C/C++ 進階班
資料結構

二元樹基礎
(Binary Tree)

李耕銘

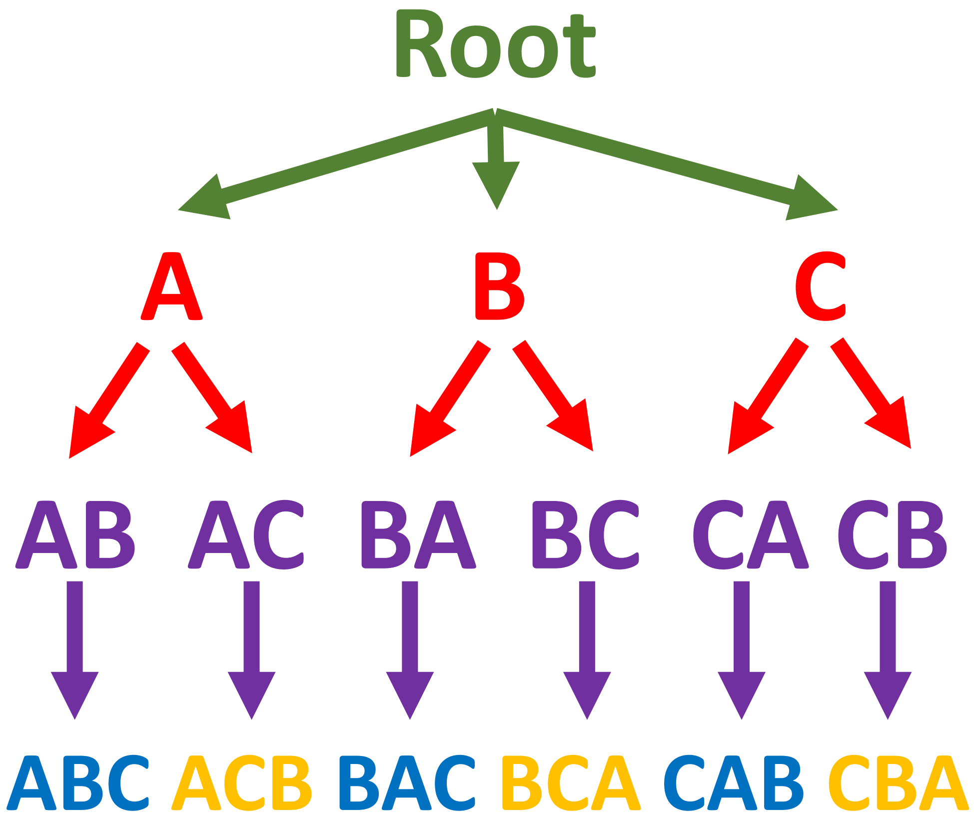
課程大綱

- 樹的簡介&定義
- 二元樹
  - 簡介&定義
  - 建立二元樹
- 二元樹的操作
  - 新增(Insert)與搜尋(Search)
  - 刪除(Delete)
  - 尋訪(Traversal)

樹的簡介&定義

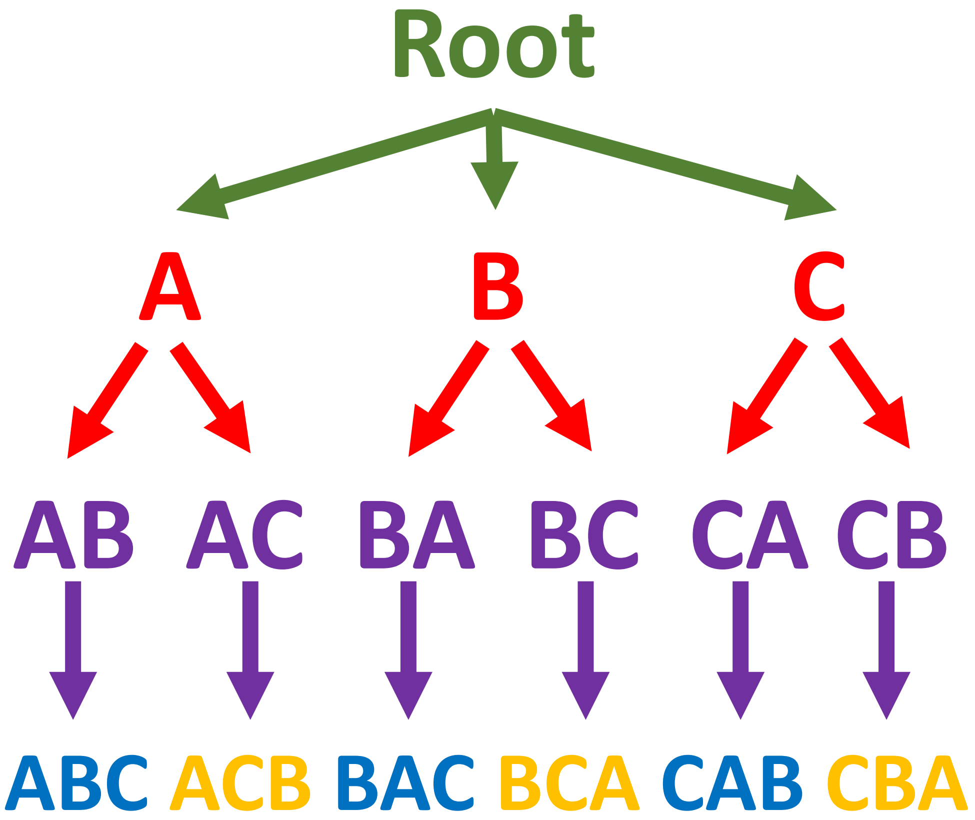
樹的簡介

- 從一個樹根出發(Root)
- 具有階層結構
  - ➢ 具有方向/階層關係
  - ➢ 但在尋訪時可以返回
- 每個節點都可以連接到零個或多個節點
  - ➢ 但無法形成環(cycle)
    - ✓ 任一點出發後無法回到原地
  - ➢ 當連結的節點個樹 ≦ 2 時，即為二元樹



樹的簡介

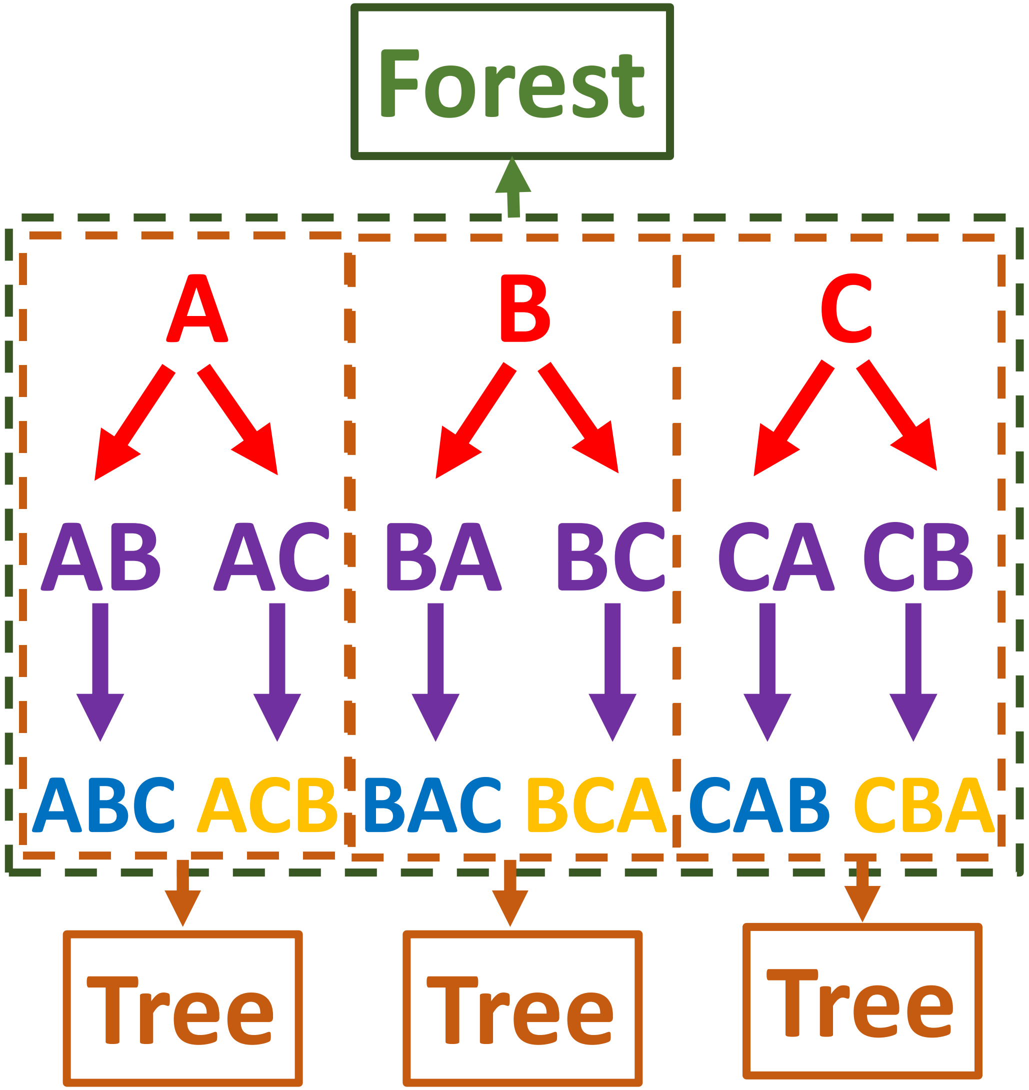
- 樹的應用
  - ➢ 列舉所有情況
  - ➢ 地圖探索 BFS
  - ➢ 資料儲存
  - ➢ 搜尋資料
  - ➢ 分類器
  - ➢ 編碼



樹的定義

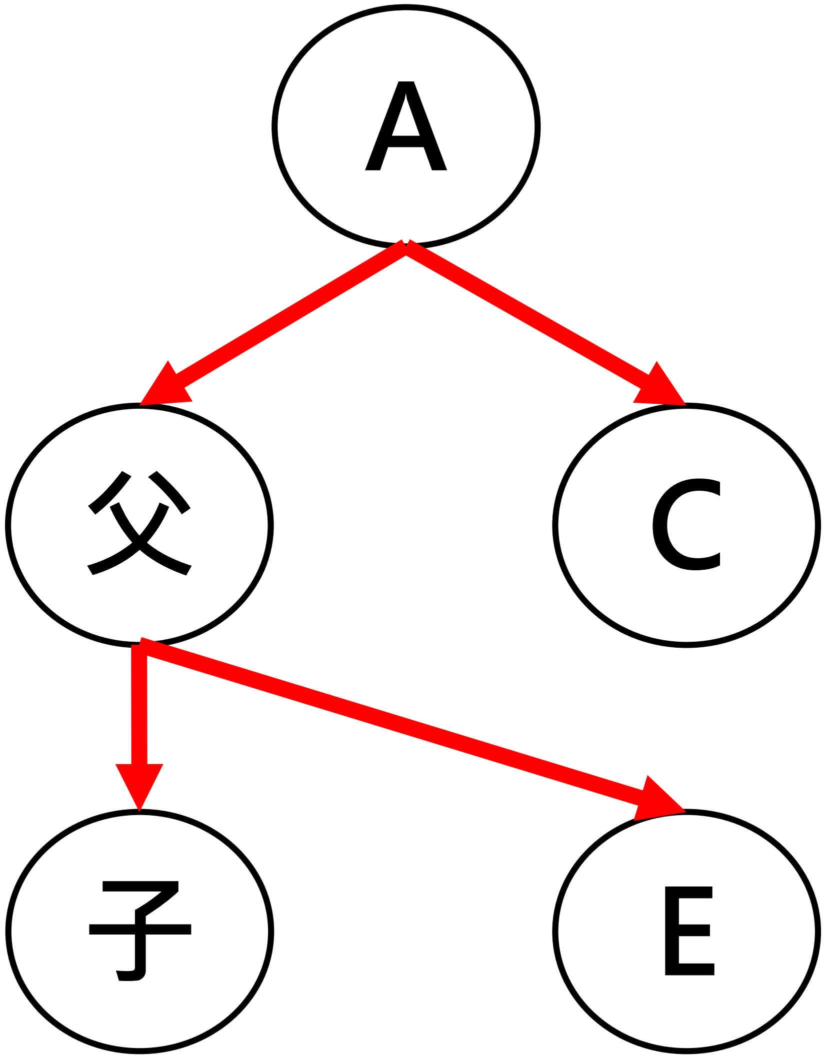
1. 由多個節點以及邊所構成
   - ➢ 但只能存在一個根節點 Root
2. 其餘節點為互不重複(互斥)的集合
   - ➢ 稱為根節點的子樹 (subtree)
   - ➢ 每個節點又可以指向0個或多個節點
   - ➢ 每個節點只能被一個節點指向
3. 無法形成環(Cycle)

- 若有多個根節點 Root，則稱為樹林 Forest



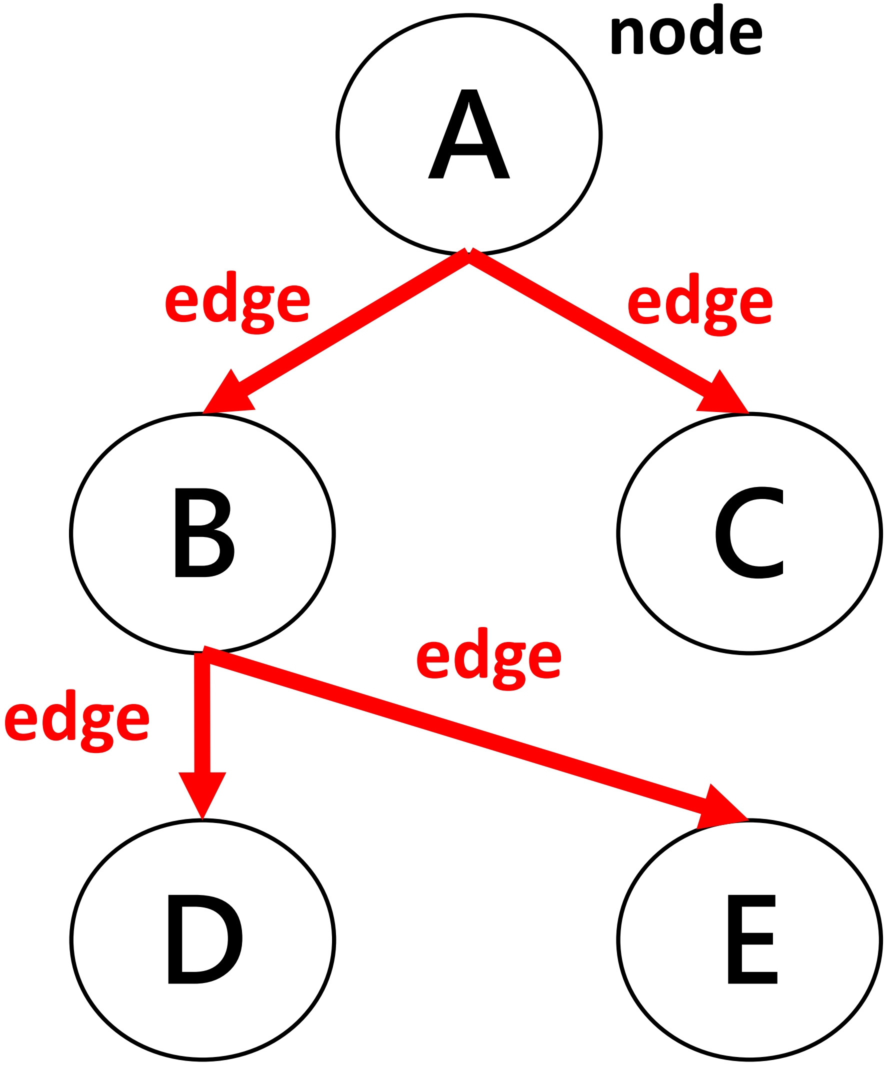
樹的簡介&定義

- 節點(Node) 分類
    1. 父節點 (Parent node)
    2. 子節點 (Child node)
    3. 根節點 (Root node)：沒有父節點，最上層
    - ➢ 方向由父節點→子節點
    - ➢ 每個節點只會有一個父節點 (根節點除外)
- 除根節點外的所有節點必須都滿足樹的定義
    - ➢ 這些又稱為子樹 subtree
- 從 Root 出發搜尋特定節點
    - ➢ 只有唯一一條路徑 (Path)



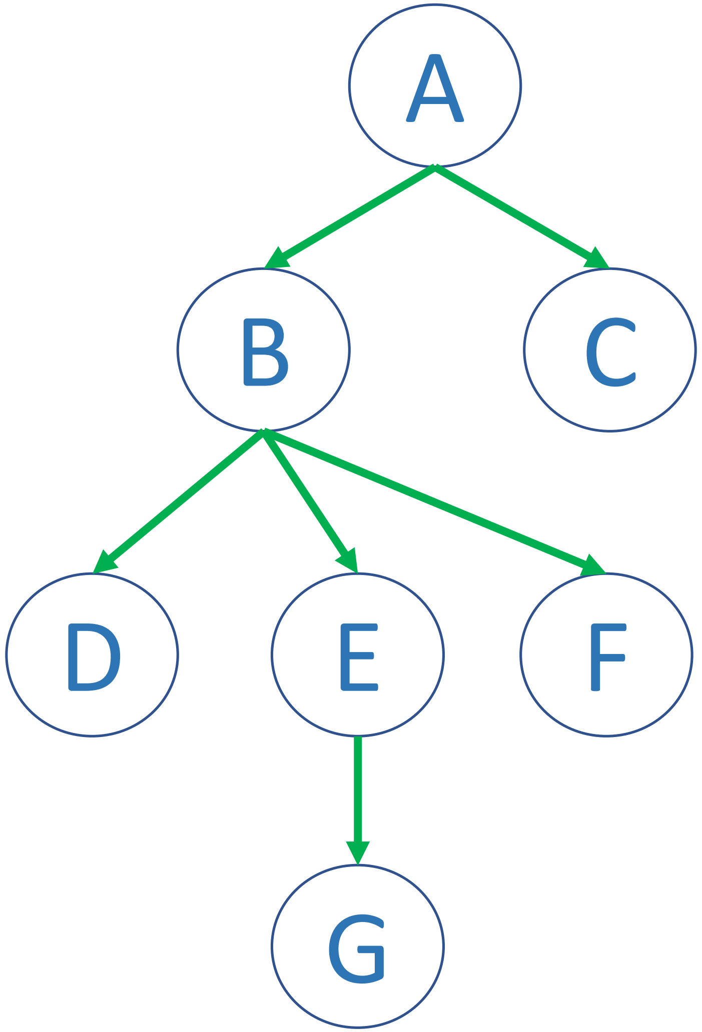
樹的簡介&定義

- A
  - 父節點：無 (此為根節點 Root)
  - 子節點：B、C
- B
  - 父節點：A
  - 子節點：D、E
- C
  - 父節點：A
  - 子節點：無



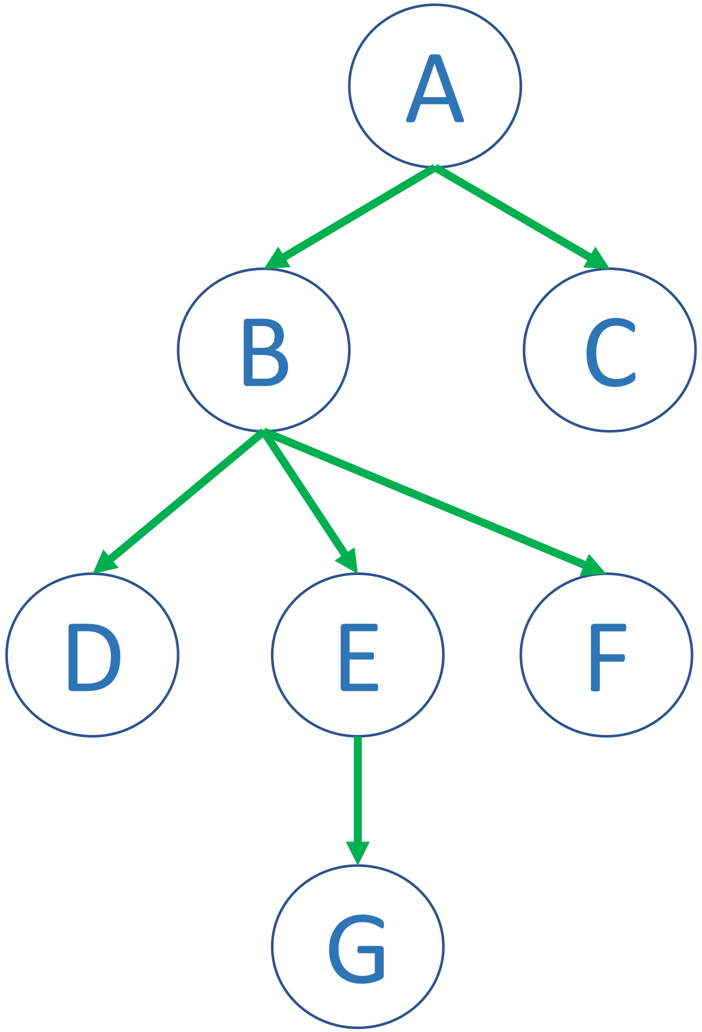
樹的簡介&定義

- 分歧度(Degree)
  - Degree of Node
    - ✓ A：2
    - ✓ B：3
    - ✓ C ：0
  - Degree of Tree = max{ Degree of all nodes }
    - ✓ Degree of Tree：3
- 二元樹的分歧度為 2



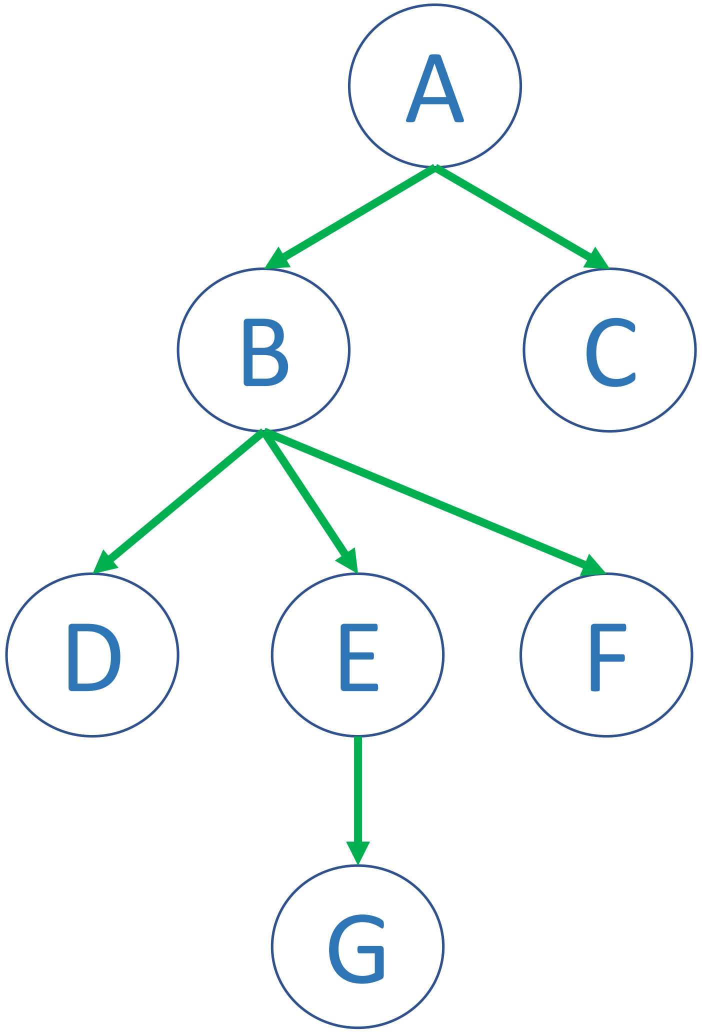
節點分類

- Leaf node
  - ✓ 沒有子節點的 node
  - ✓ 又稱為 external node
  - ✓ Ex：C、D、F、G
- Non-Leaf node
  - ✓ 有子節點的 noe
  - ✓ 又稱為 internal node
  - ✓ Ex：A、B、E



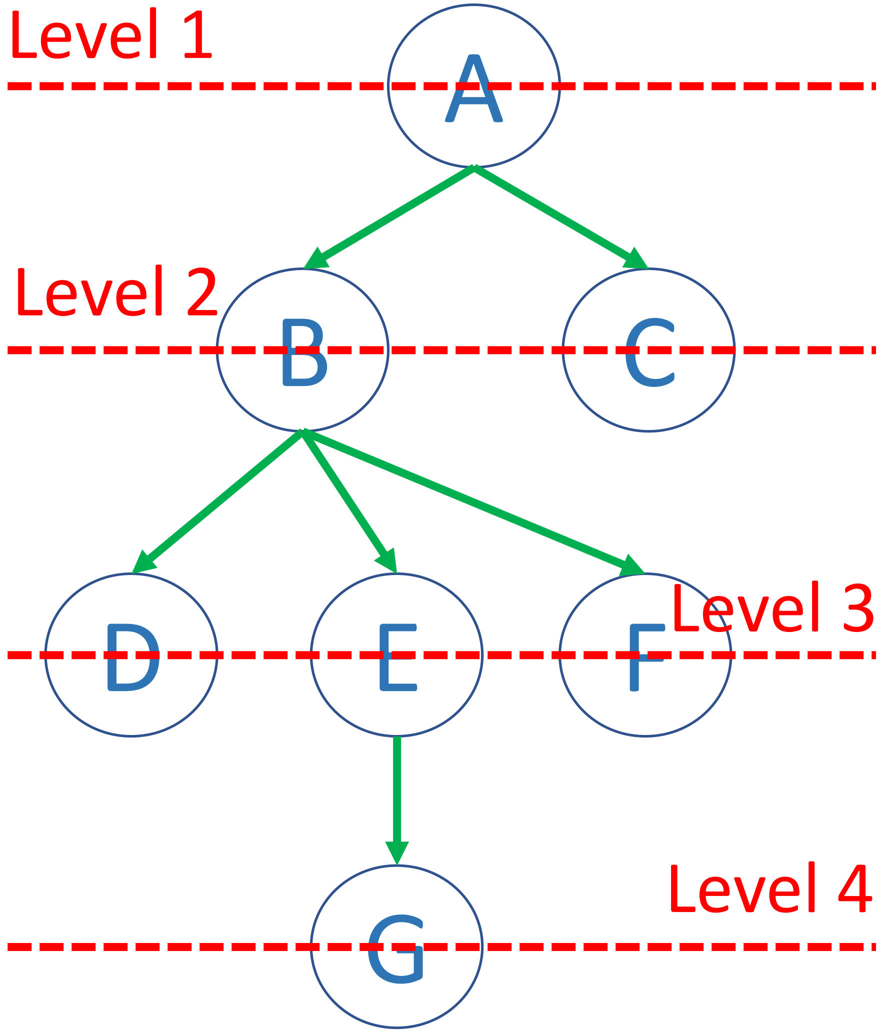
節點分類

- Sibling nodes
  - ✓ node 間彼此擁有相同的 Parent node
  - ✓ Ex：B 與 C、D 與 E 與 F
- Ancestor nodes
  - ✓ node 往 Parent 方向走會經過的 node
  - ✓ Ex：D→A、B；G→E、B、A
- Descendant nodes
  - ✓ node 往 Child 方向走會經過的 node
  - ✓ Ex：A→除 A 外的所有節點
  - ✓ Ex：B→D、E、F、G



樹的簡介&定義

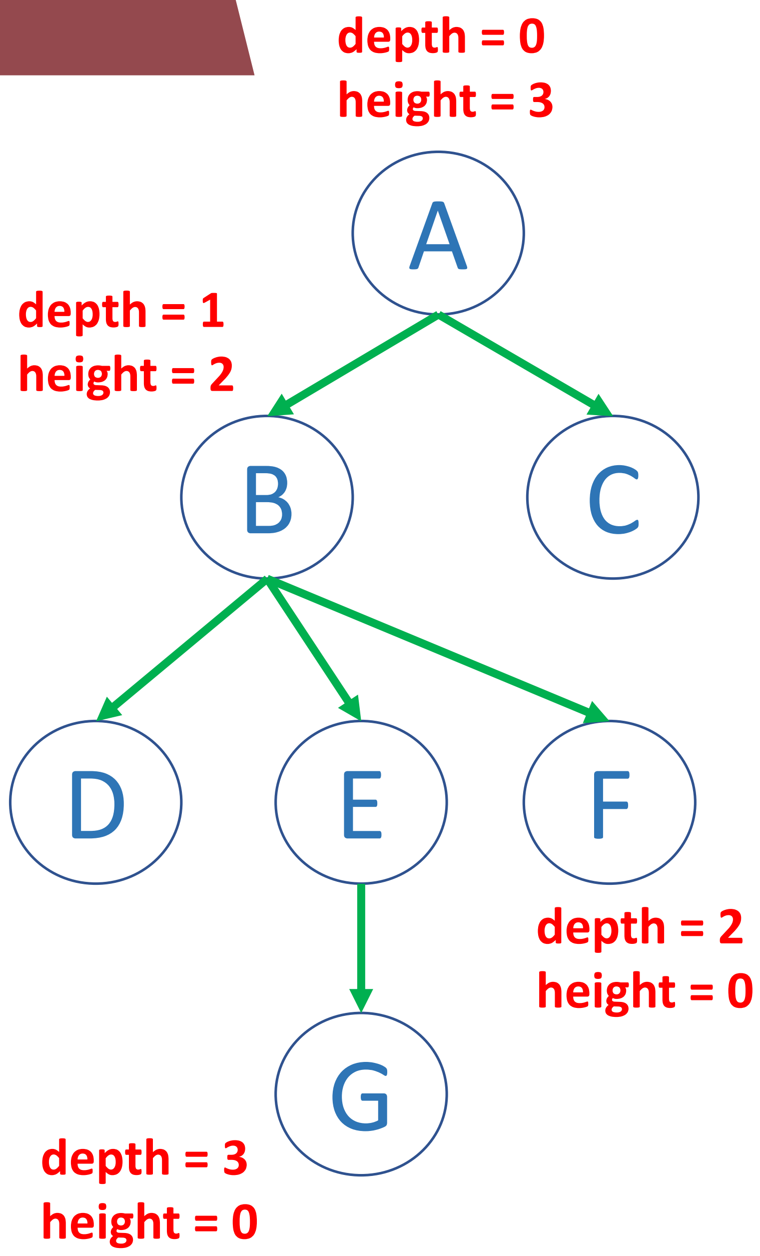
- 階層 (Level)
  - ➢ Level of node
    - ✓ 定義 Root node 的階層為 1
    - ✓ 其餘 node 的階層為其 Parent 階層+1
  - ➢ Example
    - ✓ A : 1
    - ✓ B、C : 2
    - ✓ D、E、F : 3
    - ✓ G : 4



