

資料結構



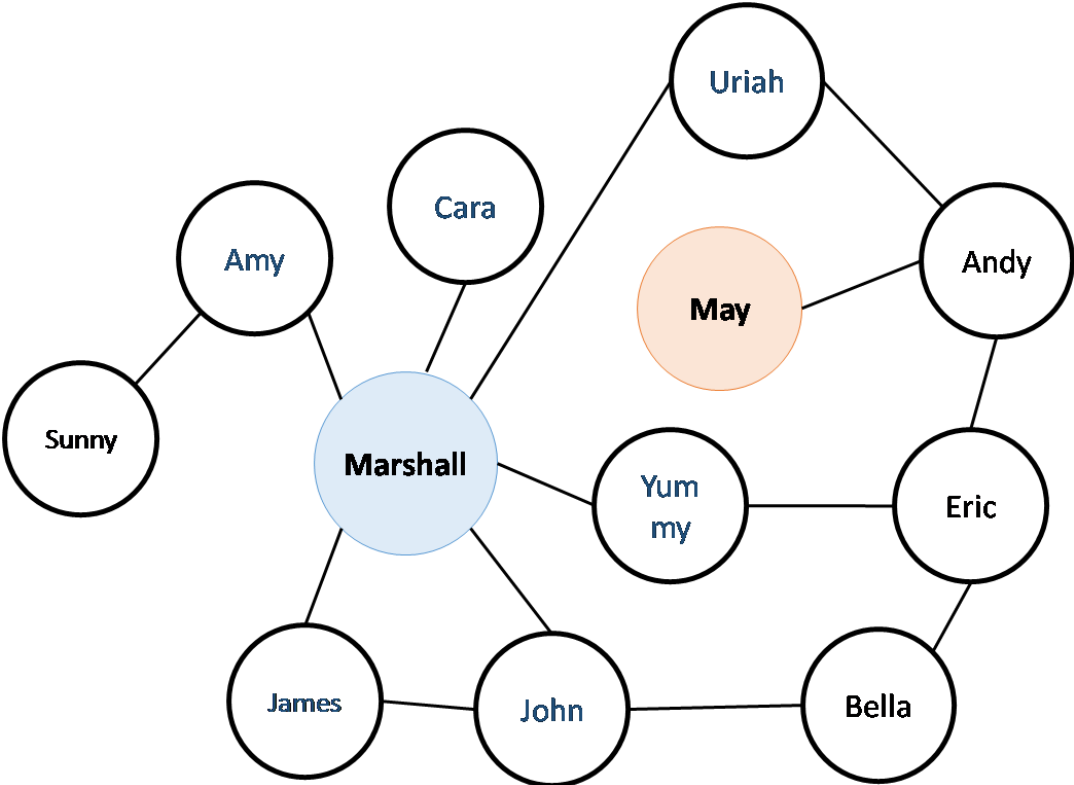
演算法

樹 Tree

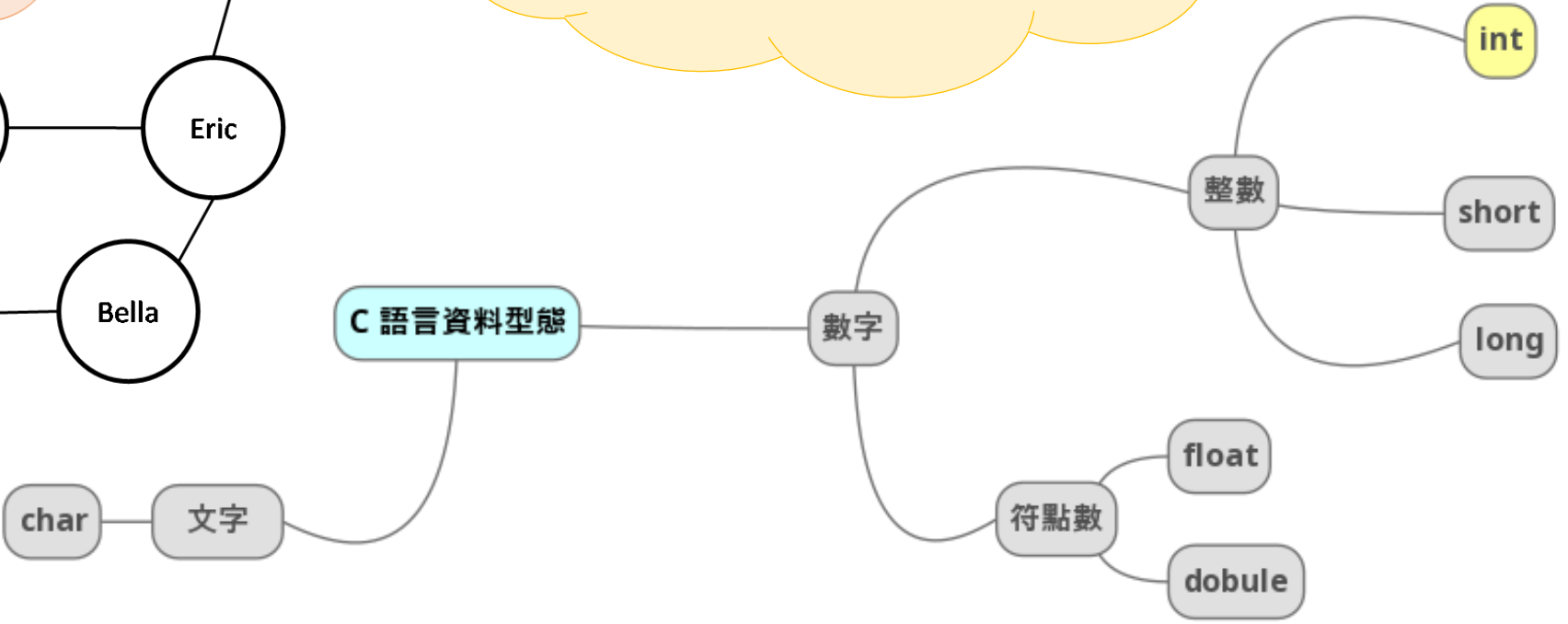
樹 Tree

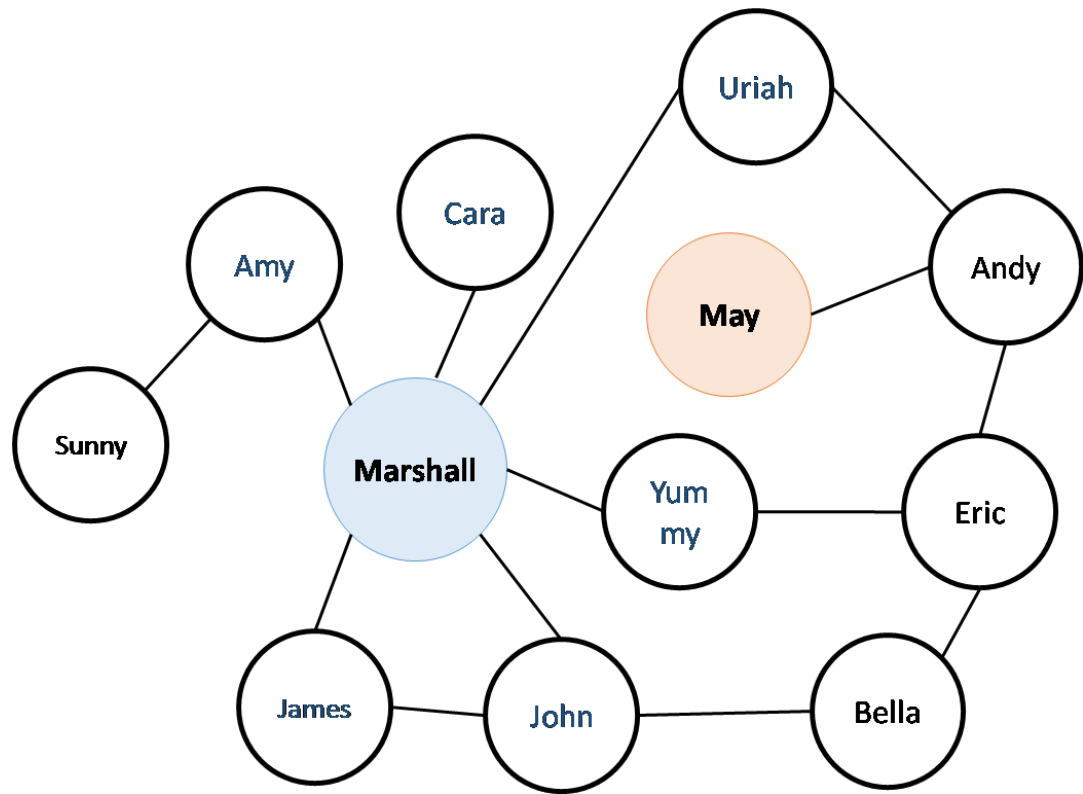


# 比較這兩個圖形結構



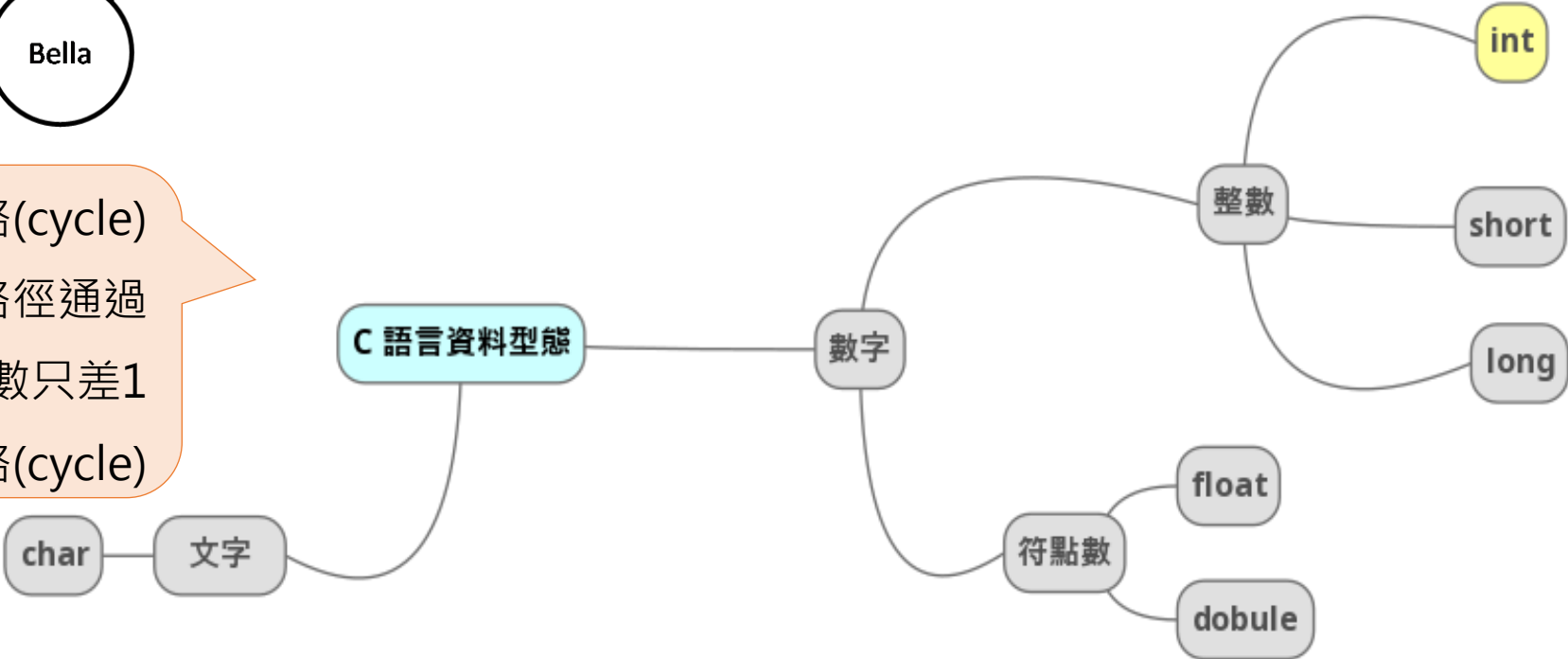
可以找出他們哪  
裡不一樣嗎？



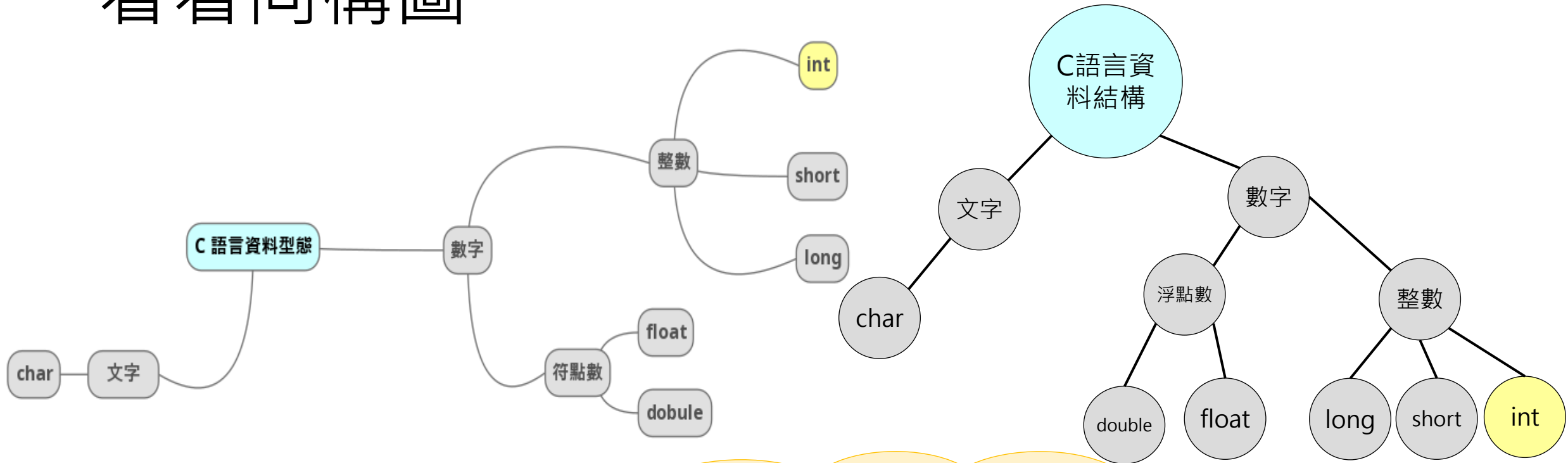


