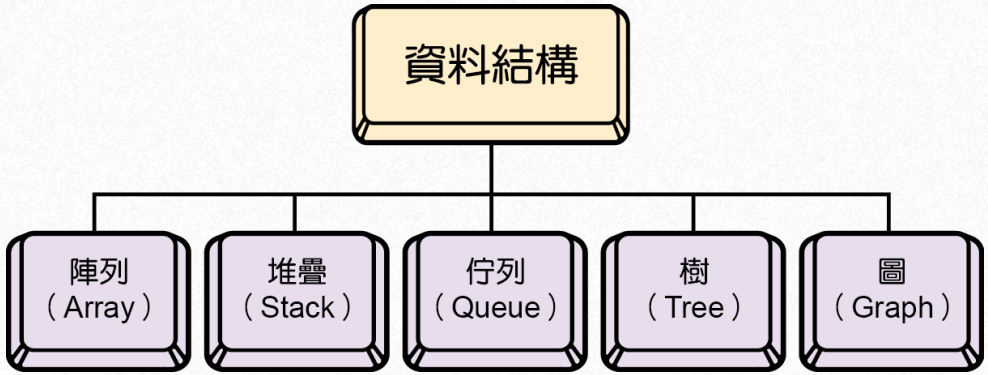
chapter1

重要資料結構的概念與應用

資料結構 (Data Structure)

主要用來規劃**如何將資料 (Data) 有組織地存放在記憶體中**，以提升其解決特定問題時的執行效能。



1-1 Array

陣列

運用**連續的表格**記錄**相同型態**的資料，使資料的記錄或查詢更便潔。

/ 例如

- 老師的點名單
- 班級成績單
- 學生身高體重紀錄表等

- 以點名單為例，其中座號資料欄位和姓名資料欄位，此即構成**一維陣列**。

1-1 Array

陣列

座號是每人專屬的序號，透過座號，就可以查詢到相對應的姓名資料，**具有一對一的專一性**。

1-1-1 / 數張點名卡整合為使用表格和座號排序的點名單較為簡潔，且可讀性高

點名單

座號	姓名
1	宋重機
2	姚承熙
3	宋會喬
4	李志恩
5	金澄五
6	蔡尚勻

1-1 Array

陣列

陣列可以透過**索引值**（序號，Index Value）存取其中的**元素**（資料，Element）。

1-1-2 / 點名單中的索引值和元素

這份點名單中

- 座號為**索引值**
- 索引值相對應資料為**元素**

點名單

座號	姓名
1	宋重機
2	姚承熙
3	宋會喬
4	李志恩
5	金澄五
6	蔡尚勻

「索引值4」，對應元素就是「李志恩」

索引值　元素

1-1 Array

陣列

若有許多**相同大小的一維陣列資料**，
可以將其**組合成二維陣列型態**。

1-1-3 / 班級週課表

二年 17 班　課表

節次 \ 星期	星期一	星期二	星期三	星期四	星期五
1	國文	英文	英文	歷史	地理
2	國文	資訊	英文	歷史	地理
3	自主學習	資訊	體育	公民	物理
4	自主學習	地科	體育	公民	物理
5	數學	生物	家政	數學	化學
6	化學	生物	家政	數學	地科

將五個一維陣列，組合成一個二維陣列

- **每天 6 節課程，視為一維陣列**
- **將 5 天組合成二維陣列**
  記錄 6 × 5 = 30 節課程資料。

1-1 Array

陣列

若有許多**相同大小的一維陣列資料**，
可以將其**組合成二維陣列型態**。

1-1-3 / 班級週課表

二年 17 班　課表

節次 \ 星期	星期一	星期二	星期三	星期四	星期五
1	國文	英文	英文	歷史	
2	國文	資訊	英文	歷史	地理
3	自主學習	資訊	體育	公民	物理
4	自主學習	地科	體育	公民	物理
5	數學	生物	家政	數學	化學
6	化學	生物	家政	數學	地科

