程式設計
Ch17. Trees

Chuan-Chi Lai 賴傳淇

Department of Communications Engineering
National Chung Cheng University

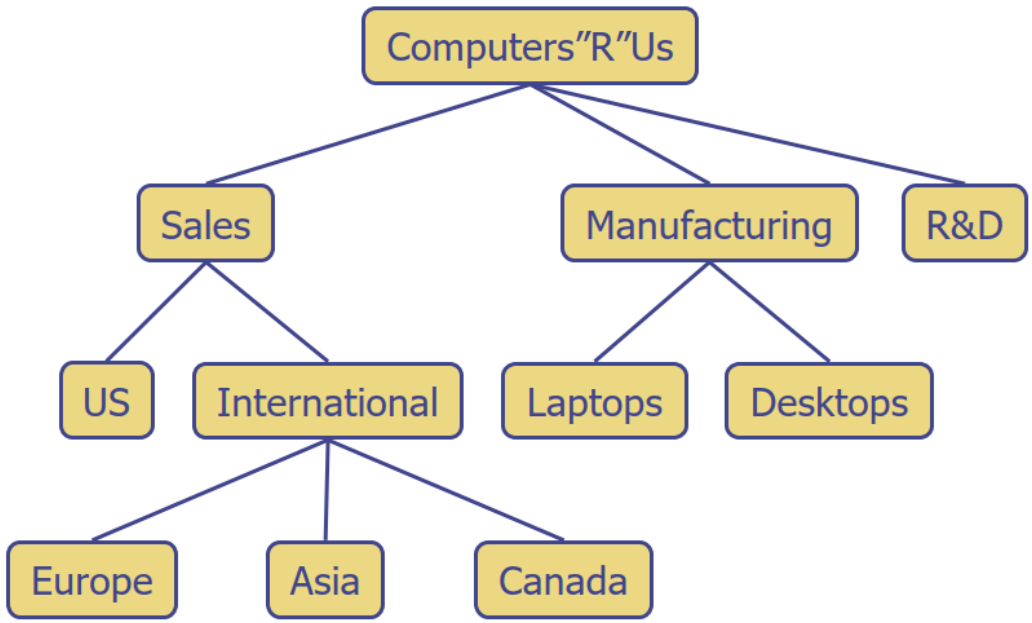
Spring Semester, 2024

Outline

樹是甚麼？
What is a Tree?

樹是甚麼？(What is a Tree?)

- 在電腦科學中，樹是層次 (hierarchical) 結構的抽像模型 (abstract model)。
- 樹由具有父子關係 (parent-child relation) 的節點組成。
- 常見相關應用：組織結構圖、檔案系統和程式設計環境等



樹的定義
Definition of Tree

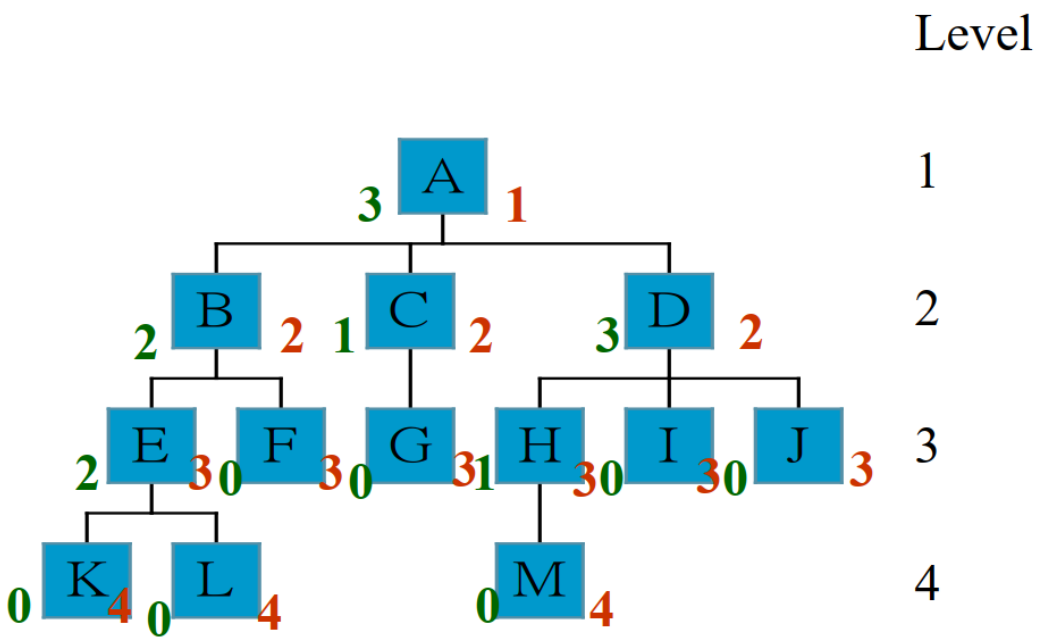
樹的定義

- 樹是一個或多個節點的有限集合，使得：
  - 存在且只有一個特定的節點，稱為根 (root)。
  - 其餘節點會被分割為 $n \geq 0$ 個不相交集合 $T_1, T_2, \ldots, T_n$，其中每個集合都是一棵樹。
  - $T_1, T_2, \ldots, T_n$ 被稱為根的子樹 (sub-trees)。
- 上述定義隱含：
  - 樹不可為空。
  - 子集合 (子樹) 間沒有交集，即 (子) 樹中節點之間的關係不存在迴路 (circle)。

樹的術語
Terminology of Tree

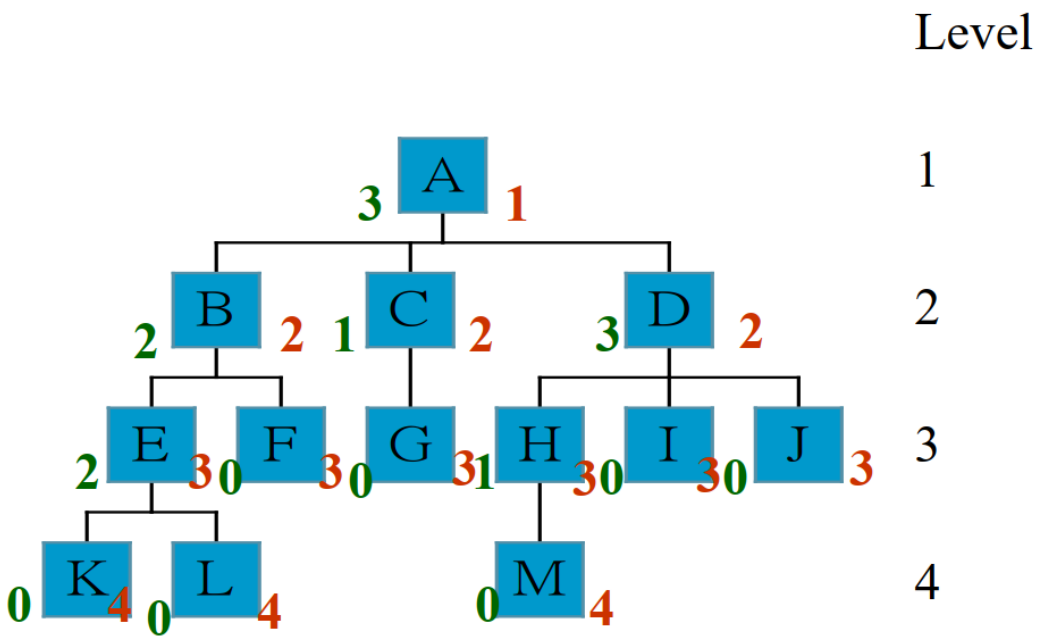
樹的術語 (Terminology of Tree)

- 針對 node/vertex：
  - 樹根 (root)：樹中最上層的 node，也是唯一一個其 parent 為 NULL 的 node。在下圖中，A 即為樹根。
  - 分歧度 (degree)：一個 node 擁有的子樹 (sub-tree) 的個數。下圖中，綠色數字為該節點的分歧度。



