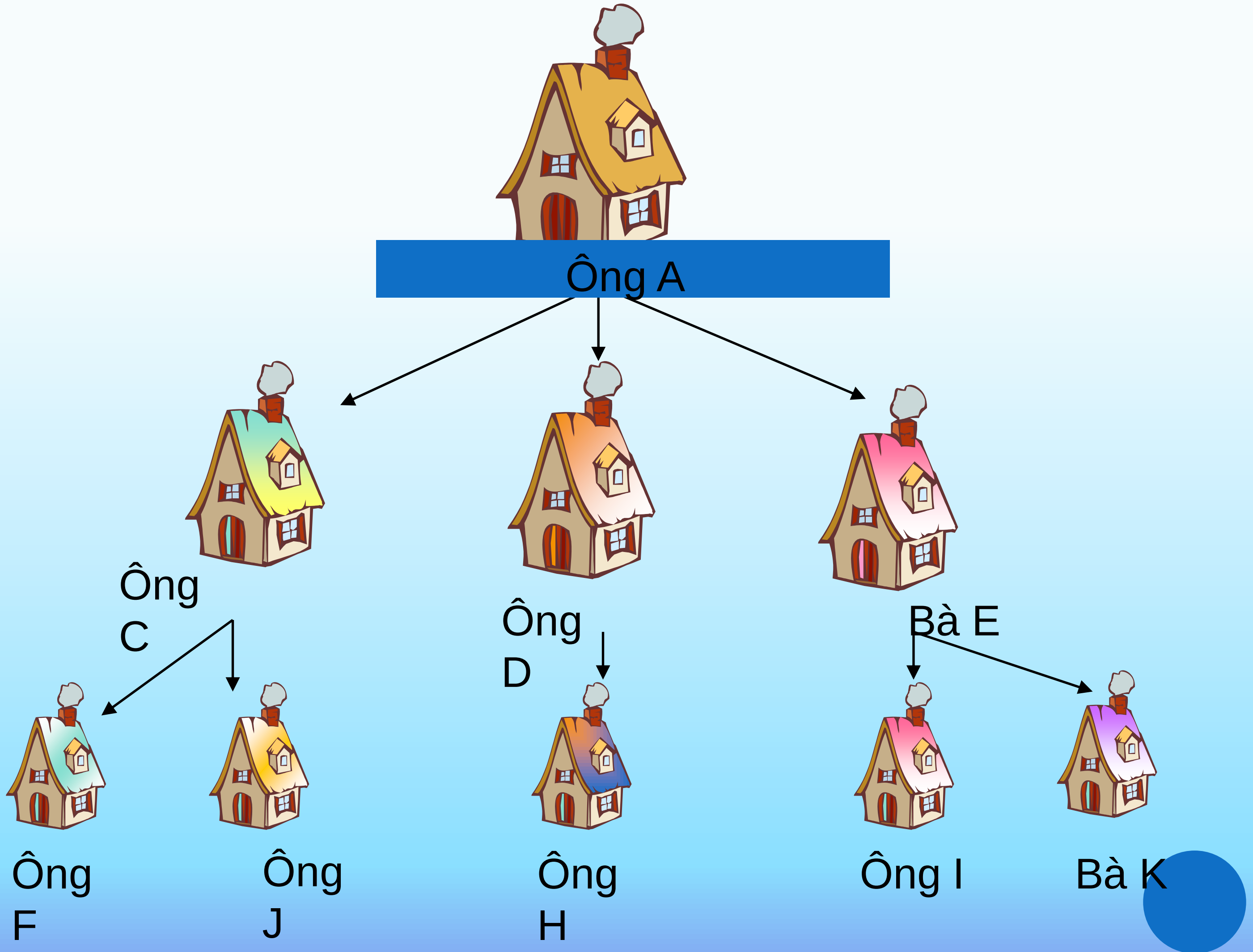