索引值

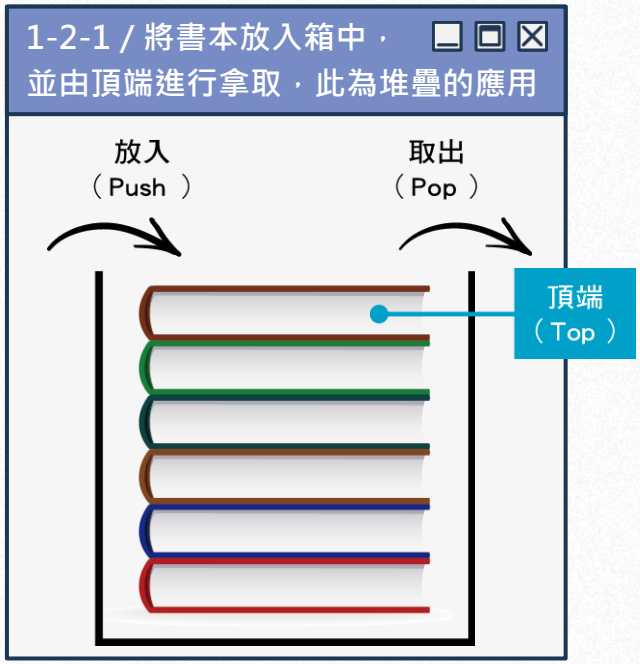
索引值

查詢「**星期二的第三節**」為「資訊課」。

得知二維陣列須以**兩個索引值查詢相對應元素值**。

1-2 Stack&Queue

堆疊與佇列 ▸ 1-2-1 堆疊



將書本由下而上**放入**（Push）箱中疊起，或從中**取出**（Pop），皆在箱中的最頂端進行書本的增減。

▶ 此過程，**皆在最上層**（頂端）**發生**，並遵守**後進先出**（LIFO）的概念，即為**堆疊**。

1-2 Stack&Queue

堆疊與佇列 ▸ 1-2-1 堆疊

實務上的應用

/ 例如

瀏覽器中「回到上一頁」按鈕

可將所經過的**頁面網址**（Push），以**堆疊的資料結構儲存於記憶體中**。

當按下**「上一頁」**，則將堆疊最上層的網頁（Pop 出前一個瀏覽的網址），重新載回瀏覽器的頁面。

實務上的應用

/ 例如

「還原」

編輯文件或圖片時，常運用的**「還原」功能**，**讓文件回到上一個步驟的編輯狀態**，藉此還原先前的資料或圖片。

Question

請問經過以下指令：

push A ▶ push B ▶ push C ▶ pop ▶ push D ▶ pop ▶ pop

最後堆疊內部結果為何？

Answer

1-2 Stack&Queue

堆疊與佇列 ▸ 1-2-2 佇列

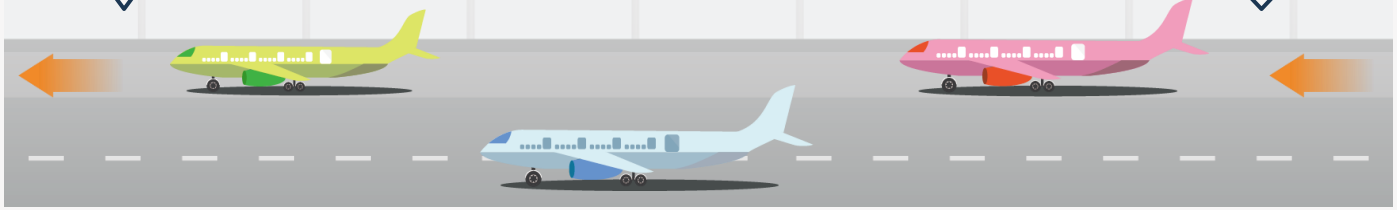
飛機在跑道等候起飛時，常以

「進入跑道排隊的先後順序」作為「起飛的先後次序」

1-2-2 / 機場塔臺指揮跑道上的飛機起飛是「先進先出」

首架進入跑道的飛機
＝優先起飛的架次

最後進入隊伍者，則需等待
前面的架次起飛後才能起飛



1-2 Stack&Queue

堆疊與佇列 ▸ 1-2-2 佇列

此種**先進先出**（FIFO）的方式，即為**佇列**結構。

1-2-3 / 排隊隊伍遵守「先進先出」的規則

- 堆疊： 加入與刪除資料都在**同一端點進行**
- 佇列： 加入資料於**尾端**（Rear）進行
  刪除資料則於**前端**（Front）進行。

佇列範例

實務上的應用

/ 例如

作業系統的工作排程

可以將先載入記憶體等待 CPU 執行的工作，優先放入 CPU 運算 (Computation) 並執行。

/ 例如 / 印表機的任務排程也是遵守佇列的規則，以送交時間的先後次序，依序執行列印

實務上的應用

印表機的任務排程

先送入印表機緩衝記憶體的工作，
會優先被列印出來。

1-2 Stack&Queue

堆疊與佇列 ▸ 1-2-2 佇列

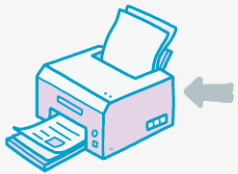
1-2-4 / 例如 / 印表機任務排程遵守佇列規則，以送交時間的先後次序，依序執行列印

實務上的應用

印表機的任務排程

列印工作3：第三章. docx

列印工作2：第二章. docx



列印工作1：第一章. docx

Question

有一佇列經過以下指令：

加入 A ▶ 加入 B ▶ 加入 C ▶ 刪除 ▶ 加入D ▶ 刪除 ▶ 刪除

請問最後該佇列的內部結果為何？

Answer

1-3 Tree

樹

樹狀結構仿效樹木生長模式，

由最高層級開始逐層分支、擴散的階層架構。

1-3-1 / 例如 / 生物分類法

生物學中的「**生物分類法**」就是樹狀結構的應用。

以**界**、**門**、**綱**、**目**、**科**、**屬**、**種**的分支作為樹狀延伸的依據。

1-3 Tree

樹

由此可知，樹狀結構是將一群**目標資料**
透過特定規則進行分類，並儲存於樹狀結構中。

樹狀結構和自然界的樹木生長方向相反：

- 為「**樹根在最上層，樹葉在最底層**」的倒立樹狀結構。

1-3-2 / 倒立的樹狀結構

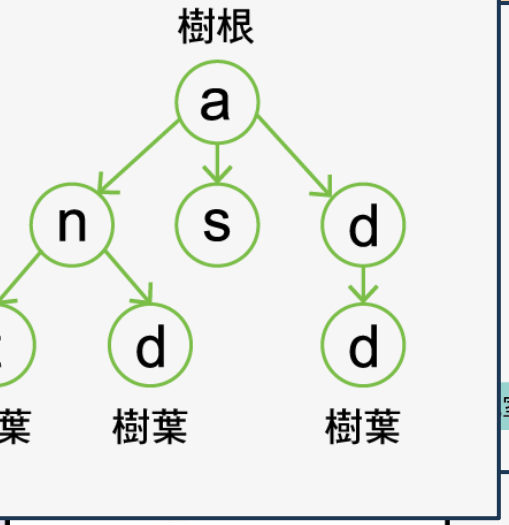
1-3 Tree

樹

1-3-3 / 例如 / 微軟 Windows 作業系統的檔案儲存結構

1-3-5 / 例如 / 字典樹

1-3-4



Browser

樹狀結構應用：

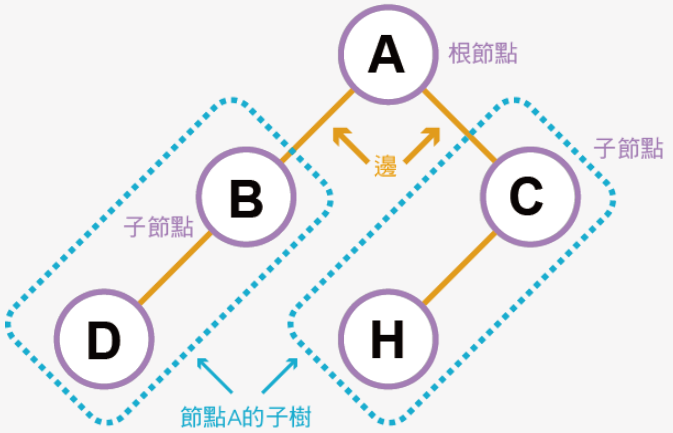
- 微軟 Windows 作業系統的檔案儲存結構
- 組織架構圖
- 單淘汰制賽程表
- 字典樹　等