- 有迴路(cycle)
- 任兩個節點不只一條路徑
- 12個點有14個邊，邊數與點數沒有相對關係

- 沒有迴路(cycle)
- 任兩個節點只有一條路徑通過
- 11個點有10個邊，也就是邊數與點數只差1
- 任意再加上一個邊就會出現迴路(cycle)



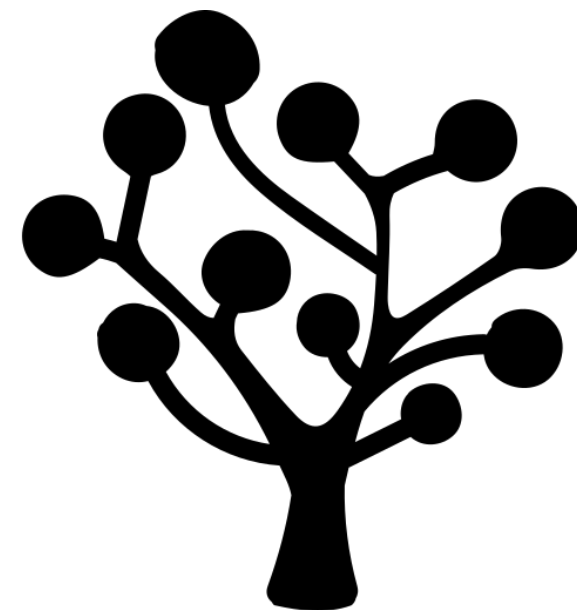
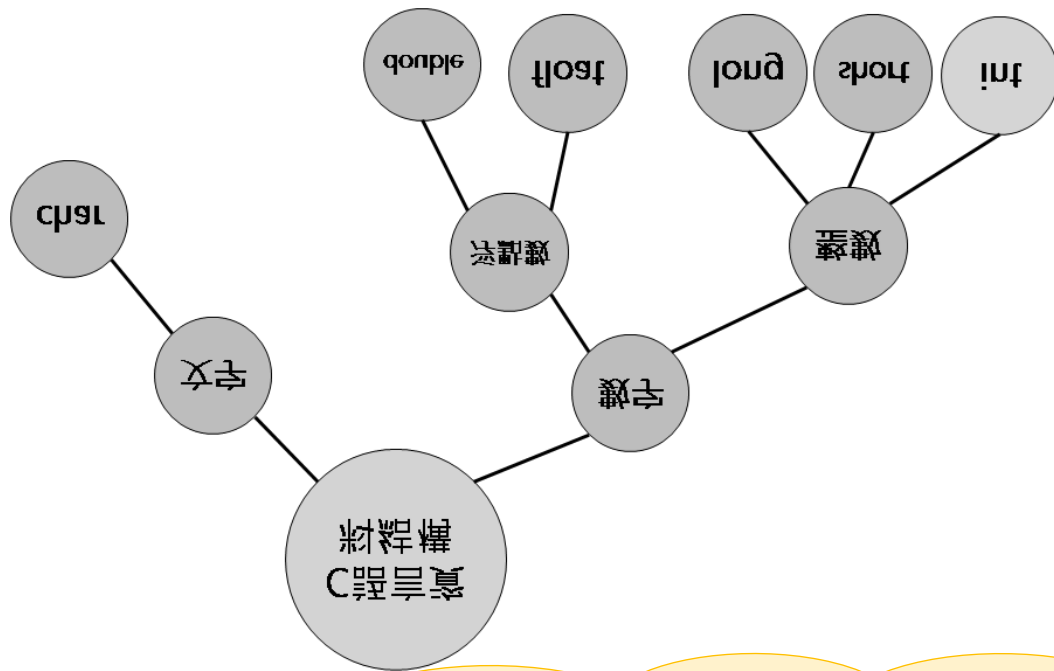
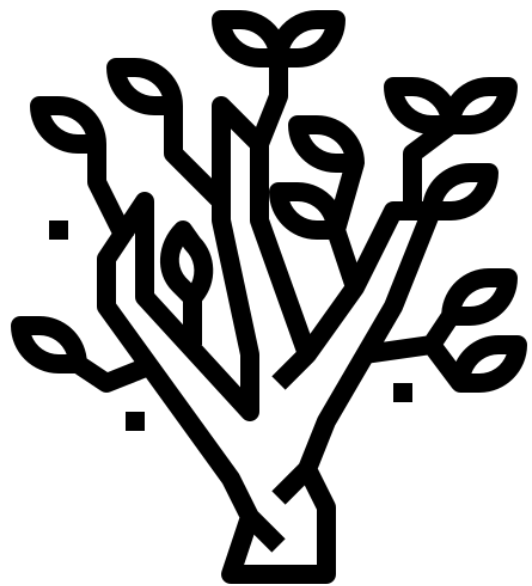
# 看看同構圖