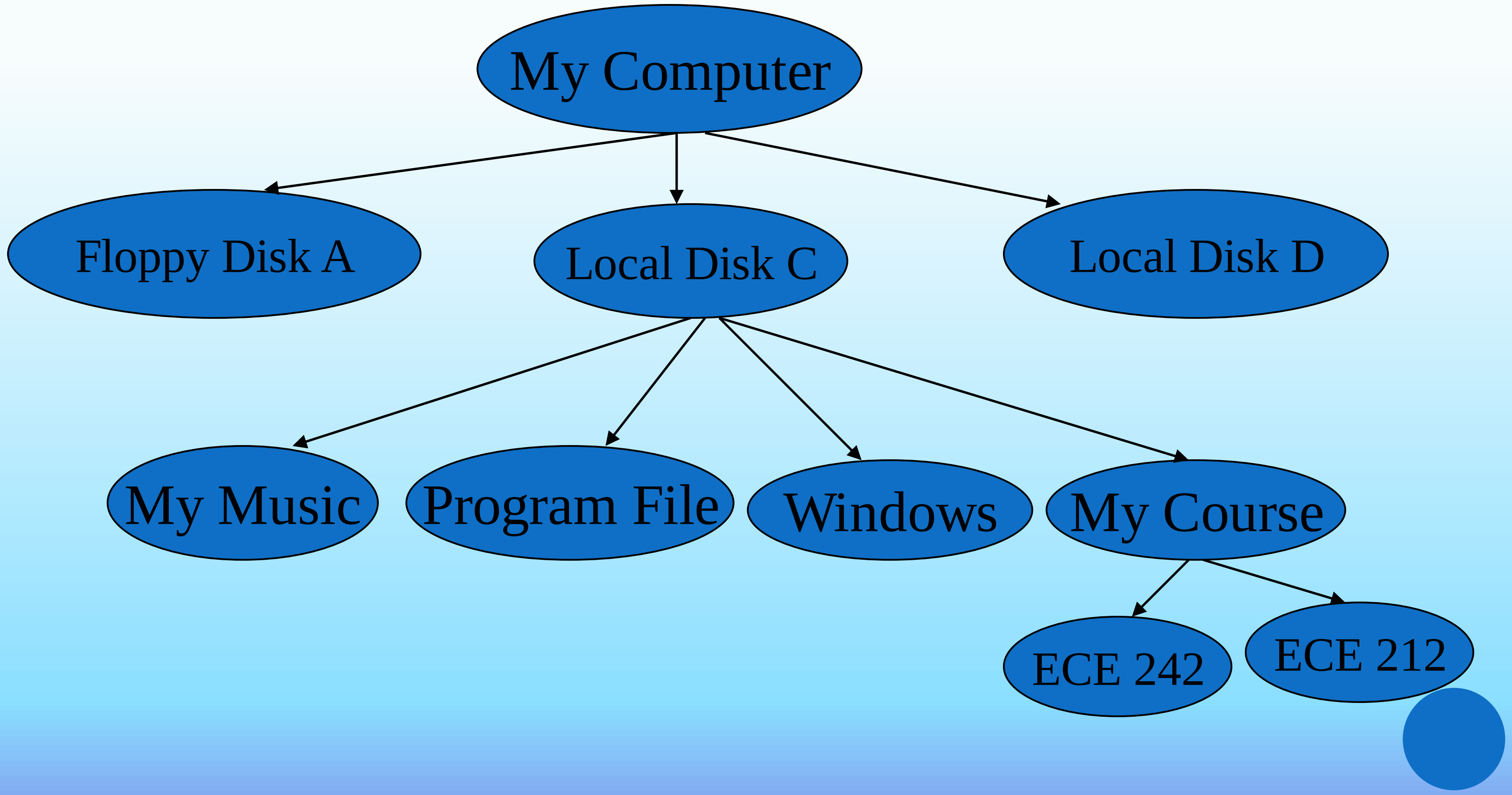
CÂY