1-3 Tree

樹

- 樹由**節點**（Nodes）和**邊**（Edges）相連形成
- 且至少由一個**根節點**（Root）所形成的集合

1-3-6 / 例如 / 節點與邊的關係



例如

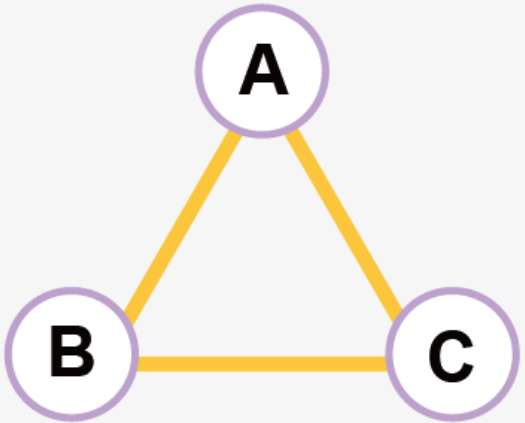
節點 A 有 2 個子節點（B、C）與其相連

1-3 Tree

樹

樹的任兩節點相通，且**只有一條路徑**，
亦**不會產生循環路徑**（Cycle）。

1-3-7 / 例如 / 樹不會產生循環，亦即不會產生起點與終點相同的迴路。



例如

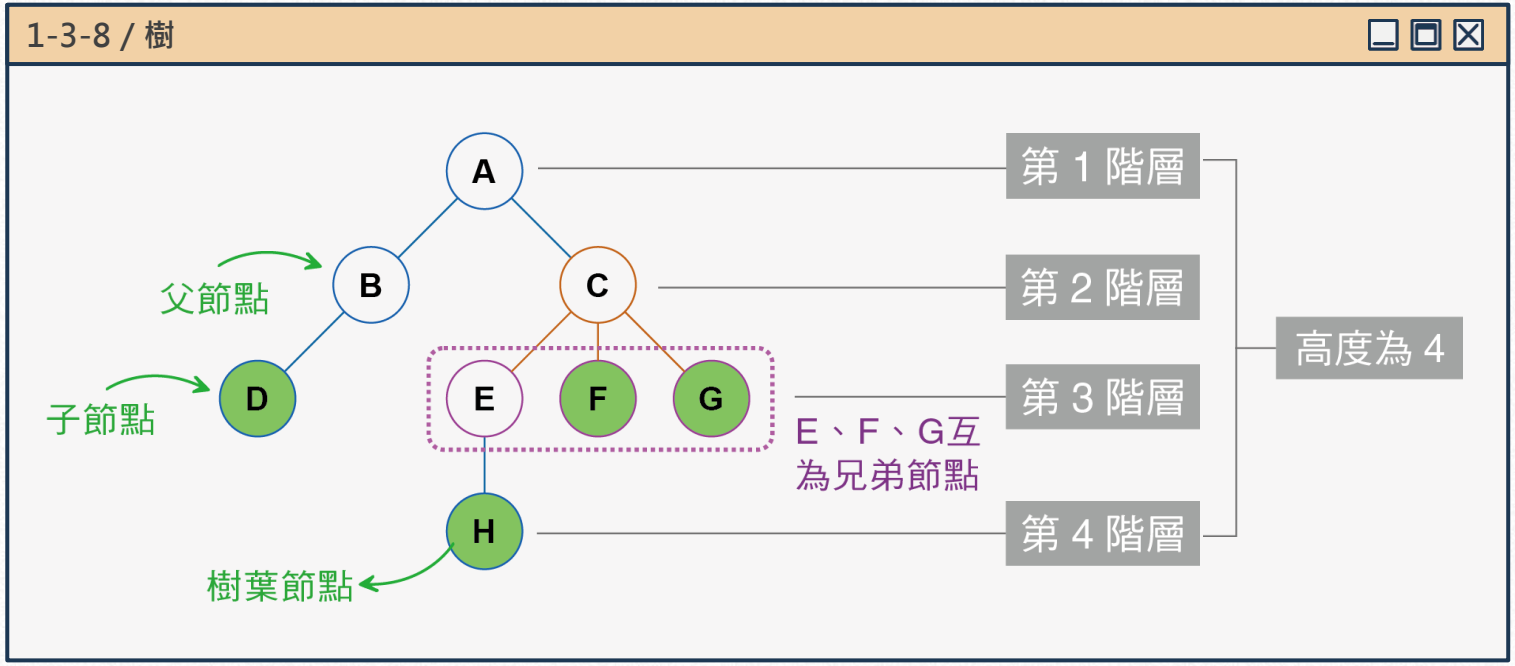
節點 B、C 若相連，
則節點 A、B、C
會產生**封閉循環**。

1-3 Tree

樹

1-3-1 樹的專有名詞

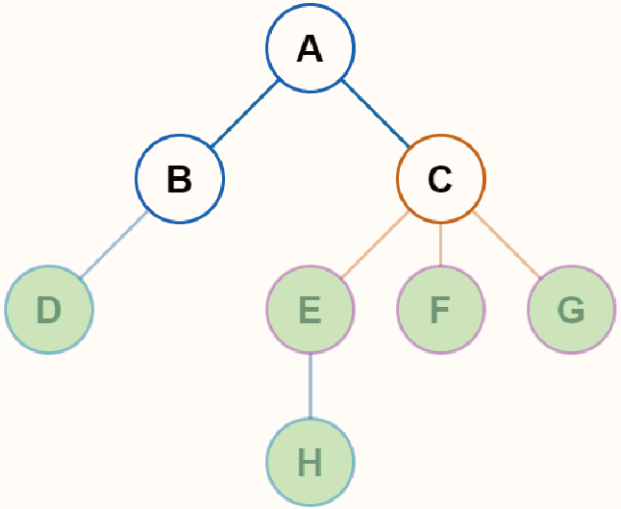
1-3-8 / 樹



1-3 Tree

樹 ▸ 1-3-1 樹的專有名詞

父 (Parent) 節點



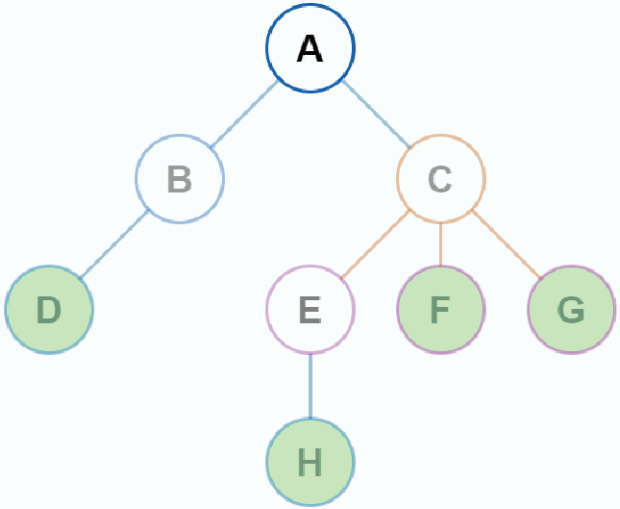
由該節點向上回溯的上一層節點。

例如 節點 B、C 的父節點皆為 A。

