

LECTURE 14

TREE



pythonTM

Big-O Coding

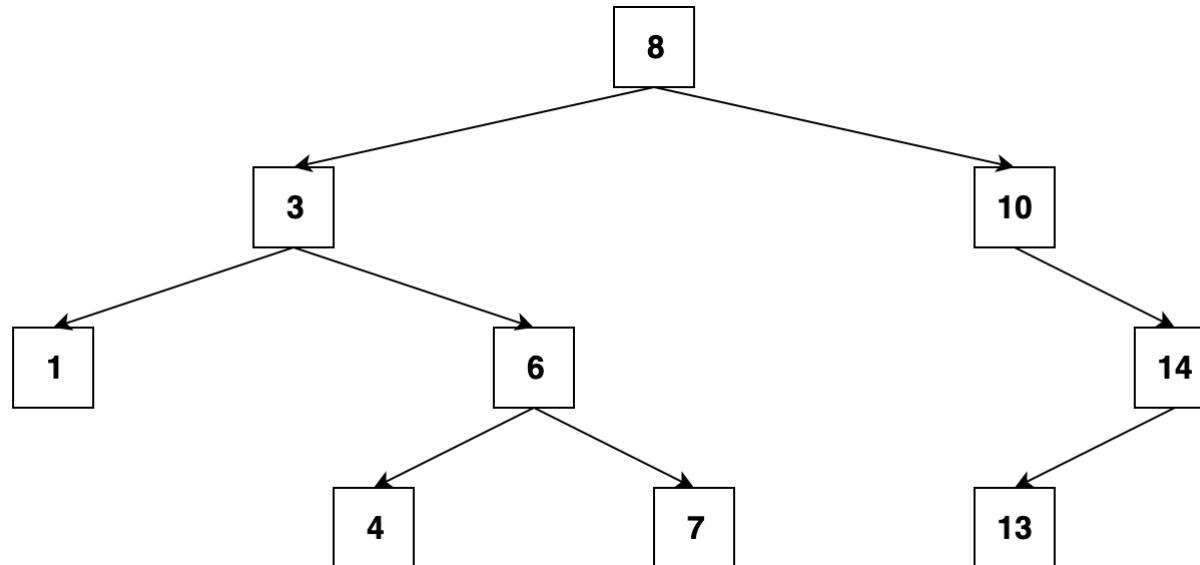
Website: www.bigocoding.com

Tree trong cuộc sống

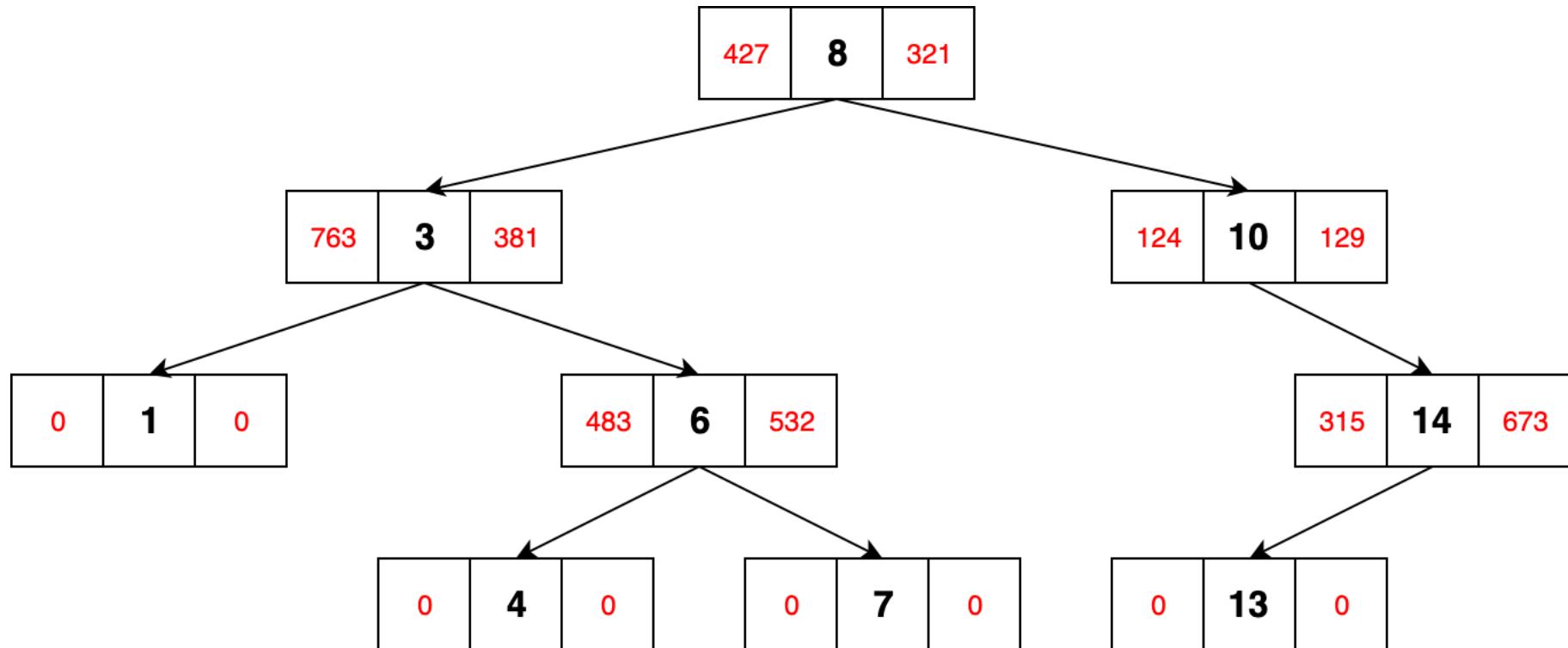


Tree trong lập trình

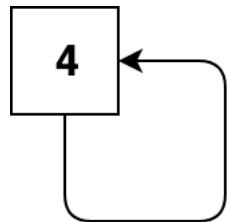
- **Tree** (cây) là **hierarchical** data structure (cấu trúc dữ liệu phân cấp), được tạo thành từ các **node**, bắt đầu từ node **root** (node gốc).
- Mỗi **node** bao gồm:
 - **data** (dữ liệu): dữ liệu cần lưu trữ trên mỗi node.
 - **branch** (nhánh): các nhánh xuất phát từ node đó.



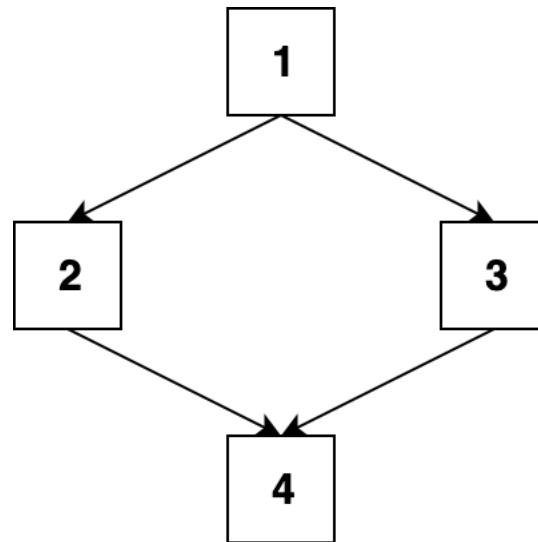
Tree trong lập trình



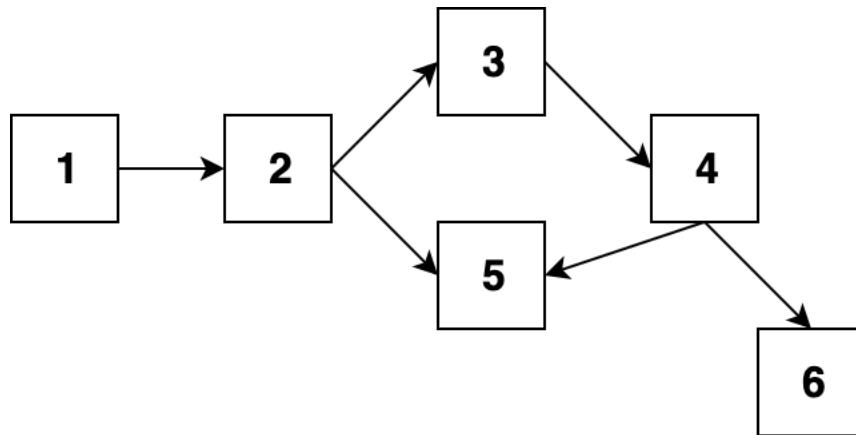
Not a tree



Not a tree

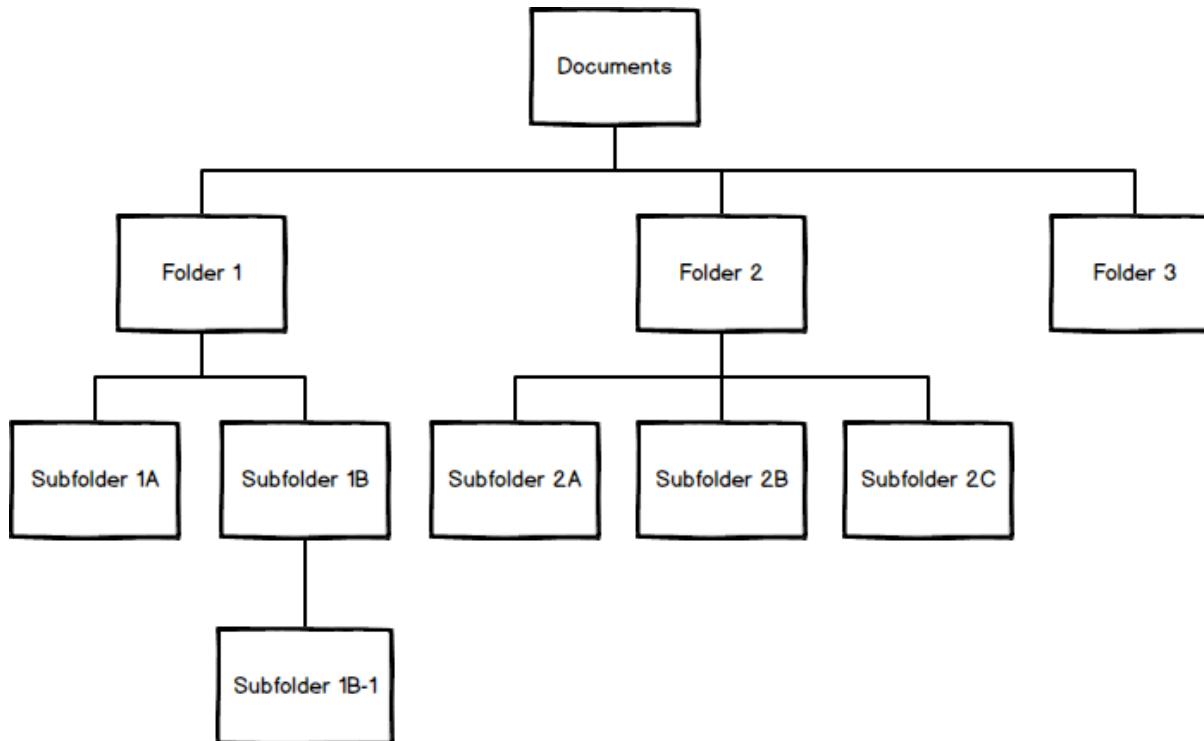


Not a tree



Ứng dụng của tree

- Tree dùng để lưu trữ dữ liệu dạng phân cấp.

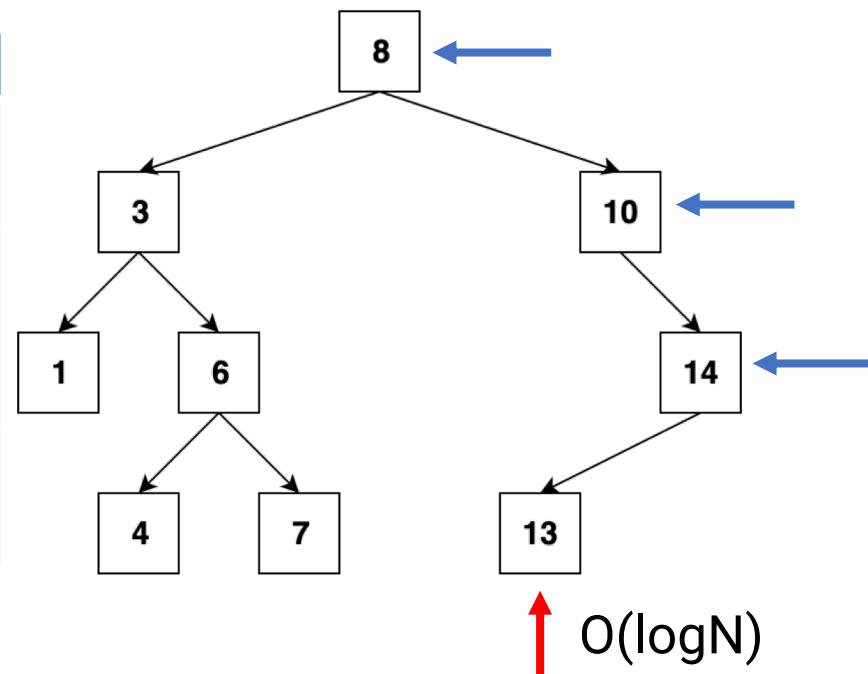


Ứng dụng của tree

- Tổ chức dữ liệu dạng tree để tìm kiếm dữ liệu nhanh hơn.

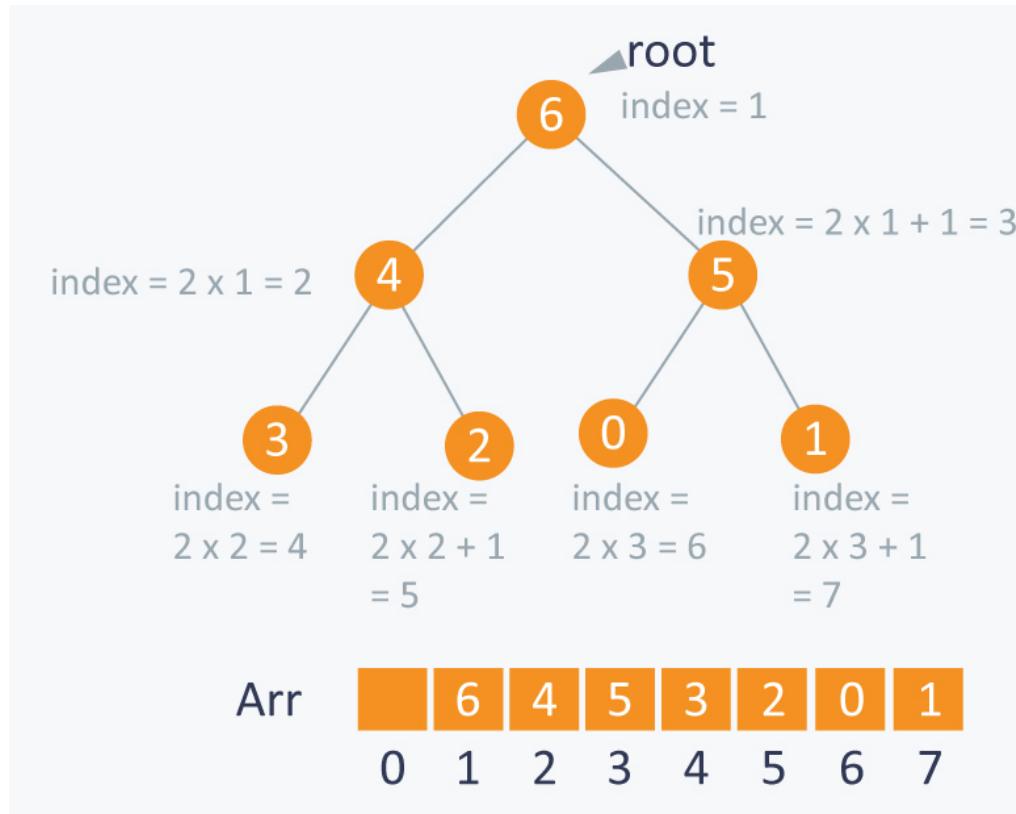


N	$\log N$
1,000	9.966
10,000	13.288
100,000	16.610
1,000,000	19.932
10,000,000	23.253
100,000,000	26.575
1,000,000,000	29.897



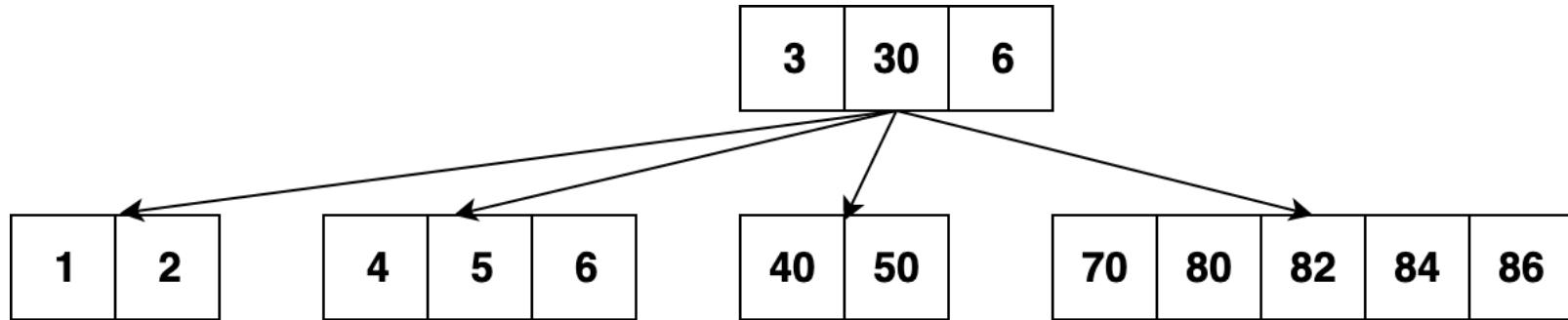
Ứng dụng của tree

- Heap: lưu trữ dạng mảng, được dùng làm priority queues.



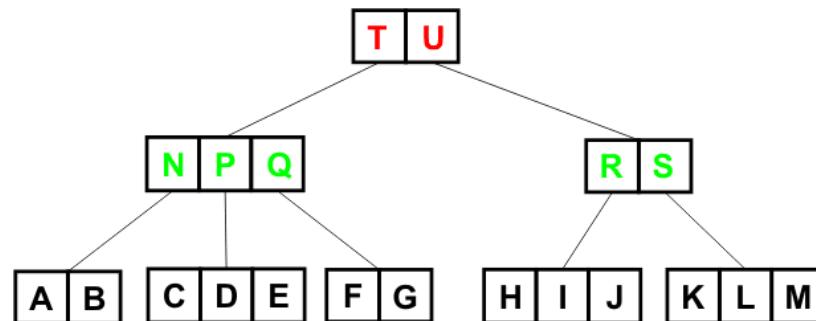
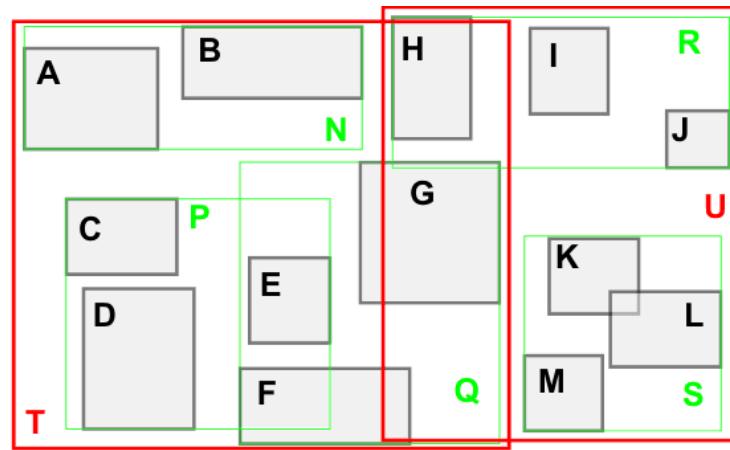
Ứng dụng của tree

- B tree: database indexing.



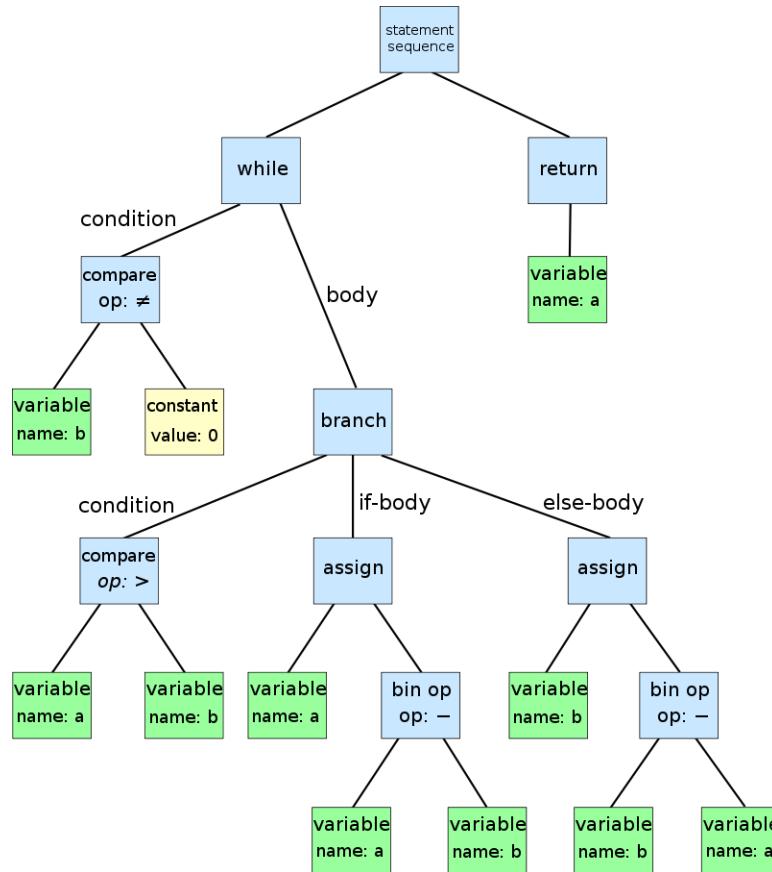
Ứng dụng của tree

- R-tree: spatial data indexing.



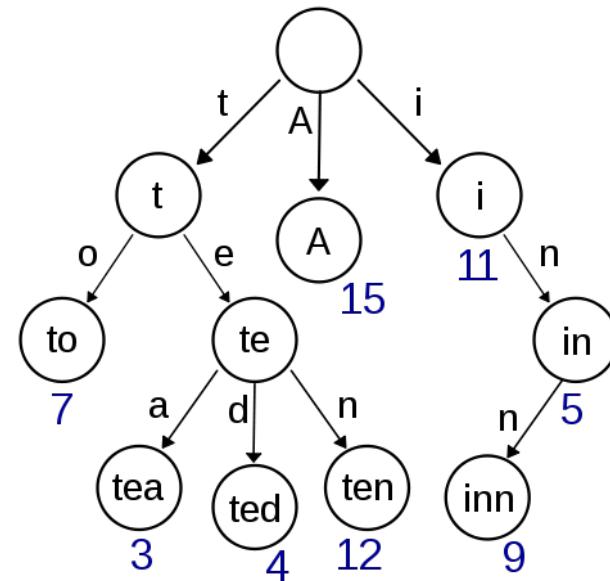
Ứng dụng của tree

- Syntax tree: được sử dụng trong các trình biên dịch.