CÂY GIA PHẢ

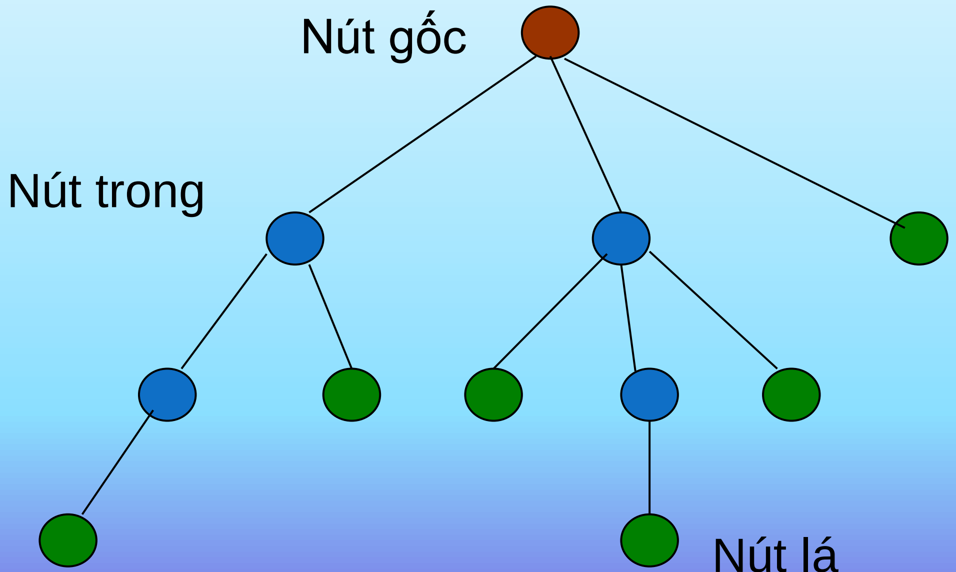


CÂY THƯ MỤC TRONG WINDOW



Formal Tree Definition

- A tree **T** as a set of nodes storing elements such that the nodes have a parent-child relationship that satisfies 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**