1-3 Tree

樹 ▸ 1-3-1 樹的專有名詞

樹根節點（Root）



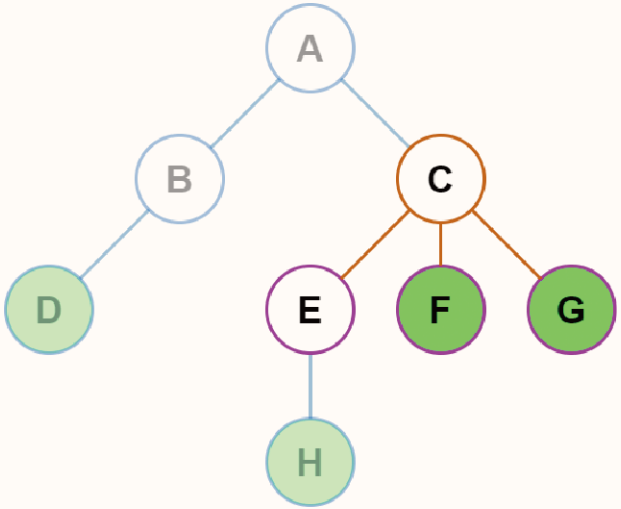
在樹的最頂層且沒有父節點的節點。

例如 節點 A。

1-3 Tree

樹 ▸ 1-3-1 樹的專有名詞

子節點 (Child)



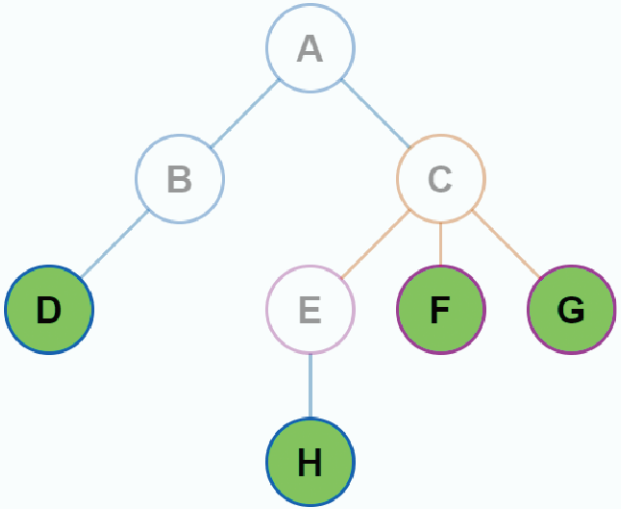
由該節點所衍生出的下一層節點。

例如 節點 C 的子節點為節點 E、F、G。

1-3 Tree

樹 ▸ 1-3-1 樹的專有名詞

樹葉 (Leaf) 節點



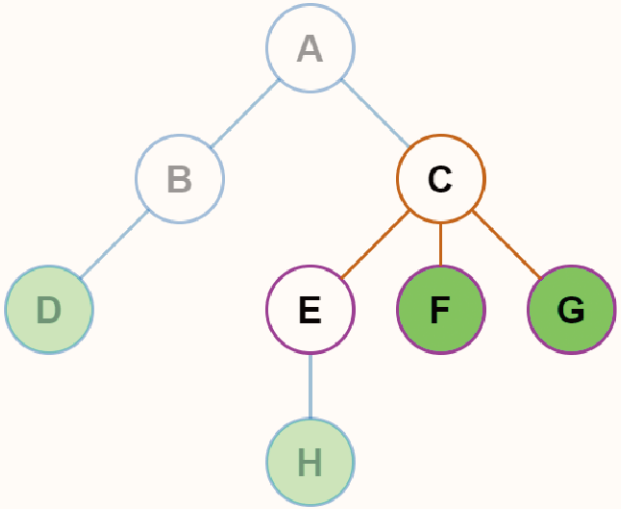
在樹的最底層，且沒有子節點的節點。

例如 節點 D、H、F、G。

1-3 Tree

樹 ▸ 1-3-1 樹的專有名詞

兄弟 (Sibling) 節點



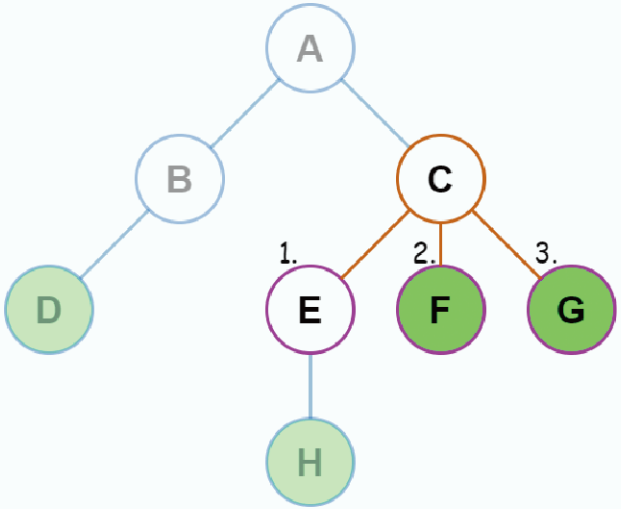
具有相同父節點的所有節點。

例如 節點 E、F、G 的父節點皆為C，故 E、F、G 互為兄弟節點。

1-3 Tree

樹 ▸ 1-3-1 樹的專有名詞

節點分支度 (Degree)