樹的簡介&定義

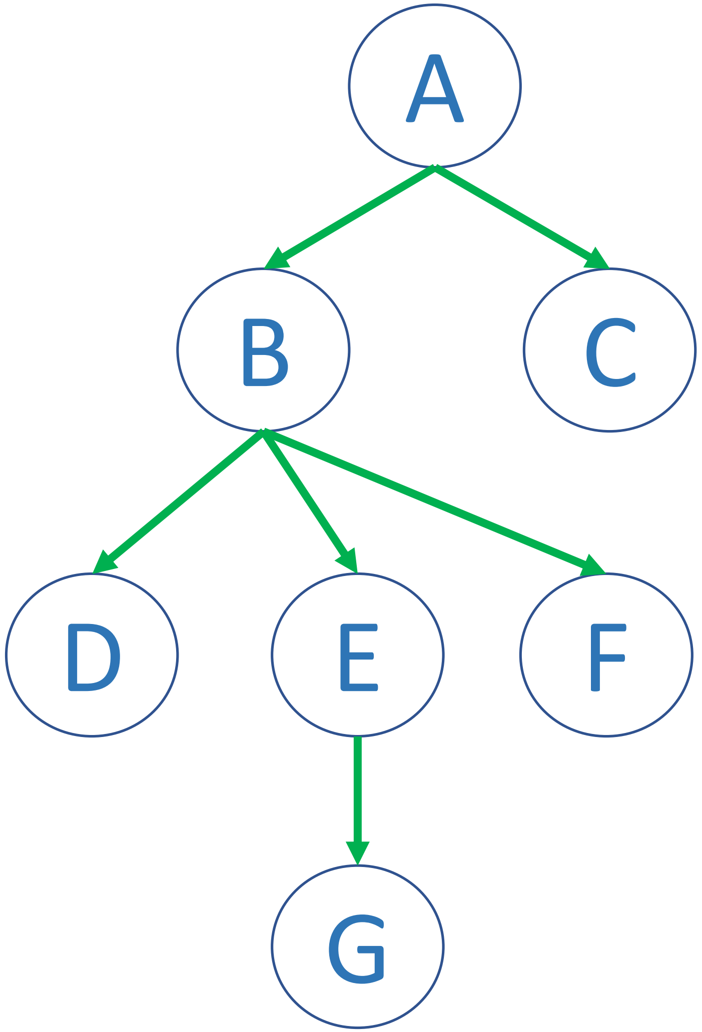
高度 (Height)

- ➢ Height of node
  - ✓ 與最遠 Descendant node 間的 edge 數
  - ✓ Example：B→2、D→0、E→1
- ➢ Height of tree：3
  - ✓ Root node 的 height
  - ✓ max{Levels of nodes}-1
- ➢ 但有些人的定義不同，解題時要注意！
  - ✓ Height of tree = max{Levels of nodes} = 4



樹的簡介&定義

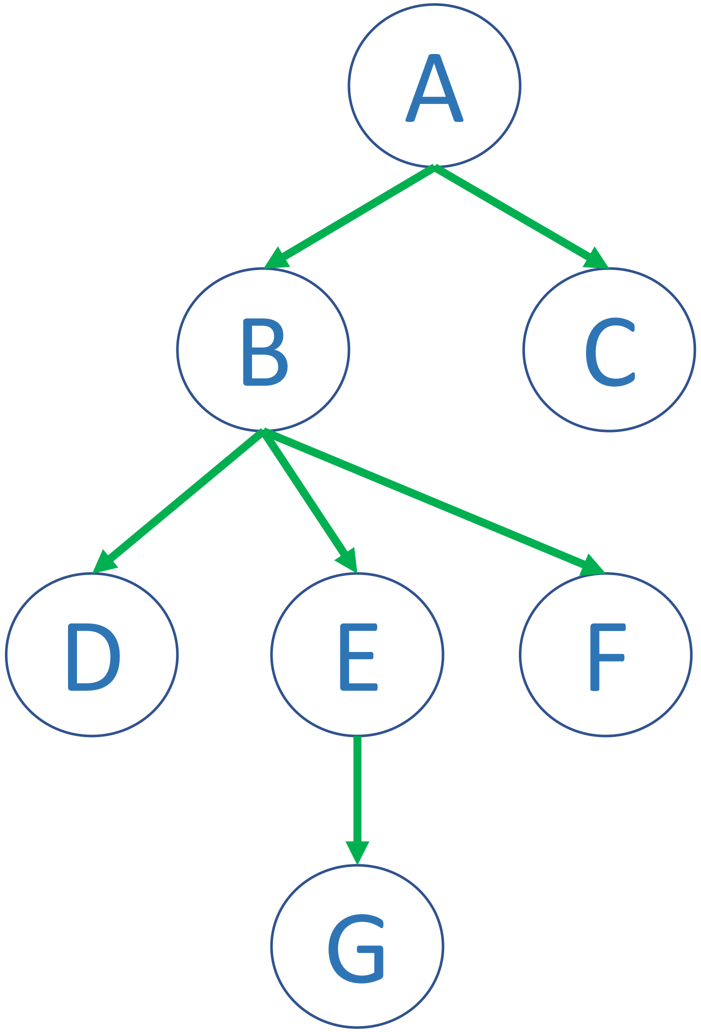
- 深度 (Depth)
  - node 與根節點間的 edge 數
  - Ex：B、C→1；D、E、F→2；G→3
- 路徑 (Path)
  - ancestor 到 descendant 連成的 edge
  - Ex：A-B-D；A-B-E-G
- 樹林 (Forest)
  - 由至少一棵、互不重複的 trees 形成的集合



樹的紀錄法

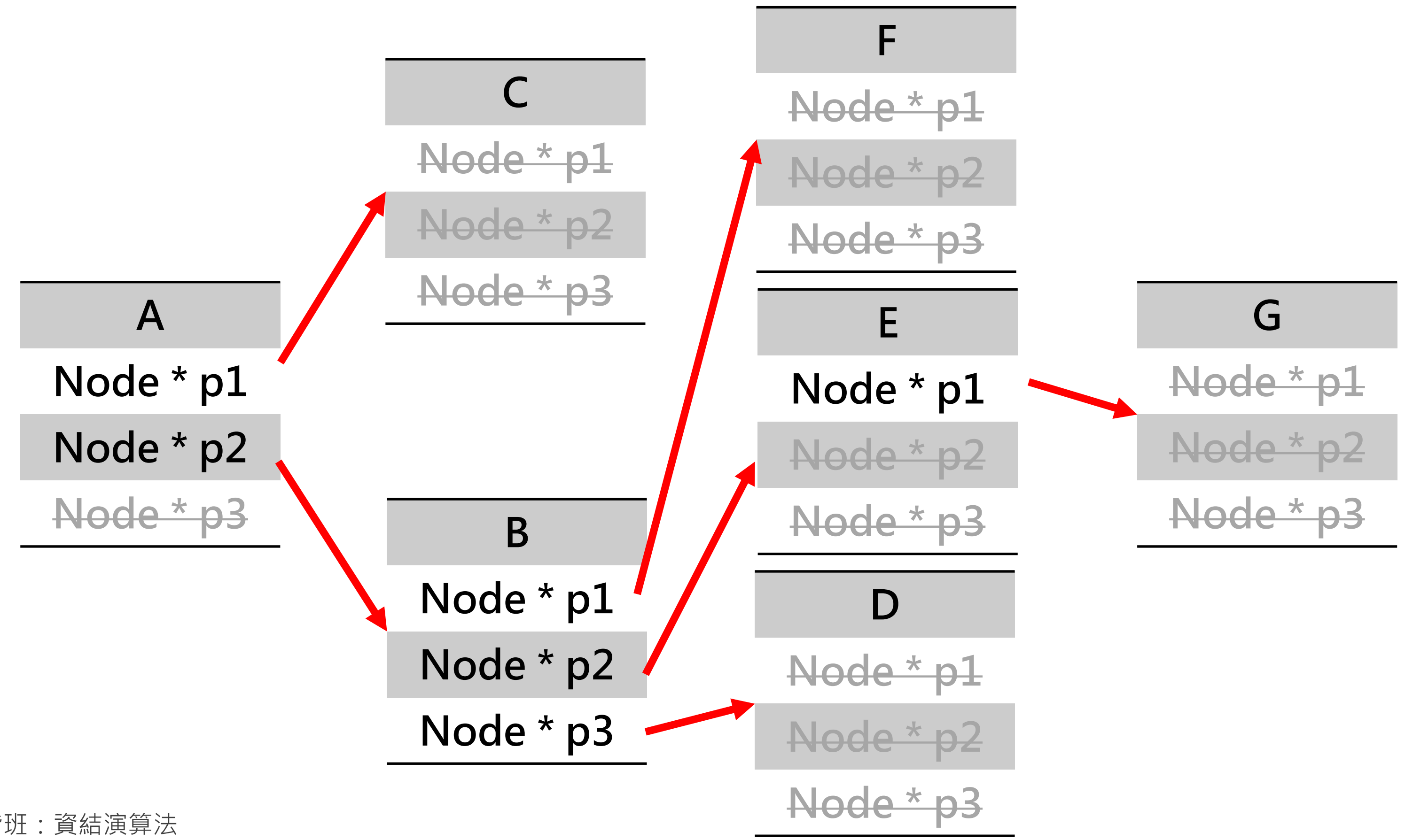
1. 括號法
   - 以括號表示父子間的從屬關係
   - 父(子1(孫)子2(孫))
   - Example：

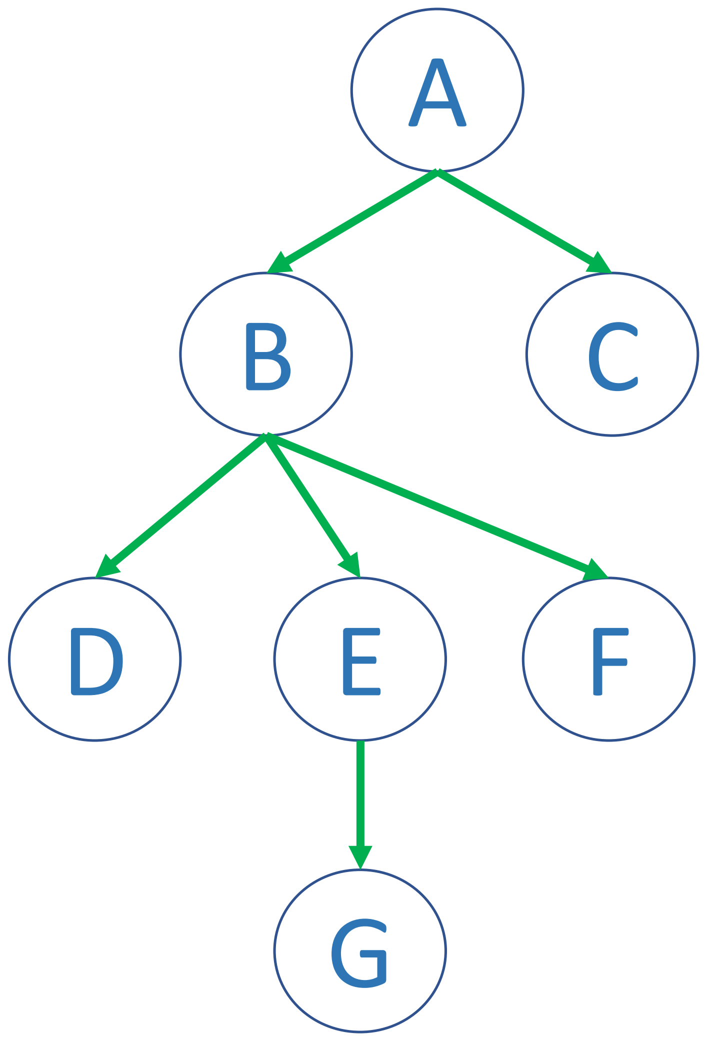
$$A(B(DE(G)F)C)$$



樹的紀錄法

2. 鏈結串列 (Linked List)
   - ➢ 透過指標連接
   - ➢ 每個節點需要有 Degree of Tree 個指標

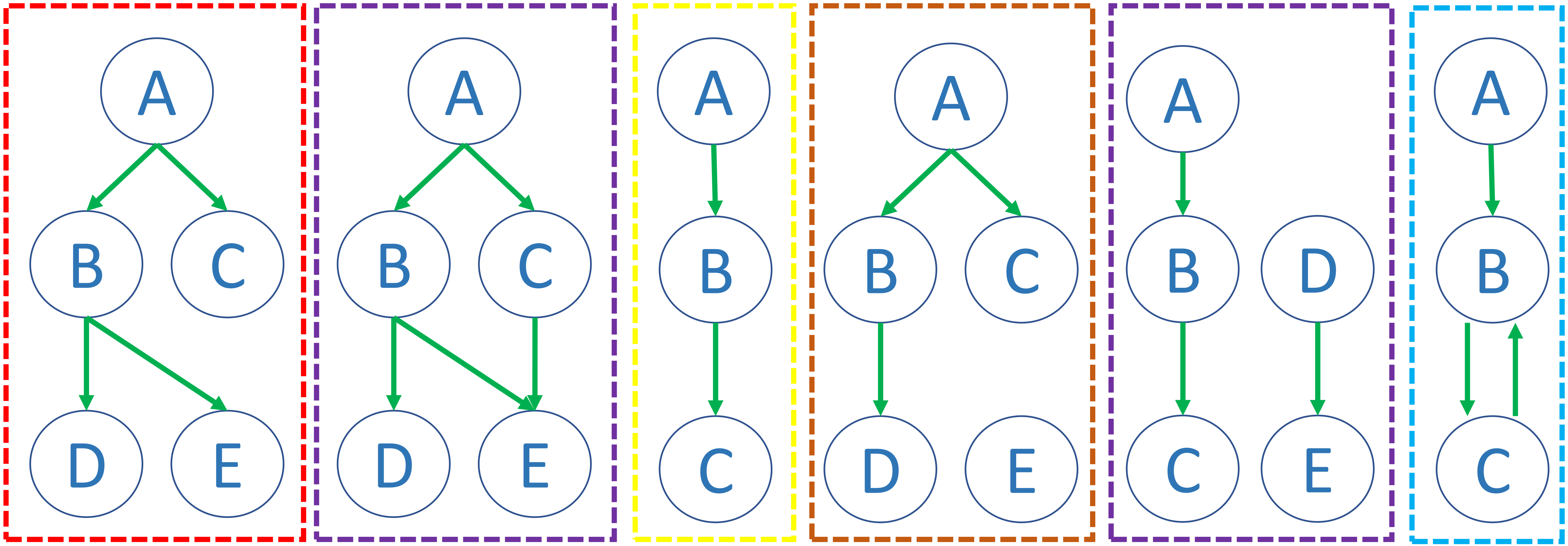




Practice

Mission

下面的六個圖中，哪些是樹？

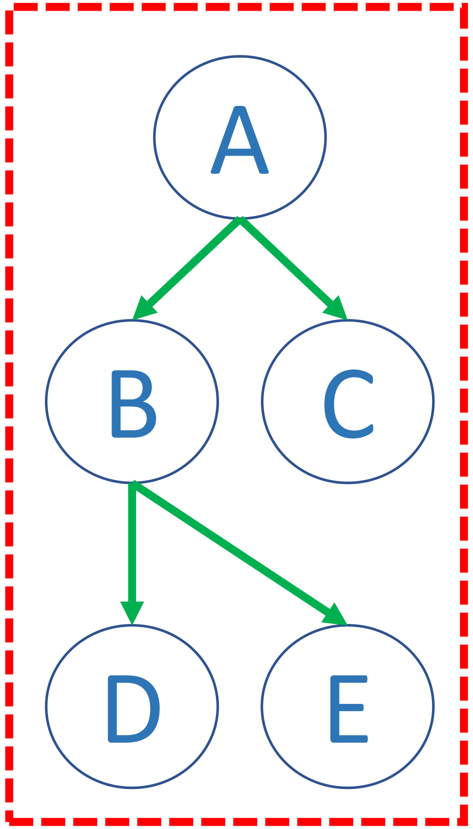


A B C D E F

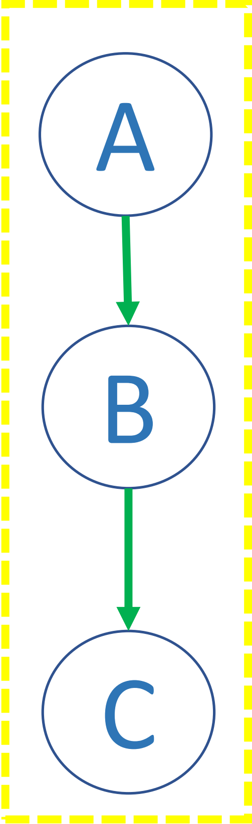
Practice

Mission

下面的六個圖中，哪些是樹？



A



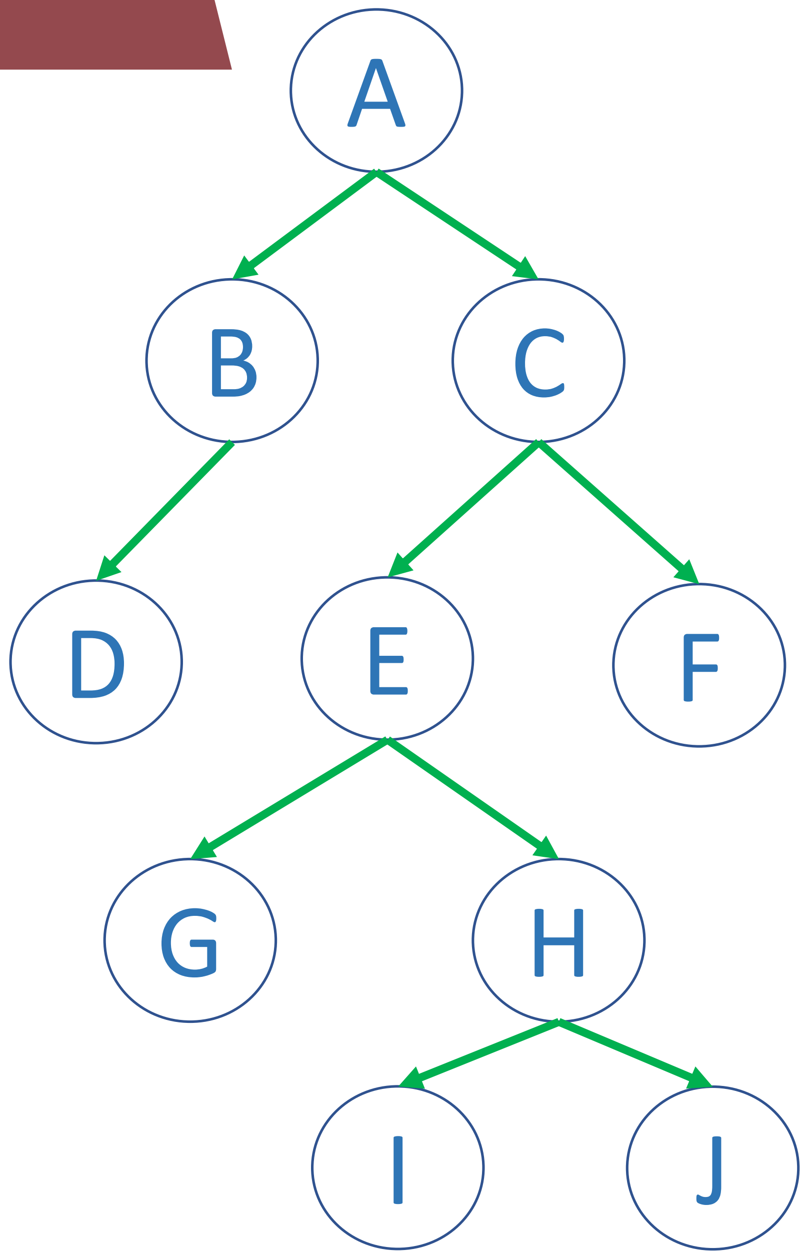
C

Practice

Mission

試著寫出右邊樹的以下性質：

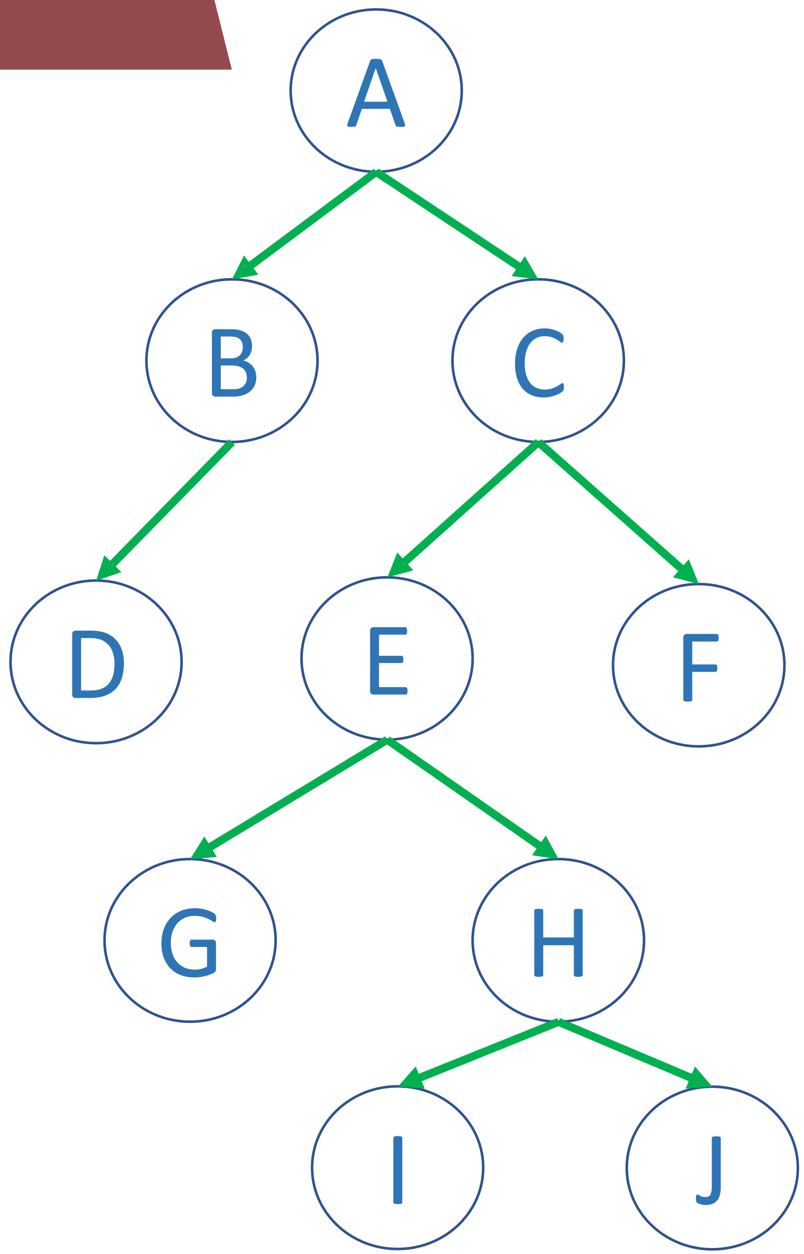
1. Height of tree
2. Leaf nodes (external nodes)
3. Non-leaf nodes (internal nodes)
4. Depth of node#E
5. Height of node#B
6. Sibling nodes of node#G



Practice

Mission

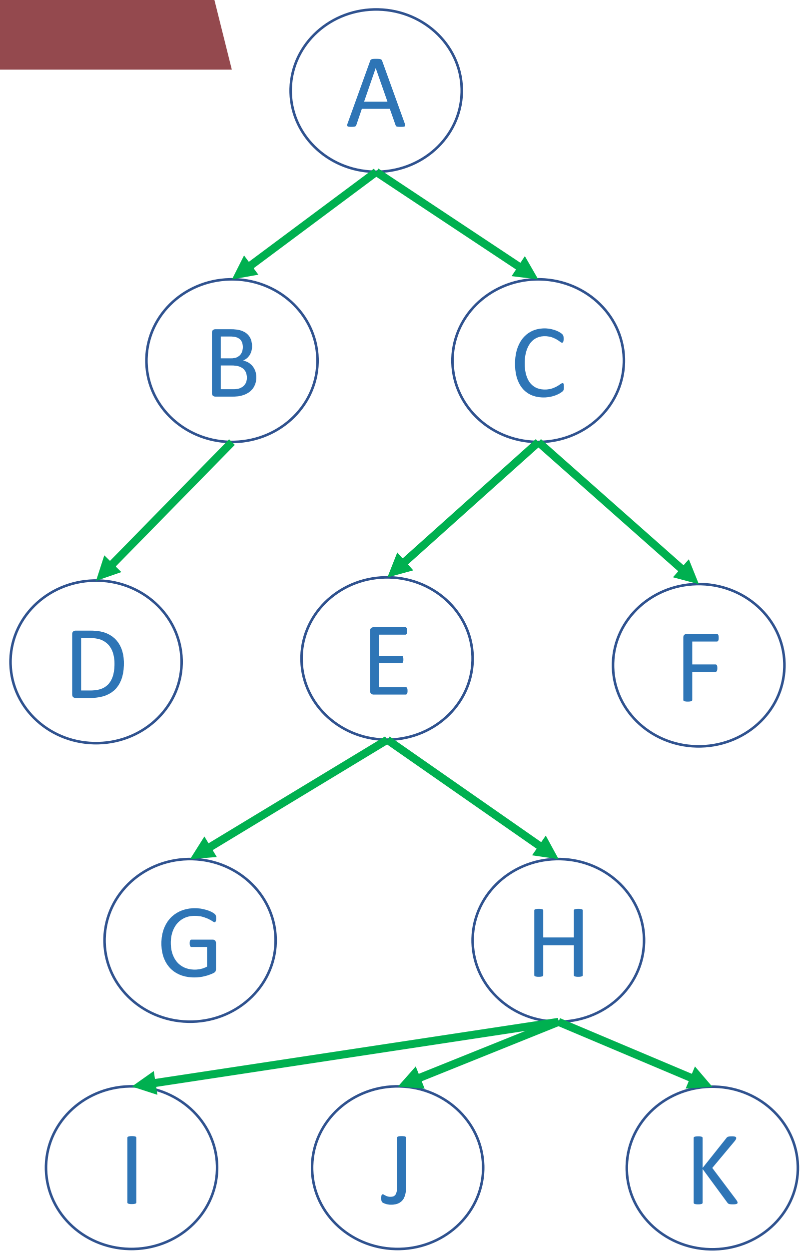
1. Height of tree
   - 4
2. Leaf nodes (external nodes)
   - DFGIJ
3. Non-leaf nodes (internal nodes)
   - ABCEH
4. Depth of node#E
   - 2
5. Height of node#B
   - 1
6. Sibling nodes of node#G
   - H



Practice

Mission

試著利用括號法描述左邊的樹

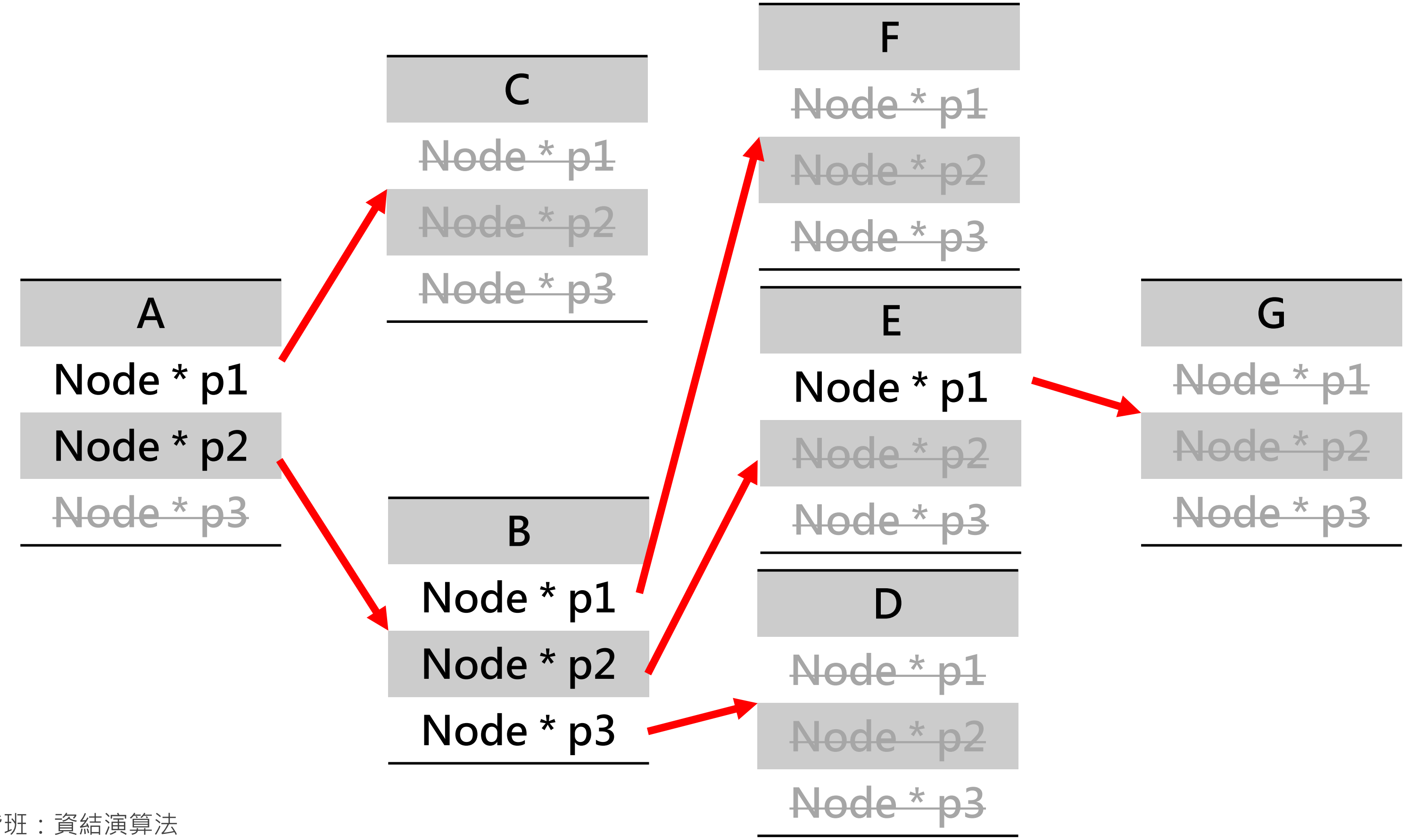


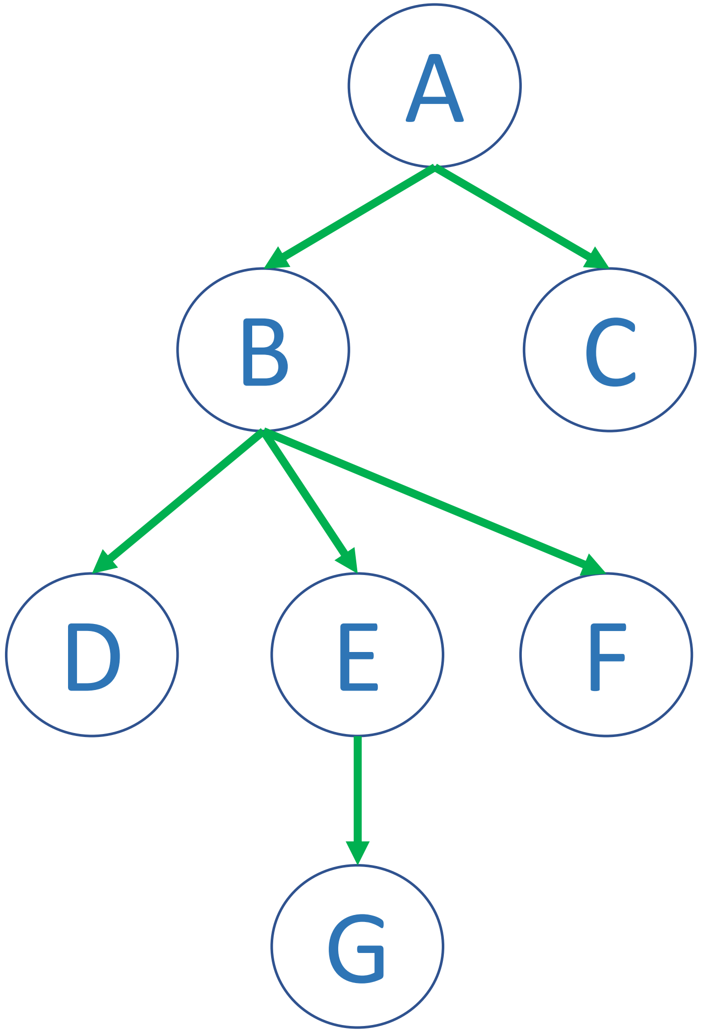
二元樹的簡介&定義

樹的紀錄法

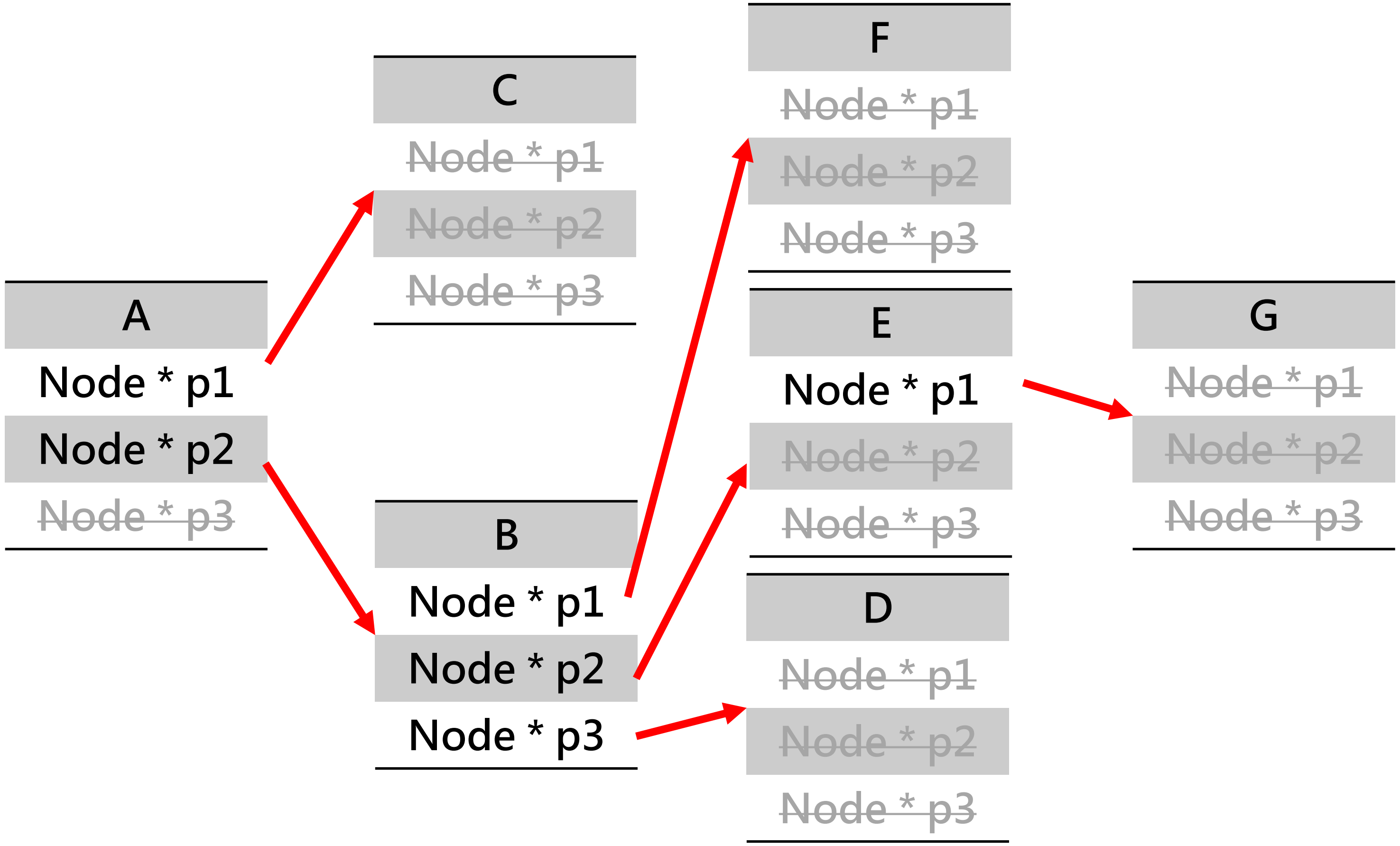
鏈結串列 (Linked List)

