Tree I

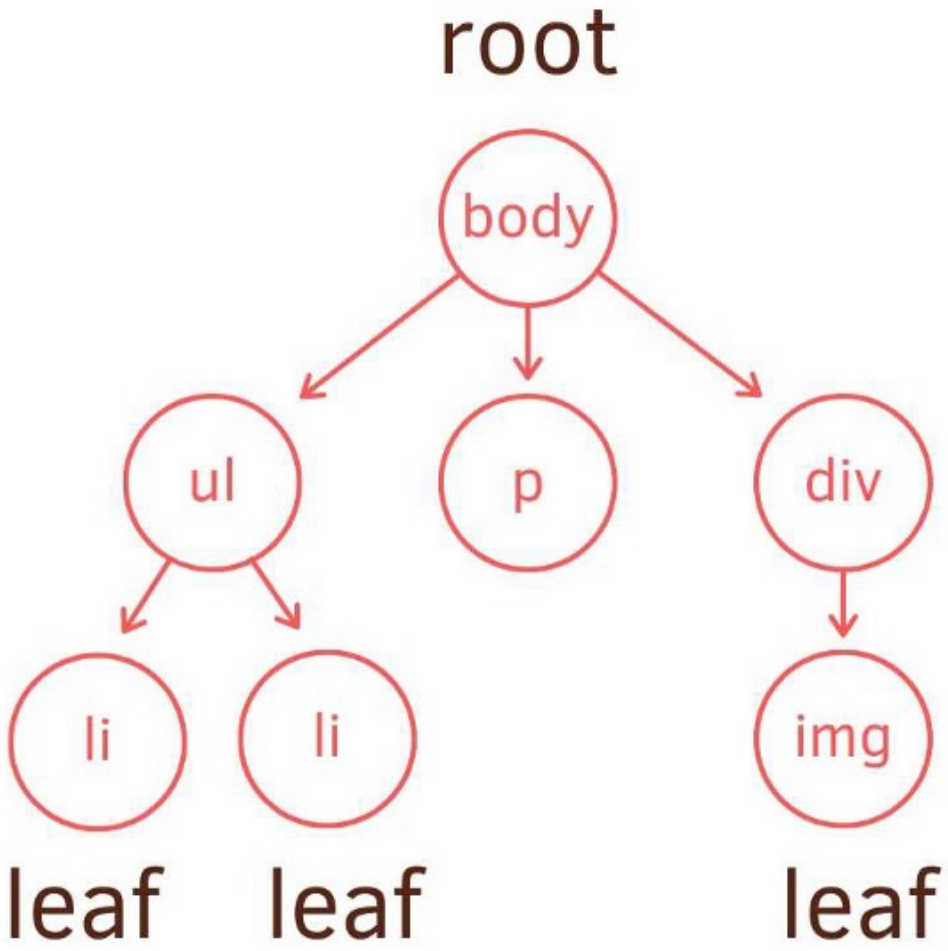
樹

Mozix Chien

Outline

- Tree
- Binary Tree
- Binary Tree Traversal
- Binary Search Tree
- Implementation

Tree



Tree

介紹

- 樹是一種 **階層式(Hierarchical)** 的 **資料結構**
- 樹的組成包含 **枝(邊)** 與 **節點**
- 樹中不可包含 **環(Ring)**
- 每個 **子節點只能有一個父節點**
- 依照有無根節點可分為 **有根樹** 與 **無根樹**
- 依照子節點個數可分為 **二元樹** 或 **K元樹**

Tree

名詞介紹

- 節點 (node) : 樹上的每個分支點
- 枝(邊) (branch) : 連接節點與節點間的邊
- 根節點 (root) : 一棵樹可想做是由一個點(根)開始分支，一棵樹上的每一個點都可以作為根。
- 葉節點 (leaf) : 無法繼續分支的節點、沒有子節點的節點
- 父節點 (parent) : 相臨的兩個點，較靠近根的為父節點
- 子節點 (child) : 相臨的兩個點，較遠離根的為子節點
- 祖先 (ancestor) : 父節點的父節點...，為點的祖先
- 子孫 (descendant) : 子節點的子節點...，為點的子孫

Tree

名詞介紹

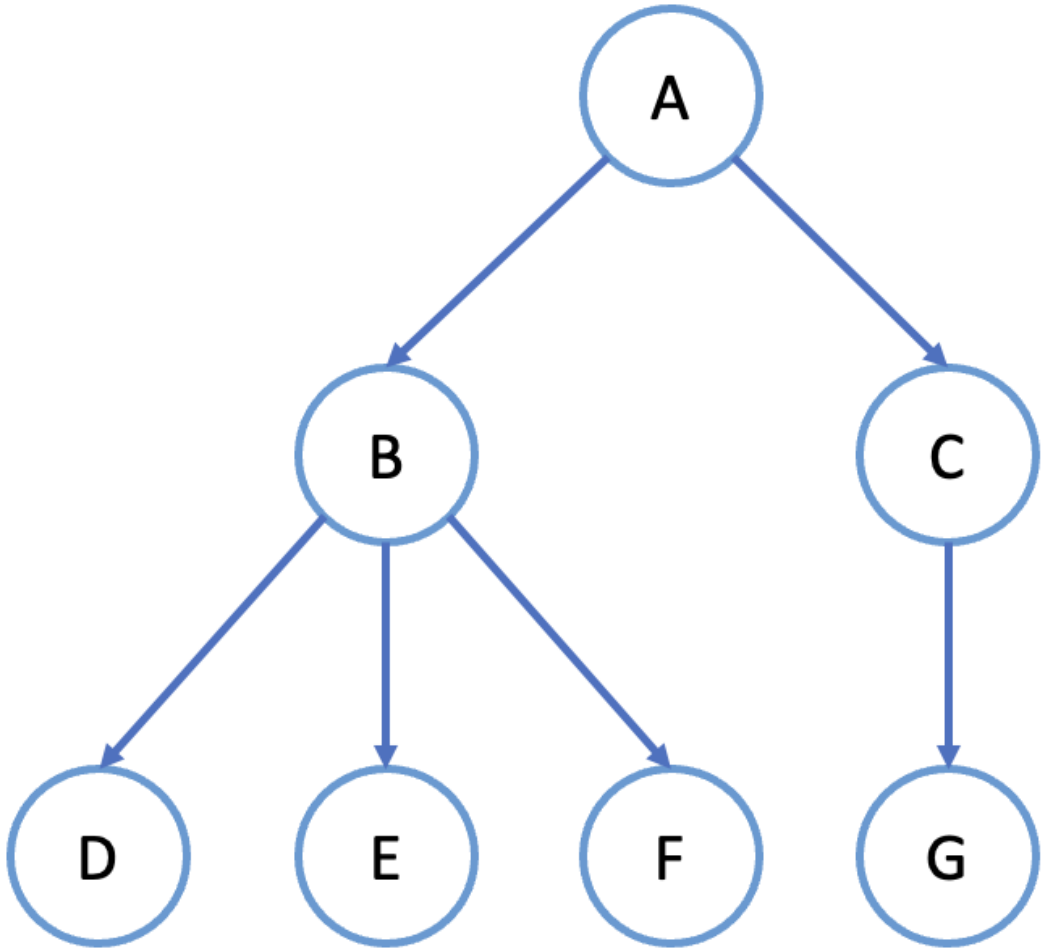
- 分支度 (degree) : 該節點子節點的數量
- 權重 (weight) : 若邊有權重，則樹權重為所有邊權重的總和
- 森林 (forest) : 一些樹的集合，一個節點也可以為一個森林
- 深度 (depth) : 該節點到根的距離
- 樹高 (high) : 樹最深的深度為樹高
- 樹直徑 (diameter) : 一棵無根樹中任兩節點中最遠的距離
- 樹半徑 (radius) : 一棵無根樹中，選定一個根節點，使得離該節點最遠的葉節點最小