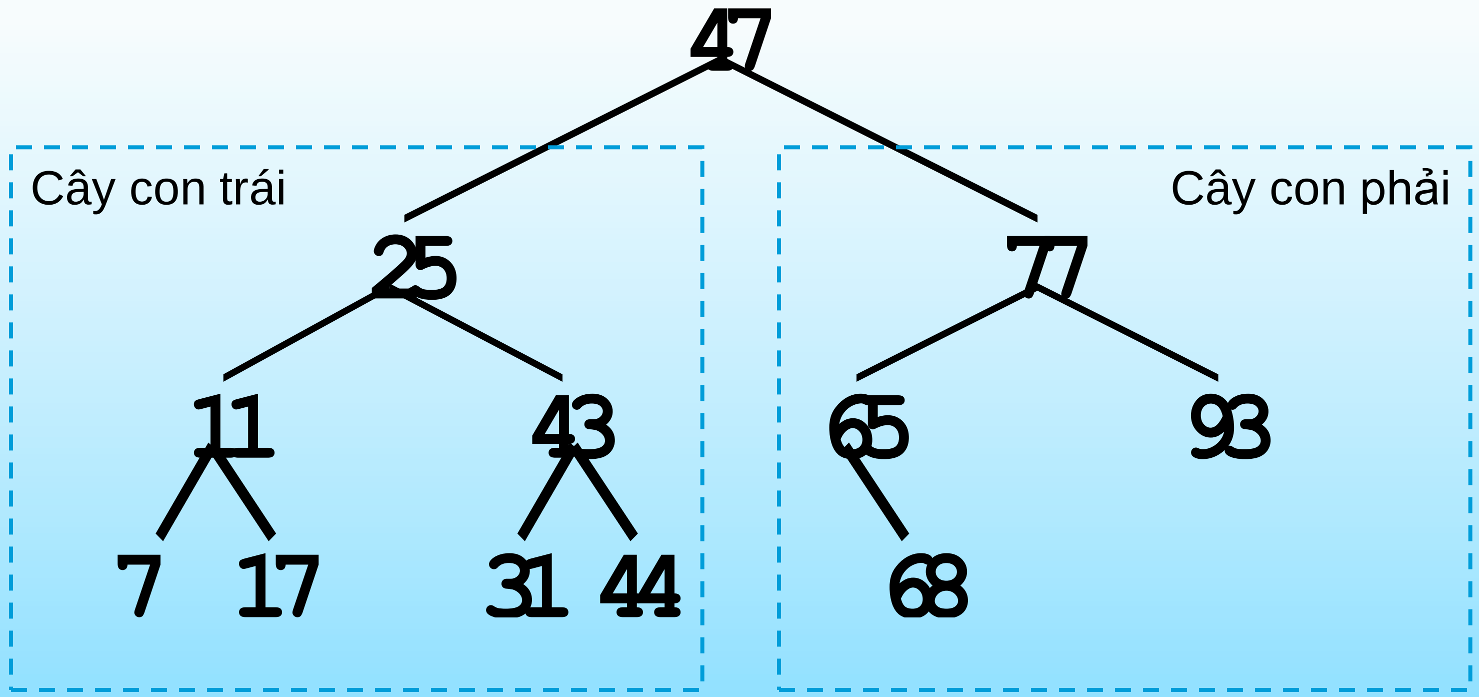
CÂY NHỊ PHÂN

CÂY NHỊ PHÂN

- Cây nhị phân 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
  - ✓ Inorder traversal
  - ✓ Preorder traversal
  - ✓ Postorder traversal

CÂY NHỊ PHÂN

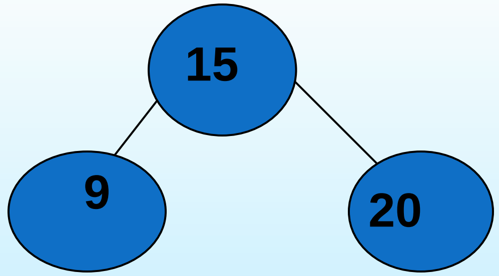
- Ví dụ về Binary Search Tree



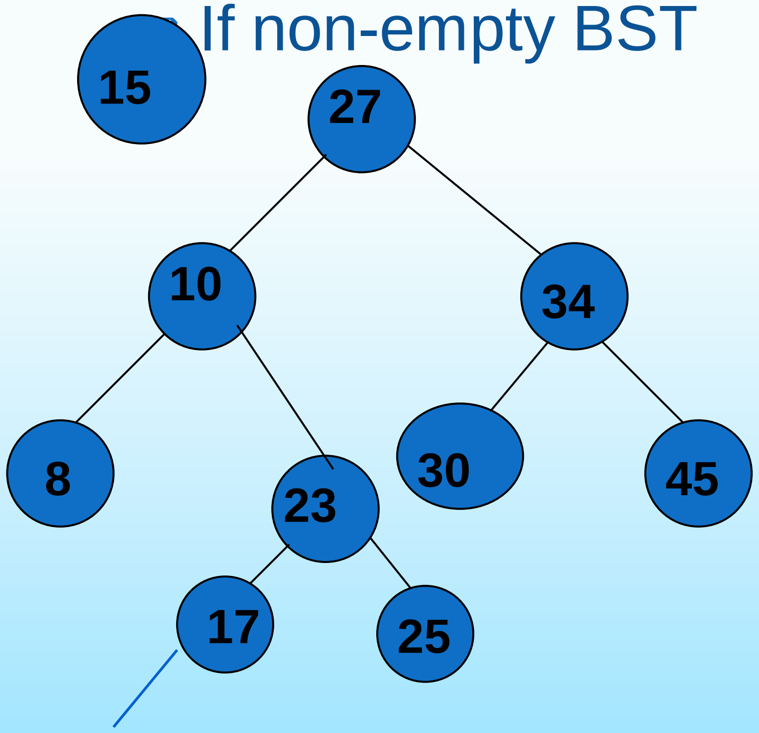
How to Insert ?

How to insert 15, 20, 9 into BST ?

