

# Trees (โครงสร้างต้นไม้)

โครงสร้างข้อมูลแบบไม่เป็นเส้นตรง  
แทนความสัมพันธ์แบบลำดับชั้น (Hierarchy)

Folder / File  
System

Organization  
Chart

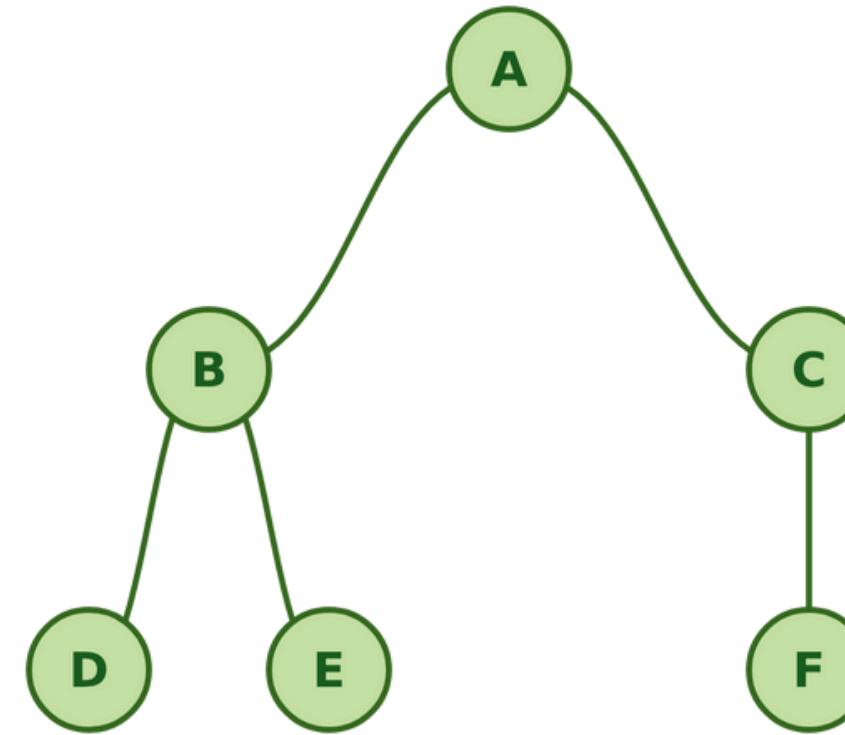
HTML  
DOM

# What is a Tree?

## Tree คืออะไร?

- โครงสร้างข้อมูลแบบไม่เป็นเส้นตรง (Non-linear)
- ใช้แทนความสัมพันธ์แบบลำดับชั้น (Hierarchy)
- พบได้ในระบบบริหาร เช่น Folder / File System, Organization Chart, HTML DOM
- จุดสำคัญ:** ไม่มี cycle และมี root เพียง 1 ตัว