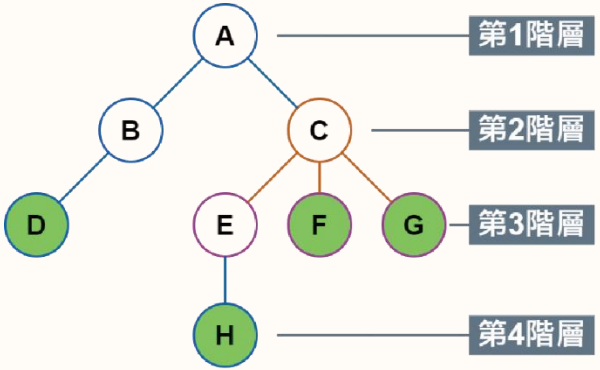
該節點所擁有的子節點數。

例如 如節點 C 的分支度為 3 。

1-3 Tree

樹 ▸ 1-3-1 樹的專有名詞

階層（Level）



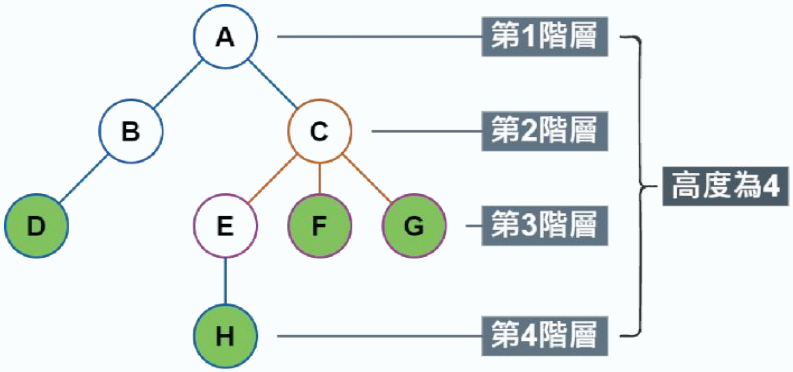
根節點為第 1 階層，而其所衍生的子節點為第 2 階層，依此類推。

例如 如H的所在位置為第 4 階層。

1-3 Tree

樹 ▸ 1-3-1 樹的專有名詞

高度（Height）

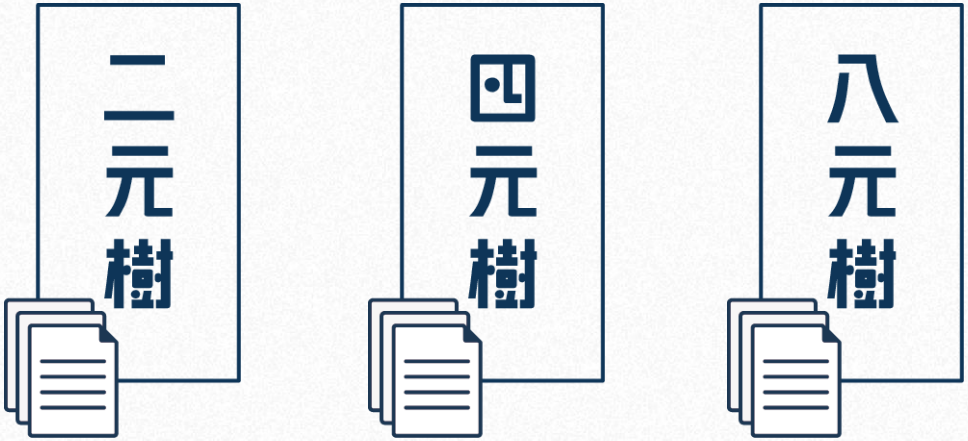


此樹的最大階層數，又稱深度（Depth）。

例如 圖的高度為 4 。

1-3-2 二元樹 (Binary Tree)

樹依據**不同的分支度**，可分為許多類型：



1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

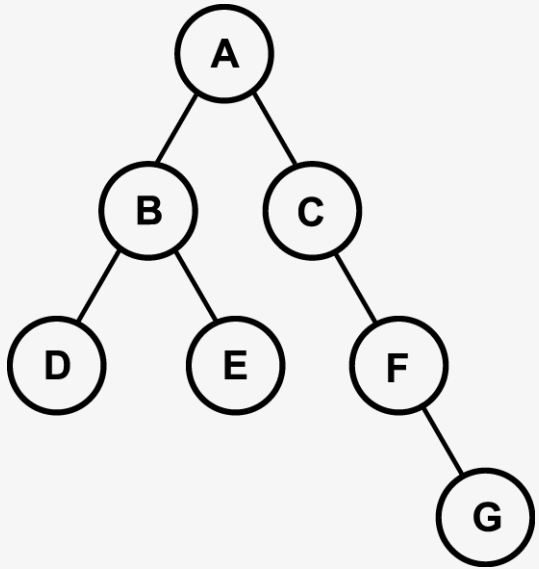
二元樹

應用廣泛，由於「最大分支度為 2 」的特性。

可應用在：

- 排序 (Sort)
- 搜尋 (Search)
- 編碼　等

1-3-9 / 二元樹示意圖



1-3 Tree

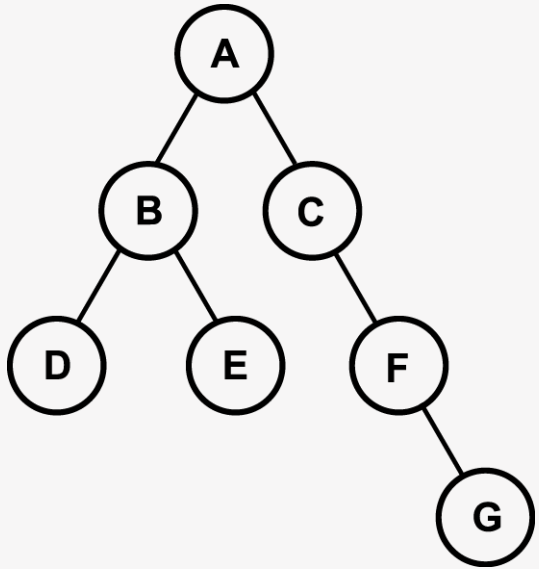
樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹

例如：

- 搜尋演算法之一的**二元搜尋樹**（Binary Search Tree）
- 資料壓縮所使用到的**霍夫曼樹**（Huffman Tree）

1-3-9 / 二元樹示意圖



1-3 Tree

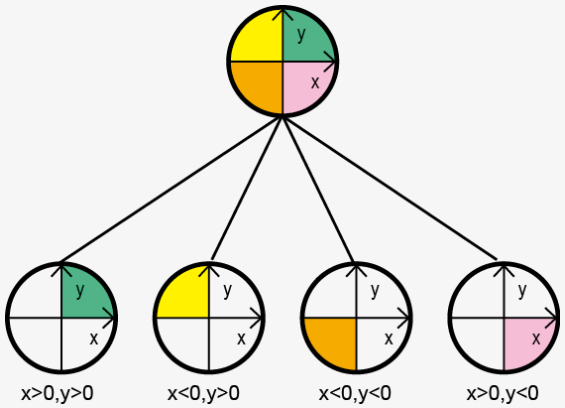
樹 ▸ 1-3-2 二元樹 (Binary Tree)

