hierarchical data structure

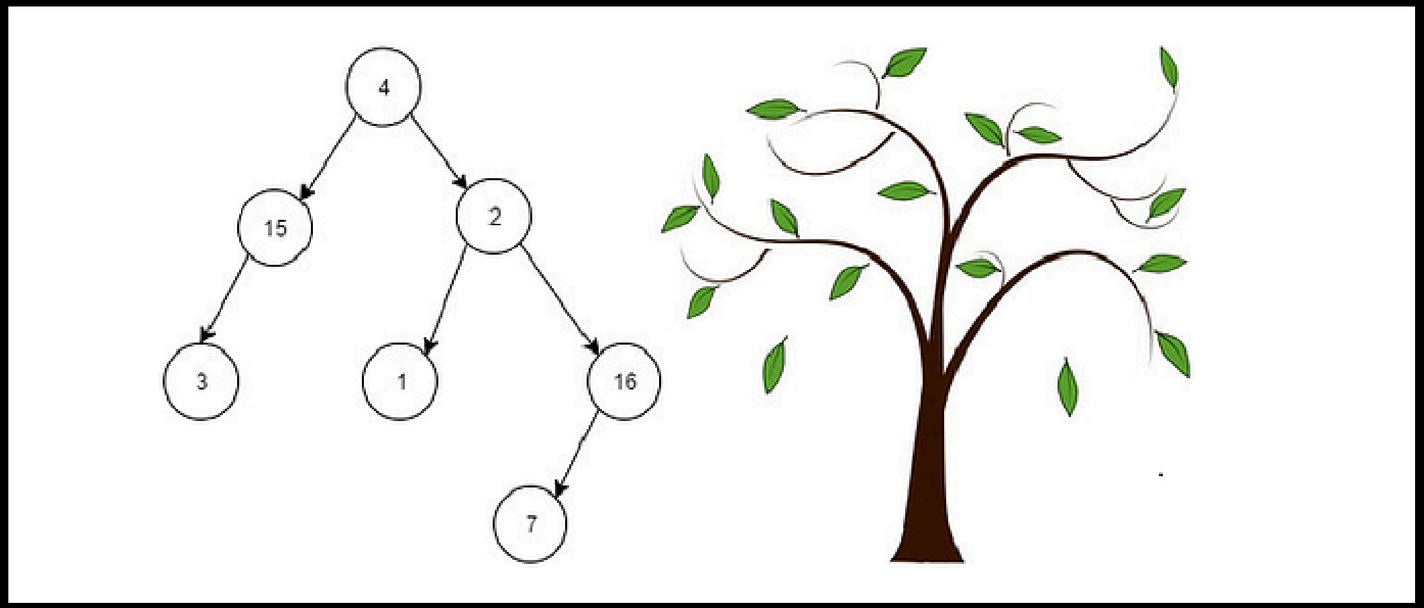
Tree

P’ มนัส

AKA P’many / P’JadeS

โครงสร้างแบบลำดับชั้น (hierarchical structure) ที่ใช้ในการแสดงและจัดระเบียบข้อมูลในลักษณะที่ง่ายต่อการค้นหา
ข้อมูลถูกเก็บเป็น Node ที่เชื่อมต่อกันด้วย Edge แสดงความสัมพันธ์แบบลำดับชั้นระหว่างโหนดต่างๆ

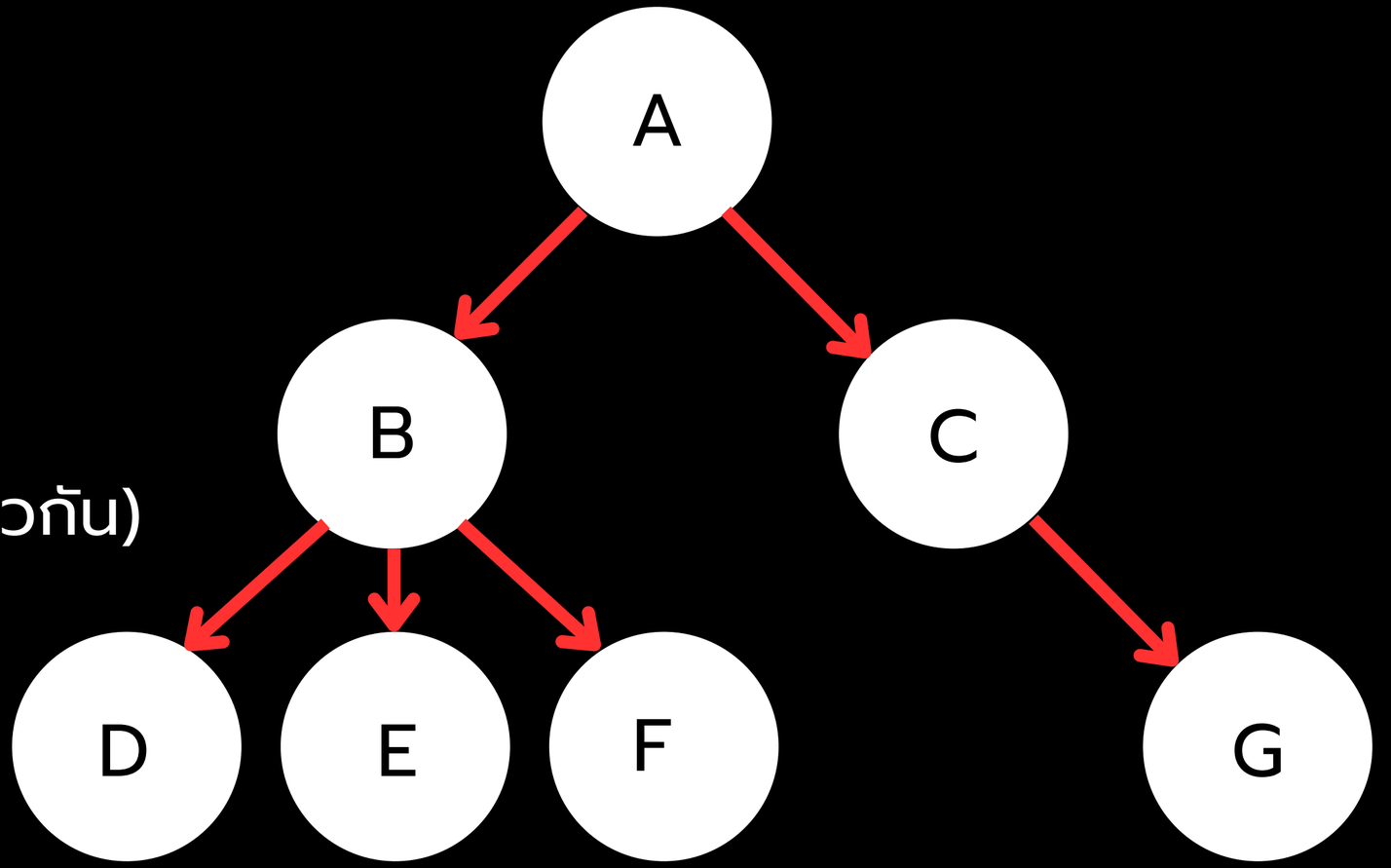
เหมือนกับแผนผังต้นไม้ ที่มีความสัมพันธ์ด้านในเป็นลำดับชั้น



https://code.likeagirl.io/all-about-tree-data-structure-0ec7eb645295

Tree Structure

- parent (โหนดแม่)
- child (โหนดลูก)
- edge (เส้นเชื่อม)
- sibling (โหนดลูกที่มาจากแม่เดียวกัน)