## ตัวอย่าง Tree



# Tree Terminology (คำศัพท์สำหรับ)

## Root

โหนดบนสุดของต้นไม้

## Child

โหนดลูก

## Leaf

โหนดที่ไม่มีลูก

## Parent

โหนดแม่

## Sibling

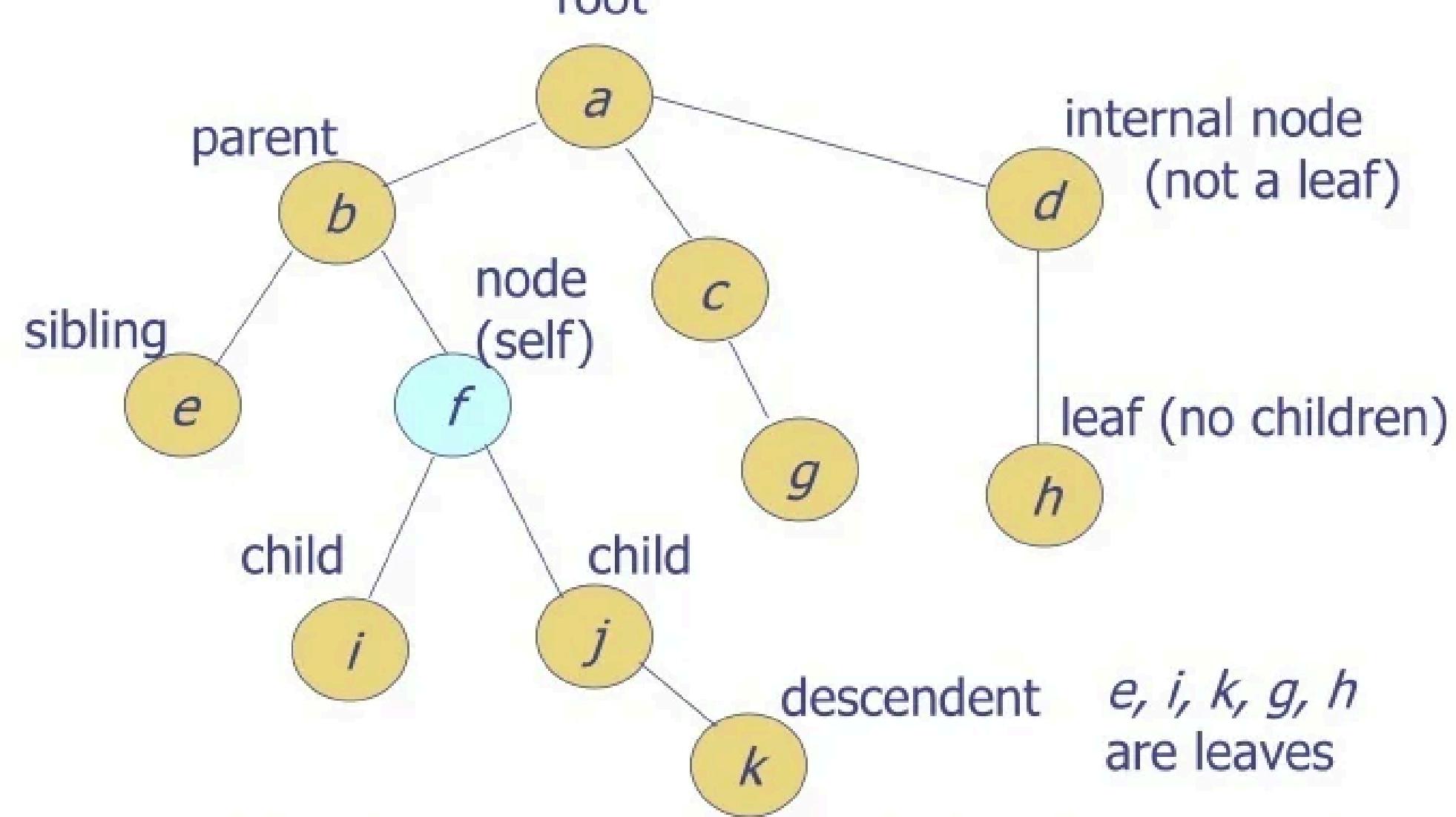
โหนดที่มี parent เดียวกัน

## Subtree

ต้นไม้ย่อย

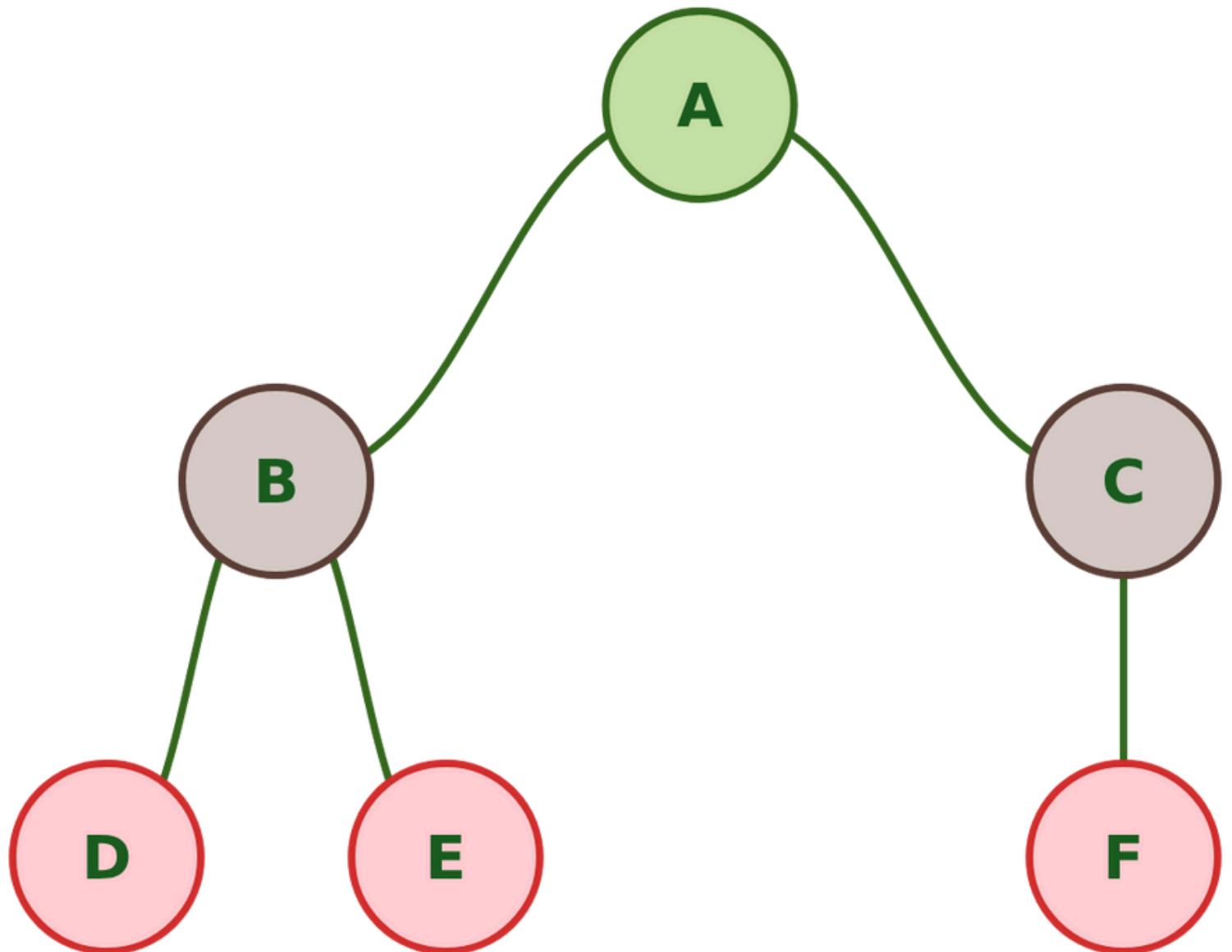
## ตัวอย่าง Tree และ Terminology

### Rooted Tree



a few terms: parent, child, decendent, ancestor, sibling, subtree, path, degree,

# Tree Diagram (อย่างง่าย)



องค์ประกอบของต้นไม้

- Root (โหนดบนสุด)**  
A
- Parent (โหนดแม่)**  
B คือ Parent ของ D และ E
- Leaf (โหนดที่ไม่มีลูก)**  
D, E, F

สัญลักษณ์:

- Root
- Parent
- Leaf

# Depth และ Height

## คำจำกัดความ



### Depth

ระยะจาก Root → Node

ระดับของโหนดจาก root (Root อยู่ที่ระดับ 0)