- ➢ 透過指標連接
- ➢ 每個節點需要有 Degree of Tree 個指標





樹的紀錄法



極度<u>浪費Linked List的空間</u>

節點至少要 Degree of Tree 個指標

n：節點數目 、k： Degree of Tree

總共能容納指標數目：$n \times k$

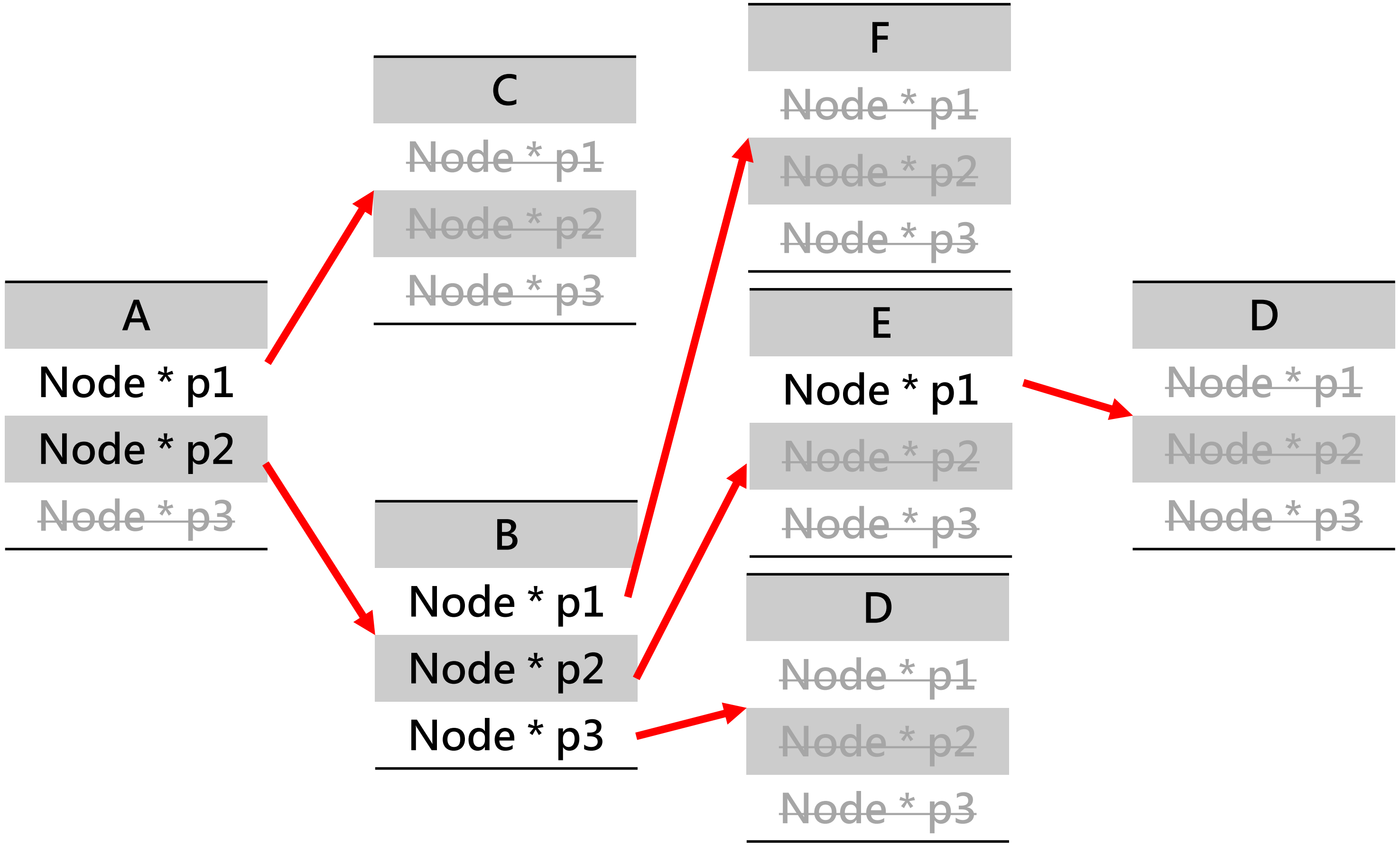
實際有用到的指標數目：$n-1$

空間浪費率：

$$\frac{n\times k-(n-1)}{n\times k}=\frac{n\times(k-1)+1}{n\times k}\sim\frac{k-1}{k}$$

<u>空間浪費率跟連接方式無關</u>

樹的紀錄法



空間浪費率：

$$\frac{n\times k-(n-1)}{n\times k}=\frac{n\times(k-1)+1}{n\times k}\sim\frac{k-1}{k}$$

$k=1$：0.0%

$k=2$：50%

$k=3$：67%

$k=4$：75%

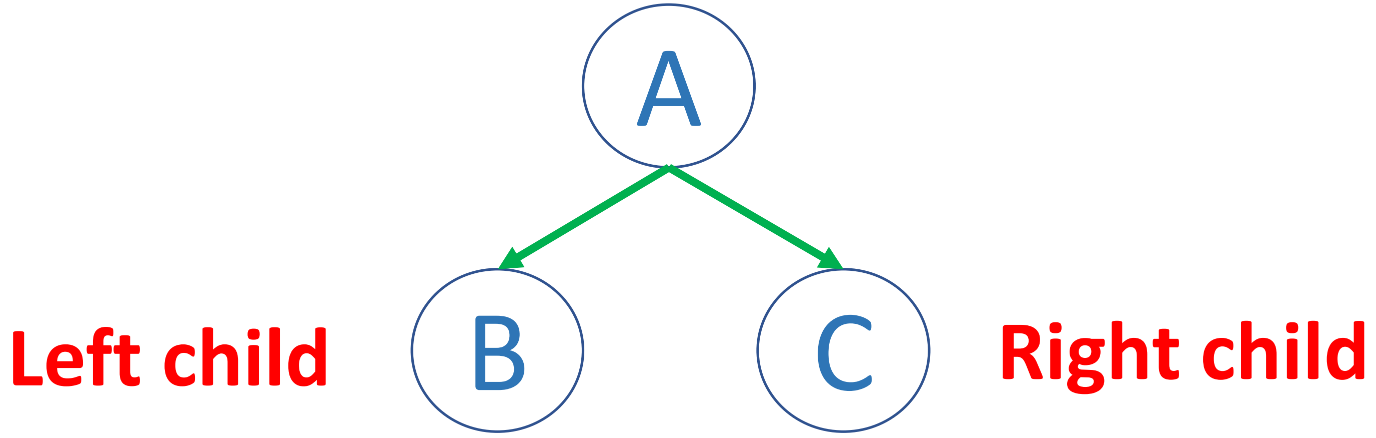
$k=1$ 時等同鏈結串列

$k=2$ 時最節省空間

二元樹最常用！

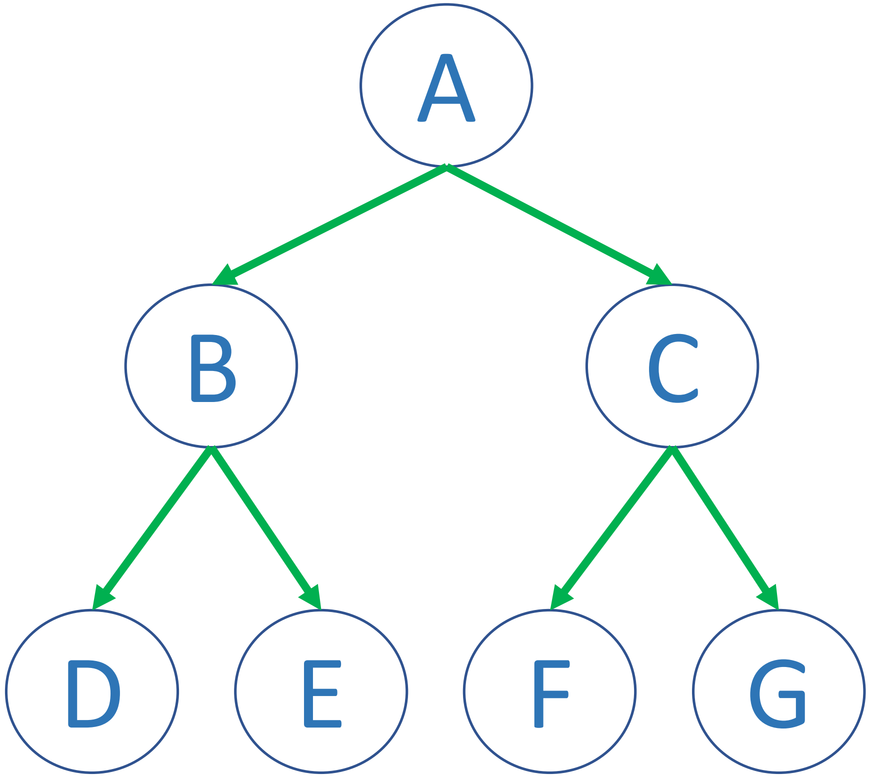
二元樹定義

- 限制 degree of tree 為 2
  - ➢ 每個父節點最多只能連到另兩個子節點
  - ➢ 這兩個子節點分別稱為 Left child 和 Right child



二元樹特性

- 在第 i 個階層上的節點個數：2^{i-1}
- 最大階層 h 的二元樹，節點最多為：$2^h - 1$
  - ➢ $1 + 2 + \cdots 2^{h-1} = 1 \times \frac{1-2^h}{1-2} = 2^h - 1$
  - ➢ 解題時注意樹高的定義有兩種版本！
- 節點數 V，邊長數 E = V-1



二元樹特性

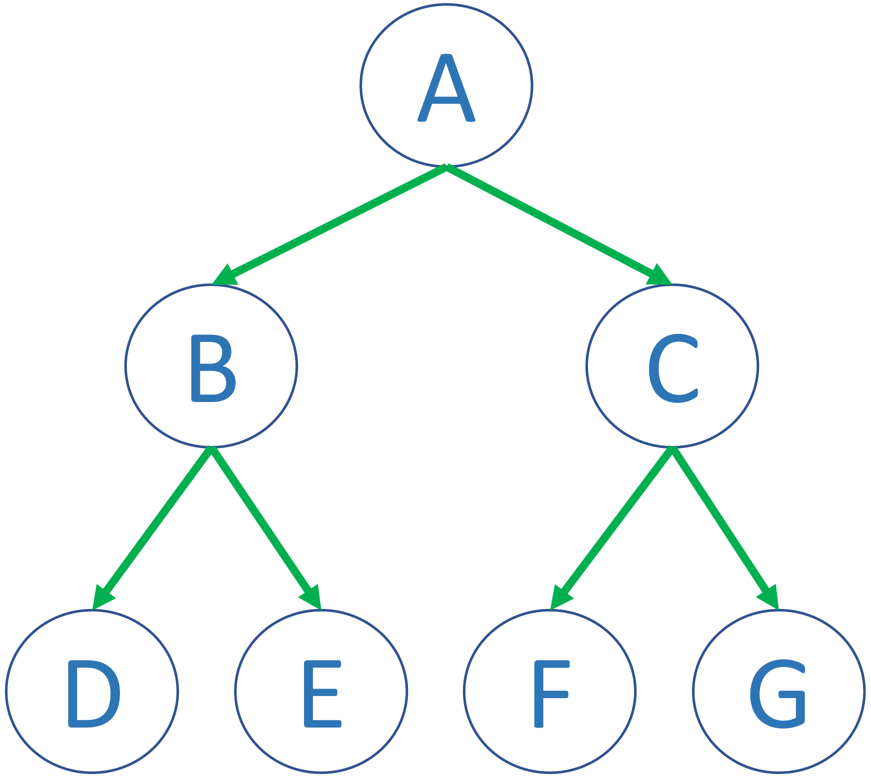
Leaf node 個數 = 分岔度 2 的 node 數 + 1

Proof:

- $N = n_0 + n_1 + n_2 = E + 1$
- $E = n_1 + 2 \times n_2$
- $n_0 + n_1 + n_2 = n_1 + 2 \times n_2 + 1$
- $n_0 = n_2 + 1$

✓ n_k：分岔度k的節點個數

✓ $k = 0$ 即為 Leaf node

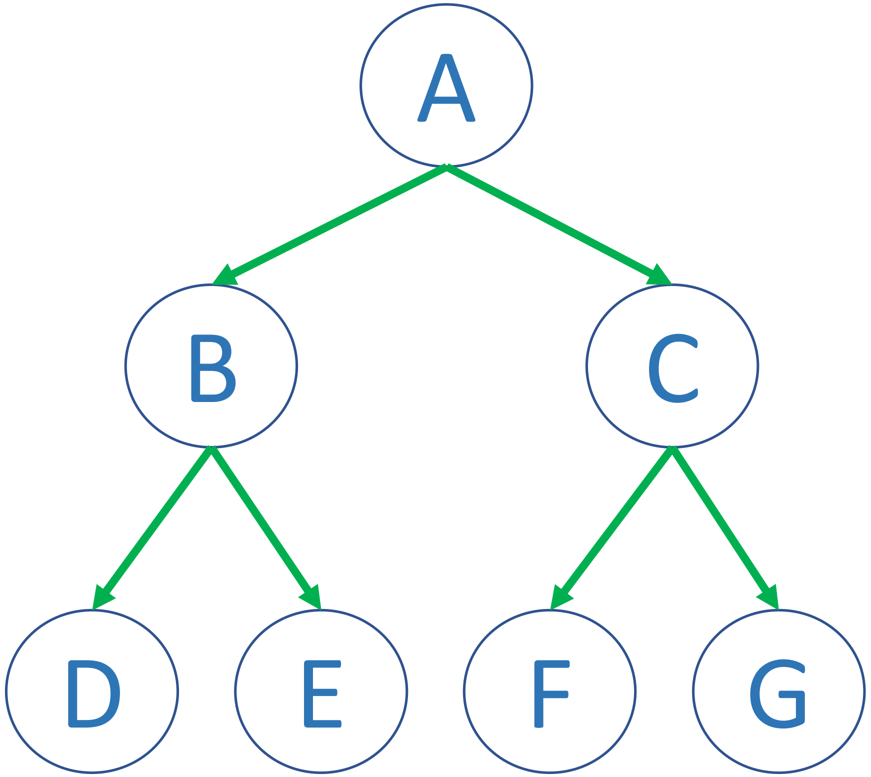


二元樹特性

n 個節點可組成 K 種二元樹？

$$K = \frac{1}{n+1} C_n^{2n}$$

可用數學歸納法證，但很麻煩 QQ



二元樹特性

- 真的會考！！！**注意高度的定義可能不同**
- (台北市國中教師聯合甄選電腦專業科目)

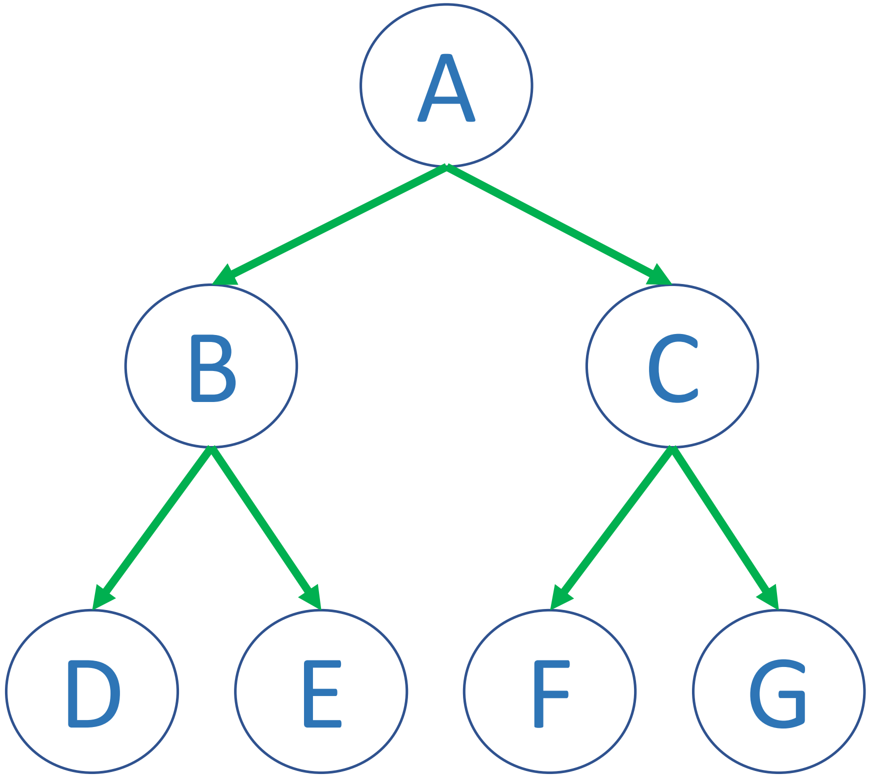
  高度為N的二元樹(根節點的高度為1)，最多能有幾個節點？

  (A) $N!$

  (B) $log_2 N + N$

  (C) $\frac{N(N-1)}{2}$

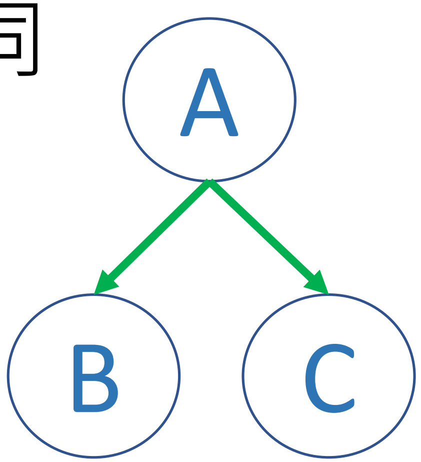
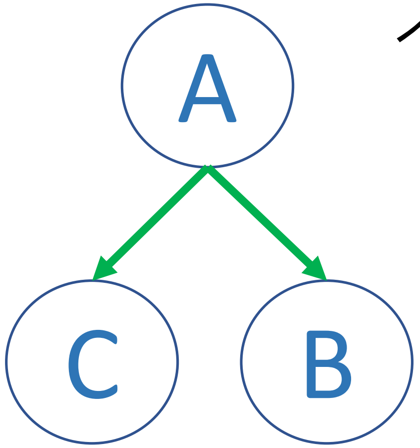
  (D)$2^N - 1$



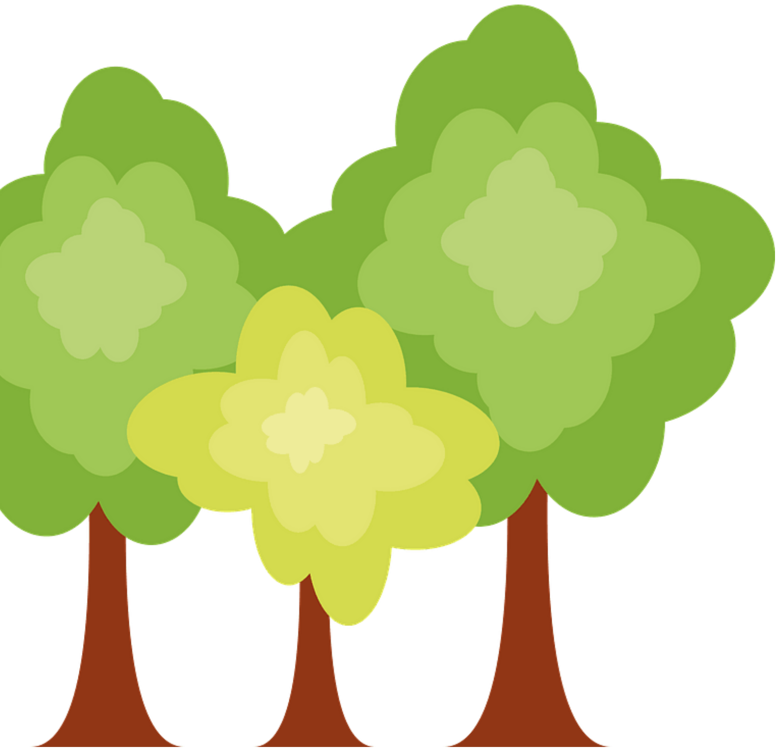
二元樹應用

- 新增、刪除：Binary Search Tree(BST)
  - C++ STL 中的 map 與 set：紅黑樹
- 檔案壓縮：Huffman Coding
- 機器學習：Decision Tree
- 儲存 Router-tables：Binary Tries
- 安排作業系統中的處理佇列Heaps：Priority queues
- 二元樹無所不在

二元樹與一般的樹

樹(Tree)	二元樹(Binary Tree)
不可以是空集合	可以是空集合
$0 \leq$分岔度	$0 \leq$分岔度≤ 2
子樹無次序之分	左右子樹有次序之分
相同 	不同 

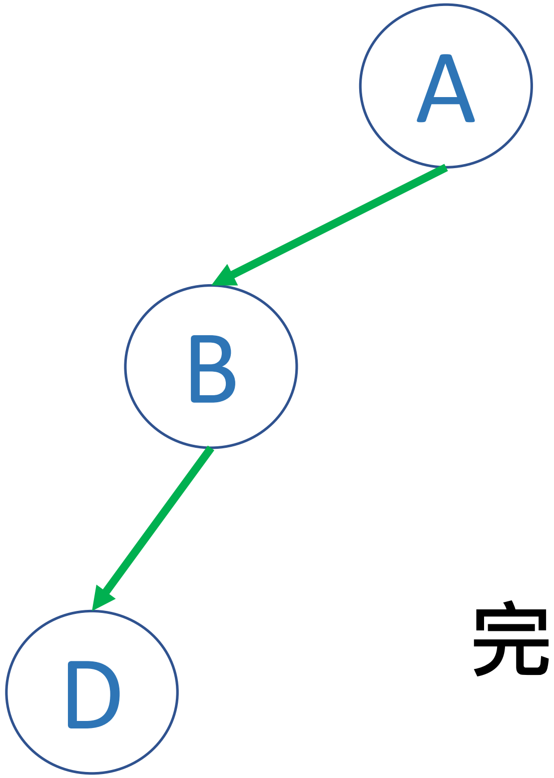
二元樹分類



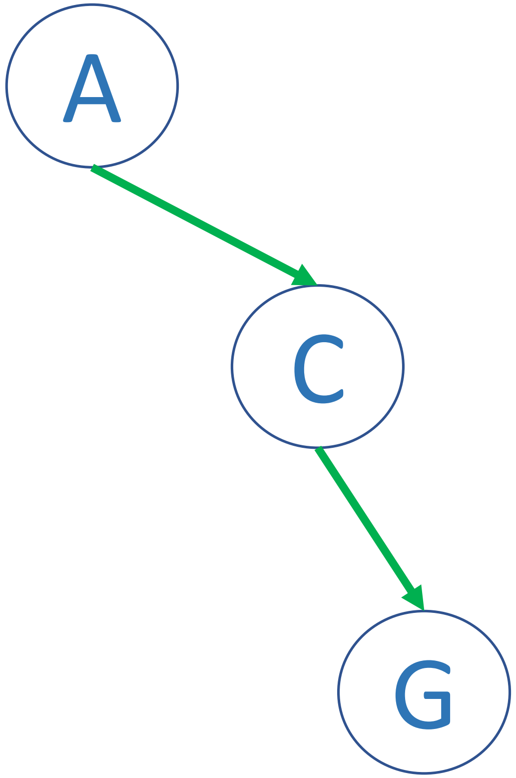
- 斜曲二元樹 (Skewed binary tree)
  - ➢ 左斜曲 (Left-skewed) 二元樹
  - ➢ 右斜曲 (Right-skewed) 二元樹
- 嚴格二元樹 (Strictly binary tree)
- 完滿二元樹 (Full Binary Tree)
- 完整二元樹 (Complete Binary Tree)

斜曲二元樹

左斜曲 (Left-skewed)

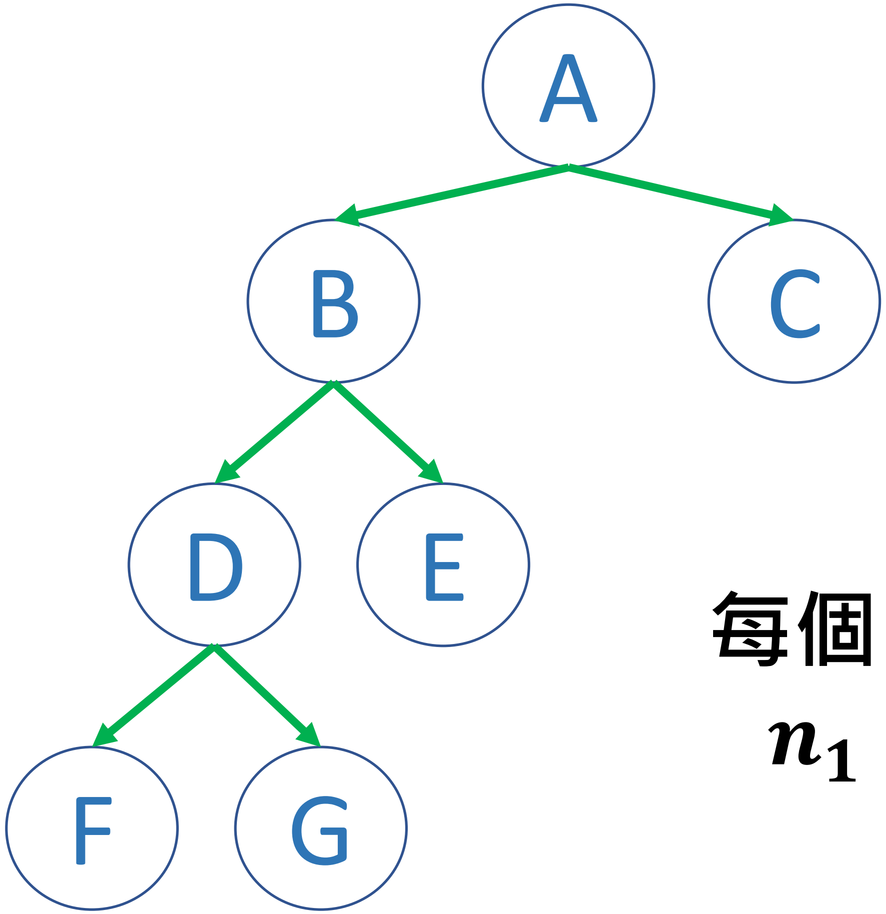


右斜曲 (Right-skewed)



完全沒有某一側的子節點
相當於 linked list !