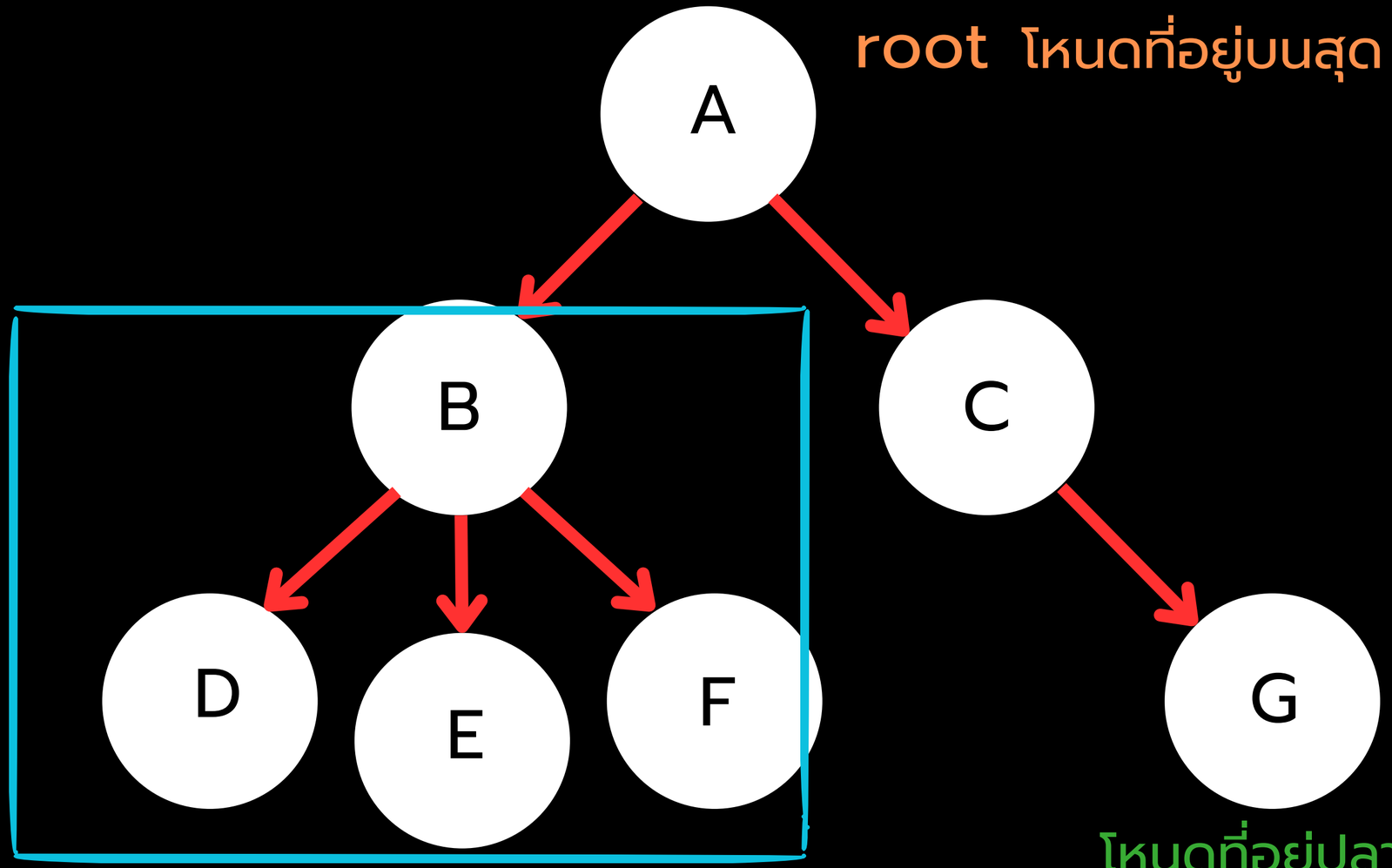


A เป็น Parent ของ B,C

G เป็น child ของ C

D,E,F เป็น sibling ของกันและกัน(มี B เป็น parent)

Tree Structure



root โหนดที่อยู่บนสุด ไม่มี parent

subtree

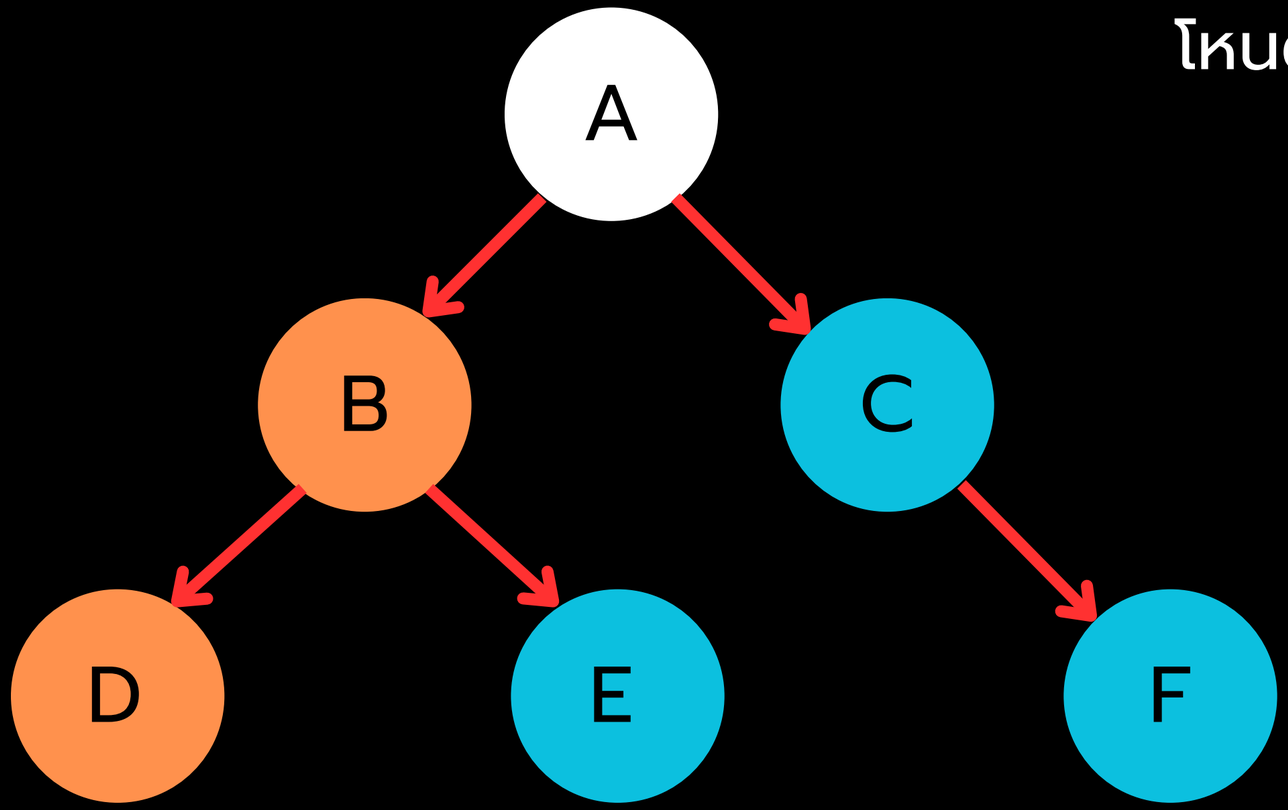
กลุ่มของโหนดที่เชื่อมกันเป็น tree ย่อย

leaf

โหนดที่อยู่ปลายสุด ไม่มี child

Binary Tree

A มี B เป็น left Child และ C เป็น Right Child



โหนดแม่สามารถมีโหนดลูกได้ไม่เกิน 2 โหนด

Left ChildNode

Right ChildNode

C มี left Child เป็น .....

Right Child เป็น .....

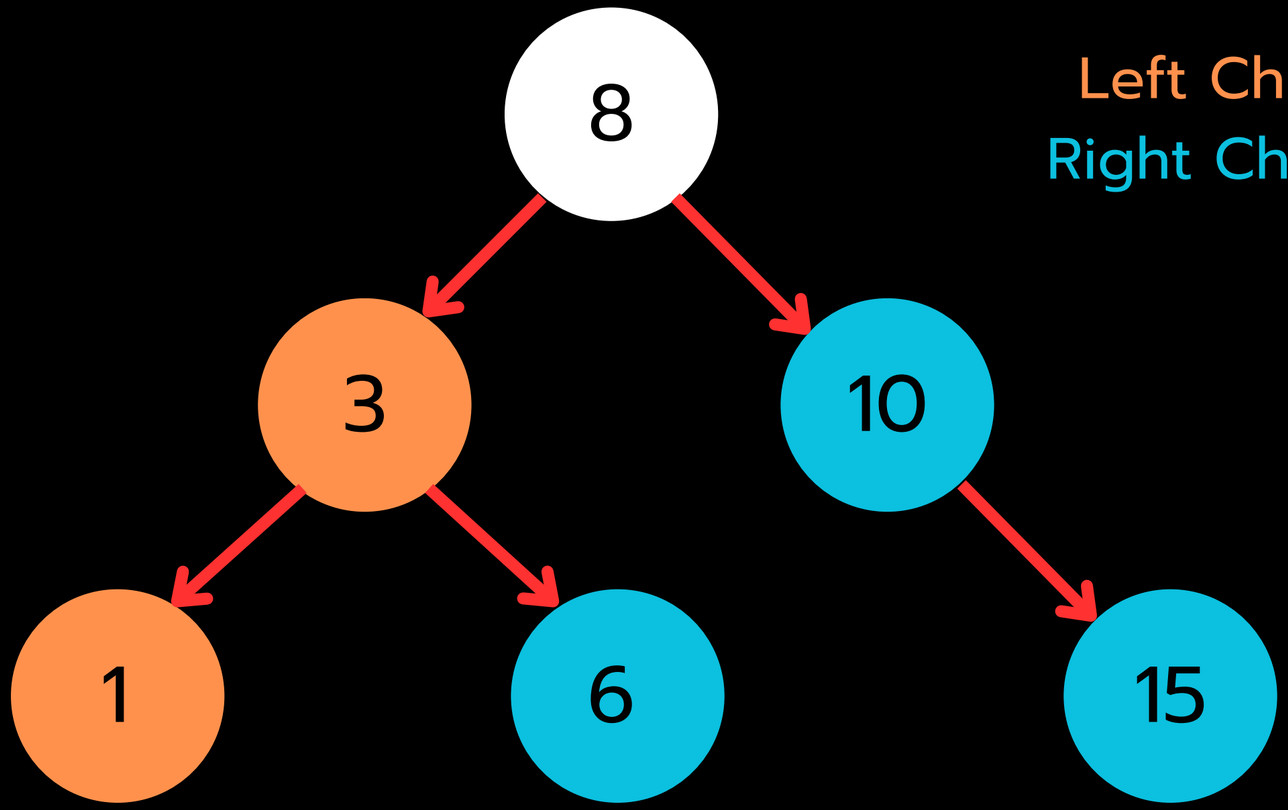
B มี left Child เป็น .....

Right Child เป็น .....

