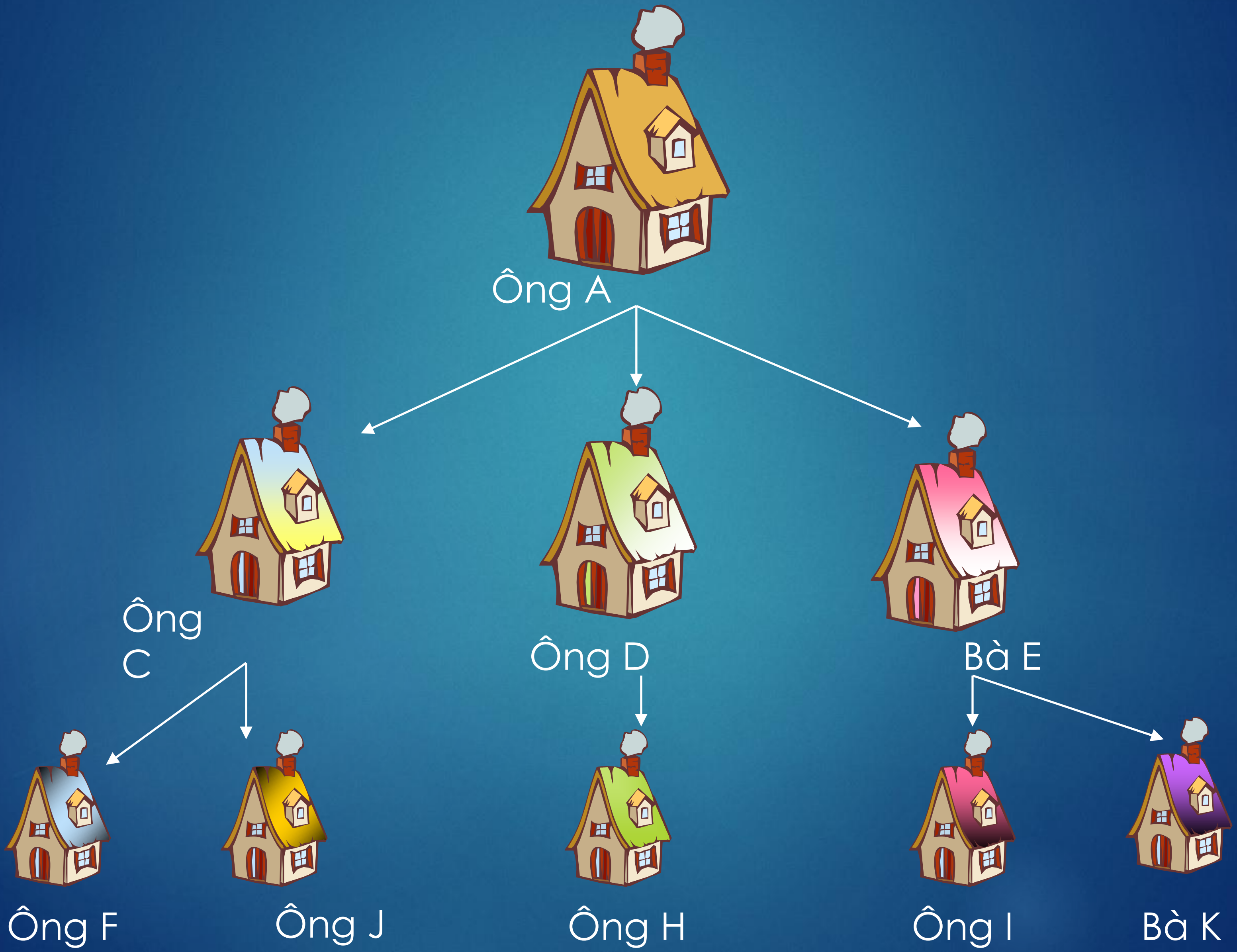
Tree

Example of Tree

Family Tree



Ông A

Ông C

Ông D

Bà E

Ông F

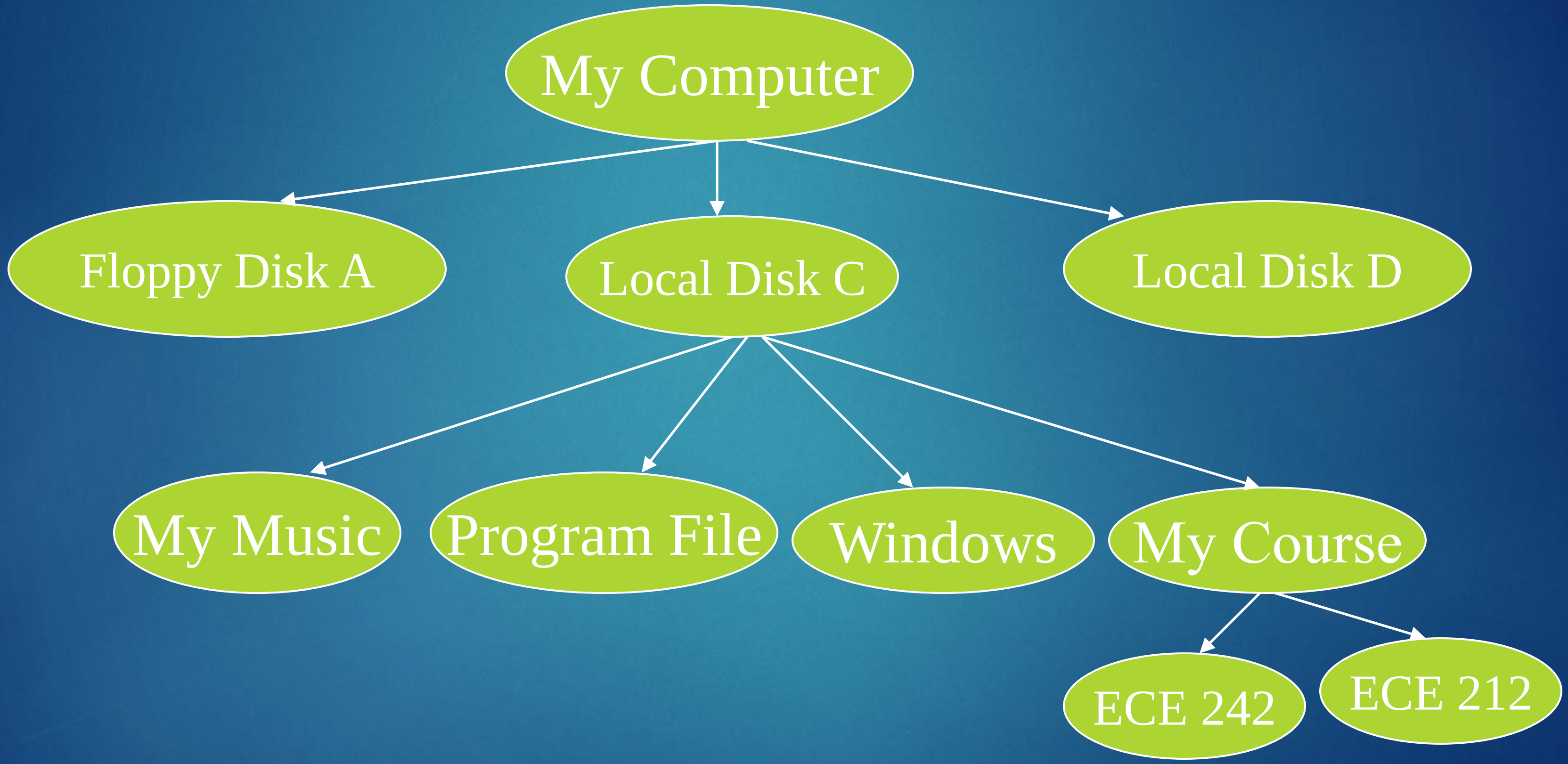
Ông J

Ông H

Ông I

Bà K

File System In Windows



My Computer

Floppy Disk A

Local Disk C

Local Disk D

My Music

Program File

Windows

My Course

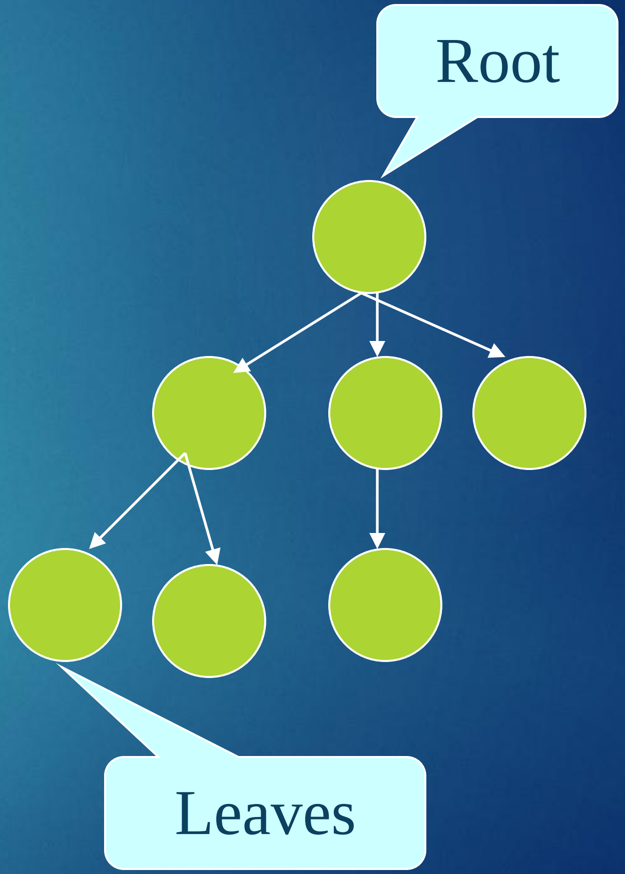
ECE 242

ECE 212

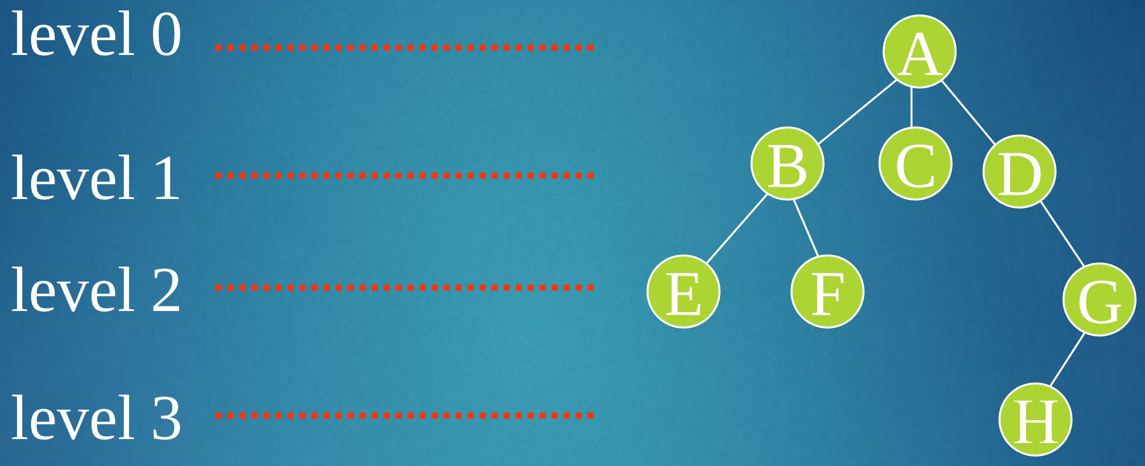
What’s Tree?