Tree

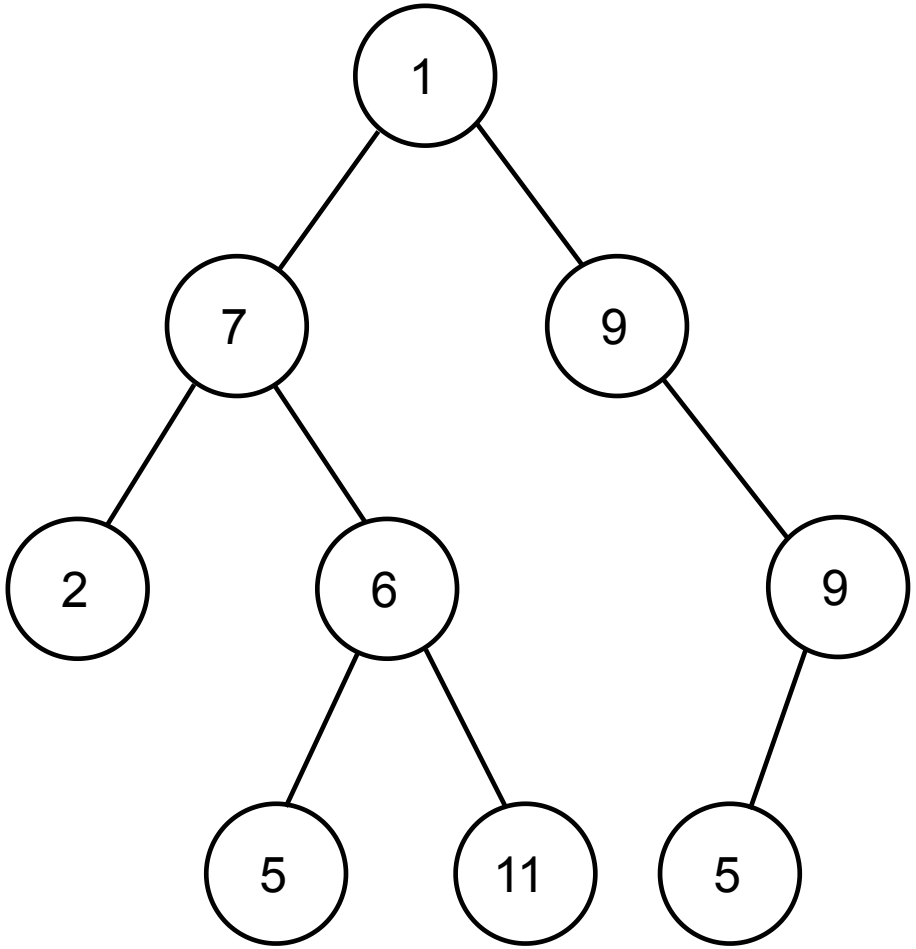
特性

1. 樹沒有環。
2. 樹上所有點之間都相連通。
3. 任意兩點之間只有唯一一條路徑。
4. 在樹上任意添加一條邊，就會產生環。
5. 在樹上任意刪除一條邊，一顆樹就裂成兩棵樹。
6. 邊數等於點數減一。