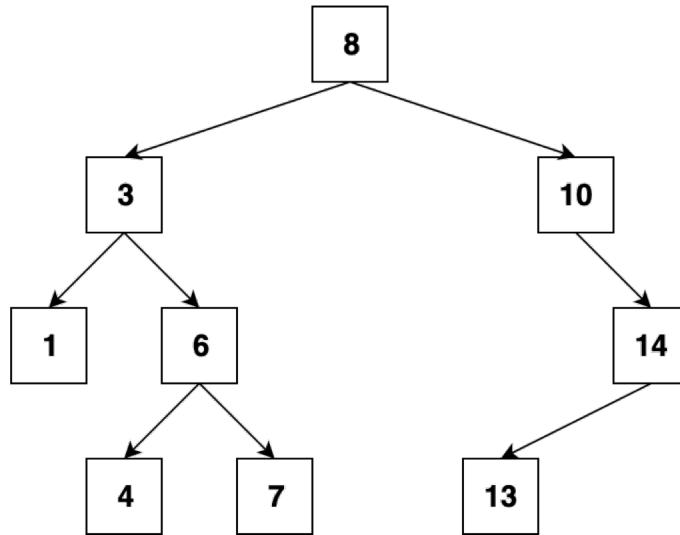
Ứng dụng của tree

- Trie: gợi ý từ trong các công cụ tìm kiếm.



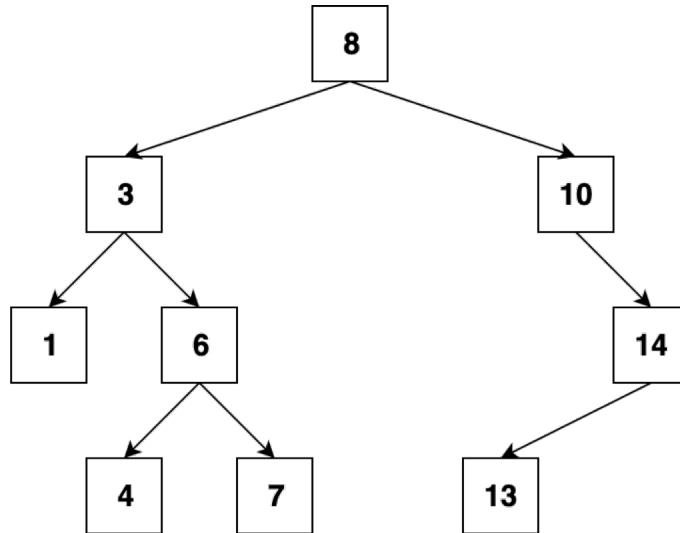
Root

- **Root:** node đầu tiên trong một tree. Mỗi tree có 1 root duy nhất.
 - Node 8 là root của tree chứa các số 1, 3, 4, 6, 7, 8, 10, 13, 14.
 - Node 3 là root của tree chứa các số 1, 3, 4, 6, 7.



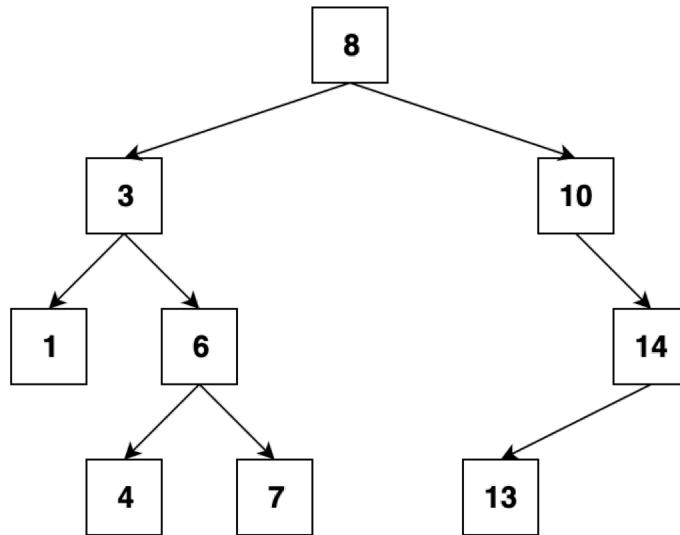
Child

- **Child:** các node là con của 1 node đang xét.
 - Các node số 1, 6 là child node của node 3.



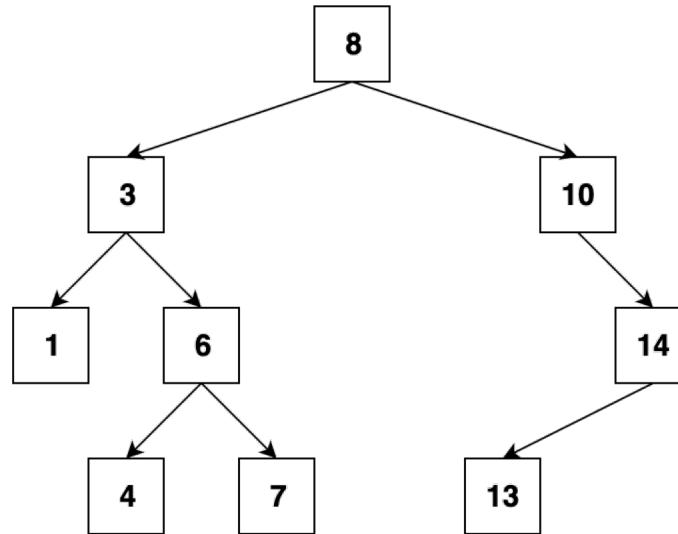
Parent

- **Parent:** mỗi node có 1 node cha (trừ root).
 - Node 3 là parent node của các node 1, 6.



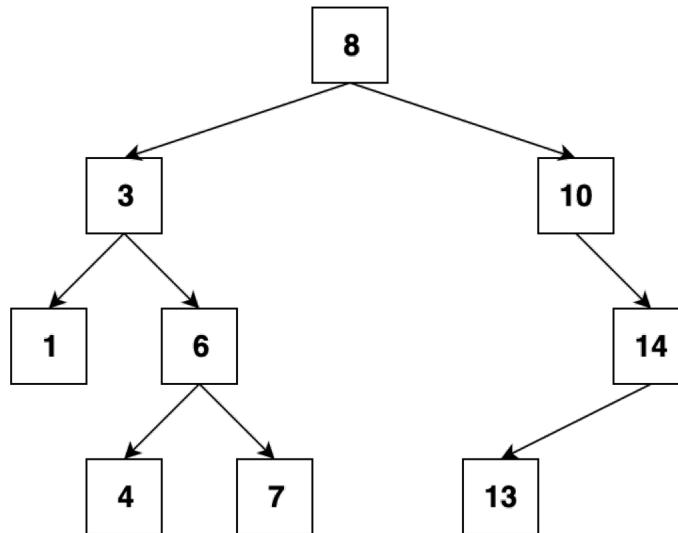
Sibling

- **Sibling**: các node anh em, có cùng node cha.
 - Các node 3, 10 là sibling node của nhau, vì có cùng parent node 8.



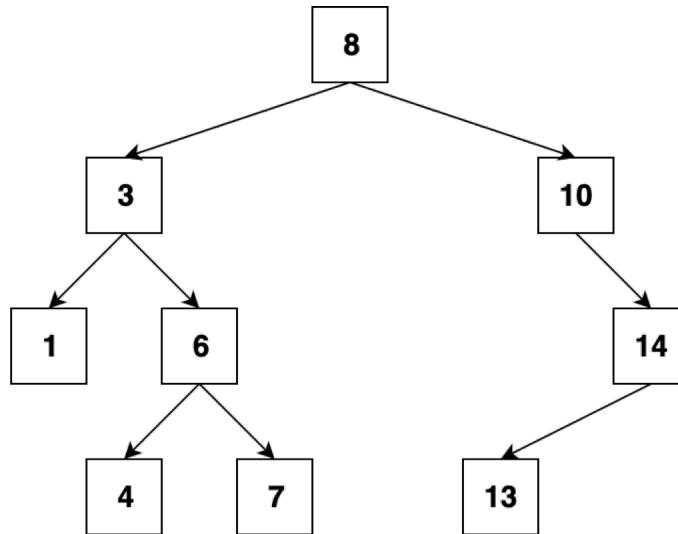
Ancestor

- **Ancestor:** node tổ tiên của 1 node.
 - Node 8 là ancestor node của node 6.



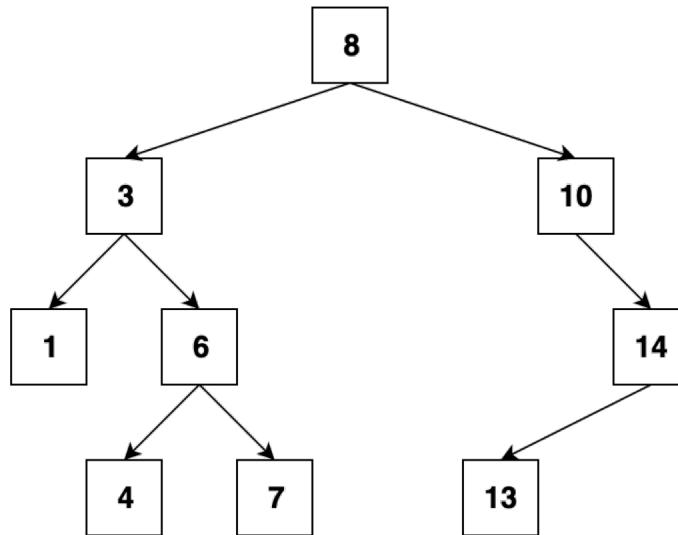
Leaf

- **Leaf** – External node: node không có nhánh con nào cả.
 - Các node 1, 4, 7, 13 được gọi là leaf.



Branch

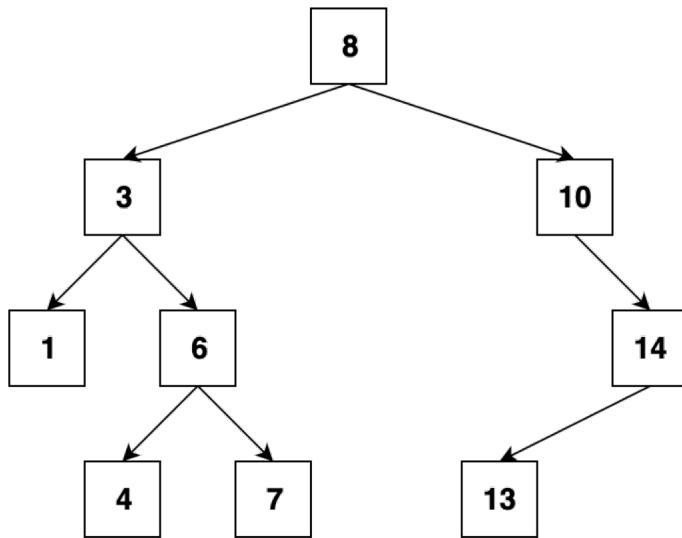
- **Branch** – Internal node: node có ít nhất 1 nhánh con.
 - Các node 3, 6, 8, 10, 14 gọi là các internal node.



Degree

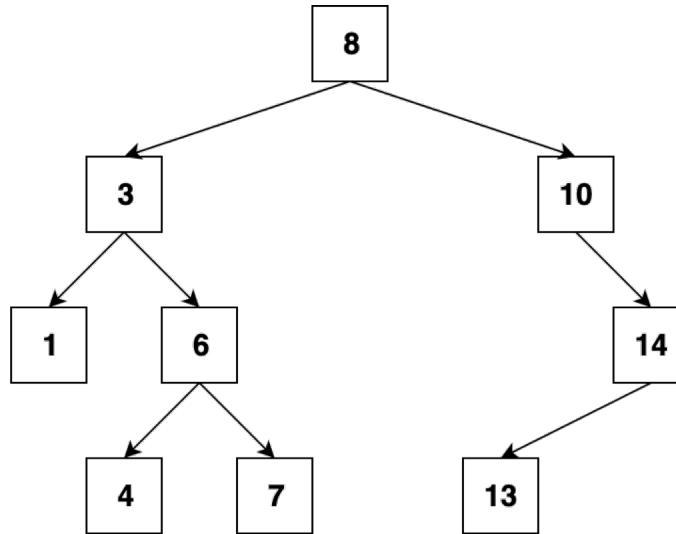
- **Degree**: số nhánh con của một node.

- $\text{Degree}(\text{node } 8) = 2$.
- $\text{Degree}(\text{node } 10) = 1$.



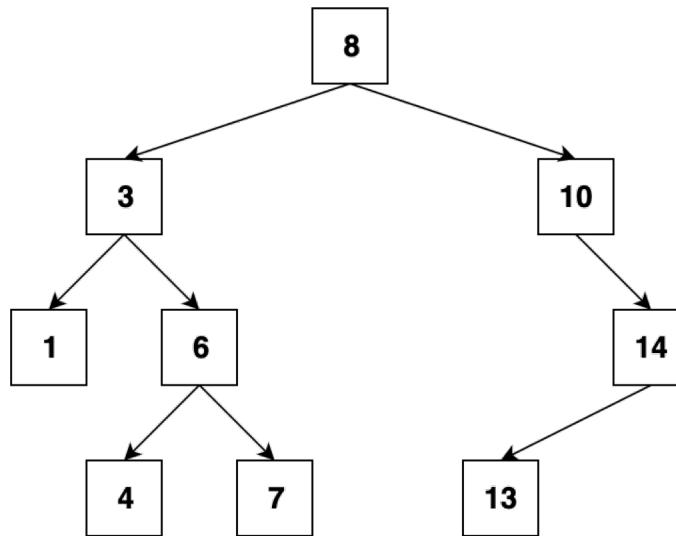
Edge

- **Edge**: đường nối trực tiếp giữa 2 node.
 - Node 8 và node 3 có edge 8-3.
 - Node 8 và node 4 không có edge (nhưng có path).



Path

- **Path:** đường nối từ 1 node tổ tiên đến 1 node con.
 - Node 8 và node 4 không có edge, nhưng có path 8-3-6-4.
 - Node 10 và node 7 không có edge, cũng không có path.



Level

- **Level:**

- $\text{Level}(\text{root}) = 0$

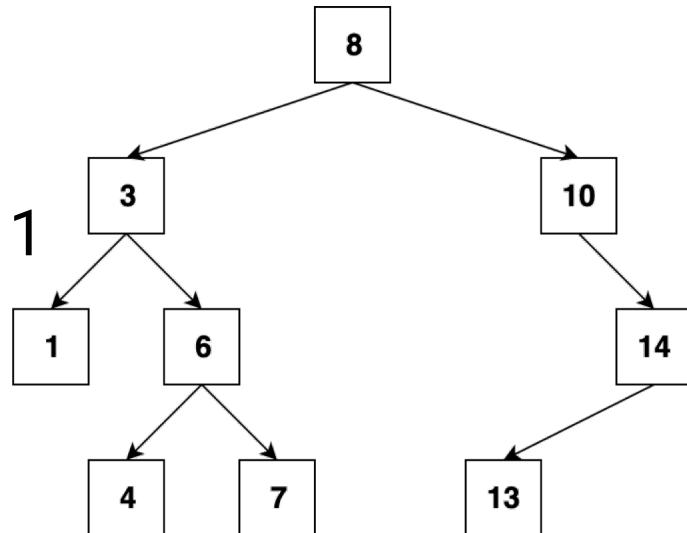
- $\text{Level}(\text{node } 8) = 0.$

- $\text{Level}(\text{node}) = \text{level}(\text{its parent}) + 1$

- $\text{Level}(\text{node } 3) = 1.$

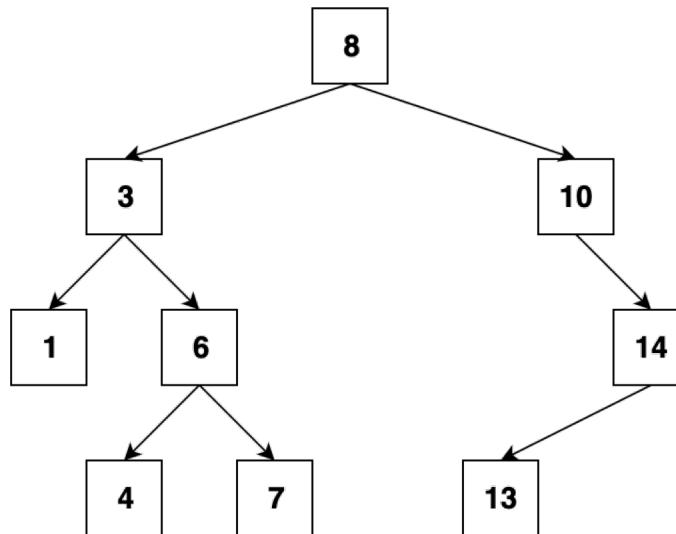
- Leaf có level cao nhất.

- $\text{Level}(\text{node } 1) = 2; \text{level}(\text{node } 4) = 3.$



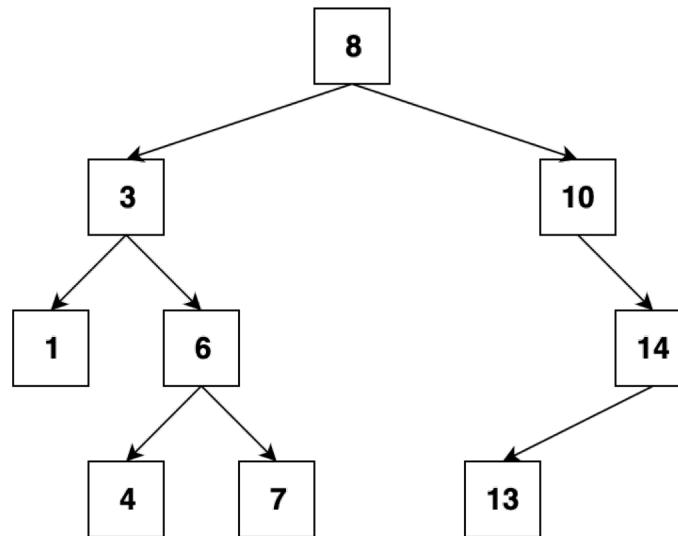
Height

- **Height**: số node trên đường dài nhất, từ root đến leaf.
 - Height(node 3) = 3.
 - Height(node 8) = 4.



Height of tree

- **Height of tree:** chiều cao của root.
 - Chiều cao của tree có root là node 3 = 3.
 - Chiều cao của tree có root là node 8 = 4.

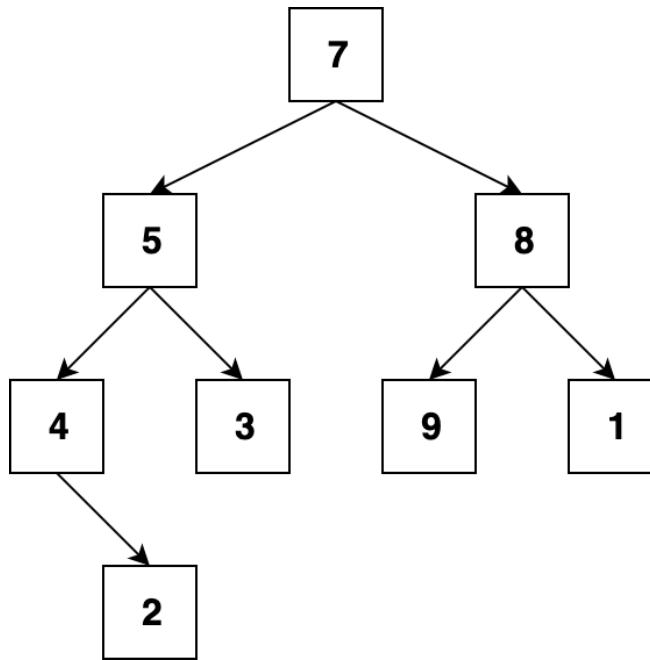


Forest

- **Forest:** bao gồm nhiều tree phân biệt.

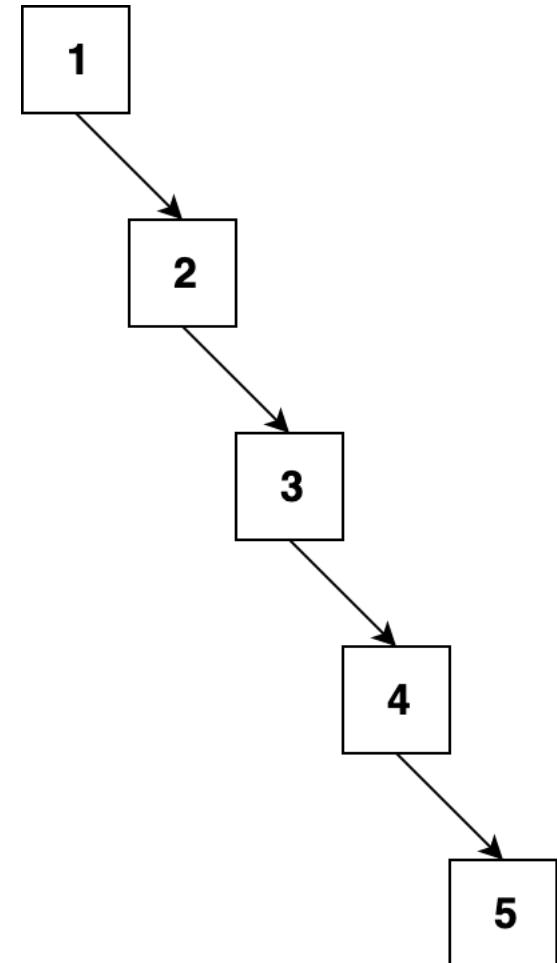
Binary tree

- Cây nhị phân là cây mà mọi node trên cây có tối đa 2 nhánh con: left branch, right branch.
 - Tức là mỗi node có thể có 0 nhánh (leaf), 1 nhánh hoặc 2 nhánh. Không thể có node nào có ≥ 3 nhánh.



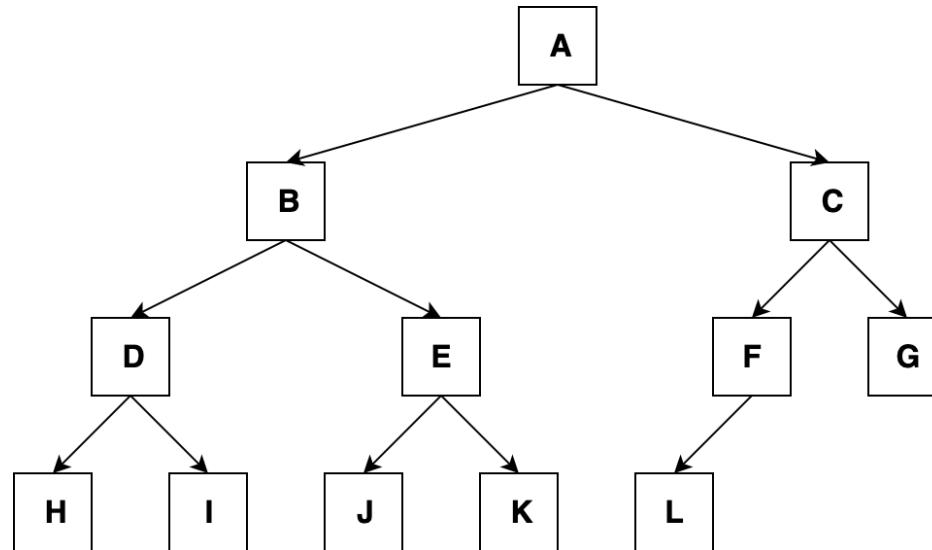
Degenerate tree

- Cây nhị phân suy biến là cây mỗi node (không tính leaf) chỉ có 1 nhánh duy nhất.
 - Cùng một số lượng node N , degenerate tree là cây cao nhất.