### Height

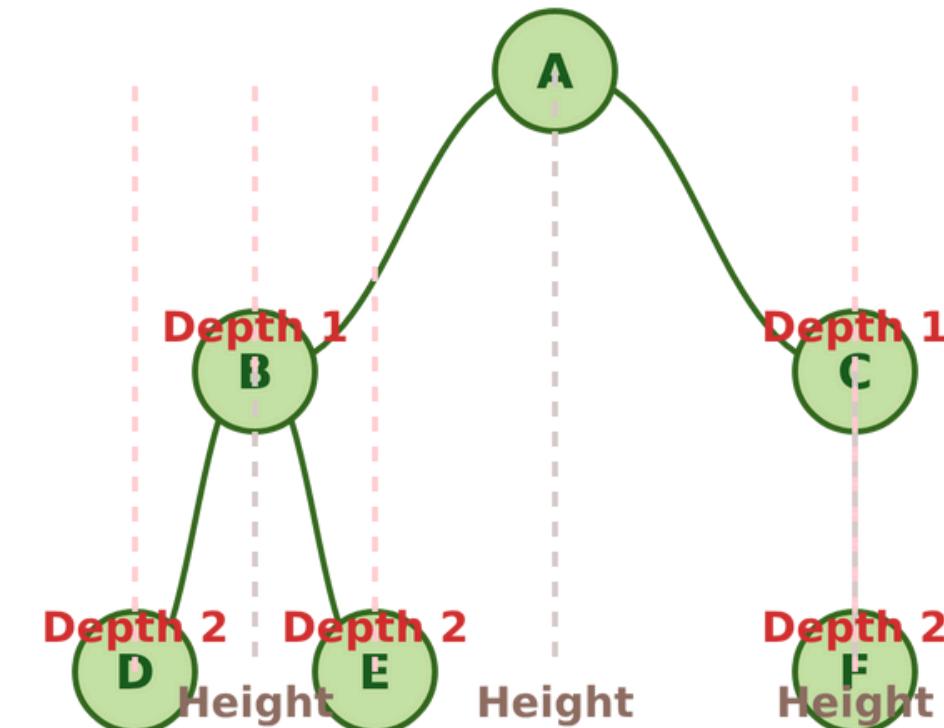
ระยะจาก Node → Leaf ที่ใกล้ที่สุด

ความสูงของต้นไม้ = Height ของ Root

❗ ตัวอย่าง: ในต้นไม้ที่ shown อยู่ทางขวา

- Depth ของ A = 0
- Depth ของ B และ C = 1
- Depth ของ D = 2

## ตัวอย่าง Tree



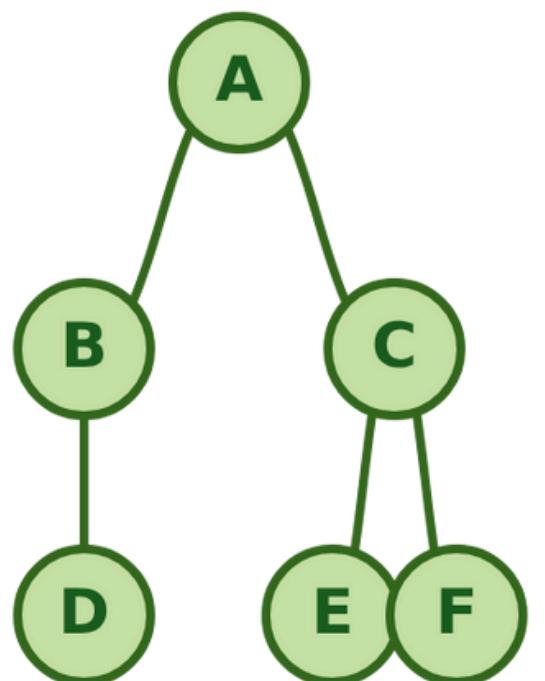
เส้นบวก Depth

เส้นบวก Height

# Types of Trees

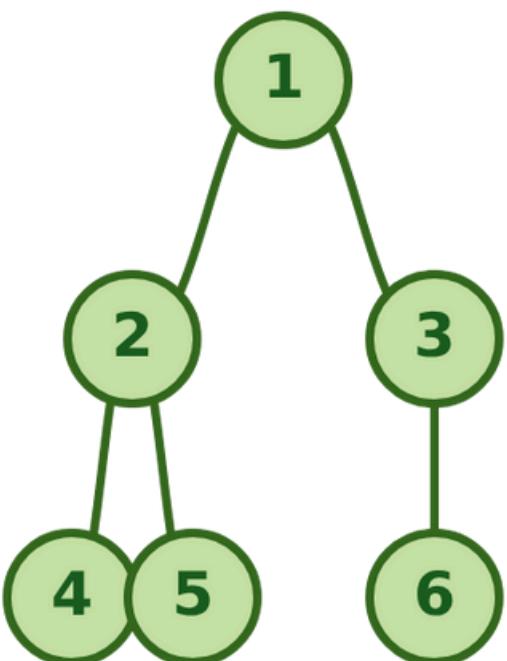
## General Tree

โครงสร้างต้นไม้กึ่งไป กี่มีโหนด  
หลายตัวที่เชื่อมโยงกัน



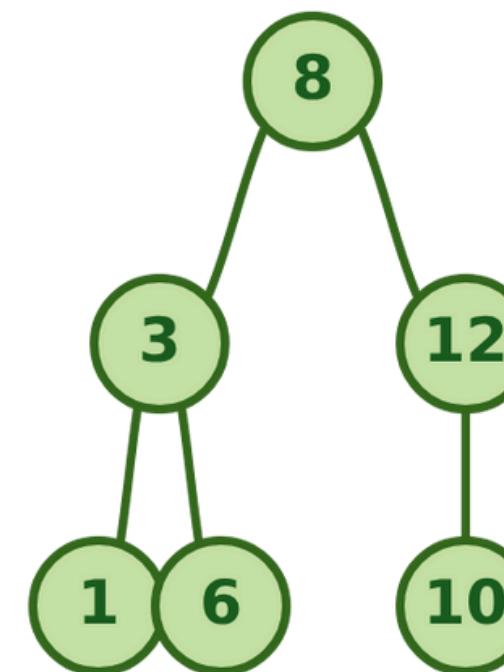
## Binary Tree

แต่ละ node มีลูกได้ไม่เกิน 2 คบ  
แบ่งเป็น Left / Right



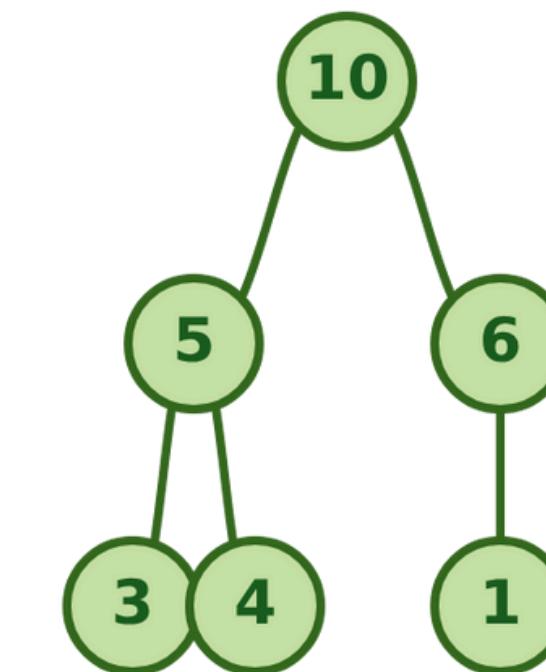
## Binary Search Tree

Binary Tree กี่มีคุณสมบัติ  
พิเศษในการจัดเก็บข้อมูล



## Heap Tree

โครงสร้างต้นไม้กี่มีคุณสมบัติว่า  
โหนดแม่จะมีค่ามากกว่าหรือน้อย  
กว่าโหนดลูก



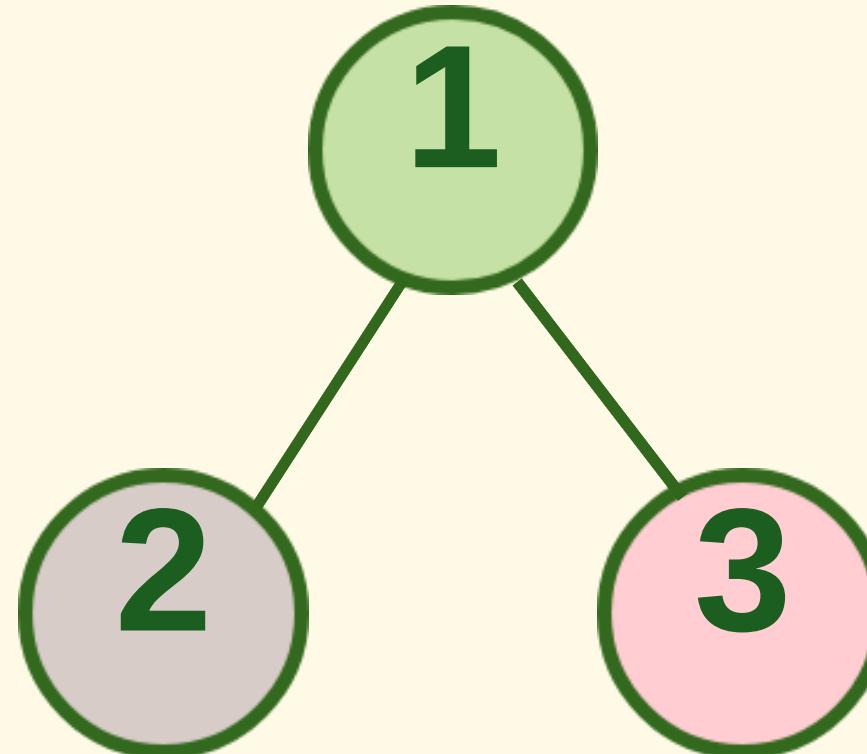
เน้น Binary Tree

# Binary Tree คืออะไร?

## Binary Tree

- ต้นไม้ที่แต่ละ node มีลูกได้ไม่เกิน 2 คน
- แบ่งเป็น Left (ซ้าย) และ Right (ขวา)