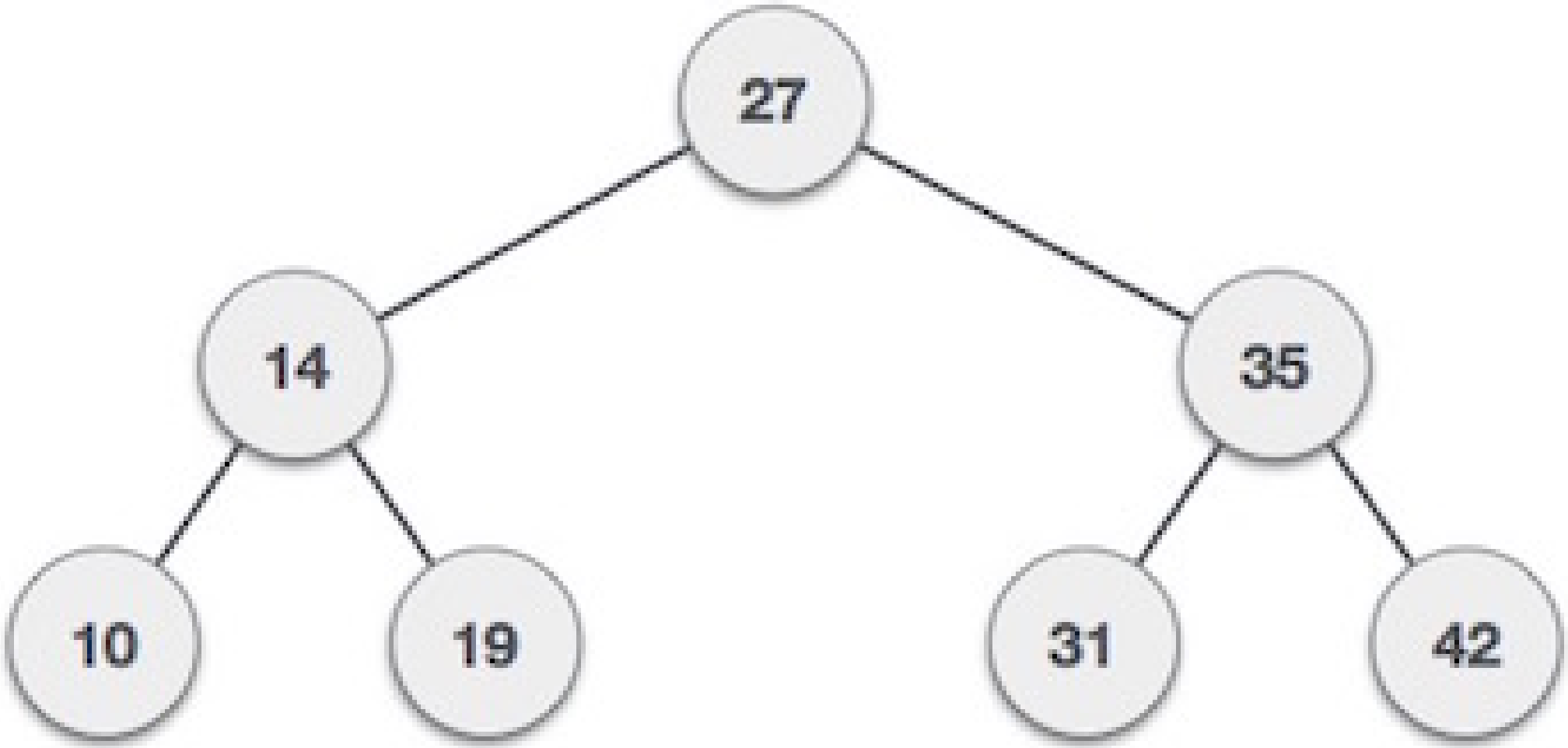
Tree



Tree



Tree



Binary Tree

Binary Tree

定義

- 樹的特化，當樹的每個節點最大分支度為 2 時，稱之為二元樹(Binary Tree)
- 每個節點可以有 0、1、2 個子節點

Binary Tree

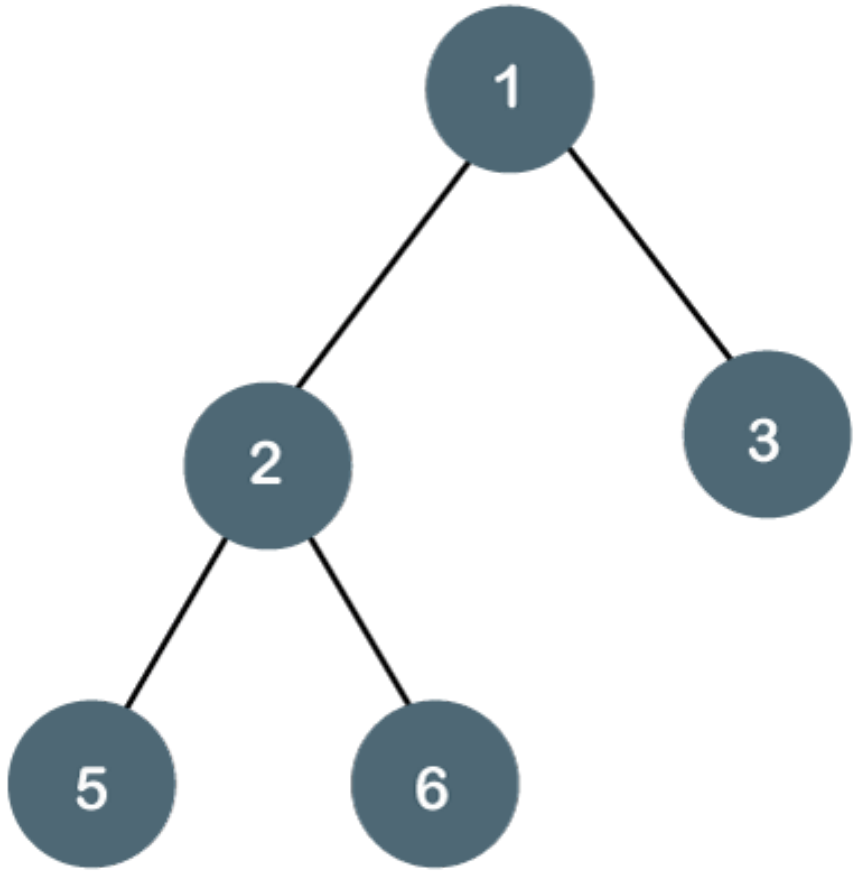
二元樹的種類

- 完滿二元樹 (full binary tree)
- 完備二元樹 (complete binary tree)
- 完美二元樹 (perfect binary tree)

Binary Tree

完滿二元樹 (full binary tree)

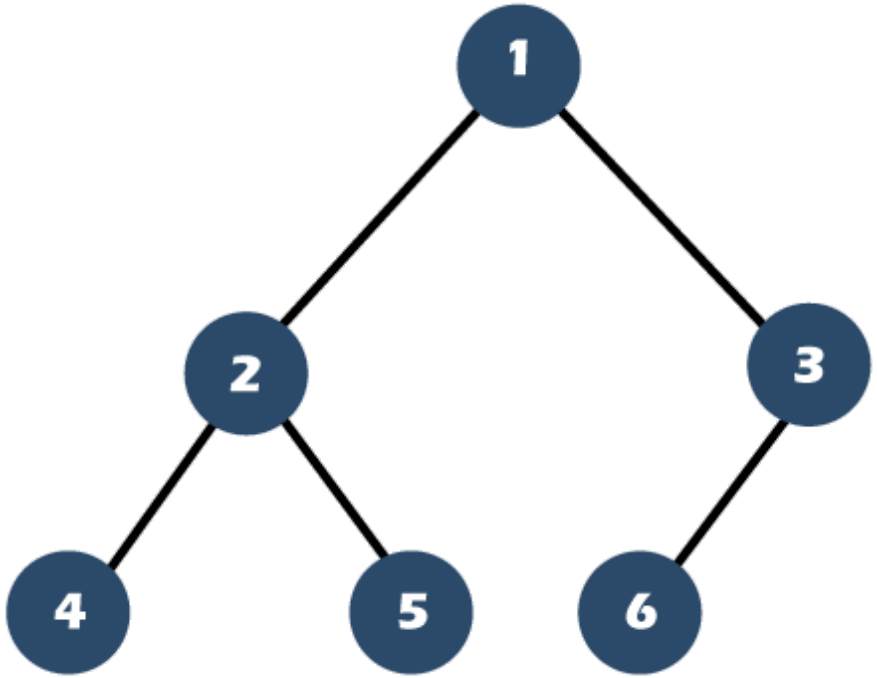
- 除了葉節點以外，每個節點都有兩個子節點。



Binary Tree

完備二元樹 (complete binary tree)

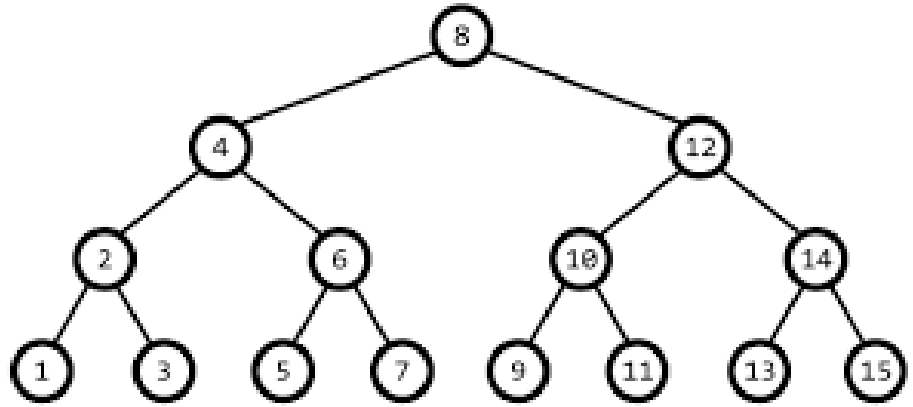
- 各層節點全滿，除了最後一層，最後一層節點全部靠左。



Binary Tree

完美二元樹 (perfect binary tree)