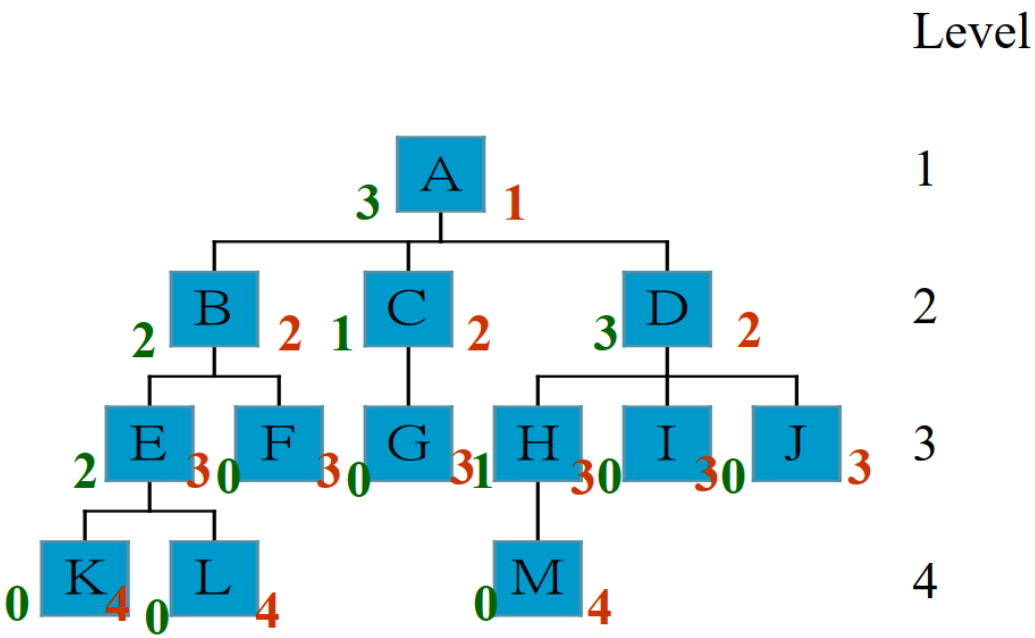
樹的術語 (Terminology of Tree)

- 針對 node/vertex：
  - 葉節點 (leaf)：沒有 child/subtree 的 node 稱為 leaf node。例如下圖中的 K、L、F、G、M、I 和 J。
  - 外部節點 (external node)：沒有 child 的 node。因此，leaf node 與 external node 同義。
  - 內部節點 (internal node)：至少有一個 child 的 node，稱為 internal node。例如下圖中的 A、B、C、D、E 和 H。



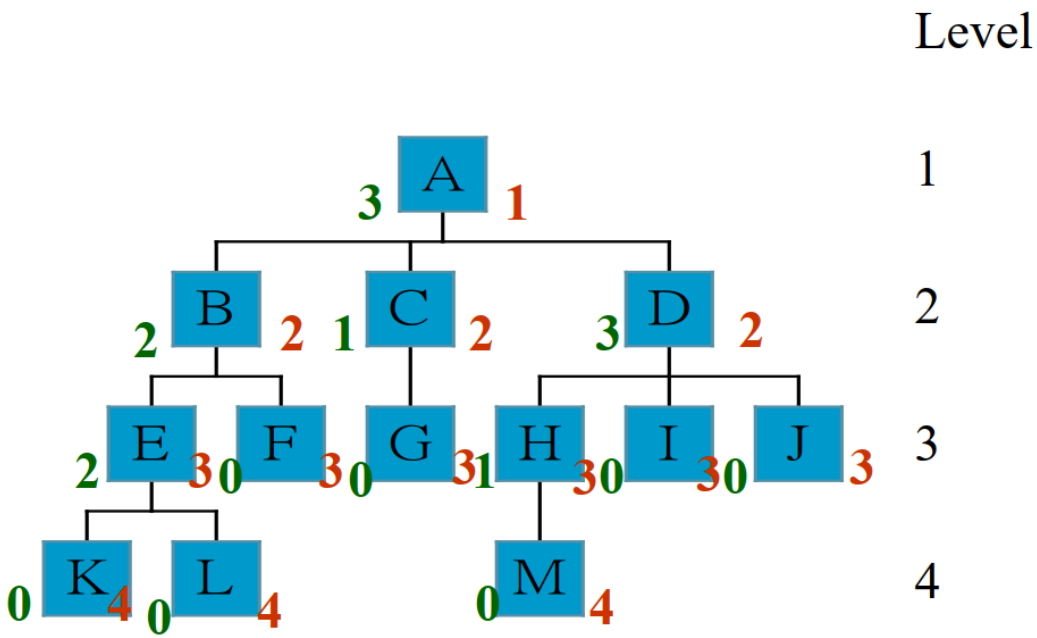
樹的術語 (Terminology of Tree)

- 針對樹：
  - 父子關係 (parent <–> child)：以 pointer 說明，被指向者 (pointed) 為 child，指向者 (point to) 為 parent。例如圖中的 A 為 B 的 parent，B 為 A 的 child，E 為 L 的 parent、L 為 E 的 child。
  - 兄弟姊妹 (siblings)：擁有相同 parent 的 node 們，互相稱兄道弟。例如圖中的 H、I、J 共同的 parent 為 D，那麼 H、I、J 即為彼此的 sibling。



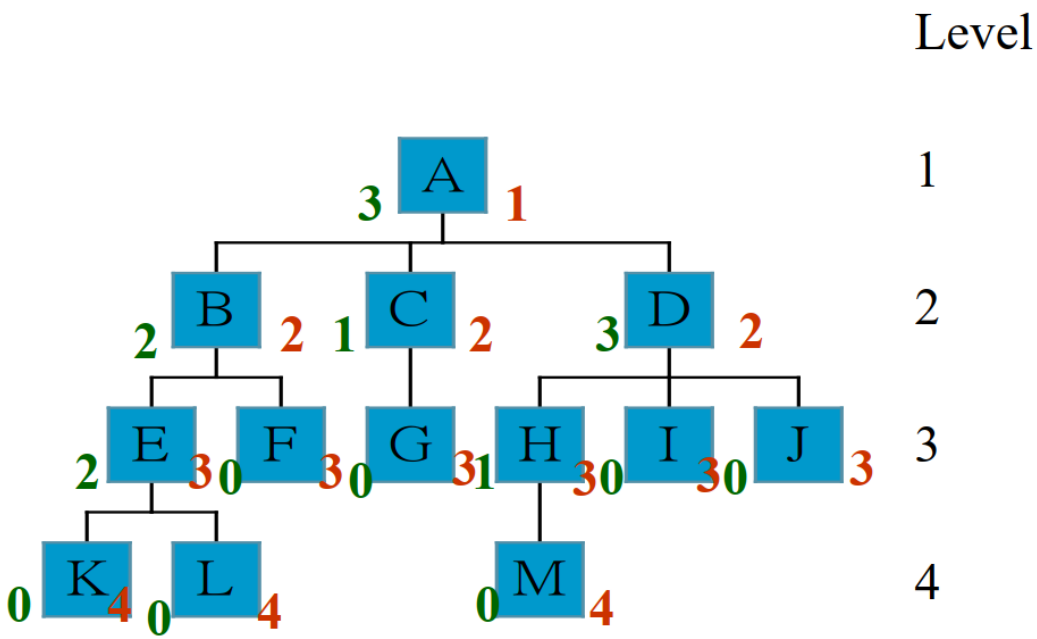
樹的術語 (Terminology of Tree)

- 針對樹：
  - 子嗣 (descendant)：圖中，站在 A，所有能夠以「parent 指向 child」的方式找到的 node，皆稱為 A 的 descendant，因此整棵樹除了 A 以外皆為 A 的 descendant。
  - 祖先 (ancestor)：圖中，站在 K，所有能夠以「尋找 parent」的方式找到的 node，皆稱為 K 的 ancestor，因此，E、B、A 皆為 K 的 ancestor。



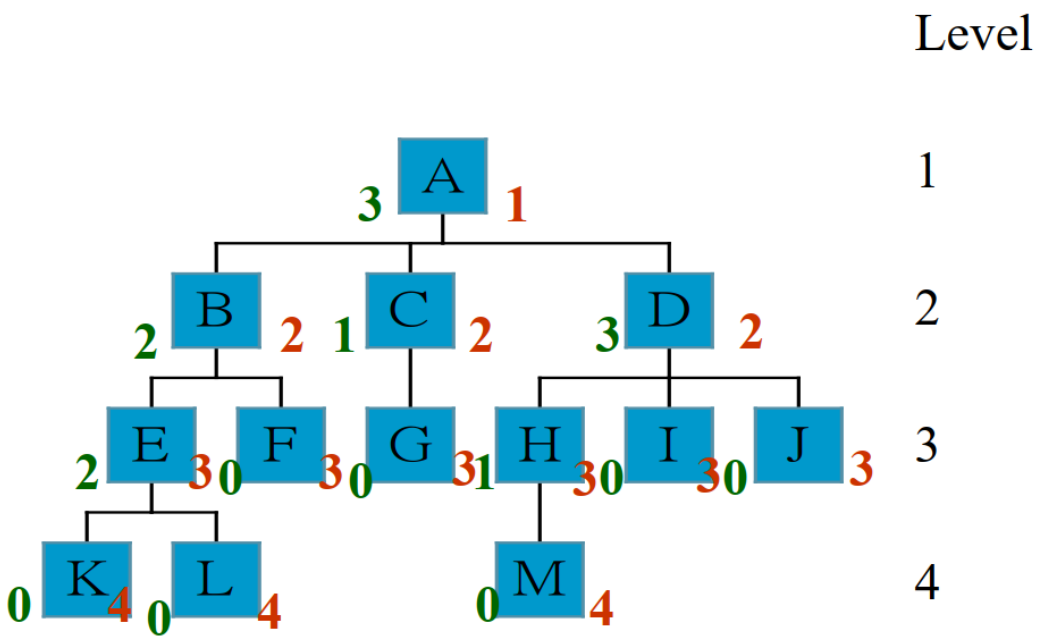
樹的術語 (Terminology of Tree)