## ตัวอย่าง Binary Tree



## Binary Tree Properties



### Left Child

โหนดลูกซ้ายของโหนดแม่



### Maximum Degree

แต่ละ node มีลูกได้ไม่เกิน 2 คน



### Right Child

โหนดลูกขวาของโหนดแม่



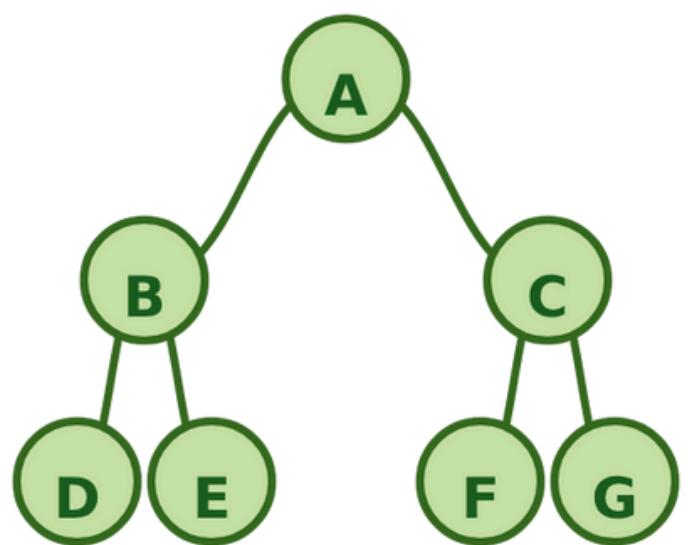
### Binary Tree

โครงสร้างข้อมูลแบบไม่เป็นเส้นตรง

# Types of Binary Tree

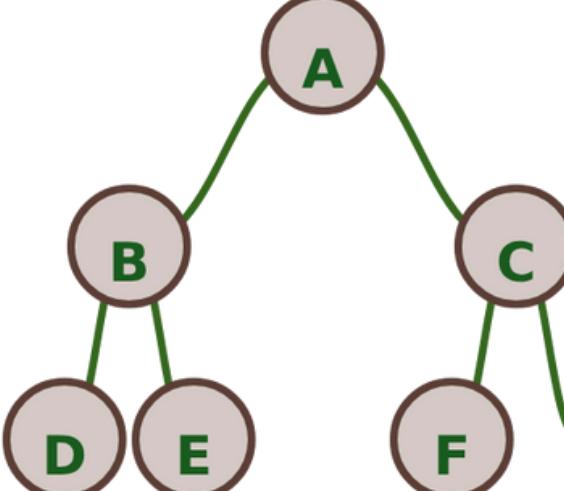
## 1. Full Binary Tree

ทุก node มี 0 หรือ 2 ลูก



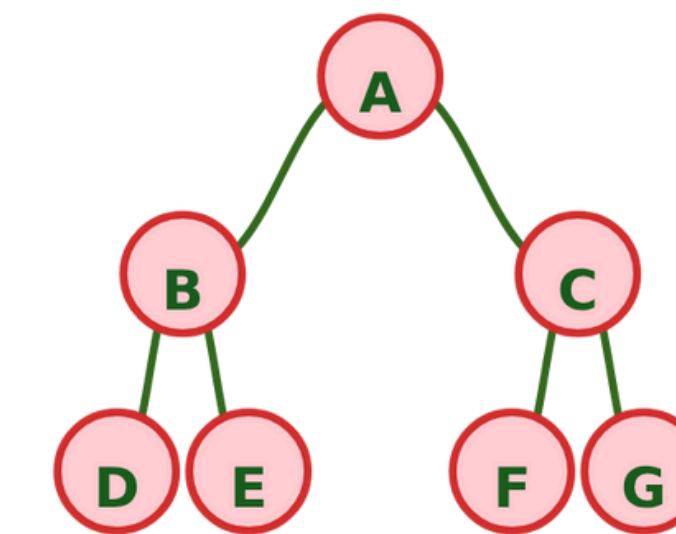
## 2. Complete Binary Tree

เต็มจากซ้ายไปขวา



## 3. Perfect Binary Tree

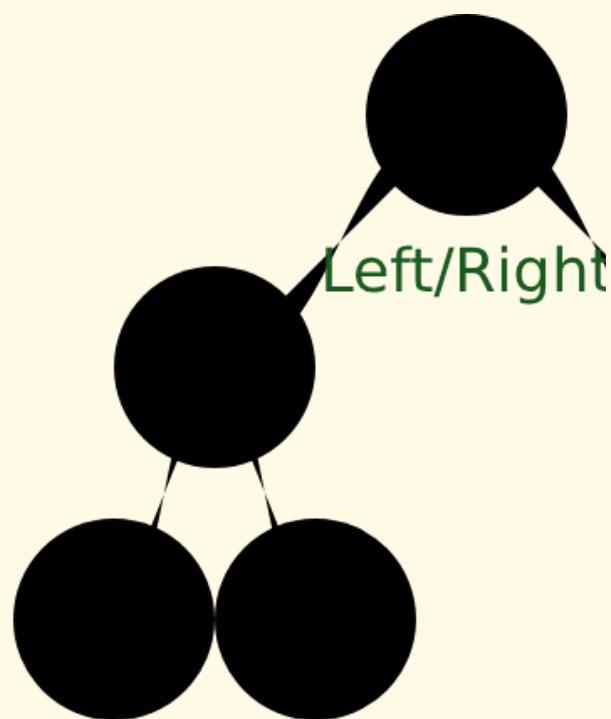
ทุกระดับเต็ม



# Binary Tree Representations

## Node-based Representation

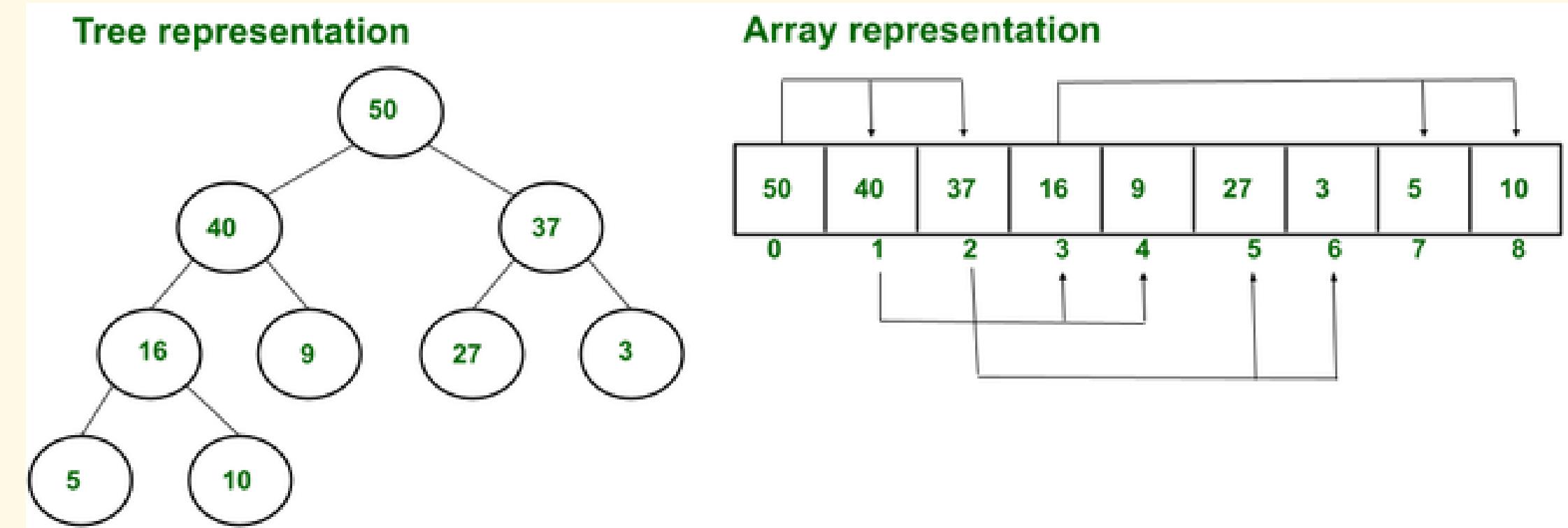
```
struct Node {  
    int data;  
    Node* left;  
    Node* right;  
};
```