嚴格二元樹

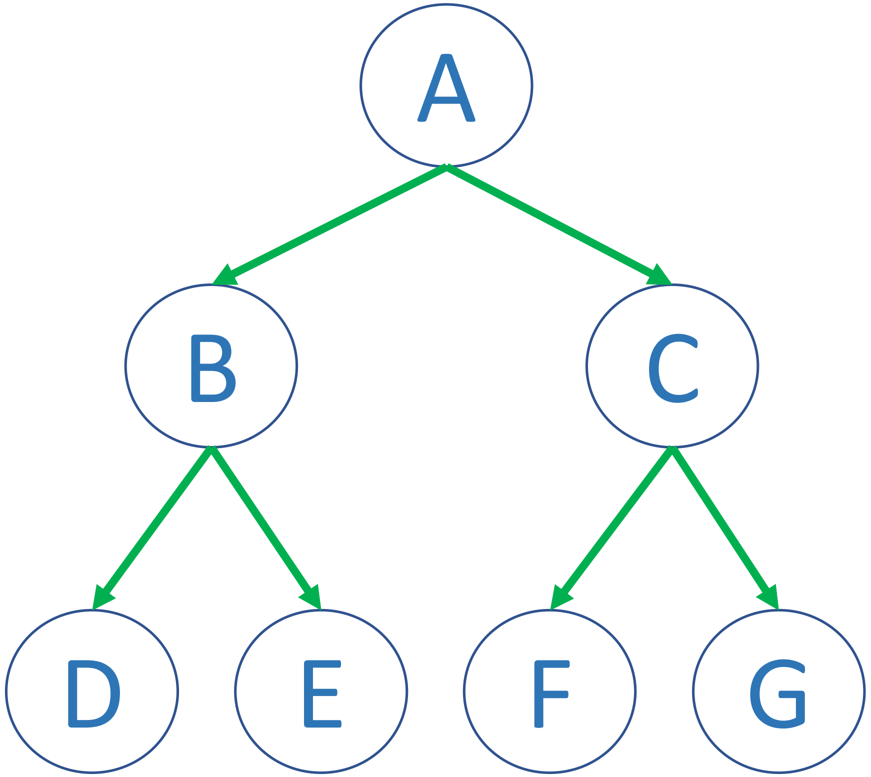


每個 non-leaf node 都有兩個子節點

$n_1 = 0$ ：分岔度 1 的節點個數為0

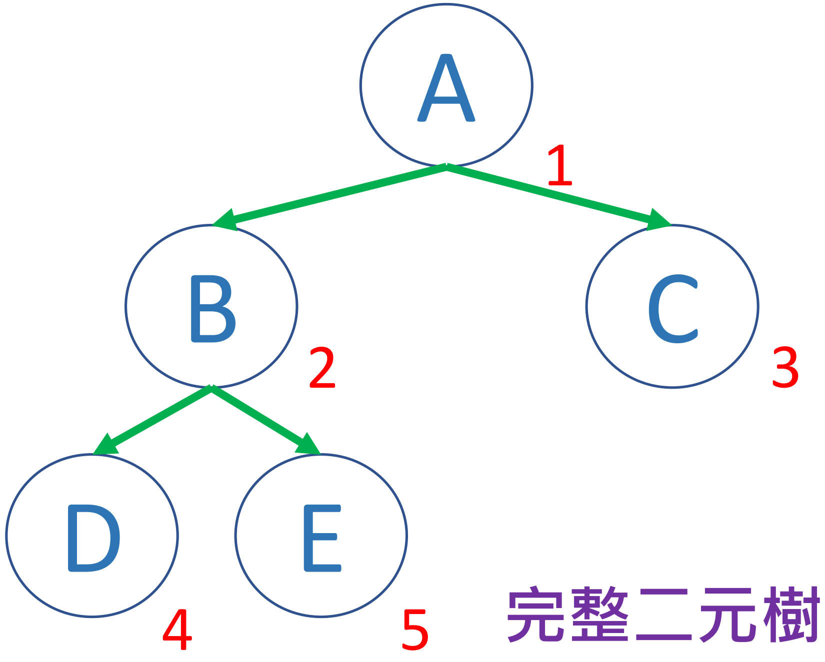
完滿二元樹

- 每個 non-leaf node 都有兩個 child node
- 所有 leaf node 的 level 都相同
- 並且，每個node與其child有以下關係：
- 最大階層 $\boldsymbol{h}$ 的完滿二元樹，節點為：$\mathbf{2}^{\boldsymbol{h}}-\mathbf{1}$個
  - ➢ 最大階層 3 ，節點為：$\mathbf{2}^{3}-\mathbf{1}=7$ 個

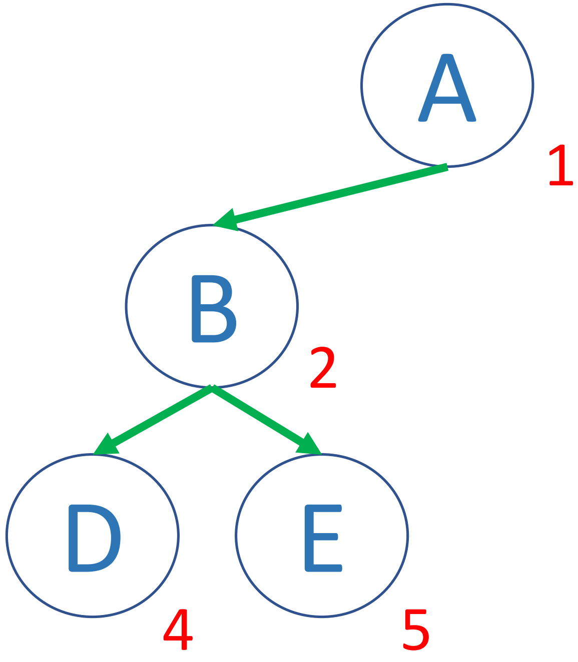


完整二元樹

- 完整二元樹 Complete Binary Tree
  - ➢ 節點依照次序排列是連續、沒有空缺
  - ➢ 次序由上至下、左至右



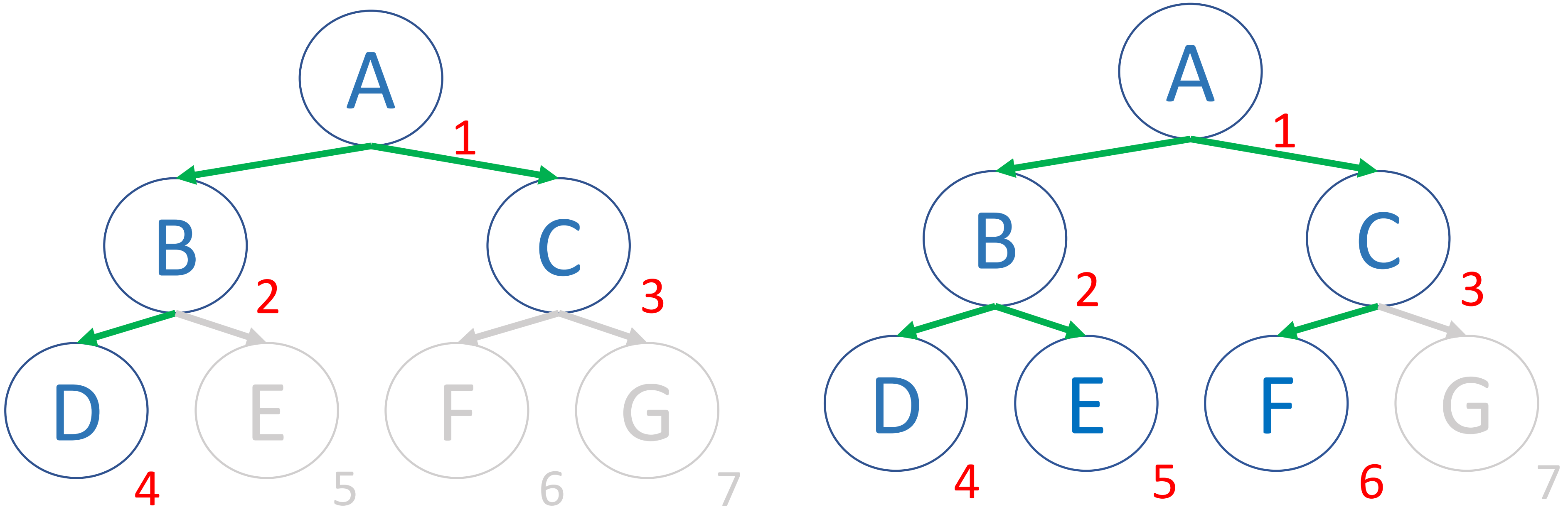
完整二元樹



非完整二元樹

完整二元樹

- 完整二元樹 Complete Binary Tree定義
  - $2^{\boldsymbol{h-1}} - \mathbf{1} <$ 節點個數 $< 2^{\boldsymbol{h}} - \mathbf{1}$
  - **節點編號是連續的**



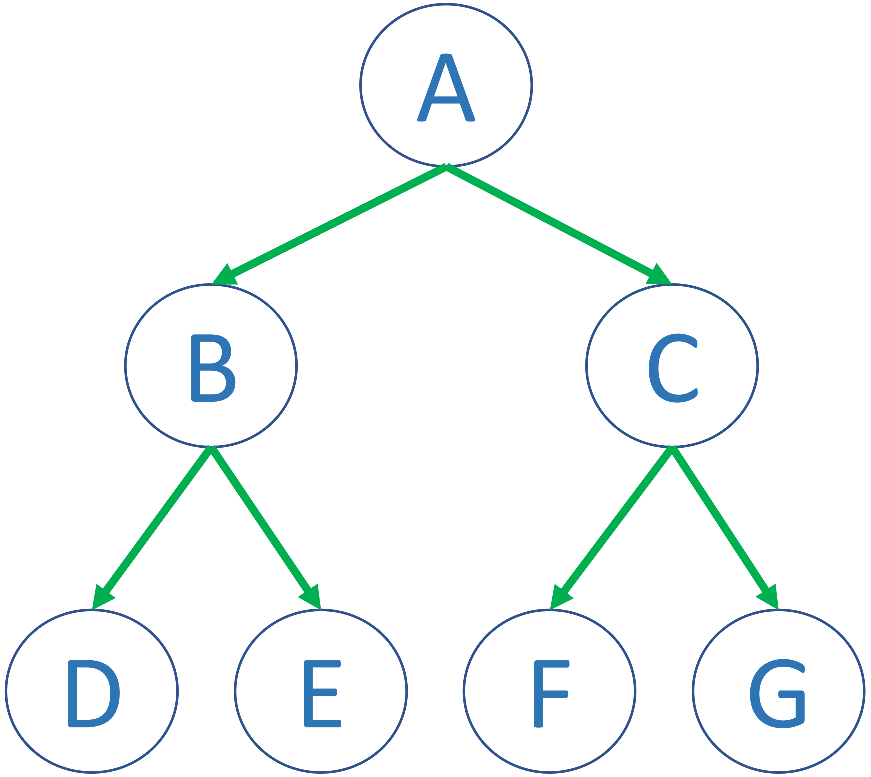
二元樹表示法

- 陣列
  - 適合完滿二元樹、完整二元樹
  - 不適合斜曲二元樹
  - 容易尋訪 (Traversal)
- 鏈結串列
  - 適合斜曲二元樹
  - 會有浪費指標空間的問題 (50%)

陣列表示法

- 若二元樹的最大階層為 h
- 節點最多為：$2^h - 1$
- 準備長度為 $2^h - 1$ 的陣列，依序把所有資料放入

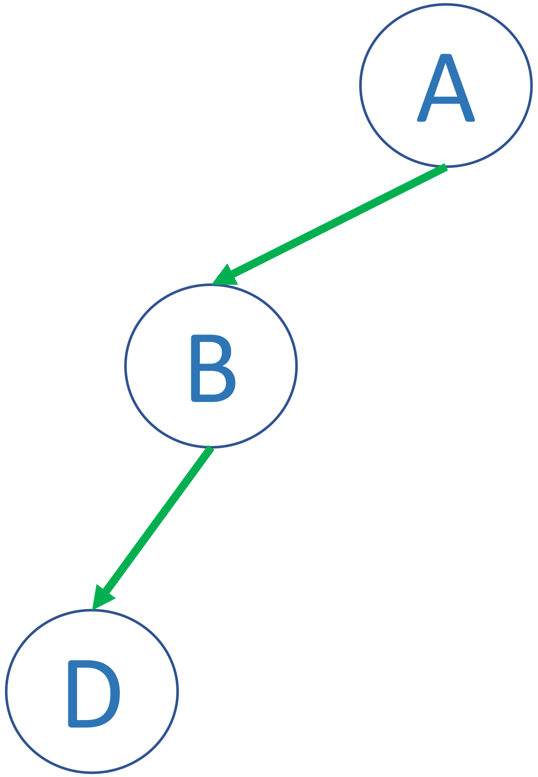
A	B	C	D	E	F	G



陣列表示法

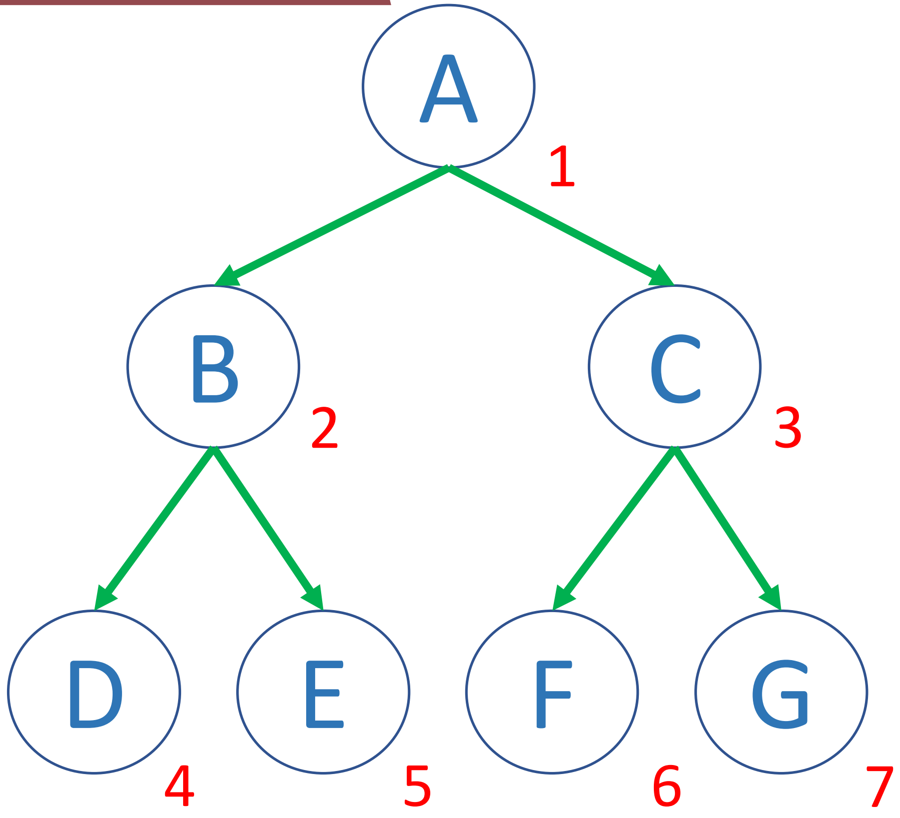
- 若二元樹的最大階層為 h
- 節點最多為：$2^h - 1$
- 準備長度為 $2^h - 1$ 的陣列，依序把所有資料放入

A	B	-	D	-	-	-



陣列表示法

- 第 k 個節點
  - ✓ Left child：索引值 $2k-1$
  - ✓ Right child：索引值 $2k$
  - ✓ Parent：索引值 $\left\lfloor\frac{k-2}{2}\right\rfloor$



A	B	C	D	E	F	G

陣列表示法

- 優點
  - ✓ 同一階層內的左、右節點容易取得
  - ✓ 完滿二元樹時不浪費任何空間
- 缺點
  - ✓ 新增與刪除資料較為困難
  - ✓ 斜曲二元樹時浪費大量空間

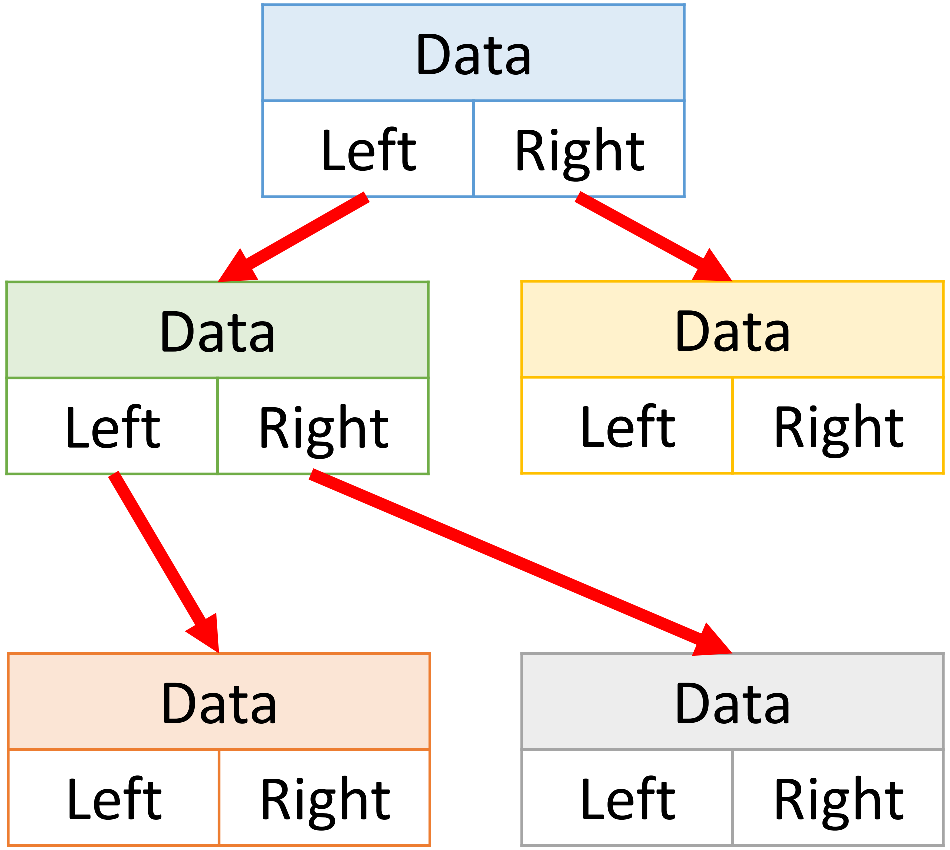
□ 空間利用率：$\frac{h}{2^h-1}$

A	B	C	D	E	F	G

A	B	-	D	-	-	-

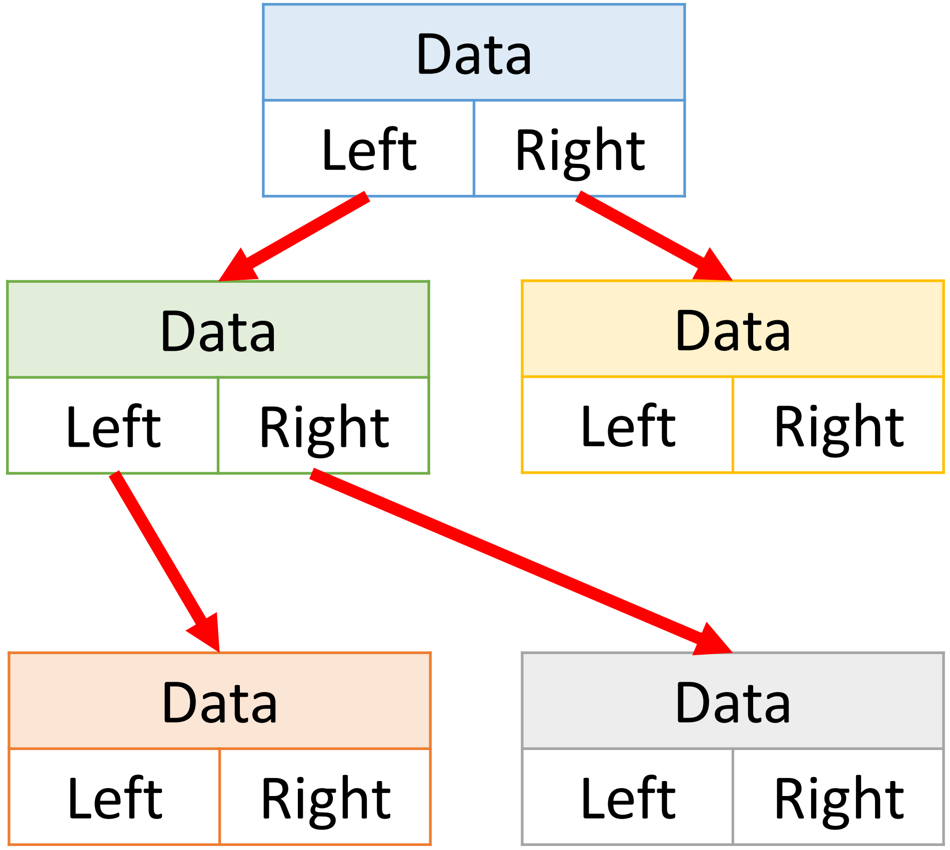
鏈結串列表示法

- 節點內容
  - ✓ 資料內容
  - ✓ 左節點 left child node 指標
  - ✓ 右節點 right child node 指標
  - ✓ (父節點的指標)



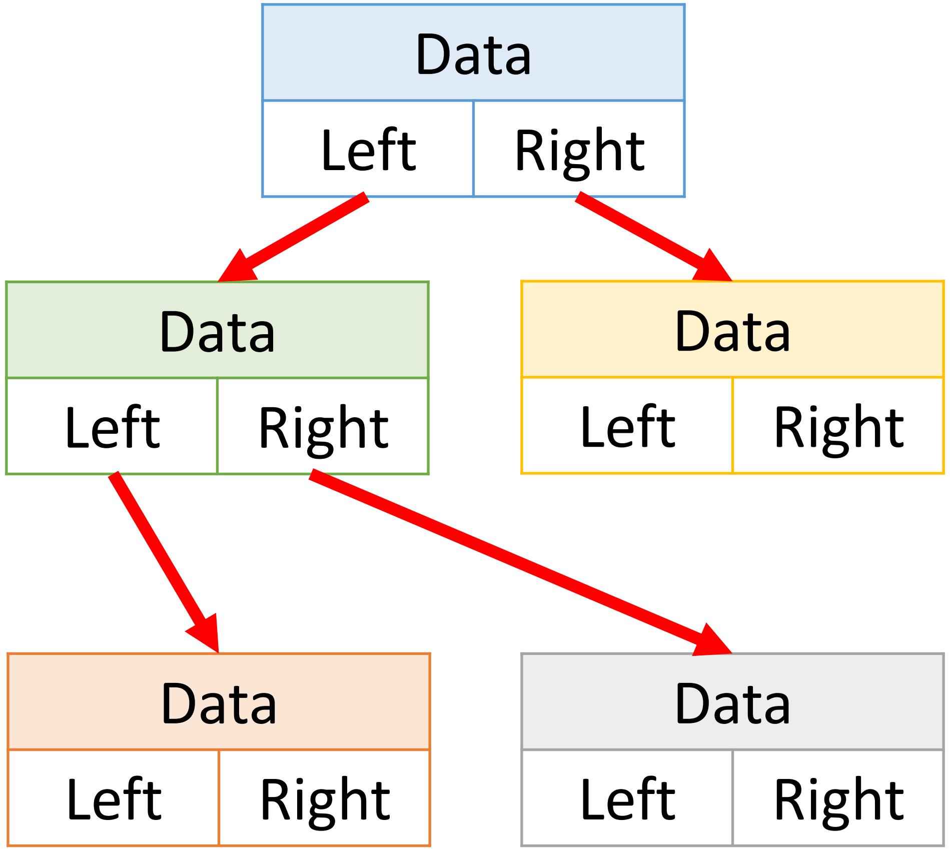
鏈結串列表示法

- ➢ 優點
  - ✓ 新增與刪除資料容易
  - ✓ 斜曲二元樹較陣列省空間
- ➢ 缺點
  - ✓ 父節點不容易到達
    - ◻ 所以有時會加入父節點指標
  - ✓ 浪費一半節點空間
    - ◻ 空間浪費率：$\frac{k-1}{k}$；$k=2$



鏈結串列表示法

- 有 n 個節點的二元樹
  - ✓ 指標數目共 $2n$ 個
  - ✓ 有用到的指標： $n-1$ 個
  - ✓ 無用的指標： $2n-(n-1)=n+1$
  - ✓ 之前的空間浪費率公式：
    - $\frac{n\times k-(n-1)}{n\times k}=\frac{n\times 2-(n-1)}{n\times 2}=\frac{n+1}{2n}$

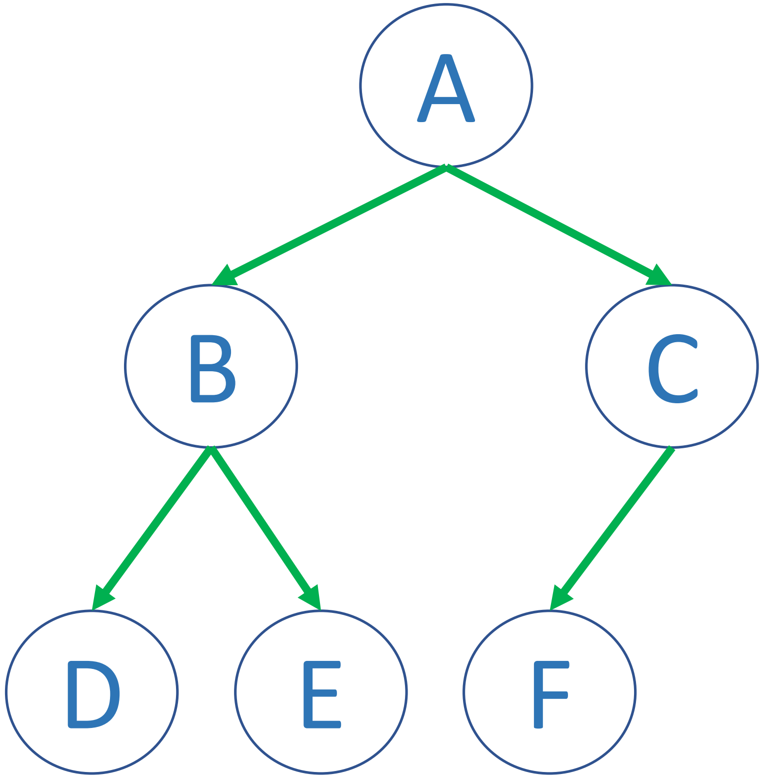


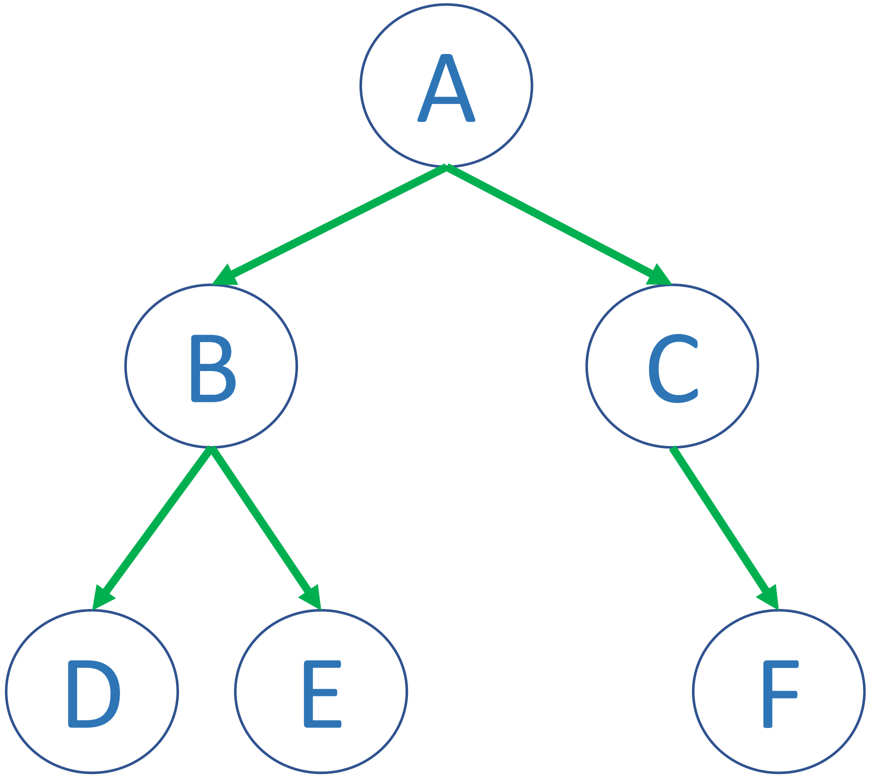
Practice

Mission

下圖左右中，哪個是完整二元樹？

試著利用陣列法表示看看！

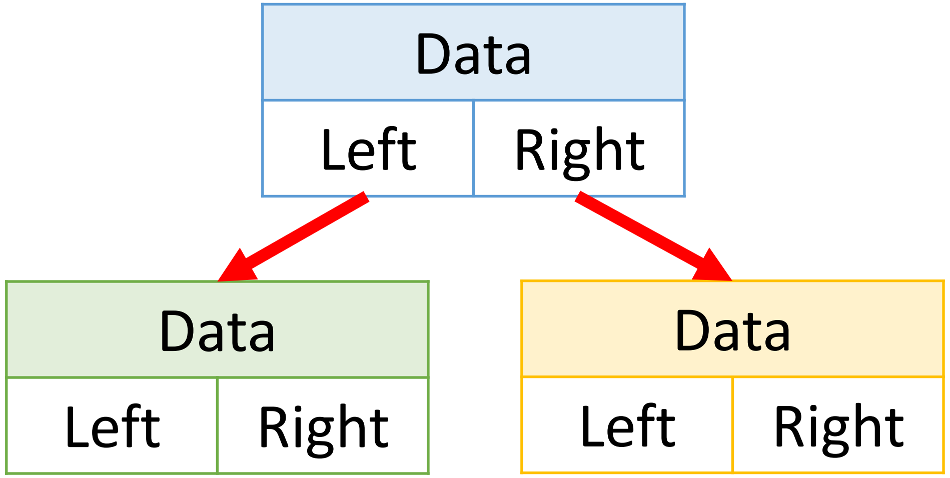




建立二元樹

二元樹建立

- 節點結構
  - ➢ 資料內容
    - ✓ 編號 (方便檢索)
    - ✓ 資料內容
  - ➢ 左節點
  - ➢ 右節點
  - ➢ 父節點 (方便返回)




```
template<typename T>
struct Node{
   int index;
   T data;
   Node<T> *left;
   Node<T> *right;
};
```