四元樹

八元樹

本質上仍可以運用**二元樹進行實作**。

1-3-10 / 四元樹範例



可應用在：

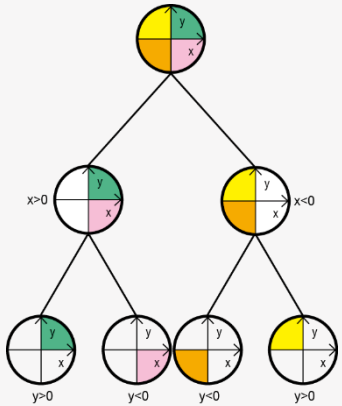
- 平面座標的象限分類，恰好可以區分四個不同象限。

四元樹

八元樹

二元樹實作四元樹範例：

1-3-11 / 例如



運用**二元樹實作四元樹**，可以先以 x > 0 或 x < 0 劃分為兩部分。

由這兩部分分別向下，再以 y > 0 或 y < 0 劃分，**最後同樣可形成四個象限的分配**。

1-3 Tree

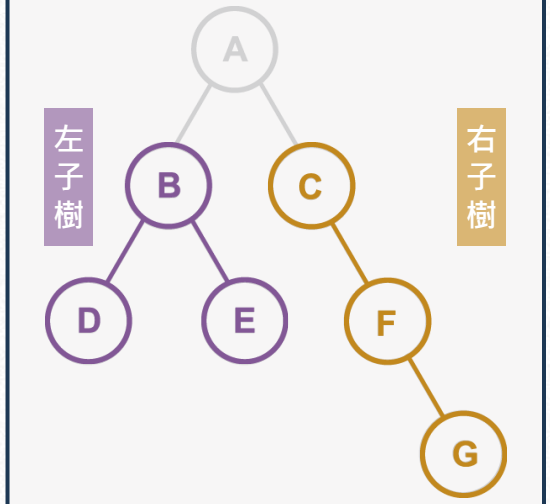
樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹

① 二元樹即為樹中**每一個「節點」的子節點**，數量為**0～2個**。

② 根節點外，可分成兩個獨立子樹：**「左子樹」**和**「右子樹」**。

1-3-12 / 1-3-13 / 左子樹與右子樹

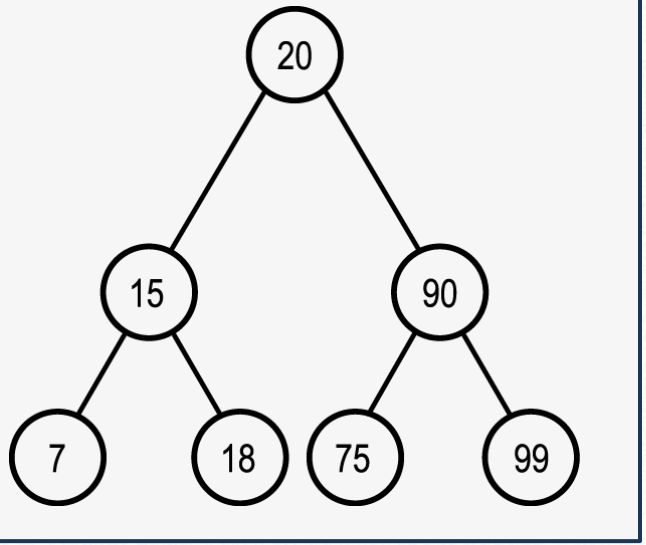


二元搜尋樹

「二元搜尋樹」是二元樹的一種延伸應用，目的在於提供資料：

- 新增
- 刪除
- 搜尋　等操作

1-3-14 / 二元搜尋樹示意圖



二元搜尋樹

依下列規則，將數字依序排列，形成一二元搜尋樹

1. 二元搜尋樹**所有節點皆儲存不同數值**
2. 二元搜尋樹中，**左子節點**的數值必**小於**該所屬**父節點**的數值
3. 二元搜尋樹中，**右子節點**的數值必**大於**該所屬**父節點**的數值
4. 每一父節點所衍生的**左、右子樹，皆符合二元搜尋樹的規則**