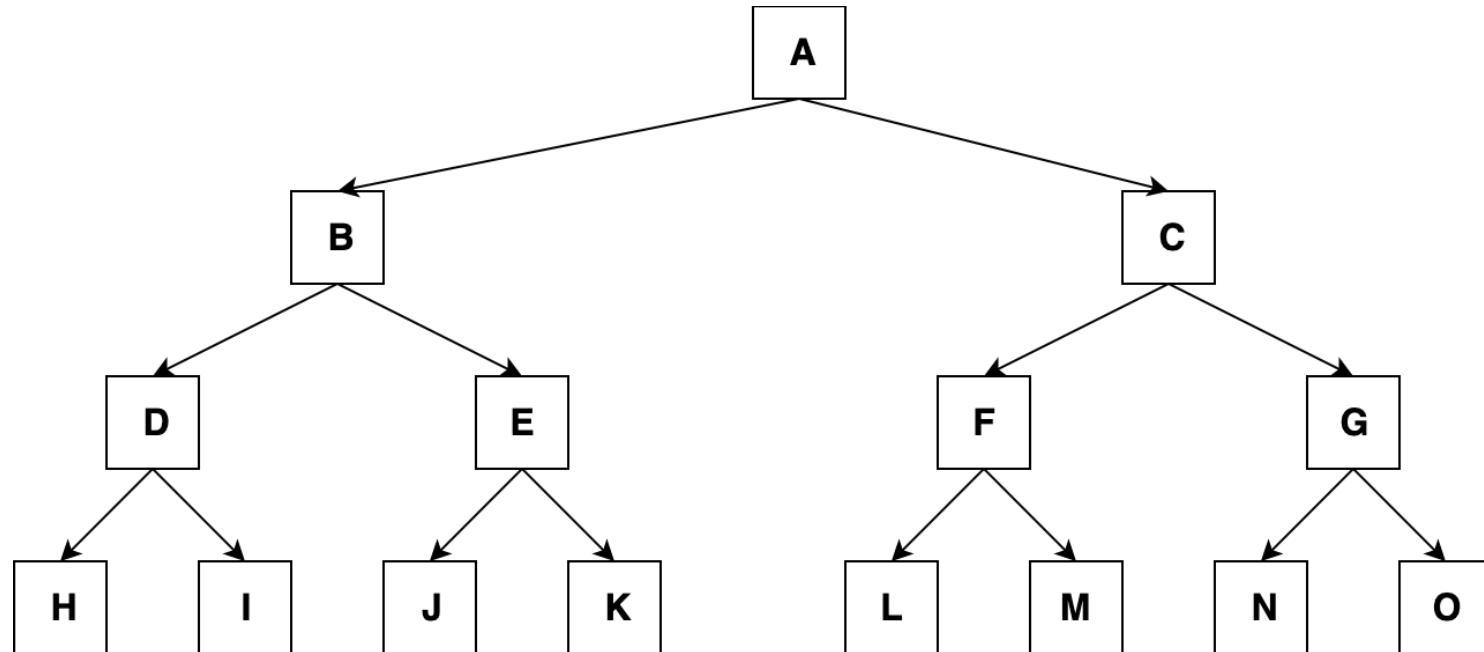
Complete binary tree

- Complete tree là cây nhị phân mà mỗi node, ngoại trừ node cuối, đều có đầy đủ con, được hoàn chỉnh từ trái sang phải.
 - Cùng số lượng node N, complete tree là cây có chiều cao thấp nhất.



Full binary tree

- Full binary tree là cây mà mỗi node, ngoại trừ node lá đều có đầy đủ 2 nhánh con left, right.

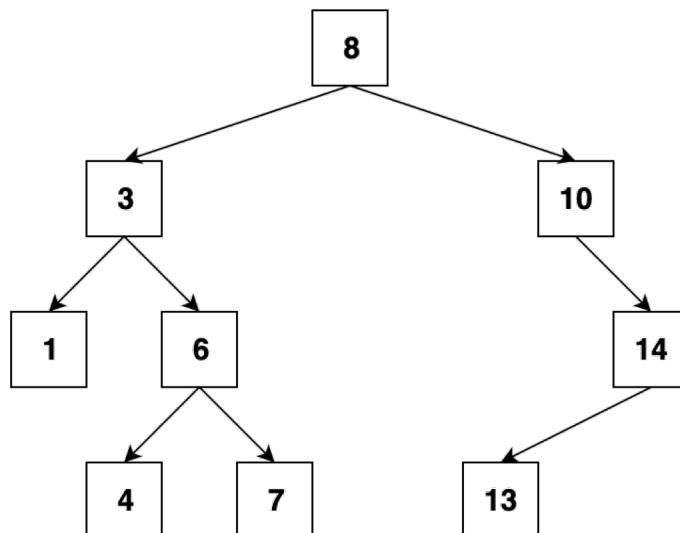


Nhận xét

- Với số lượng node như nhau:
 - Height(degenerate binary tree) lớn nhất.
 - Height(complete binary tree) nhỏ nhất.
- Cây nhị phân có chiều cao h có tối đa $2^h - 1$ node.
- Chiều cao của complete binary tree có N node $h = \text{ceil}(\log N) + 1$.

Binary search tree

- Binary search tree (BST) là một binary tree có thêm các tính chất sau:
 - Các node không chứa các giá trị trùng nhau. VD: không thể có 2 node cùng có data = 77.
 - Với mỗi node x bất kì:các node bên nhánh trái của x < node x < các node bên nhánh phải của x



Biểu diễn BST – struct/class Node

- struct/class Node đại diện cho 1 node trên cây BST, bao gồm 3 thuộc tính:
 - data: dữ liệu cần lưu trữ.
 - left: nhánh trái của cây.
 - right: nhánh phải của cây.

Node	
-	data: int
-	left: Node
-	right: Node
+	createNode(x: int): Node
+	addToNode(x: int): void

struct Node (C++)

```
struct Node
{
    int data;
    Node* left;
    Node* right;
};

Node* createNode(int x)
{
    Node* p = new Node;
    p->data = x;
    p->left = NULL;
    p->right = NULL;
    return p;
}
```

Node
- data: int - left: Node - right: Node
+ createNode(x: int): Node + addToNode(x: int): void

class Node (Java)

```
class Node{
    public int data;
    public Node left;
    public Node right;

    public Node(){
        data = 0;
        left = null;
        right = null;
    }

    public Node(int x){
        data = x;
        left = null;
        right = null;
    }
}
```

Node
- data: int - left: Node - right: Node
+ createNode(x: int): Node + addTreeNode(x: int): void

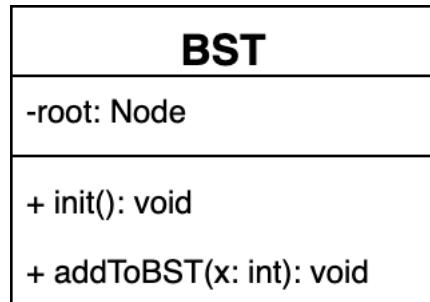
class Node (Python)

```
class Node:  
    def __init__(self, x = 0, left = None, right = None):  
        self.data = x  
        self.left = left  
        self.right = right
```

Node
- data: int
- left: Node
- right: Node
+ createNode(x: int): Node
+ addTreeNode(x: int): void

Biểu diễn BST – struct/class BST

- struct/class BST đại diện cho 1 cây nhị phân tìm kiếm, bao gồm 1 thuộc tính duy nhất:
 - root: node gốc của cây.



struct BST (C++)

```
struct BST
{
    Node* root;
};

void init(BST &t)
{
    t.root = NULL;
}
```

BST

-root: Node

+ init(): void

+ addToBST(x: int): void

class BST (Java)

```
class BST{  
    private Node root;  
  
    public BST(){  
        root = null;  
    }  
}
```

BST
-root: Node
+ init(): void
+ addToBST(x: int): void

class BST (Python)

```
class BST:  
    def __init__(self):  
        self.root = None
```

BST
-root: Node
+ init(): void
+ addToBST(x: int): void

Xử lí trên cây BST

- Gọi hàm xử lí của cây BST.
 - Gọi hàm xử lí của node root.
 - Gọi đệ qui hàm xử lí của node left.
 - Gọi đệ qui hàm xử lí của node right.
 - Tổng hợp kết quả và return.

BT1 – TÌM PHẦN TỬ NHỎ NHẤT

- Cho N số, bạn hãy xây dựng cây nhị phân tìm kiếm từ N số đó. Sau đó hãy tìm phần tử nhỏ nhất trong cây nhị phân này.



BT1 – Gợi ý – Xử lí chính

1. Đọc vào N.
2. Đọc vào mảng số nguyên.
3. Khởi tạo cây BST.
4. Duyệt qua mảng, lần lượt gọi hàm **addToBST()** để thêm từng số vào cây BST.
5. Gọi hàm **minBST()** để tìm số nhỏ nhất trong cây BST.



BT1 – Gợi ý – struct/class

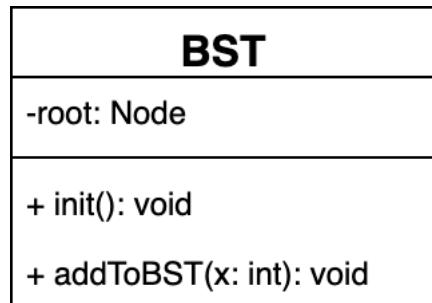
1. Khai báo struct/class Node.
2. Khai báo struct/class BST.
3. Khai báo và cài đặt hàm addToBST() trong BST để thêm một giá trị vào cây BST.
4. Khai báo và cài đặt hàm minBST() trong cây BST.

Node
- data: int
- left: Node
- right: Node
+ createNode(x: int): Node
+ addToNode(x: int): void
+ minNode(): Node

BST
-root: Node
+ init(): void
+ addToBST(x: int): void
+ minBST(): Node

BT1 – Gợi ý – Hàm addToBST()

1. Nếu cây BST là rỗng,
 - a. Tạo node p cho x.
 - b. Gán root là node p.
2. Ngược lại, nếu root không rỗng,
 - a. Gọi hàm root.addNode(x)



BT1 – Gợi ý – Hàm addToBST() (C++)

```
void addToBST(BST &t, int x){  
    if(t.root == NULL){  
        Node *p = createNode(x);  
        t.root = p;  
    }  
    else{  
        addToNode(t.root, x);  
    }  
}
```

BT1 – Gợi ý – Hàm addToBST() (Java)

```
class BST{  
    public void addToBST(int x){  
        if(root == null){  
            Node p = new Node(x);  
            root = p;  
        }  
        else{  
            root.addNode(x);  
        }  
    }  
}
```

BT1 – Gợi ý – Hàm addToBST() (Python)

```
class BST:
    def __init__(self):
        self.root = None

    def addToBST(self, x):
        if(self.root == None):
            p = Node(x)
            self.root = p
        else:
            self.root.addNode(x)
```

BT1 – Gợi ý – Hàm addToNode()

1. Nếu $x < \text{data}$,
 - a. Nếu left là `NULL / null / None`,
 - i. Tạo node p
 - ii. $\text{left} = p$
 - b. Ngược lại
 - i. Gọi đệ qui: $\text{left.addNode}(x)$
2. Nếu $x > \text{data}$,
 - a. Nếu right là `NULL / null / None`,
 - i. Tạo node p
 - ii. $\text{right} = p$
 - b. Ngược lại
 - i. Gọi đệ qui: $\text{right.addNode}(x)$

Node
- <code>data: int</code>
- <code>left: Node</code>
- <code>right: Node</code>
+ <code>createNode(x: int): Node</code>
+ <code>addNode(x: int): void</code>

BT1 – Gợi ý – Hàm addToNode() (C++)

```
void addToNode(Node *&root, int x){  
    if(x < root->data){  
        if(root->left == NULL){  
            Node *p = createNode(x);  
            root->left = p;  
        }  
        else{  
            addToNode(root->left, x);  
        }  
    }  
    else if (x > root->data){  
        if(root->right == NULL){  
            Node *p = createNode(x);  
            root->right = p;  
        }  
        else{  
            addToNode(root->right, x);  
        }  
    }  
    else{  
        // Do nothing  
    }  
}
```

BT1 – Gợi ý – Hàm addTreeNode() (Java)

```
class Node{  
    public void addTreeNode(int x){  
        if(x < data){  
            if(left == null){  
                Node p = new Node(x);  
                left = p;  
            }  
            else{  
                left.addNode(x);  
            }  
        }  
        else if(x > data){  
            if(right == null){  
                Node p = new Node(x);  
                right = p;  
            }  
            else{  
                right.addNode(x);  
            }  
        }  
        else{  
            // Do nothing  
        }  
    }  
}
```

BT1 – Gợi ý – Hàm addTreeNode() (Python)

```
class Node:  
    def addTreeNode(self, x):  
        if(x < self.data):  
            if(self.left == None):  
                p = Node(x)  
                self.left = p  
            else:  
                self.left.addTreeNode(x)  
        elif(x > self.data):  
            if(self.right == None):  
                p = Node(x)  
                self.right = p  
            else:  
                self.right.addTreeNode(x)  
        else:  
            return  
        # Do nothing
```

Minh họa addToBST(x)

- Add lần lượt 6, 3, 2, 1, 4 vào cây nhị phân tìm kiếm.

Minh họa addToBST(6)

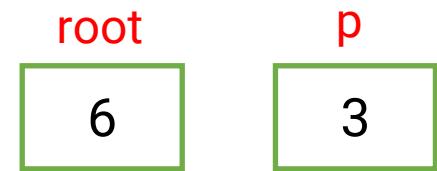


Minh họa addToBST(6)

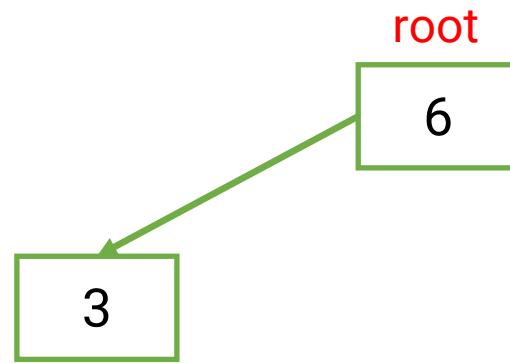
root

6

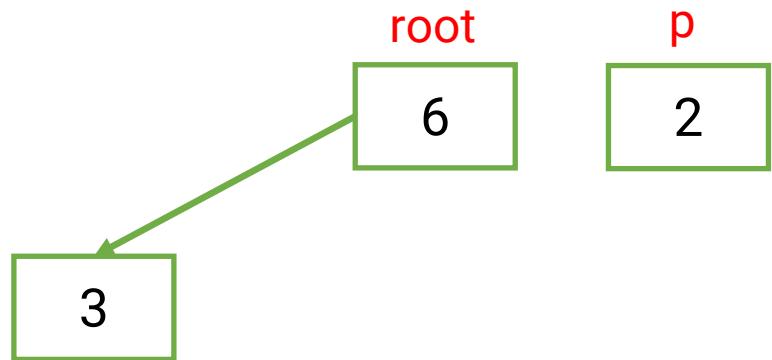
Minh họa addToBST(3)



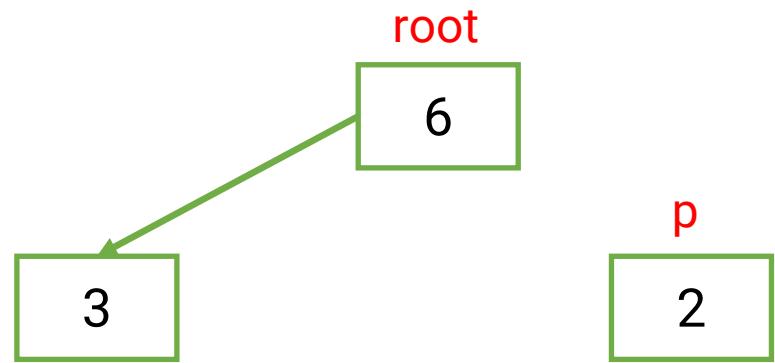
Minh họa addToBST(3)



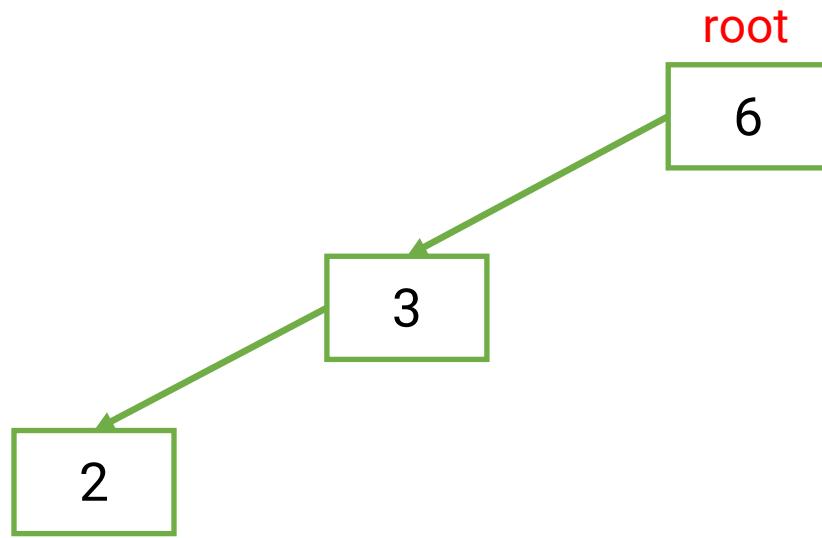
Minh họa addToBST(2)



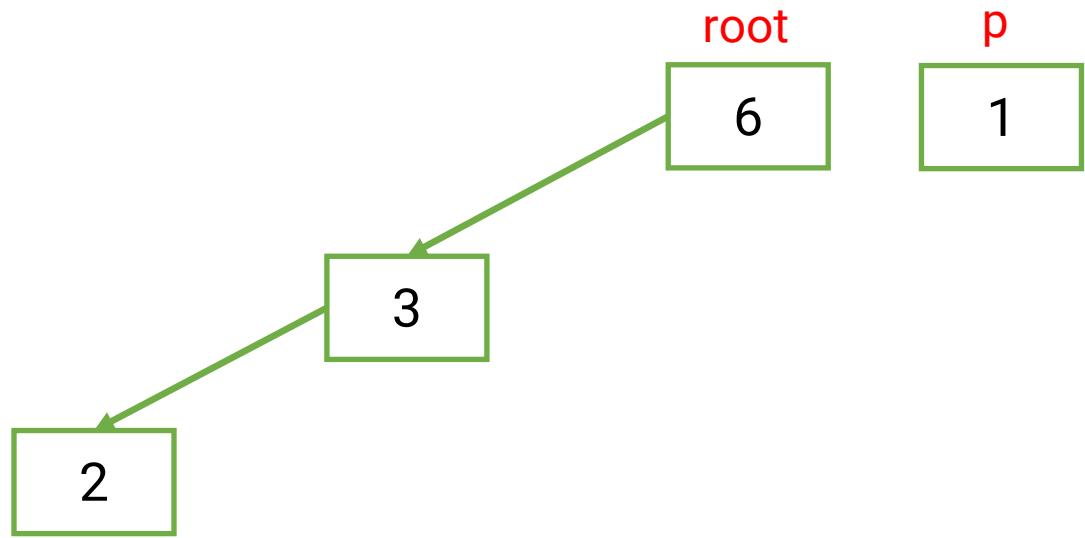
Minh họa addToBST(2)



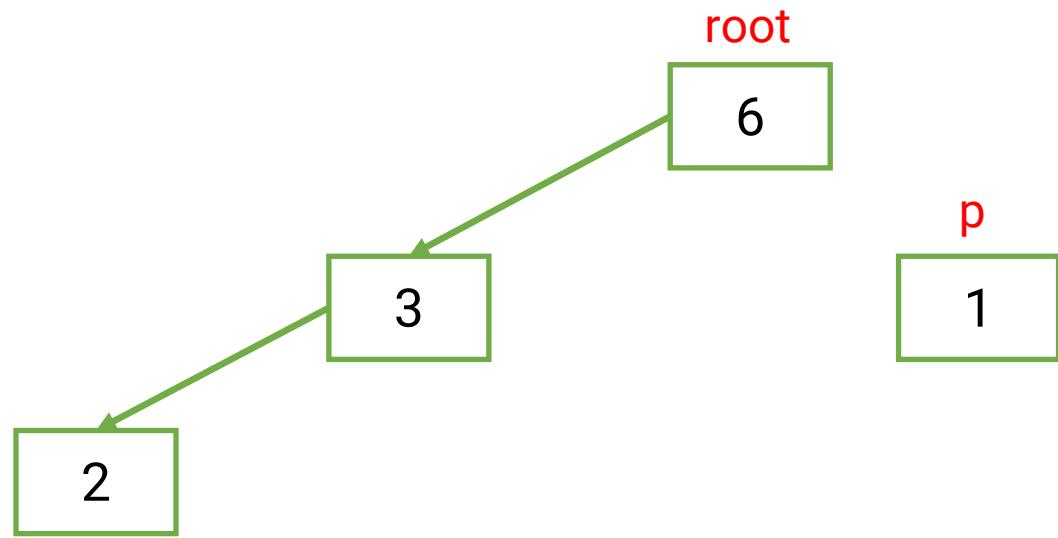
Minh họa addToBST(2)



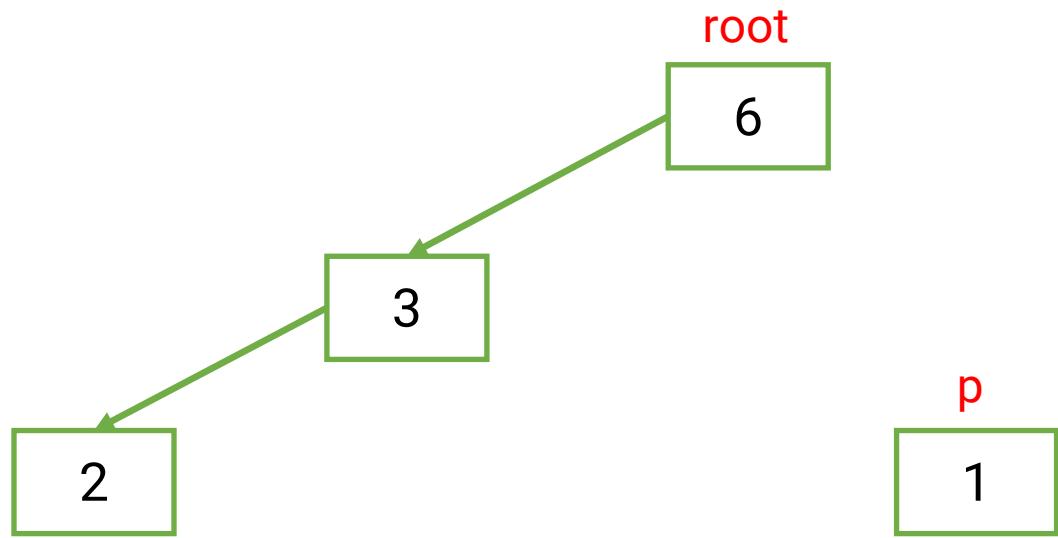
Minh họa addToBST(1)