Tree

- Consist of *nodes* and *arcs*
- Depicted upside down with the *root* at the top and *leaves* at bottom
- No circle in the tree



Terminology



level 0

level 1

level 2

level 3

Parent: A, B, D, G
Children: B, C, D, E, F, G, H
Sibling: {B, C, D}, {E, F}
Leaves: C, E, F, H

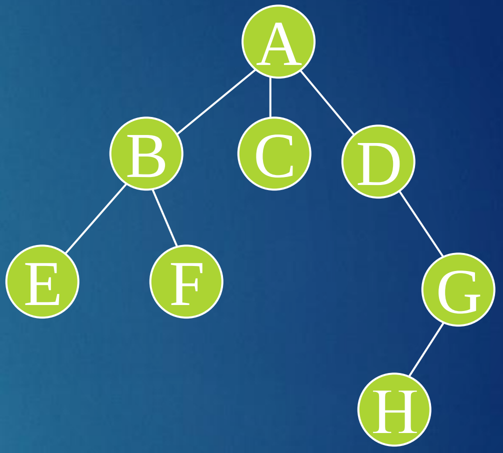
Definition TNode

- An abstract definition for general tree node

```
private class TNode  {
   Object obj;
   TNode  firstchild;
   TNode  nextsibling;
}
```

Print Node A’s Children

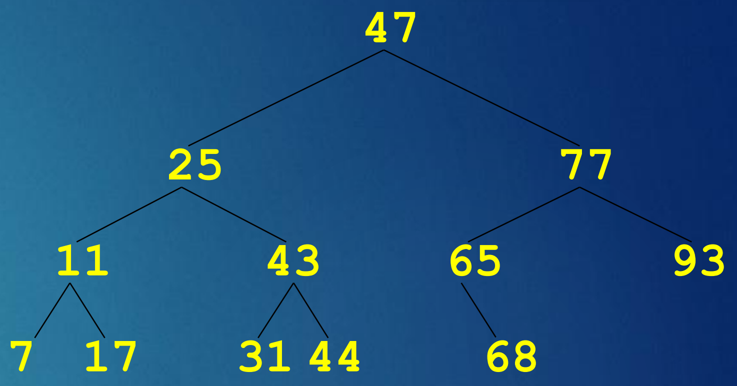
- Pseudo-code for print Node A’s children:



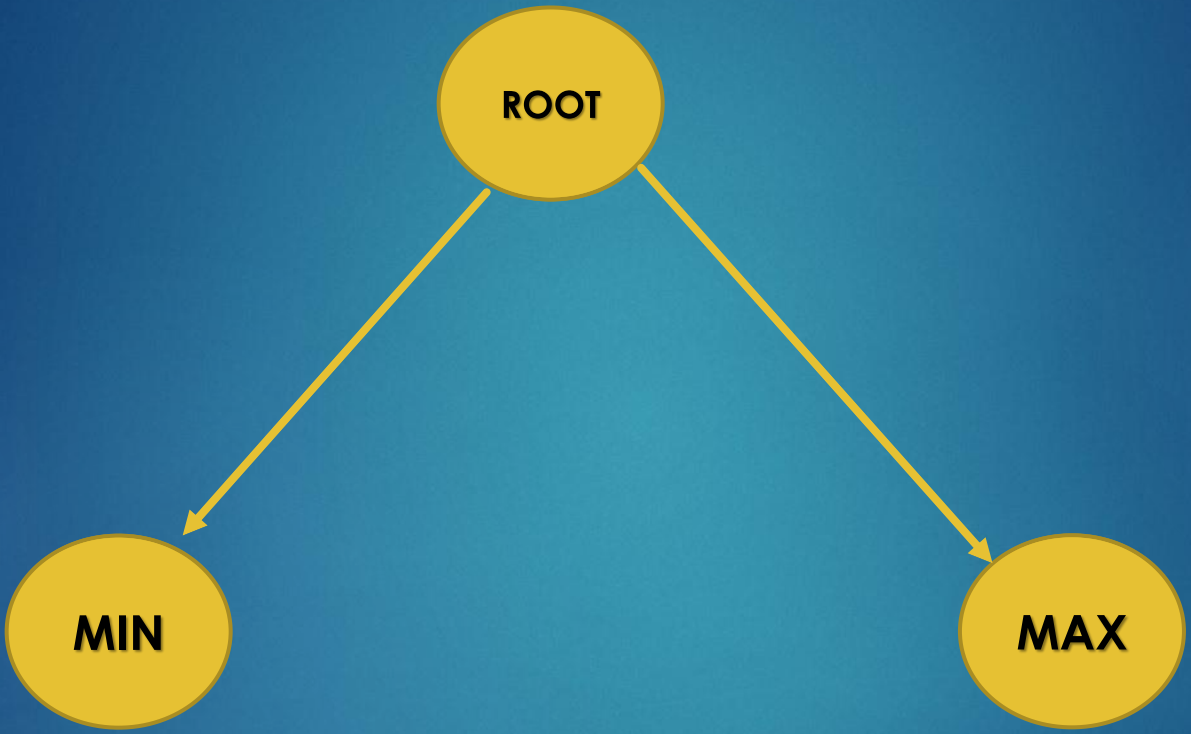
```
tmpnode = nodeA.firstchild;
while ( tmpnode!=null ) {
    print “ information of tmpnode”;
    tmpnode = tmpnode.nextsibling;
}
```

Tree traversals

- In-order
  - Left subtree, print node, right subtree
  - 7, 11, 17, 25, 31, 43, 44, 47, 65, 68, 77, 93
- Preorder
  - Print node, left subtree, right subtree
  - 47, 25, 11, 7, 17, 43, 31, 44, 77, 65, 68, 93
- Postorder
  - Left subtree, right subtree, print node
  - 7, 17, 11, 31, 44, 43, 25, 68, 65, 93, 77, 47
- Start from root
  - Recursively traverse (inorder, preorder, or postorder)



MAX MIN IN BST

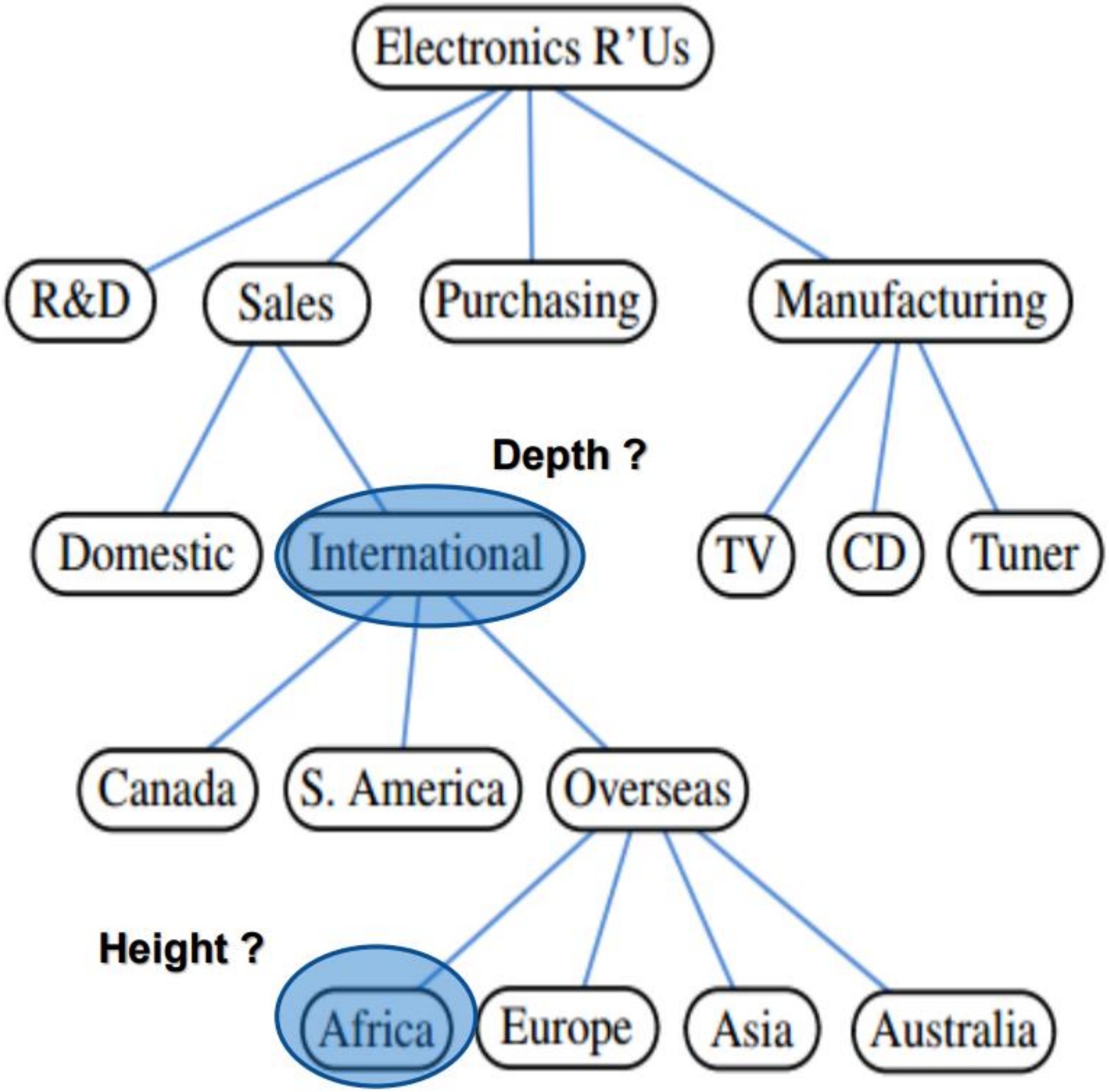


DEPTH

- Let p be a position within tree T. The depth of p is the number of ancestors of p, other than p itself.
- The depth of p can also be recursively defined as follows: □ If p is the root, then the depth of p is 0.
- the depth of p is one plus the depth of the parent of p.