## Array (Heap) Representation

A	B	C	D	E	F
i=0	i=1	i=2	i=3	i=4	i=5

- Parent:  $i$
- Left:  $2i + 1$
- Right:  $2i + 2$



# Tree Traversal คืออะไร?

## Traversal คืออะไร?

🚶 Traversal = การเดินดูทุก node ในต้นไม้

💡 วิธีการเยี่ยมชมโหนดทุกตัวในต้นไม้

❗ อักสອบແນ່ນອນ

## 3 วิธีหลักในการ Traversal

### Preorder

Root → Left → Right

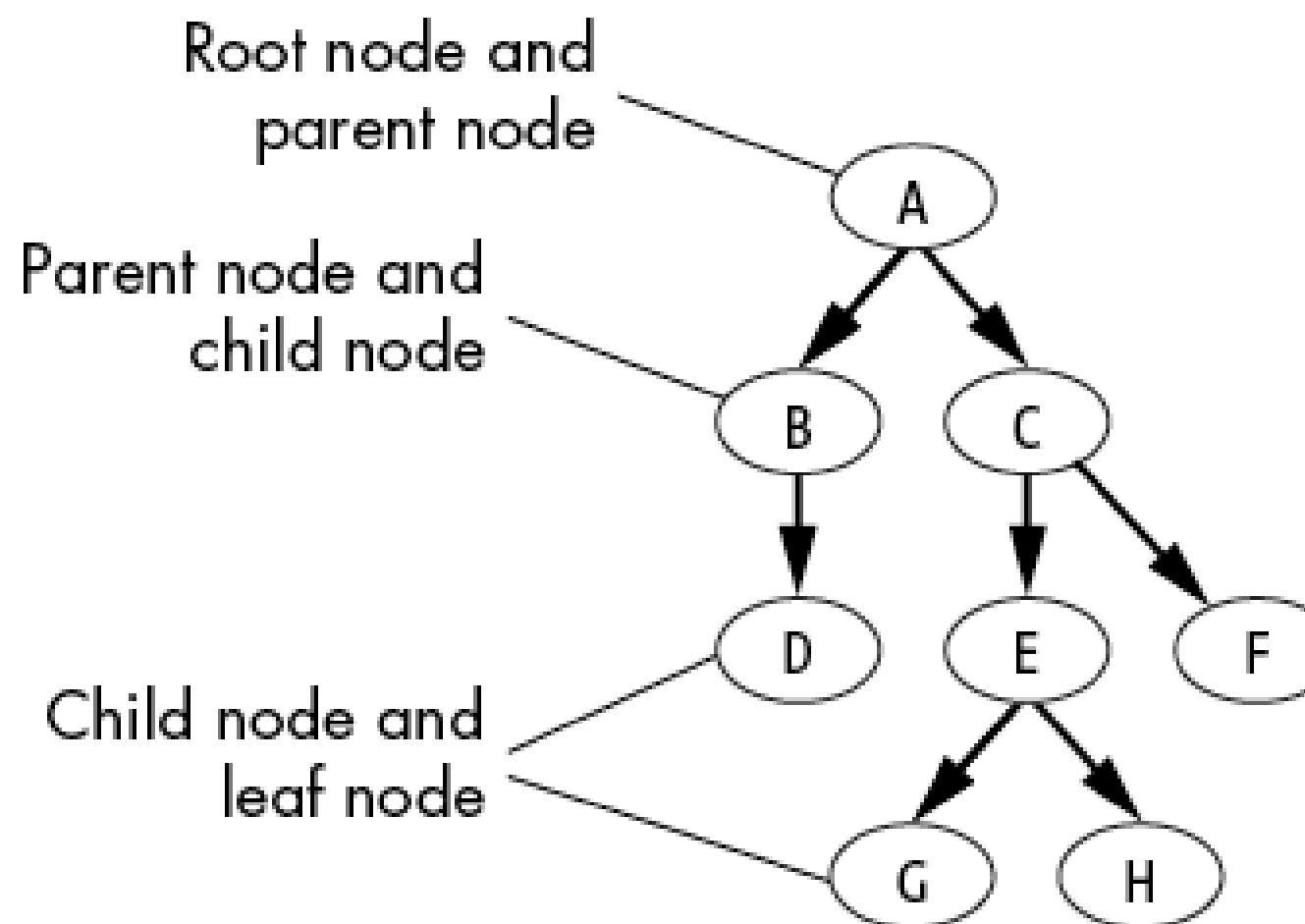
### Inorder

Left → Root → Right

### Postorder

Left → Right → Root

## ตัวอย่าง Traversal



Preorder:

A B D C E G H F

Postorder:

D B G H E F C A

Inorder:

D B A G E H C F

# Recursive Traversal (Concept)

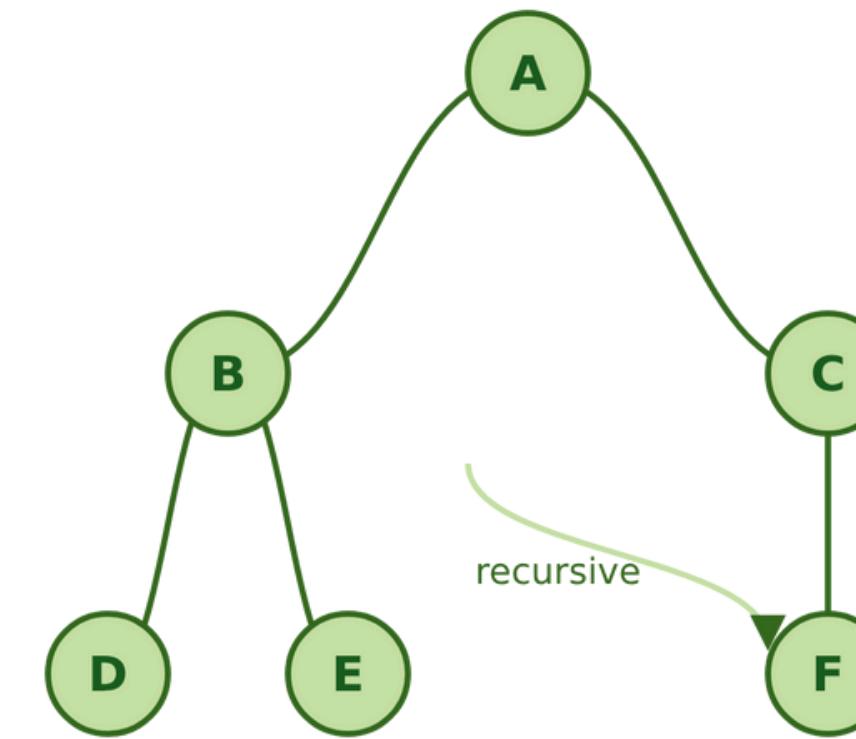
## แนวคิด Traversal และ Recursive

- ឧ Traversal ใช้ recursion ได้ดีที่สุด
- គ กำช้ำໂຄຮງສ້າງເດີມກັບ subtree
- !**Base case: node == null**