將各點與各邊調整一下位置後，是不是能明確的看出各點之間的階層概念？

看這樣的階層圖是否有聯想到其他東西呢？

就像一顆『**樹 Tree**』的『**圖 Graph**』



是的，我們將這種看起來像樹的特殊圖形，另外給了一個稱呼：

『**樹 Tree**』！

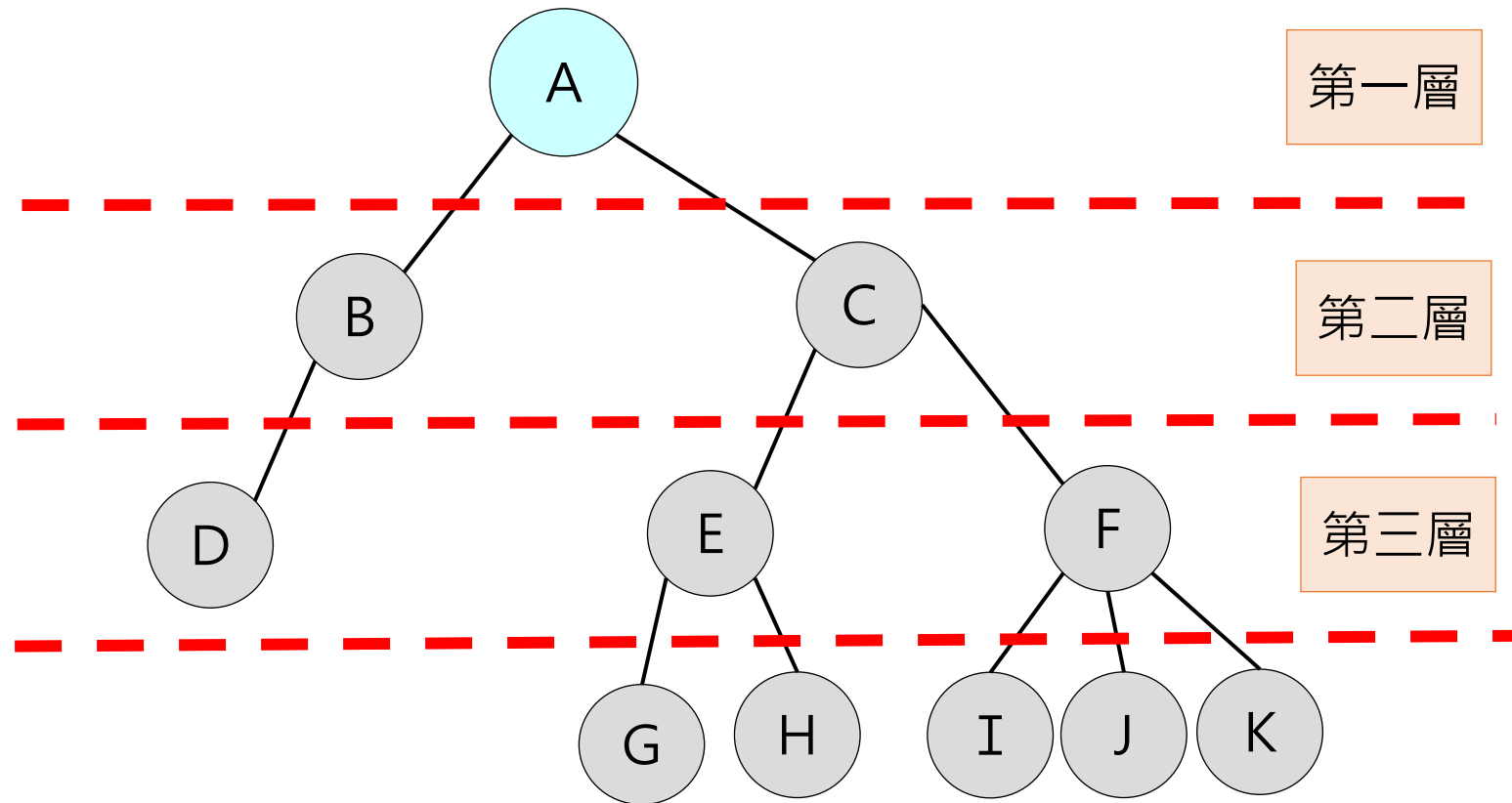
<樹 Tree>

基本概念



# 樹(Tree) 是？

- 一種有**階層架構**的**非線性資料集合**。

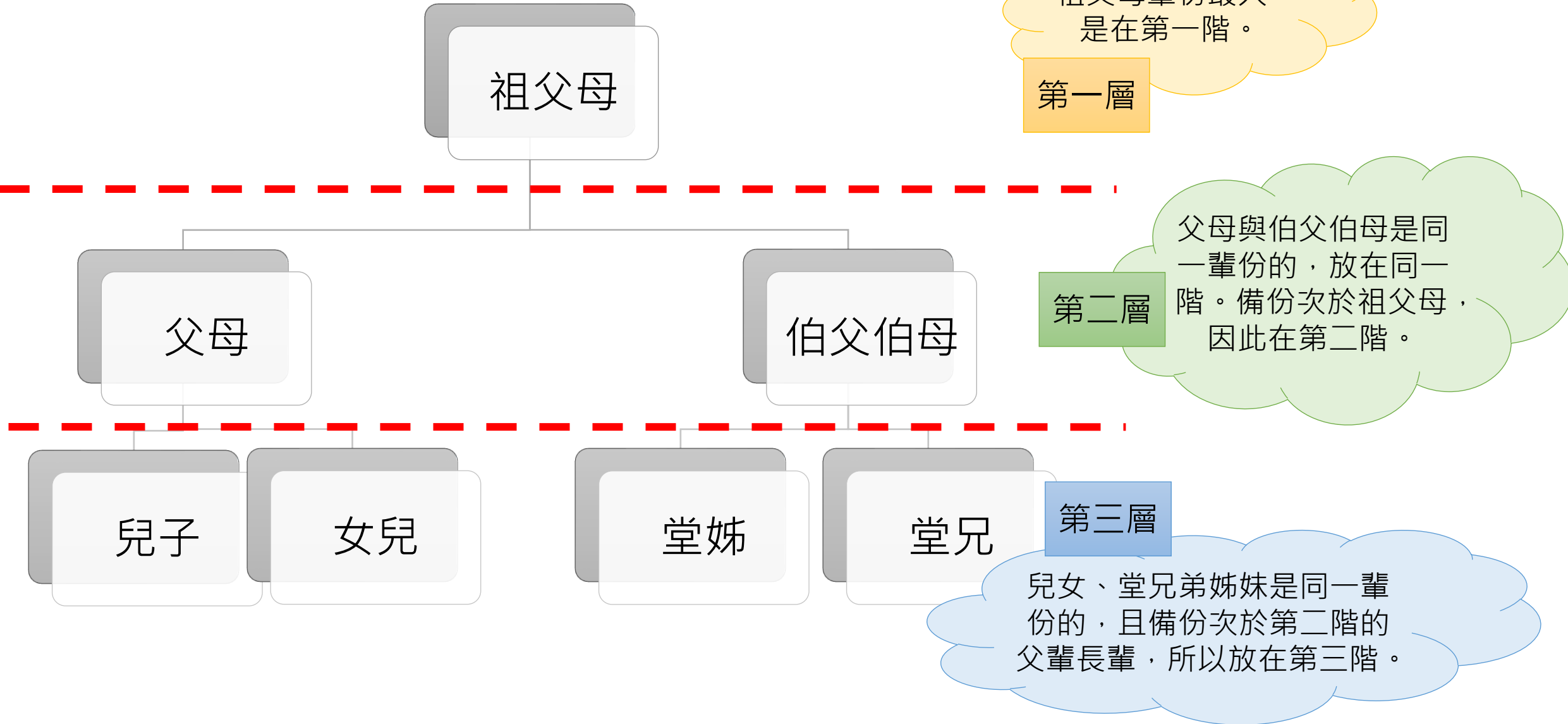


# 樹的定義

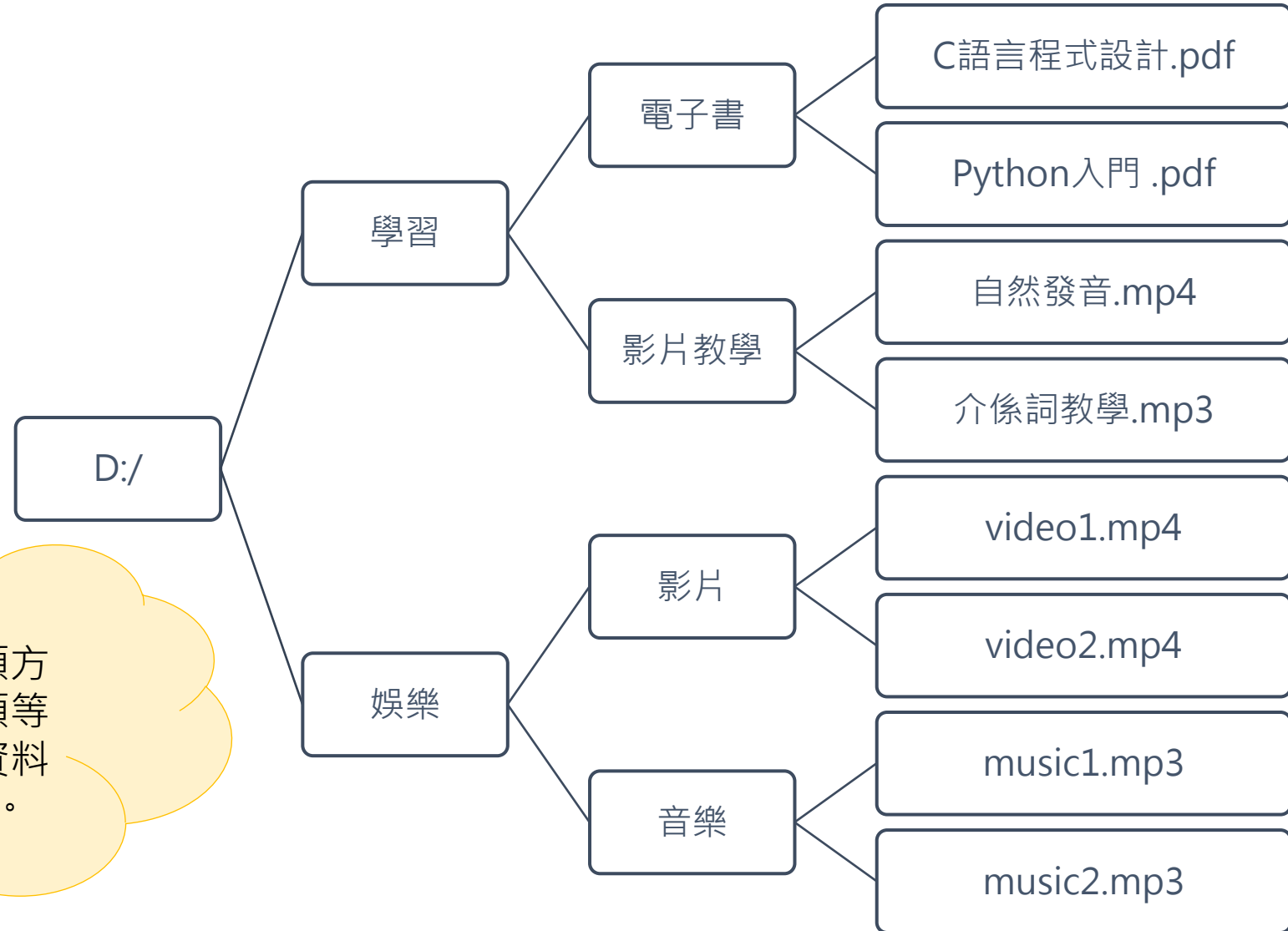
- 樹狀結構是由一個或多個節點組合的有限集合，包含以下特性：
  - 任兩個節點只有一條路徑通過