HEIGHT

- The height of a tree to be equal to the maximum of the depths of its positions (or zero, if the tree is empty).
- It is easy to see that the position with maximum depth must be a leaf.



Electronics R'Us

R&D

Sales

Purchasing

Manufacturing

Domestic

International

TV

CD

Tuner

Canada

S. America

Overseas

Africa

Europe

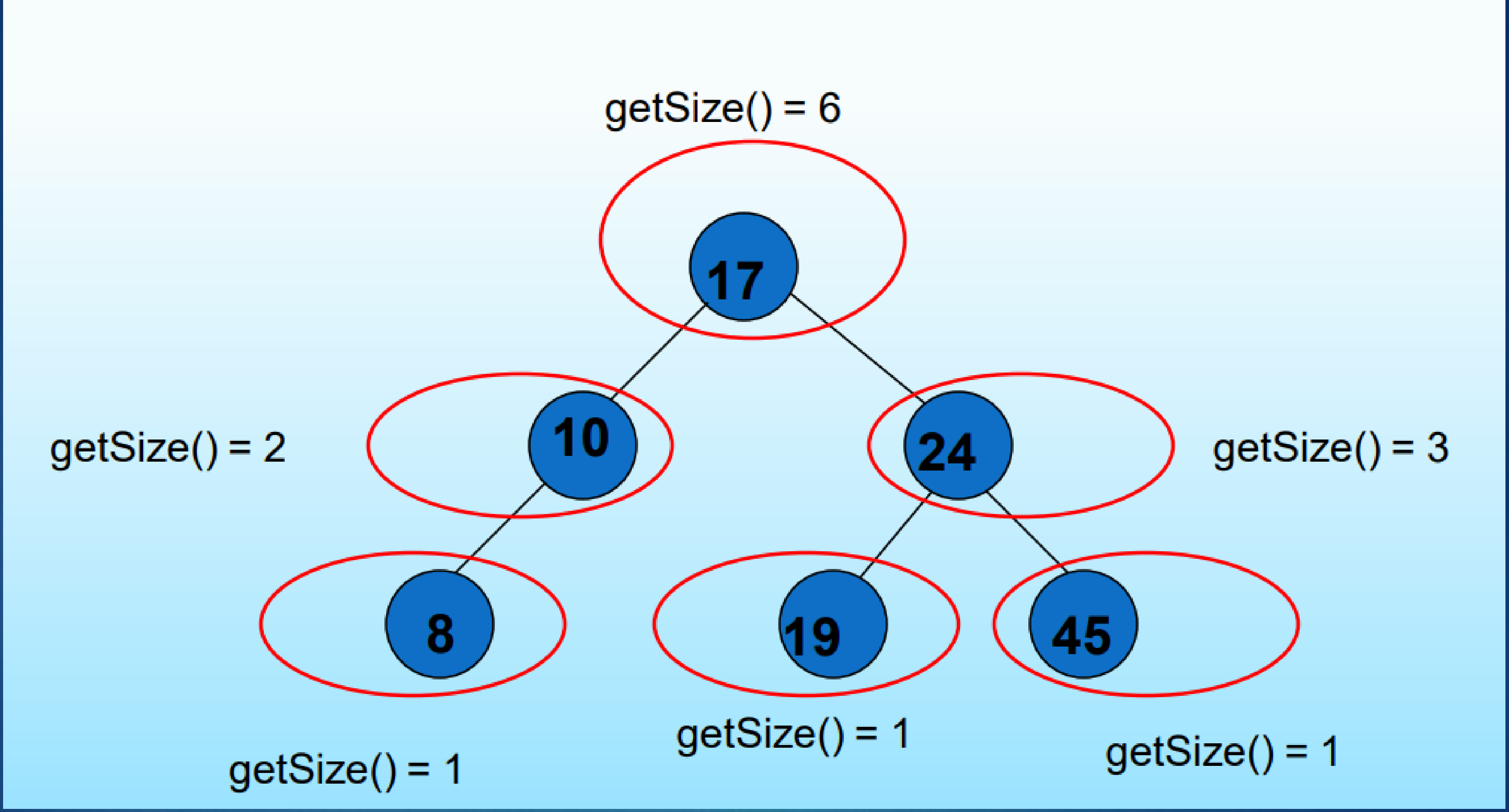
Asia

Australia

Depth ?

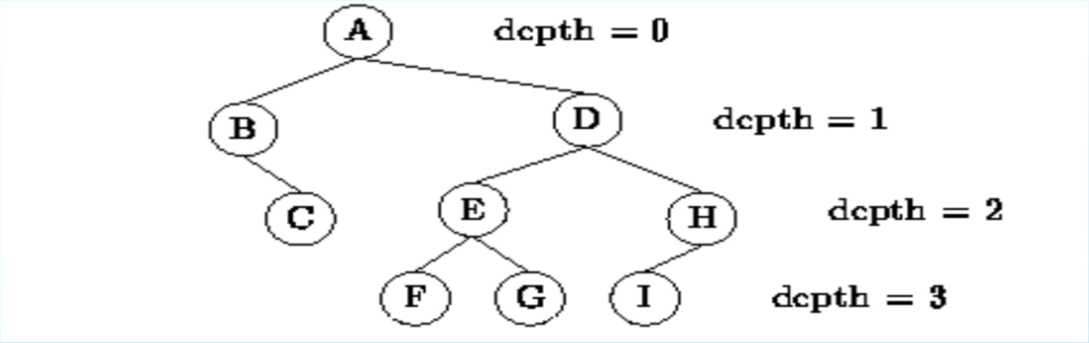
Height ?

HOW TO GET SIZE?



BREADTH-FIRST TRAVERSAL

- The breadth-first traversal of a tree visits the nodes in the order of their depth in the tree. Breadth-first traversal first visits all the nodes at depth zero (i.e., the root), then all the nodes at depth one, and so on. At each depth the nodes are visited from left to right.



A → B → D → C → E → H → F → G → I

Implementation

- Class **`TreeNode`**
  - Has data and two references (**`left`** and **`right`**)
- Method **`insert`**
  - Compares value to be inserted with data in root
  - Ignores duplicate values
  - If value < data
    - If left link is null, assign to new node (containing value)
    - Else recursively call insert for left node
  - If value > data
    - If right link null, assign to new node (containing value)
    - Else recursively call insert for right node

Implementation

- Implementation
  - Class **`Tree`**
    - Has reference to root node
    - Method **`insertNode( int data )`**
      - If root **`null`**, creates new **`TreeNode`** with data
      - Else, calls **`insert`**
  - **`PreorderTraversal()`**
    - Calls **`preorderHelper( root )`**
      - Prints node
      - Calls **`preorderHelper( node.left )`**
      - Calls **`preorderHelper( node.right )`**
  - **`PostorderTraversal, inorderTraversal`** similar

```java
// Fig. 22.16: Tree.java
package com.deitel.jhtp3.ch22;

// Class TreeNode definition
class TreeNode {
   // package access members
   TreeNode left;   // left node
   int data;        // data item
   TreeNode right;  // right node

   // Constructor: initialize data to d and make this a leaf
   public TreeNode( int d )
   {
      data = d;
      left = right = null;  // this node has no children
   }

   // Insert a TreeNode into a Tree that contains nodes.
   // Ignore duplicate values.
   public synchronized void insert( int d )
   {
      if ( d < data ) {
         if ( left == null )
            left = new TreeNode( d );
         else
            left.insert( d );
      }
```

`Tree` has two links, to left and right subtrees.