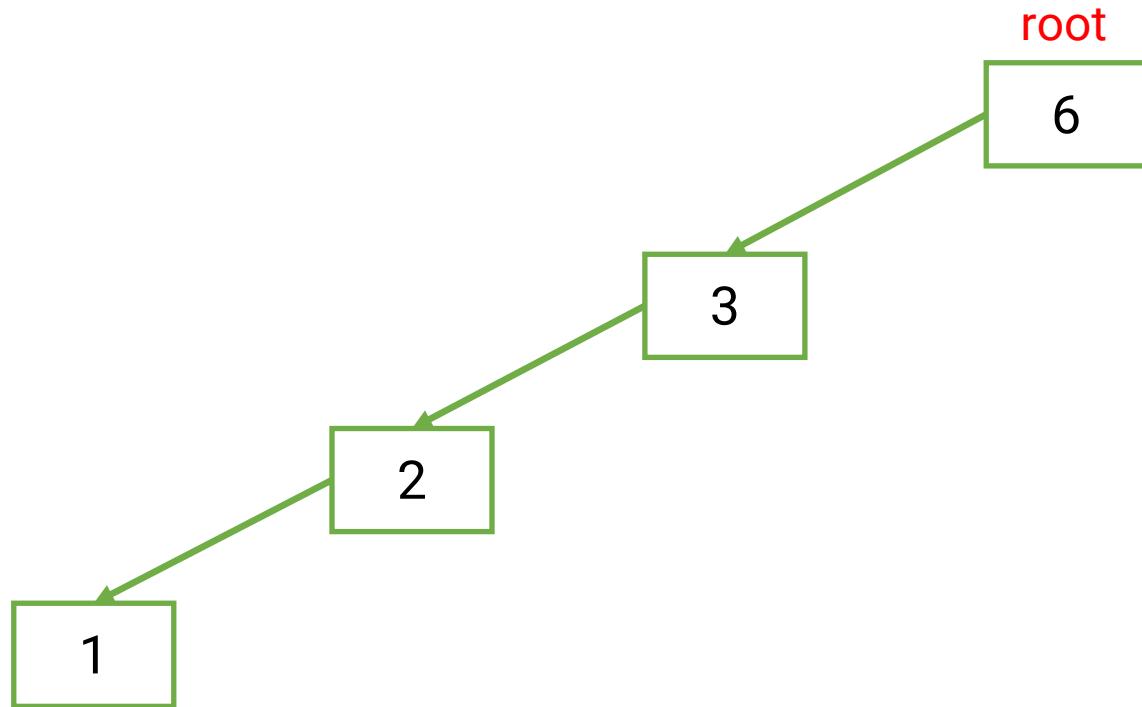
Minh họa addToBST(1)



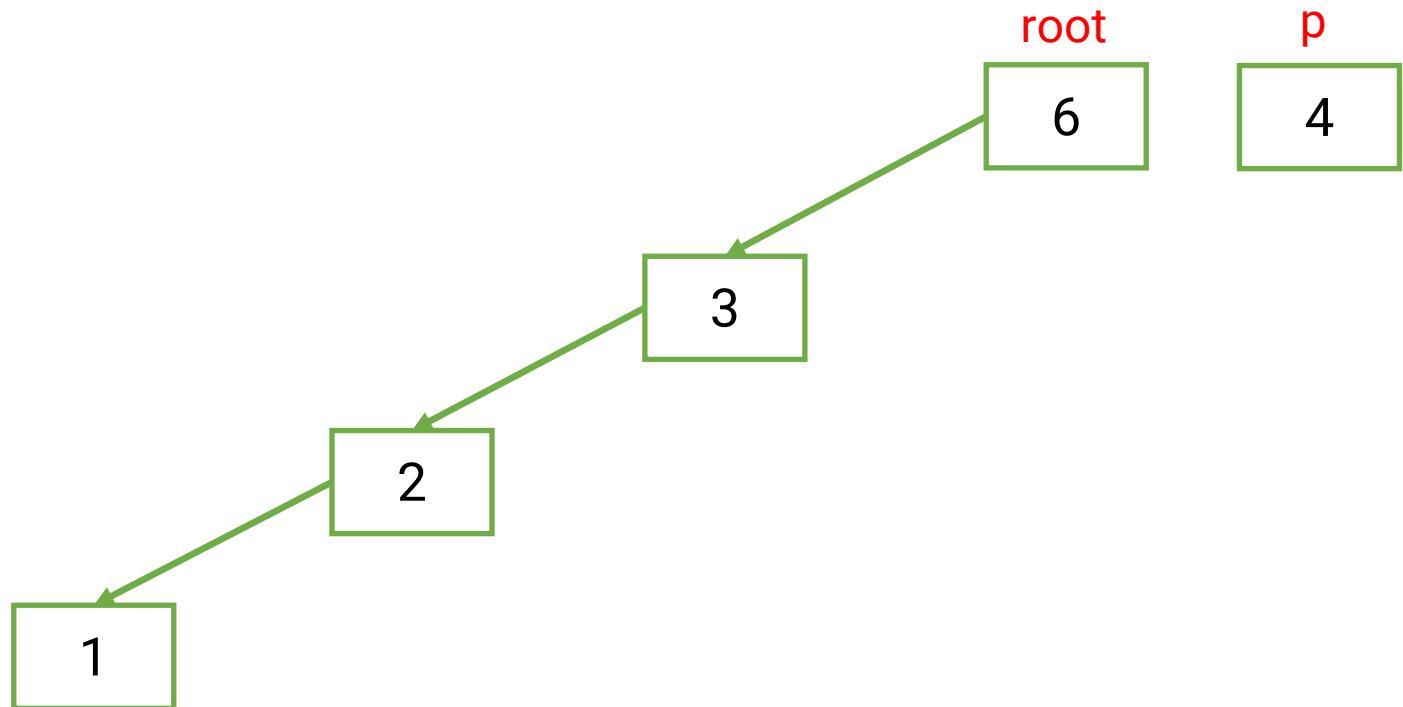
Minh họa addToBST(1)



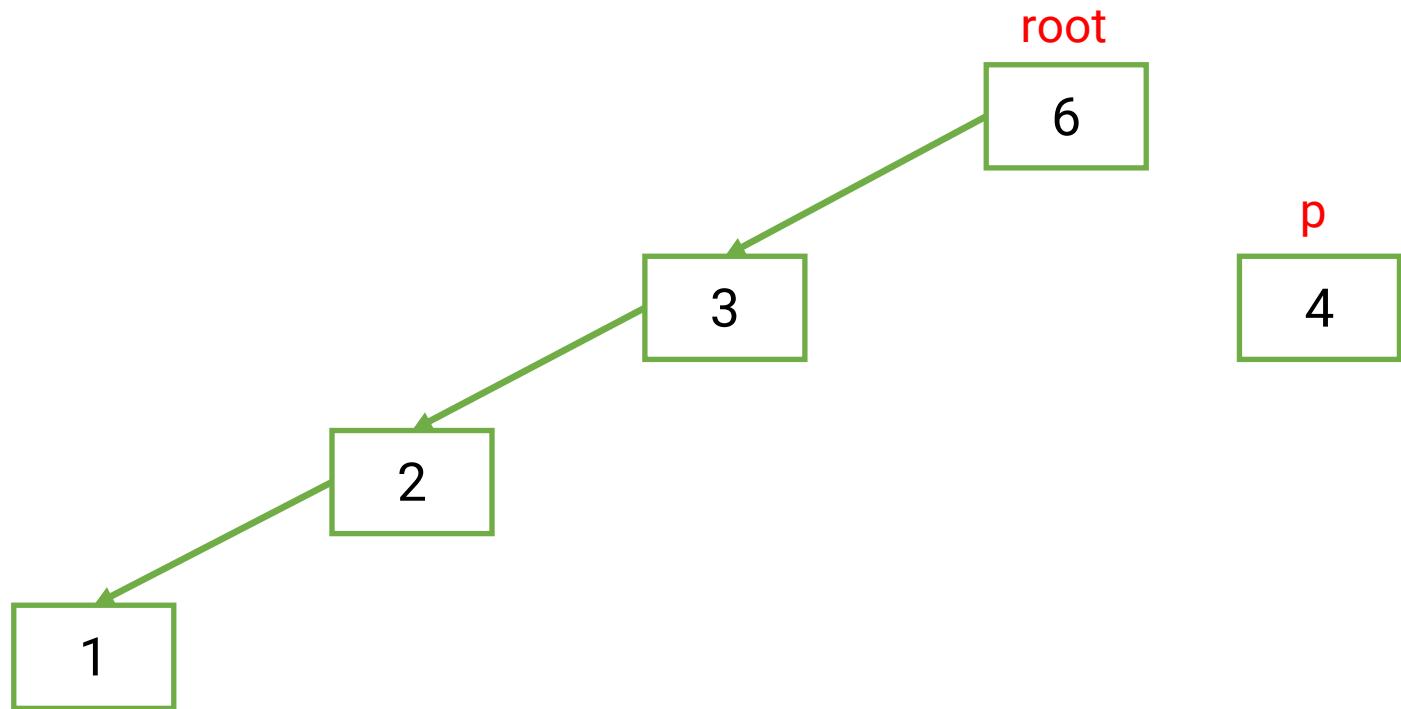
Minh họa addToBST(1)



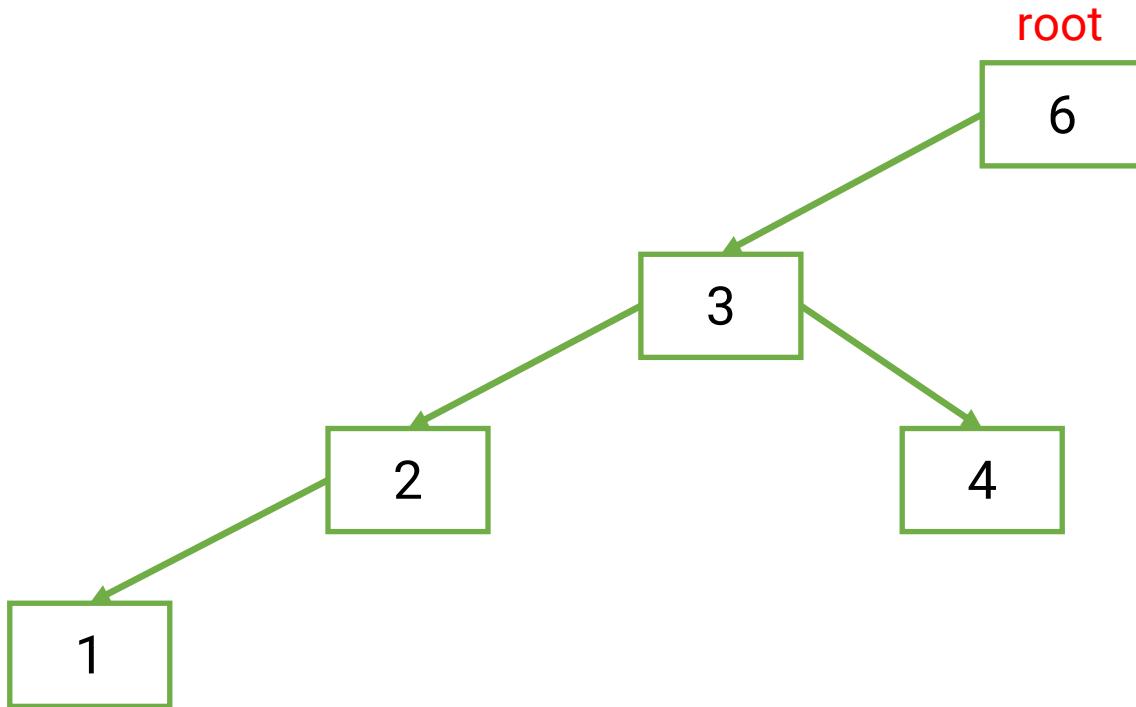
Minh họa addToBST(4)



Minh họa addToBST(4)



Minh họa addToBST(4)



BT1 – Gợi ý – Hàm minBST()

- Gọi hàm minNode() của Node.

BT1 – Gợi ý – Hàm minBST() (C++)

```
int minBST(BST t){  
    return minNode(t.root);  
}
```

BT1 – Gợi ý – Hàm minBST() (Java)

```
class BST{  
    public int minBST(){  
        return root.minNode();  
    }  
}
```

BT1 – Gợi ý – Hàm minBST() (Python)

```
class BST:  
    def minBST(self):  
        return self.root.minNode()
```

BT1 – Gợi ý – Hàm minNode() của Node

- Nếu left là NULL / null / None,
 - min chính là data → return data.
- Ngược lại,
 - Gọi đệ qui hàm minNode() cho left.

BT1 – Gợi ý – Hàm minNode() (C++)

```
int minNode(Node *root){  
    if(root->left == NULL){  
        return root->data;  
    }  
    else{  
        return minNode(root->left);  
    }  
}
```

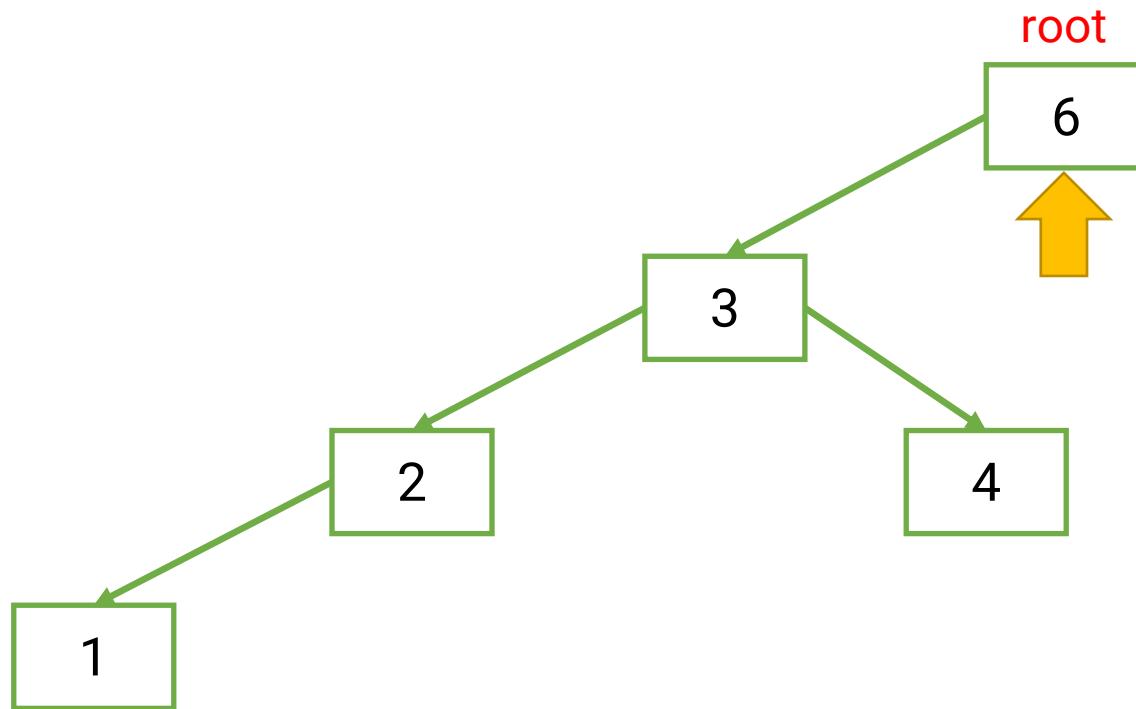
BT1 – Gợi ý – Hàm minNode() (Java)

```
class Node{
    public int minNode(){
        if(left == null){
            return data;
        }
        else{
            return left.minNode();
        }
    }
}
```

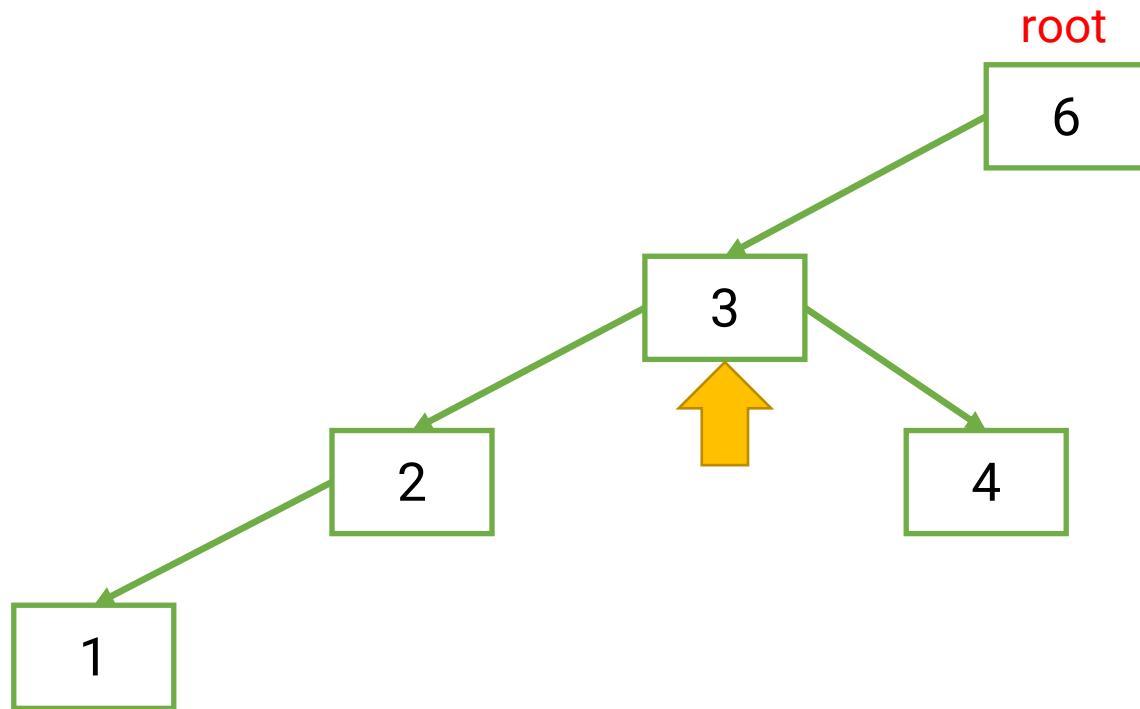
BT1 – Gợi ý – Hàm minNode() (Python)

```
class Node:  
    def minNode(self):  
        if(self.left == None):  
            return self.data  
        else:  
            return self.left.minNode()
```

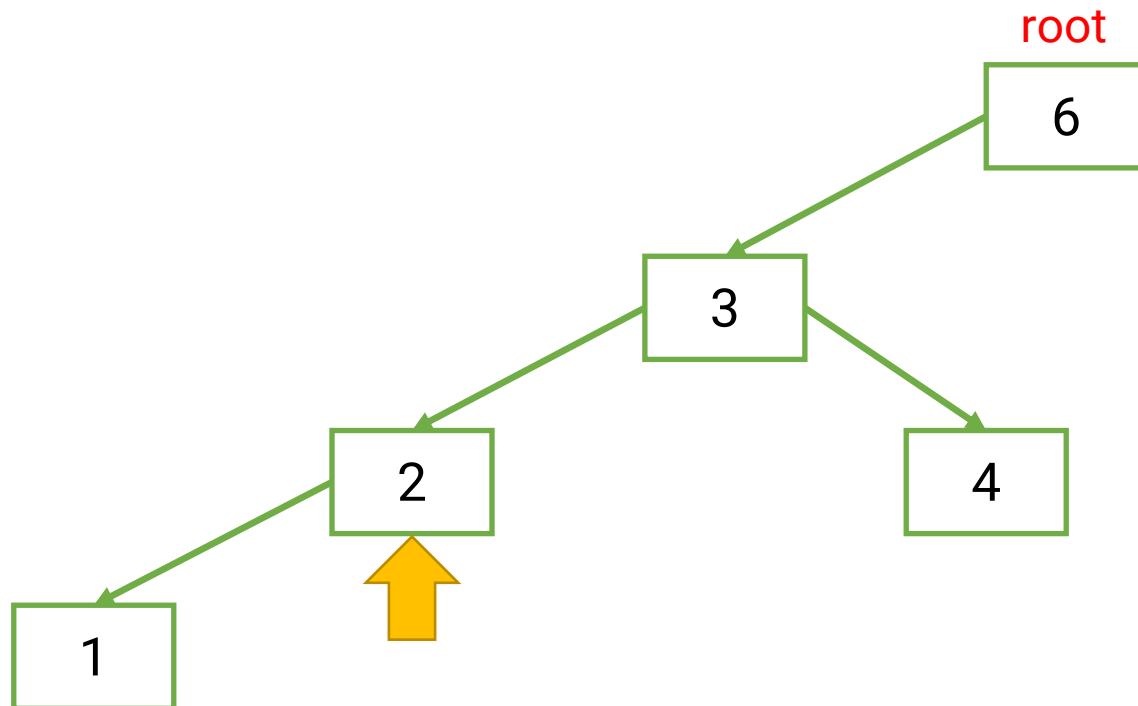
Minh họa hàm minBST()



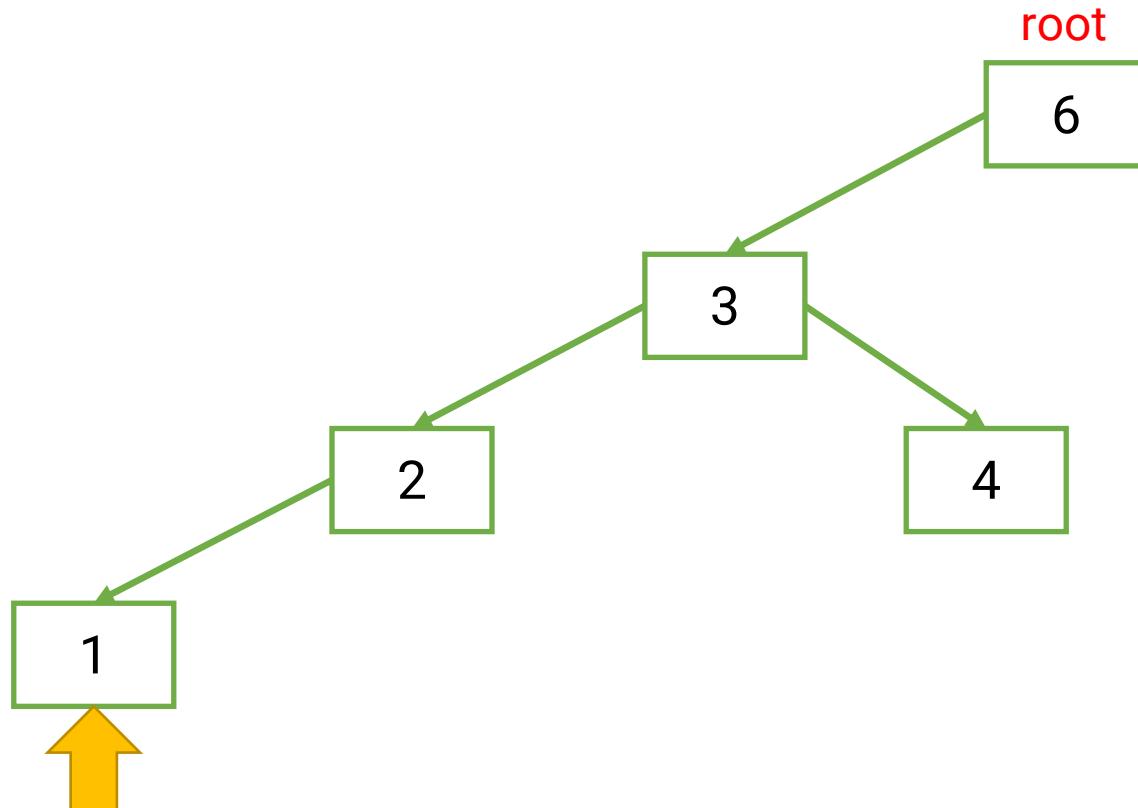
Minh họa hàm minBST()



Minh họa hàm minBST()



Minh họa hàm minBST()



BT2 – IN TĂNG DẦN

- Cho N phần tử, hãy xây dựng cây nhị phân tìm kiếm với N rồi in giá trị các nút trong cây đó ra theo chiều tăng dần.