  - 沒有迴路(cycle)
  - 有  $N$  個節點，那就會有  $N-1$  個邊
  - 任意再加上一個邊就會出現迴路(cycle)



# 樹的應用 – 家族族譜



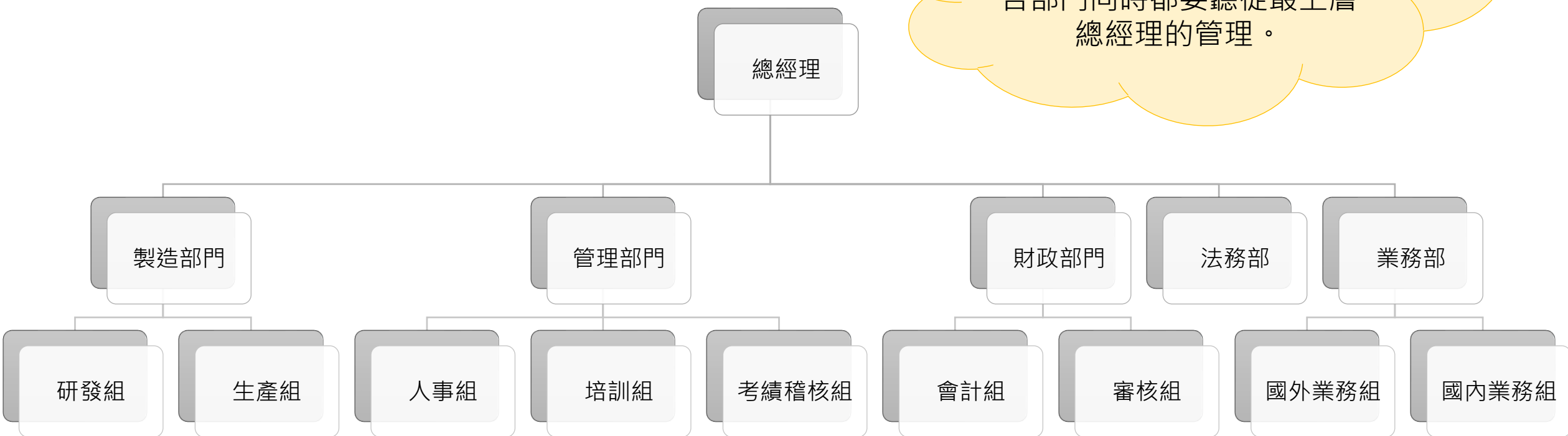
# 樹的應用 – 檔案目錄



不同使用者根據各種分類方式（檔案種類、時間分類等等），將電腦硬碟內的資料分成各種階層目錄存放。

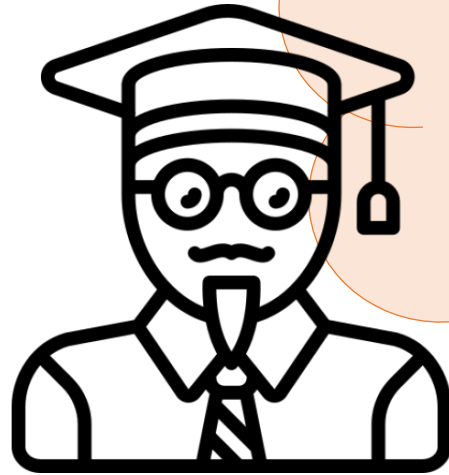
# 樹的應用 – 公司組織圖

公司內根據不同的工作項目會分成多個部門，每個部門又可以細分成不同組別，但各部門同時都要聽從最上層總經理的管理。



# 樹的應用

- 家族族譜
- 檔案目錄
- 公司組織圖
- if else 決策圖
- 技術學習圖
- ...