- 針對樹：
  - 路徑 (path)：由 descendant 與 ancestor 關係連結成的 edge，例如 A-B-E-K、A-C-G。
  - level：定義 root 的 level 為 1，其餘 node 的 level 為其 parent 的 level 加一。圖中的紅色數字。



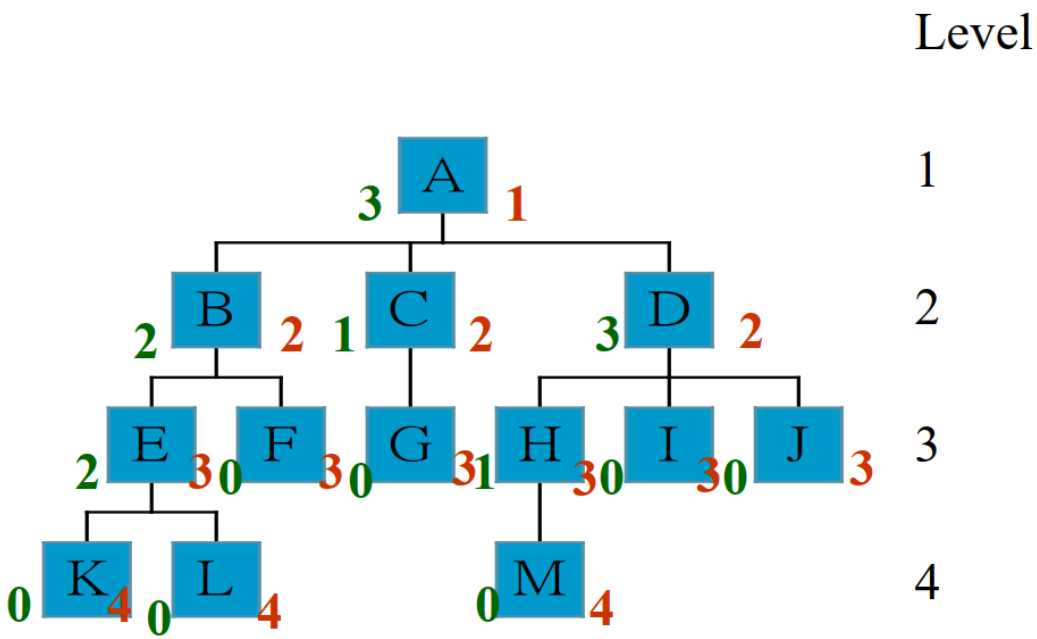
樹的術語 (Terminology of Tree)

- 針對樹：
  - 路徑 (path)：由 descendant 與 ancestor 關係連結成的 edge，例如 A-B-E-K、A-C-G。
  - 階層 (level)：定義 root 的 level 為 1，其餘 node 的 level 為其 parent 的 level 加一。圖中的紅色數字。
  - 深度 (depth)：某一 node 與 root 之間的節點數。例如 F 的深度為 3，M 的深度為 4。



樹的術語 (Terminology of Tree)

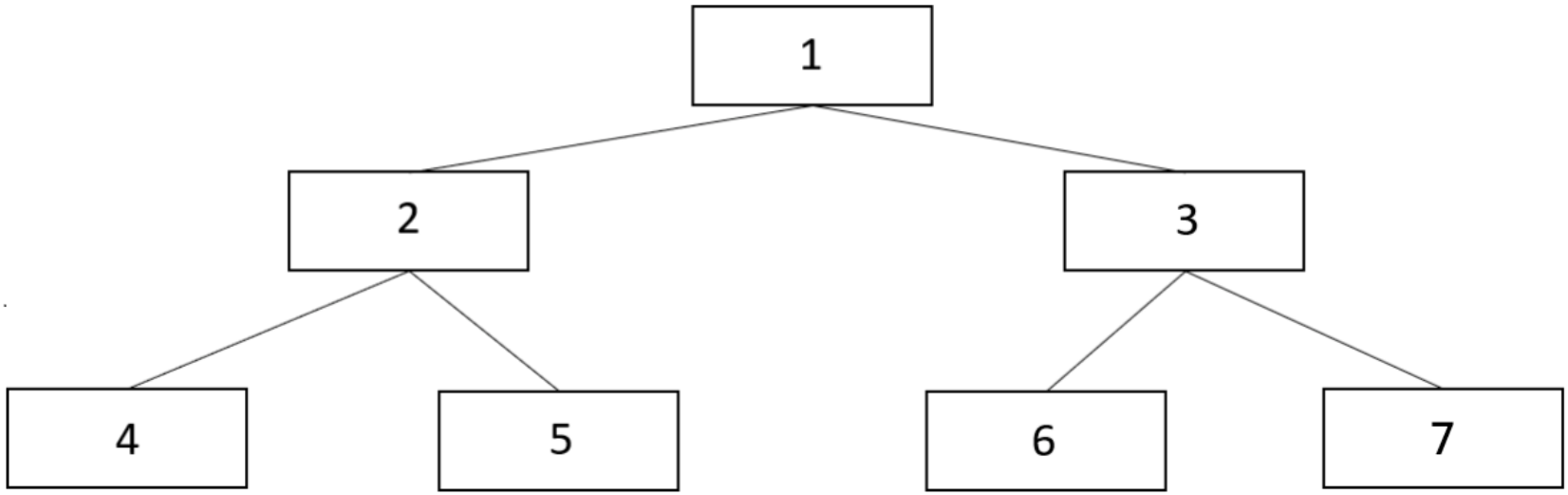
- 針對樹：
  - 節點高度 (height of node)：某一 node 與其最長 path上之 descendant leaf node 之間的節點數。
  - 樹高 (height of tree)：樹的 height 即為 root 的 height。下圖的樹高為 4。



二元樹
Binary Tree

二元樹

- 二元樹是一種樹狀結構，其中每個節點最多有兩個子節點，通常稱為左子樹和右子樹。
- 二元樹的節點可以用來表示數據，它可以用於快速查找、排序、數據壓縮等領域。



二元樹定義
Definition of Binary Tree

二元樹定義 (Definition of Binary Tree)

二元樹定義

- 二元樹可以為空集合
- 二元樹是有 ≥ 0 個 nodes 所構成的有限集合，若不為空集合，則為一個根節點和兩個不相交的二元樹 (稱為左子樹和右子樹) 組成。
- 所有子樹 (不論左右) 皆為二元樹。
- 樹中每 1 個節點最多只有 2 個子節點，左節點與右節點。

二元樹定義 (Definition of Binary Tree)

- 樹 (tree) 與二元樹 (binary tree) 的比較：

樹	**二元樹**
不可以為空	可以為空 (空集合)
Node's degree ≥ 0 即可	$0 \leq$ Node's degree ≤ 2
子樹之間無次序/方向之分	子樹有左右之分

二元樹的基本定理
Fundamental Theorem of Binary Tree

二元樹的基本定理

定理一

令 root level 為 1。二元樹中，第 $i \geq 1$ 個 level 的 node 個數最多有 2^{i-1} 個。

證明 (數學歸納法)

1. 當 level $= 1$ 時，最多只有 root 一個點，符合 $2^{1-1} = 2^0 = 1$，初值成立。
2. 令 level $= i-1$ 時，此定理成立。
3. 當 level $= i$ 時，其最多 Node 數必定是 $=$ (第 $i-1$ 個 level 之最多 Node 數 $)\times 2 = 2^{(i-1)-1} \times 2 = 2^{i-1}$。

   因此，由數學歸納法得證。

二元樹的基本定理

定理二

令 root level 為 1。高 (深) 度為 $h \geq 1$ 的二元樹，其 node 個數最多有 $2^h - 1$ 個 ($n_{\max} = 2^h - 1$)。

證明

$$\sum_{i=1}^{h} (\text{level } i \text{ 之 Node 數}) = 2^0 + 2^1 + \cdots + 2^{h-1}$$

$$= \frac{2^{(h-1)+1} - 2^0}{2 - 1}$$

$$= 2^h - 1 \qquad \square$$

二元樹的基本定理

定理三

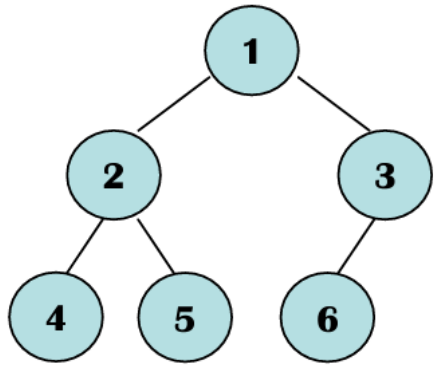
令 root level 為 1。非空二元樹若 leaf 個數為 n_0 個，degree 為 2 的 node 個數為 n_2 個，則 $n_0 = n_2 + 1$。

證明

假設 n 代表 Node 總數，n_i 代表 degree $= i$ 之 Node 數，B 代表分支 (Branch) 總數。

$$
\begin{aligned}
n &= n_0 + n_1 + n_2 = B + 1 = (n_1 + 2 \times n_2) + 1 \\
&\rightarrow n_0 + n_1 + n_2 = (n_1 + 2 \times n_2) + 1 \\
&\rightarrow n_0 = n_2 + 1
\end{aligned}
$$

□



課堂練習

- 若有 15 個 leafs，則 degree 為 2 的 node 數 = ?
- 若有 10 個 degree 為 2 的 nodes，則 leaf 個數 = ?
- 若二元樹有 53 個 nodes，其中 degree 為 1 的 node 數有 22 個，則 leaf 個數 = ?