## ตัวอย่าง Code

```
// Preorder Traversal ด้วย recursion
void preorder(Node node) {
    if (node == null) // Base case
        return;
    print(node.data);
    preorder(node.left);
    preorder(node.right);
}
```

## การเรียก recursive



💡 Recursive ช่วยให้เราเขียน code ง่ายขึ้นและเข้าใจง่ายกว่า iterative approach

# Tree Time Complexity

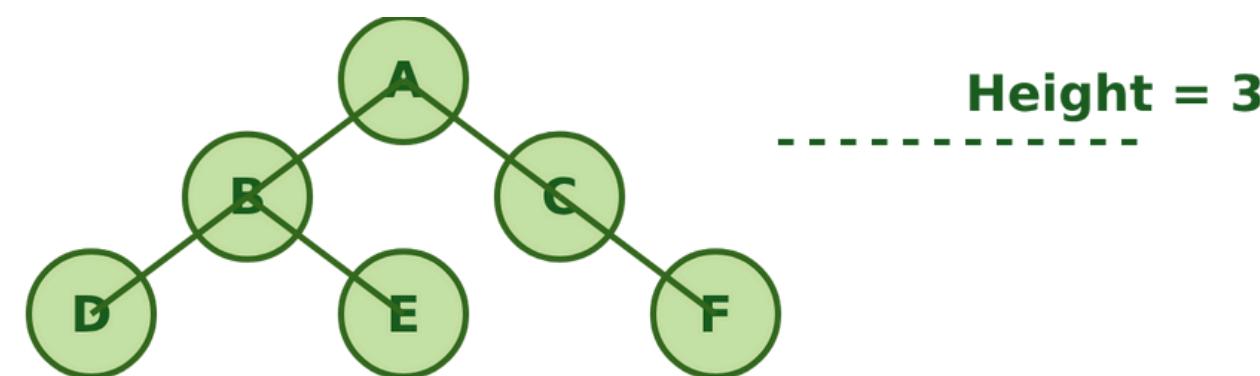
## ⌚ Time Complexity

- ✓ Traversal ทุกแบบ: **O(n)**
- ✓ ทุก node จะถูกเข้าถึงเพียงครั้งเดียว
- ✓ ไม่ขึ้นกับโครงสร้างของต้นไม้

## 💾 Space Complexity

- ✓ Space (recursion): **O(h)**
- ✓  $h$  = ความสูงของต้นไม้
- ✓ ขึ้นกับความสูงของต้นไม้

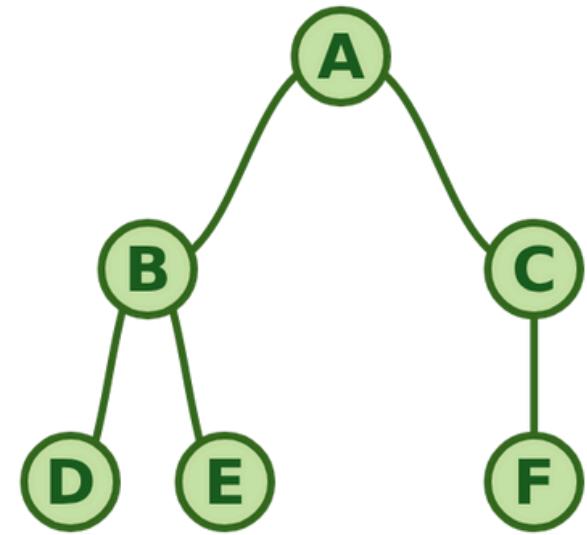
## ความสูงของต้นไม้ (Height)



# Tree vs Linear Structure

โครงสร้างข้อมูล

Tree



Linear



→ → →  
≡ เส้นตรง (Linear)

└ ลำดับชั้น (Hierarchy)

## Comparison Table

Structure

Array

Linked List

Tree

ลักษณะ

เป็นเส้นตรง

เป็นเส้นตรง

เป็นลำดับชั้น

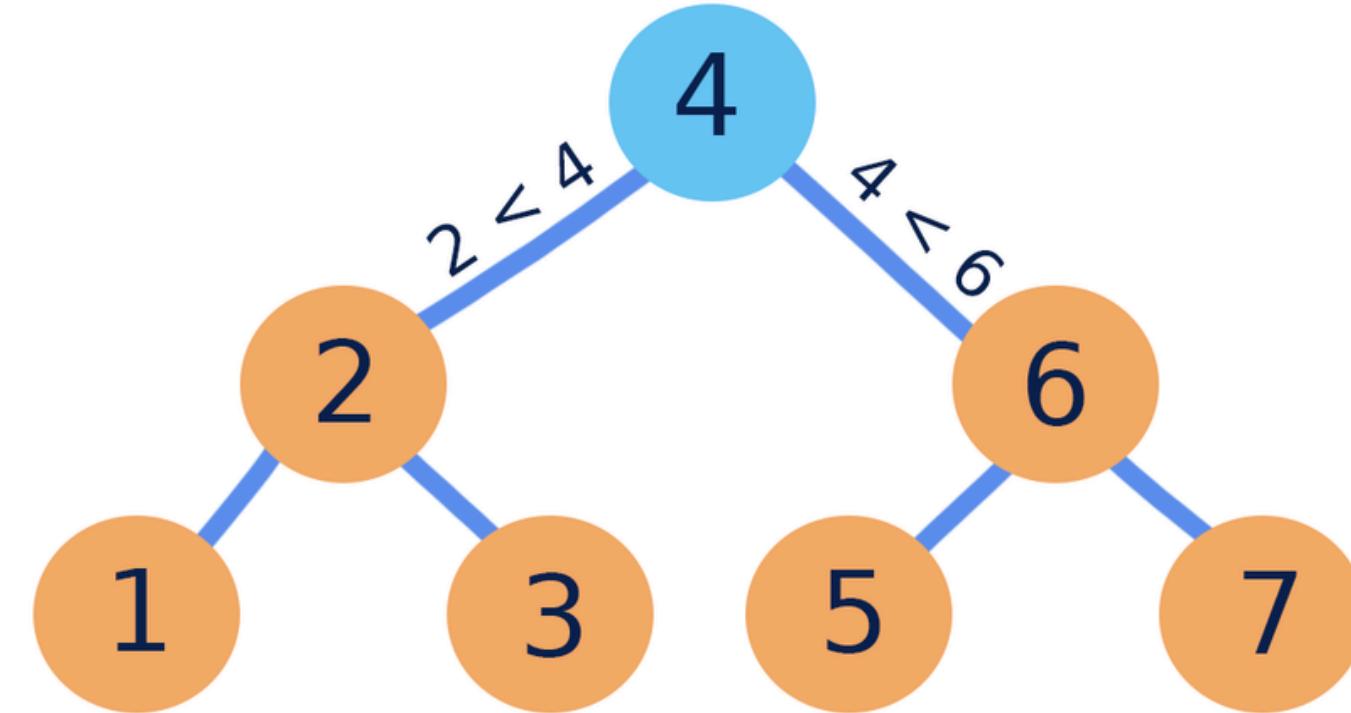
# อะไรคือ Binary Search Tree?