樹狀結構雖然也可以視為是一種圖形結構，但因為樹的特殊定義，因此另外有專屬於樹的專有名詞。

再探究樹的其它應用前，我們同樣要先來認識樹的專有名詞。

<樹 Tree>

專有名詞



# 樹的專有名詞

|                      |                            |               |                    |
|----------------------|----------------------------|---------------|--------------------|
| 節點<br>node           | 兄弟節點<br>sibling node       | 分支度<br>branch | 子樹<br>subtree      |
| 邊<br>edge            | 祖先節點<br>ancestor node      | 階度<br>level   | 二元樹<br>binary tree |
| 樹根節點<br>root         | 子孫節點<br>descendant node    | 高度<br>height  |                    |
| 父節點<br>parent node   | 非終點節點<br>non-terminal node | 深度<br>depth   |                    |
| 子節點<br>children node | 終點節點<br>terminal node      | 樹林<br>forest  |                    |

# 節點與邊

- 節點 node
  - 樹上的每個點都稱為節點
  - 每個節點都是一項資料
- 邊 edge
  - 節點通往其它節點的路

# 各種節點 (1)

- **樹根節點 root**
  - 樹的頂端(最上層)節點
  - 每顆樹都有唯一一個 root
- **父節點 parent node**
  - 一個節點的上一層節點
- **子節點 children node**
  - 一個節點的下一層節點
- **兄弟節點 sibling node**
  - 有著相同父節點的所有節點彼此稱為兄弟節點
  - 彼此皆互稱為兄弟節點，沒有先來後到順序，也沒有區分成兄節點或弟節點。

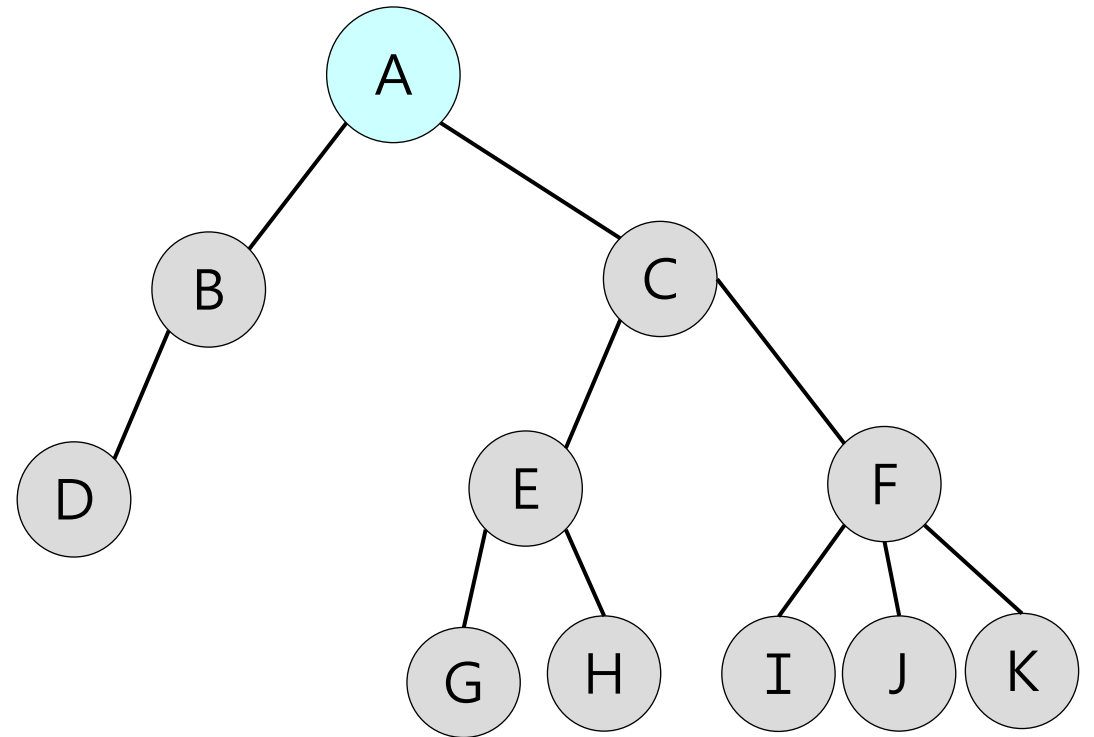


# 各種節點 (2)

- **祖先節點 ancestor node**
  - 從 root 到指定節點(node-K) 的通過路徑上的所有節點(包含 root)都是指定節點(node-K)的祖先節點
  - root 是所有節點的祖先結點
  - 父節點同時也會是子節點
- **子孫節點 descendant node**
  - 當 A 節點(node-A)可做為 B 節點(node-B)的祖先節點時，那 B 節點(node-B)也會被稱為是 A 節點(node-A)的子孫節點
  - 子節點同時也會是子孫節點
- **非終點節點 non-terminal node**
  - 至少有一個子節點的節點
  - 也稱為**內部節點 internal node**
- **終點節點 terminal node**
  - 沒有子節點的節點
  - 也稱為**葉子節點 leaf node**
  - 也稱為**外部節點 external node**

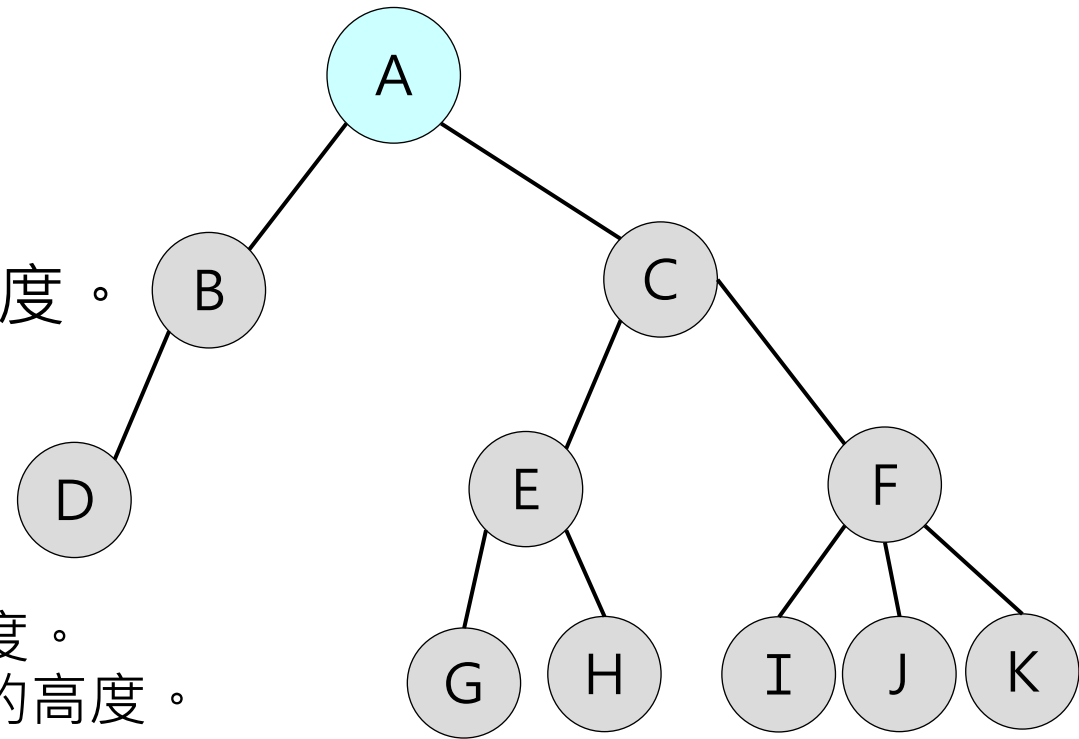
# 各種節點 – 範例

- **A 節點是 root**
- A 節點是 B 節點與 C 節點的父節點
- B 節點是 D 節點的父節點
- D 節點是 B 節點的子節點
- C 節點是 E 節點與 F 節點的父節點
- E 節點與 F 節點是 C 節點的子節點
- A 節點、C 節點、E 節點都是 G 節點與 H 節點的祖先節點
- G 節點與 H 節點是 A 節點、C 節點、E 節點的子孫節點
- G 節點、H 節點、I 節點、J 節點、K 節點彼此是兄弟節點
- **D 節點、G 節點、H 節點、I 節點、J 節點、K 節點都是葉子節點(外部節點)**
- A 節點、B 節點、C 節點、E 節點、F 節點都是內部節點