Sol:

$$n = 53 = n_0 + n_1 + n_2$$

$$= n_0 + 22 + n_2$$

$$\therefore n_0 + n_2 = 53 - 22 = 31 \ldots ①$$

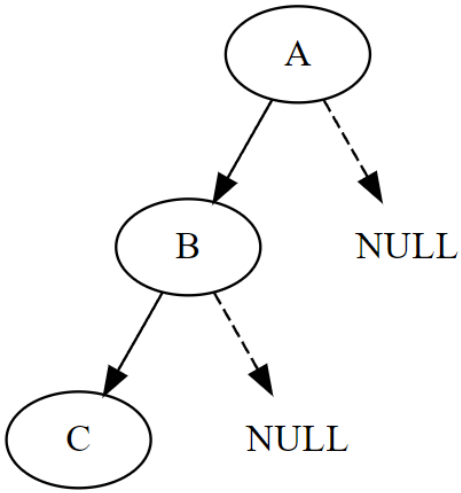
又: $n_0 = n_2 + 1 \ldots ②$

$$\Rightarrow n_0 = 16$$

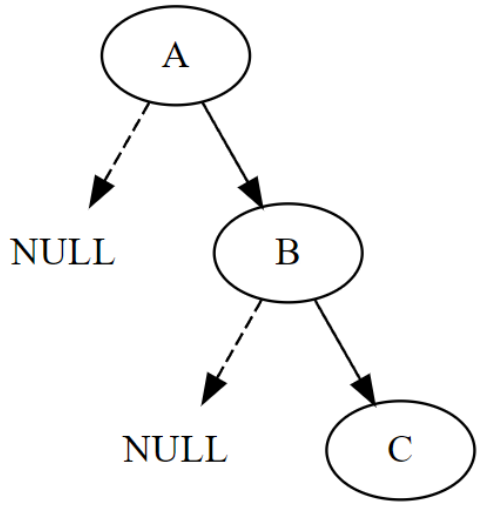
二元樹種類
Kinds of Binary Tree

歪斜樹 (Skewed Binary Tree)

- 若一棵樹的節點的子節點都為左節點或都為右節點，則可以稱此樹為歪斜樹。
- 此類型的 tree 之高度 h 剛好為節點個數 n。



Left-skewed B.T.

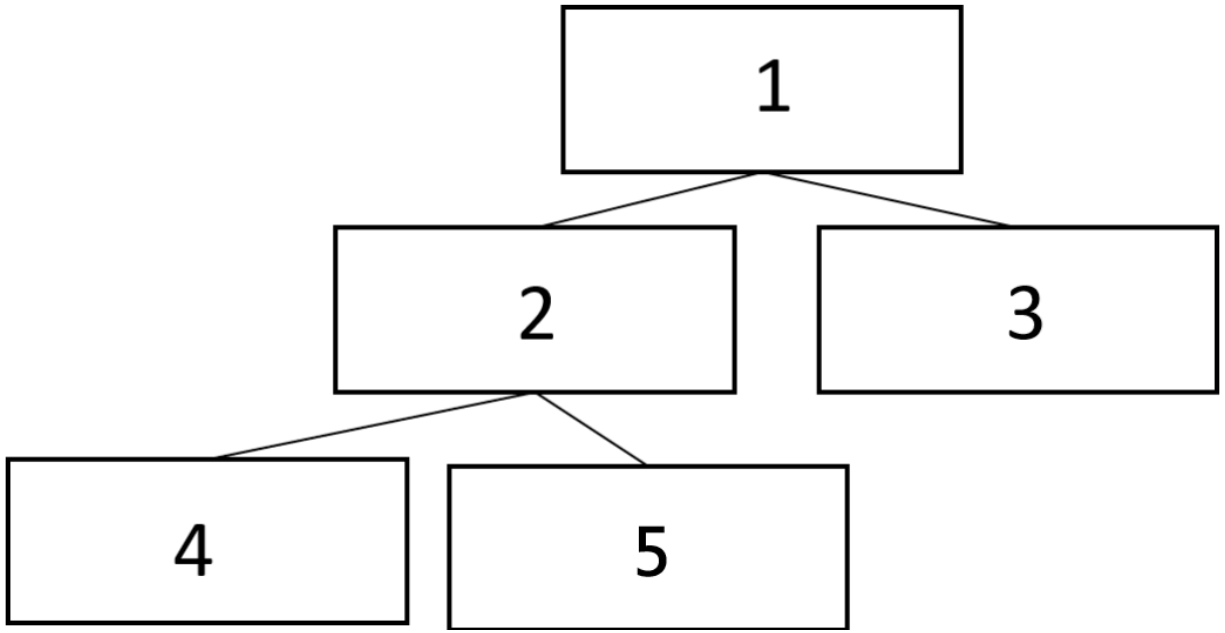


Right-skewed B.T.

二元樹種類 (Kinds of Binary Tree)

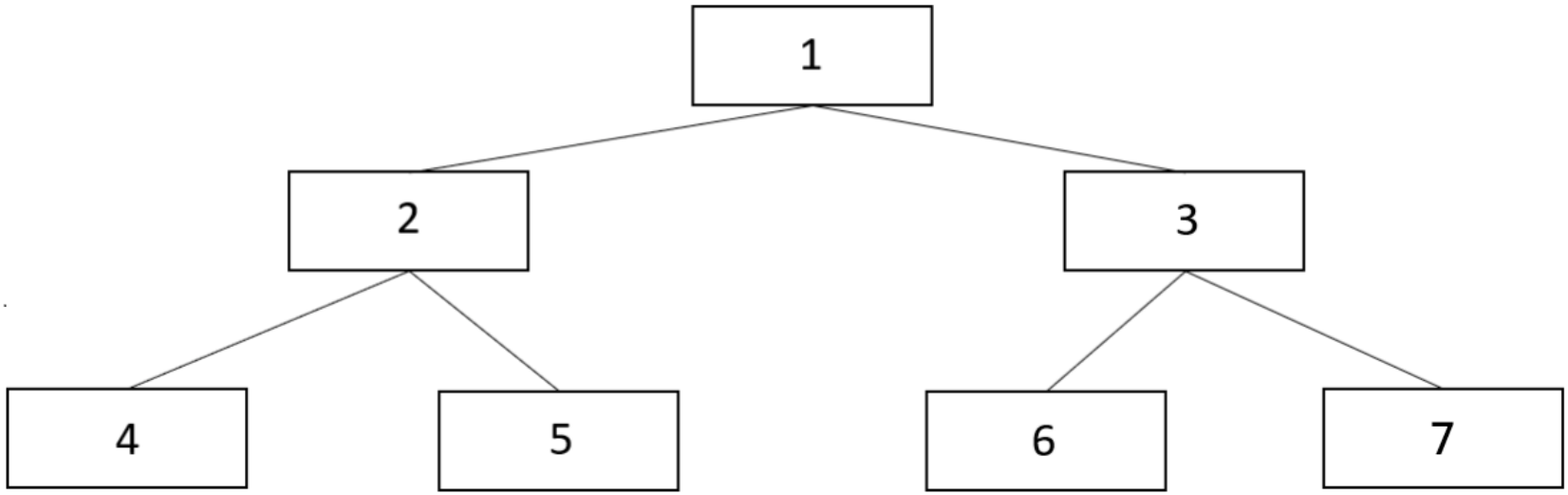
完滿二元樹 (Full Binary Tree)