範例 1-3-1 | 二元搜尋樹的建立

將下列數值資料建立二元搜尋樹：15、21、16、7、3、14、25、12

1

首先，放入**資料 15**
作為樹根節點

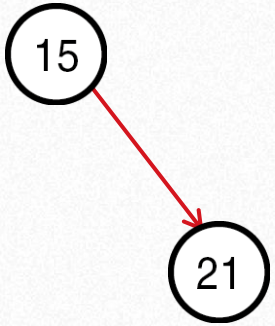
範例 1-3-1｜二元搜尋樹的建立

將下列數值資料建立二元搜尋樹：15、21、16、7、3、14、25、12

2

放入**資料 21**

（21 > 15，往右子樹放）



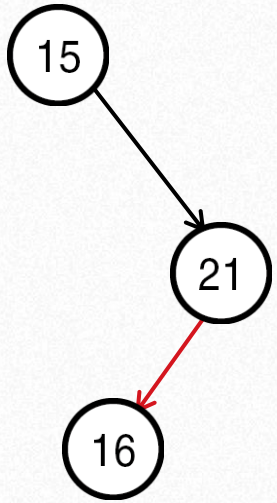
範例 1-3-1 | 二元搜尋樹的建立

將下列數值資料建立二元搜尋樹： 15、21、16、7、3、14、25、12

3

放入**資料 16**

（16 > 15，往右子樹放；
16 < 21，往左子樹放）



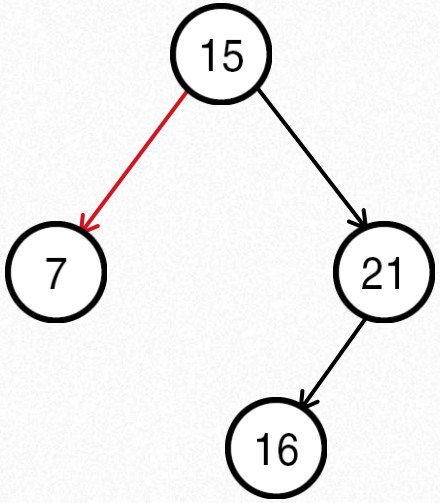
範例 1-3-1 | 二元搜尋樹的建立

將下列數值資料建立二元搜尋樹： 15、21、16、7、3、14、25、12

4

放入**資料 7**

（ 7 < 15，往左子樹放 ）



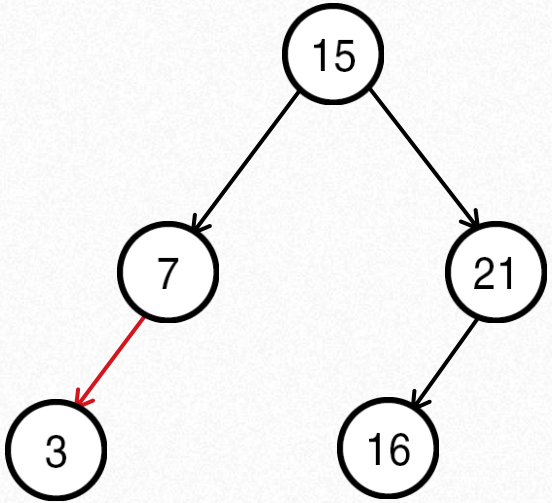
範例 1-3-1 | 二元搜尋樹的建立

將下列數值資料建立二元搜尋樹： 15、21、16、7、3、14、25、12

5

放入**資料 3**

（ 3 < 15，往左子樹放；
3 < 7 ，往左子樹放）



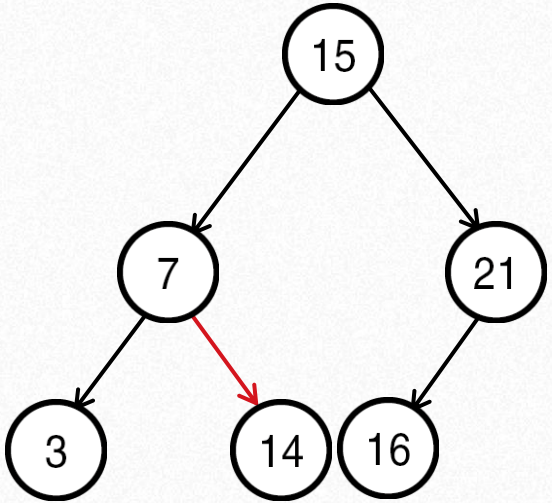
範例 1-3-1｜二元搜尋樹的建立

將下列數值資料建立二元搜尋樹：15、21、16、7、3、14、25、12

6

放入**資料 14**

（14 < 15，往左子樹放；
14 > 7，往右子樹放）



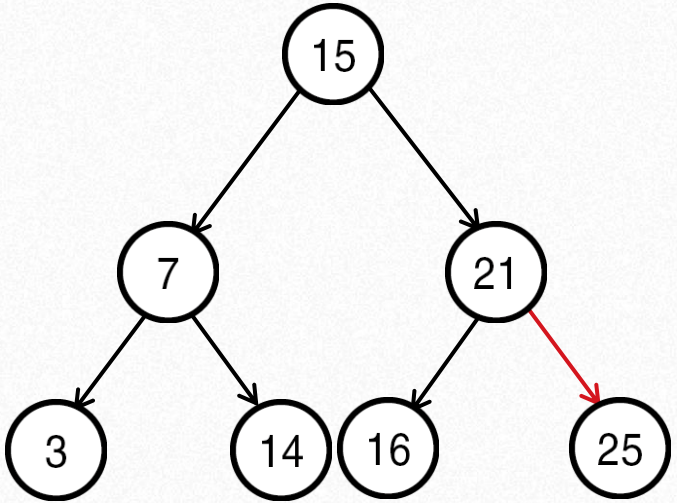
範例 1-3-1 | 二元搜尋樹的建立

將下列數值資料建立二元搜尋樹： 15、21、16、7、3、14、25、12

7

放入**資料 25**

（25 > 15，往右子樹放；
25 > 21，往右子樹放）



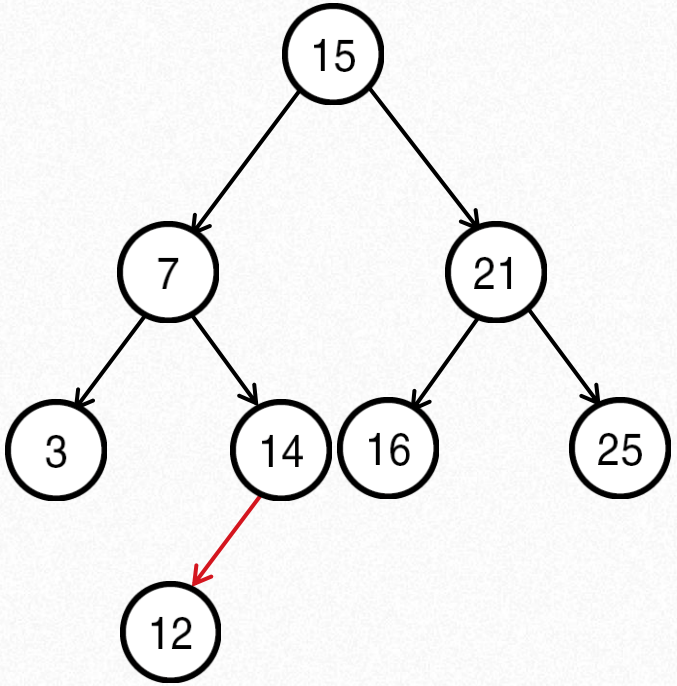
範例 1-3-1 | 二元搜尋樹的建立

將下列數值資料建立二元搜尋樹： 15、21、16、7、3、14、25、12

8

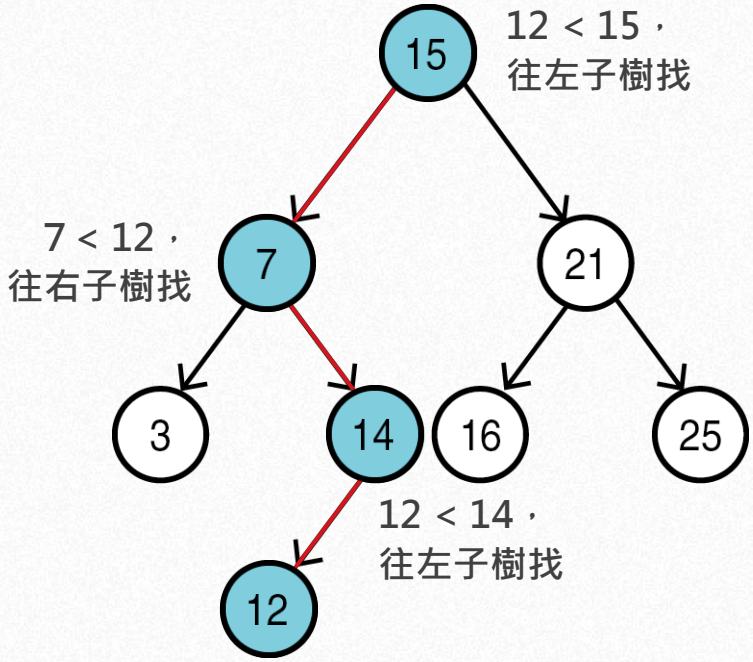
放入**資料 12**

（12 < 15，往左子樹放；
12 > 7 ，往右子樹放；
12 < 14，往左子樹放）



範例 1-3-2｜二元搜尋樹的搜尋

運用**範例 1-3-1** 建立之二元搜尋樹搜尋下列數值：