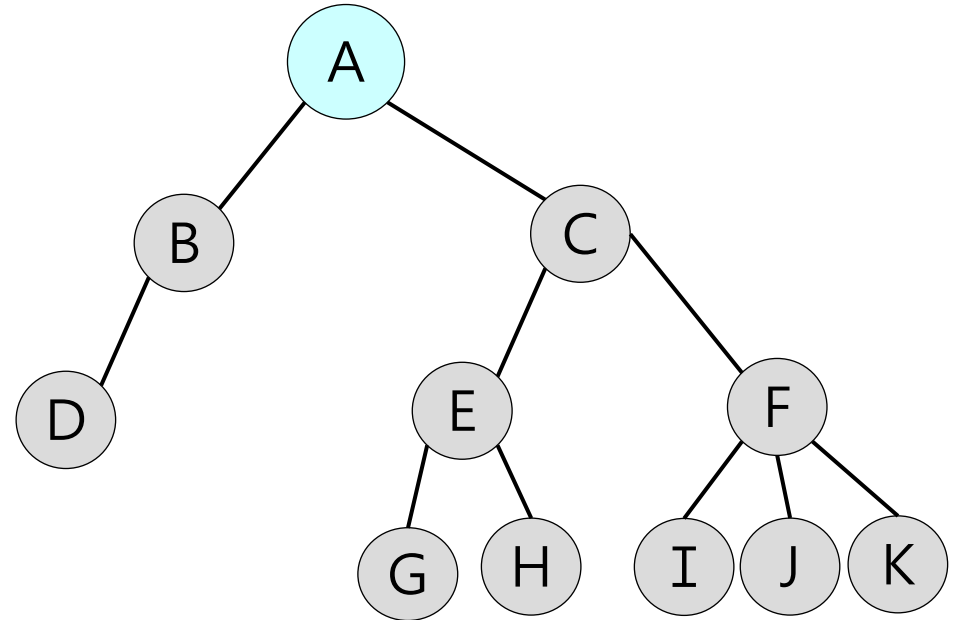
# 階層、深度、高度、分支度

- 階層 level：節點的階層位置。
  - root 是第 1 階，root 的子節點是在第 2 階。
  - 也可稱為「階度」。
- 深度 depth：從 root 到此節點的路徑長度。
  - root 的子節點深度為 1。
  - 某節點的深度等於該節點的階層+1。
- 高度 height：
  - 節點的高度：節點到葉子節點的最常路徑長度。
  - 樹的高度：樹中節點的最大階層就是這顆樹的高度。
- 分支度 branch：一個節點的子節點數目



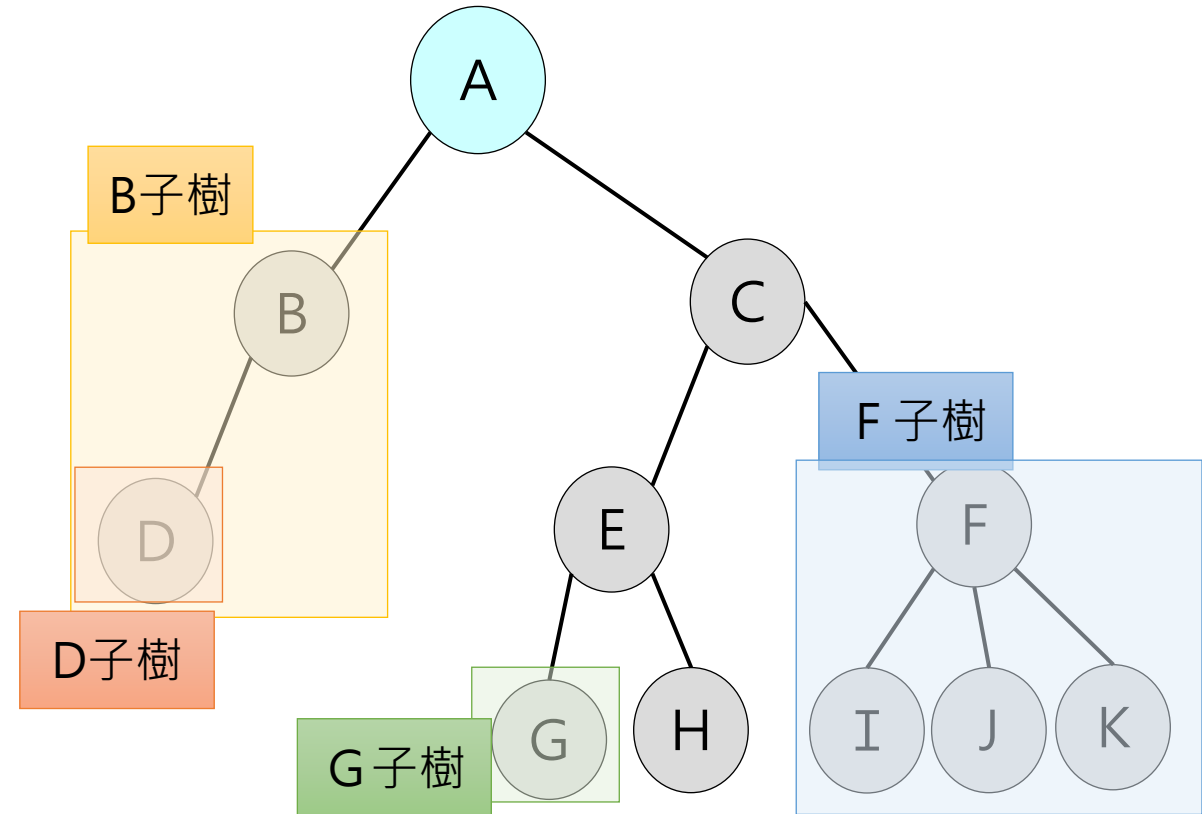
# 各種屬性 - 範例

|     | 階層 | 深度 | 高度 | 分支度 |
|-----|----|----|----|-----|
| A節點 | 1  | 0  | 3  | 2   |
| B節點 | 2  | 1  | 1  | 1   |
| C節點 | 2  | 1  | 2  | 2   |
| D節點 | 3  | 2  | 0  | 0   |
| E節點 | 3  | 2  | 1  | 2   |
| F節點 | 3  | 2  | 1  | 3   |
| G節點 | 4  | 3  | 0  | 0   |
| H節點 | 4  | 3  | 0  | 0   |
| I節點 | 4  | 3  | 0  | 0   |
| J節點 | 4  | 3  | 0  | 0   |
| K節點 | 4  | 3  | 0  | 0   |



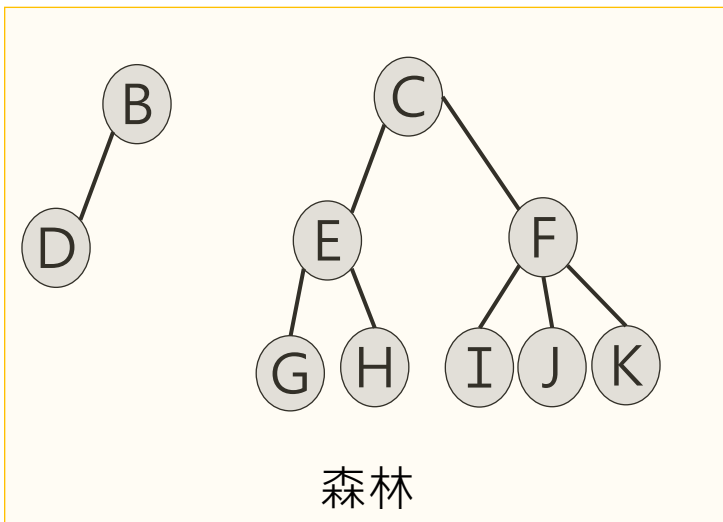
# 子樹 subtree

- 子樹 subtree：與子圖(subgraph)的概念相似，子圖就是在 root 下的相連部分。
  - 一個節點也可是一個子樹。
  - 子樹若包含不只一個節點，那這個子樹又可再拆成多個子樹。
  - 子樹的根節點編號就是這顆子樹的名稱。



# 森林 forest

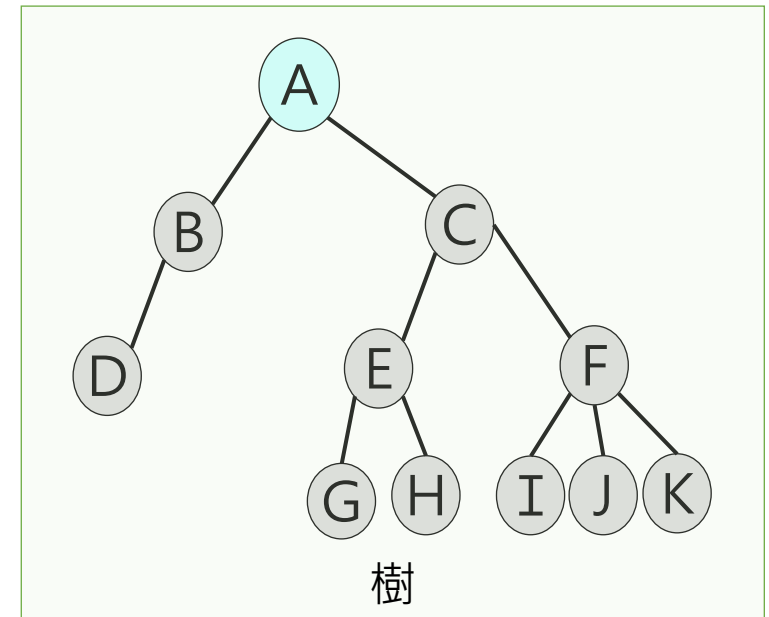
- 森林 forest：多顆彼此不相鄰且節點不重複的樹的集合。
  - 當一顆樹移除 root 後，那剩下的子樹們就會被稱為森林 forest。
  - 也可稱為森林。
  - 右圖是有兩個子樹的森林。