- 除了葉節點以外的節點都有左右兩個子節點的樹為完滿二元樹。



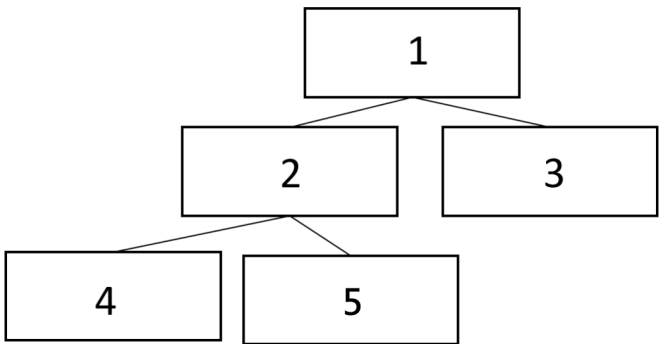
完美二元樹 (Perfect Binary Tree)

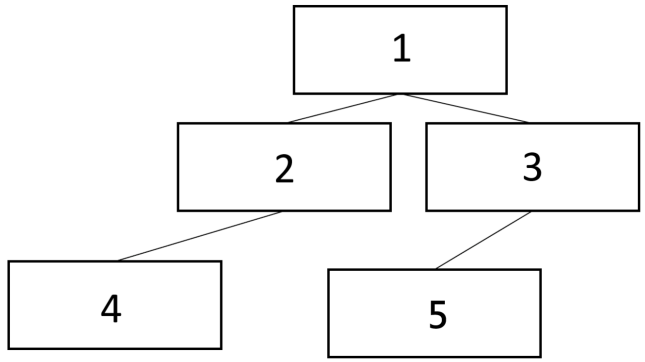
- 葉節點都在同一階層的情況下，除了葉節點以外的節點都有左右兩個子節點的樹為完美二元樹。
- 與其他二元樹相比，在相同樹高下，完美二元樹的擁有最多的節點數。
- 若數高為 h，完美二元的節點數量為 $2^h - 1$。



完整二元樹 (Complete Binary Tree)

- 定義：若一棵二元樹高度為 h，節點個數為 n，則滿足
  1. $2^{h-1} - 1 < n < 2^h - 1$
  2. n 個 node 之編號與高度 h 的完美二元樹之前的 n 個 node 編號一一對應，不能跳號。
- 白話文就是一棵樹的節點如果按照完美二元樹的次序排列 (由上至下，由左至右)，則稱此樹為完整二元樹。
- 舉例，左下的圖為完整二元樹，右下圖則為非。





二元樹種類 (Kinds of Binary Tree)

完整二元樹之定理

完整二元樹之定理

一個完整二元樹有 n 個節點編號:$1, 2, \ldots, n$，若某 Node 編號為 i，則

1. 其左子點編號：$2i$ (若 $2i > n$，則無左子點)
2. 其右子點編號：$2i+1$ (若 $2i+1 > n$，則無右子點)
3. 其父點編號：$\left[\frac{i}{2}\right]$ (取整數) (若 $\left[\frac{i}{2}\right] < 1$，則無父點)

證明在下一頁。

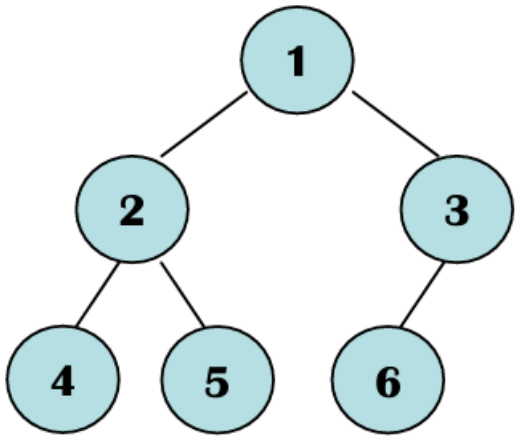
二元樹種類 (Kinds of Binary Tree)

完整二元樹定理之證明

(數學歸納法，只要證 (1),(2) 成立，再由 (1)(2)，(3) 自然得證)

1. 當 $i = 1$ 時，此點必為 root，而 root 若有左子點，必為 root 的下一個節點，因此編號為 2，滿足 $2 \times i = 2 \times 1 = 2$，初值成立。
2. 令編號 i—1 時，此定理成立。
3. 當編號 $= i$ 時，其左子點的編號必為編號 i—1 的節點的左子點編號 $+2$，得到 $2(i-1) + 2 = 2$。

因此，由數學歸納法得證。



二元樹的儲存
Storage of Binary Tree

二元樹的儲存 (Storage of Binary Tree)

- 在二元樹中，我們可以使用兩種方式來儲存，分別是：
  - 陣列
  - 鏈結串列

以陣列儲存
Store with Array

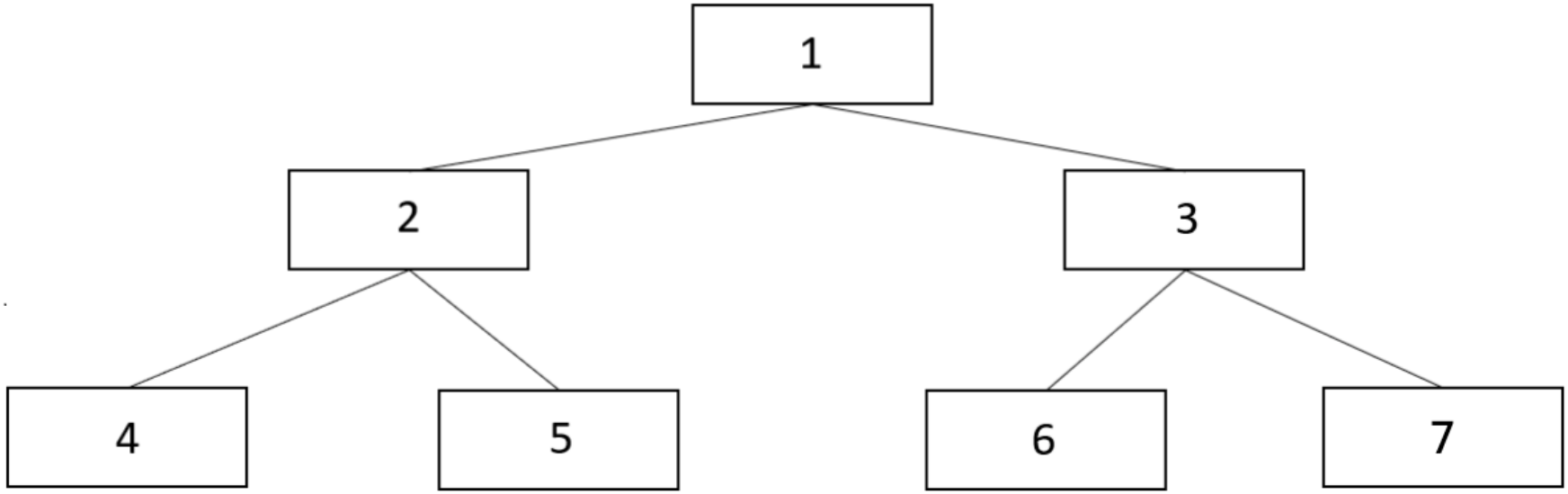
二元樹的儲存 (Storage of Binary Tree)

以陣列儲存 (Store with Array)

- 二元樹以陣列儲存時有以下的規則:
  - array[0] 不使用
  - array[1] 為樹根
  - array[i] 的左節點為 array[2*i]
  - array[i] 的右節點為 array[2*i+1]
- 可以發現上述規則跟隨完整二元樹之定理。

二元樹的儲存 (Storage of Binary Tree)

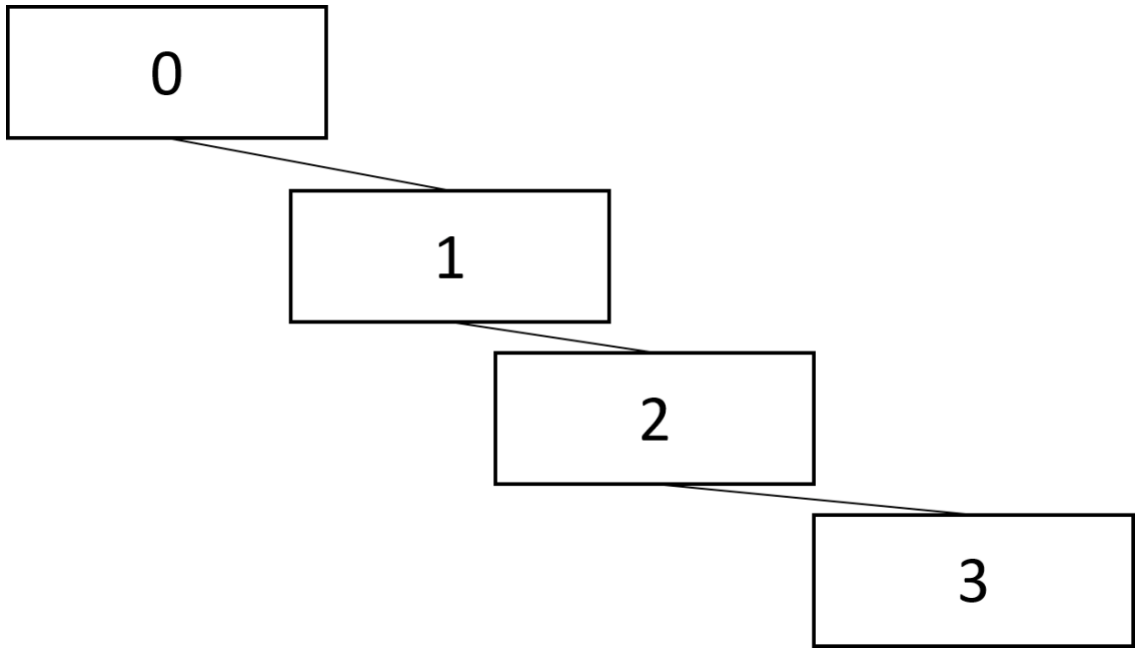
- 用一棵完美二元樹為範例：



index	0	1	2	3	4	5	6	7
value		1	2	3	4	5	6	7

二元樹的儲存 (Storage of Binary Tree)