- 各層節點全滿。既是 full binary tree 也是 complete binary tree 。

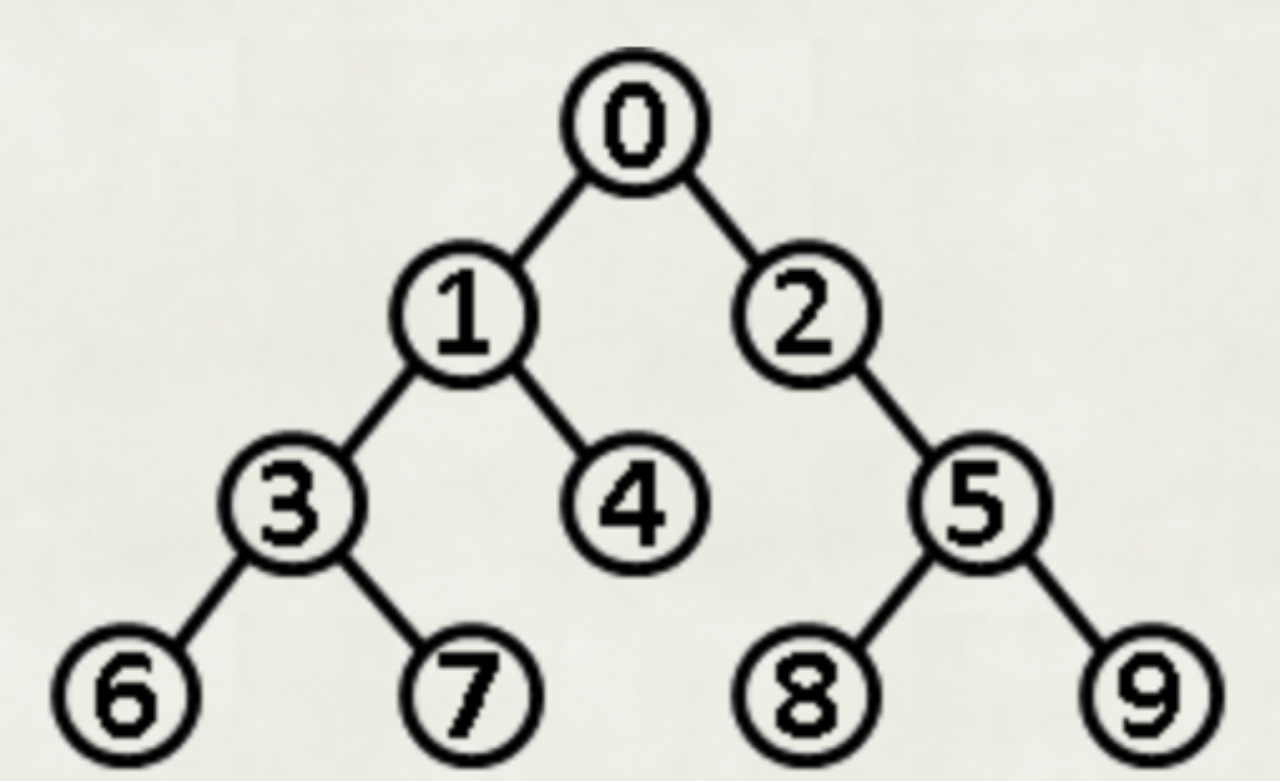


Binary Tree Traversal

Binary Tree Traversal

遍歷二元樹

- 層序走訪 (Levelorder)
- 前序走訪 (Preorder)
- 中序走訪 (Inorder)
- 後序走訪 (Postorder)

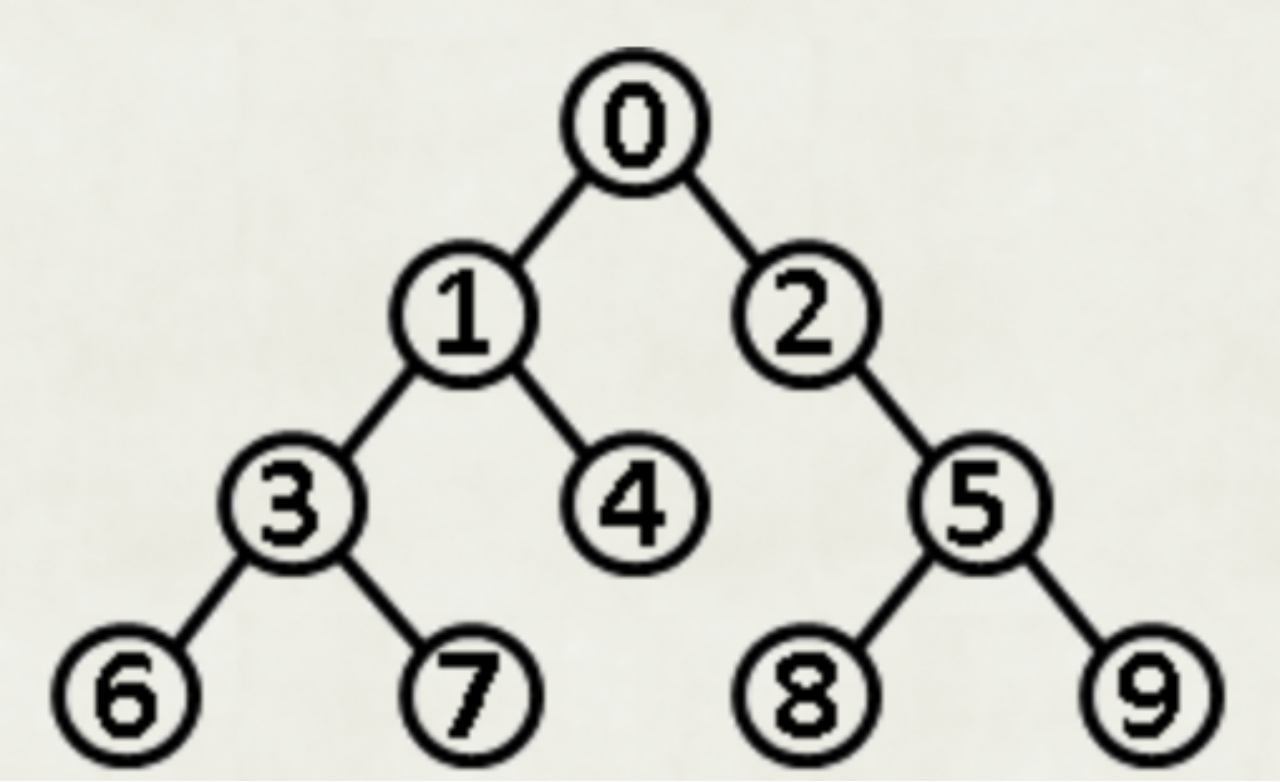


Binary Tree Traversal

層序走訪 (Levelorder)