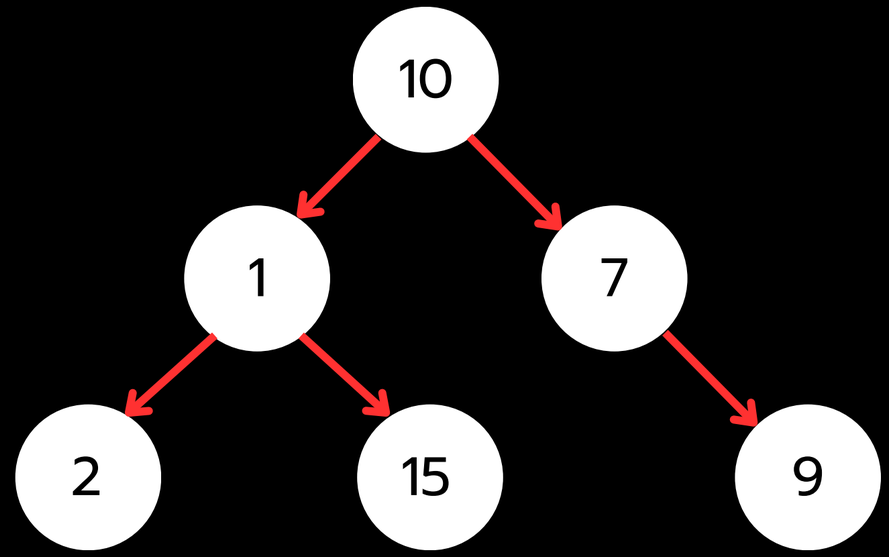
E มี left Child เป็น .....

Right Child เป็น .....

Binary Search Tree

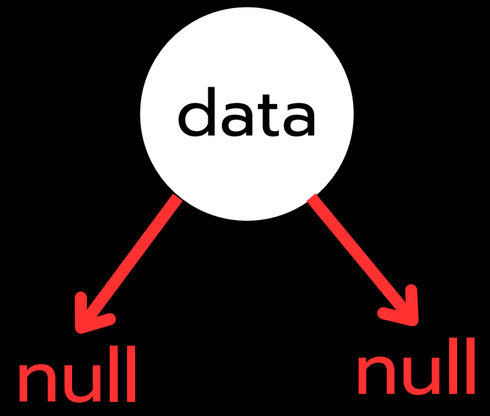


Left ChildNode มีค่า **น้อยกว่า** Parent Node และ
Right ChildNode มีค่า **มากกว่า** Parent Node เสมอ



TreeNode Class

```java
public class Node{
        int data;
        Node left, right;
        public Node(int d) {
            data = d;
            left = null;
            right = null;
        }
```




```java
public String toString() {
    if(left != null && right != null){//both children
        return left.data +"<-"+ data + "->" + right.data;
    }
    else{
        if(left!=null){// left-child-only,
            return left.data +"<-"+ data + "->null";
        }
        else if(right!=null){// right-child-only,
            return "null<-"+ data + "->" + right.data;
        }
        else{// no child
            return "null<-" + data + "->null";
        }
    }
}
```

แสดง Node เป็น String
ค่าของ Node, ค่าของ Left Child และ Right Child

Binary Search Tree Class

BST.java

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

    public Node getRoot() {
        return root;
    }

    public class Node{
        int data;
        Node left, right;
        public Node(int d) {
            data = d;
            left = null;
            right = null;
        }

        @Override
        public String toString()
```

กำหนดค่าเริ่มต้นของ Tree เมื่อถูกสร้าง

method ให้ return root Node

สร้าง Node เป็น inner Class

method แสดงค่าเป็น String ของ Class Node

BSTtest.java

```java
public class BSTtest {
    public static void main(String[] args) {
        BST bst = new BST();

        System.out.println(bst.getRoot());
    }
}
```

```
null
```

Root

null

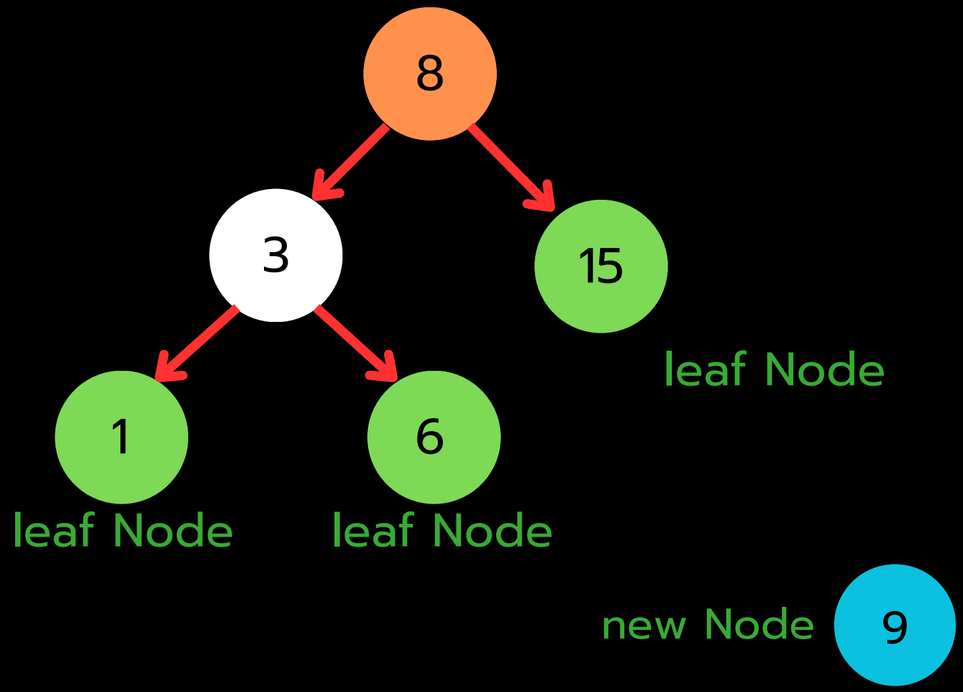
Add tree node

```java
public void insert(int d) {
    if (root == null) {
        root = new Node(d);
    } else {
        Node cur = root;
        while (cur != null) {
            if (d < cur.data) {//go left
                if (cur.left != null){
                    cur = cur.left;
                }
                else {
                    cur.left = new Node(d);
                    return;
                }
            } else { //go right
                if (cur.right != null){
                    cur = cur.right;
                }
                else {
                    cur.right = new Node(d);
                    return;
                }
            }
        }//while
    }
}
```

ค่าใหม่น้อยกว่า จะไปทาง left Child

ค่าใหม่มากกว่า จะไปทาง right Child

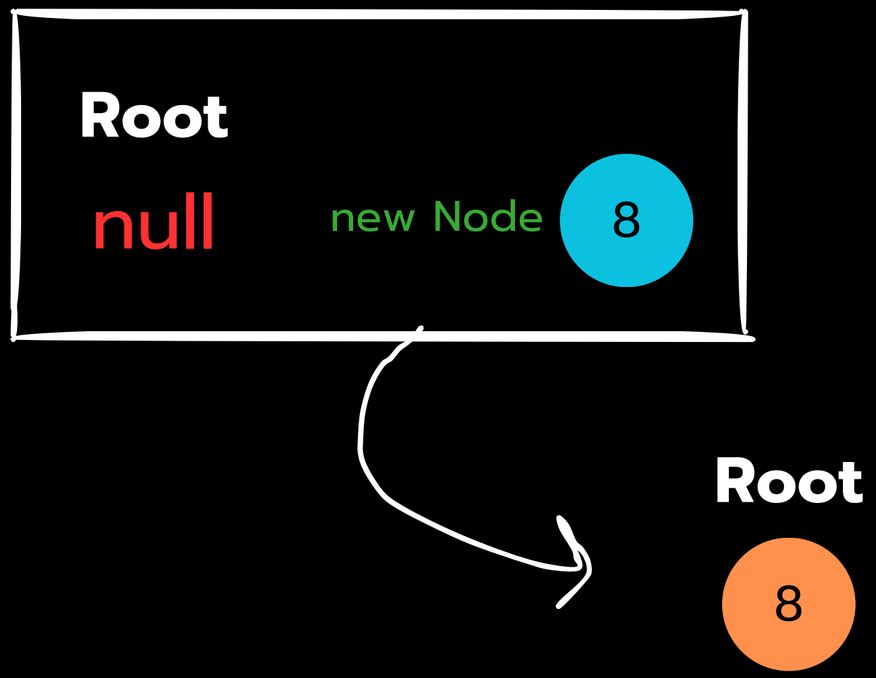
ในการเพิ่มค่า Node ใหม่จะถูก
ต่อ ณ leaf Node เสมอ
โดยมีเงื่อนไขว่า ถ้ามีค่าน้อยกว่าจะไป Node ซ้าย หากค่ามากกว่าจะไปที่ Node ขวา



Add tree node

```
2  public void insert(int d) {
3      if (root == null) {
4          root = new Node(d);
5      } else {
6          Node cur = root;
7          while (cur != null) {
8              if (d < cur.data) {//go left
9                  if (cur.left != null){
10                     cur = cur.left;
11                 }
12                 else {
13                     cur.left = new Node(d);
14                     return;
15                 }
16             } else { //go right
17                 if (cur.right != null){
18                     cur = cur.right;
19                 }
20                 else {
21                     cur.right = new Node(d);
22                     return;
23                 }
24             }
25         }//while
26     }
27 }
```