- If Empty BST



- If non-empty BST

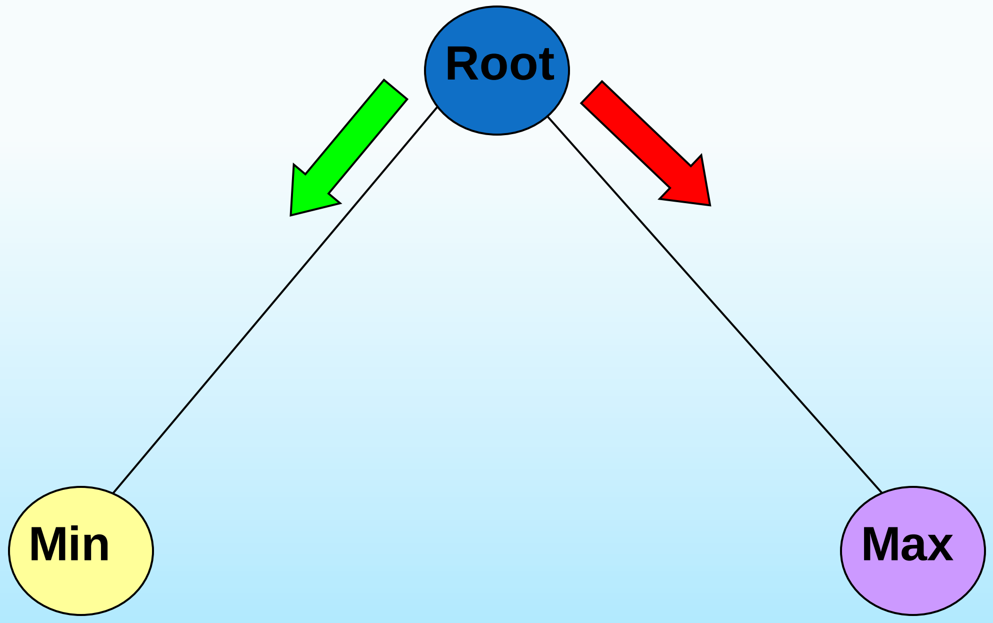


Could you try insert 20, 9 into BST ?

How to delete?

- **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).