- 從上到下，由左而右走訪 (Breadth-first Search)

> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

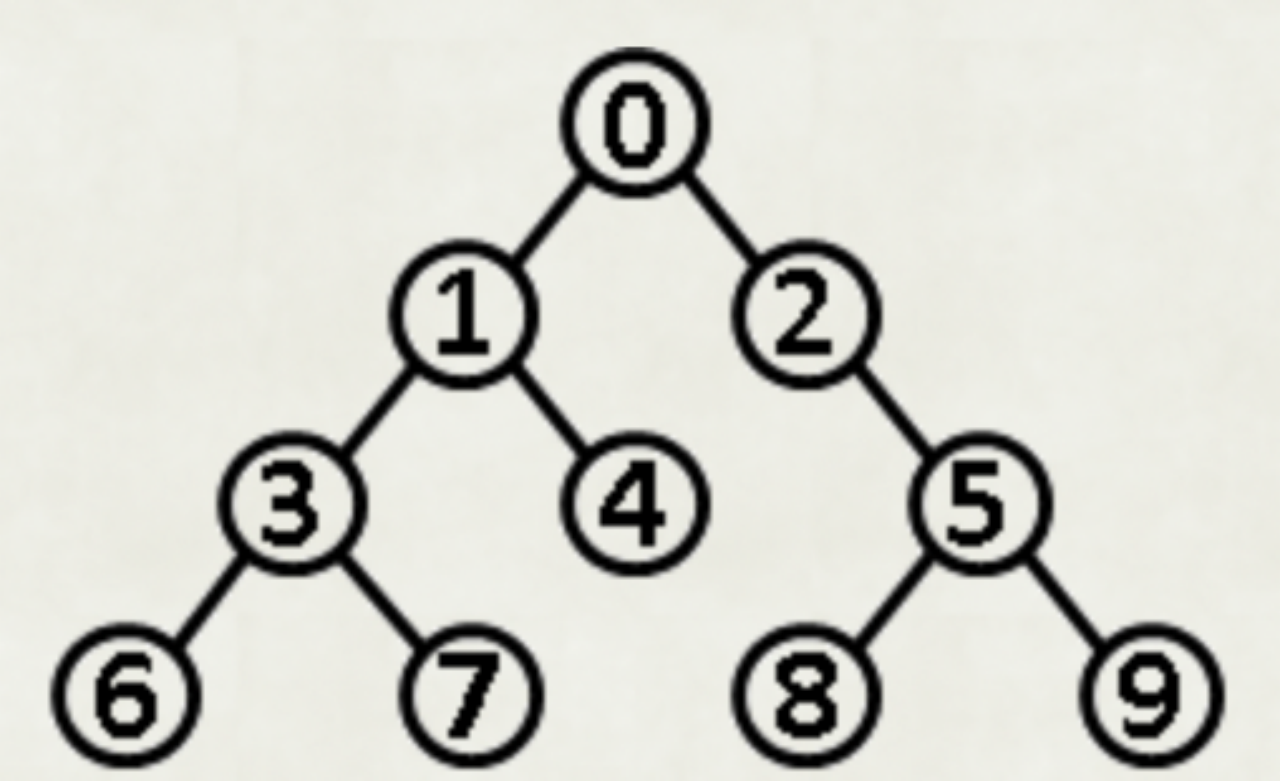


Binary Tree Traversal

前序走訪 (Preorder)

- 根、左子樹、右子樹
  (Depth-first Search)

> 0, 1, 3, 6, 7, 4, 2, 5, 8, 9

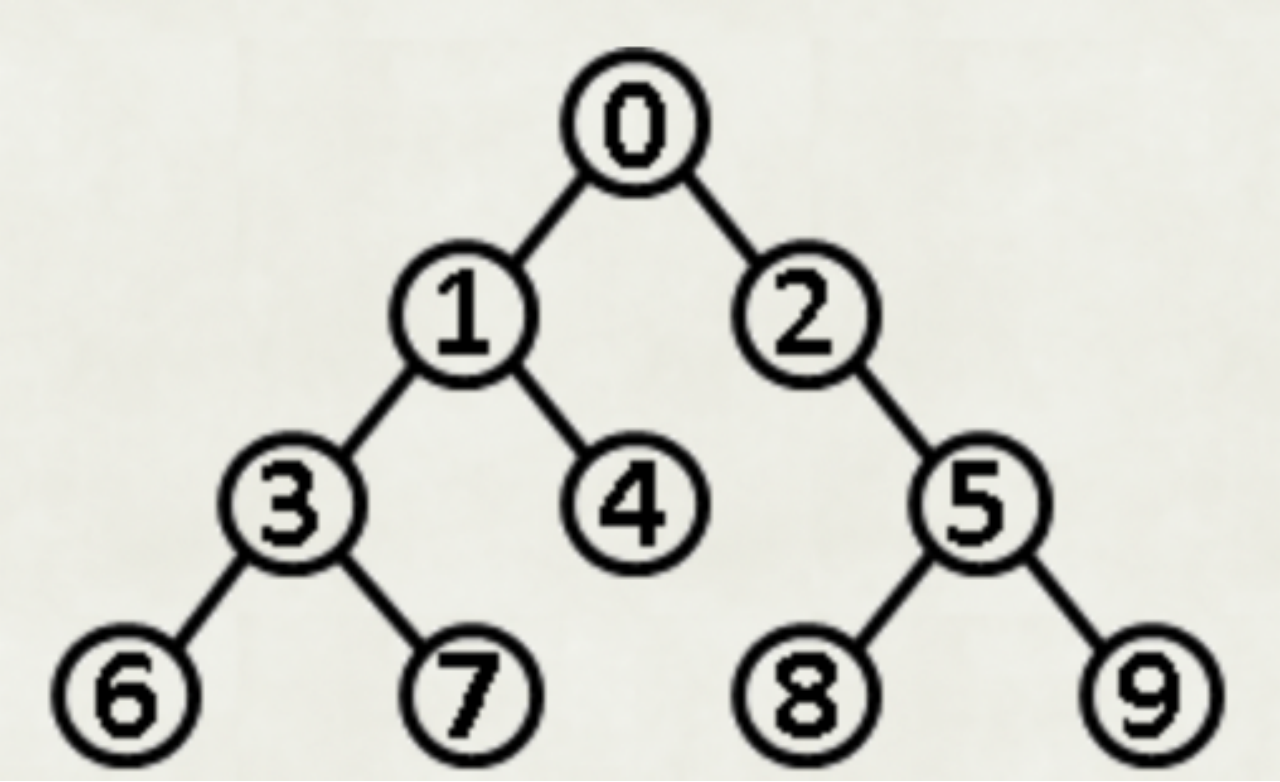


Binary Tree Traversal

中序走訪 (Inorder)