Insert checks value to insert (**`d`**) against its **`data`**. If no children, creates node there. If has children, recursively calls **`insert`**. Ignores duplicate entries.

```
      else if ( d > data ) {
         if ( right == null )
            right = new TreeNode( d );
         else
            right.insert( d );
      }
   }
}

// Class Tree definition
public class Tree {
   private TreeNode root;

   // Construct an empty Tree of integers
   public Tree() { root = null; }

   // Insert a new node in the binary search tree.
   // If the root node is null, create the root node here.
   // Otherwise, call the insert method of class TreeNode.
   public synchronized void insertNode( int d )
   {
      if ( root == null )
         root = new TreeNode( d );
      else
         root.insert( d );
   }

   // Preorder Traversal
   public synchronized void preorderTraversal()
      { preorderHelper( root ); }

   // Recursive method to perform preorder traversal
   private void preorderHelper( TreeNode node )
   {
      if ( node == null )
         return;

      System.out.print( node.data + " " );
      preorderHelper( node.left );
      preorderHelper( node.right );
   }

   // Inorder Traversal
   public synchronized void inorderTraversal()
      { inorderHelper( root ); }

   // Recursive method to perform inorder traversal
   private void inorderHelper( TreeNode node )
   {
      if ( node == null )
         return;

      inorderHelper( node.left );
      System.out.print( node.data + " " );
      inorderHelper( node.right );
   }

   // Postorder Traversal
   public synchronized void postorderTraversal()
      { postorderHelper( root ); }

```

Preorder traversal: print node, left, right.

inorder traversal: left, print node, right.

```java
   // Recursive method to perform postorder traversal
   private void postorderHelper( TreeNode node )
   {
      if ( node == null )
         return;

      postorderHelper( node.left );
      postorderHelper( node.right );
      System.out.print( node.data + " " );
   }
}
// Fig. 22.16: TreeTest.java
// This program tests the Tree class.
import com.deitel.jhtp3.ch22.Tree;

// Class TreeTest definition
public class TreeTest {
   public static void main( String args[] )
   {
      Tree tree = new Tree();
      int intVal;

      System.out.println( "Inserting the following values: " );

      for ( int i = 1; i <= 10; i++ ) {
         intVal = ( int ) ( Math.random() * 100 );
         System.out.print( intVal + " " );
         tree.insertNode( intVal );
      }

```

postorder traversal: left, right, print node.

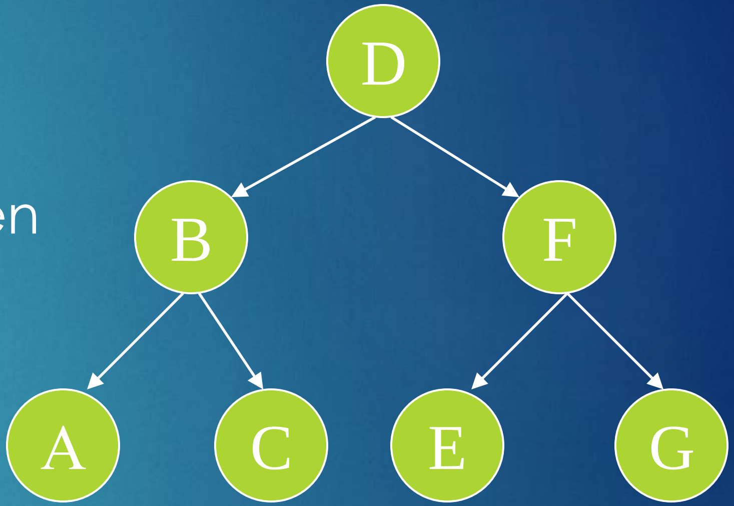
```
      System.out.println ( "\n\nPreorder traversal" );
      tree.preorderTraversal();

      System.out.println ( "\n\nInorder traversal" );
      tree.inorderTraversal();

      System.out.println ( "\n\nPostorder traversal" );
      tree.postorderTraversal();
      System.out.println();
   }
}
```

```
Inserting the following values:
39 69 94 47 50 72 55 41 97 73

Preorder traversal
39 69 47 41 50 55 94 72 73 97

Inorder traversal
39 41 47 50 55 69 72 73 94 97

Postorder traversal
41 55 50 47 73 72 97 94 69 39
```

PROGRAM OUTPUT

Binary Tree

- A **binary tree**
  - root
  - left subtree
  - right subtree
- Each node has at most two children

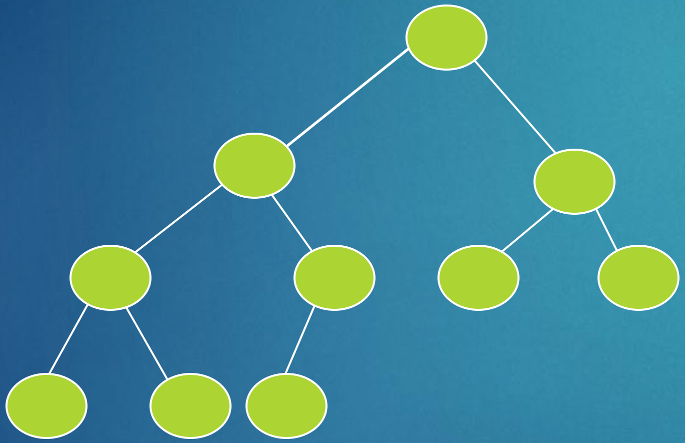


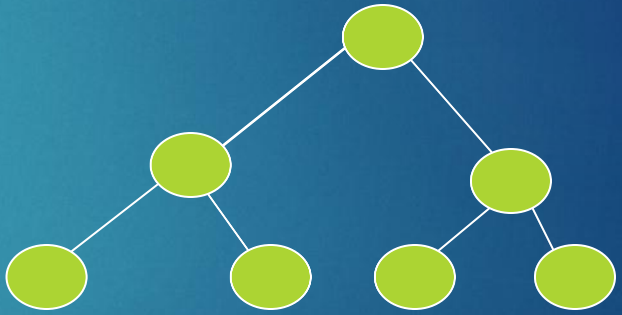
Complete Binary Tree (CBT)

- Complete Binary Tree (CBT)
  - Binary Tree
  - At level *i*, except last level, there are 2^i nodes
  - All nodes in the last level is as far left as possible

Examples for CBTs

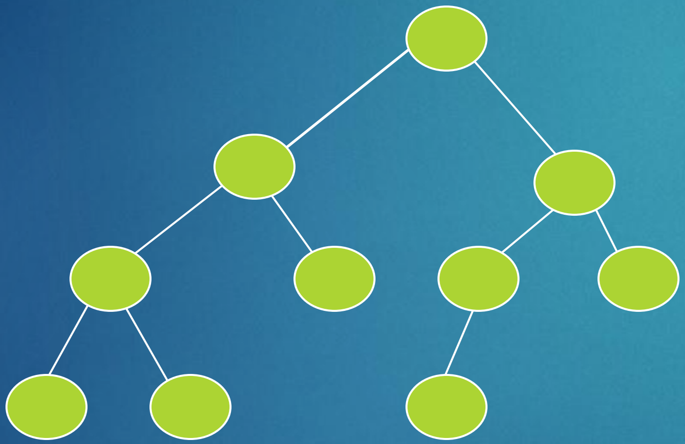
- These are CBTs

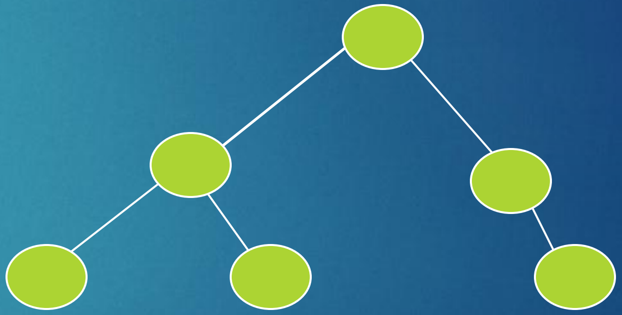




Examples for not CBTs

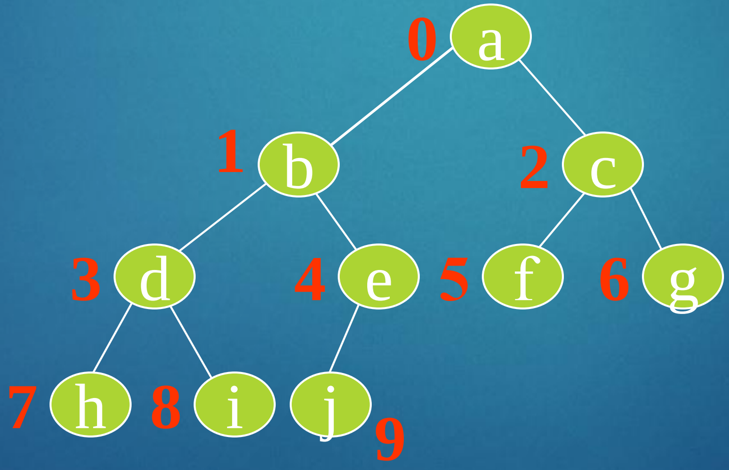
- These are not CBTs





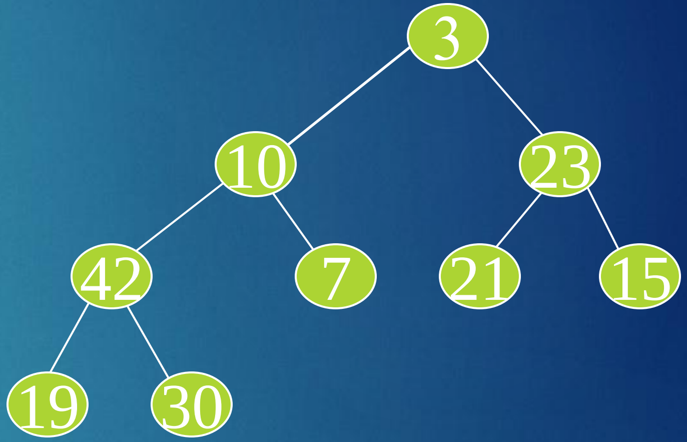
Relationship Between CBT And Array

- Enumerate the nodes from top to down, from left to right, we get the array
- array: a b c d e f g h i j
- index: 0 1 2 3 4 5 6 7 8 9



CBT To Array

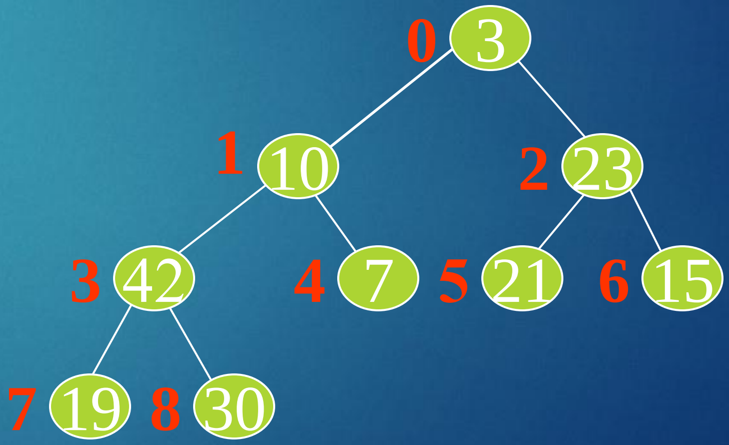
- Given a CBT, we can easily get an array
- CBT shown as left
- Get the elements from top to bottom, from left to right