- 用一棵歪斜樹為範例：



index	0	1	2	3	4	5	6	7
value		0		1				2

二元樹的儲存 (Storage of Binary Tree)

- 從以上兩個範例來看，我們可以很明顯地察覺出來完美二元樹非常適合用陣列來儲存。
- 而歪斜樹浪費了太多空間，甚至到了存不下去的地步，因此接下來就要介紹以鏈結串列儲存二元樹。

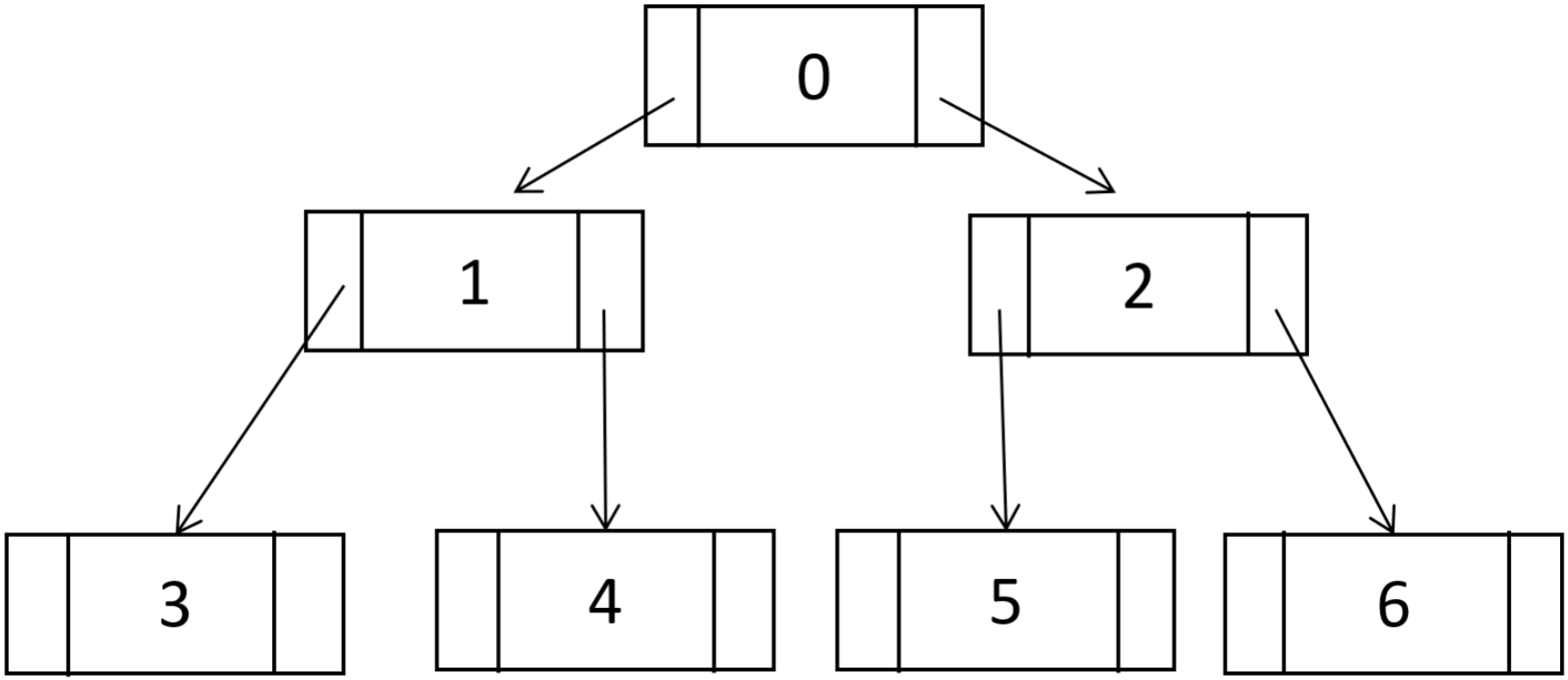
以鏈結串列儲存
Store with Linked List

二元樹的儲存 (Storage of Binary Tree)

- 用鏈結串列儲存二元樹就相對地比較直覺了，只要在節點中設立左指標及右指標分別指向左節點及右節點就可以了。

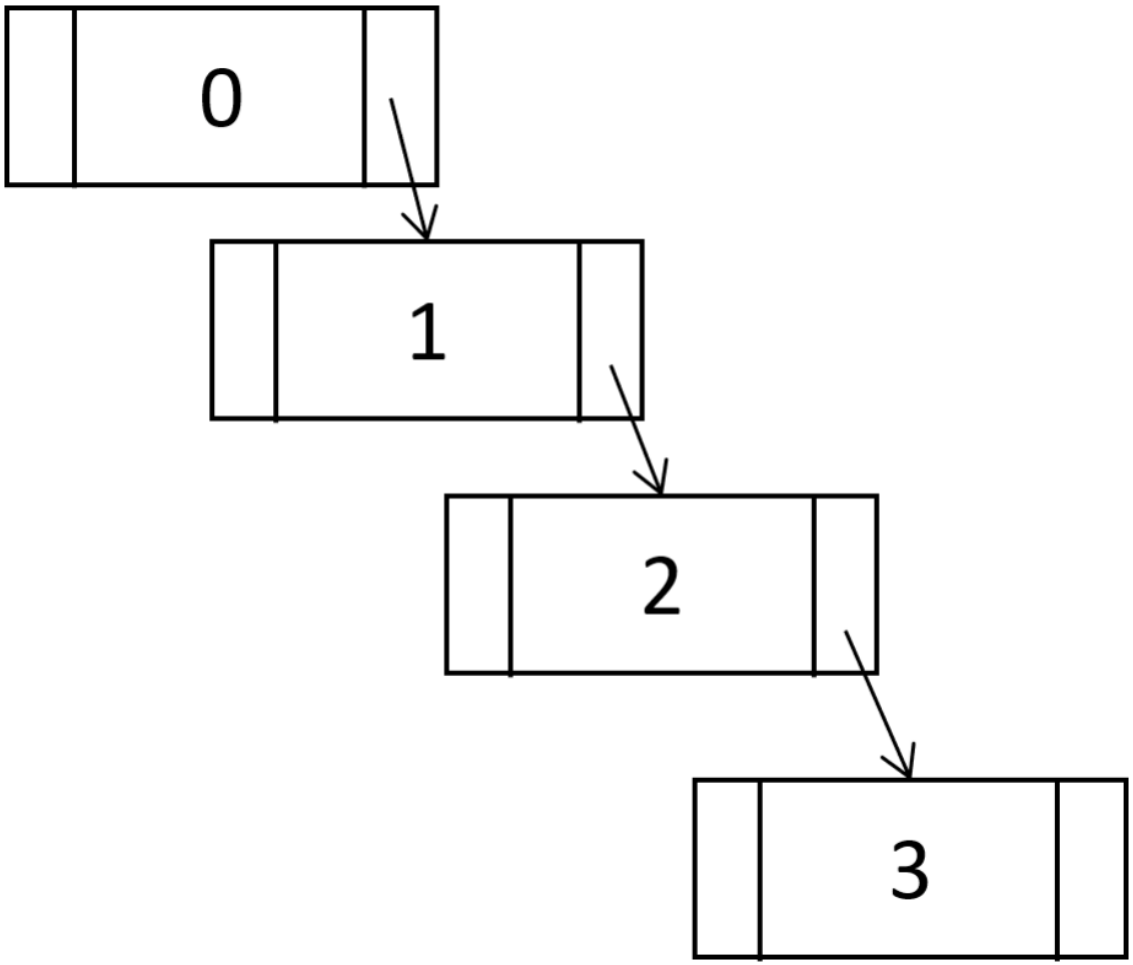
二元樹的儲存 (Storage of Binary Tree)

- 用先前同一棵完美二元樹為範例：



二元樹的儲存 (Storage of Binary Tree)