```
2   bst.insert(8);
```



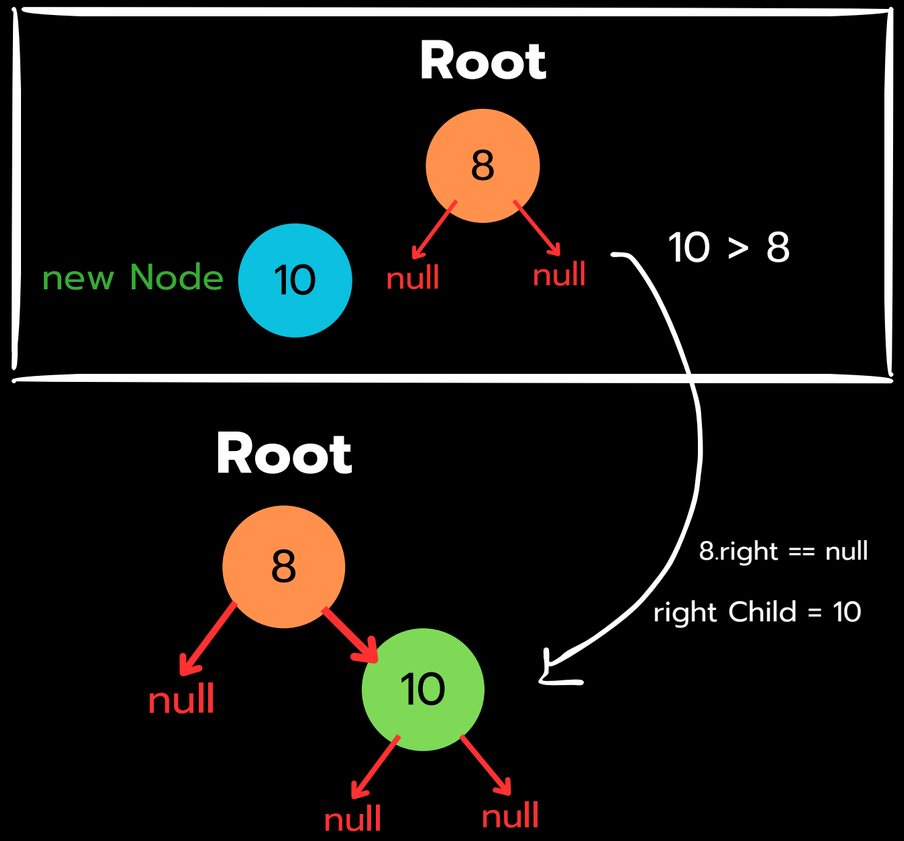

```
System.out.println(bst.getRoot()); //print root (8)
```

```
null<-8->null
```

Add tree node

```java
public void insert(int d) {
    if (root == null) {
        root = new Node(d);
    } else {
        Node cur = root;
        while (cur != null) {
            if (d < cur.data) {//go left
                if (cur.left != null){
                    cur = cur.left;
                }
                else {
                    cur.left = new Node(d);
                    return;
                }
            } else { //go right
                if (cur.right != null){
                    cur = cur.right;
                }
                else {
                    cur.right = new Node(d);
                    return;
                }
            }
        }//while
    }
}
```

```java
bst.insert(10);
```



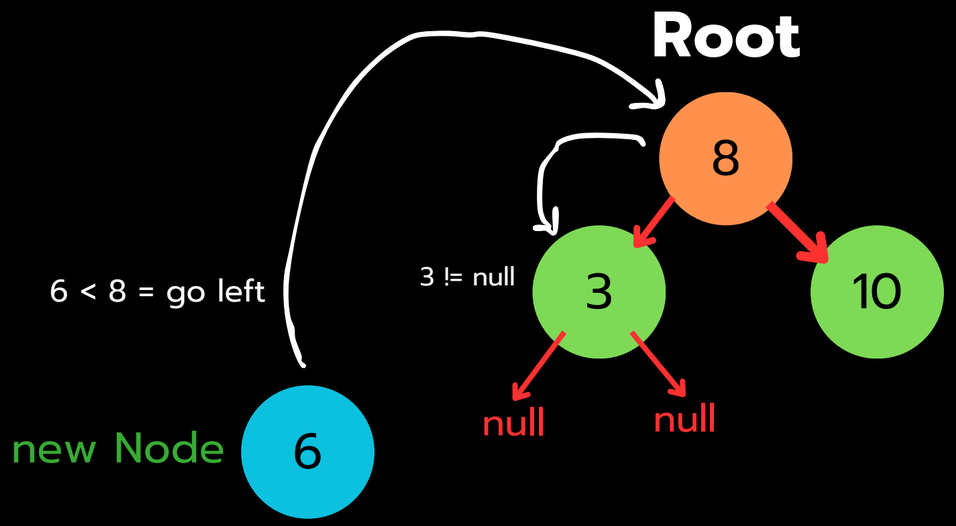

```java
System.out.println(bst.getRoot()); //print root (8)
System.out.println(bst.getRoot().right); //print root.left (10)
```

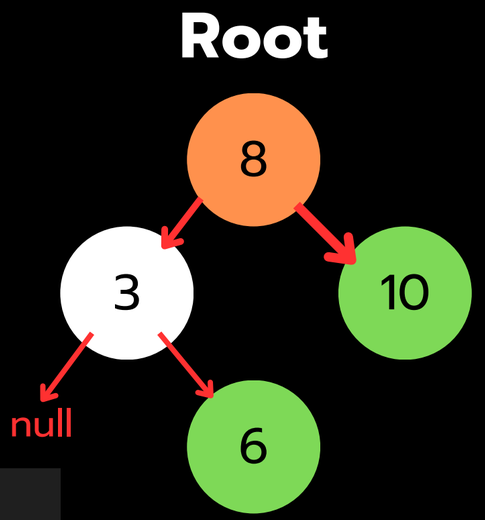
```
null<-8->10
null<-10->null
```

Add tree node

```java
public void insert(int d) {
    if (root == null) {
        root = new Node(d);
    } else {
        Node cur = root;
        while (cur != null) {
            if (d < cur.data) {//go left
                if (cur.left != null){
                    cur = cur.left;
                }
                else {
                    cur.left = new Node(d);
                    return;
                }
            } else { //go right
                if (cur.right != null){
                    cur = cur.right;
                }
                else {
                    cur.right = new Node(d);
                    return;
                }
            }
        }//while
    }
}
```

```java
bst.insert(3);
bst.insert(6);
```





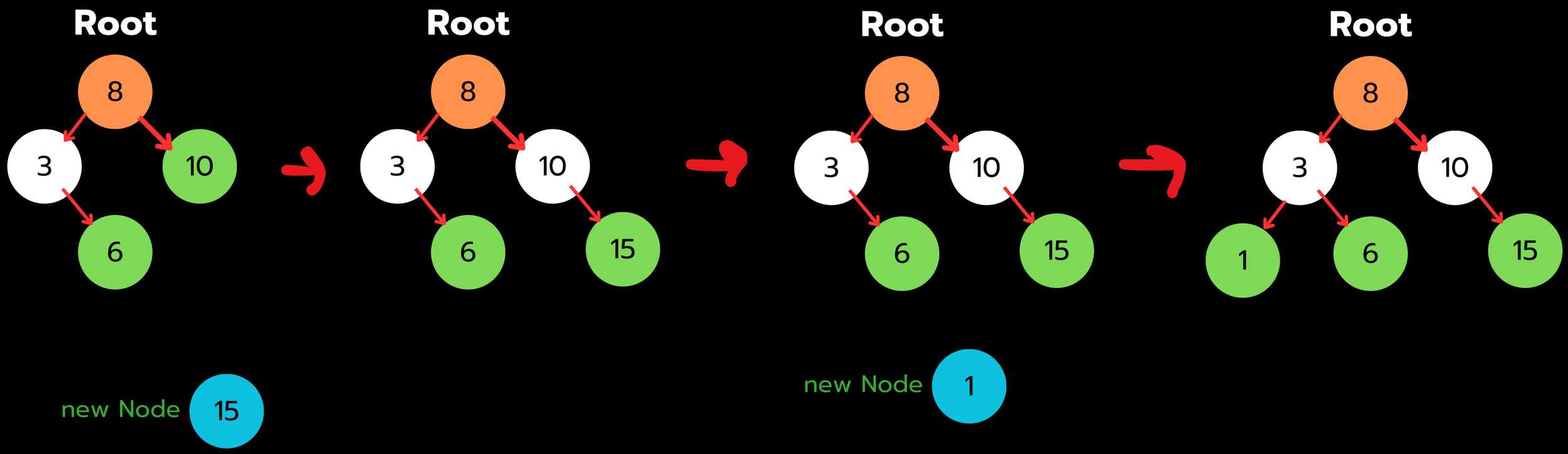

```java
System.out.println(bst.getRoot()); //print (8)
System.out.println(bst.getRoot().left); //print (3)
System.out.println(bst.getRoot().left.right); //print root.left (6)
```