- Then get Array
  - { 3, 10, 23, 42, 7, 21 , 15, 19, 30}

Array To CBT

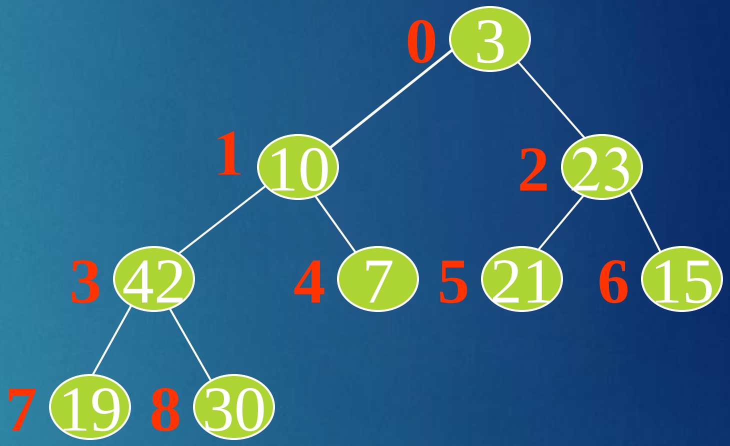
- Given an array, we can construct CBT
- Array: 3, 10, 23, 42, 7, 21, 15, 19, 30
- Put the elements from top to bottom, from left to right



Index In Array

- Array: 3, 10, 23, 42, 7, 21, 15, 19, 30
- Index: 0 1 2 3 4 5 6 7 8

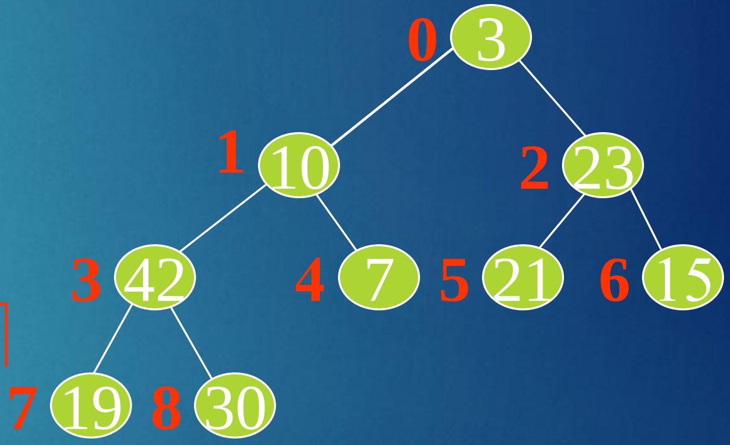
index	left child index	right child index
0	1	2
1	3	4
2	5	6
3	7	8
…	…	…
i	???	???



Left child index = 2*i+1 ?
Right child index =
2*i+2?

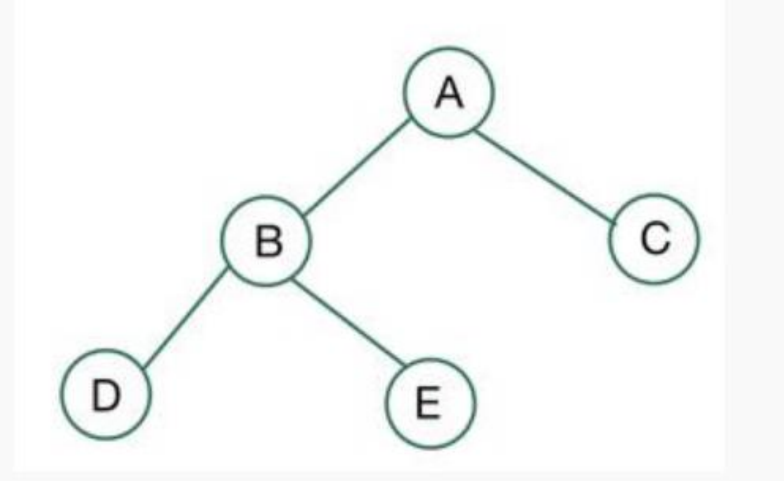
CBT Property

- Given the node with index *i*
  - parent's index is *(i-1)/2*
  - left child's index is *(2*i+1)*
  - right child's index is *(2*i+2)*
- total *n* nodes in CBT
  - height of CBT is $\lceil \log_2(n + 1) \rceil$



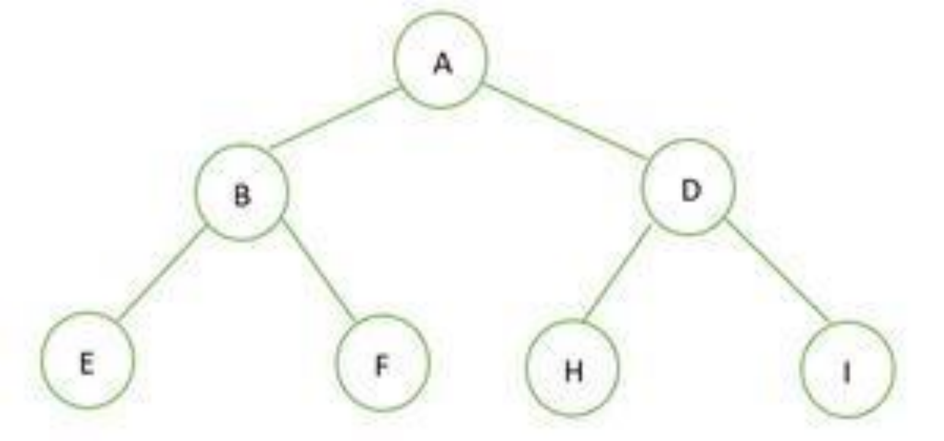
FULL BINARY TREE

- A full binary tree is a binary tree in which **all of the nodes have either 0 or 2 offspring**



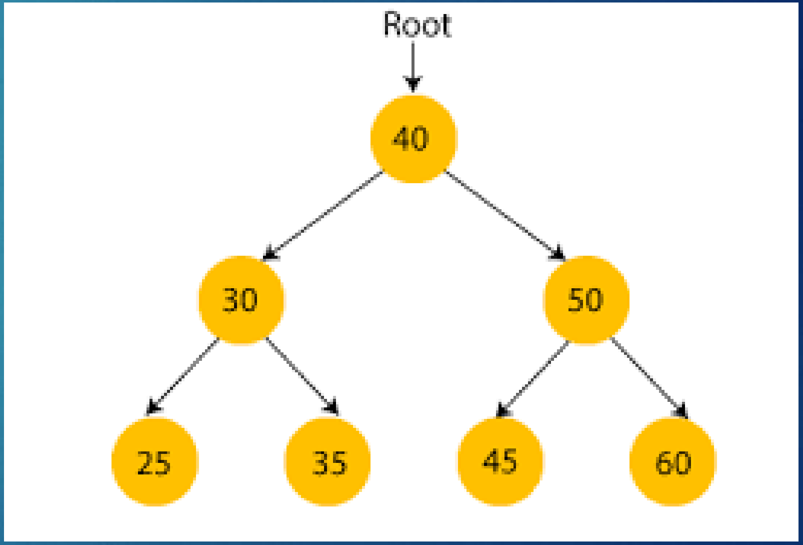
PERFECT BINARY TREE

- A binary tree of height ‘h’ having the maximum number of nodes is a perfect binary tree.
- For a given height h, the maximum number of nodes is 2^{h+1} - 1.(2 $^{2+1}$ -1 = 8 -1 = 7)



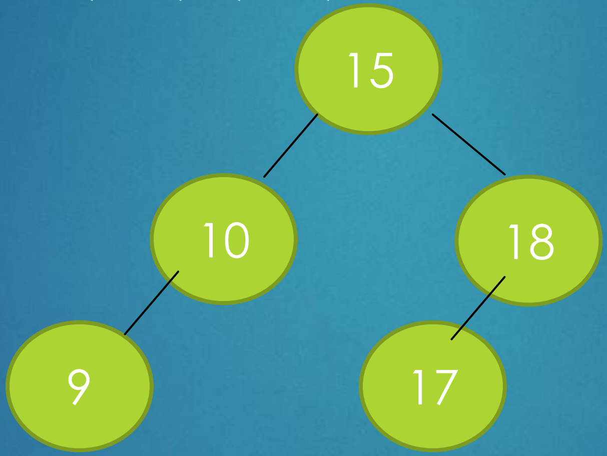
BINARY SEARCH TREE (BST)

- Cây nhị phân tìm kiếm: là cây mà mỗi node không có quá 2 node con.
- Cây tìm kiếm nhị phân là cây nhị phân mà:
  - Giá trị các nút thuộc cây con bên trái nhỏ hơn giá trị của nút cha.
  - Giá trị các nút thuộc cây con bên phải lớn hơn giá trị của nút cha.
- Duyệt cây nhị phân tìm kiếm giống như traversal tree
  - Inorder traversal
  - Preorder traversal
  - Postorder traversal



HOW TO INSERT IN BINARY SEARCH TREE?