Max and Min

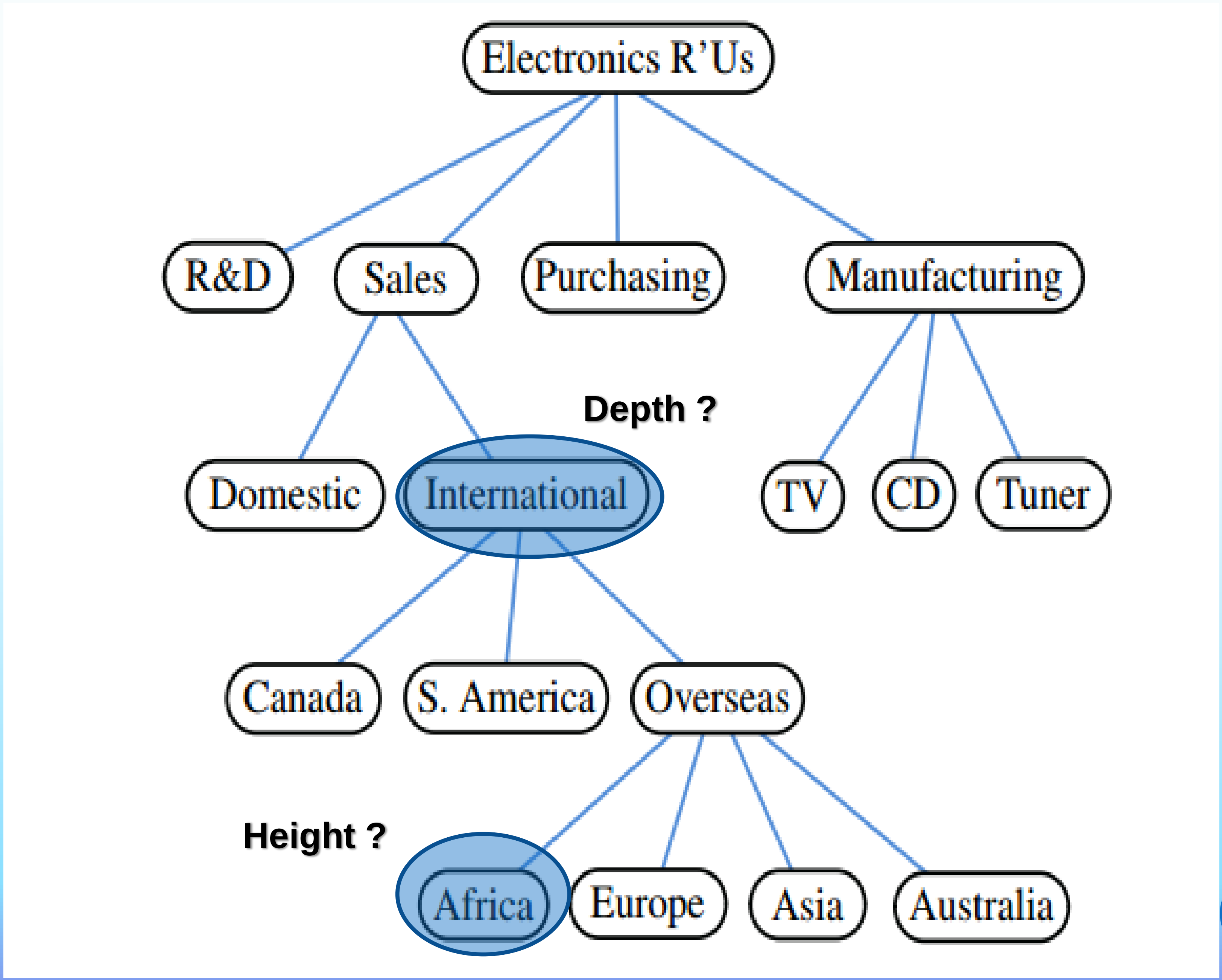


DEPTH

- 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.
  - ✓ Otherwise, 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
Depth ?
Domestic
International
TV
CD
Tuner
Canada
S. America
Overseas
Height ?
Africa
Europe
Asia
Australia

EXERCISES

1. Build insert(int insertedNumber) method, return void
2. Build insert(List<Integer> insertedNumbers) method, return void
   or insert(Integer[] insertedNumbers)
3. Build isContain(int inputNumber)method, return boolean
4. *Build isContain(BSTree tree) method, return boolean
5. Build greaterThan(int compareNum) method, return the list or array of numbers > compareNum.
6. Build lessThan(int compareNum) method, return the list or array of numbers < compareNum.
7. Build delete(int delNum) method, return void
8. Build delete(List<Integer> delNums) method, return void
   or delete(int[] insertedNumbers)
9. *Build deleteAtBegin(int delNum) method, return void
10. Build max(), min() methods
11. Build depth(int node)
12. Build height ()

CÀI ĐẶT BINARY SEARCH TREE

```
public class TreeNode
{
    int data;
    TreeNode leftNode, rightNode;
    public TreeNode( int nodeData )
    {
        data = nodeData;
        leftNode = rightNode = null;
    }
    public void insert( int value )
    {
       if ( value < data ) {
           if (leftNode == null)  leftNode = new TreeNode(value);
           else leftNode.insert( value );
       } else if ( value > data ) {
           if ( rightNode == null ) rightNode = new TreeNode(value);
           else rightNode.insert( value );
       }
    }
}
```

CÀI ĐẶT BINARY SEARCH TREE

```
public class Tree
{
    private TreeNode root;

    public Tree()
    {
        root = null;
    }

    public void insertNode( int insertValue )
    {
        if ( root == null )
            root = new TreeNode( insertValue );
        else
            root.insert( insertValue );
    }

    public void preorderTraversal()
    {
        preorder( root );
    }
```

CÀI ĐẶT BINARY SEARCH TREE

```
public void inorderTraversal()
{
    inorder( root );
}

public void postorderTraversal()
{
    postorder( root );
}

private void preorder( TreeNode node )
{
    if ( node == null )
        return;
    System.out.print( node.data + " " );
    preorder( node.leftNode );
    preorder( node.rightNode );
}
```