```
3<-8->10
null<-3->6
null<-6->null
```

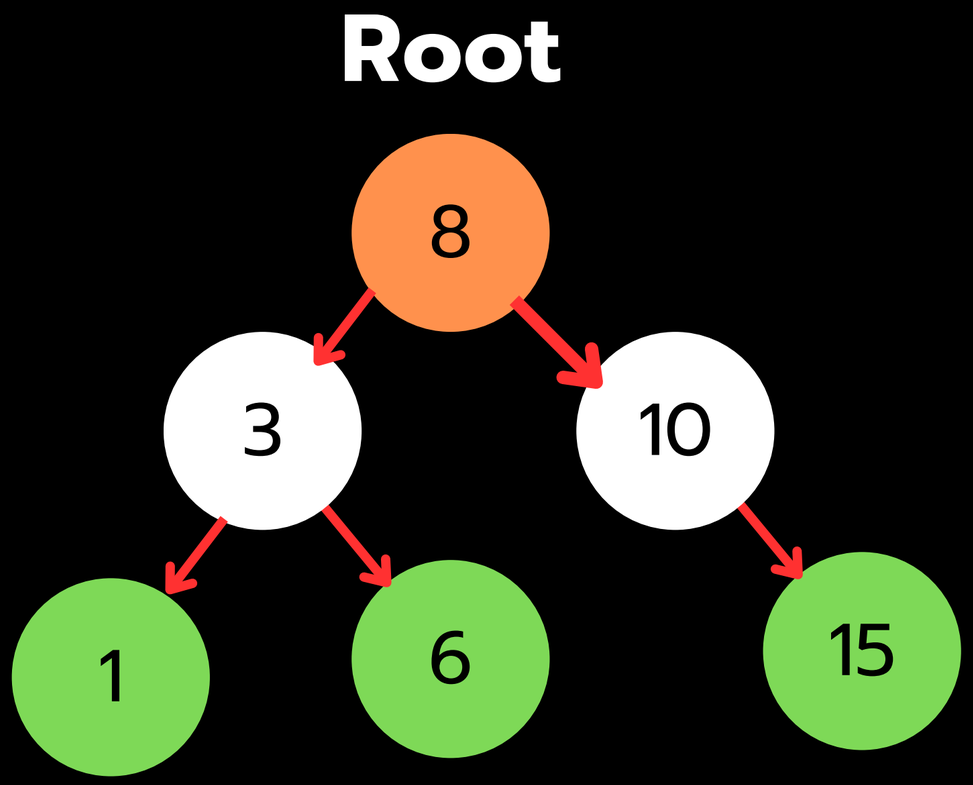
Add tree node

```
6   bst.insert(15);
```

```
7   bst.insert(1);
```



Tree travesal



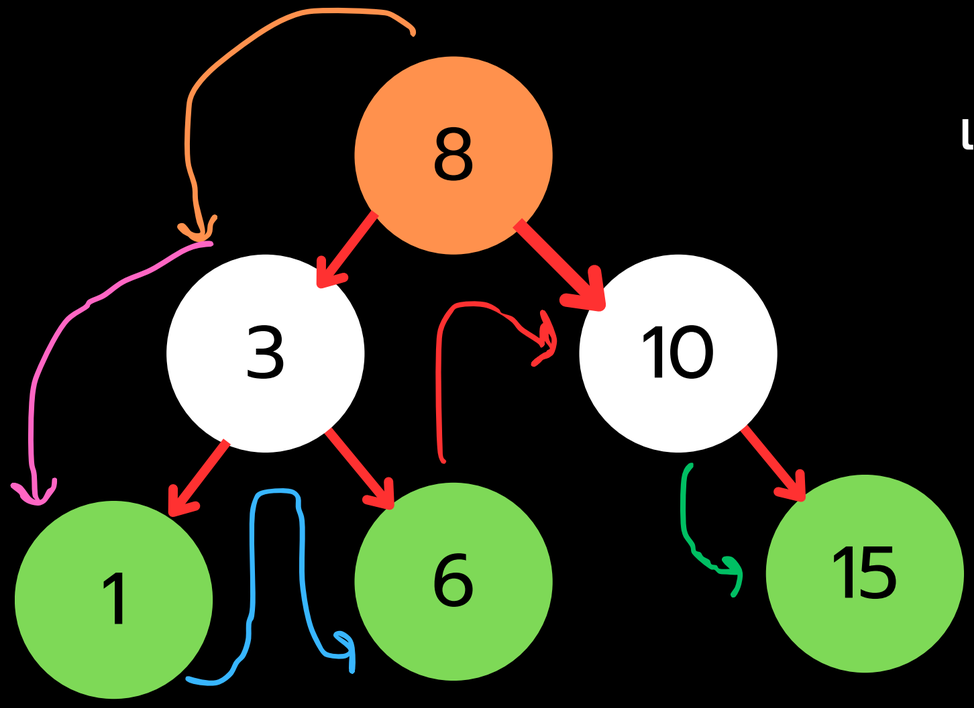
การเข้าถึงข้อมูลที่ถูกเก็บใน tree เนื่องจากข้อมูลถูกเก็บเป็น Node ต่อๆกัน ดังนั้นเวลาไล่ Node จะต้อง เริ่มต้นที่โหนดราก (root) ก่อนเสมอ

Tree travesal มี 3 รูปแบบได้แก่

- Pre-Order
- In-Order
- Post-Order

pre-order

- Pre Order Traversal: จะเริ่มจาก Root, Left, Right



เวลาท่องจะแสดงค่าของ Node ที่เจอออกมาในทันที
ก่อนที่จะวิ่งไปหา Node ถัดไป

8 3 1 6 10 15

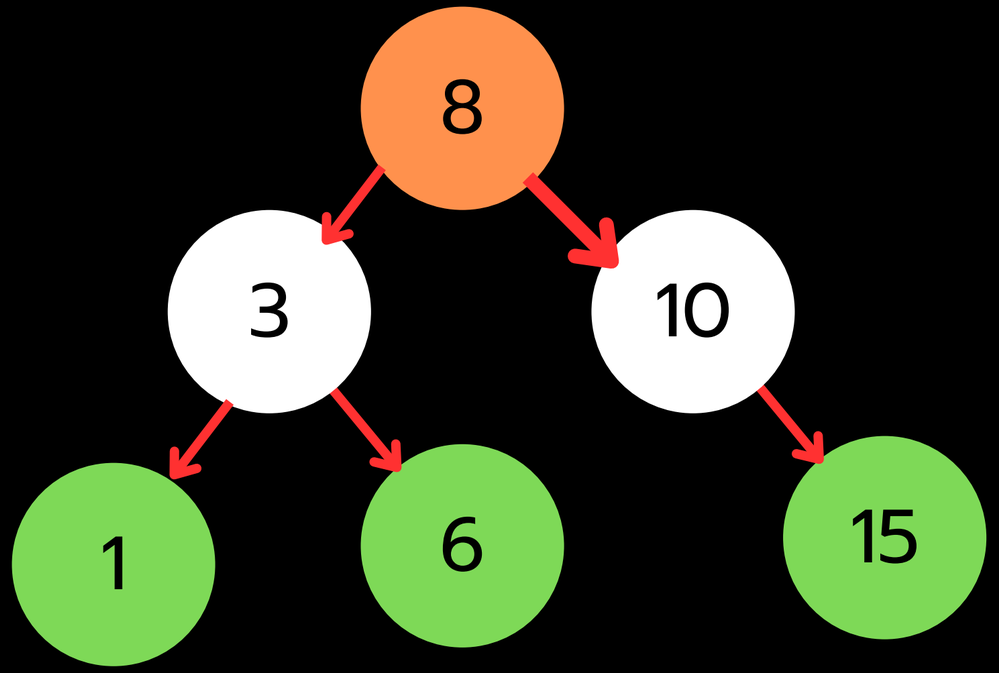
pre-order

ใช้การ recursive มาช่วยในการท่องแต่ละ child Node

```java
public void printPreOrderRecurse(Node node) {
    if (node == null)
        return;
    System.out.printf("%d ", node.data);
    printPreOrderRecurse(node.left);
    printPreOrderRecurse(node.right);
}
```

```java
bst.printPreOrderRecurse(bst.getRoot());
```

```
8 3 1 6 10 15
```

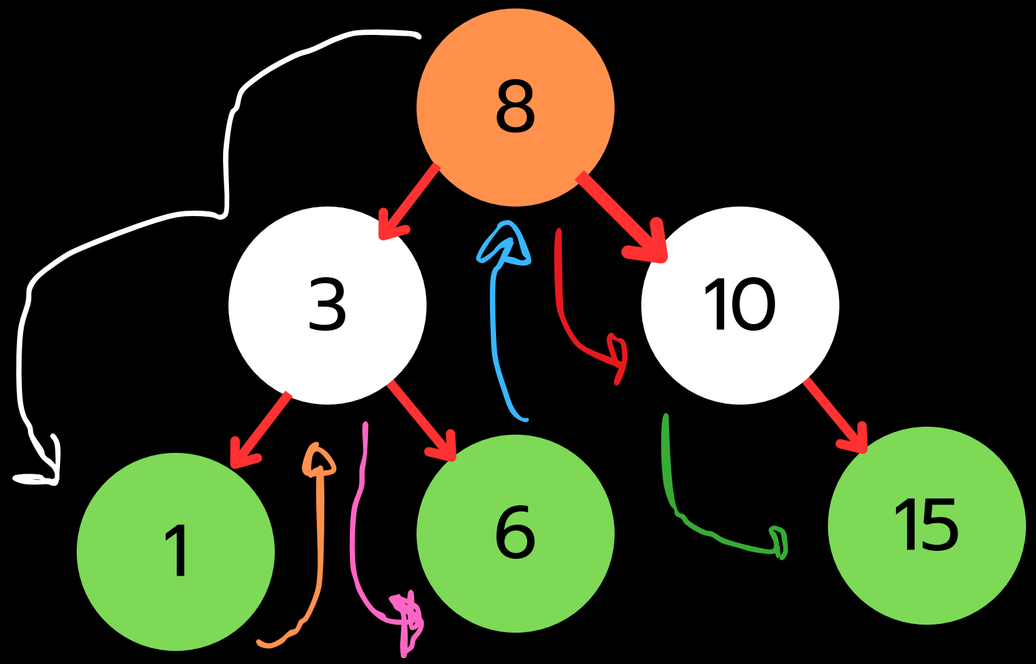



```
8
go to left child of 8
3
go to left child of 3
1
go to left child of 1
go to right child of 1
go to right child of 3
```