森林加上 root 節點就變成樹

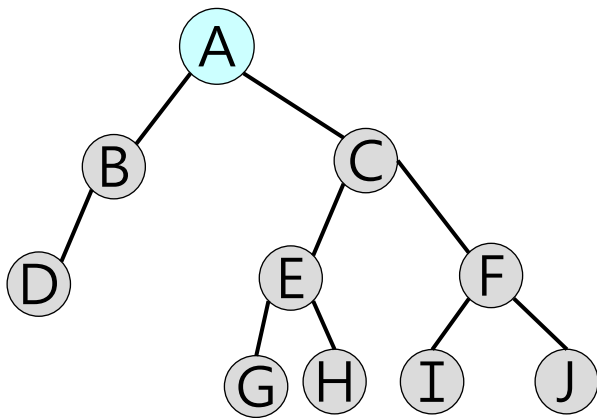


樹移除 root 就變成森林

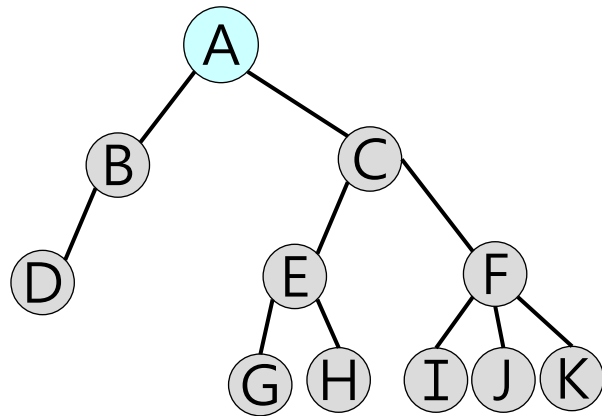


# n 元樹

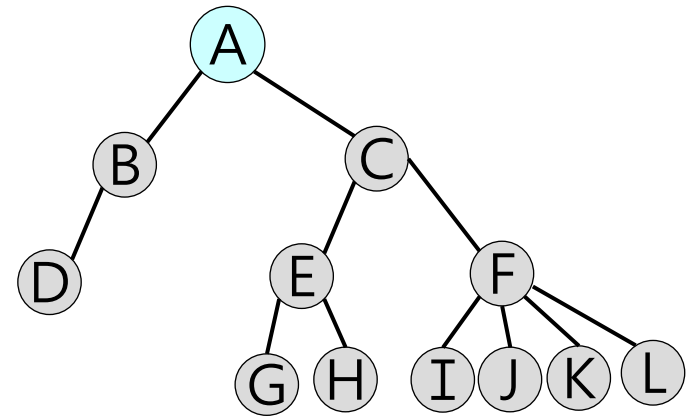
- n元樹：樹的任一個節點最多有  $n$  個子節點
  - $n$  為大於等於 2 的數
- 二元樹 **binary tree**：樹的任一個節點最多有 2 個子節點
- 三元樹：樹的任一個節點最多有 3 個子節點



二元樹



三元樹



四元樹

<樹 Tree>

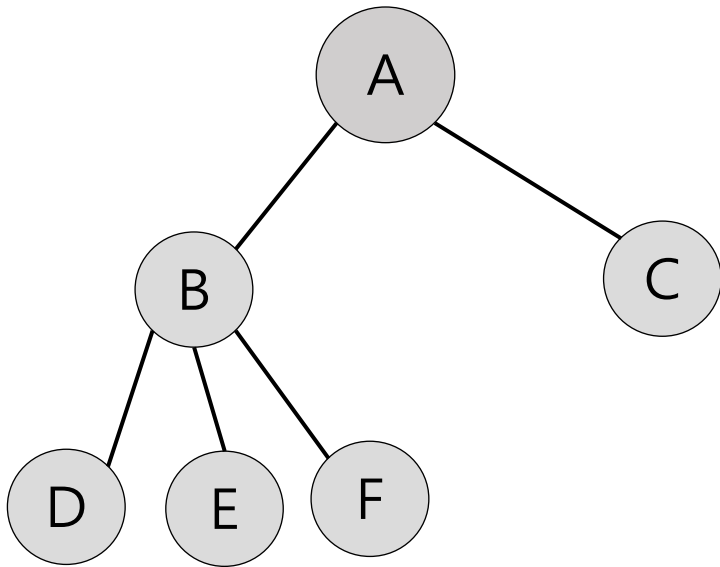
表示法





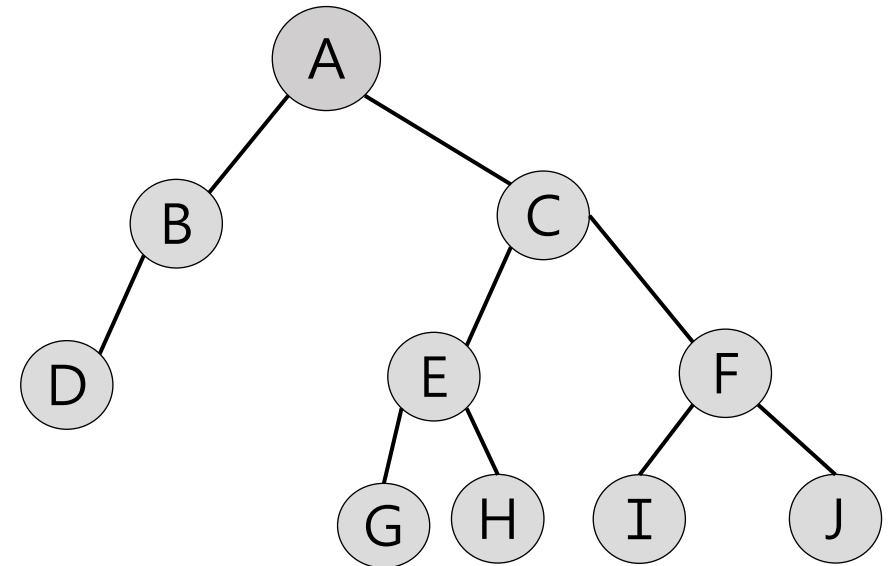
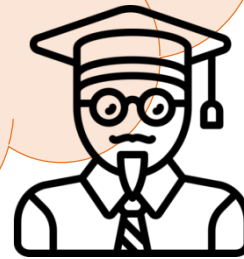
# 串列表示法

- 使用括號呈現，括號內先列出樹根節點，接著用括號包圍每一個子樹，每個子樹再用相同的方式表示。



(A (B (D, E, F), C))

串列表示法是用文字來描述一顆樹，並非程式內使用的資料結構。

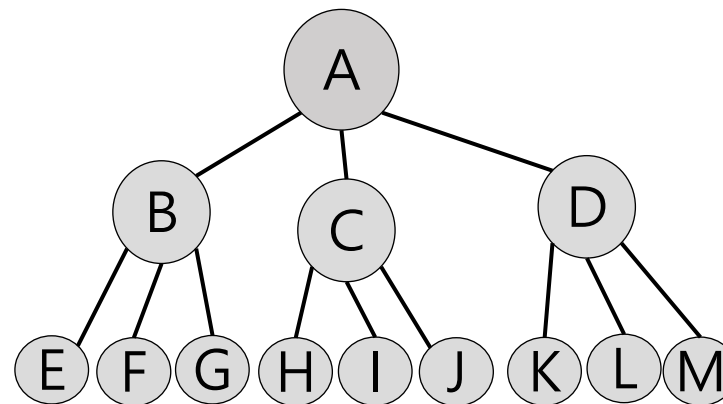
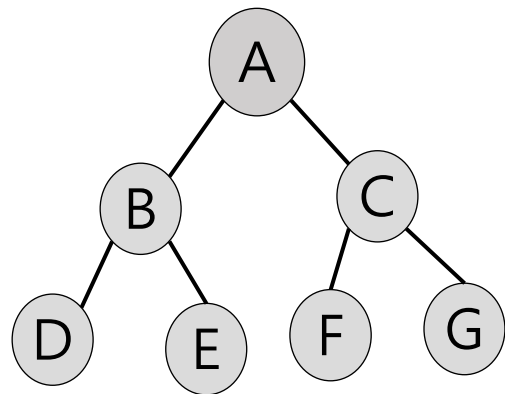


(A (B (D), C (E (G, H), F (I, J))))

# 陣列表示法

- 使用陣列表示法必須先知道樹內節點的**最大分支度(n)**以及**樹的高度**。
- 陣列表示法會根據每個階層最多的節點數，保留對應的陣列空間，總陣列大小就是每階層保留的空間加總。
  - 第 1 層: 保留 1 個空間
  - 第 2 層: 保留  $n^{2-1}$  個空間
  - 第 3 層: 保留  $n^{3-1}$  個空間
  - 第 h 層: 保留  $n^{h-1}$  個空間
- 若階層內的節點數沒被填滿時，保留的陣列空間就會被閒置