CÀI ĐẶT BINARY SEARCH TREE

```
    private void inorder( TreeNode node )
    {
        if ( node == null )
            return;
        inorder( node.leftNode );
        System.out.print( node.data + " " );
        inorder( node.rightNode );
    }

    private void postorder( TreeNode node )
    {
        if ( node == null )
            return;
        postorder( node.leftNode );
        postorder( node.rightNode );
        System.out.print( node.data + " " );
    }
}
```

Sử dụng Binary Search Tree

```
public class TreeTest
{
    public static void main( String[] args )
    {
        Tree tree = new Tree();
        int value;

        for ( int i = 1; i <= 10; i++ ) {
            value = ( int ) ( Math.random() * 100 );
            tree.insertNode( value );
        }


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

BÀI TẬP TẠI LỚP

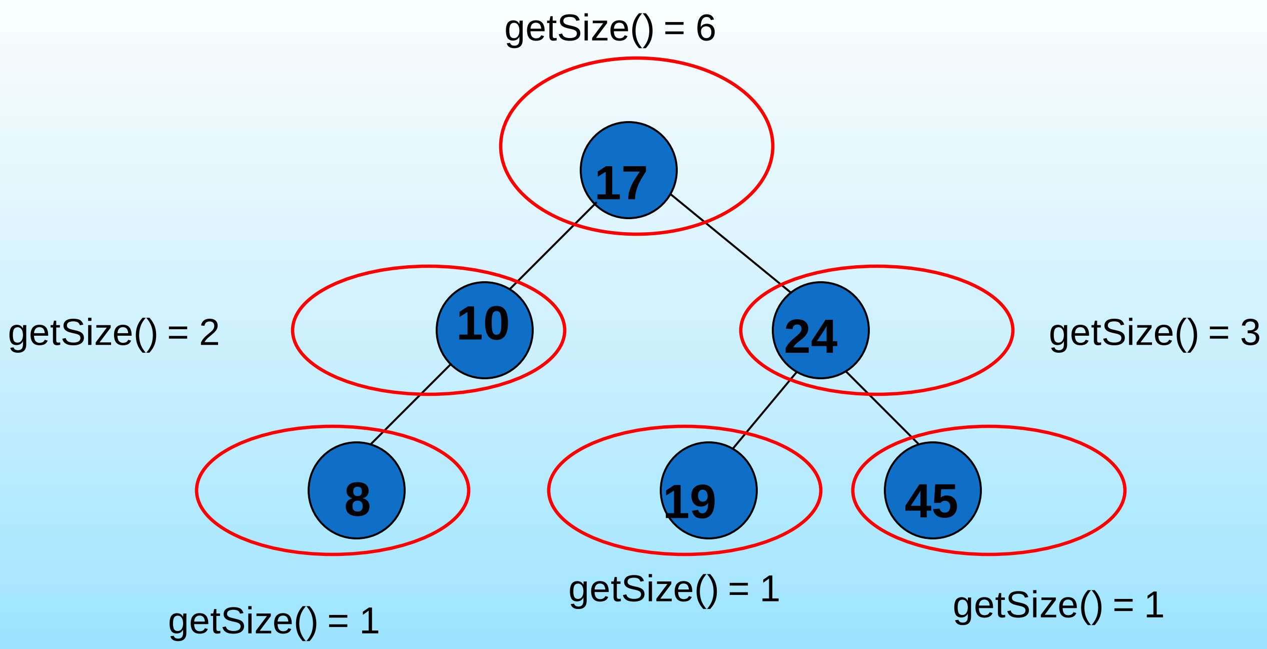
- Bài 1: Dùng Stack để viết chương trình in ra dạng nhị phân của một số nguyên dương cho trước.
- Bài 2: Cài đặt phương thức search trong lớp Tree để tìm một phần tử có giá trị cho trước.

How to build Tree Traversal?