```
6
go to left child of 6
go to right child of 6
go to right child of 8
10
go to left child of 10
go to right child of 10
15
go to left child of 15
go to right child of 15
```

In-order

- In Order Traversal: จะเริ่มจาก Left, Root, Right



เวลาท่องจะวิ่งไปจนถึง Node สุดท้ายของซ้าย ก่อน ถึงจะแสดงค่าที่เจอออกมา แล้วค่อยไป Node ถัดไปทางขวา

1 3 6 8 10 15

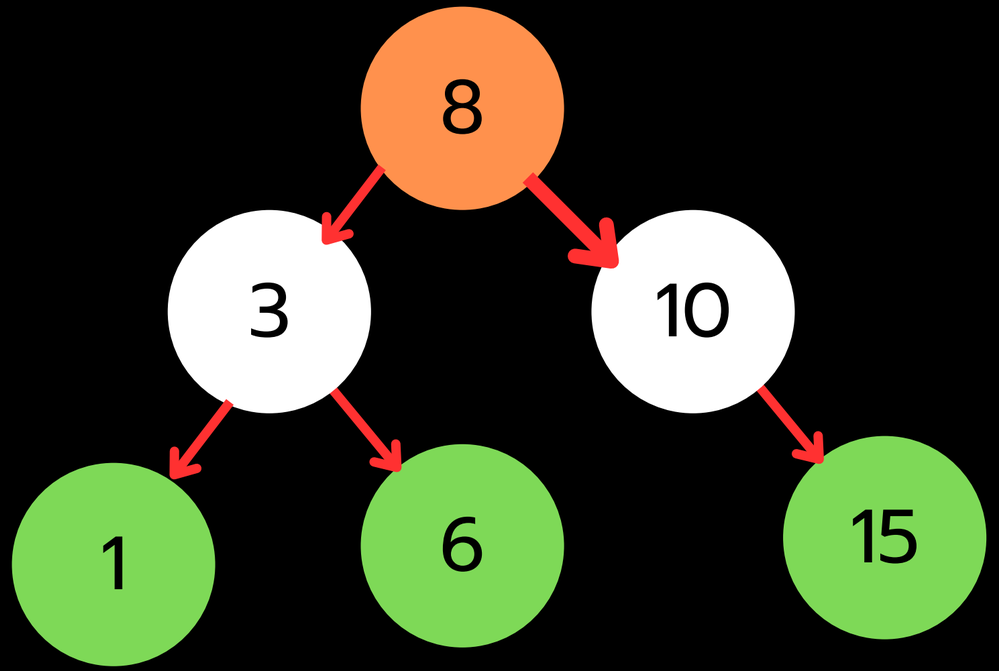
In-order

ใช้การ recursive มาช่วยในการท่องแต่ละ child Node

```java
public void printInOrderRecurse(Node node) {
    if (node == null)
        return;
    printInOrderRecurse(node.left);
    System.out.printf("%d", node.data);
    printInOrderRecurse(node.right);
}
```

```java
bst.printInOrderRecurse(bst.getRoot());
```

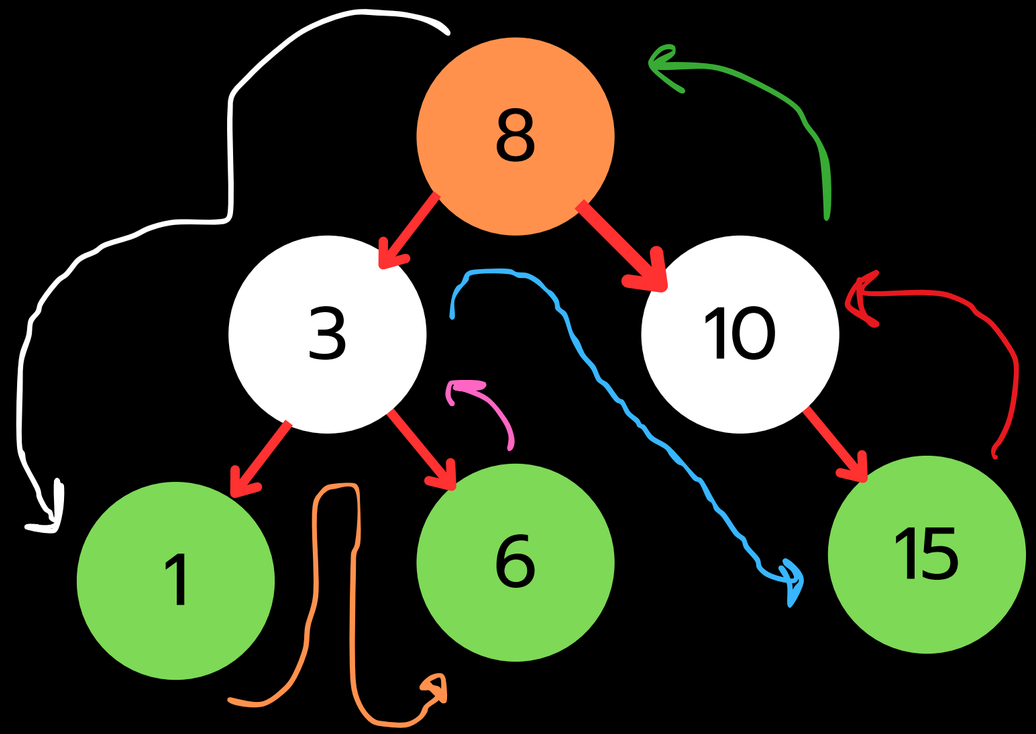
```
1 3 6 8 10 15
```




```
go to left child of 8
go to left child of 3
go to left child of 1
1
go to right child of 1
3
go to right child of 3
go to left child of 6
6
go to right child of 6
8
go to right child of 8
go to left child of 10
10
go to right child of 10
go to left child of 15
15
go to right child of 15
```

Post-order

- Post Order Traversal: จะเริ่มจาก Left, Right, Root



เวลาท่องจะไปให้ครบทุก Node ที่เป็น child Node
ก่อน ถึงจะแสดงค่าของ Node ปัจจุบัน

1 6 3 15 10 8

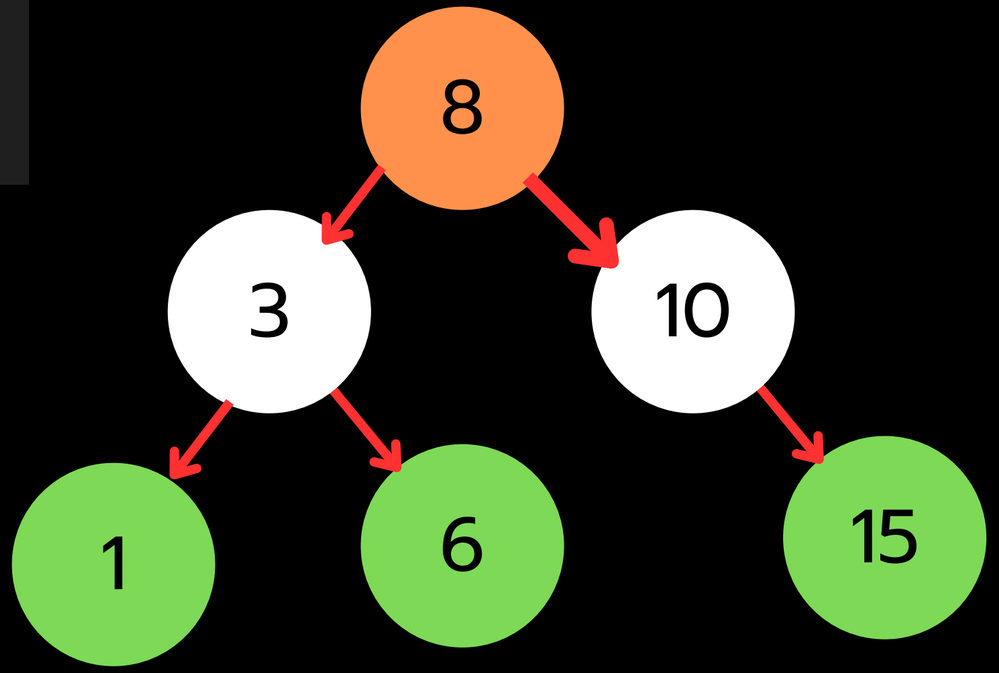
Post-order

ใช้การ recursive มาช่วยในการท่องแต่ละ child Node