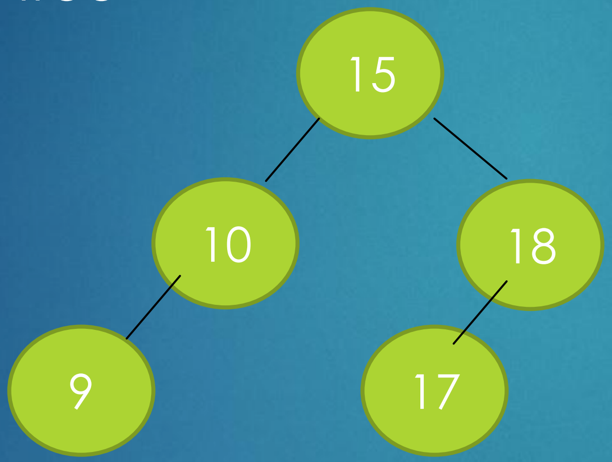
- Insert into empty tree
- Example: 15, 10, 9, 18, 17



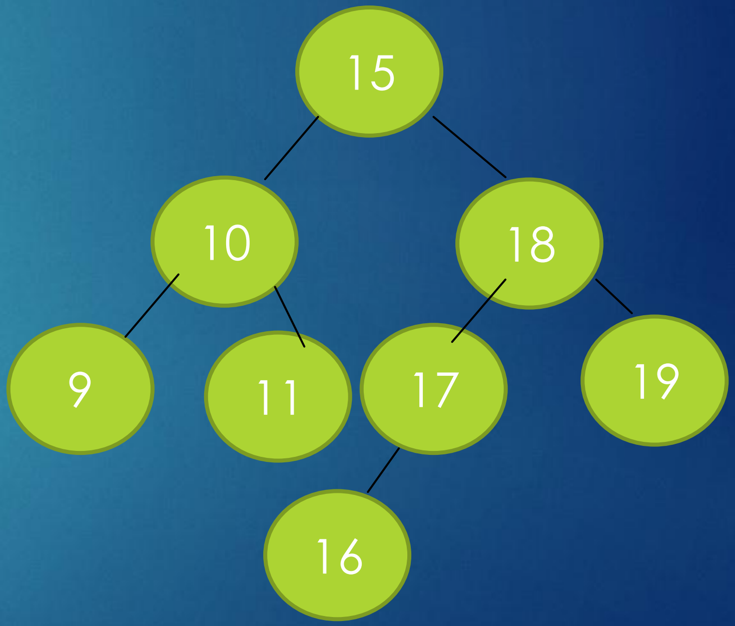
HOW TO INSERT IN BINARY SEARCH TREE?

- Insert 11, 16, 19 into

non_empty tree



- Result:



ORDER IN BINARY SEARCH TREE

Duyệt cây theo in Order, pre Order, post Order

in Order: (LNR)

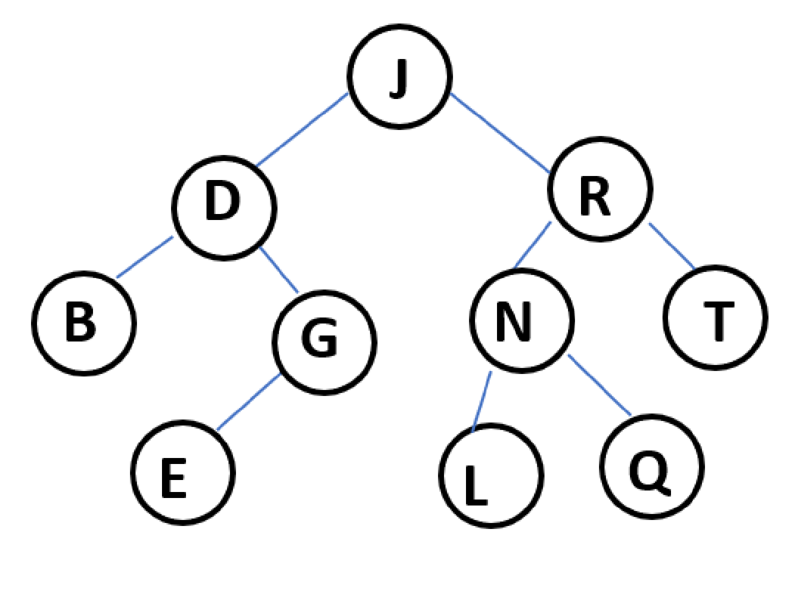
B D E G J L N Q R T

pre Order: (NLR)

J D B G E R N L Q T

post Order: (LRN)

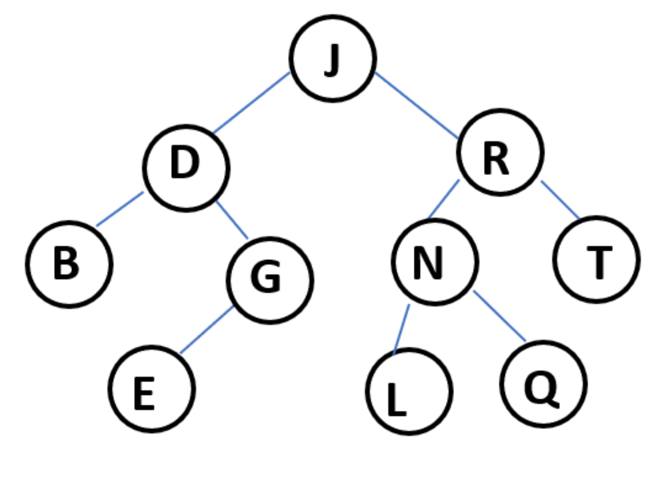
B E G D L Q N T R J



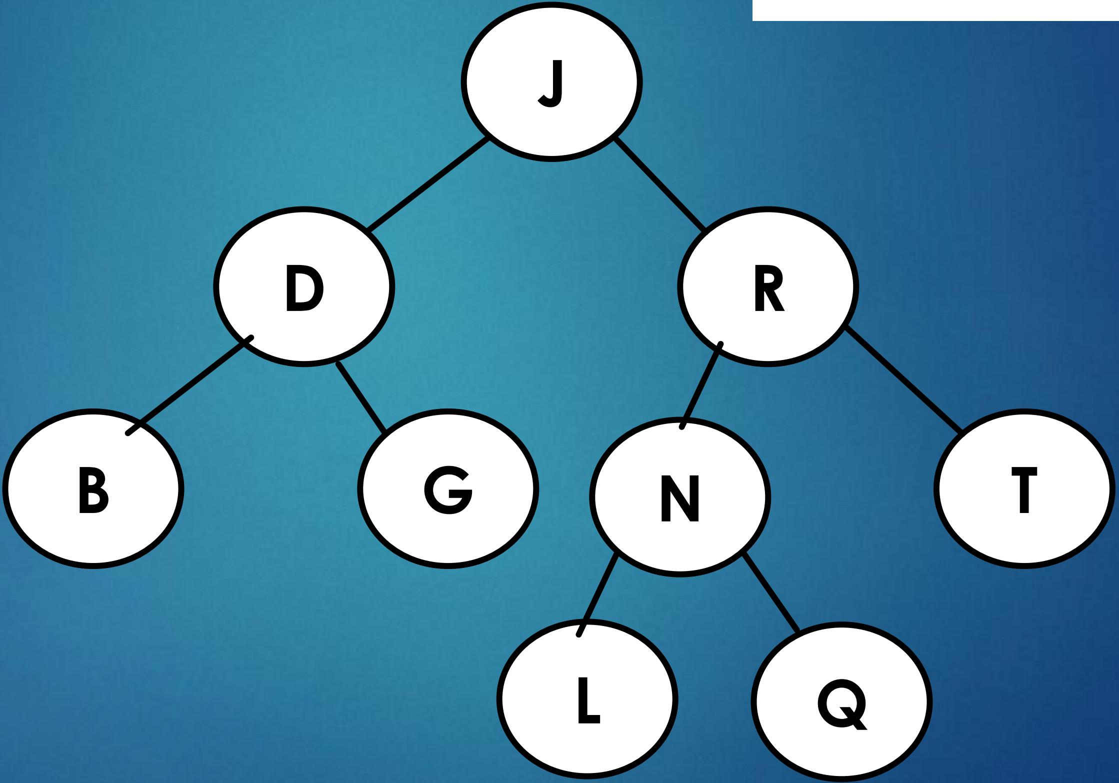
HOW TO DELETE IN BINARY SEARCH TREE?

- 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***).