- 左子樹、根、右子樹
  (Depth-first Search)

> 6, 3, 7, 1, 4, 0, 2, 8, 5, 9

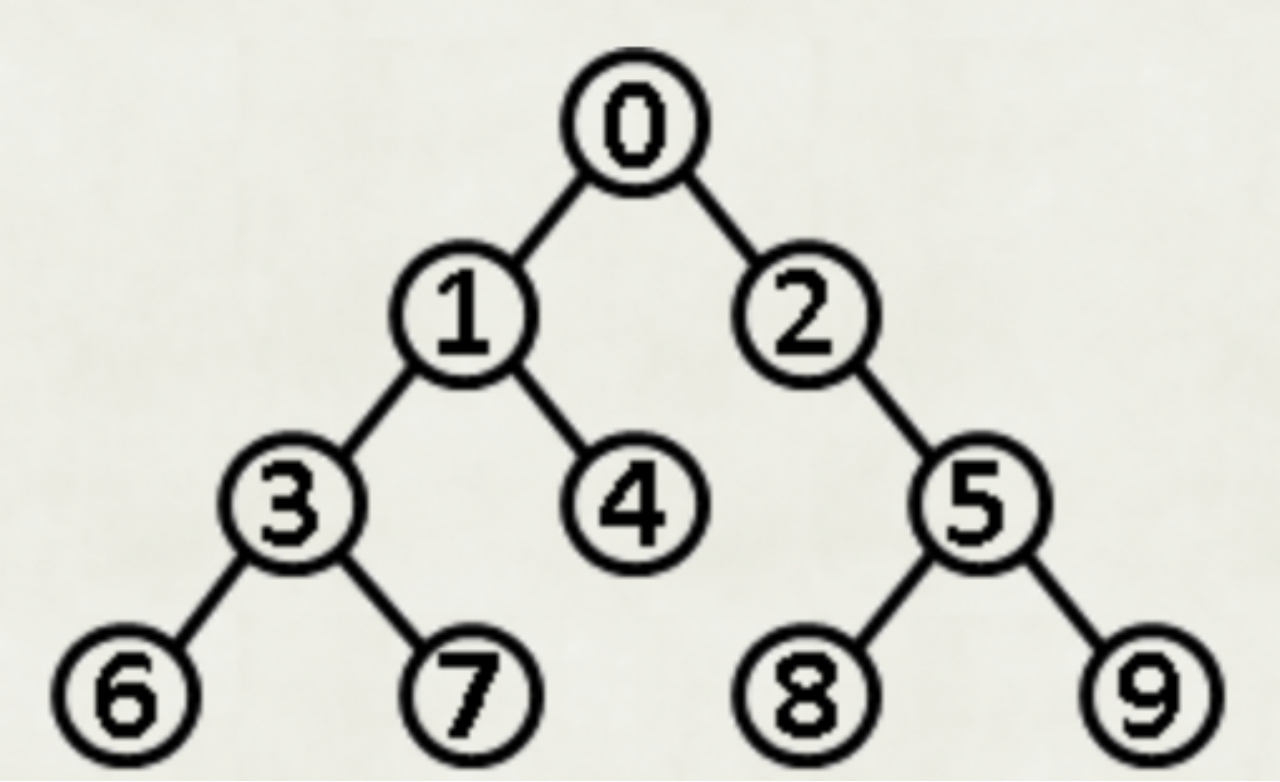


Binary Tree Traversal

後序走訪 (Postorder)

- 左子樹、右子樹、根
  (Depth-first Search)