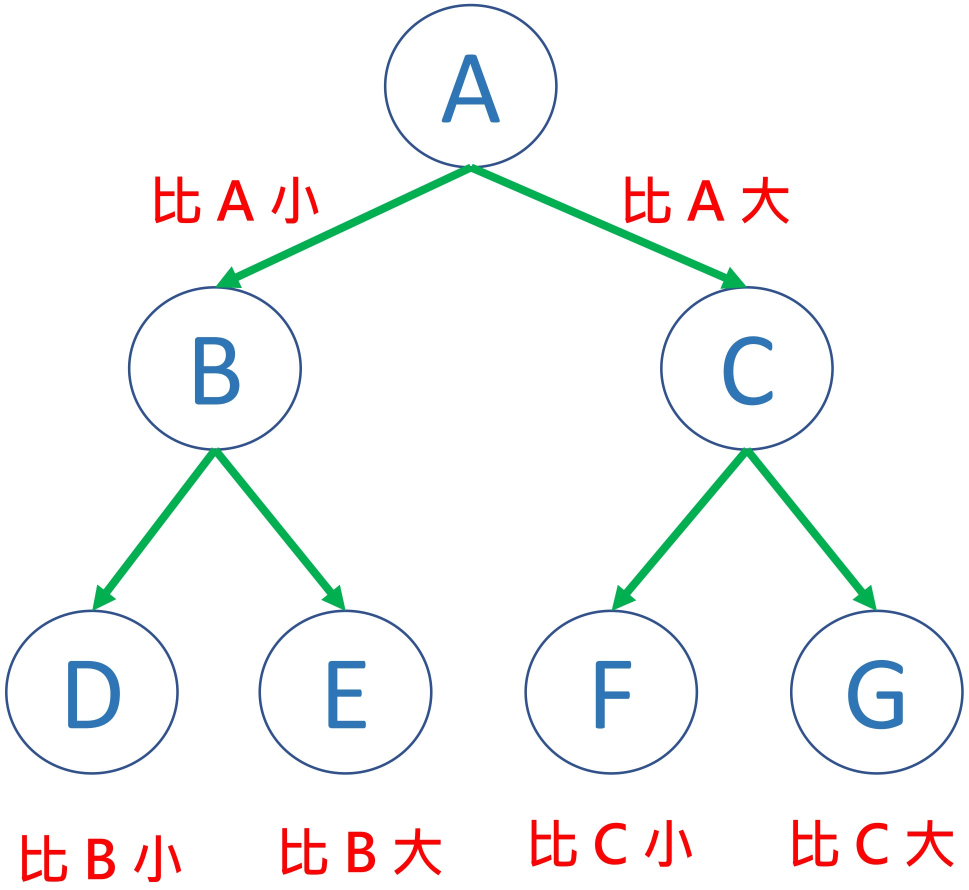
二元樹建立

- 二元樹類別
  - 根節點
  - 印出特定節點
  - 新增資料
  - 搜尋資料
  - 刪除資料
  - 尋訪
    - 前序
    - 中序
    - 後序

```
template<typename T>
class Binary_Tree{
    public:
        Node<T>* root;
        Binary_Tree();
        void Print(Node<T>);
        bool Insert(int,T);
        Node<T>* Search(int);
        bool Delete(int);
};
```

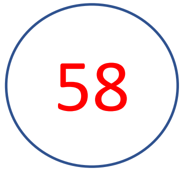
二元樹的新增(Insert)

- 為了檢索/搜尋需求必須加以排序
  - ➢ 另外制定規則很麻煩
  - ➢ 利用額外、不重複的編號來確認方向
    - ✓ 比 Parent node 編號小→左子樹
    - ✓ 比 Parent node 編號大→右子樹



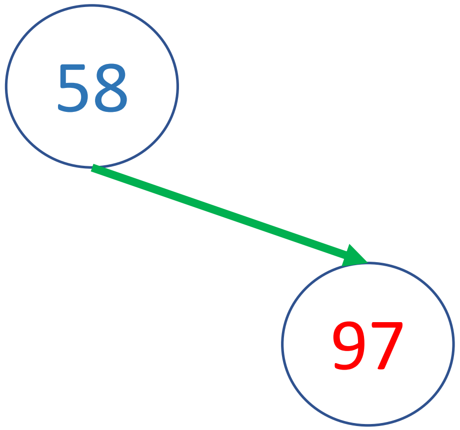
二元樹的新增(Insert)

插入次序：
58



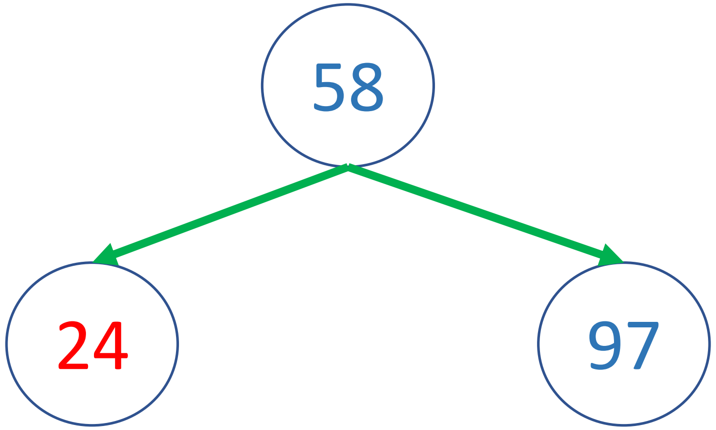
二元樹的新增(Insert)

插入次序：
58
97



二元樹的新增(Insert)

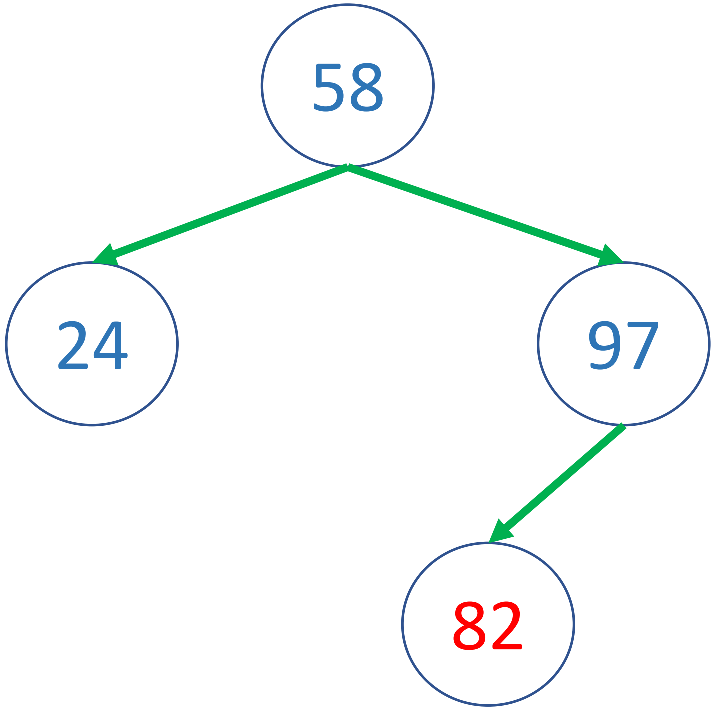
插入次序：
58
97
24



二元樹的新增(Insert)

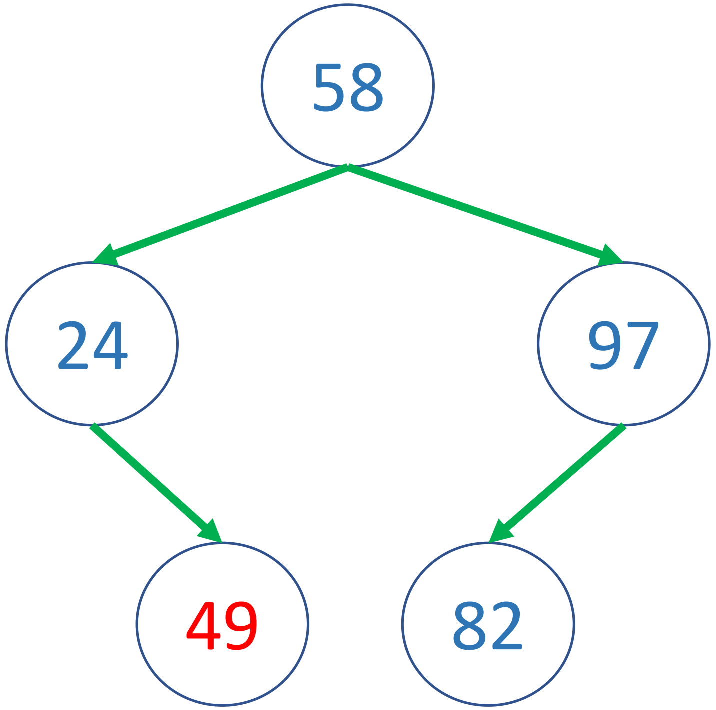
插入次序：

58
97
24
82



二元樹的新增(Insert)

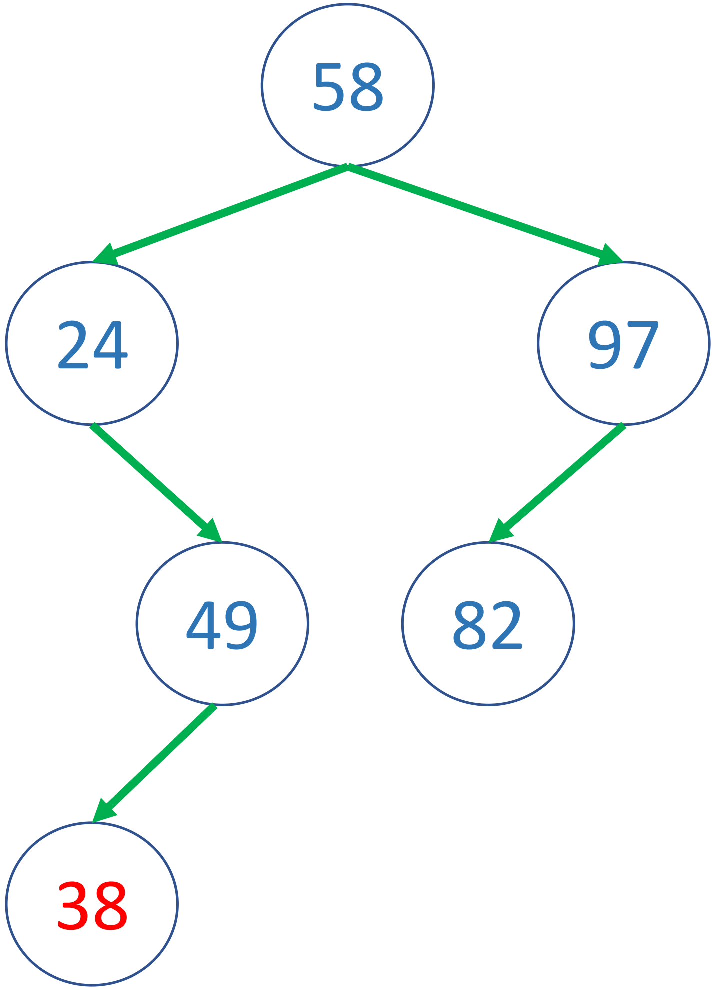
插入次序：
58
97
24
82
49



二元樹的新增(Insert)

插入次序：

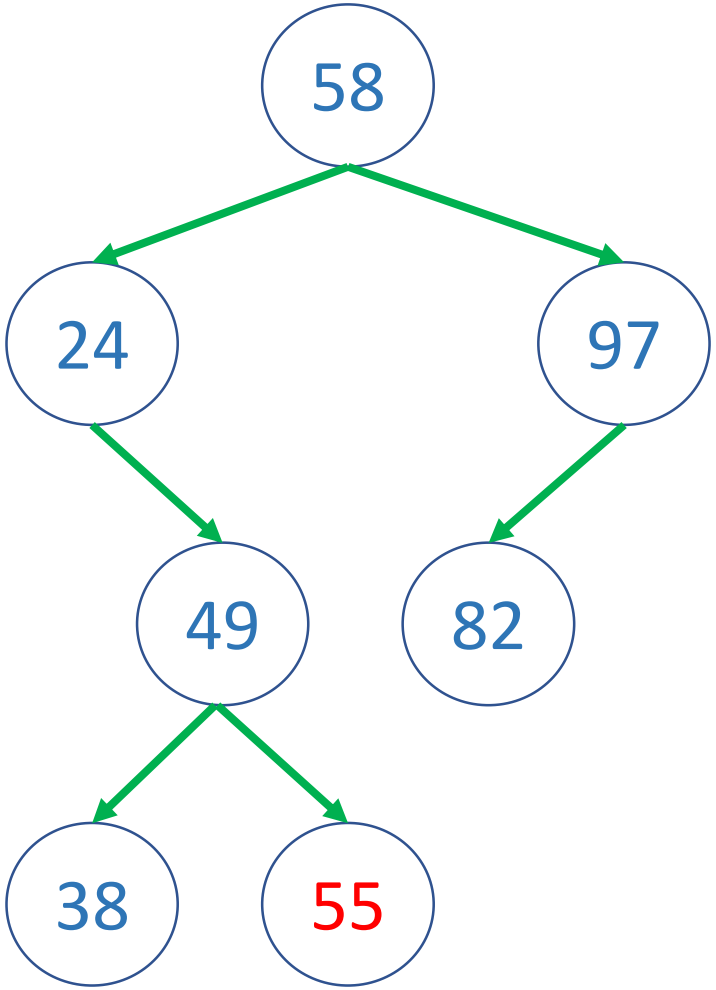
58
97
24
82
49
38



二元樹的新增(Insert)

插入次序：

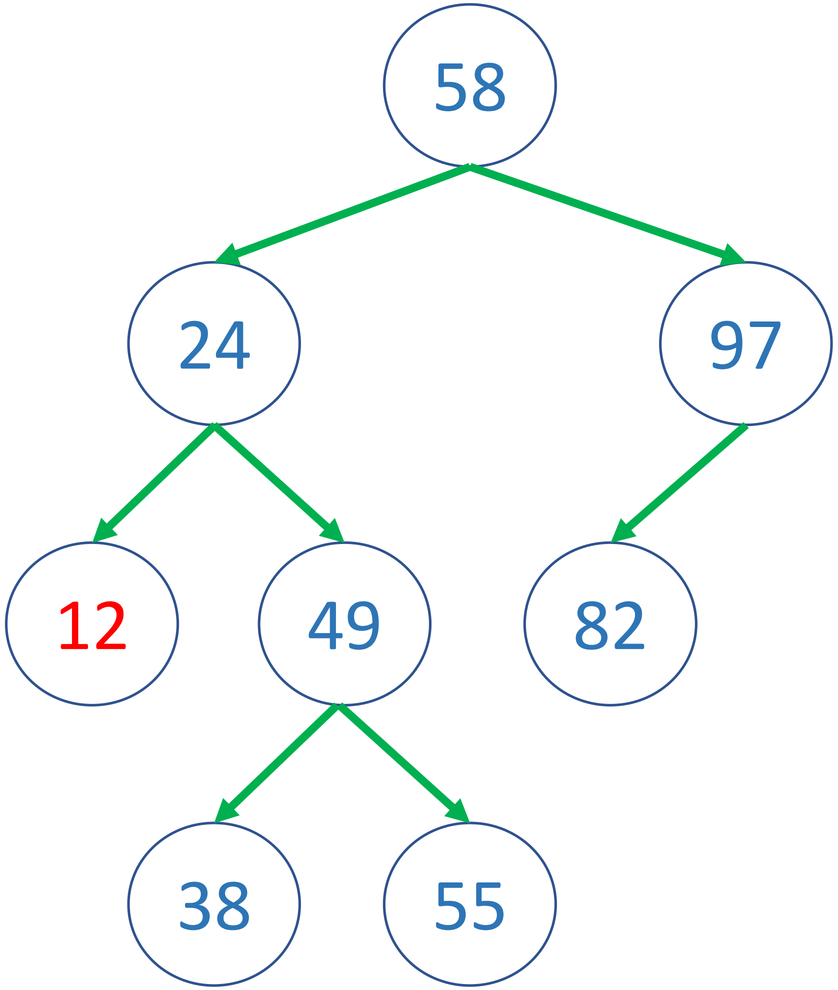
58
97
24
82
49
38
55



二元樹的新增(Insert)

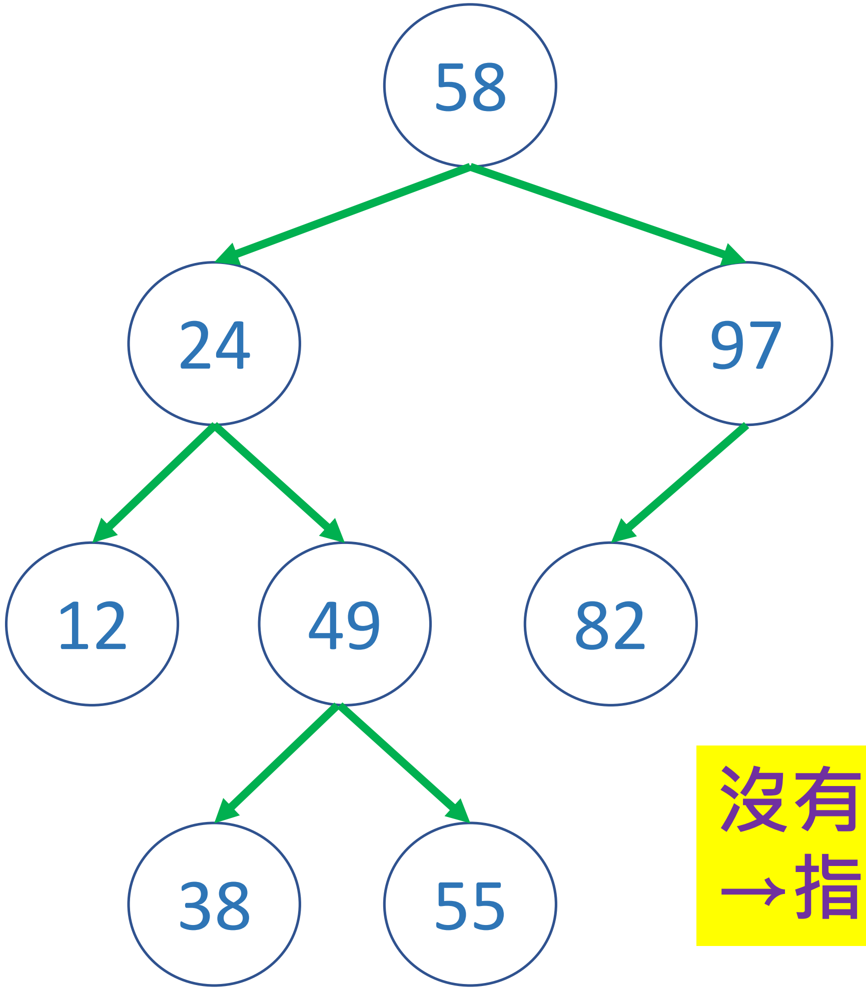
插入次序：

58
97
24
82
49
38
55
12



二元樹的新增(Insert)

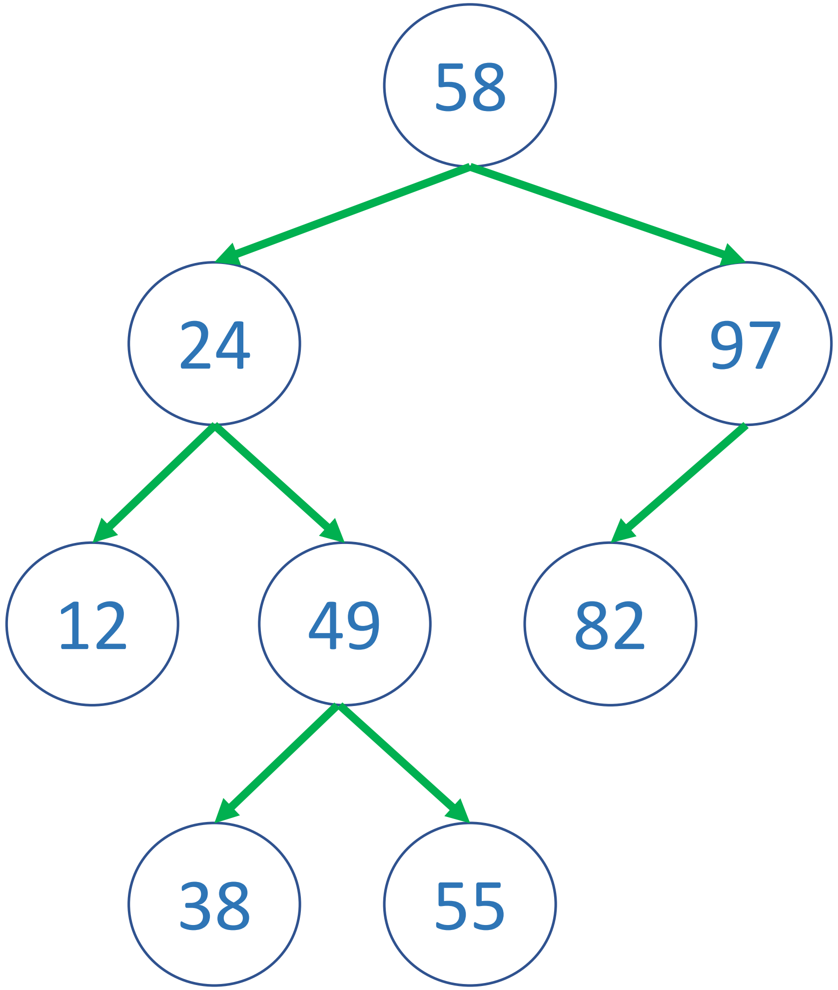
插入次序：
58
97
24
82
49
38
55
12



沒有child node的話
→指向空指標 nullptr

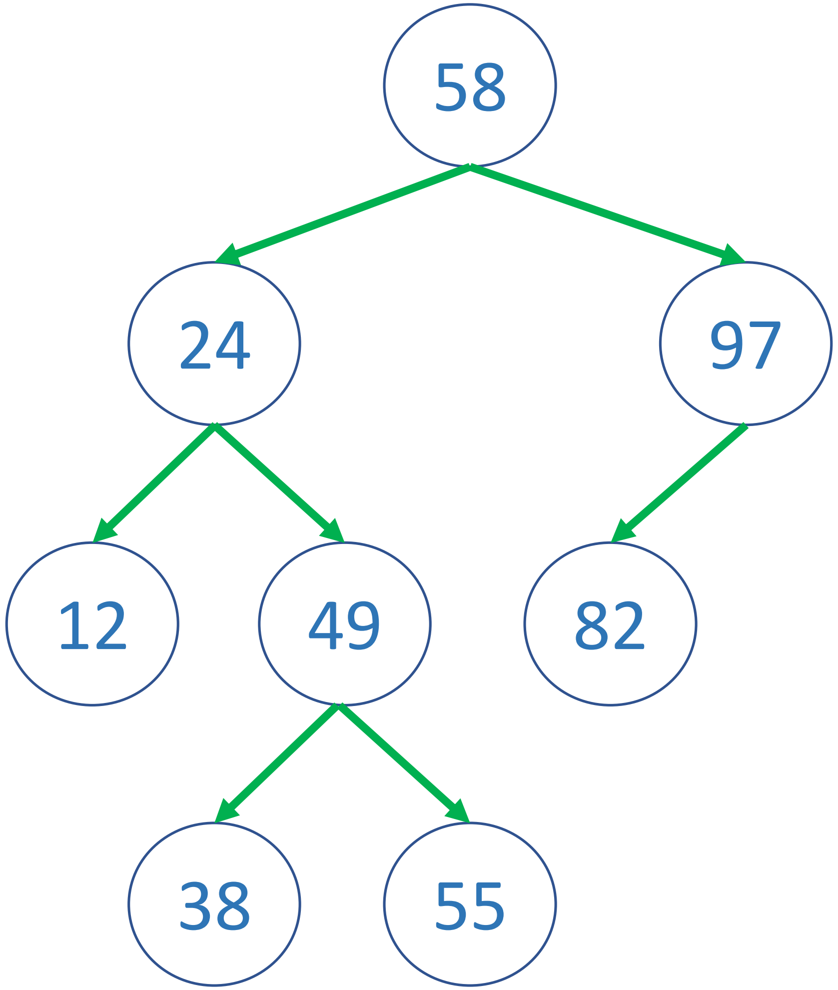
二元樹的搜尋(Search)

- 當要找的編號跟現處節點一致
  - ➢ 結束
- 當要找的編號跟現處節點不一致
  - ➢ 尋找的編號比節點編號小
    - ✓ 往左節點移動
  - ➢ 尋找的編號比節點編號大
    - ✓ 往右節點移動
- 當現在的節點為 leaf node
  - ➢ 結束，回傳空指標



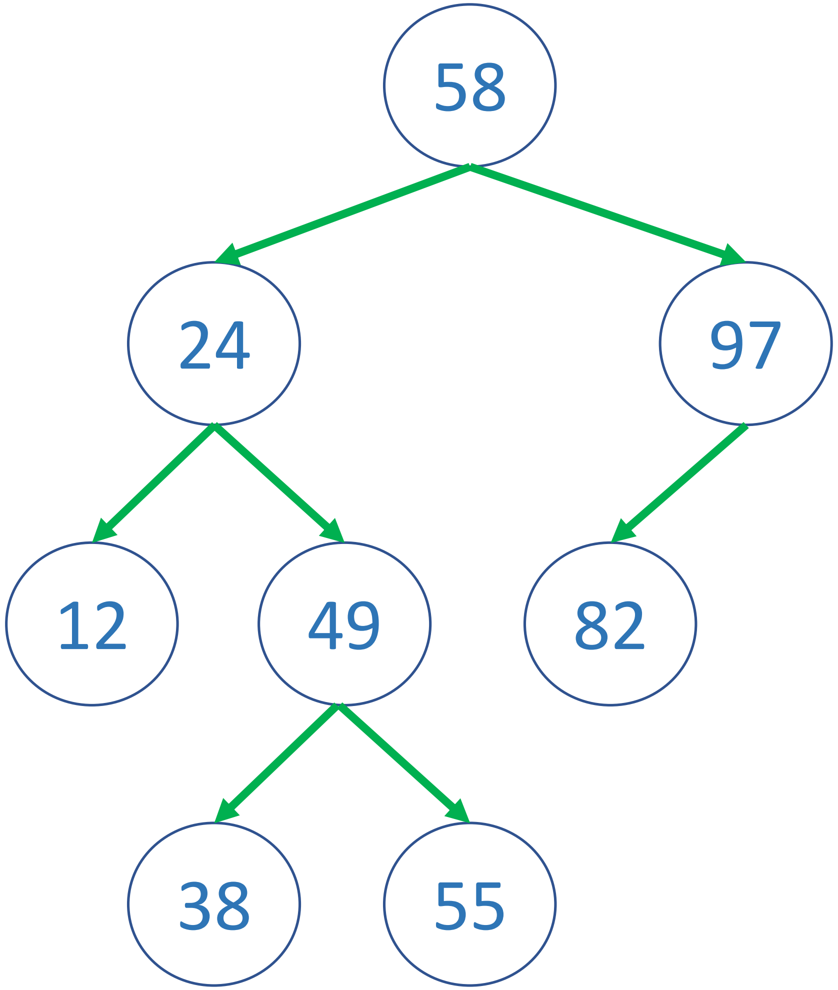
二元樹的搜尋(Search)

- 欲找尋 38
  1. 從根節點 58 開始移動
  2. 因 38 < 58 往左節點移動
  3. 因 38 > 24 往右節點移動
  4. 因 38 < 48往左節點移動
  5. 找到，結束搜尋

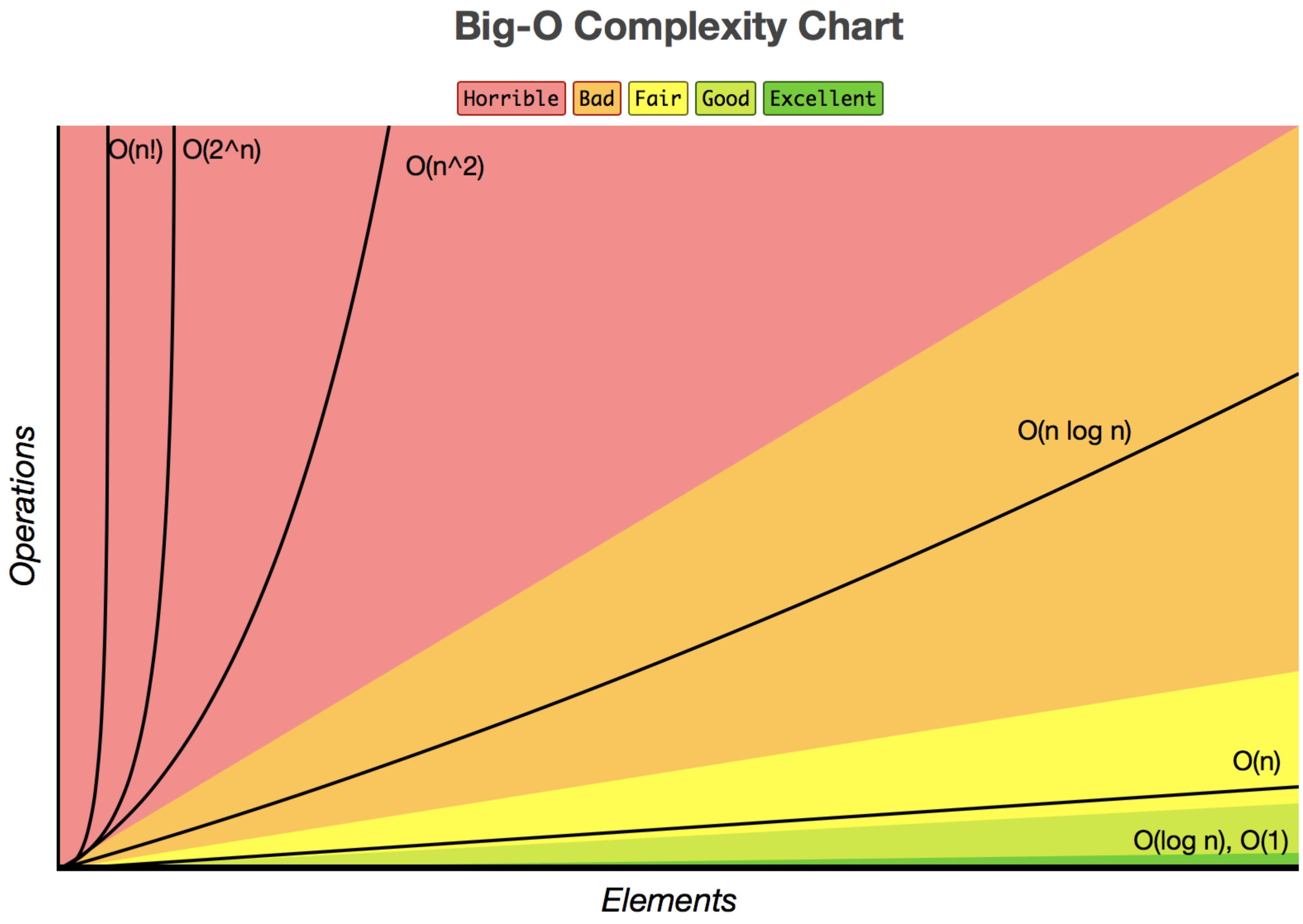


搜尋複雜度

- ➢ 若是完滿二元樹
  - ✓ 每經過一次分岔就可以刪去 ½
  - ✓ n 次搜尋，可以找到 $2^n - 1$筆資料
  - ✓ 搜尋次數 ~ log_2(資料數目)
  - ✓ 搜尋：O(log_2N)
  - ✓ 新增：O(log_2N)
  - ✓ 刪除：O(log_2N)
  - ✓ 通常資料結構/演算法的 log 底數為2



搜尋複雜度



I am here！

搜尋複雜度

- 二元樹
  - ✓ 複雜度 ~ $log_2(n)$
- 三元樹
  - ✓ 複雜度 ~ $log_3(n)$
- 四元樹
  - ✓ 複雜度 ~ $log_4(n)$
- ……以此類推

空間浪費率：

$$\frac{n\times k-(n-1)}{n\times k}=\frac{n\times(k-1)+1}{n\times k}\sim\frac{k-1}{k}$$