```
public class TraversalTree {
    private String node;
    private TraversalTree left;
    private TraversalTree right;

    private TraversalTree Tree(String node, TraversalTree left,
      TraversalTree right) {
    this.node = node;
    this.left = left;
    this.right = right;
}
```

How to get size?



Ex0: buid getSize() method

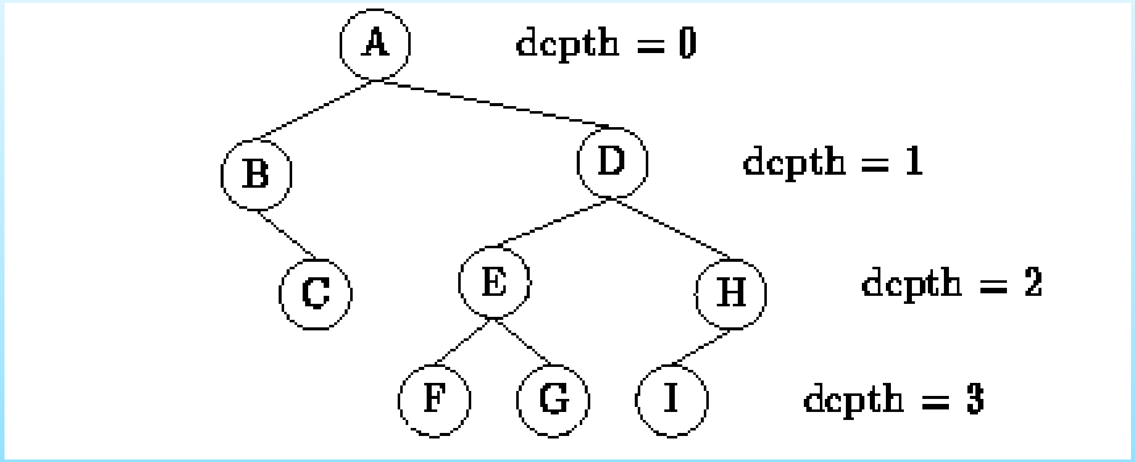
Pre Order, In Order, Post Order

- Preorder, where we visit the node, then visit the left and right subtrees
- Inorder, where we visit the left subtree, then visit the node, then visit the right subtree
- Postorder, where we visit the left and right subtrees, then visit the node

Ex1: buid printPreOrder(), printInOrder(), printPostOrder() methods

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

Ex2: write a method getNodesByDepth(), return List or Array node

Expression Tree

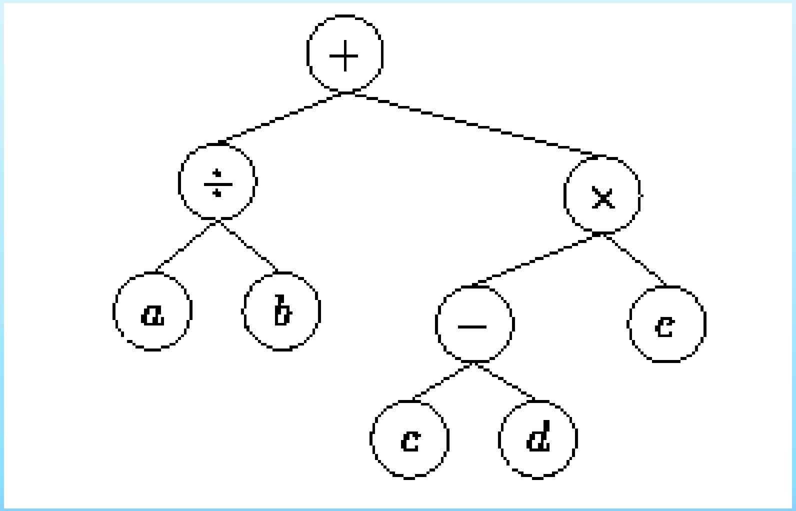
(Cây biểu thức)

What's Expression Tree ?

- From expression :

  a / b + (c - d) * e

  To Expression Tree:



Ex0 : Build the method to change from expression to expression tree

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

Ex1: Build the method to print Expression from Expression Tree

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
        …………………………………..
}
```

Ex2: build getTotal method to calculate expression tree (use recursive)