BT2 – Gợi ý – Xử lí chính

1. Đọc vào N.
2. Đọc vào mảng số nguyên.
3. Khởi tạo cây BST.
4. Duyệt qua mảng, lần lượt gọi hàm **addToBST()** để thêm từng số vào cây BST.
5. Gọi hàm **printInorderBST()** để in các số theo thứ tự tăng dần.



3 cách in tree

- **Inorder traversal:** Left – **Node** – Right.
 - Dùng để duyệt câu theo thứ tự tăng dần.
- **Preorder traversal:** **Node** – Left – Right.
 - Dùng để sao chép từ cây t1 qua cây t2.
- **Postorder traversal:** Left – Right – **Node**.
 - Thứ tự khi xóa cây (xóa từ leaf lên root)

BT2 – Gợi ý – Hàm printInorderBST()

- Gọi hàm printInOrderNode() của root.

BT2 – Gợi ý – Hàm printInorderBST() (C++)

```
void printInorderBST(BST t)
{
    printInorderNode(t.root);
}
```

BT2 – Gợi ý – Hàm printInorderBST() (Java)

```
class BST{  
    public void printInorderBST(){  
        root.printInorderNode();  
    }  
}
```

BT2 – Gợi ý – Hàm printInorderBST() (Python)

```
class BST:  
    def printInorderBST(self):  
        self.root.printInorderNode()
```

BT2 – Gợi ý – Hàm printInorderNode()

1. Nếu left khác NULL / null / None,
 - a. Gọi đệ qui: printInOrderNode() của left.
2. In data.
3. Nếu right khác NULL / null / None,
 - a. Gọi đệ qui: printInOrderNode() của right.

BT2 – Gợi ý – Hàm printInorderNode() (C++)

```
void printInorderNode(Node* root)
{
    if(root->left != NULL)
        printInorderNode(root->left);

    cout << root->data << " ";

    if(root->right != NULL)
        printInorderNode(root->right);
}
```

BT2 – Gợi ý – Hàm printInorderNode() (Java)

```
class Node{
    public void printInorderNode(){
        if(left != null)
            left.printInorderNode();

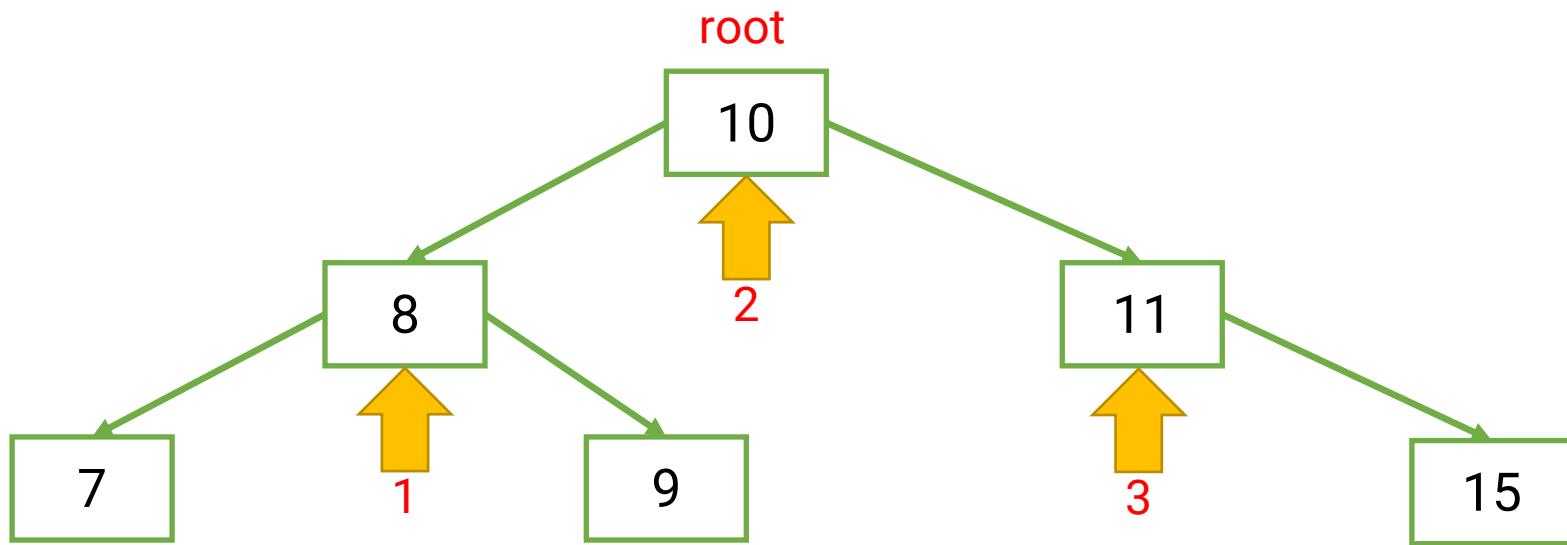
        System.out.printf("%d ", data);

        if(right != null)
            right.printInorderNode();
    }
}
```

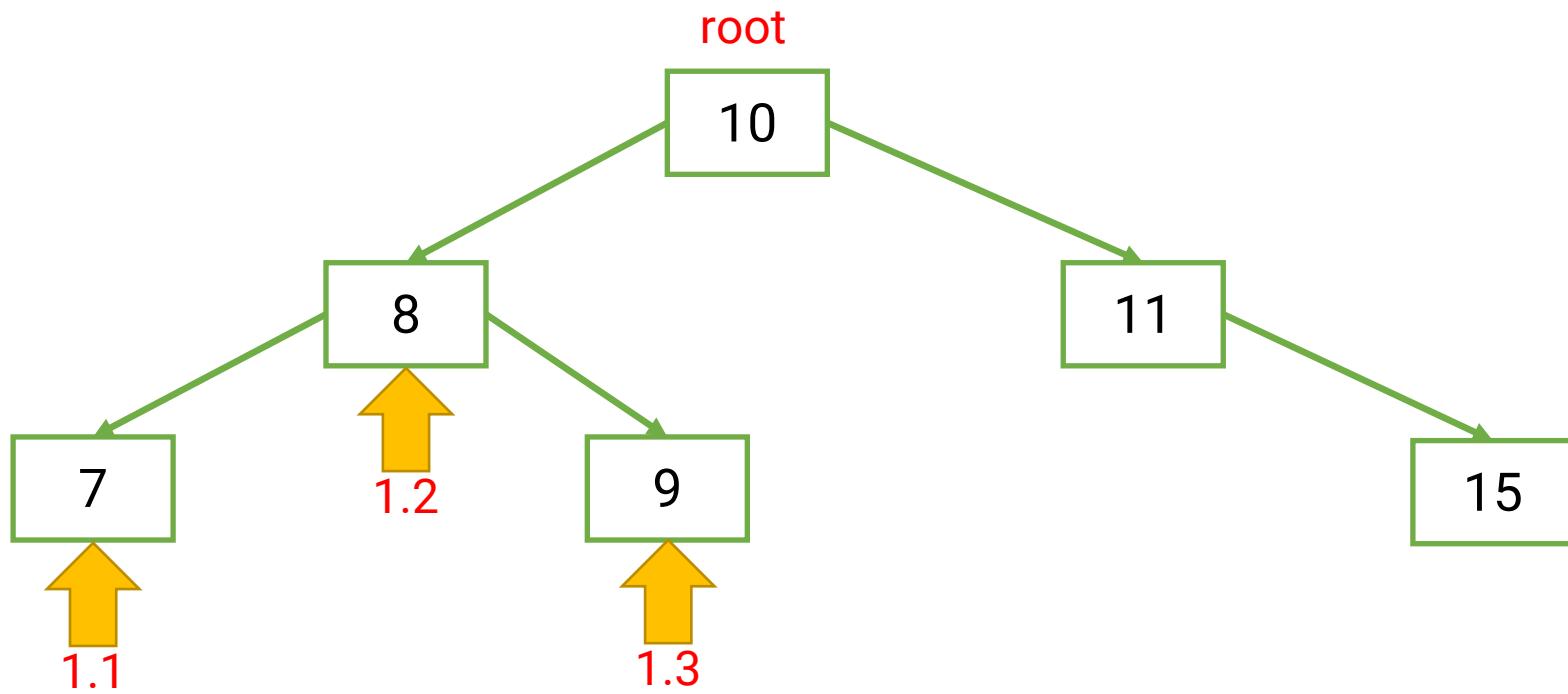
BT2 – Gợi ý – Hàm printInorderNode() (Python)

```
class Node:  
    def printInorderNode(self):  
        if(self.left != None):  
            self.left.printInorderNode()  
  
        print(self.data, end=" ")  
  
        if(self.right != None):  
            self.right.printInorderNode()
```