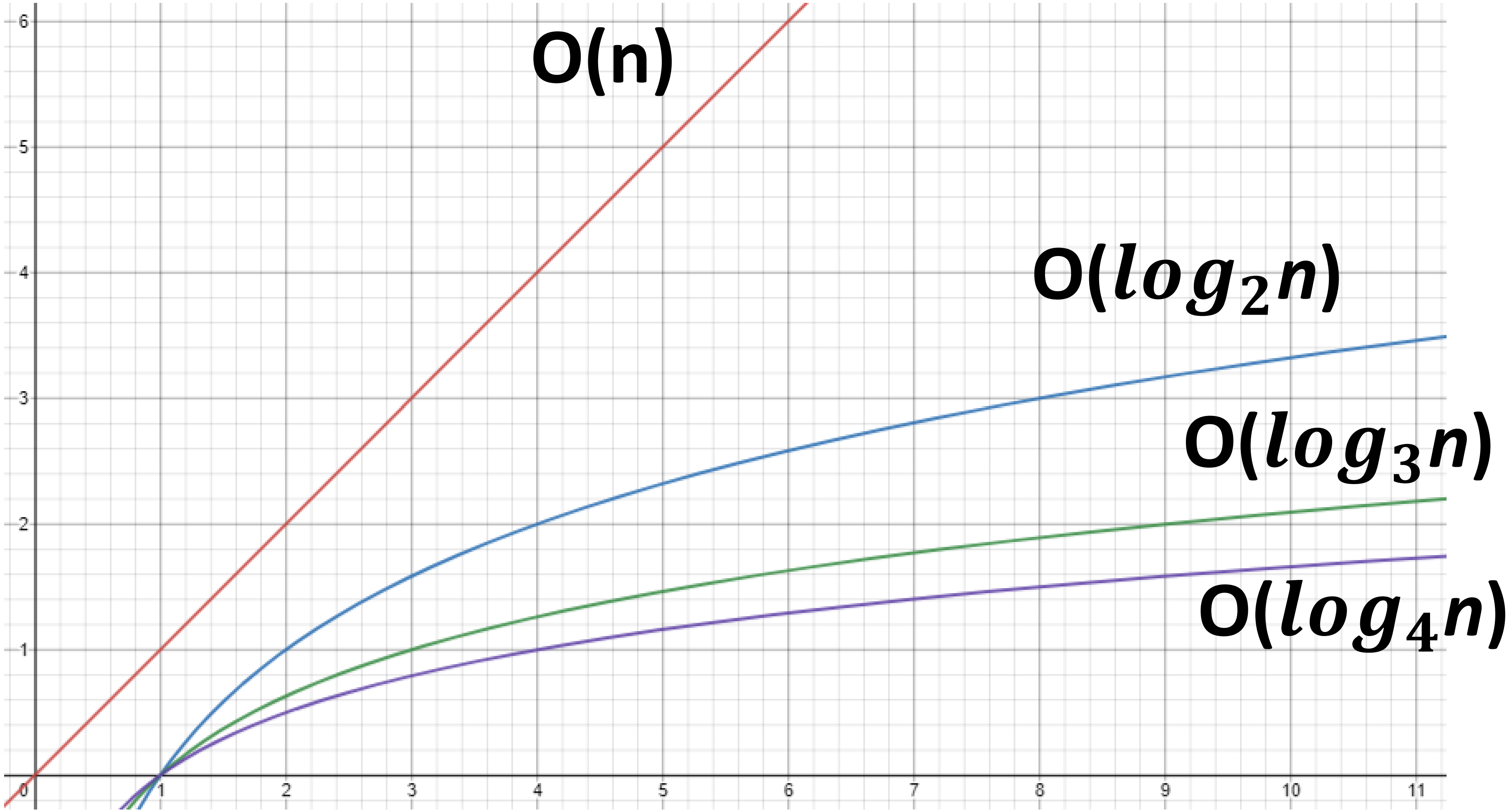
$k=1$：0.0%

$k=2$：50%

$k=3$：67%

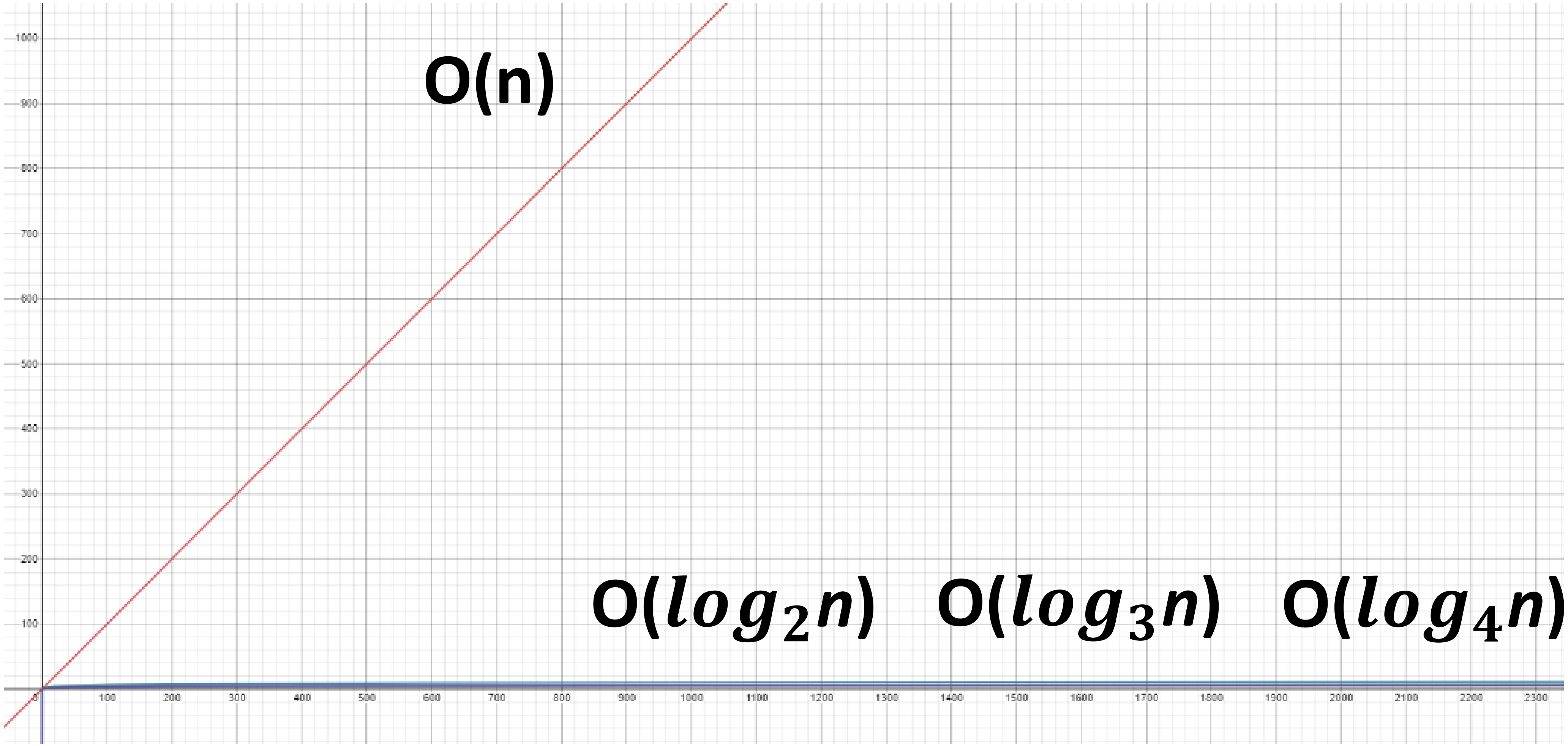
$k=4$：75%

搜尋複雜度



https://www.desmos.com/calculator/unkry1dlhm

搜尋複雜度



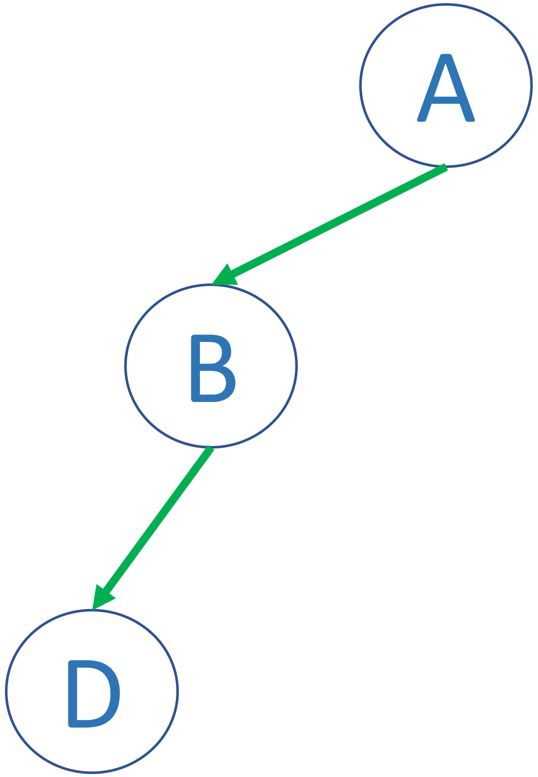
https://www.desmos.com/calculator/unkry1dlhm

搜尋複雜度

	10	100	1000	10000	100000	空間浪費率
O(n)	10.00	100.00	1000.00	10000.00	100000.00	0%
O(log_2n)	3.32	6.64	9.97	13.29	16.61	50%
O(log_3n)	2.10	4.19	6.29	8.38	10.48	67%
O(log_4n)	1.66	3.32	4.98	6.64	8.30	75%
O(log_5n)	1.43	2.86	4.29	5.72	7.15	80%
O(log_6n)	1.29	2.57	3.86	5.14	6.43	83%

搜尋複雜度

- 若是斜曲二元樹
  - ✓ 等同於鏈結串列
  - ✓ 還比鏈結串列浪費空間 QQ
  - ✓ 搜尋：O(N)
  - ✓ 新增：O(N)
  - ✓ 刪除：O(N)



Example Code

Mission

初始化一個二元樹的架構，並且完成以下兩函式：

1. Constructor(建構式)
2. Print

Practice

Mission

試完成以下兩函式：

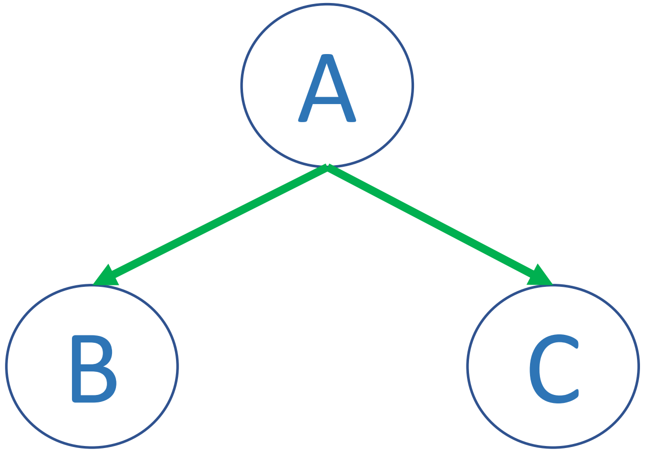
1. Insert
2. Search

二元樹的尋訪(Traversal)

二元樹的尋訪

尋訪 (Traversal) 代表向該地所有連結的地方移動

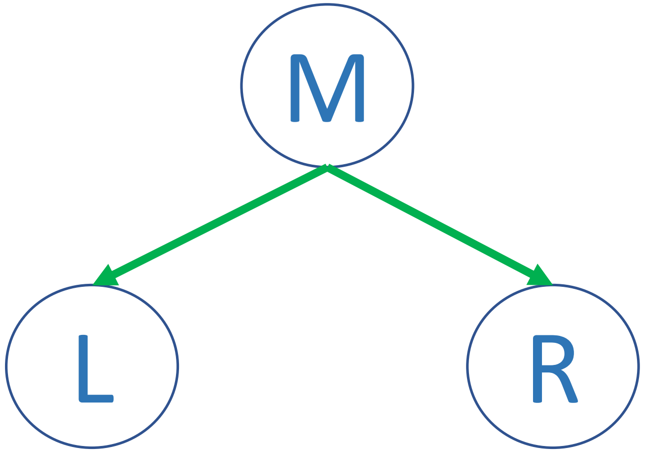
- 移動後可進行讀寫、新增、刪除等操作
- 如下圖，A 可分別至 B、C 進行操作
- 樹的尋訪代表希望每個節點都能**剛好被操作一次**



二元樹的尋訪

從單一子樹的角度

- 共有 L、M、R 三個方向可以操作
  1. 操作 M 節點的資料
  2. 往 L 方向繼續探索
  3. 往 R 方向繼續探索
- 因此有六種可能
  - ✓ 因左右對稱為免重複
  - ✓ 規定 L 方向一定要在 R 之前
  - ✓ 剩下三種



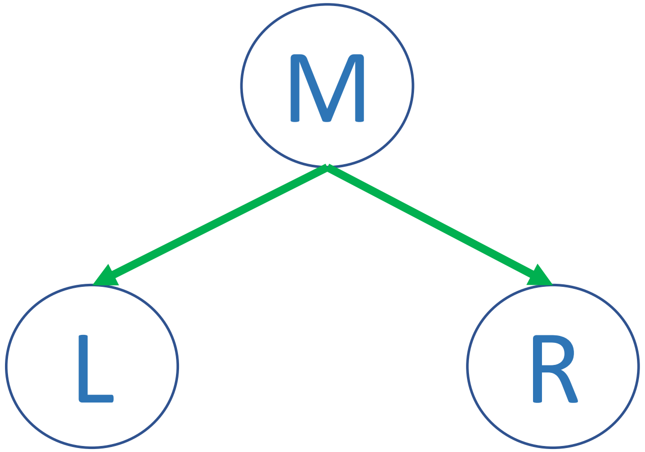
MLR
~~MRL~~
LMR
LRM
~~RLM~~
~~RML~~

MLR	前序
LMR	中序
LRM	後序

二元樹的尋訪

尋訪方式分三種

- ➢ 前序 (Pre-order)
  - ✓ 中→左→右
- ➢ 中序 (In-order)
  - ✓ 左→中→右
- ➢ 後序 (Post-order)
  - ✓ 左→右→中



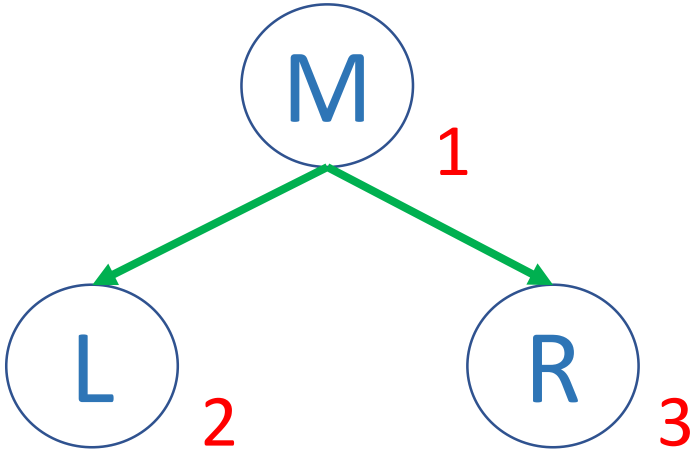
MLR
~~MRL~~
LMR
LRM
~~RLM~~
~~RML~~

MLR	前序
LMR	中序
LRM	後序

前序尋訪

尋訪順序(前序尋訪)

1. **處理當前節點的資料**
2. 往左探索 (遞迴)
3. 往右探索 (遞迴)

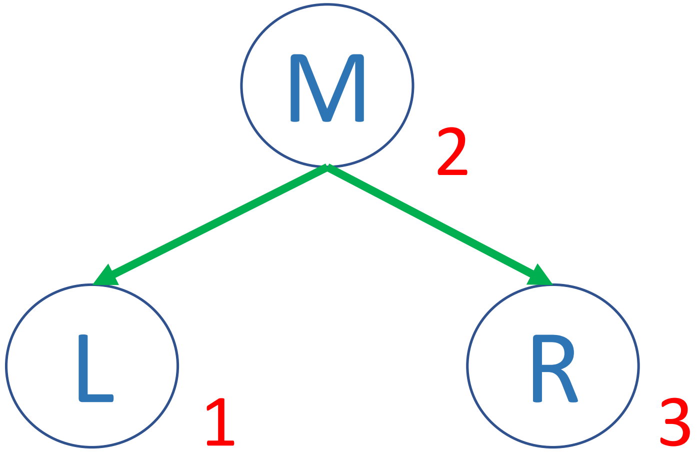



```
void Pre_Order(Node* n){
        n->data;
        Pre_Order(n->left);
        Pre_Order(n->right);
}
```

中序尋訪

尋訪順序(中序尋訪)

1. 往左探索 (遞迴)
2. 處理當前節點的資料
3. 往右探索 (遞迴)

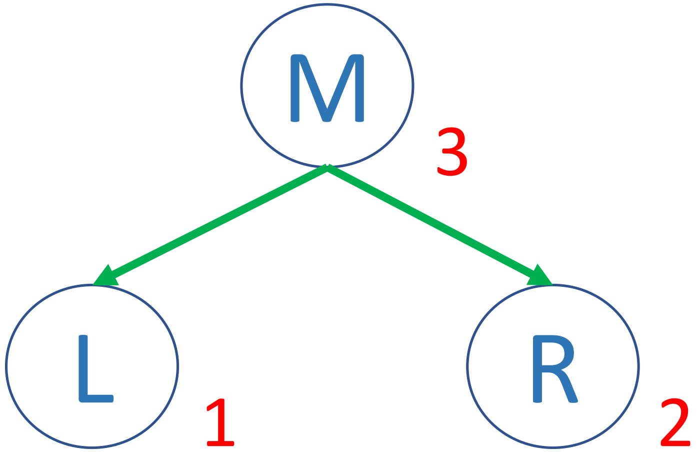



```
void In_Order(Node* n){
        In_Order(n->left);
        n->data;
        In_Order(n->right);
}
```

後序尋訪

尋訪順序(後序尋訪)

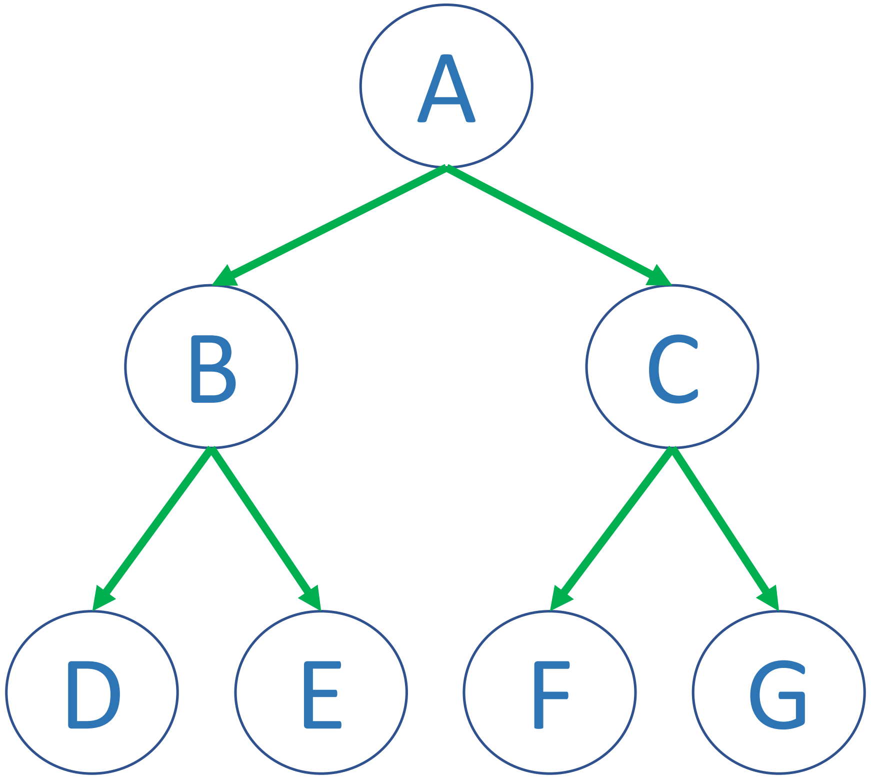
1. 往左探索 (遞迴)
2. 往右探索 (遞迴)
3. **處理當前節點的資料**




```
void Post_Order(Node* n){
        Post_Order(n->left);
        Post_Order(n->right);
        n->data;
}
```

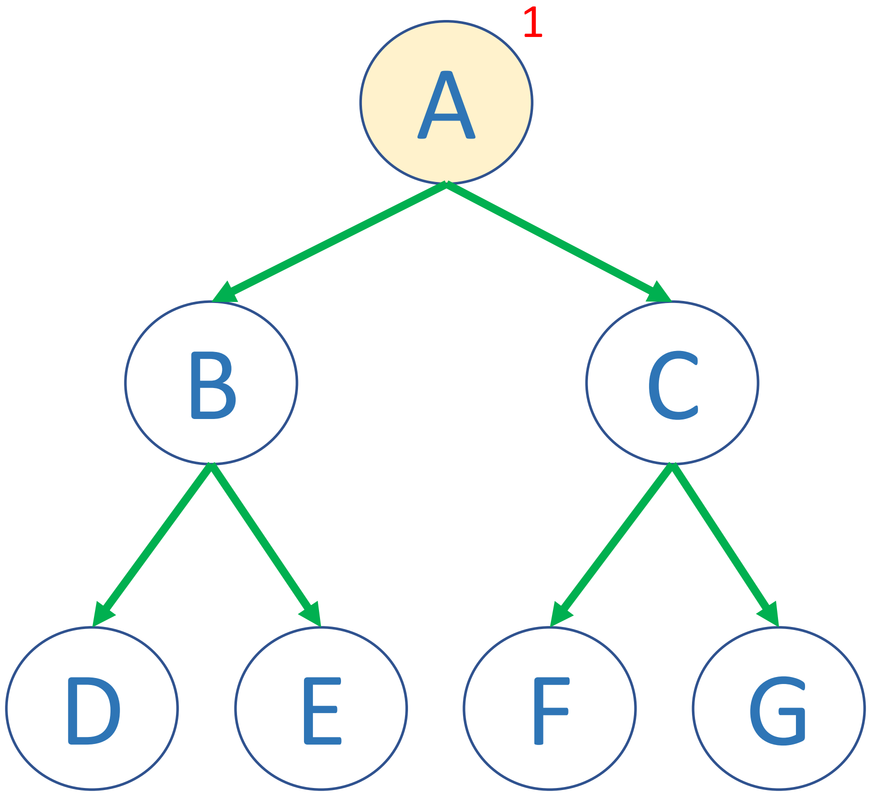
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點



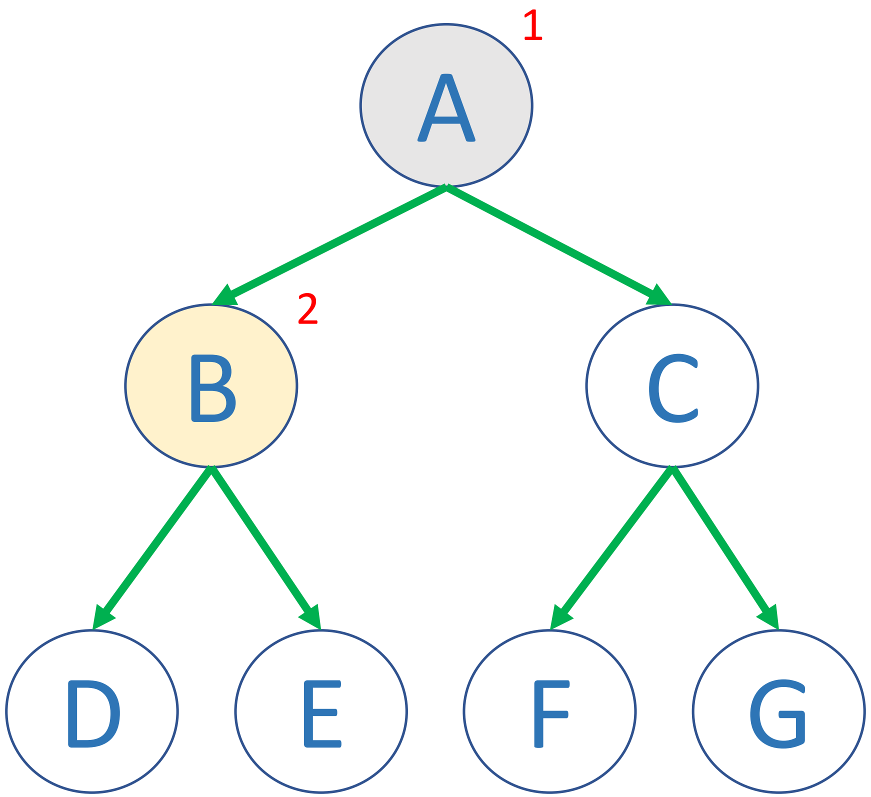
二元樹的尋訪

- ➢ 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ A



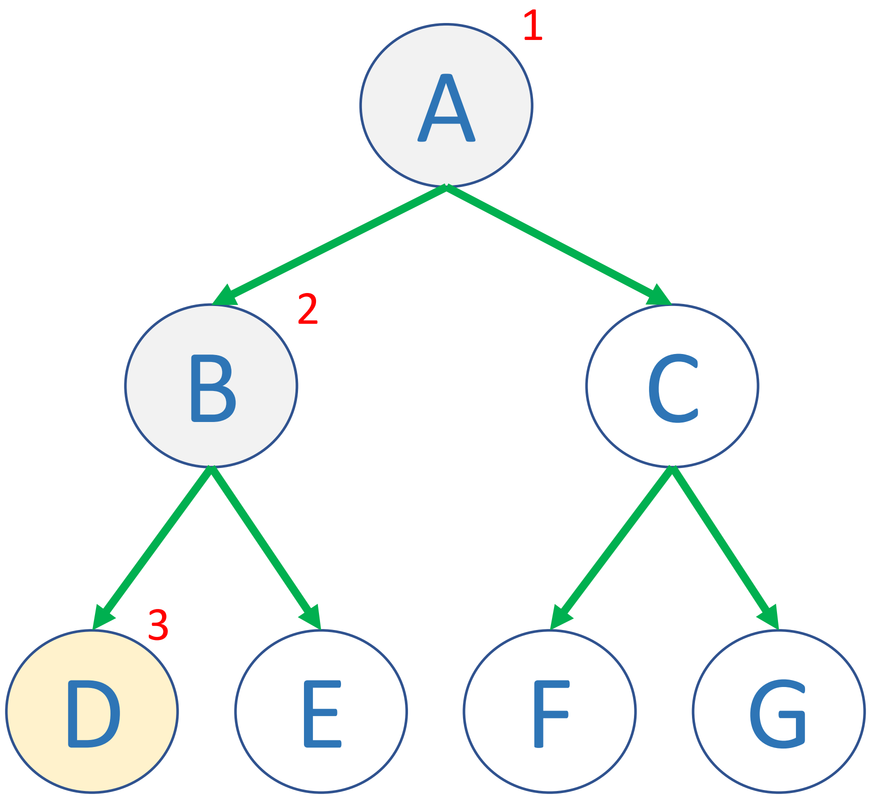
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ AB



二元樹的尋訪

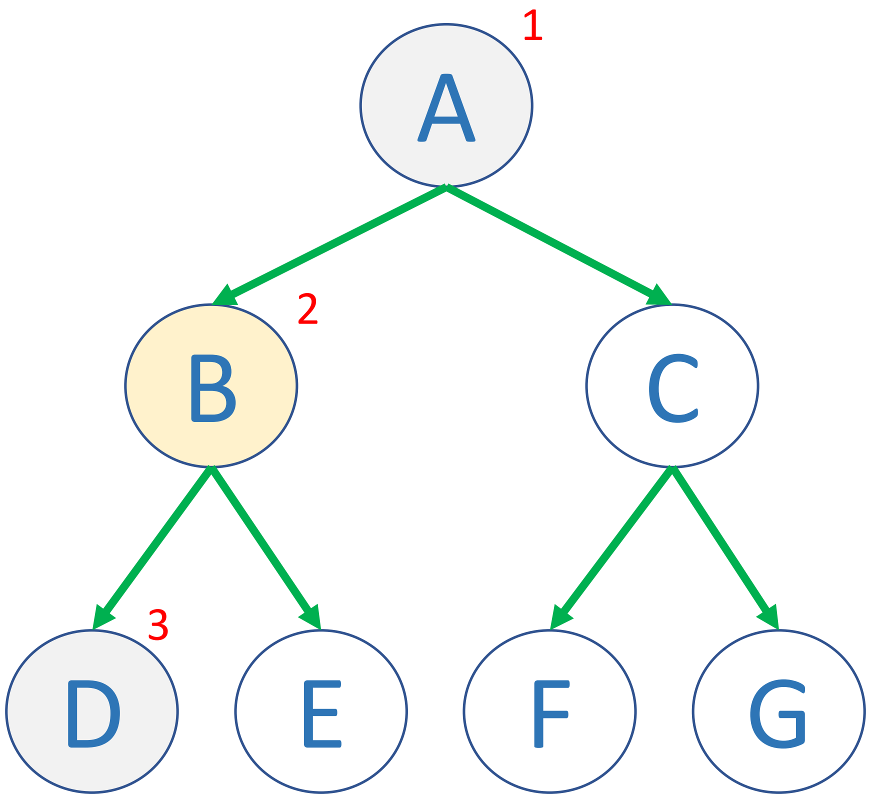
- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABD



二元樹的尋訪

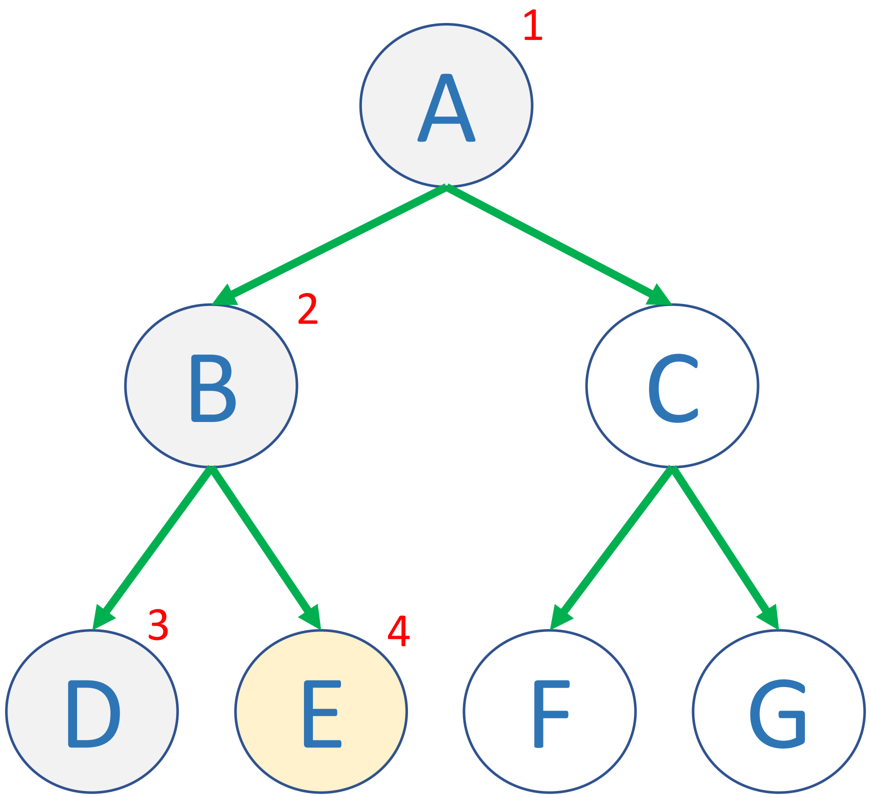
➢ 前序 (Pre-order)