> 6, 7, 3, 4, 1, 8, 9, 5, 2, 0



Binary Search Tree

Binary Search Tree

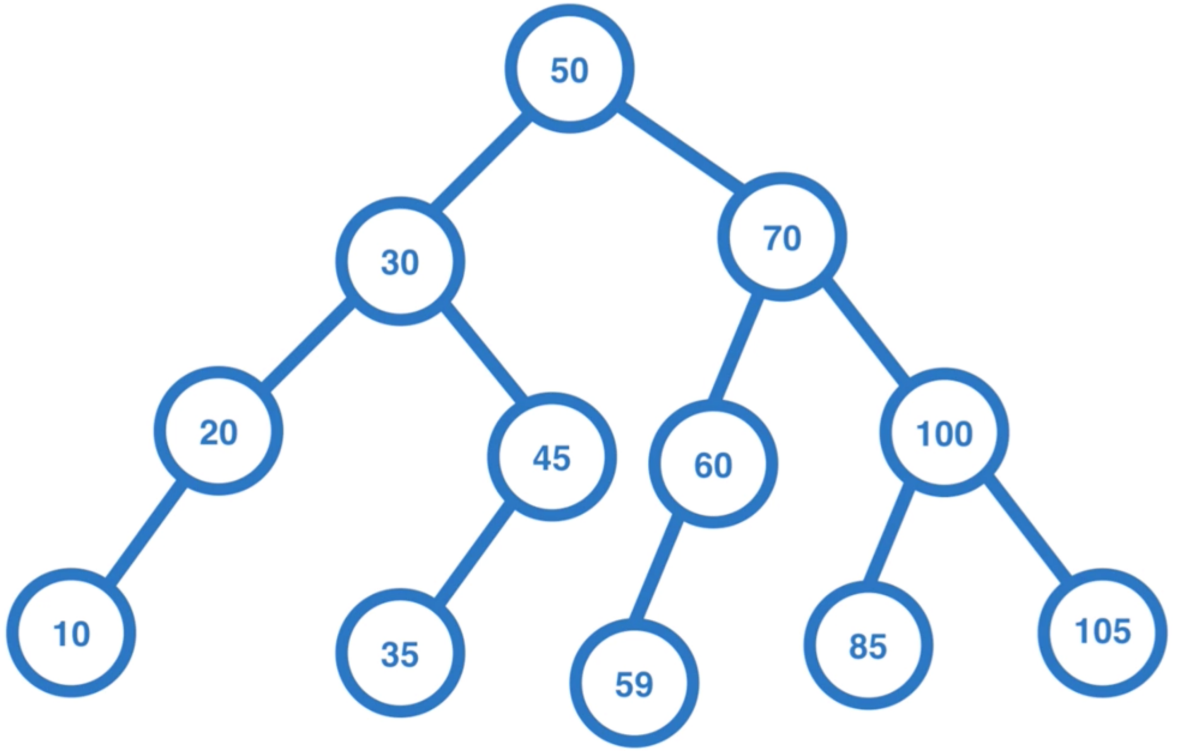
二元搜尋樹的機制

1. 可隨時增加數值
2. 可隨時刪除數值
3. 檢查數值是否存在

Binary Search Tree

二元搜尋樹原理

- 左子樹小於父節點
- 右子樹大於父節點

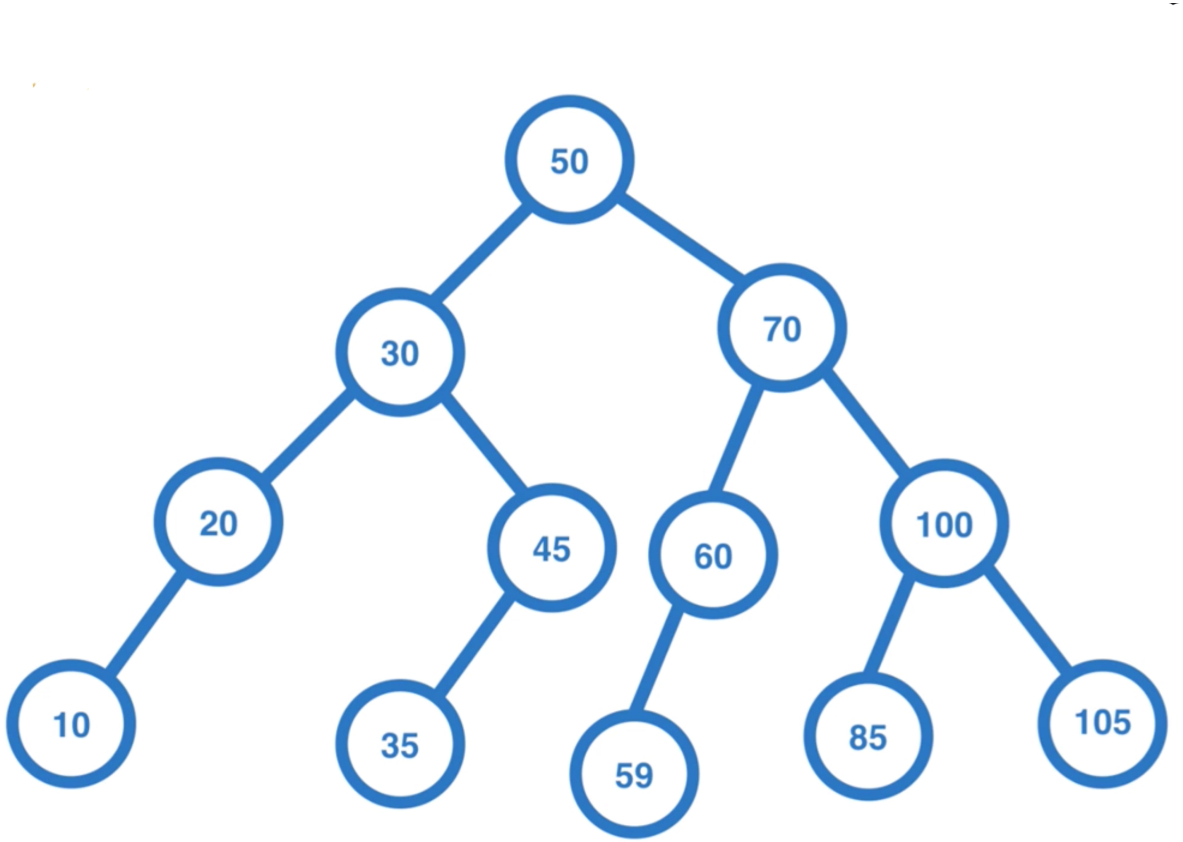


Binary Search Tree

二元搜尋樹查詢

例：找查 45 是否存在

> 45 < 50 (找左子樹)
> 45 > 30 (找右子樹)
> 45 == 45 (找到了！)

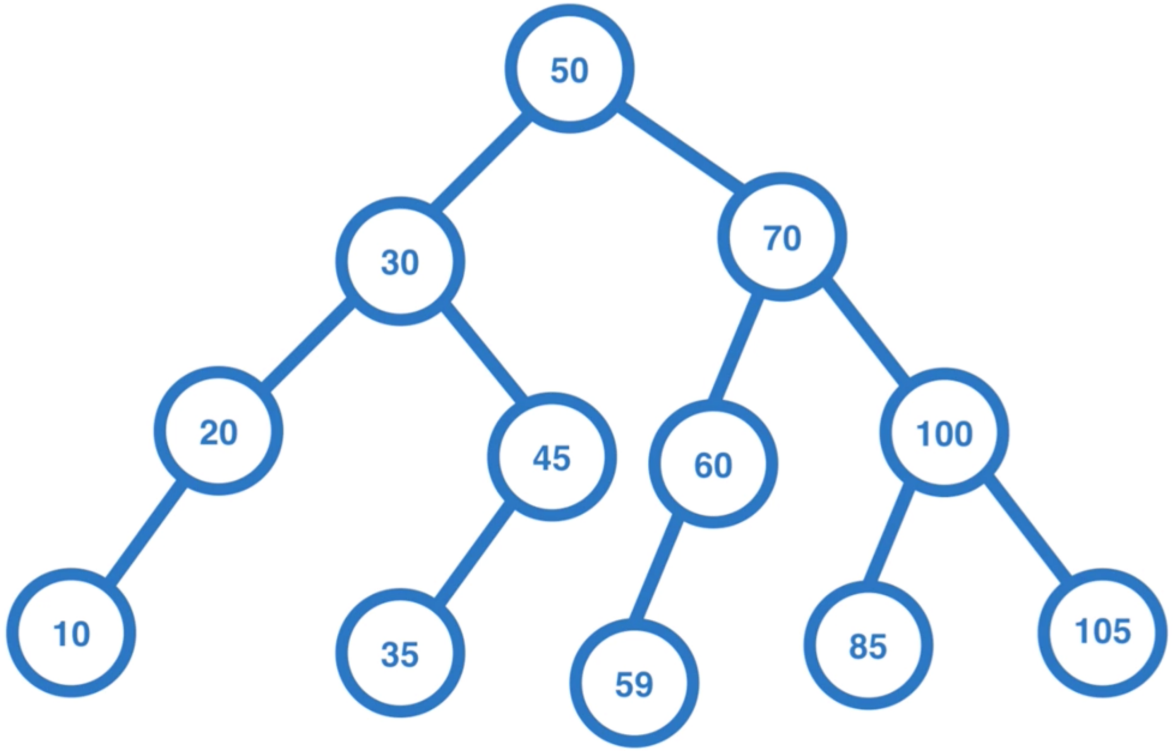


Binary Search Tree

二元搜尋樹查詢

例：找查 40 是否存在

> 40 < 50 (找左子樹)
> 40 > 30 (找右子樹)
> 40 < 45 (找左子樹)
> 40 > 35 (找右子樹)
> 右子樹為空 (不存在)