- 用先前同一棵歪斜樹為範例：



二元樹的儲存 (Storage of Binary Tree)

- 雖然用鏈結串列儲存歪斜樹會相對陣列來說表面上省了很多空間，但其實已鏈結串列也會浪費掉很多指標，例如節點的另一邊指標以及葉子的左右指標。

二元樹遍歷
Binary Tree Traversal

二元樹遍歷

- 在二元樹的操作中，遍歷是一個重要的概念。遍歷指的是按照某種特定的順序訪問二元樹的所有節點，以達到對樹結構的完整遍歷和搜尋。
- 下面即將介紹二元樹的三種遍歷方式：
  - 前序遍歷 (Pre-order)
  - 中序遍歷 (In-order)
  - 後序遍歷 (Post-order)

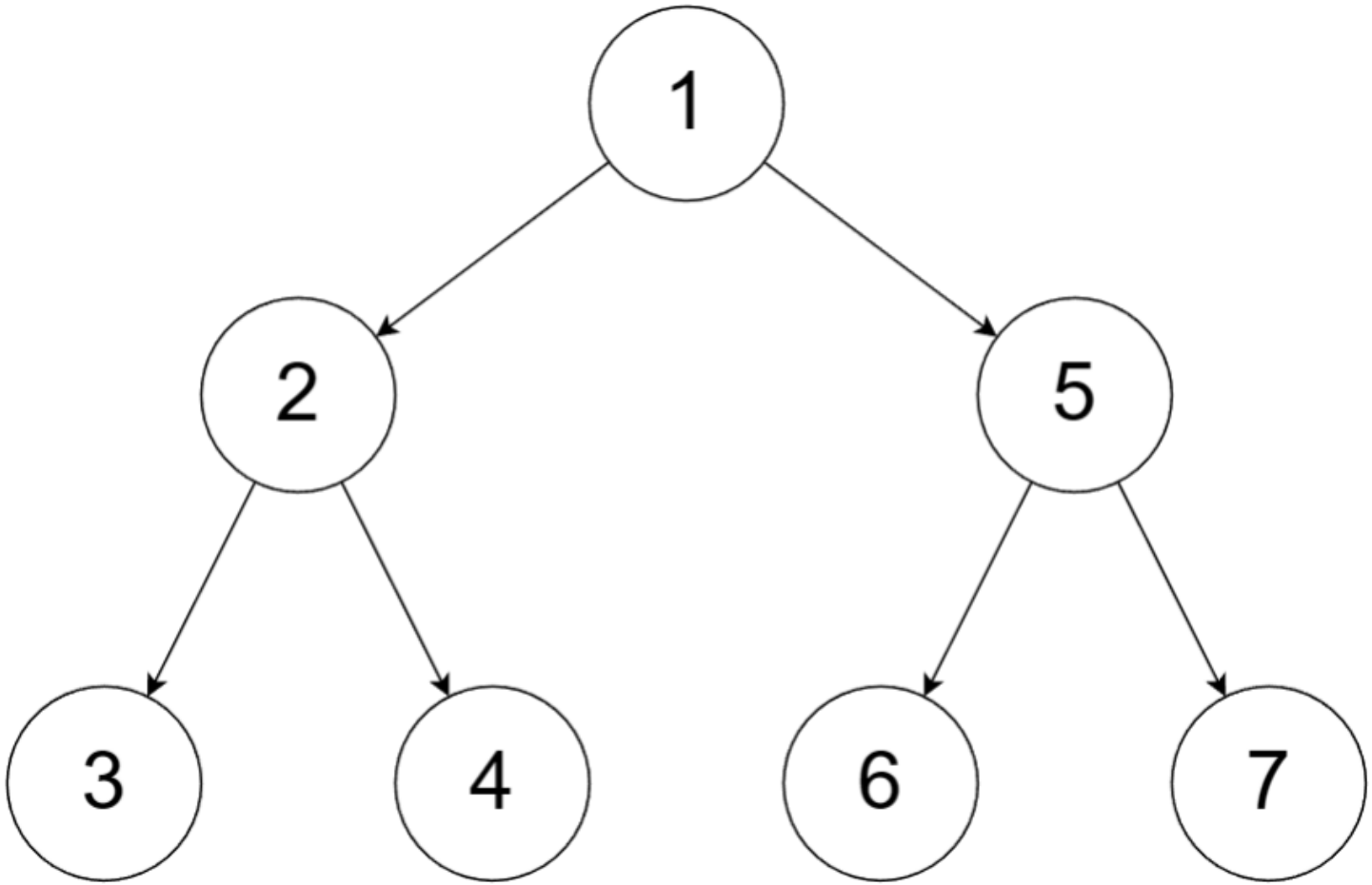
前序遍歷
Pre-order Traversal

前序遍歷

- 前序遍歷是指從根節點開始，按照「當前節點 -> 左子樹 -> 右子樹」的順序進行遍歷。具體的步驟如下：
  1. 檢查當前節點，如果為空則返回。
  2. 印出當前節點的值。
  3. 遞迴地遍歷當前節點的左子樹。
  4. 遞迴地遍歷當前節點的右子樹。

前序遍歷 (Pre-order Traversal)

- 示意圖如下 (數字代表前序遍歷走訪輸出順序)：



前序遍歷 (Pre-order Traversal)

- 前序遍歷的程式碼範例如下：

```
1 void preorderTraversal(struct Node* root) {
2     if (root != NULL) {
3         printf("%d ", root->data);
4         preorderTraversal(root->left);
5         preorderTraversal(root->right);
6     }
7 }
```

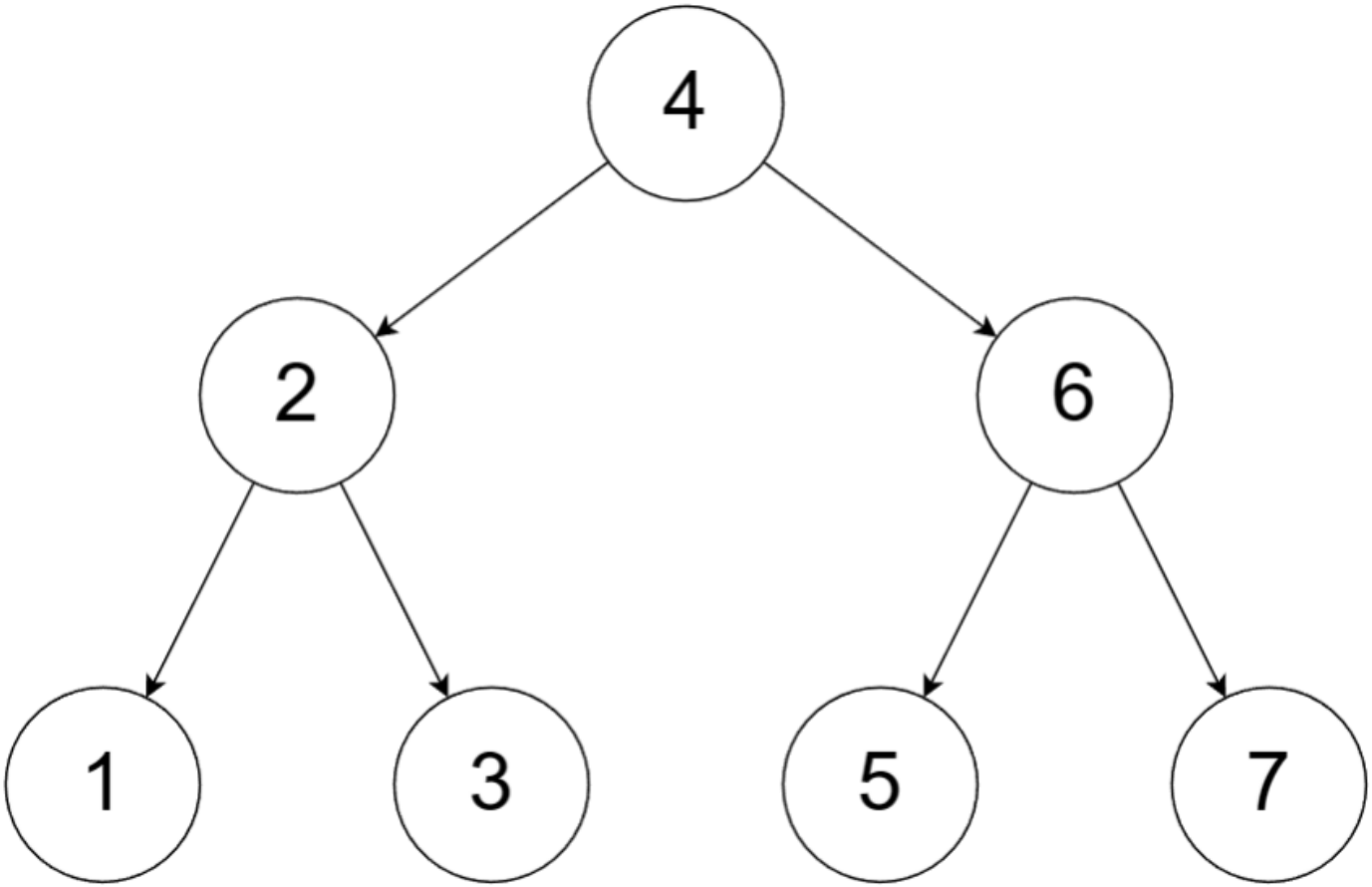
中序遍歷
In-order Traversal

中序遍歷

- 中序遍歷是指從根節點開始，按照「左子樹 -> 當前節點 -> 右子樹」的順序進行遍歷。具體的步驟如下：
  1. 檢查當前節點，如果為空則返回。
  2. 遞迴地遍歷當前節點的左子樹。
  3. 印出當前節點的值。
  4. 遞迴地遍歷當前節點的右子樹。