- ✓ 每經過一個新節點就先處理該節點
- ✓ ABD



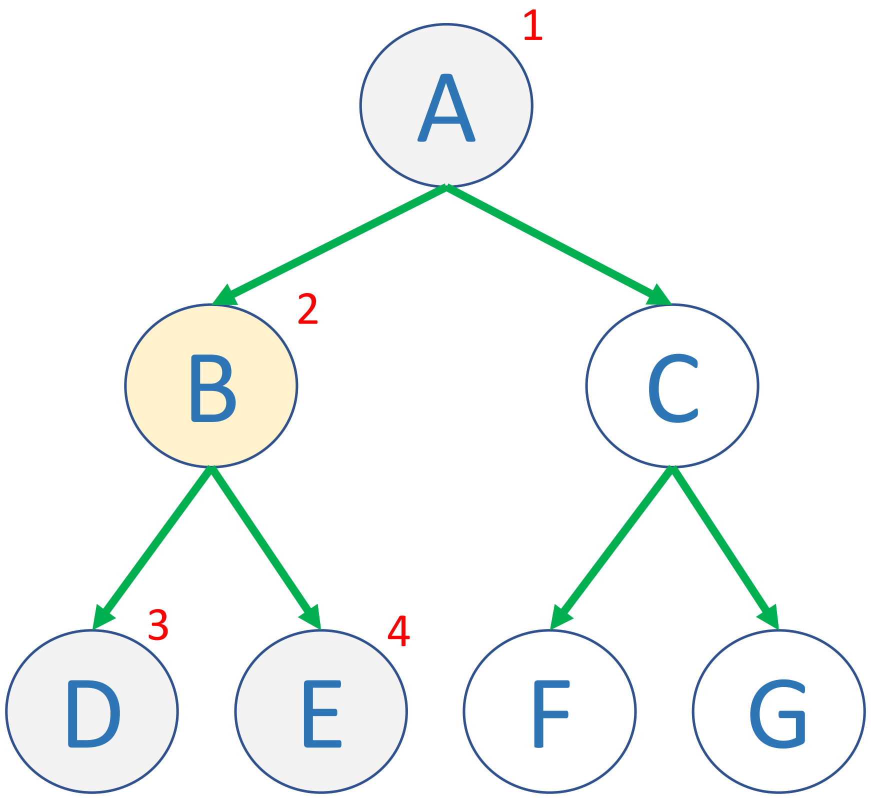
二元樹的尋訪

➢ 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABDE



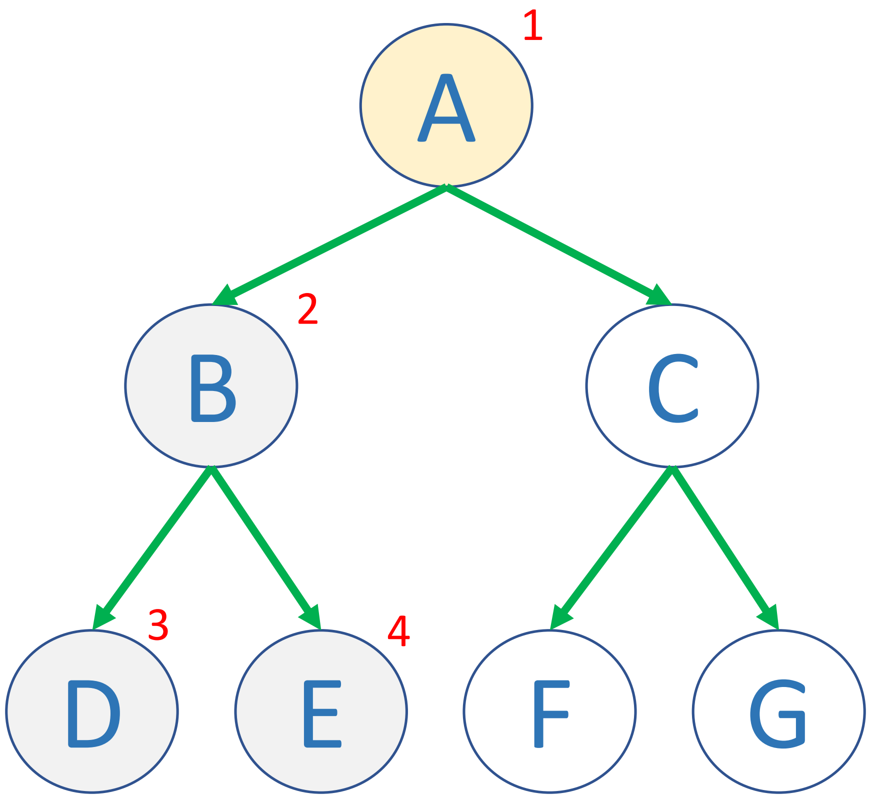
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABDE



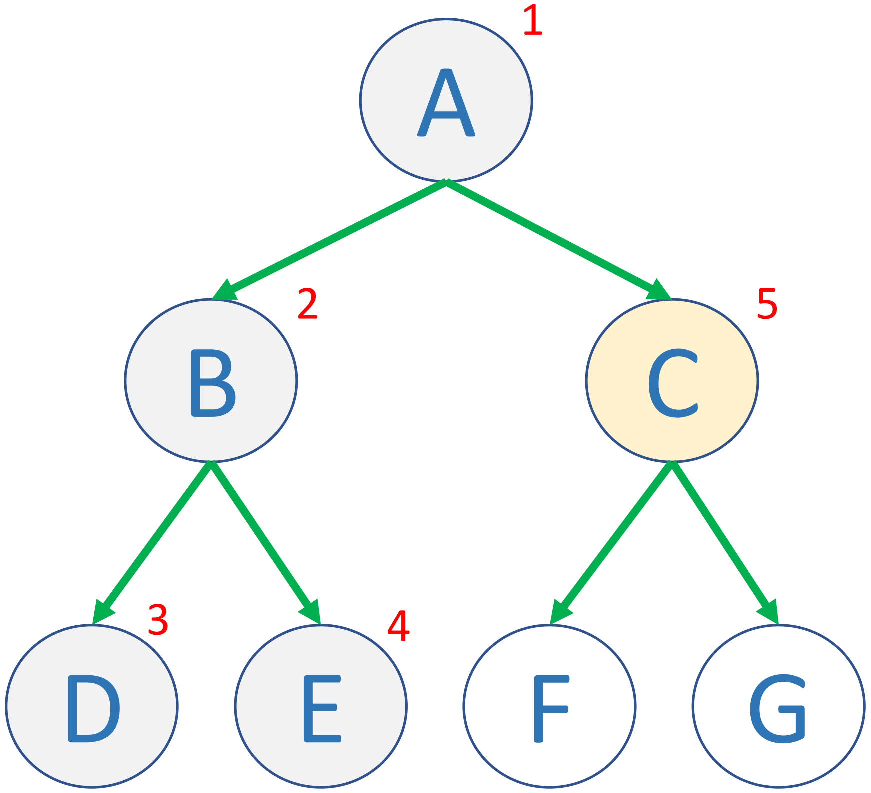
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABDE



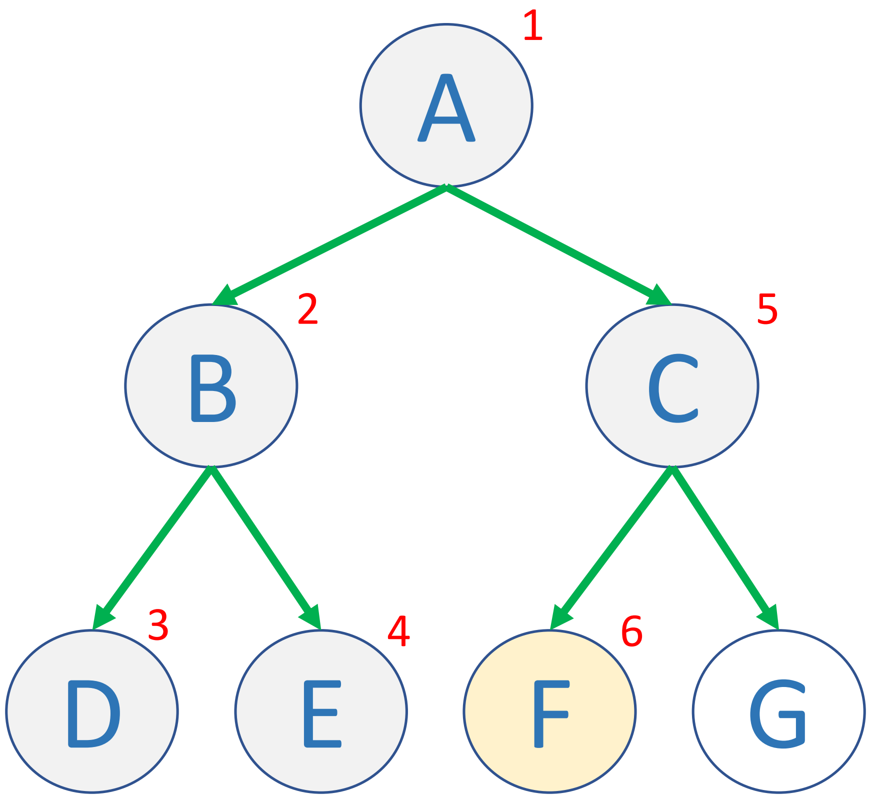
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABDEC



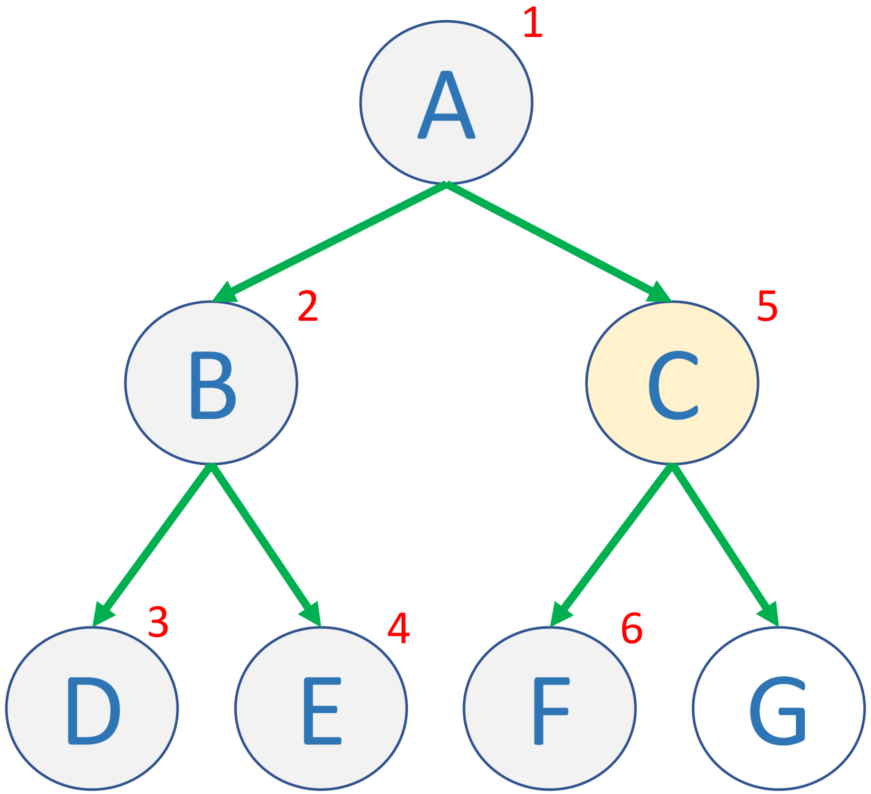
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABDECF



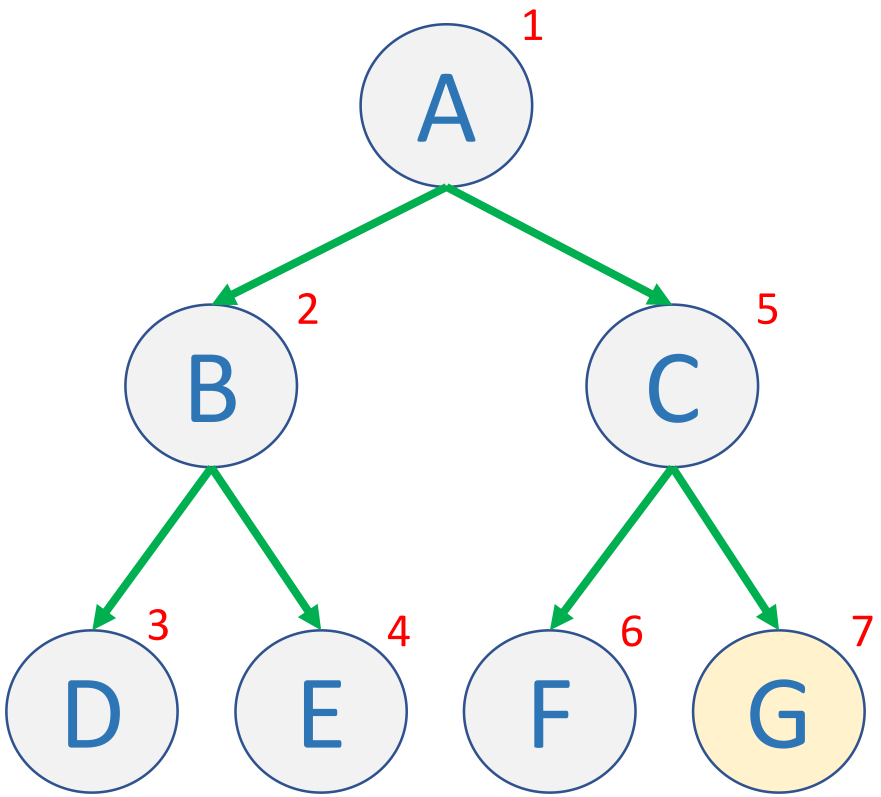
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABDECF



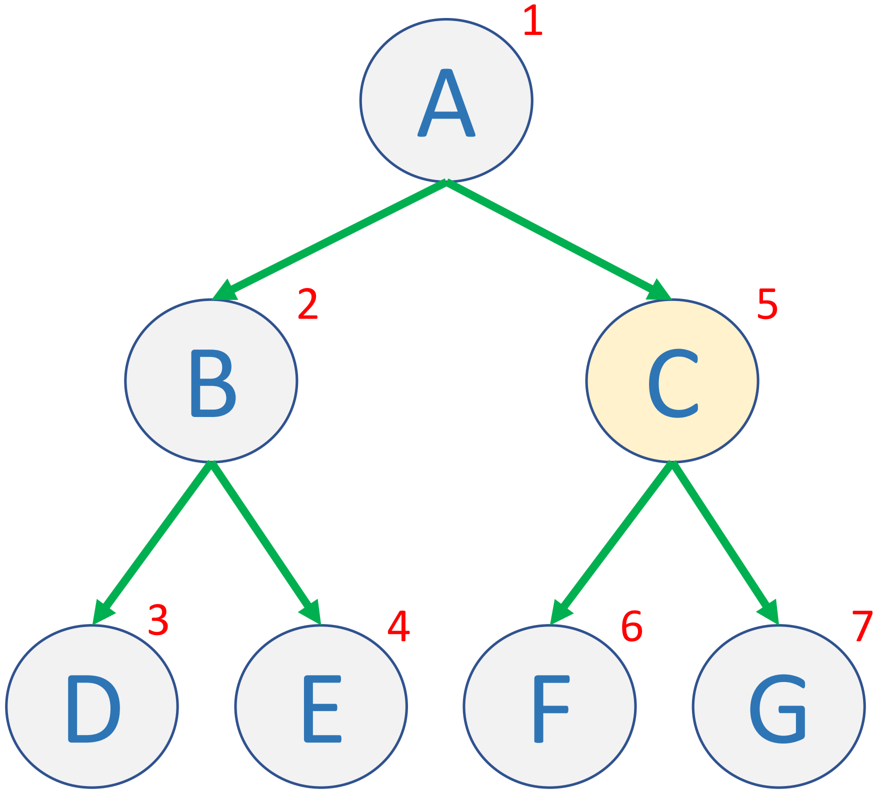
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABDECFG



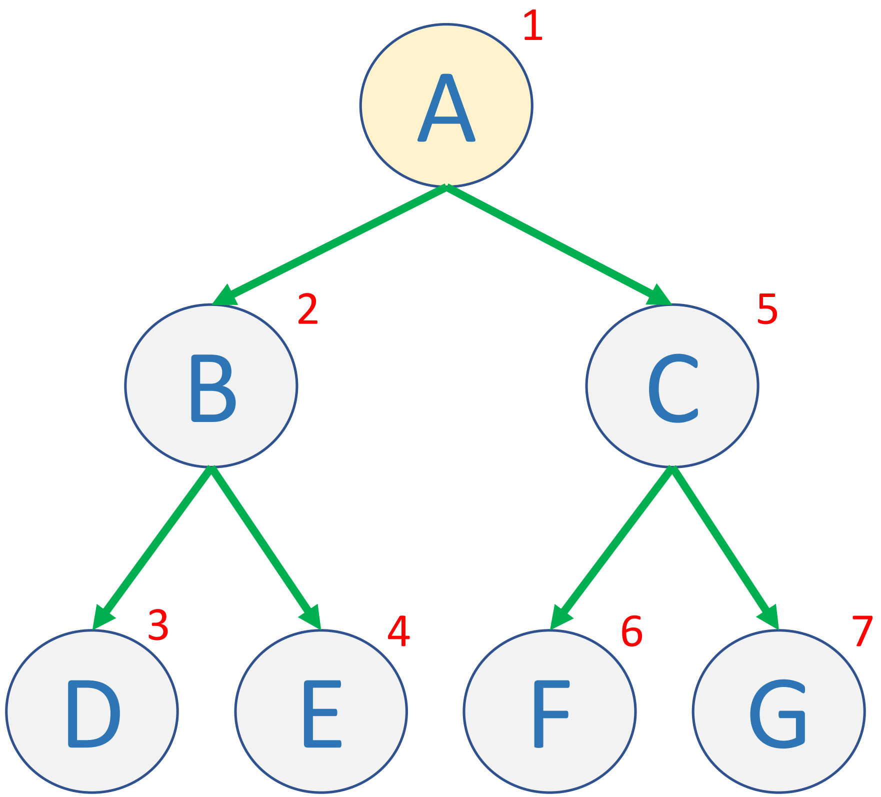
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABDECFG



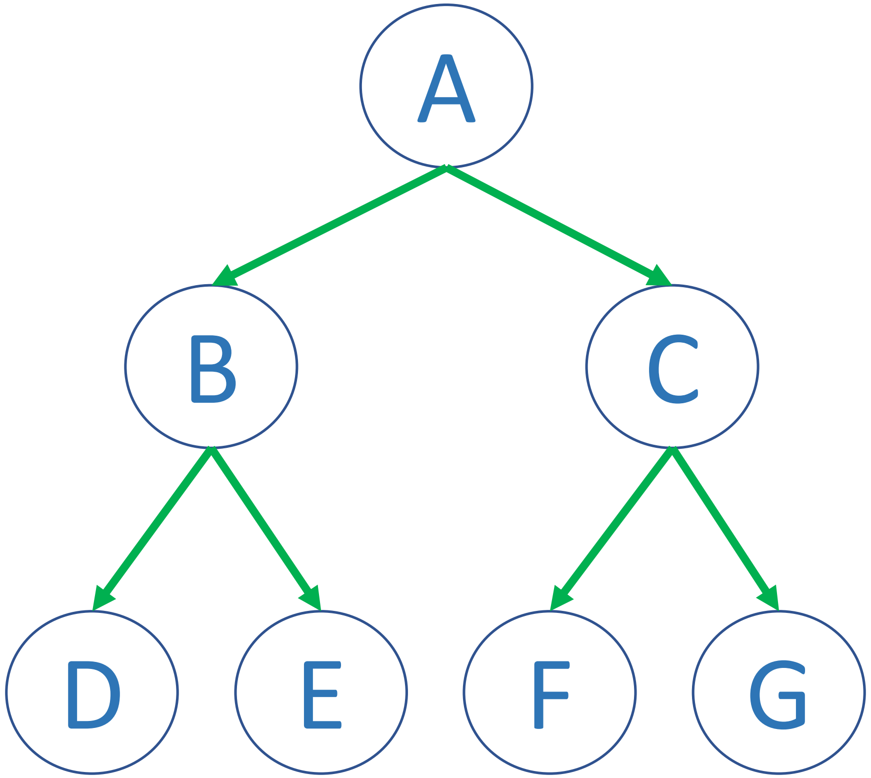
二元樹的尋訪

- 前序 (Pre-order)
  - ✓ 每經過一個新節點就先處理該節點
  - ✓ ABDECFG



二元樹的尋訪

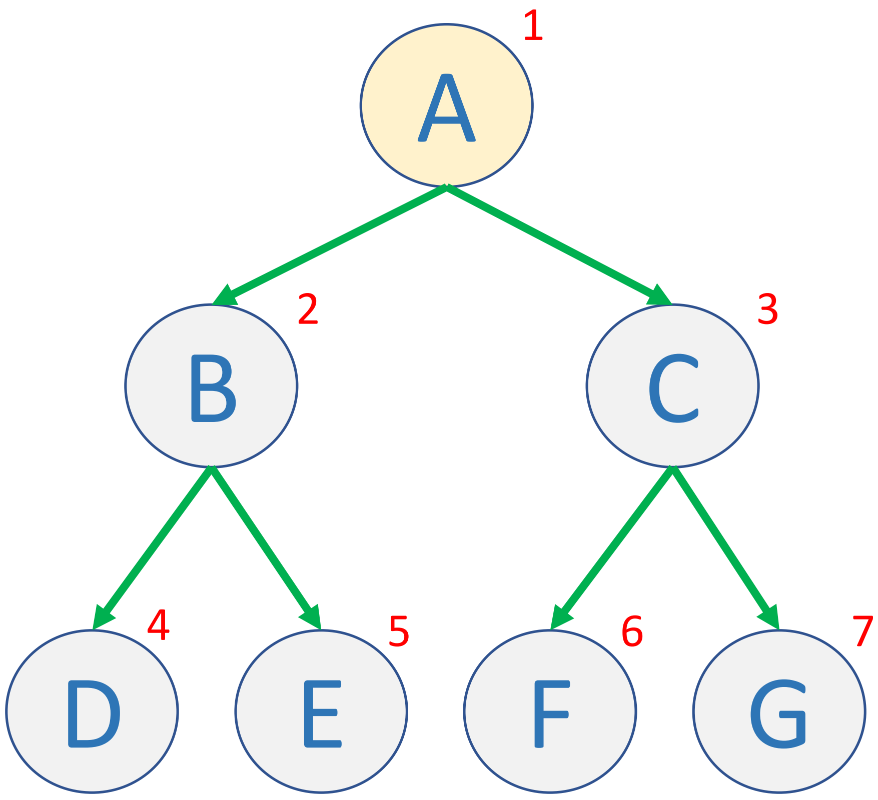
- 前序 (Pre-order)
  - ✓ ABDECFG
  - ✓ 每經過一個新節點就先處理該節點
- 中序 (In-order)
  - ✓ DBEAFCG
  - ✓ 左子樹的資料都處理完再處理該節點
- 後序 (Post-order)
  - ✓ DEBFGCA
  - ✓ 左右子樹的資料都處理完再處理該節點



紅字為根節點

二元樹的尋訪

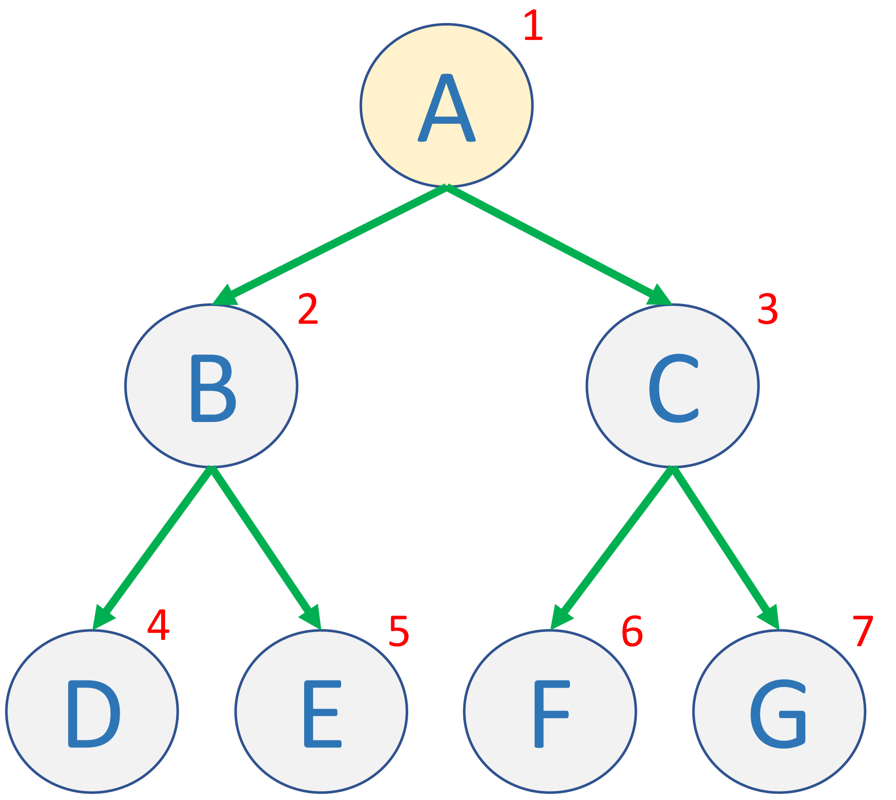
- Level-Order Traversal
  - ✓ 依照階層 (level) 順序由上至下
  - ✓ 同一階層 (level) 內由左至右
  - ✓ 做法：
    1. 準備一個 Queue
    2. 把根節點放入 Queue
    3. 依序從 Queue 裡取出節點
    4. 把取出節點的子節點放入 Queue
    5. 重複步驟 3~4
  - ✓ ABCDEFG



二元樹的尋訪

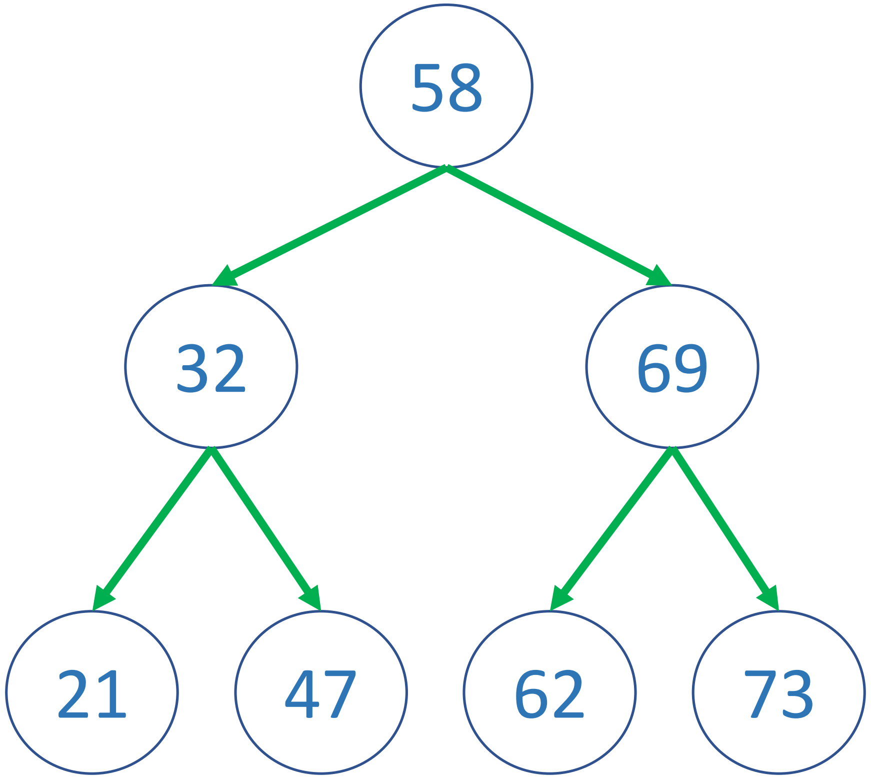
➢ Level-Order Traversal

1. 把根節點 A 放入 Queue(Q)
   - ✓ Q = {A}
2. 取出 A，把 A 的子節點 B、C 放入 Q
   - ✓ Q = {B, C}
3. 取出 B，把 B 的子節點 D、E 放入 Q
   - ✓ Q = {C, D, E}
4. 取出 C，把 C 的子節點 F、G 放入 Q
   - ✓ Q = {D, E, F, G}
5. 依序取出 D、E、F、G，結束



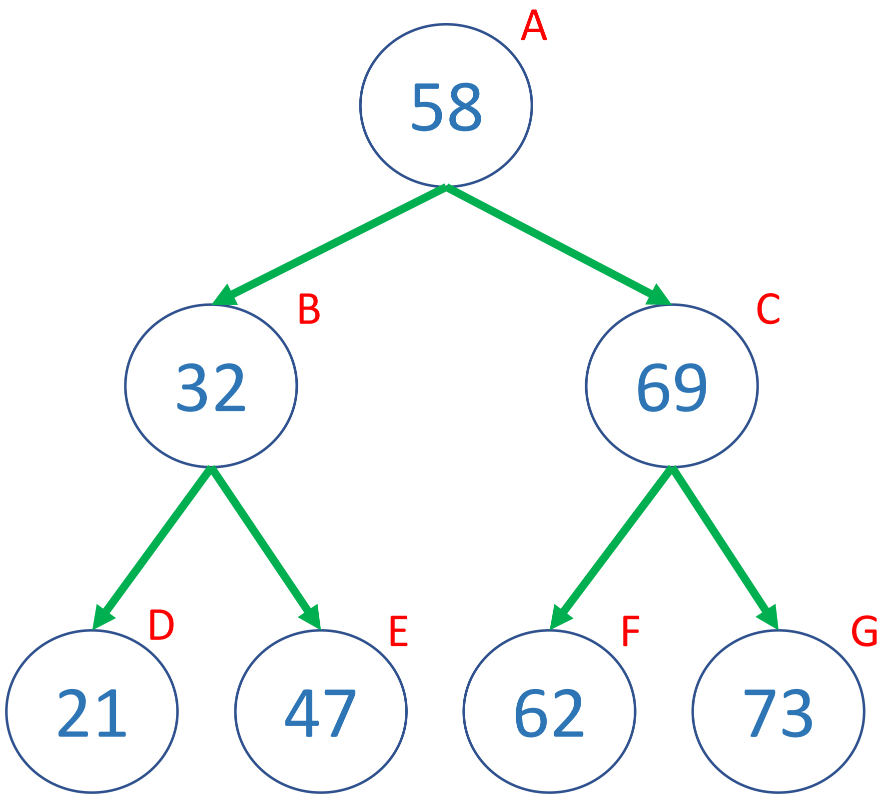
二元樹的尋訪

- ➢ Left most
  - ✓ 回傳該子樹中，最左(小)的節點
  - ✓ 不斷往左節點走，直至空指標
  - ✓ Left most of 58：21
- ➢ Right most
  - ✓ 回傳該子樹中，最右(大)的節點
  - ✓ 不斷往右節點走，直至空指標
  - ✓ Right most of 58：73