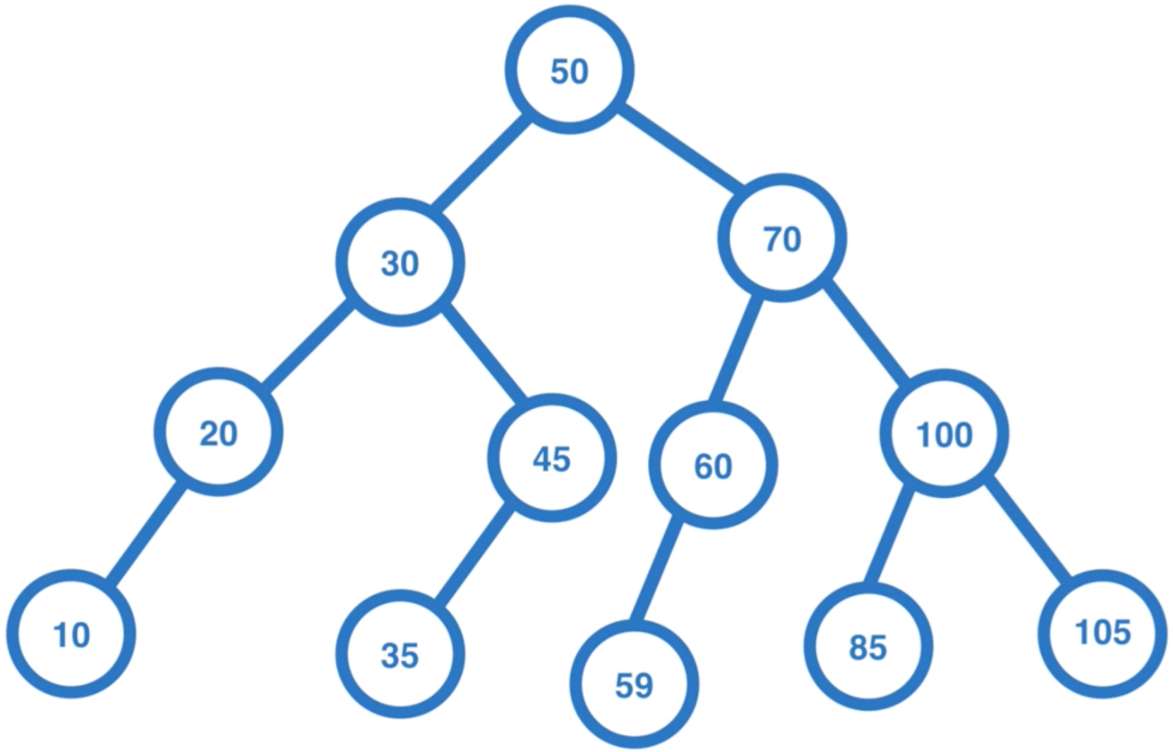


Binary Search Tree

二元搜尋樹新增

例：新增 40

> 40 < 50 (找左子樹)
> 40 > 30 (找右子樹)
> 40 < 45 (找左子樹)
> 40 > 35 (找右子樹)
> 右子樹為空 (插入 40)

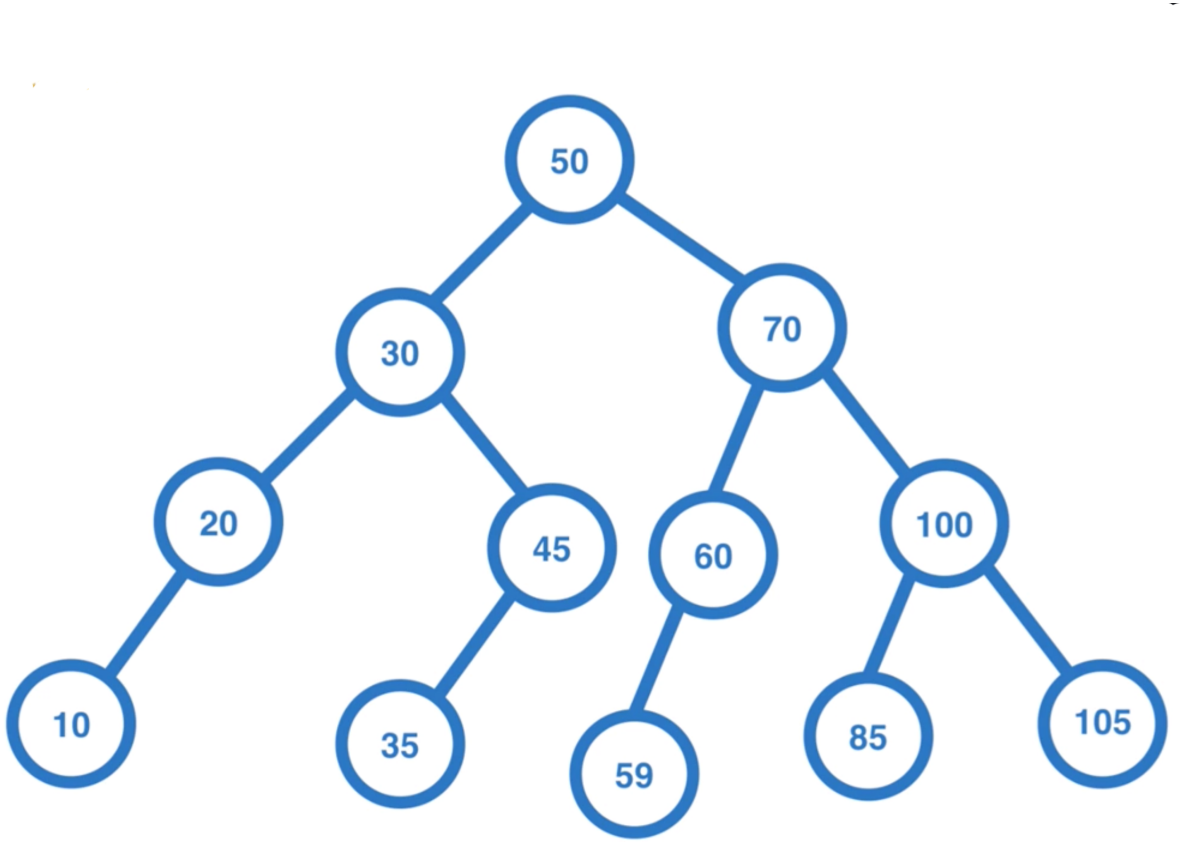


Binary Search Tree

二元搜尋樹刪除

例：刪除 30

> 左子樹的最右節點 45
> 刪除 30 替換成 45
> 35 成為 30 的右節點



Binary Search Tree

二元搜尋樹比較

	新增	刪除	搜尋
陣列	O(1)	O(N)	O(N)
二元搜尋樹	O(log N)	O(log N)	O(log N)

Implementation

Implementation

樹的實作

1. 使用指標
2. 使用陣列

Implementation

樹的實作

- 使用指標

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

Implementation

樹的實作

- 使用陣列

```
int data[N];
int left[N];
int right[N];
```

Implementation

樹的走訪

- 使用遞迴走訪 (Depth-first Search)
- 循序走訪 (Breadth-first Search)

Exercise

Exercise

easy

- Tree Postorder
- 四則運算計算機
- 加分二叉树
- 打印樹
- Binary Search Tree (BST)

Exercise

medium

- apcs 樹狀圖分析 (Tree Analyses)
- Tree Recovery
- 樹葉節點到根節點之路徑

hard

- 是否為樹
- 找關鍵人物

Any Question ?