```java
public void printPostOrderRecurse(Node node) {
    if (node == null)
        return;
    printPostOrderRecurse(node.left);
    printPostOrderRecurse(node.right);
    System.out.printf("%d ", node.data);
}
```

```java
bst.printPostOrderRecurse(bst.getRoot());
```

```
1 6 3 15 10 8
```

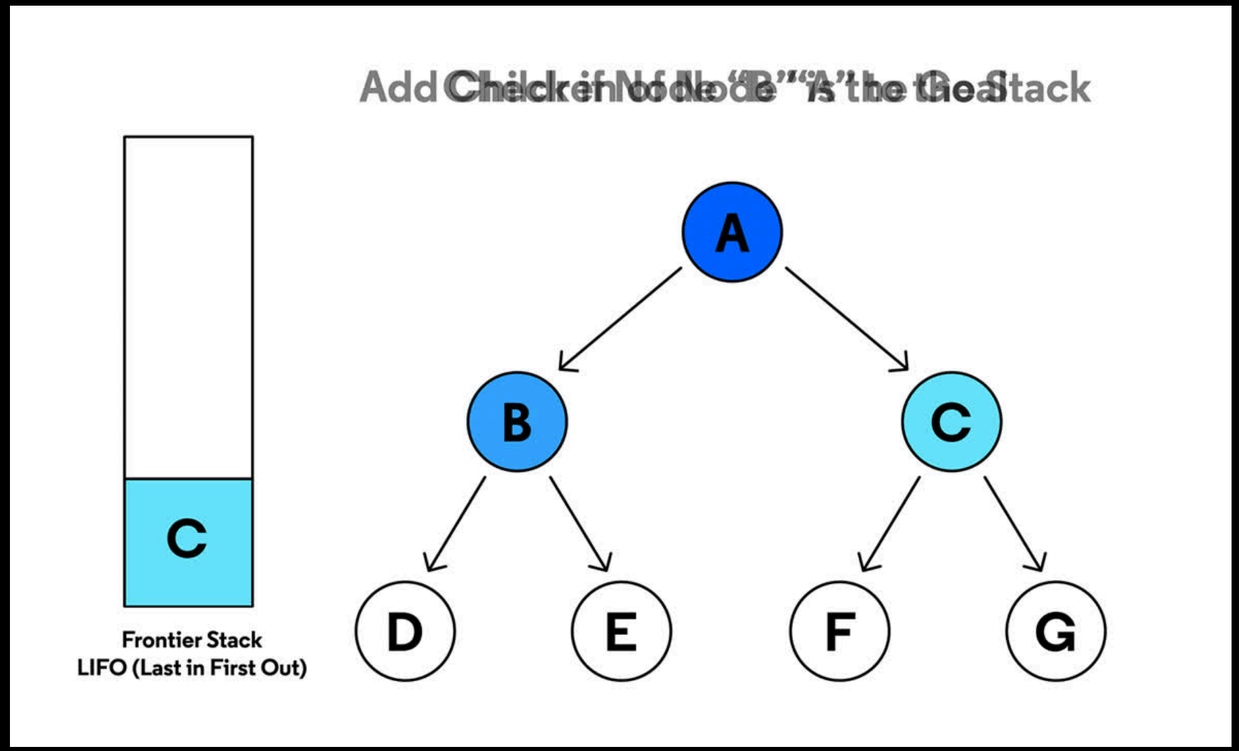



```
go to left child of 8
go to left child of 3
go to left child of 1
go to right child of 1
1
go to right child of 3
go to left child of 6
go to right child of 6
6
3
```