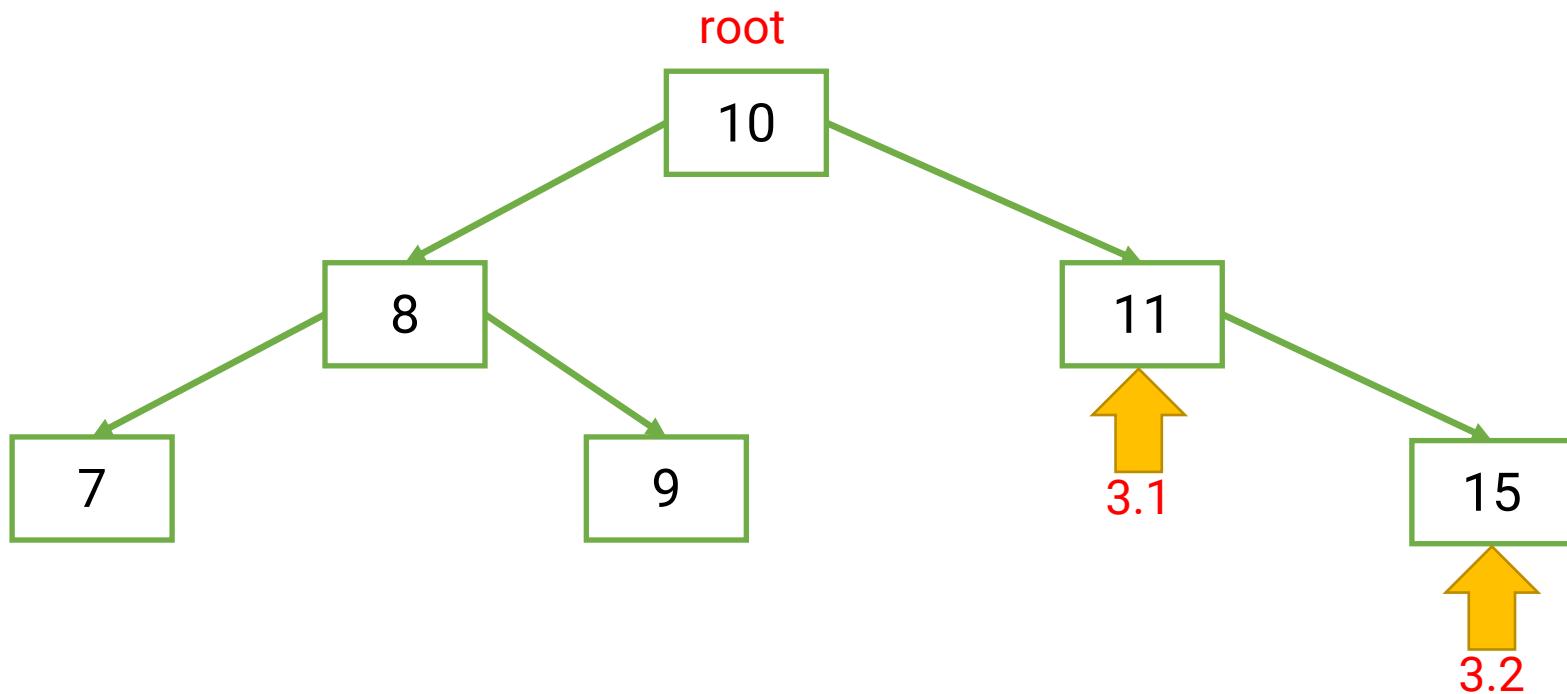
Inorder traversal – LNR



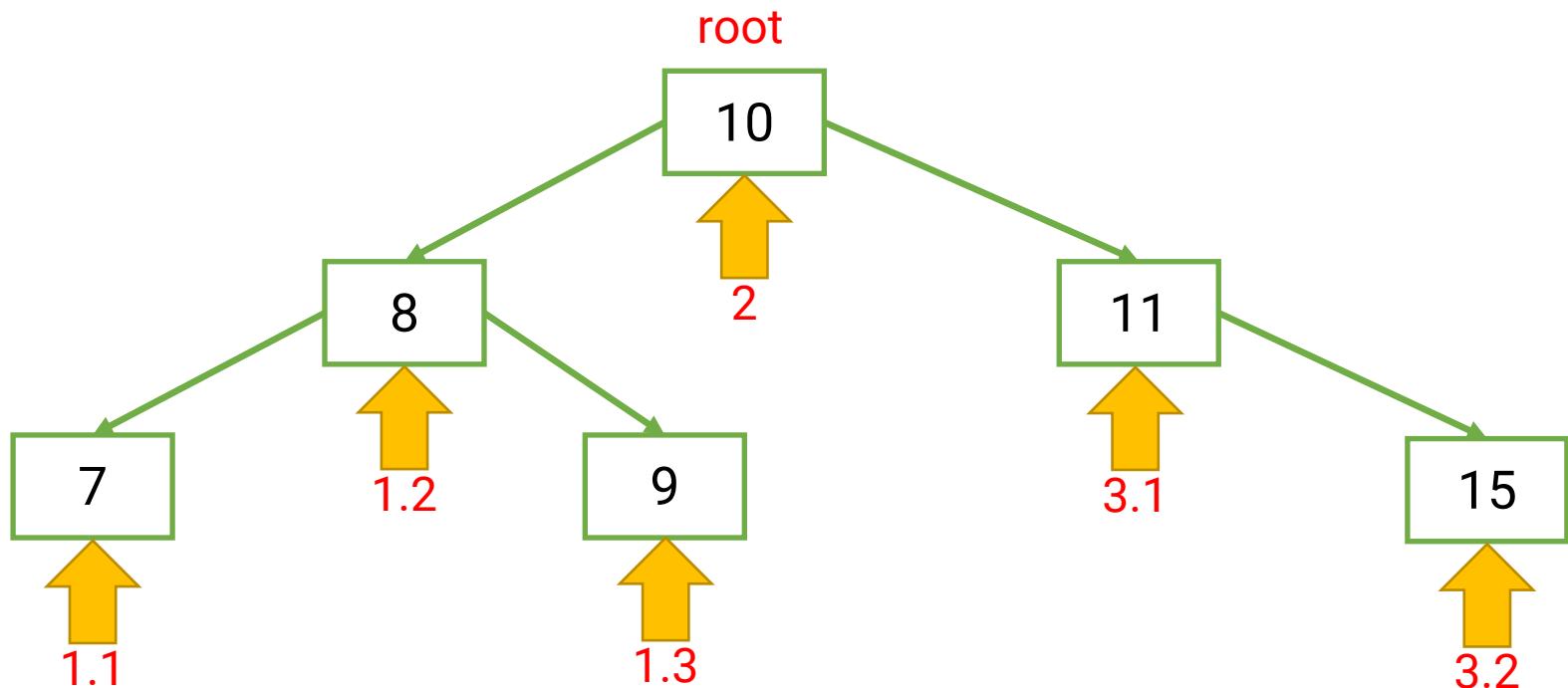
Inorder traversal – LNR



Inorder traversal – LNR

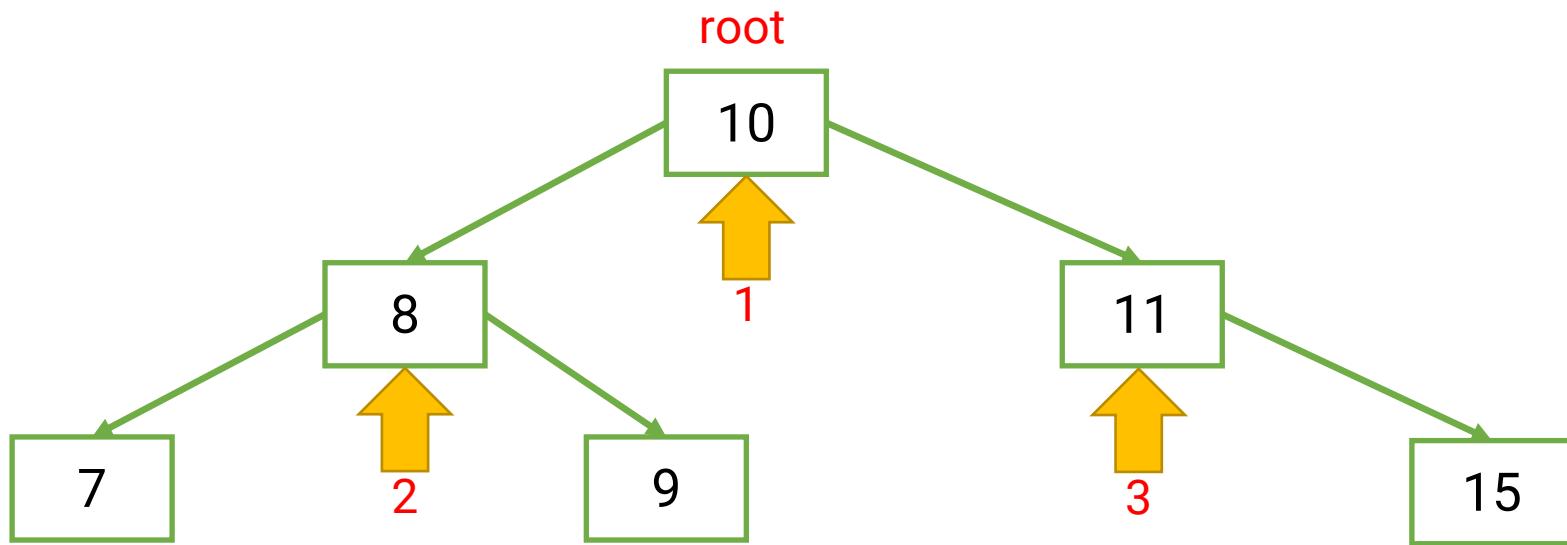


Inorder traversal – LNR

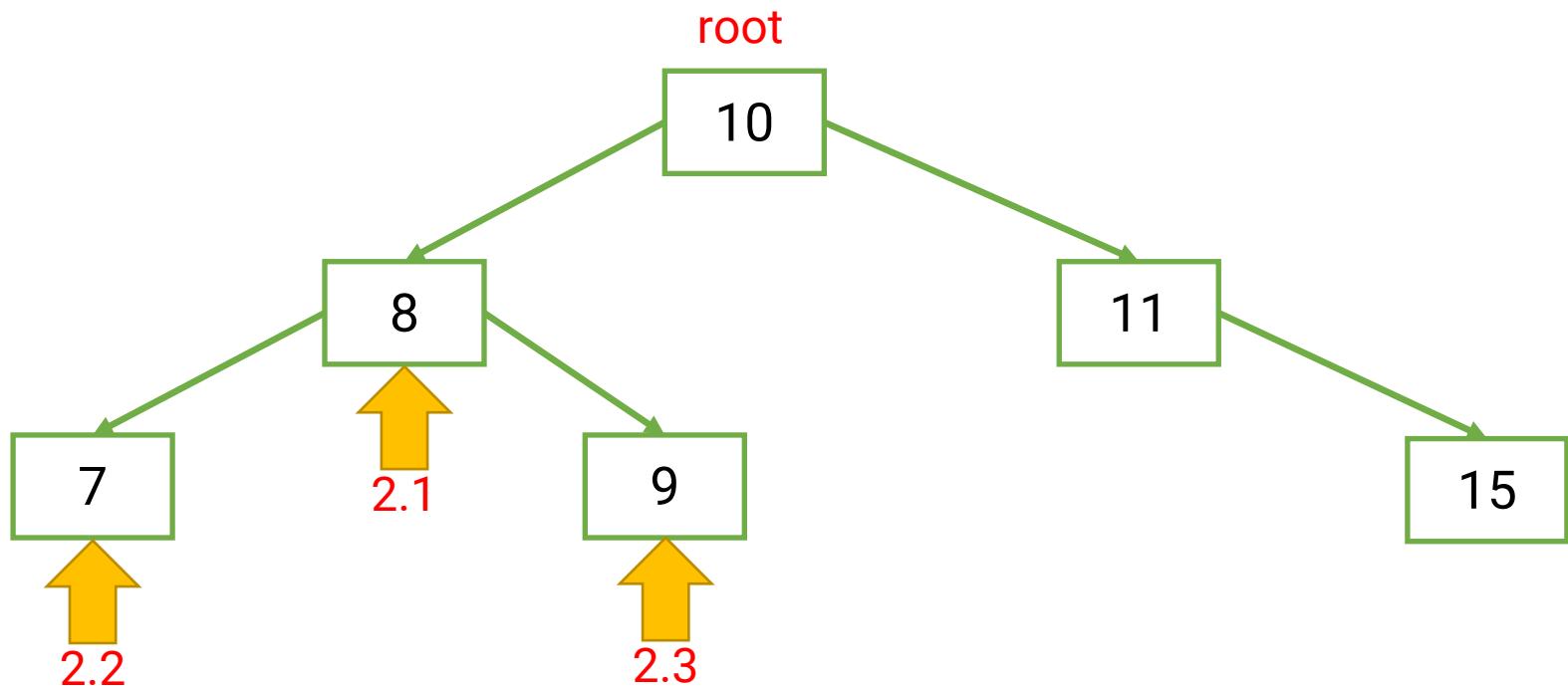


7, 8, 9, 10, 11, 15

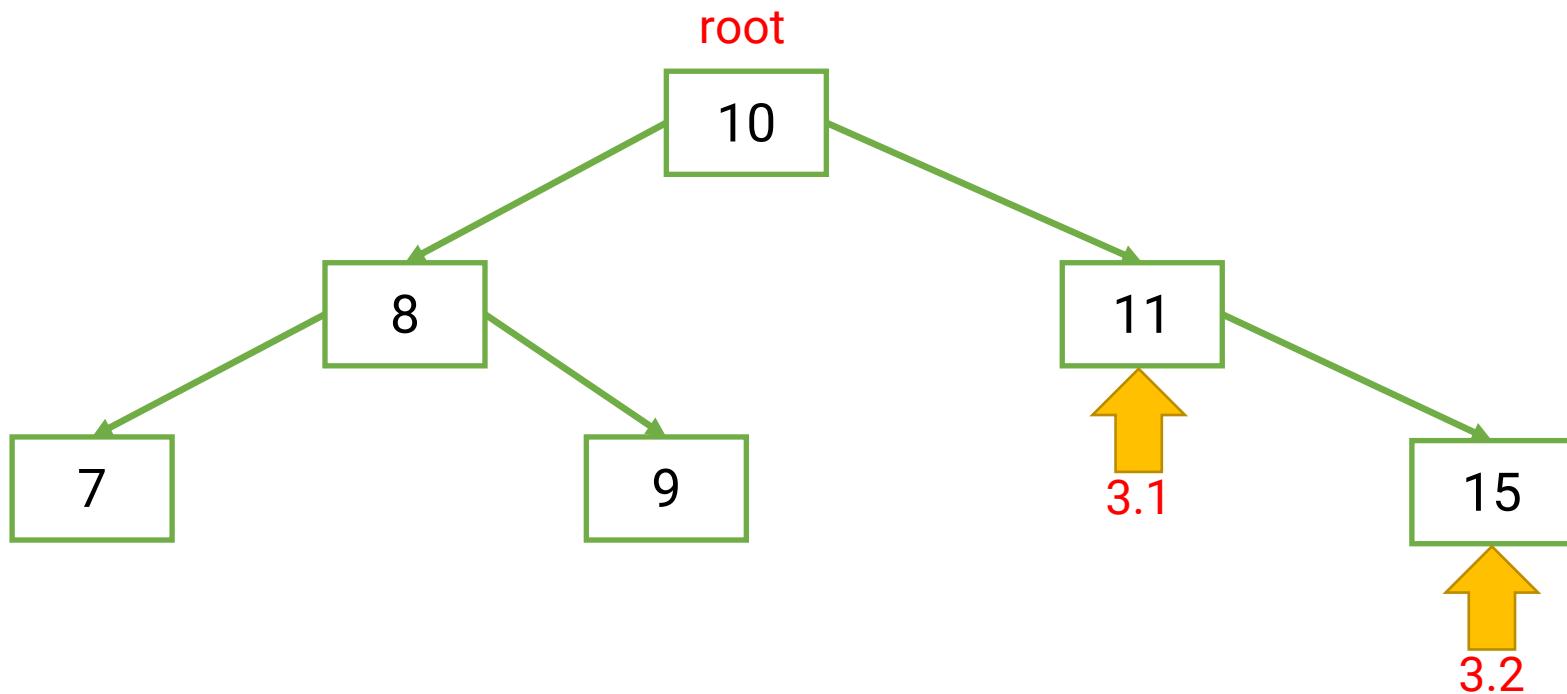
Preorder traversal – NLR



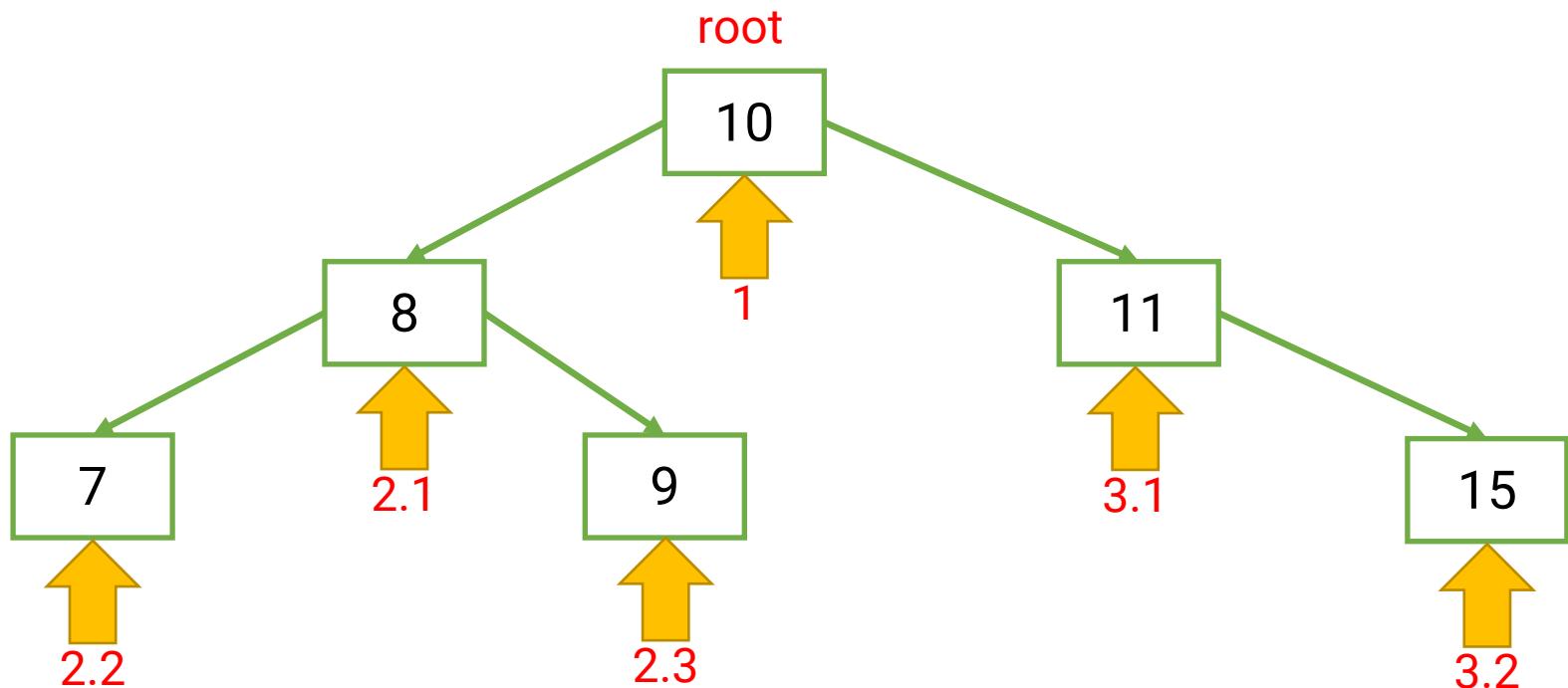
Preorder traversal – NLR



Preorder traversal – NLR

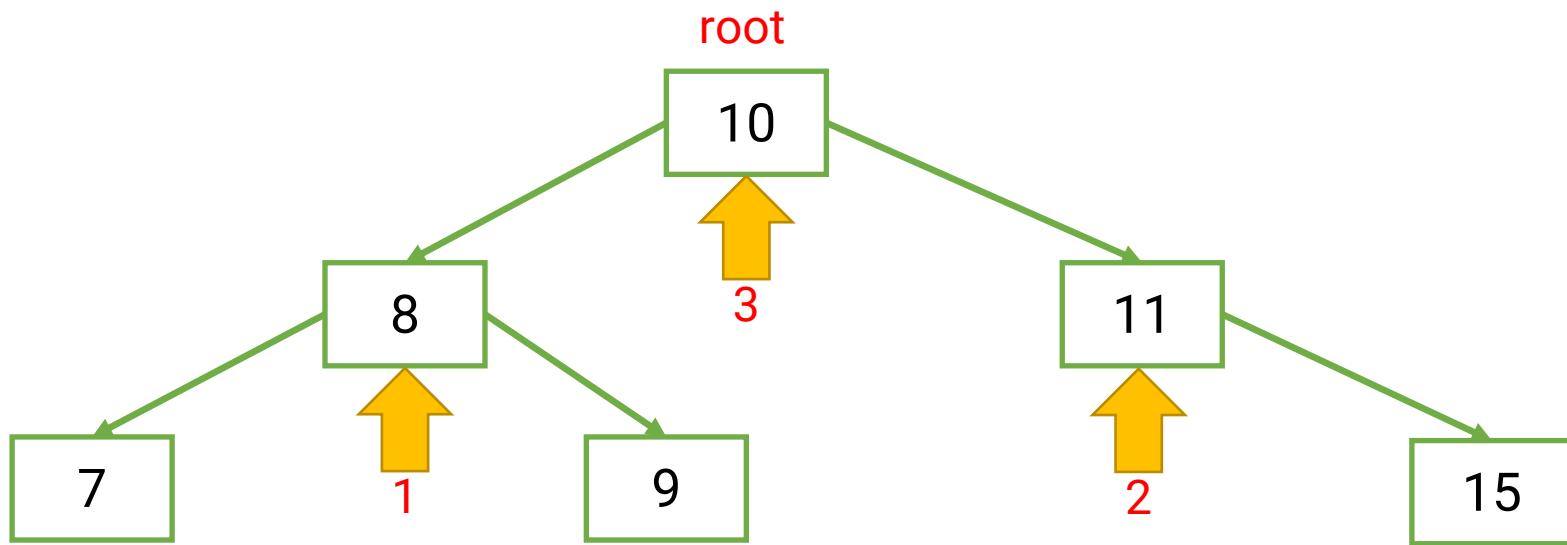


Preorder traversal – NLR

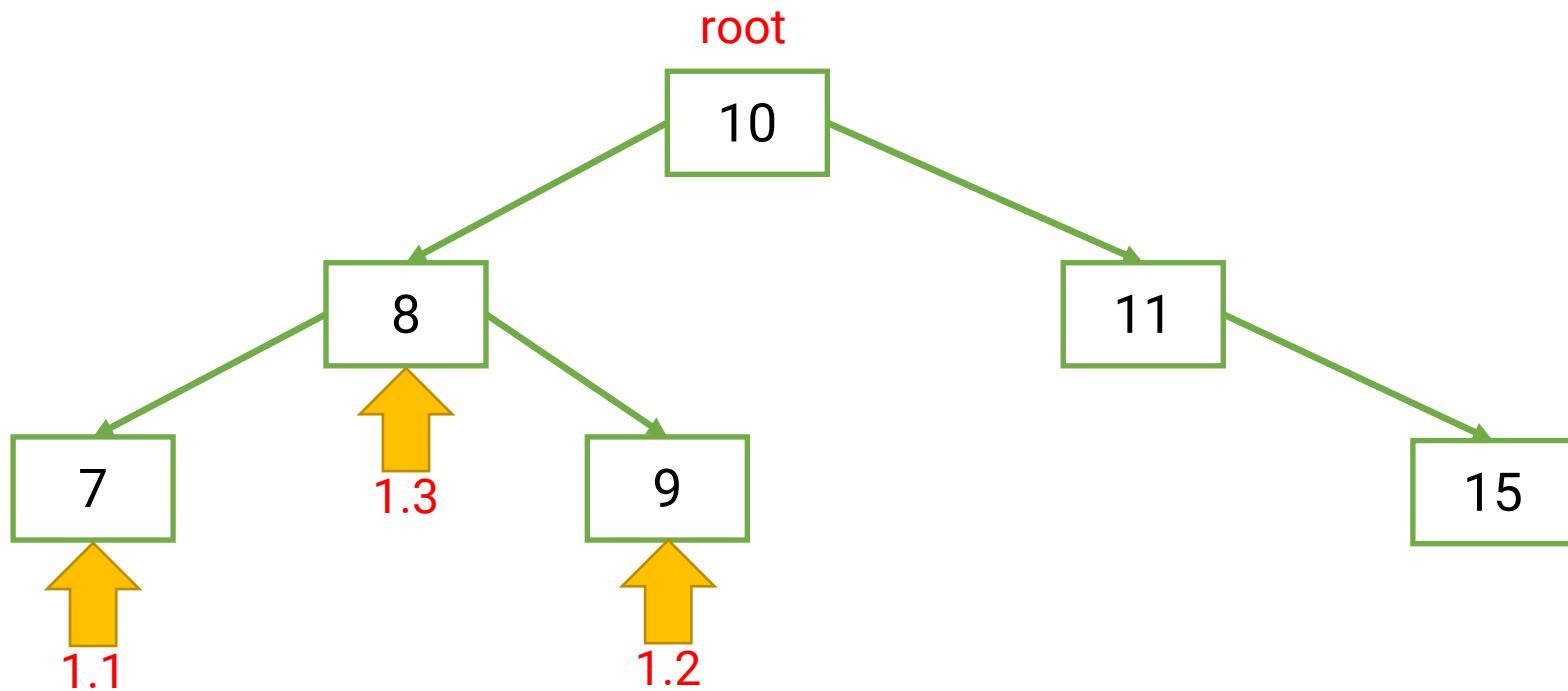


10, 8, 7, 9, 11, 15

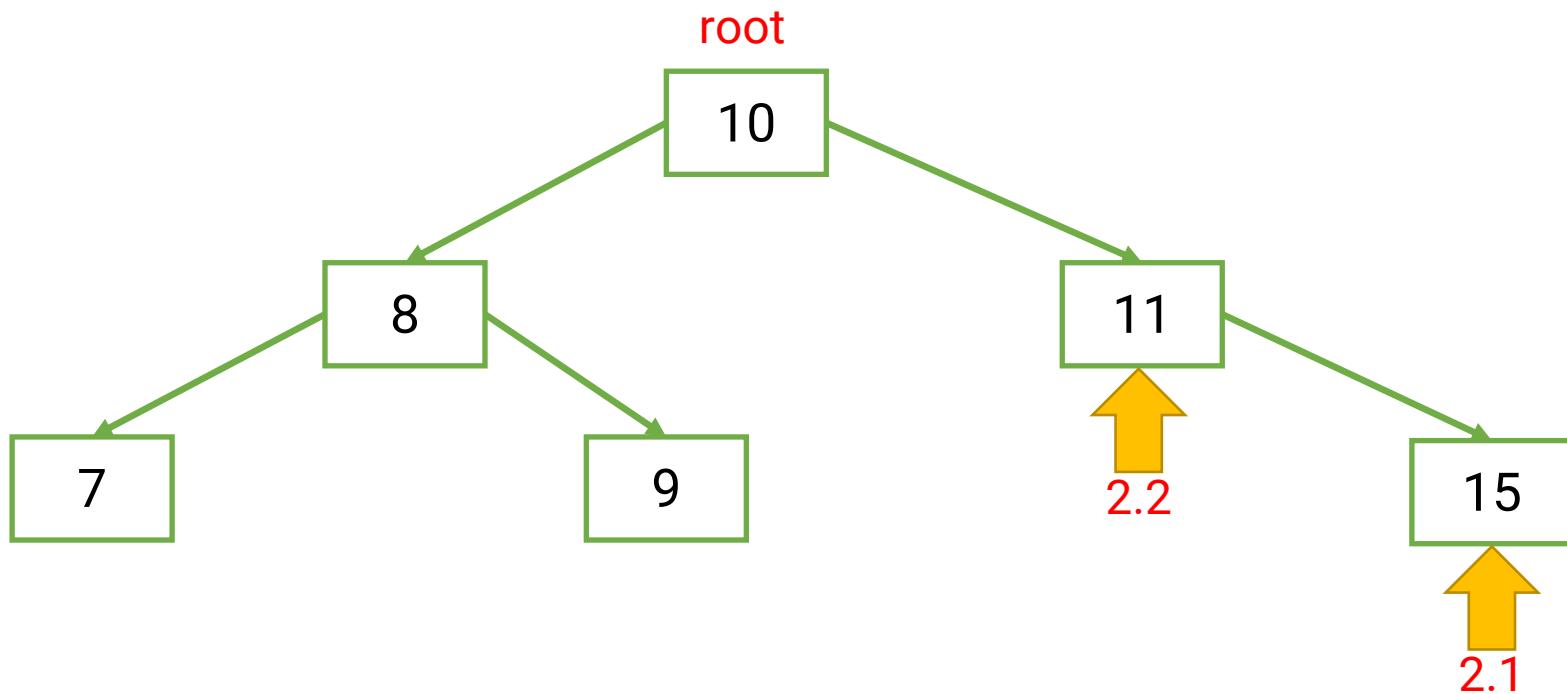
Postorder traversal – LRN



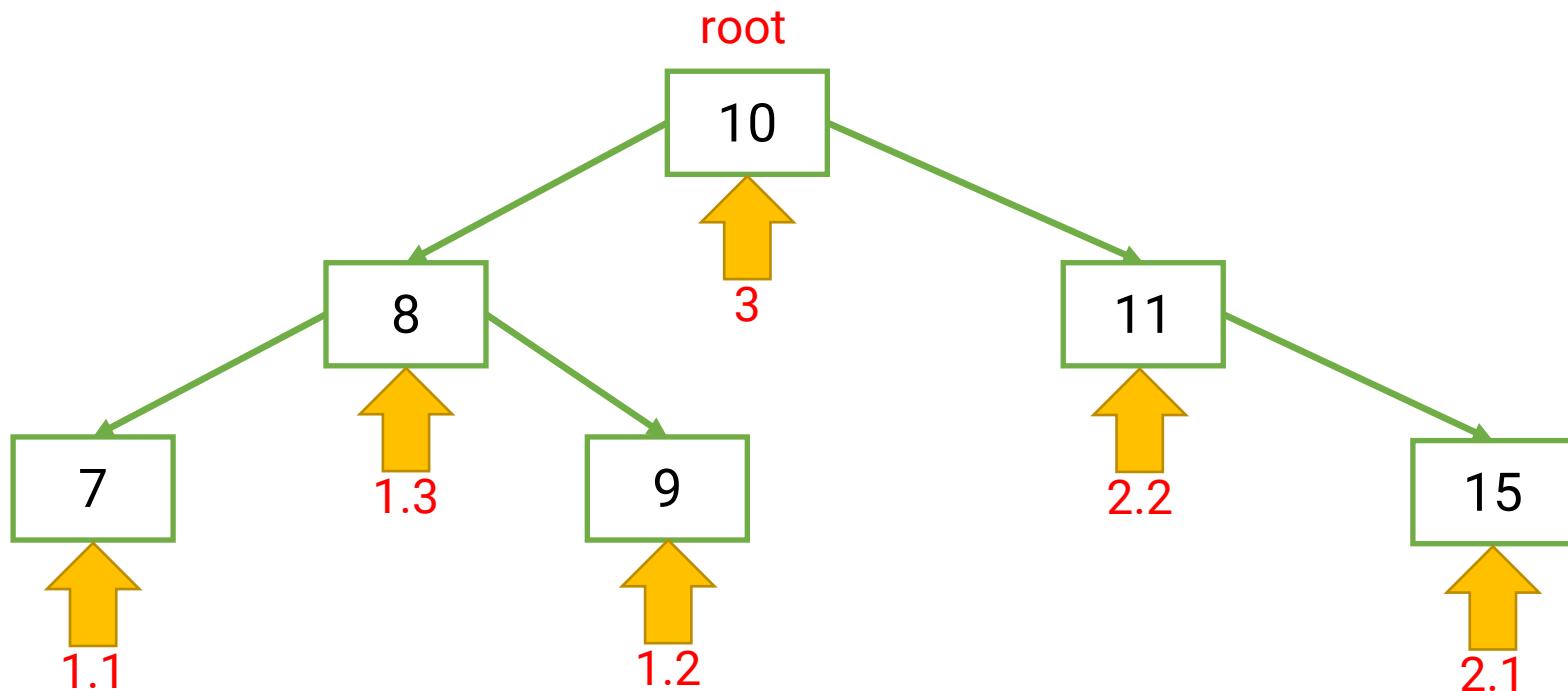
Postorder traversal – LRN



Postorder traversal – LRN



Postorder traversal – LRN



7, 9, 8, 15, 11, 10

BT3 – TÍNH CHIỀU CAO CỦA CÂY

- Cho một cây nhị phân tính chiều cao của cây.



Gợi ý hàm getHeighBST()

- Gọi hàm getHeightNode() của root.

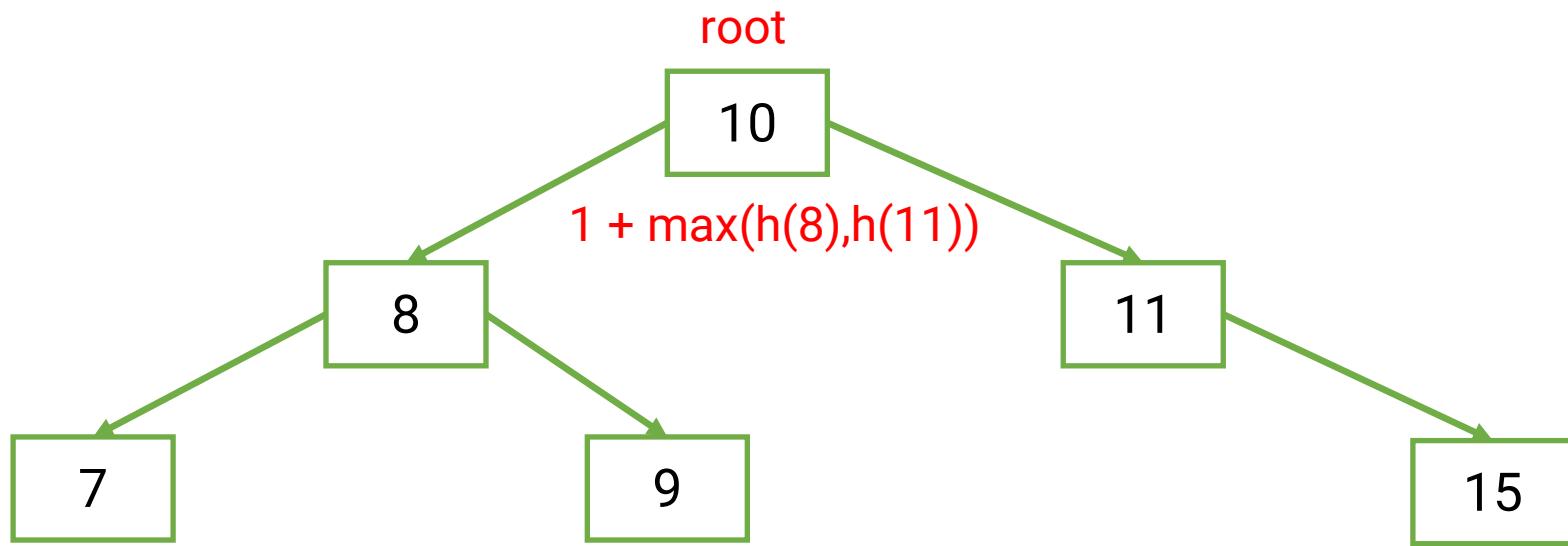


Gợi ý hàm getHeightNode()

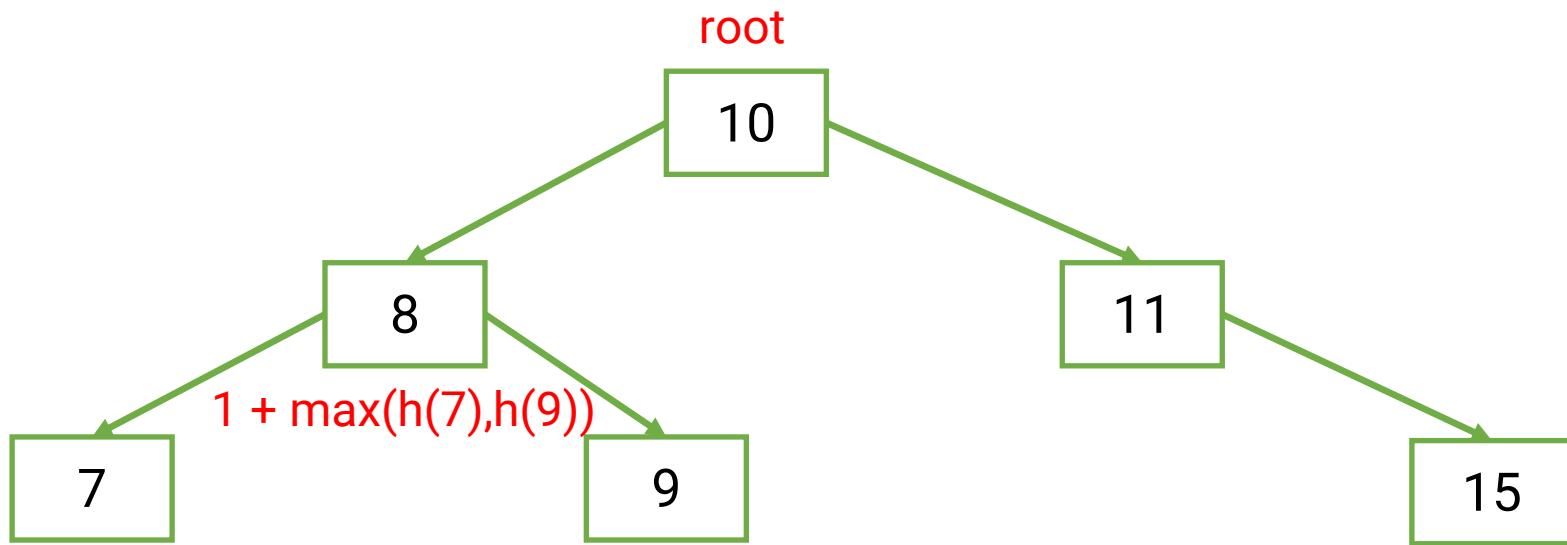
- Nếu left khác NULL / null / None,
 - $hL = left.getHeightNode()$.
- Nếu right khác NULL / null / None,
 - $hR = right.getHeightNode()$.
- $\text{height}(\text{node hiện tại}) = 1 + \max(hL, hR)$.