EXAMPLE DELETE NODE IN BST

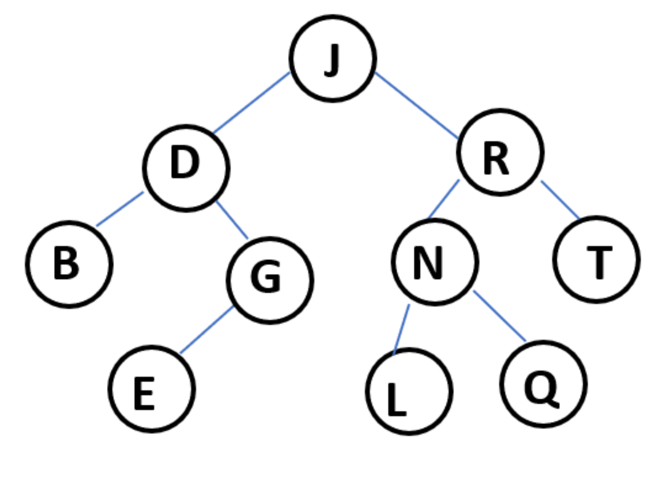


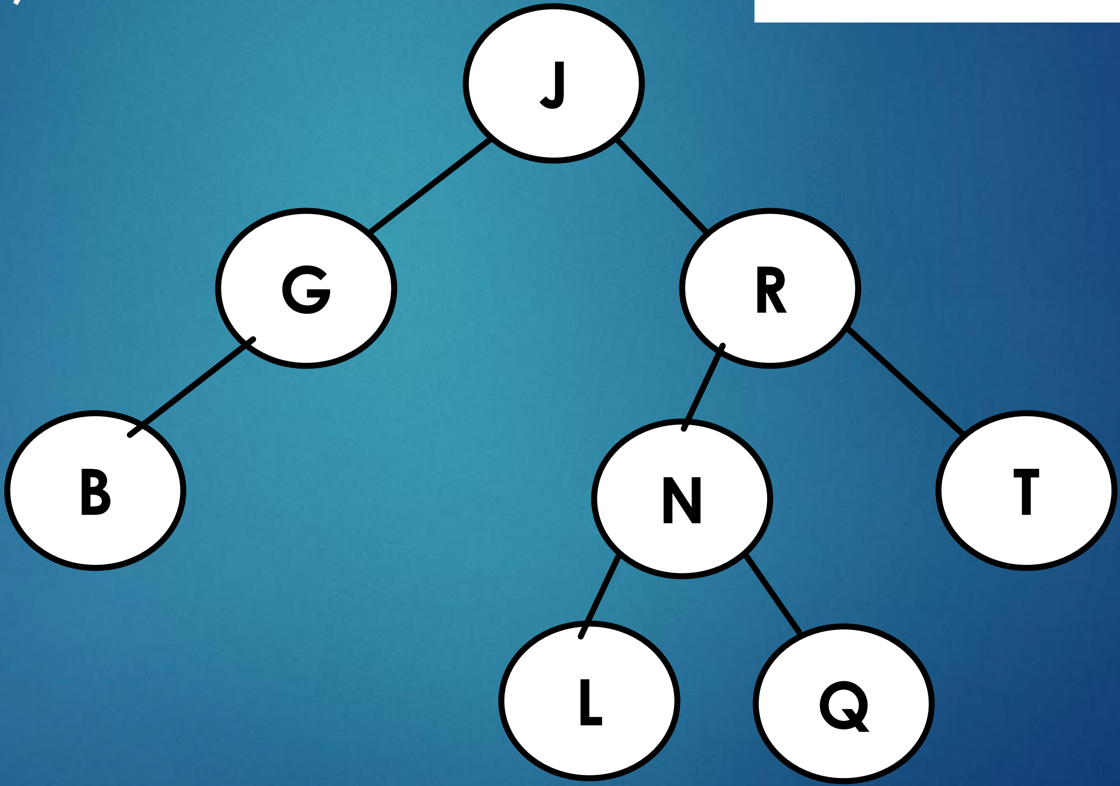
DELETE NODE **E**



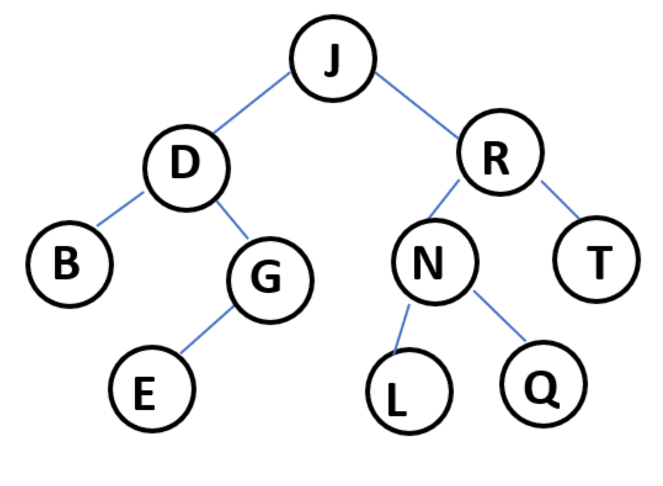
EXAMPLE DELETE NODE IN BST

DELETE NODE **E, D**

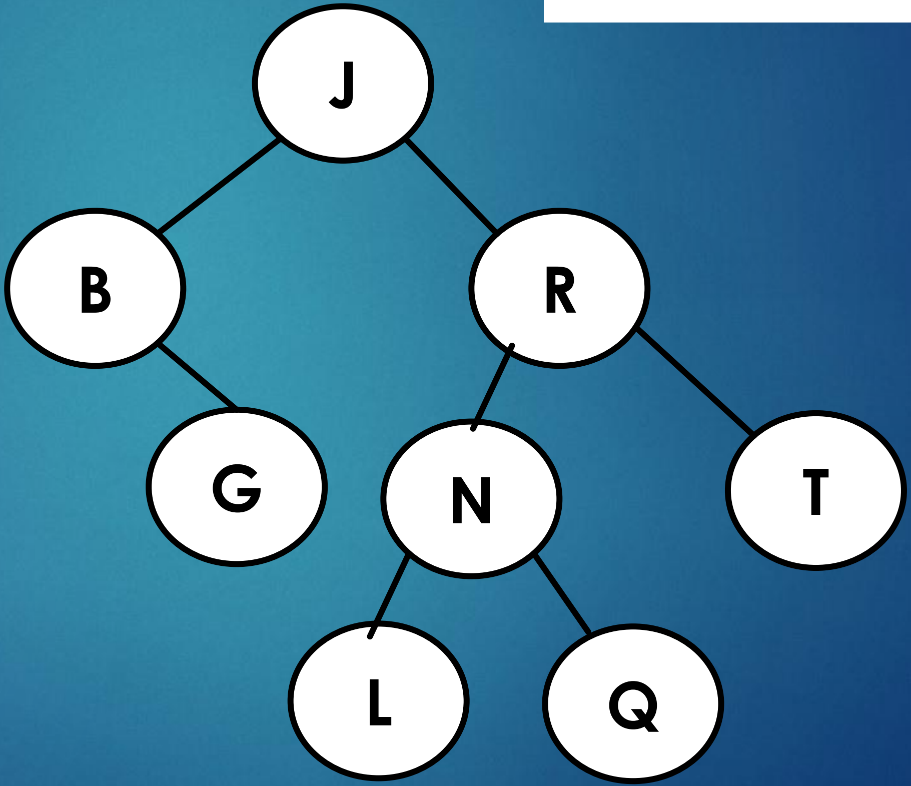


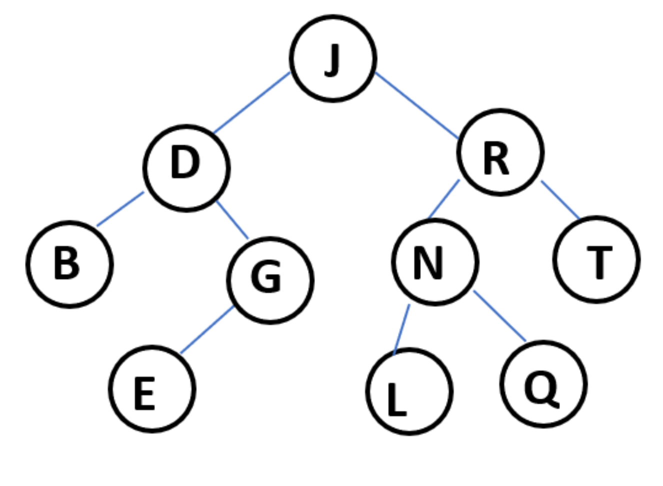


EXAMPLE DELETE NODE IN BST



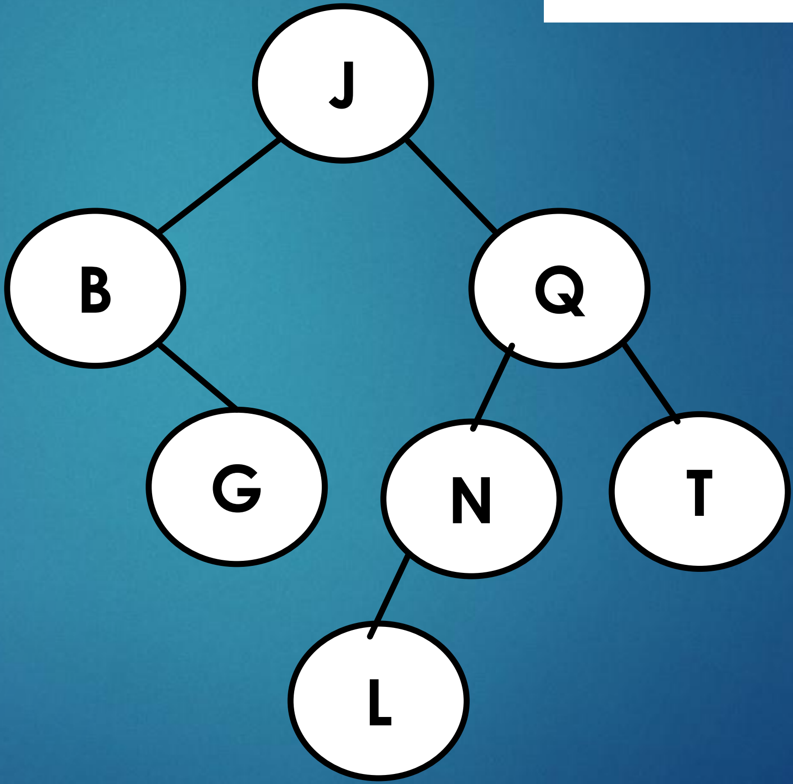
DELETE NODE **E, D**





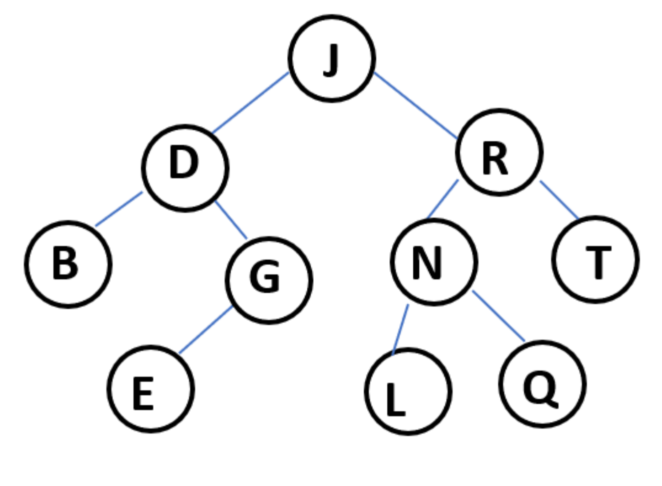
EXAMPLE DELETE NODE IN BST

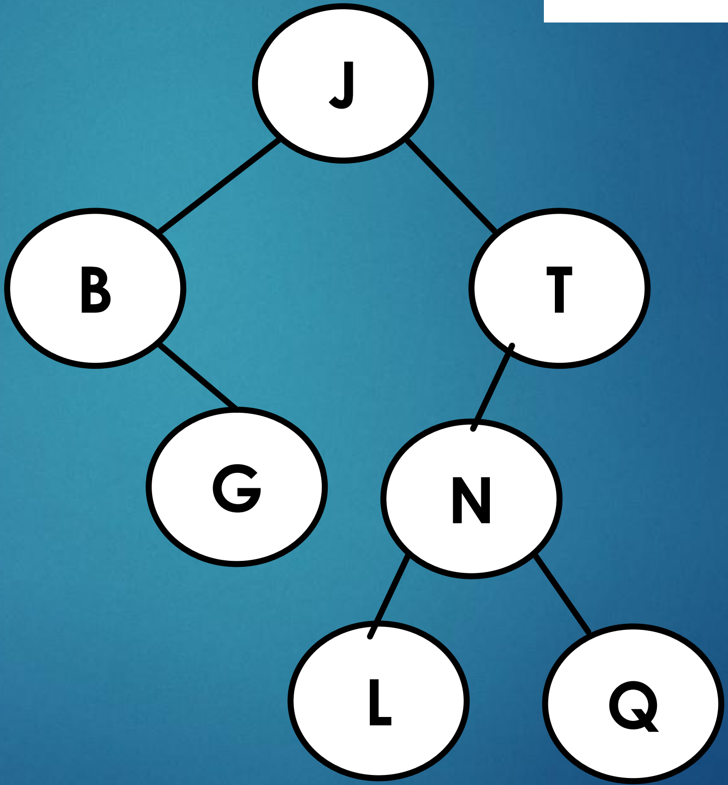
DELETE NODE **E, D, R**



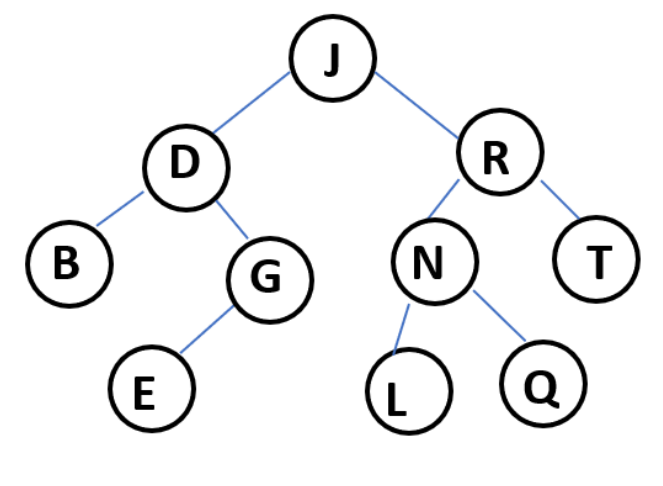
EXAMPLE DELETE NODE IN BST

DELETE NODE **E, D, R**

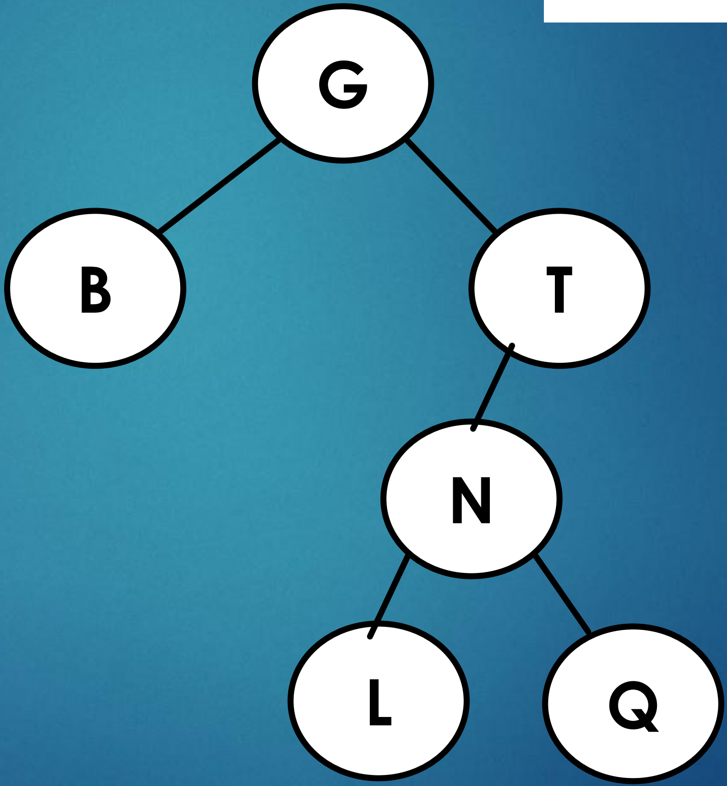




EXAMPLE DELETE NODE IN BST