# 陣列表示法



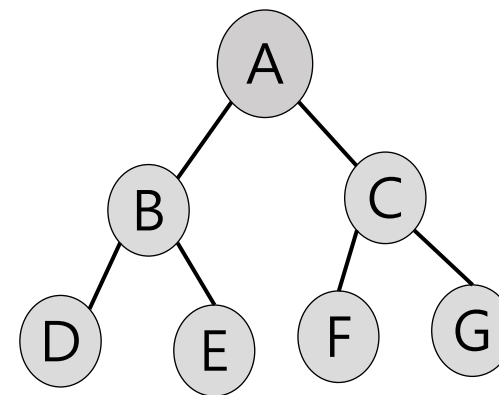
| 陣列大小 | 2元樹                     | 3元樹                      | 4元樹                       |
|------|-------------------------|--------------------------|---------------------------|
| 高度=1 | 1                       | 1                        | 1                         |
| 高度=2 | 1+2                     | 1+3                      | 1+4                       |
| 高度=3 | 1+2+4                   | 1+3+9                    | 1+4+16                    |
| 高度=4 | 1+2+4+8                 | 1+3+9+27                 | 1+4+16+64                 |
| 高度=n | $1+2+4+8+\dots+2^{n-1}$ | $1+3+9+27+\dots+3^{n-1}$ | $1+4+16+64+\dots+4^{n-1}$ |

# 陣列表示法

- 陣列內元素對應的索引值

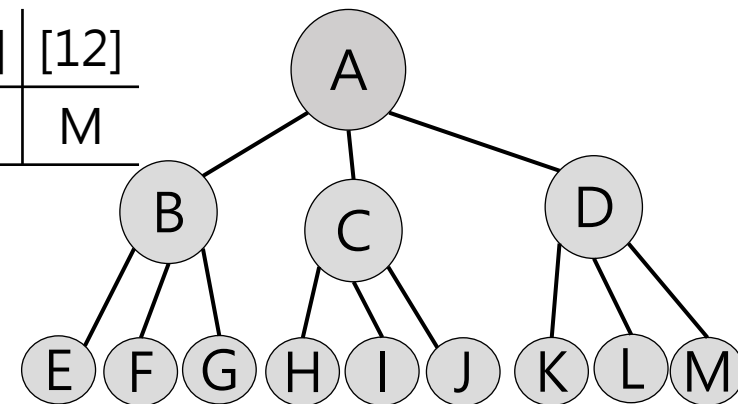
- 2元樹

|       |       |     |       |     |     |     |
|-------|-------|-----|-------|-----|-----|-----|
| [0]   | [1]   | [2] | [3]   | [4] | [5] | [6] |
| A     | B     | C   | D     | E   | F   | G   |
| 第 1 層 | 第 2 層 |     | 第 3 層 |     |     |     |



- 3元樹

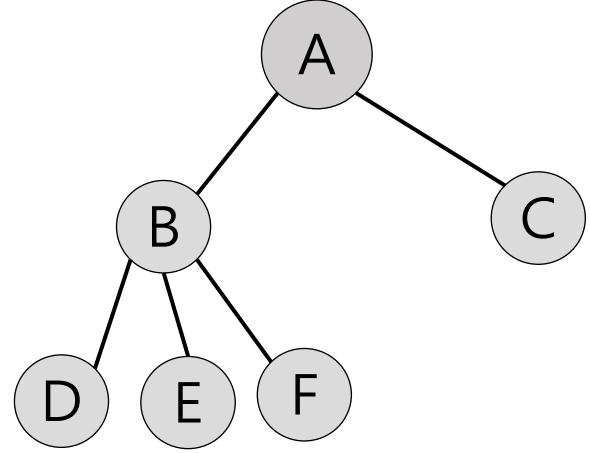
|       |       |     |     |       |     |     |     |     |     |      |      |      |
|-------|-------|-----|-----|-------|-----|-----|-----|-----|-----|------|------|------|
| [0]   | [1]   | [2] | [3] | [4]   | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
| A     | B     | C   | D   | E     | F   | G   | H   | I   | J   | K    | L    | M    |
| 第 1 層 | 第 2 層 |     |     | 第 3 層 |     |     |     |     |     |      |      |      |



# 陣列表示法

- 缺點：
  - 因為陣列表示法會保留每個階層最多的節點數，因此會有**空間浪費**的問題（除非樹的每個節點分支度都是一樣的）。
- 改善：
  - 為了節省空間的浪費，多數會將樹轉換成**二元樹**後再處理。

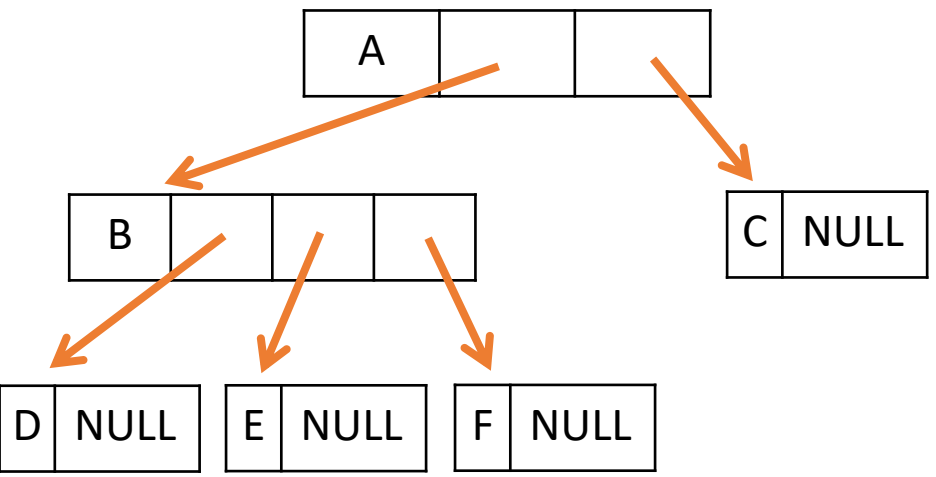
# 鏈結串列表示法



鏈節串列表示法可以分成兩種方式：

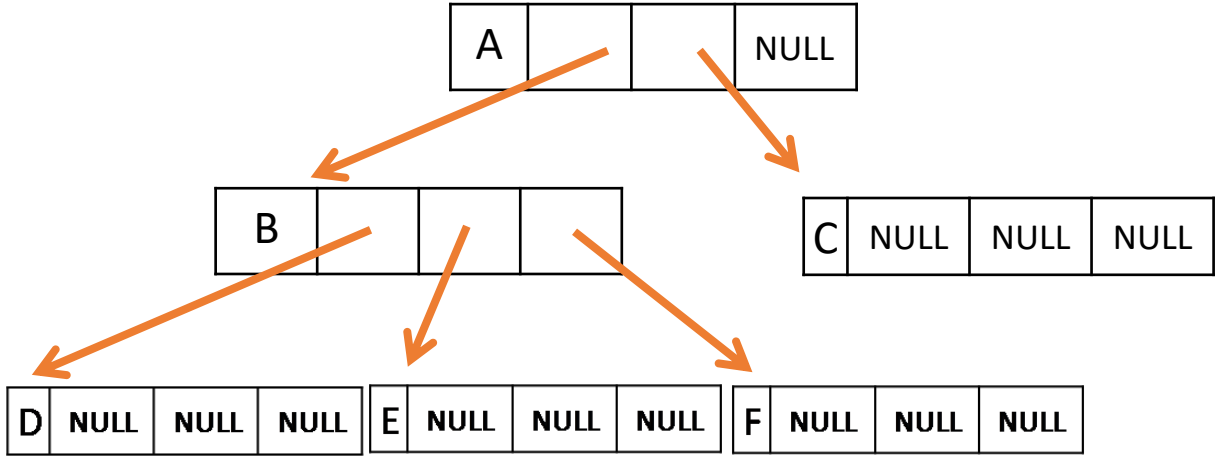
### 節點欄位數不固定

- 每個節點依照子節點數配置 link 數量
- 葉子節點會保留一個 NULL 的 ink



### 節點欄位數固定

- 每個節點會預留固定的 link
- n 元樹: 每個節點會預留 n 個的 link



# 鏈結串列表示法

鏈節串列表示法可以分成兩種方式：

## 節點欄位數不固定

- 缺點：
  - 每個節點的 link 空間都要動態配置，當子節點有增減時，都需要比較多的處理。

## 節點欄位數固定

- 缺點：
  - 與陣列表示相似，會多配置 link 空間，有空間浪費的問題。
- 改善：
  - 與陣列表示法一樣，為了節省空間的浪費，多數會將樹轉換成二元樹後再處理。



延伸的概念



# 概念1: 樹 -> 二元樹

- 樹型結構是一個重要的資料結構，但當分支度大時，圖型會有很大的變異空間，也容易造成資料結構的設計困難，因此在實際應用上，多數會將問題簡化成二元樹的應用，才能讓程式的實作上更為容易。