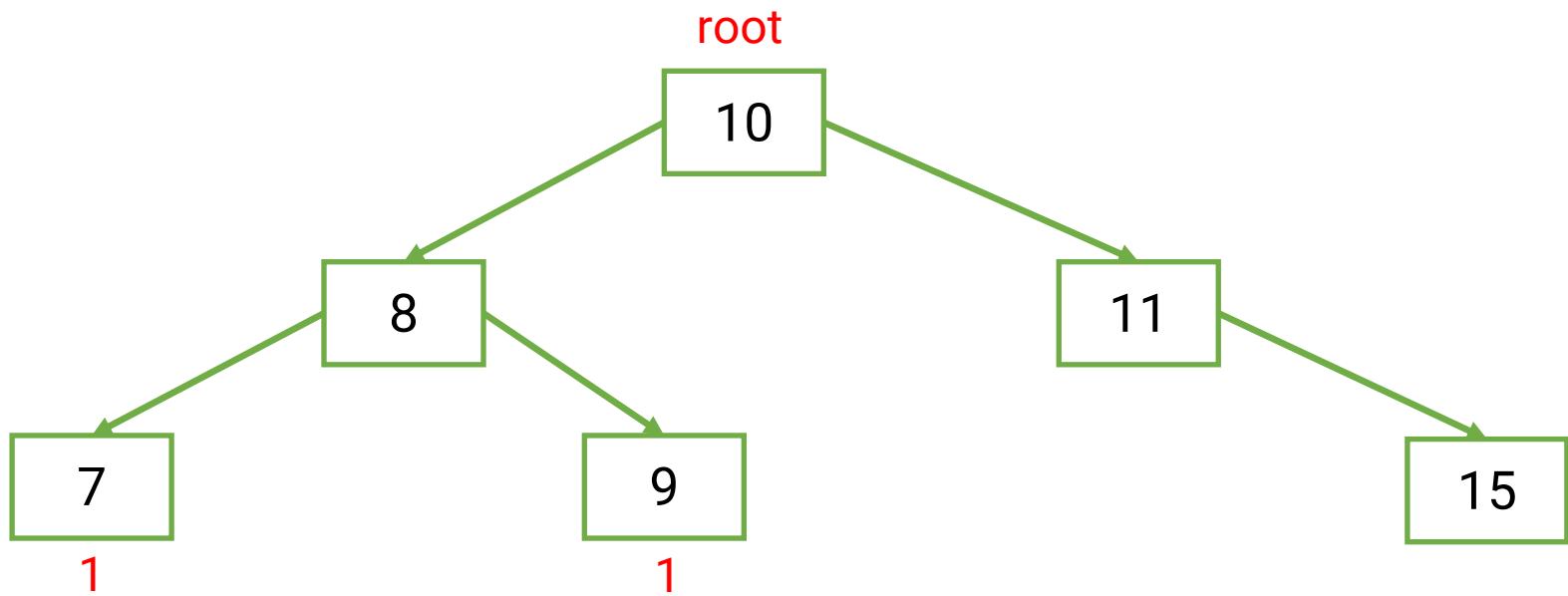
Minh họa hàm getHeightBST()



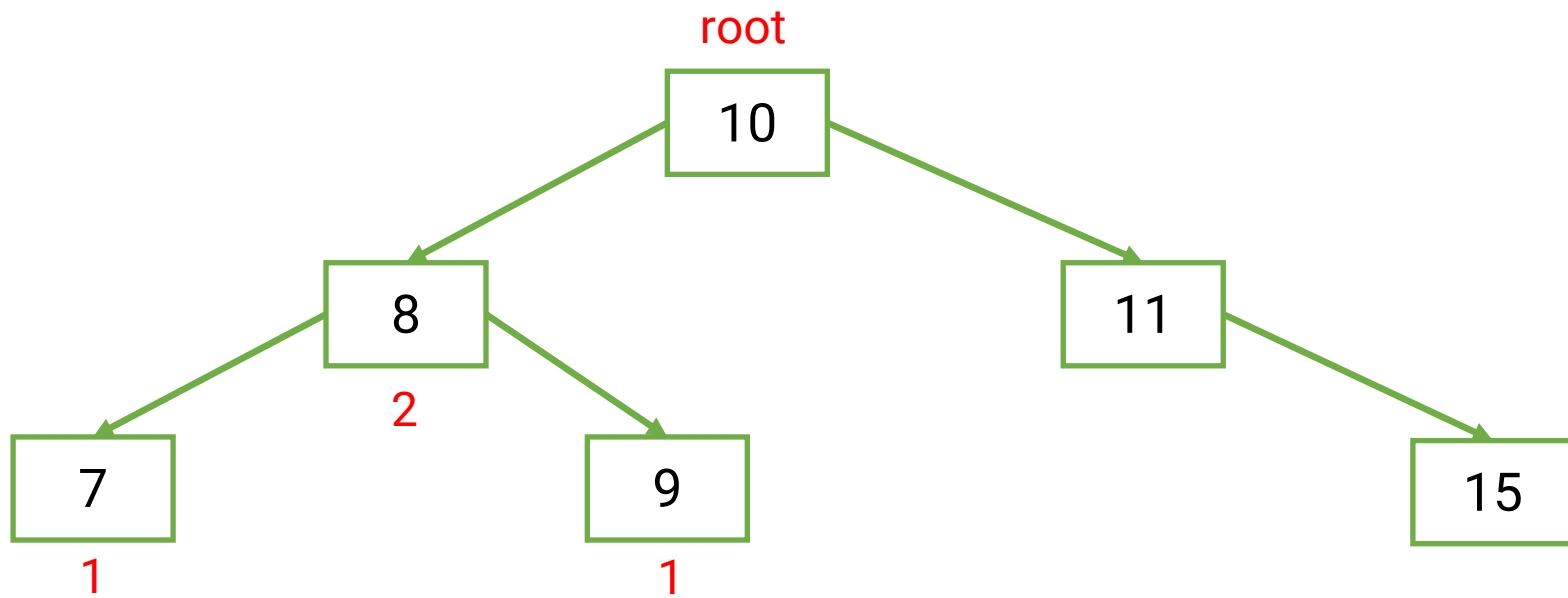
Minh họa hàm getHeightBST()



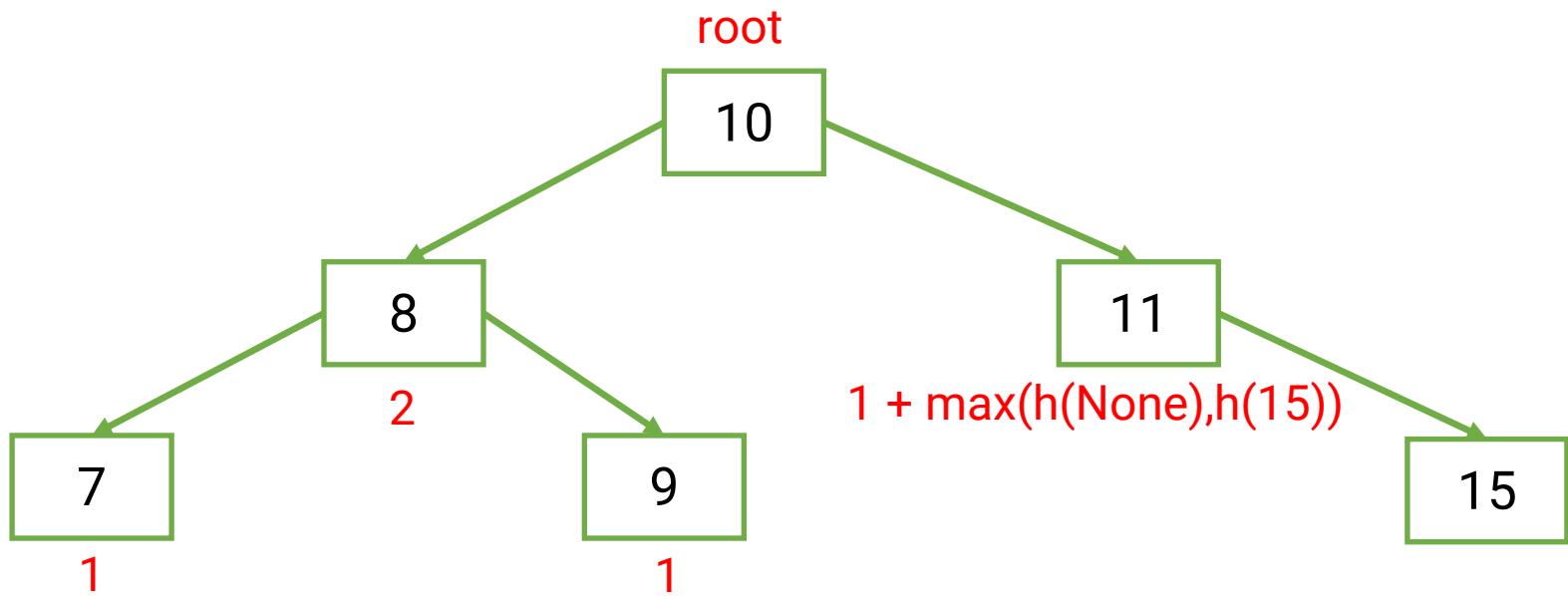
Minh họa hàm getHeightBST()



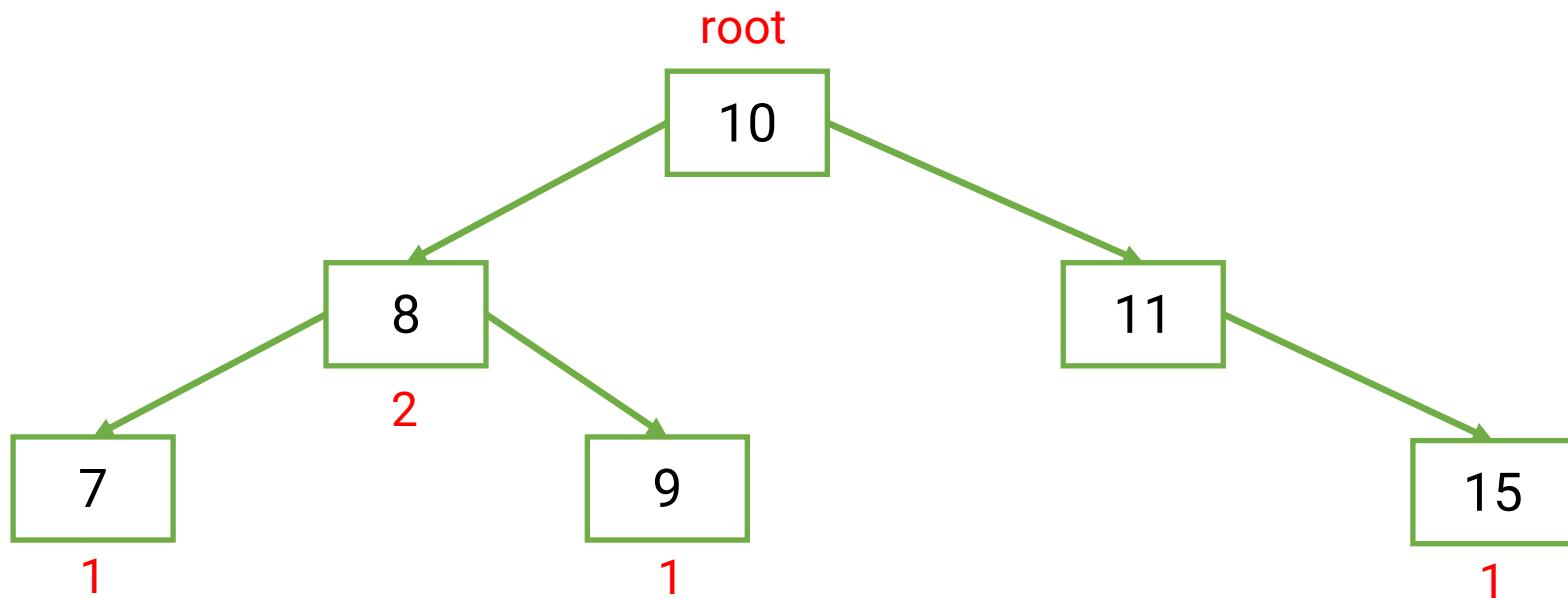
Minh họa hàm getHeightBST()



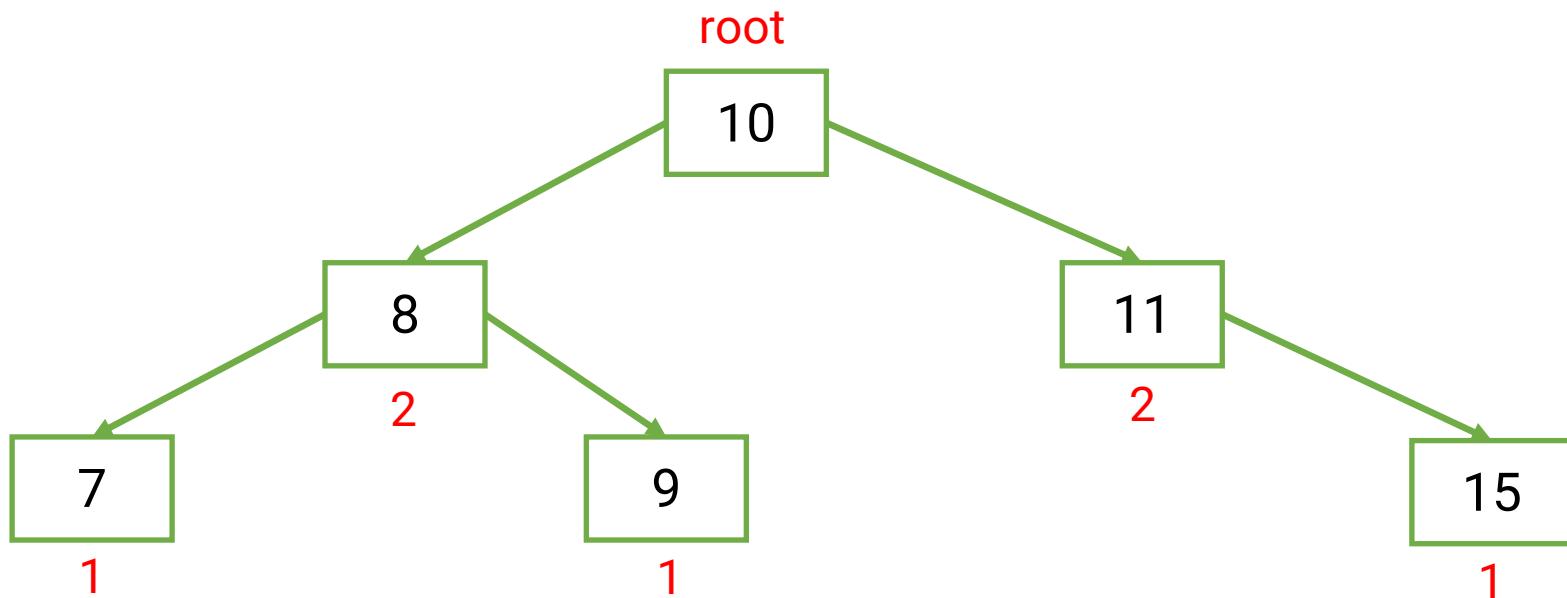
Minh họa hàm getHeightBST()



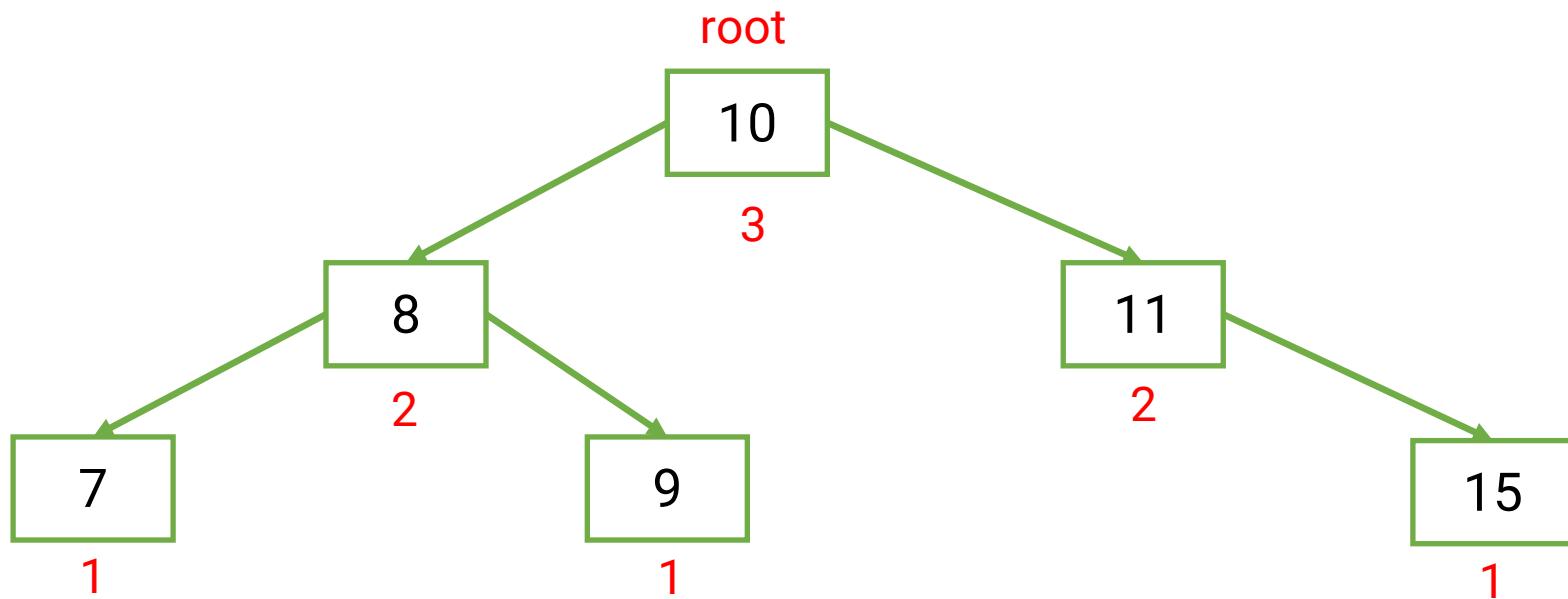
Minh họa hàm getHeightBST()



Minh họa hàm getHeightBST()



Minh họa hàm getHeightBST()



BT4 – TỔNG NÚT LÁ CÂY

- Cho N số, bạn hãy tạo thành cây nhị phân tìm kiếm và tính tổng các nút lá của cây.



Gợi ý hàm sumLeafBST()

- Gọi hàm sumLeafNode() của root.

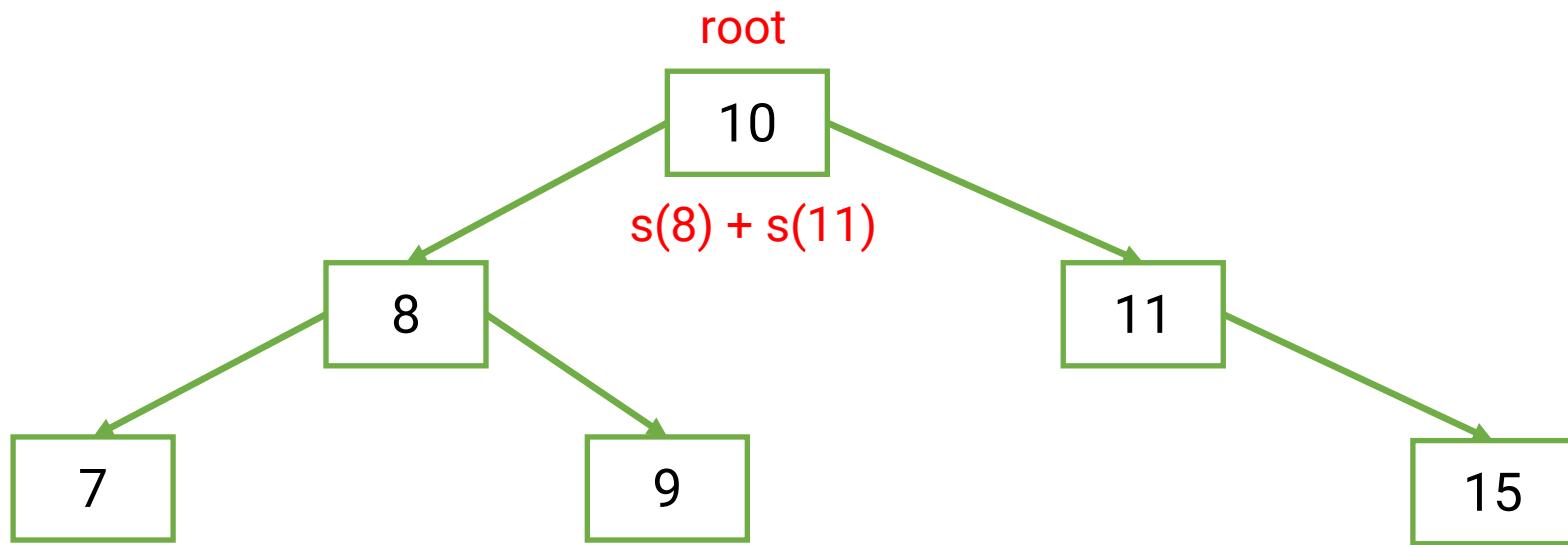


Gợi ý hàm sumLeafNode()

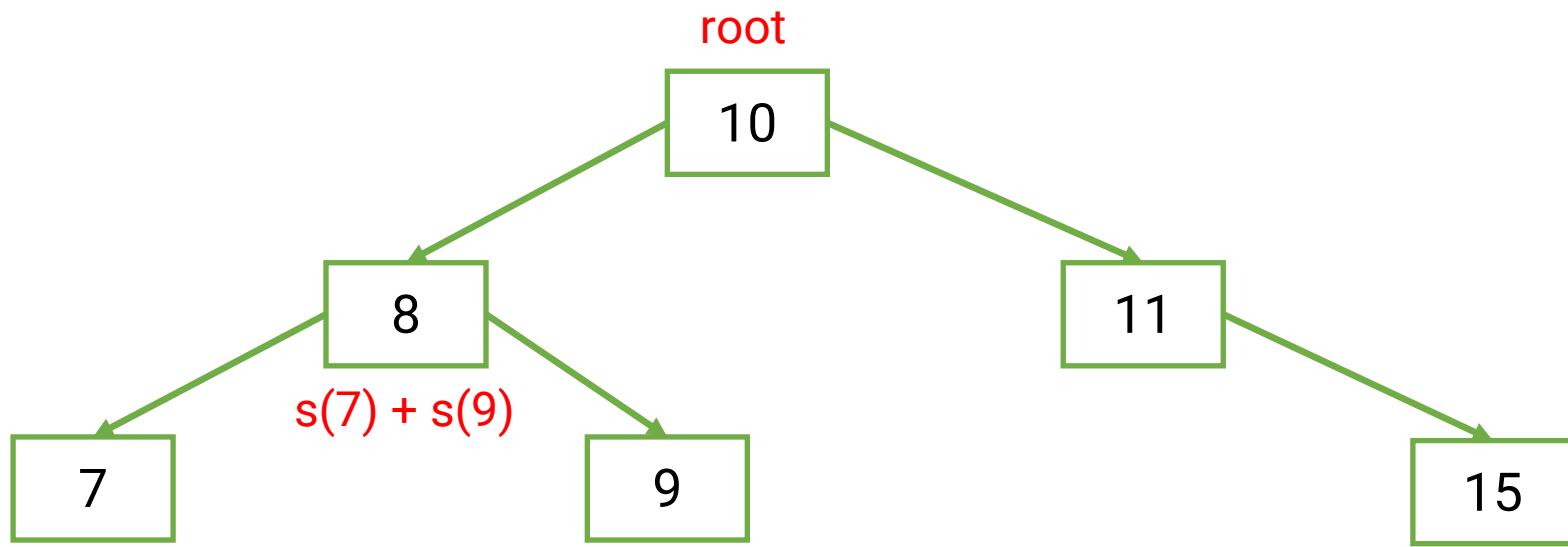
- Nếu node hiện tại là node lá,
 - return data.
- Ngược lại, (node hiện tại không phải là node lá)
 - Trả về left.sumLeafNode() + right.sumLeafNode().