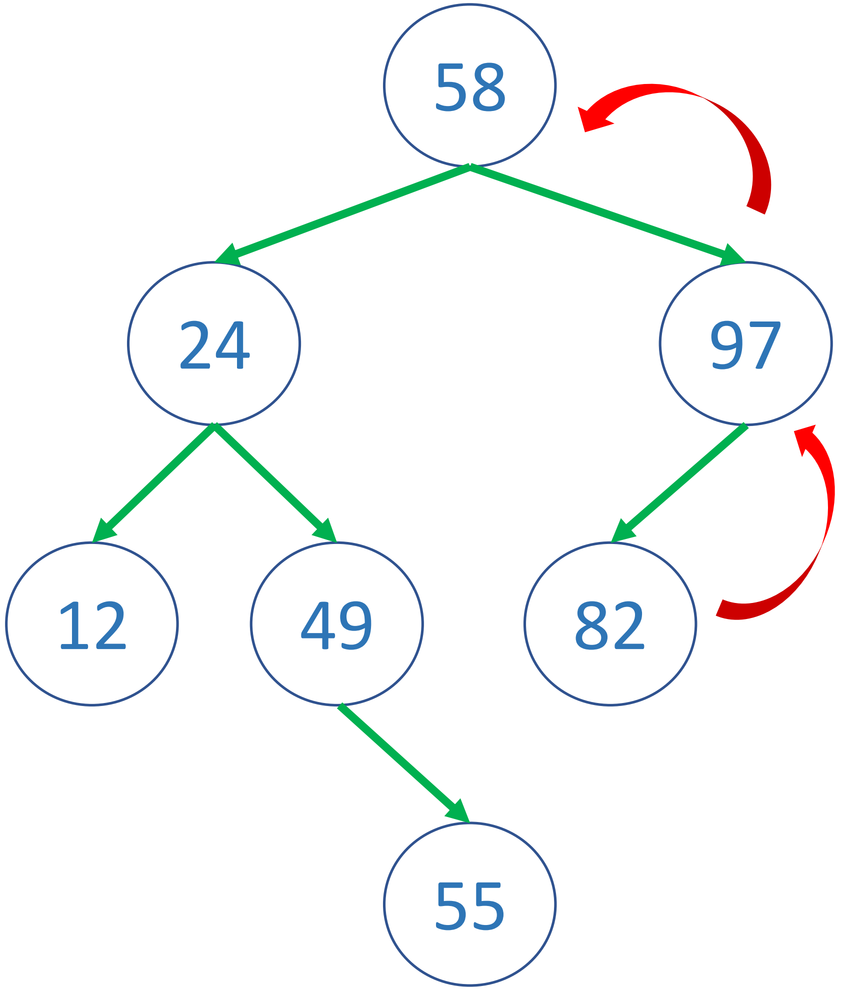
二元樹的尋訪

- Inorder 順序就是編號(索引)大小順序
  - ✓ DBEAFCG
  - ✓ 21→32→47→58→62→69→73
- Predecessor
  - ✓ 依照編號大小順序，該節點的**前一個**節點
  - ✓ 又名 Inorder_Predecessor
  - ✓ Predecessor of 32：21
- Successor
  - ✓ 依照編號大小順序，該節點的**後一個**節點
  - ✓ 又名 Inorder_Successor
  - ✓ Predecessor of 32：47



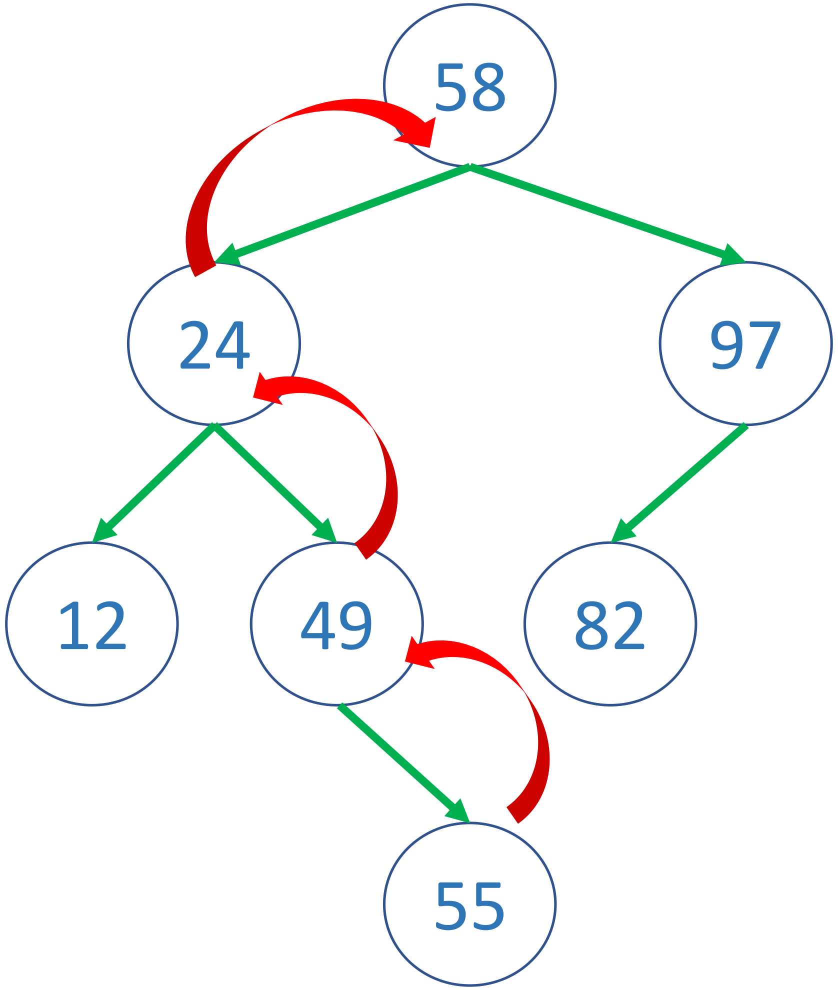
二元樹的尋訪

- Predecessor
  - ✓ 該節點的前一個節點
  1. 左子樹不為空
     - ✓ 左子樹的 Right most
  2. 左子樹為空
     - ✓ 找該節點的 Ancestor Node
     - ✓ 且該節點在 Ancestor Node 的右子樹
  - ✓ Predecessor of 24：12
  - ✓ Predecessor of 12：null
  - ✓ Predecessor of 97：82
  - ✓ Predecessor of 82：58
  - ✓ 有 Parent 指標會較方便



二元樹的尋訪

- Successor
  - ✓ 該節點的**後一個**節點
  1. 右子樹不為空
     - ✓ 右子樹的 Left most
  2. 右子樹為空
     - ✓ 找該節點的 Ancestor Node
     - ✓ 且該節點在 Ancestor Node 的**左子樹**
  - ✓ Successor of 24：49
  - ✓ Successor of 12：24
  - ✓ Successor of 97：null
  - ✓ Successor of 55：58
  - ✓ 有 Parent 指標會較方便

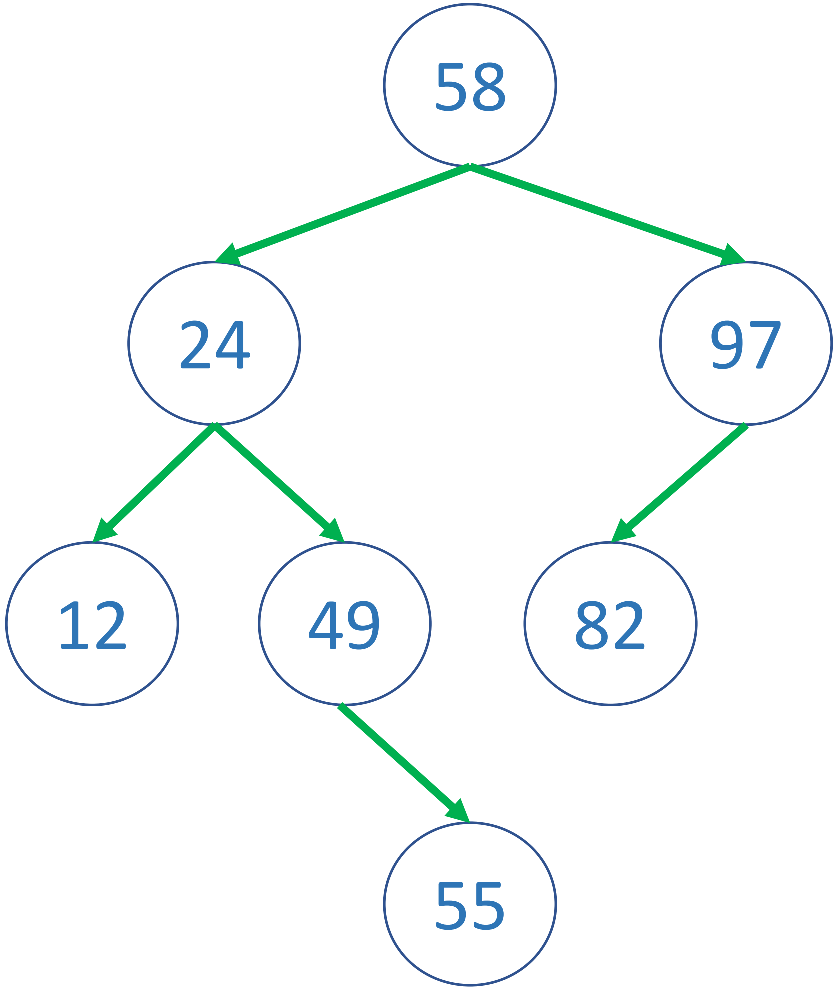


Example Code

Mission

寫出以下四種尋訪函式並印出

1. Pre-order (前序)
2. In-order (中序)
3. Post-order (後序)
4. Level-Order Traversal

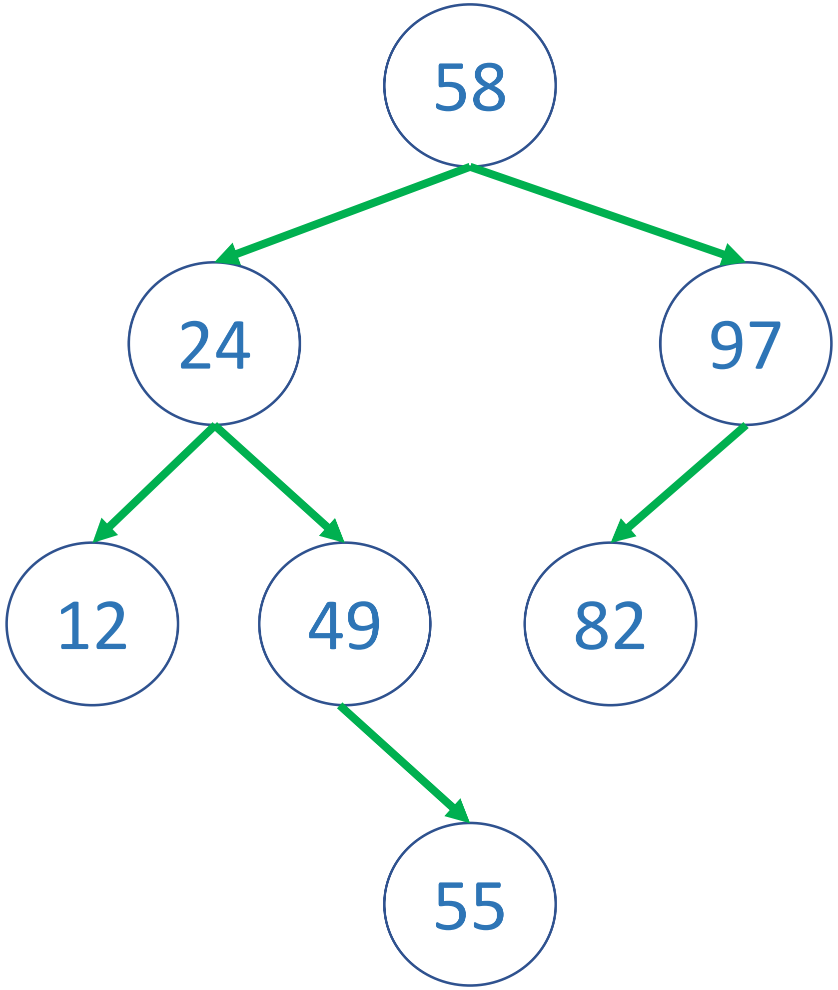


Practice

Mission

試寫出以下四種函式

1. Left_Most
2. Right_Most
3. Predecessor
4. Successor

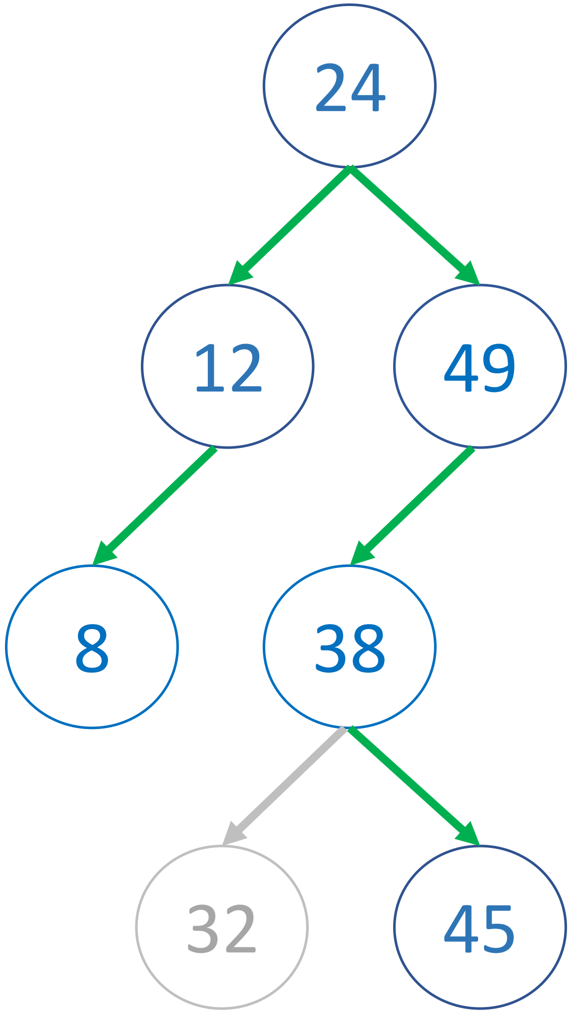


二元樹的刪除(Delete)

二元樹的刪除(Delete)

依據刪除的節點種類不同分成三種狀況

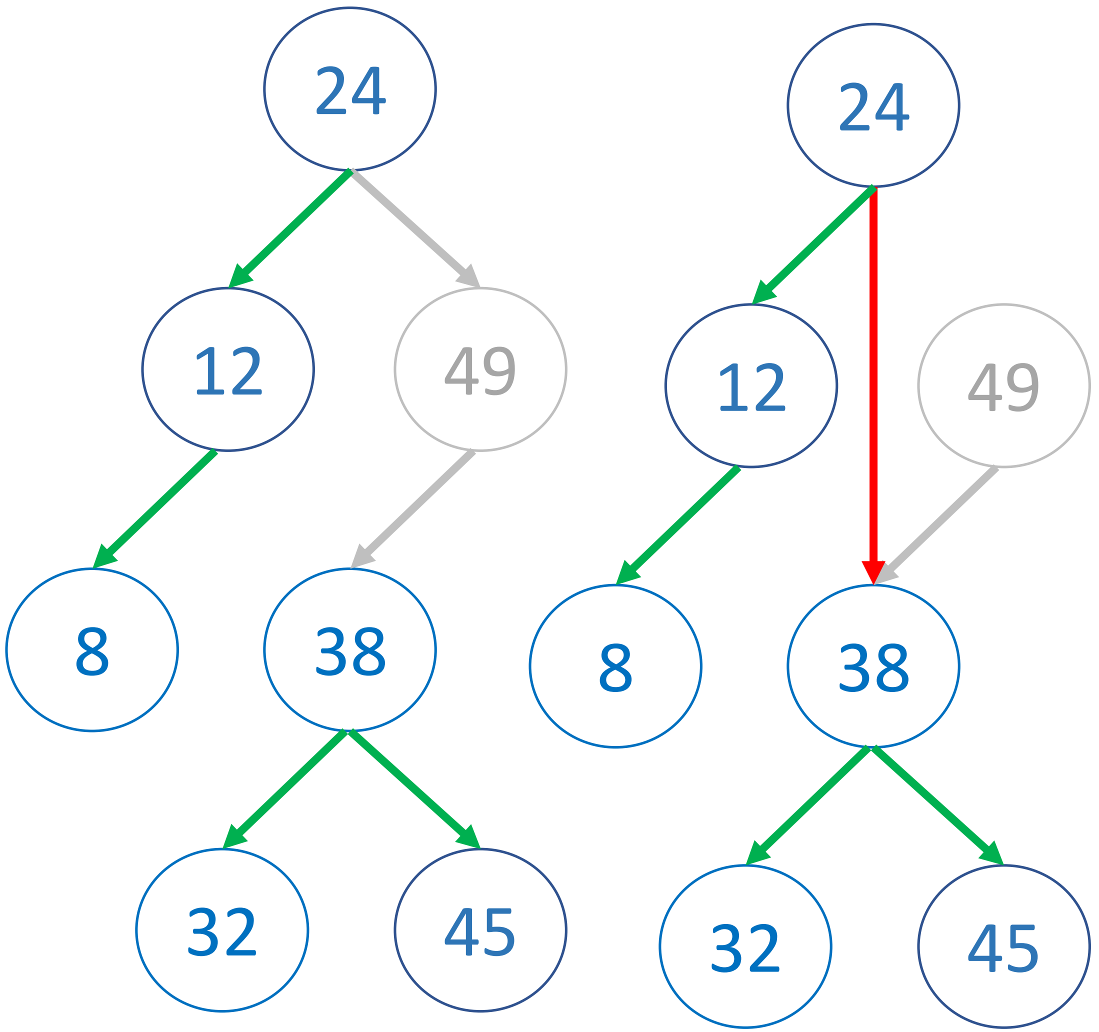
1. 沒有任何子節點 (葉節點)
   - ➢ 直接刪除即可
2. 有一個子節點
   - ➢ 把該節點的角色用子節點替代
3. 有兩個子節點
   - ➢ 刪除後必須找個節點來替代



二元樹的刪除(Delete)

依據刪除的節點種類不同分成三種狀況

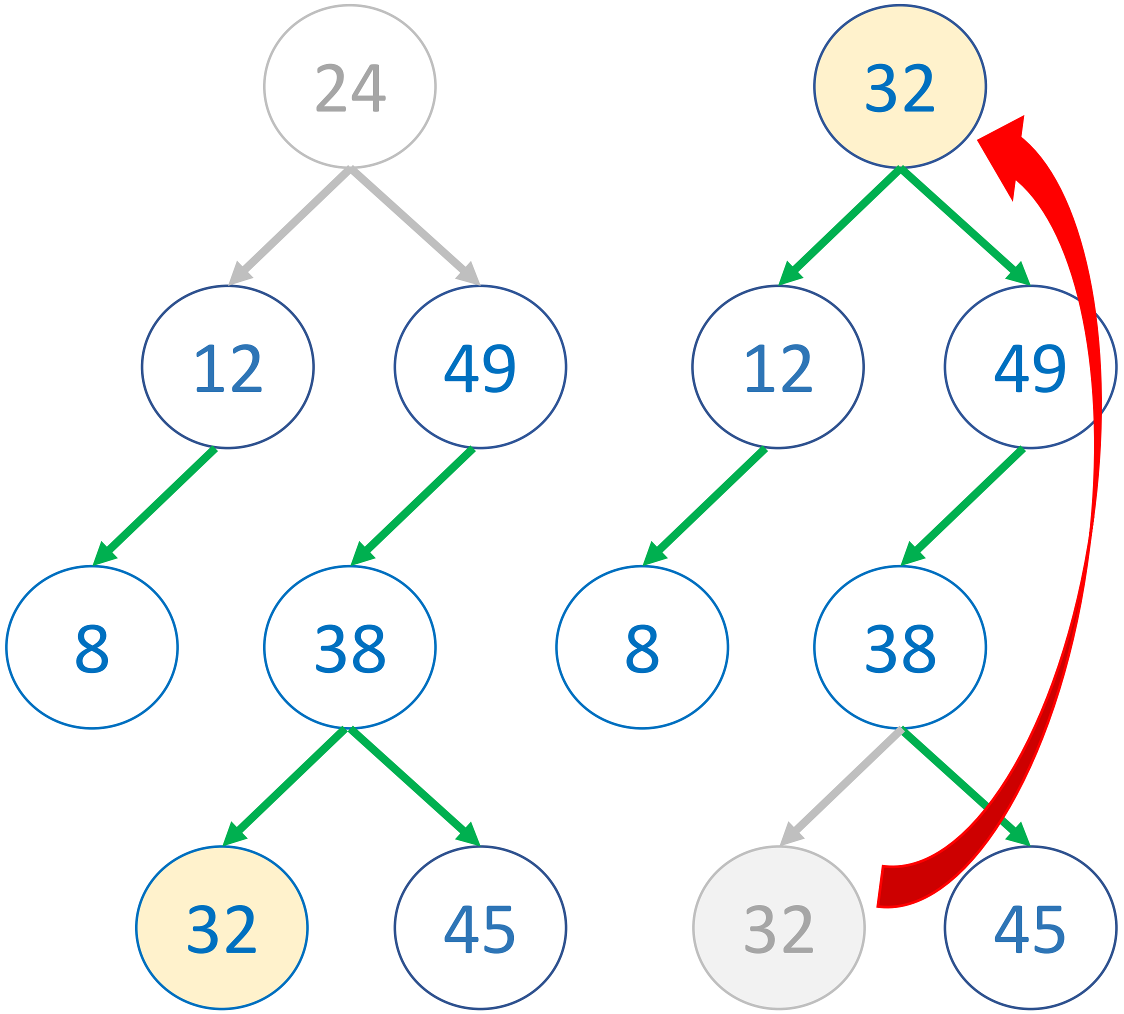
1. 沒有任何子節點 (葉節點)
   - ➢ 直接刪除即可
2. 有一個子節點
   - ➢ 把該節點的角色用子節點替代
3. 有兩個子節點
   - ➢ 刪除後必須找個節點來替代



二元樹的刪除(Delete)

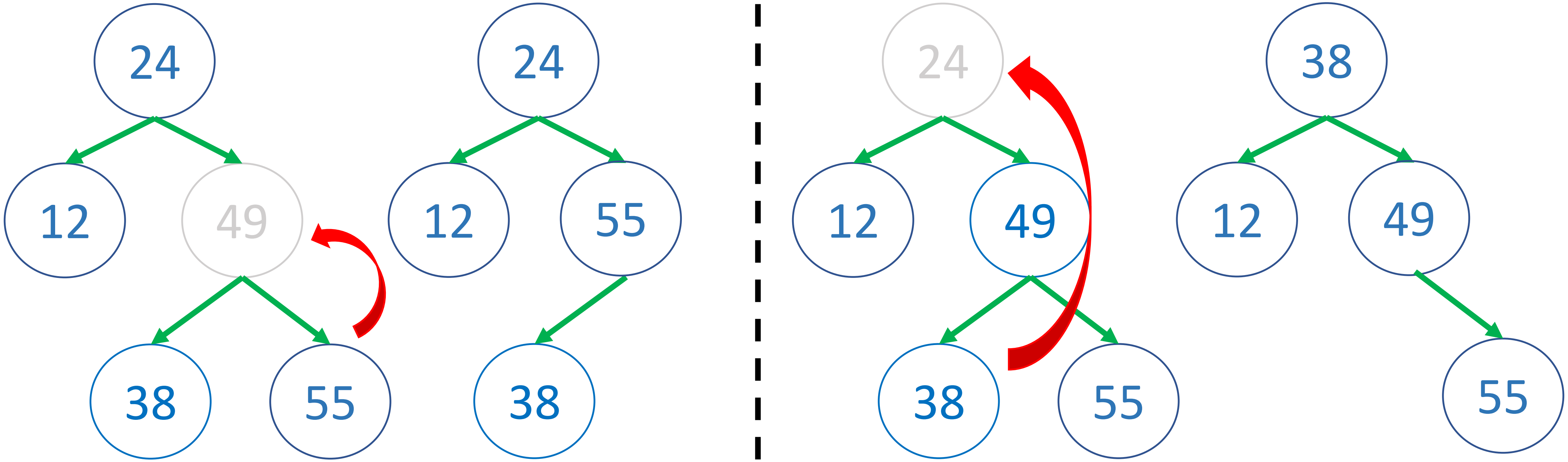
依據刪除的節點種類不同分成三種狀況

1. 沒有任何子節點 (葉節點)
   ➢ 直接刪除即可
2. 有一個子節點
   ➢ 把該節點的角色用子節點替代
3. 有兩個子節點
   ➢ 刪除後必須找個節點來替代



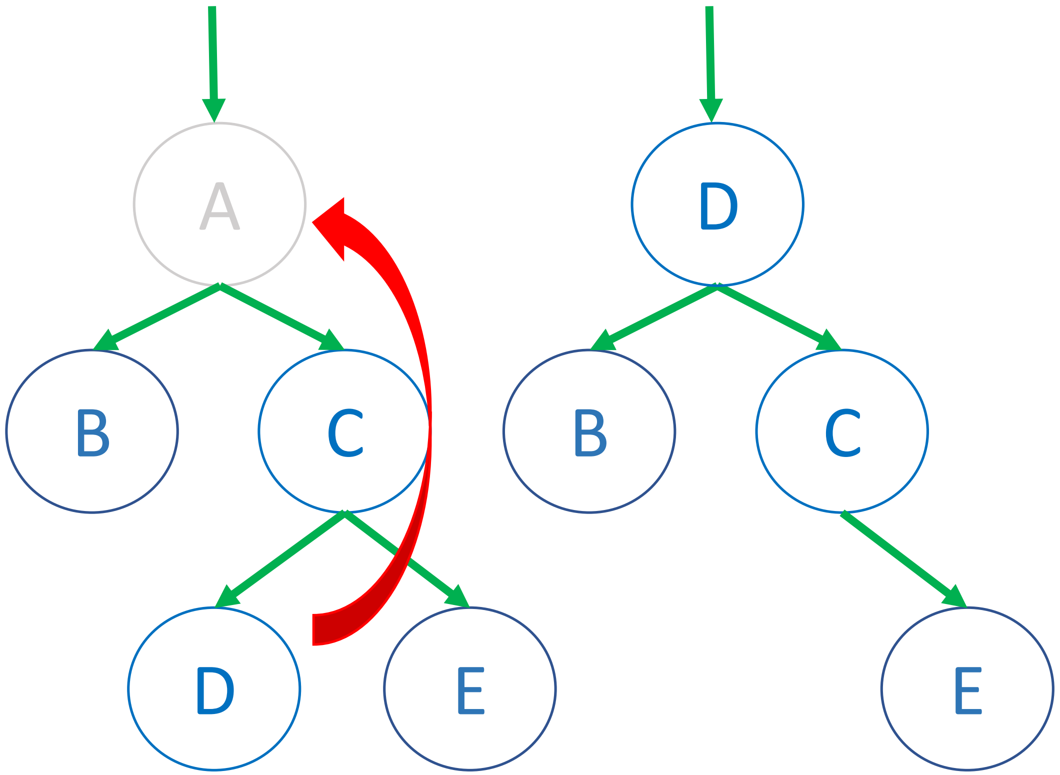
二元樹的刪除(Delete)

- 欲刪除的節點有兩個子節點
  - ➢ 刪除後的二元樹必須遵守原本的規則
  - ➢ 從其右子樹找最小的 node 來取代 (Successor)
  - ➢ 或選左子樹中的最大的 node (Predecessor)



二元樹的刪除(Delete)

- 移動找到的節點 (D) 至刪除的節點 (A)
  1. 把 D 的資料複製給 A
  2. 把 D 節點的記憶體釋放掉
  3. 把 C 原本指向 D 的指標設定成空指標



Example Code

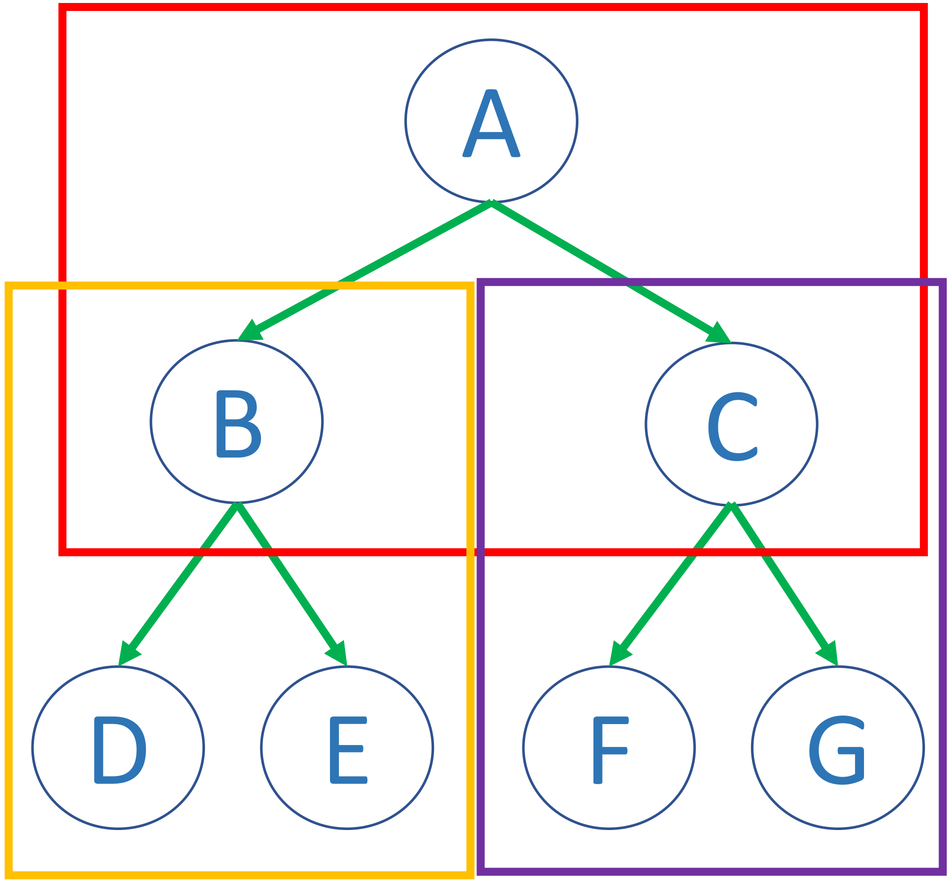
Mission

試完成**刪除**函式

實戰演練

二元樹解題

- 大部分二元樹的題目都可以用遞迴解
- 父節點
  1. 左子樹
  2. 右子樹



Example Code

Mission

LeetCode #226. Invert Binary Tree

Given the root of a binary tree, invert the tree, and return its root.

https://leetcode.com/problems/invert-binary-tree/

Tweet by Max Howell:

*Google: 90% of our engineers use the software you wrote (Homebrew), but you can't invert a binary tree on a whiteboard so f*** off.*

Practice

Mission

LeetCode #94. Binary Tree Inorder Traversal

Given the root of a binary tree, return the inorder traversal of its nodes' values.

Ref: https://leetcode.com/problems/binary-tree-inorder-traversal/

Practice

Mission

LeetCode #450. Delete Node in a BST

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST.

Basically, the deletion can be divided into two stages:

- Search for a node to remove.
- If the node is found, delete the node.

Follow up: Can you solve it with time complexity O(height of tree)?

Ref: https://leetcode.com/problems/delete-node-in-a-bst/

Practice

Mission

LeetCode #543. Diameter of Binary Tree

Given the root of a binary tree, return the length of the diameter of the tree.

The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

The length of a path between two nodes is represented by the number of edges between them.

Ref: https://leetcode.com/problems/diameter-of-binary-tree/