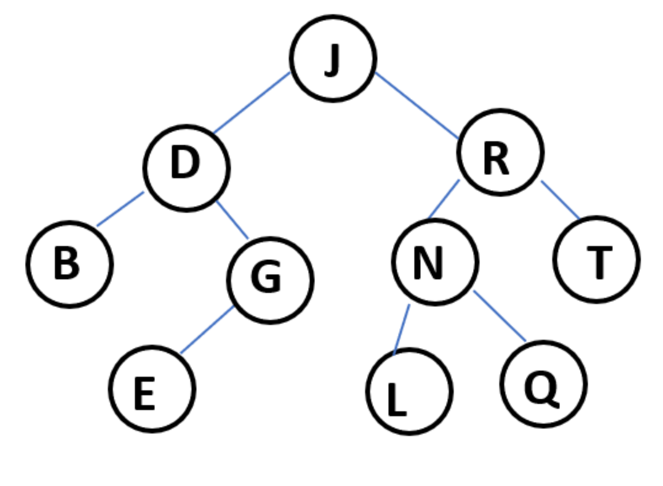


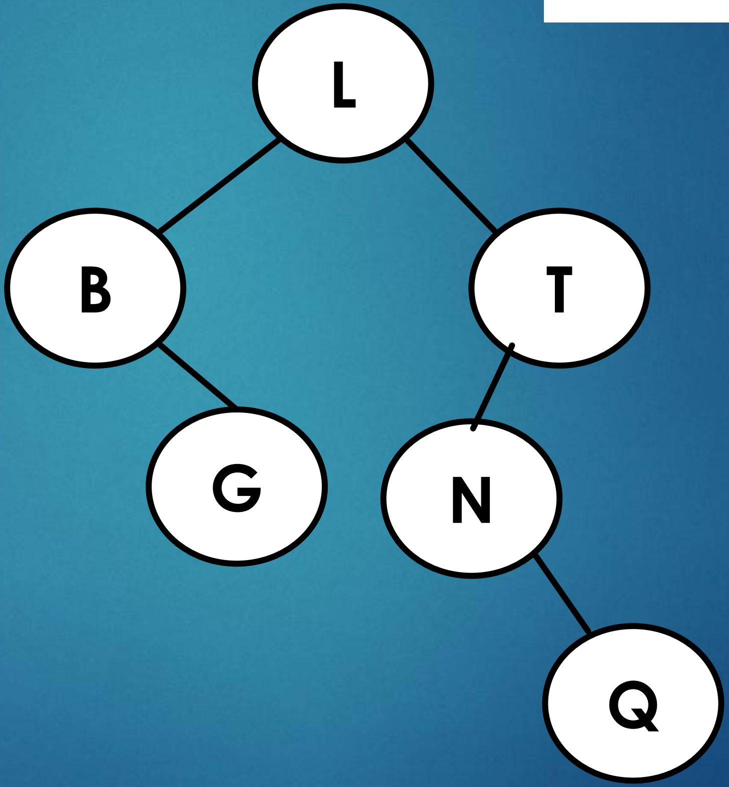
DELETE NODE **E, D, R, J**



EXAMPLE DELETE NODE IN BST

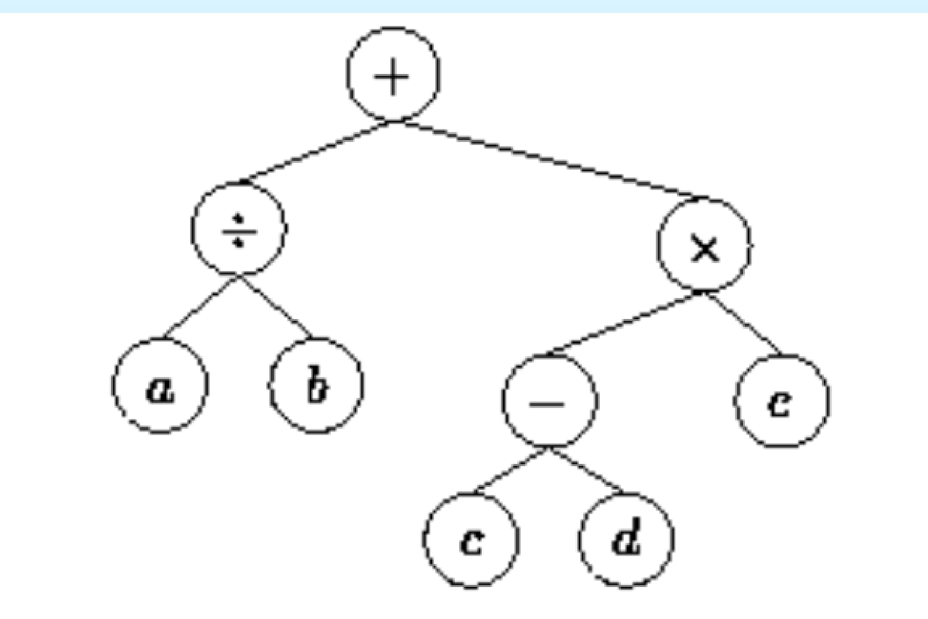
DELETE NODE **E, D, R, J**





EXPRESSION TREE (CÂY BIỂU THỨC)

- From expression :
- a / b + (c - d) * e
- To Expression Tree:



EXPRESSION TREE (CÂY BIỂU THỨC)

- a - b * (c + d)
- PRE ORDER: - a * b (+ c d)
- POST ORDER: a b (c d +) * -

HOW TO BUILD EXPRESSION TREE?

- Create ExpressionTree class

```
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 PRINT EXPRESSION TREE?

- Print Expr 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 CALCULATE EXPRESSION FROM EXPRESSION TREE?

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