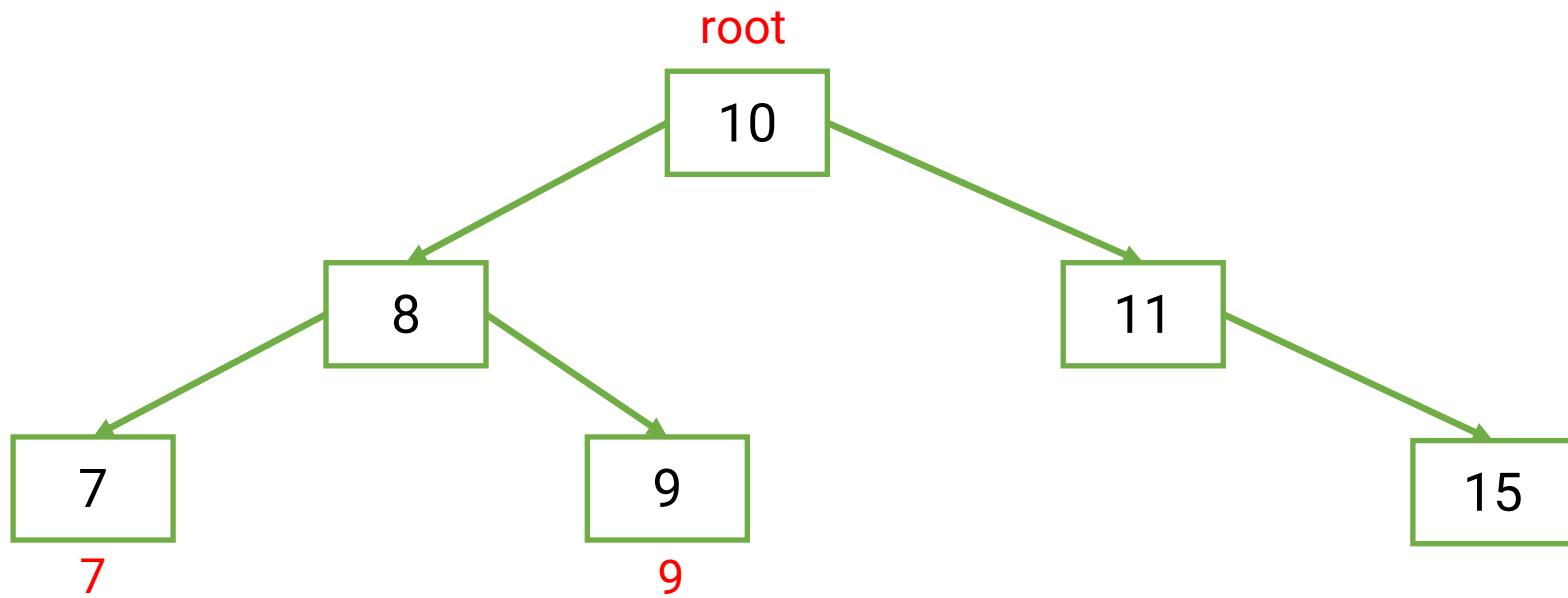
Minh họa hàm sumLeafBST()



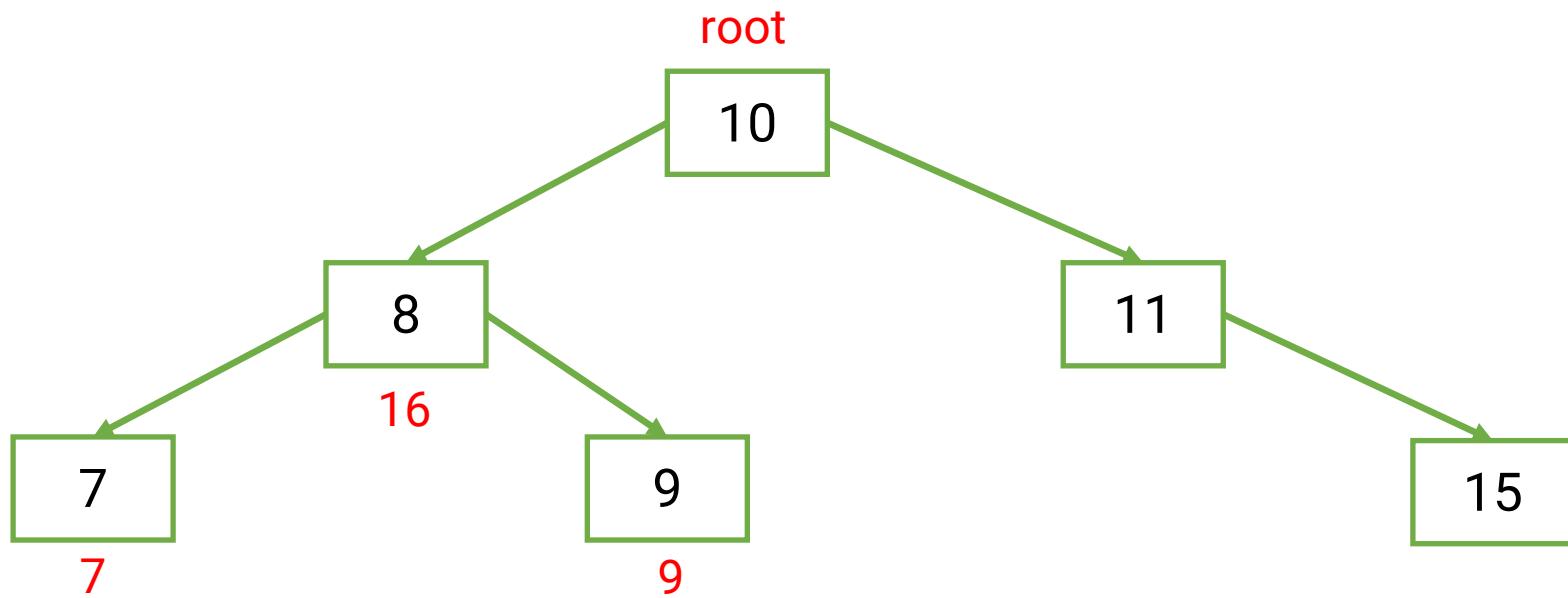
Minh họa hàm sumLeafNode()



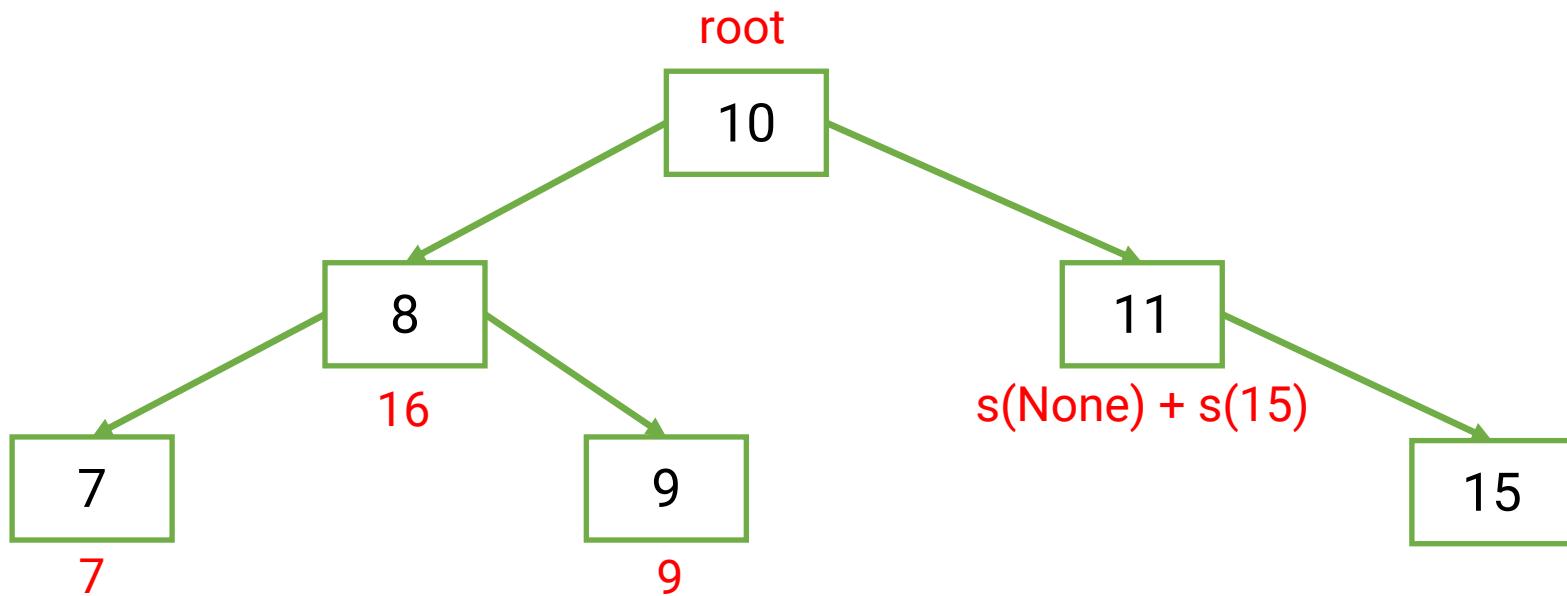
Minh họa hàm BST.sumLeaf()



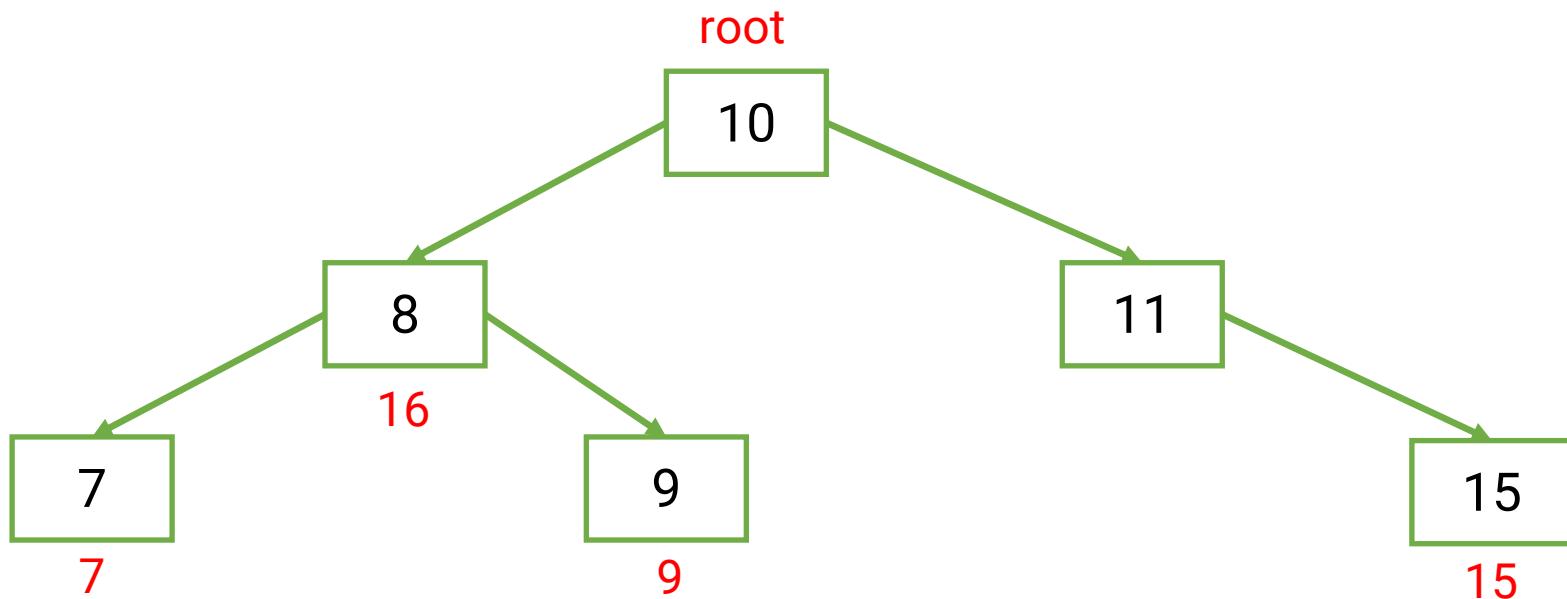
Minh họa hàm BST.sumLeaf()



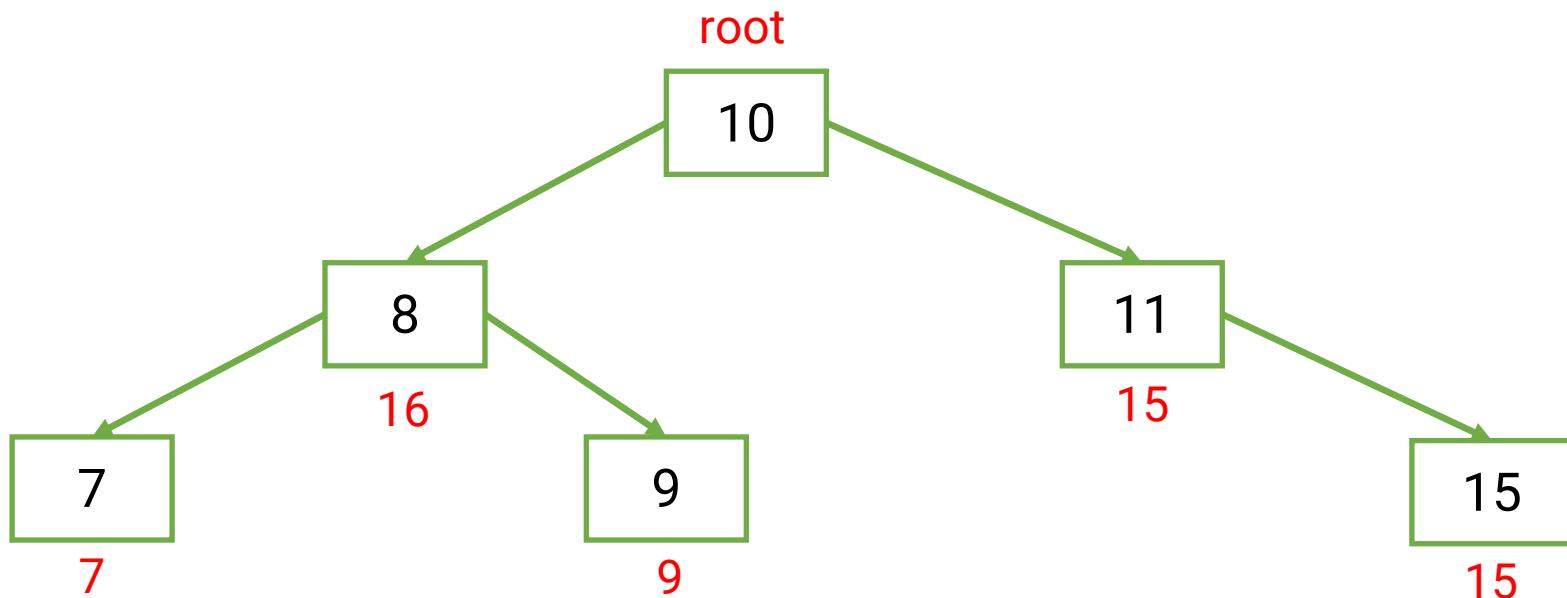
Minh họa hàm BST.sumLeaf()



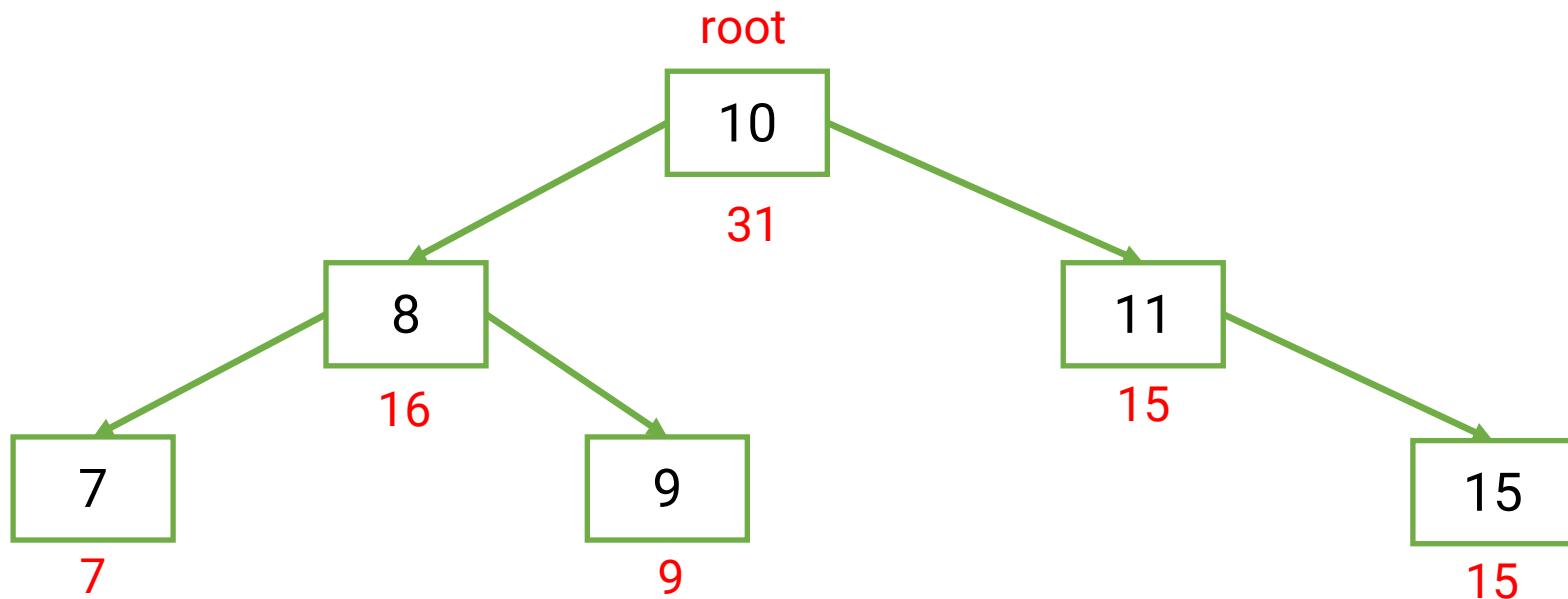
Minh họa hàm sumLeafNode()



Minh họa hàm BST.sumLeaf()



Minh họa hàm sumLeafNode()

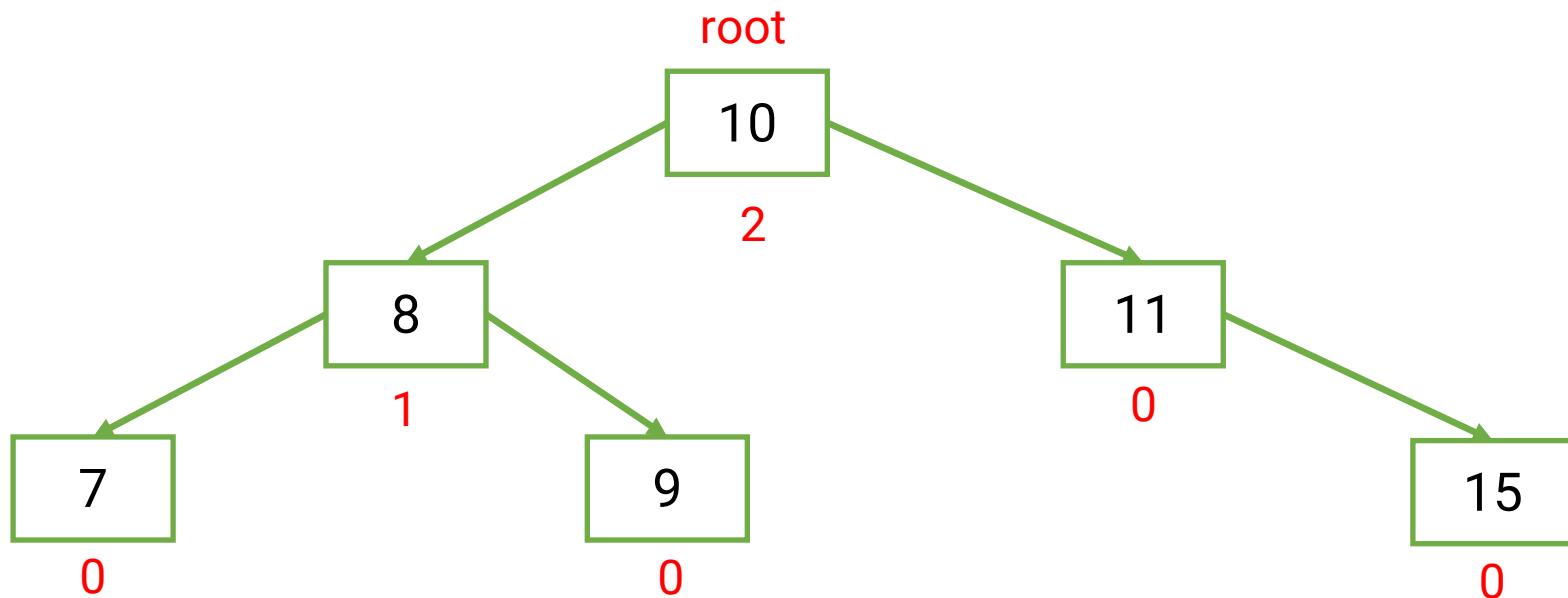


BT5 – ĐẾM SỐ NÚT CÓ ĐÚNG 2 CON

- Cho N số, bạn hãy tạo thành cây nhị phân tìm kiếm và đếm số nút có đúng 2 con.



Minh họa hàm countFullBST()



Hỏi đáp