中序遍歷 (In-order Traversal)

- 示意圖如下 (數字代表中序遍歷走訪輸出順序)：



中序遍歷 (In-order Traversal)

- 中序遍歷的程式碼範例如下：

```
1 void inorderTraversal(struct Node* root) {
2     if (root != NULL) {
3         inorderTraversal(root->left);
4         printf("%d ", root->data);
5         inorderTraversal(root->right);
6     }
7 }
```

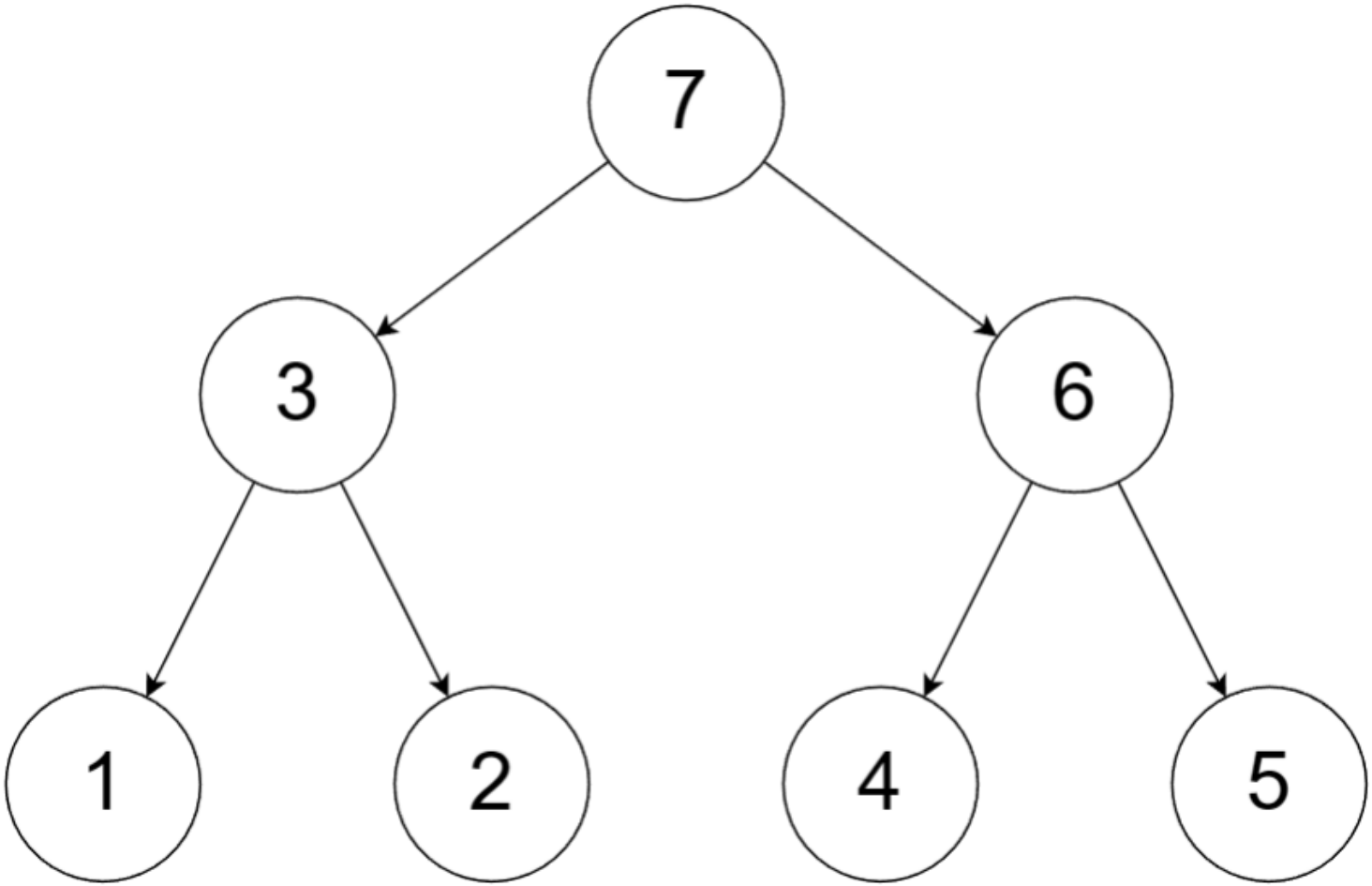
後序遍歷
Post-order Traversal

後序遍歷

- 後序遍歷是指從根節點開始，按照「左子樹 -> 右子樹 -> 當前節點」的順序進行遍歷。具體的步驟如下：
  1. 檢查當前節點，如果為空則返回。
  2. 遞迴地遍歷當前節點的左子樹。
  3. 遞迴地遍歷當前節點的右子樹。
  4. 印出當前節點的值。

後序遍歷 (Post-order Traversal)

- 示意圖如下 (數字代表後序遍歷走訪輸出順序)：



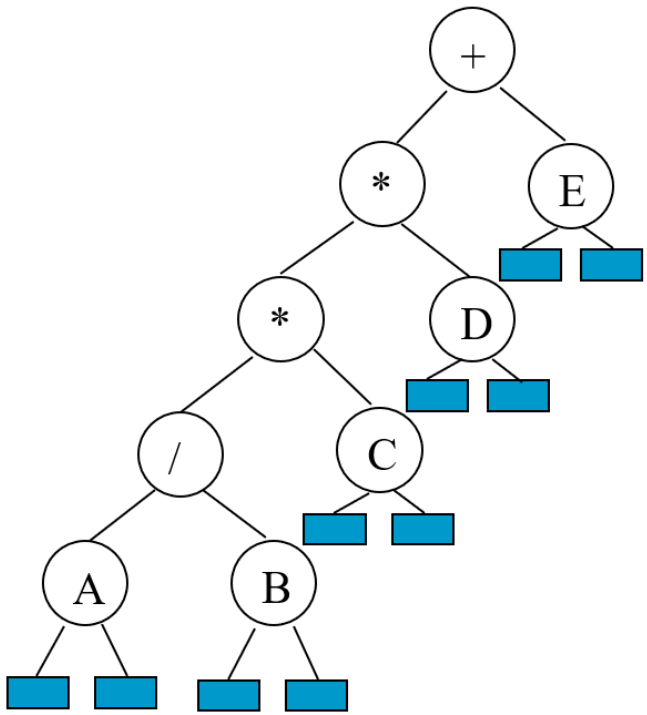
後序遍歷 (Post-order Traversal)

- 後序遍歷的程式碼範例如下：

```
1 void postorderTraversal(struct Node* root) {
2     if (root != NULL) {
3         postorderTraversal(root->left);
4         postorderTraversal(root->right);
5         printf("%d ", root->data);
6     }
7 }
```

二元樹遍歷應用範例

- 我們可以利用二元樹儲存算術表達式 (arithmetic expression)。
- 使用不同的遍歷函式，即可將算術表達式分別轉換為前序、中序和後序的表達式。



inorder traversal
A / B * C * D + E
infix expression
preorder traversal
+ * * / A B C D E
prefix expression
postorder traversal
A B / C * D * E +
postfix expression

二元樹的搜尋
Search of Binary Tree

二元樹的搜尋

- 講述完如何在二元樹上做遍歷後，藉由這個方式我們可以在二元樹上進行搜尋。
- 方法也很簡單，就是在原本輸出當前節點的地方，改成判斷節點是否為我們所要找的。

二元樹的搜尋 (Search of Binary Tree)

- 二元樹的搜尋程式碼範例如下：

```
1  struct Node* searchNode1(struct Node* root, int find) {
2      if (root == NULL || root->data == find) {
3          return root;
4      }
5
6      struct Node* leftResult = searchNode1(root->left, find);
7      if (leftResult != NULL) {
8          return leftResult;
9      }
10
11     struct Node* rightResult = searchNode1(root->right, find);
12     return rightResult;
13 }
```

Q & A