```
go to right child of 8
go to left child of 10
go to right child of 10
go to left child of 15
go to right child of 15
15
10
8
```

Depth First search

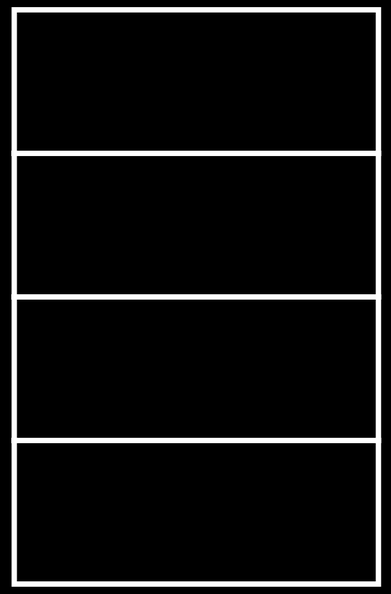
คือการค้นหาเชิงลึก การท่องจะเป็นการไล่ในแนวลึกคือ วิ่งไปที่ child Node เป็นอันดับแรก จนกว่าจะถึง Node ที่อยู่ลึกที่สุด จากนั้นไปยัง sibling Node (ถ้ามี) เมื่อครบจึงจะย้อนกลับไปยัง parent Node

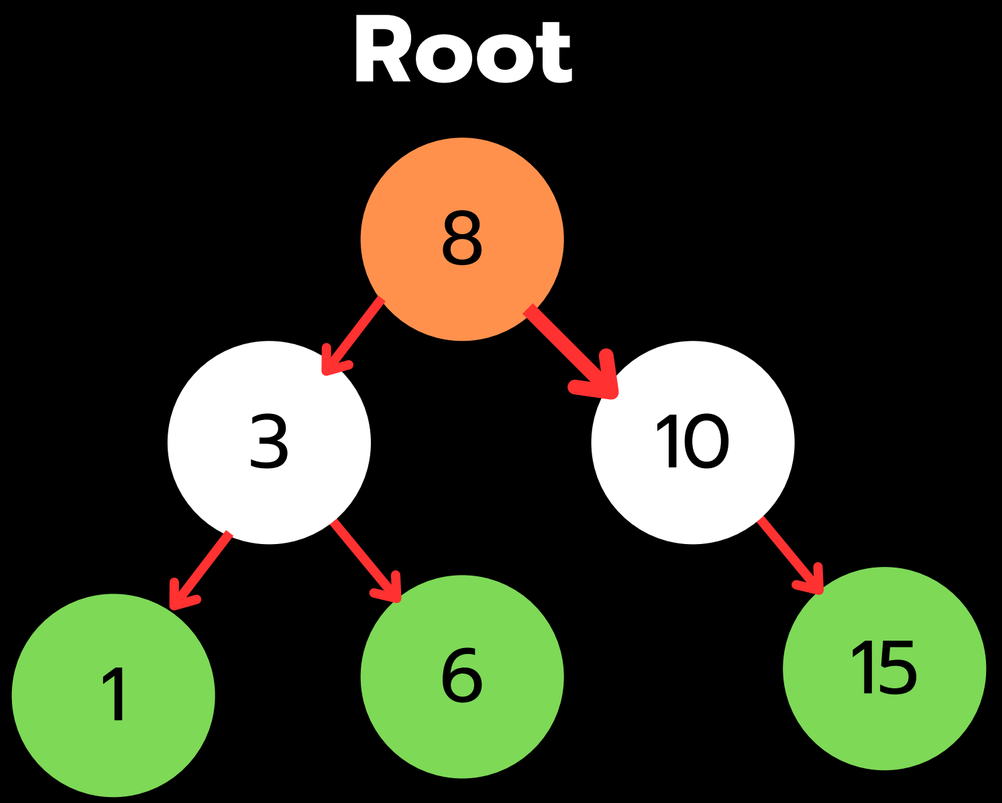


https://www.codecademy.com/article/depth-first-search-conceptual

DFS (Stack/LIFO)

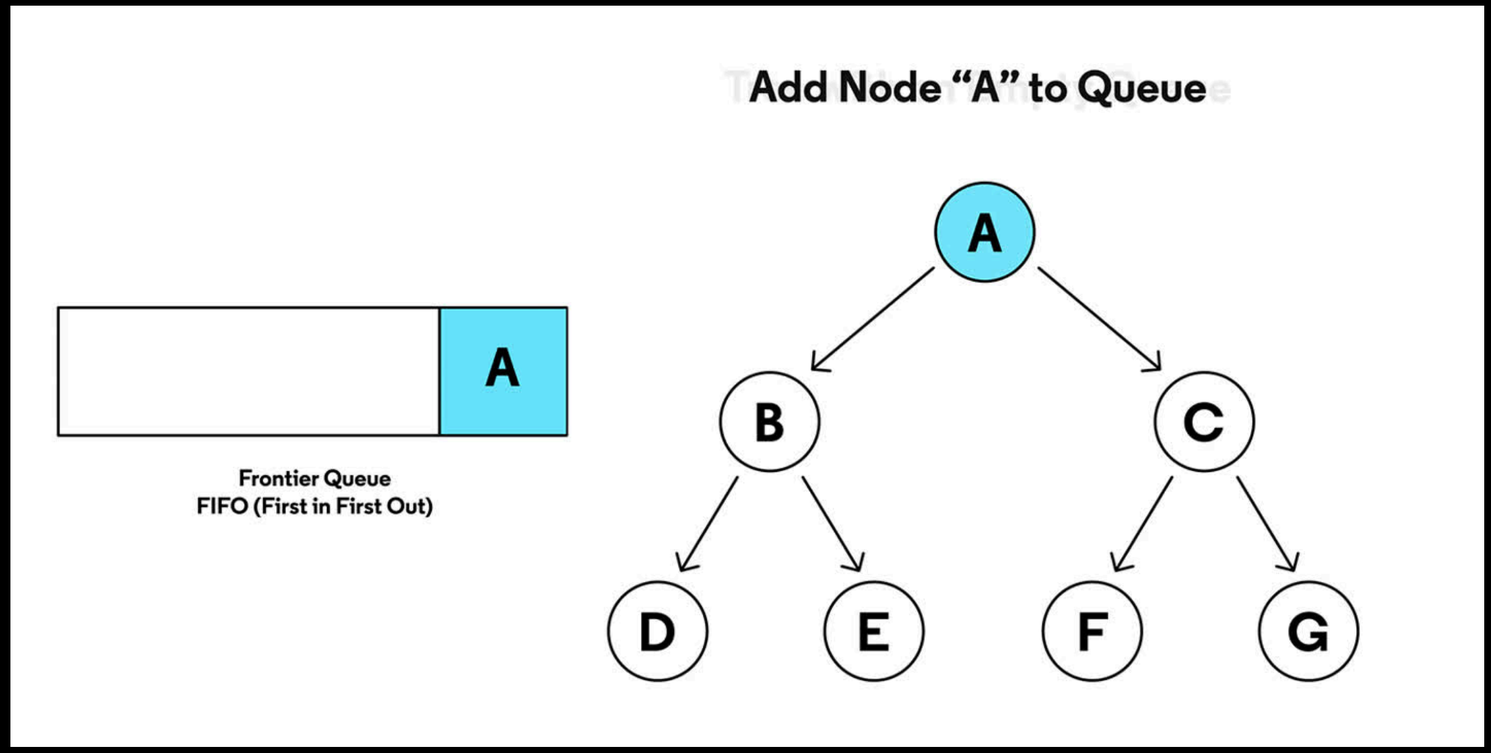
```
push root to Stack
while (Stack is not Empty) :
        pop node
        process node
        push node.left, node.right
```





Breadth First Search (BFS)

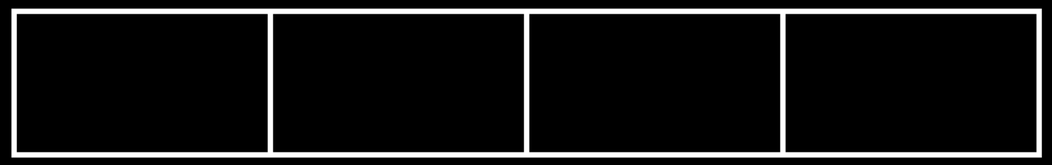
Level Order Traversal การท่องแนวกว้าง โดยจะให้ความสำคัญที่ sibling Node ก่อน child Node การไล่จะไล่ตามระดับความลึกของ tree จาก root ไปจนถึง leaf Node

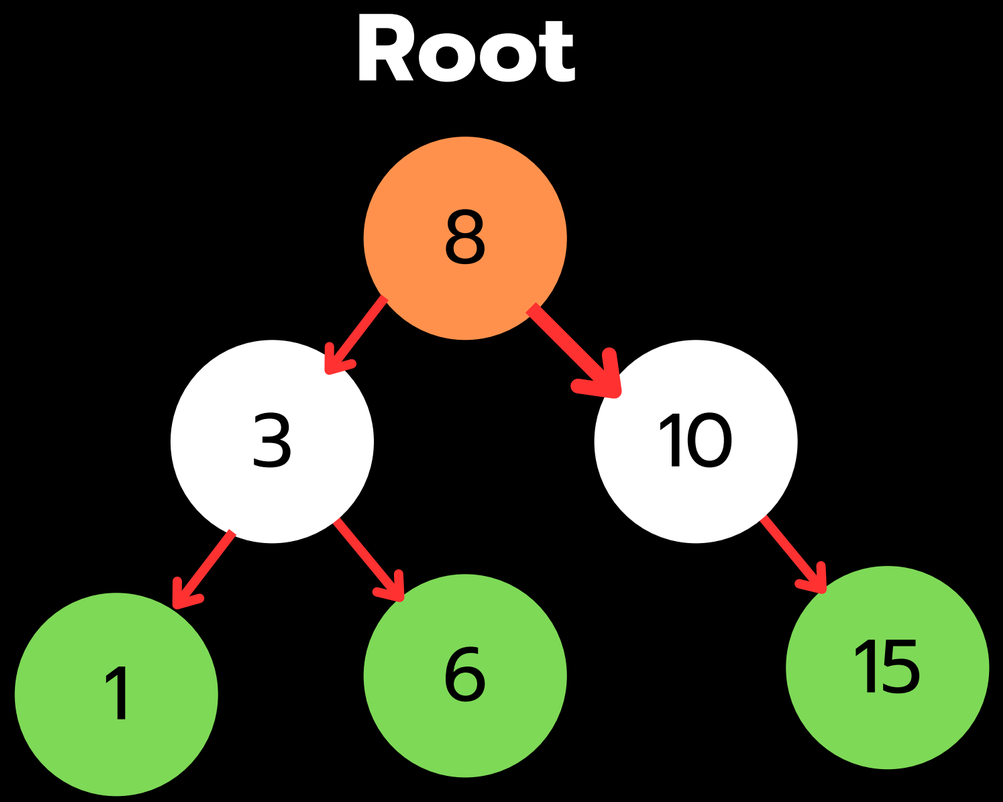


https://www.codecademy.com/article/tree-traversal

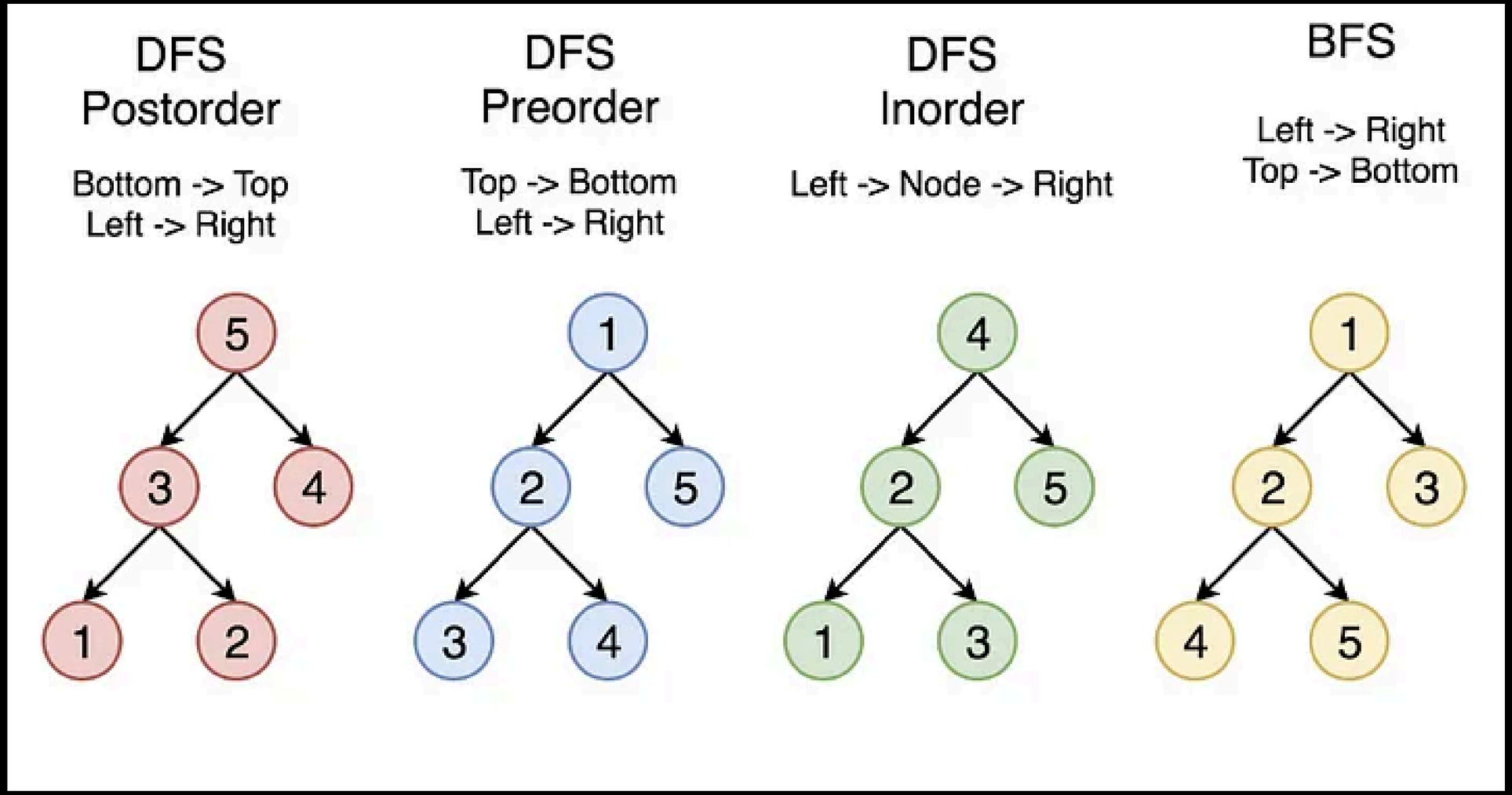
BFS (Queue/FIFO)

```
add root to Queue
while (Queue is not Empty) :
        dequeue node
        process node
        add node.left, node.right
```





Recap



https://medium.com/@Roshan-jha/a-comprehensive-guide-to-binary-tree-traversal-in-java-74c86ee23725

Thank you