## 💡 Binary Search Tree (BST)

คือ Binary Tree ที่มีการจัดเรียงข้อมูลตามกฎพิเศษ

### ✓ กฎของ BST:

- ค่าในฝั่งซ้าย  $<$  Node ปัจจุบัน
- ค่าในฝั่งขวา  $>$  Node ปัจจุบัน
- ทุก subtree ต้องเป็น BST เช่นกัน



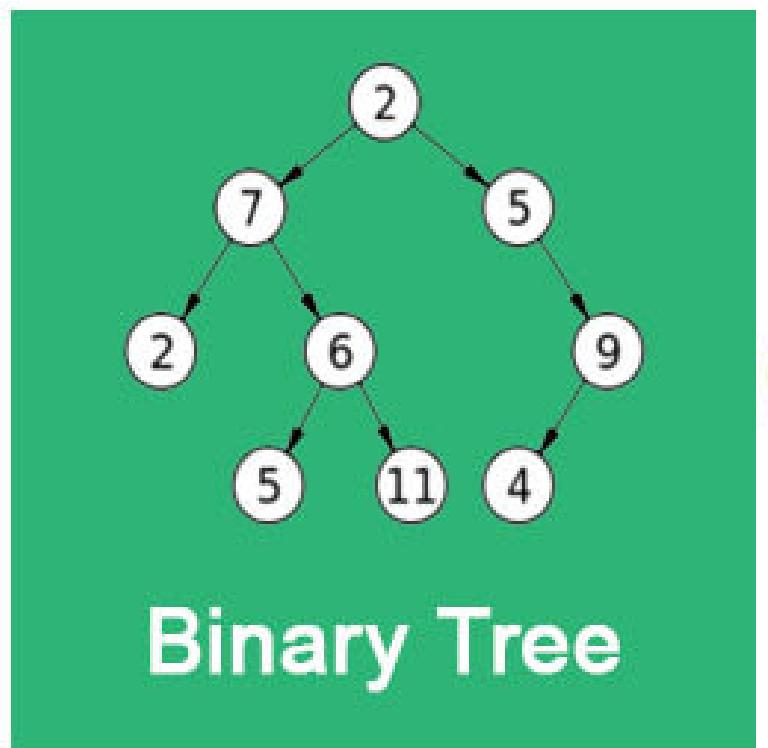
In Order Traversal: 1 2 3 4 5 6 7



# BST ทำไมจึงสำคัญ?

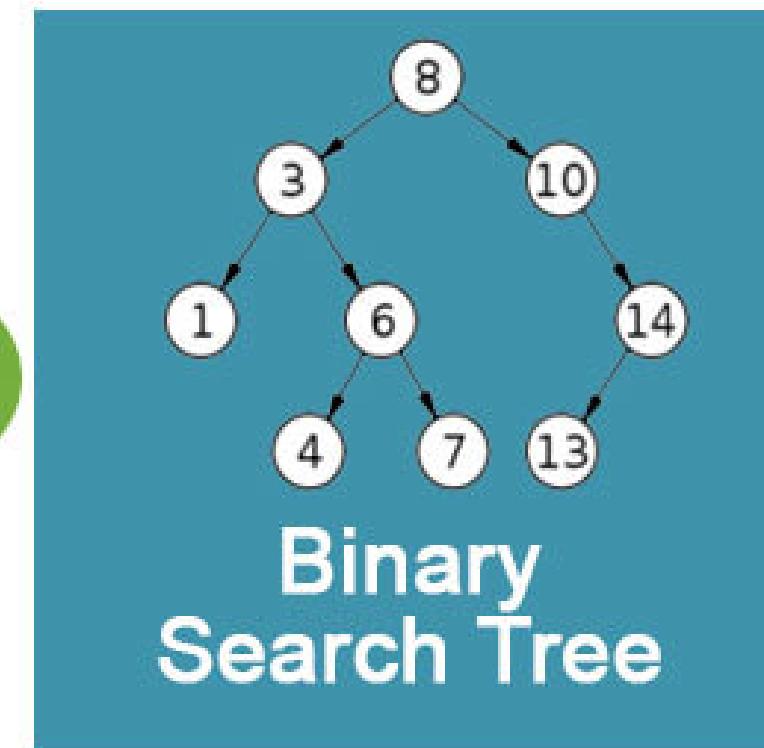
## ⚖️ การเปรียบเทียบ

Binary Tree กว้างไป



🔍 Search:  $O(n)$

Binary Search Tree



🔍 Search:  $O(\log n)$

## ⭐ ข้อดีของ BST

- ✓ Search เเร็วกว่า Binary Tree กว้างไป
- ✓ ใช้หลักเดียวกับ Binary Search
- ✓ เป็นพื้นฐานของ Tree ขั้นสูง (AVL, Red-Black)

### 💡 สมดุลย์มีผลมาก

สมดุล

Search:  $O(\log n)$

ไม่สมดุล



Search:  $O(n)$

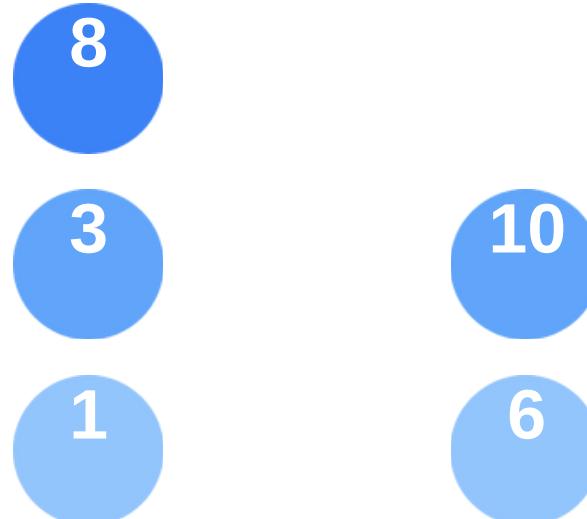
# BST Insert และ Delete Cases

## + Insert

### ✓ หลักการ

- คั่บหาตำแหน่งว่างตามกฎ BST
- เพิ่ม node ใหม่เป็น leaf

❗ โดยทั่วไป \*\*ไม่อนุญาตค่าซ้ำ\*\*



## 刪 Delete Cases

การลบใน BST มี 3 กรณีสำคัญ

1 Node ไม่มีลูก  
ลับได้กันที

2 Node มีลูก 1  
ดึงลูกขึ้นมาแทน

3 Node มีลูก 2  
ใช้ Inorder Successor

⚠️ สำคัญ

# กรณีที่ au Node ที่มีลูก 2 คน

## ⚠ กรณีที่ยากรากที่สุด

การลบ Node ที่มีลูก 2 คน ต้องใช้ Strategy พิเศษ

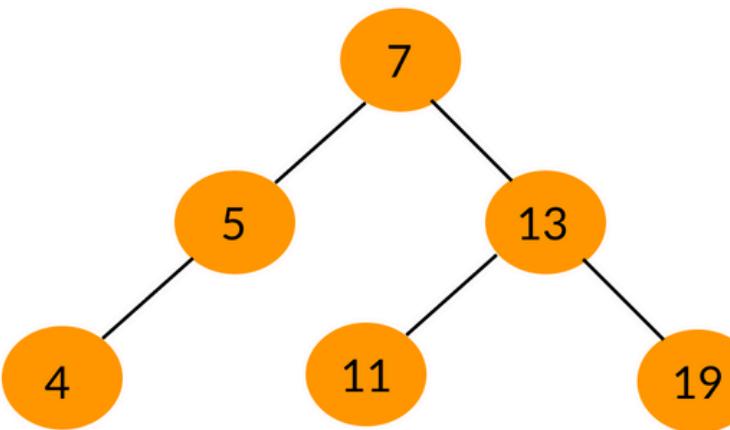
### ขั้นตอนหลัก:

- หา **Inorder Successor** (ค่าบ้านอยู่ที่สุดใน Right Subtree)
- แทนค่าที่ต้องลบด้วย Successor
- ลบ Successor ซ้ำอีกครั้ง (จะกลับเป็น Case 1 หรือ Case 2)

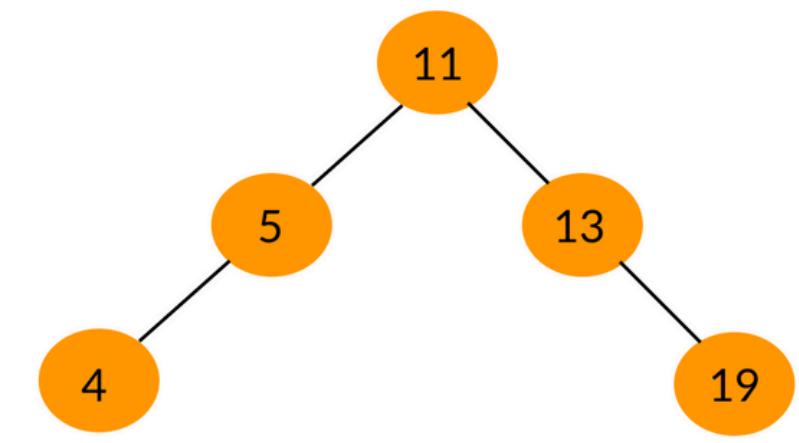
**Inorder Successor** = โหนดถัดไปในการเดินแบบ inorder

## 🌲 ตัวอย่าง: ลบ Node 7

Before deleting 7



After deleting 7



Inorder : 4 5 7 11 13 19

Inorder : 4 5 11 13 19

# สรุป



## Tree = โครงสร้างไม่เป็นเส้นตรง

ใช้แทนความสัมพันธ์แบบลำดับชั้น (Hierarchy) พบได้ในระบบบจจุร  
เช่น Folder / File System, Organization Chart, HTML DOM



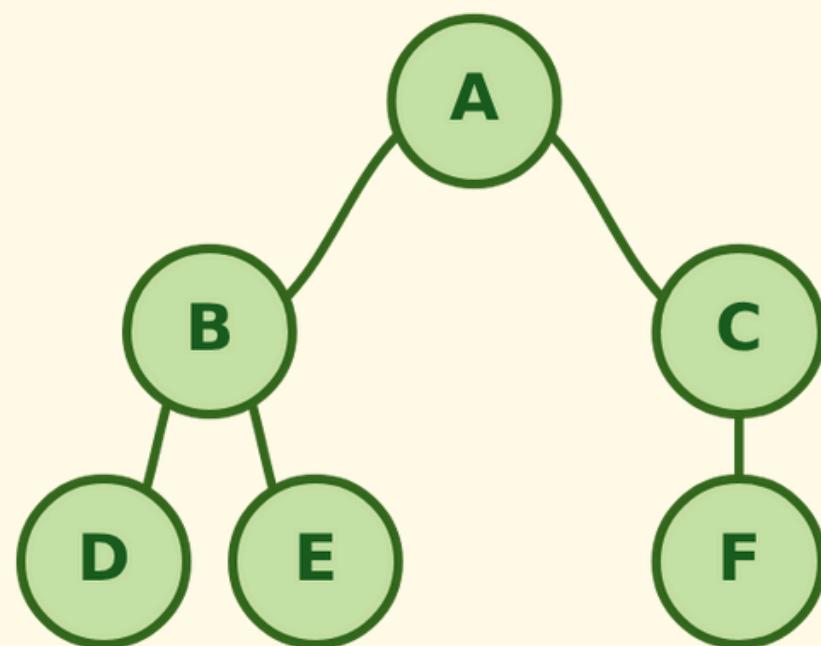
## Binary Tree มีลูกไม่เกิน 2

แต่ละ node มีลูกได้ไม่เกิน 2 คน แบ่งเป็น Left / Right



## Traversal สำคัญมาก

3 วิธีหลัก: Preorder, Inorder, Postorder อักษรสับเปลี่ยน



## Binary Search Tree

- 🔍 **Search** - การค้นหาค่าในต้นไม้
- ➕ **Insert** - การเพิ่มค่าลงในต้นไม้
- ➖ **Delete** - การลบค่าออกจากต้นไม้
- ⌚ **Time Complexity** - การวิเคราะห์ประสิทธิภาพ

### Preorder

Root → Left → Right

### Inorder

Left → Root → Right

### Postorder

Left → Right → Root

# Practice (ฝึก)

## การฝึกฝนที่แนะนำ



### วาด Binary Tree เอง

สร้างต้นไม้ไบนารีตัวเองเพื่อเข้าใจโครงสร้าง



### หา Pre/In/Postorder

ฝึกหาลำดับการ traversal ของต้นไม้ที่กำหนด



### เขียน traversal แบบ recursion

ฝึกเขียนโค้ด recursive สำหรับแต่ละแบบ traversal

## ตัวอย่างการฝึก

### Preorder

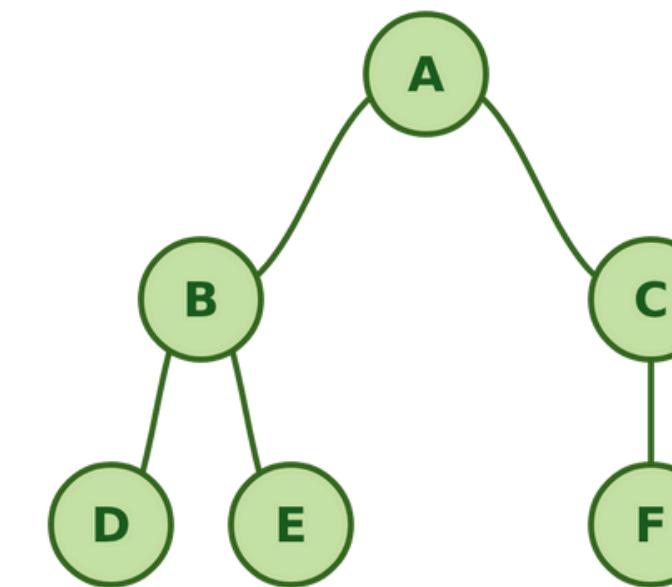
A → B → D →  
E → C → F

### Inorder

D → B → E →  
A → C → F

### Postorder

D → E → B →  
F → C → A





## แบบฝึกหัด

### รายละเอียดโปรแกรม

- 📄 สร้างโปรแกรมใช้ BST (Binary Search Tree)
- 🕒 ทำ CRUD operations

### สิ่งที่ต้องเตรียม

- UrlParser
- github และการ deployment