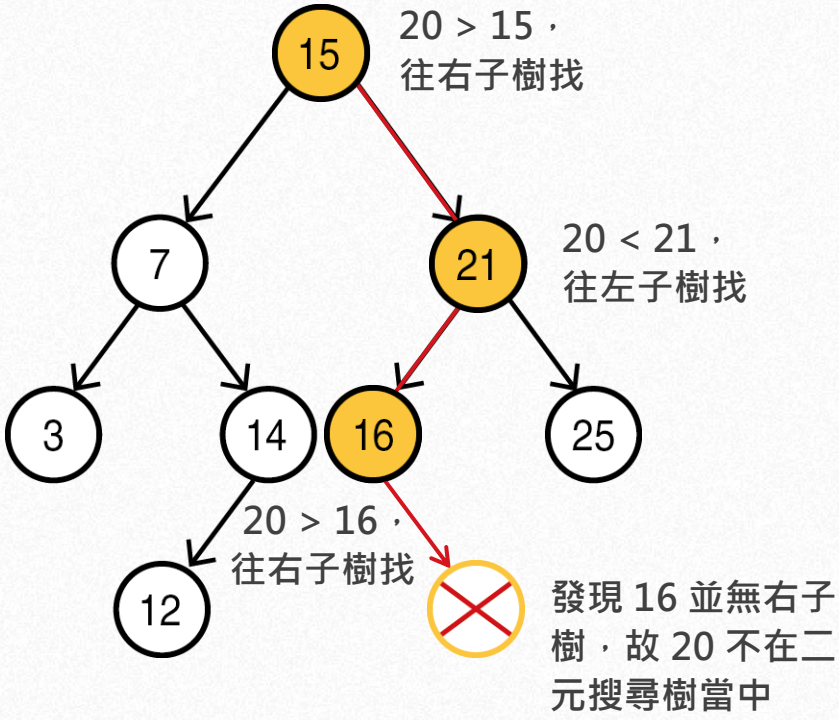


1

搜尋數值 12

範例 1-3-2 | 二元搜尋樹的搜尋

運用**範例 1-3-1** 建立之二元搜尋樹搜尋下列數值：



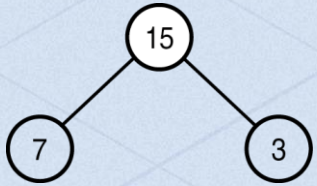
2

搜尋數值 20

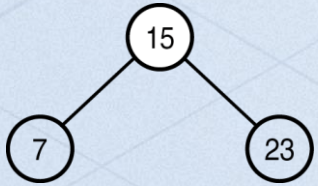
科技即戰力

Question 請問下列圖形是否為二元搜尋樹？

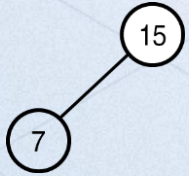
（1）



（2）



（3）



Answer

霍夫曼編碼 (Huffman Coding)

ASCII 編碼，屬於固定長度編碼方式；
霍夫曼編碼屬於**非固定長度編碼**方式的一種。

/ 常用於

- **字元 (char) 的編碼**
- **資料的壓縮 等**

其運用資料出現頻率的差異性，進而降低資料儲存的空間。

霍夫曼編碼

其三步驟流程如下：

1. 先將目標資料中，相異字元出現的頻率由大到小進行排列
2. 建立霍夫曼樹
3. 再以建立的霍夫曼樹制定新的編碼規則

霍夫曼編碼

我們先將下列字串作為目標資料，並且以 **ASCII 碼**及**霍夫曼編碼**比較兩者的差異：

/ 目標資料

a a a a a a b b b b c c c c c d d e

霍夫曼編碼

ASCII 碼

霍夫曼編碼

字元	a	b	c	d	e
二進位表示	1100001	1100010	1100011	1100100	1100101

ASCII 碼每個字元皆以 **7 個位元** (bits) **表示**。

目標資料

- 6 個 a
- 4 個 b
- 5 個 c
- 2 個 d
- 1 個 e

共 (6 + 4 + 5 + 2 + 1) = 18 個字元，

故 7 (bits) × 18 (字元) = 126 位元

1. 先將目標資料的相異字元出現的頻率，由大到小表列。

字元	a	c	b	d	e
出現頻率	6	5	4	2	1

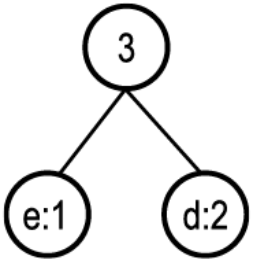
2. 開始根據出現頻率列表，逐步建立霍夫曼樹。

STEP 1

① 挑選頻率最小的 e、d 作為左右節點

② 並加總頻率作為兩者樹根

③ 將此樹根列入頻率表排序



字元	a	c	b	
出現頻率	6	5	4	3

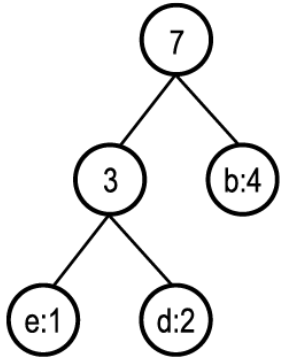
2. 開始根據出現頻率列表，逐步建立霍夫曼樹。

STEP 2

① 挑選頻率最小的3、b
作為左右節點

② 並加總頻率作為兩者樹根

③ 將此樹根列入頻率表排序



字元		a	c
出現頻率	7	6	5

2. 開始根據出現頻率列表，逐步建立霍夫曼樹。

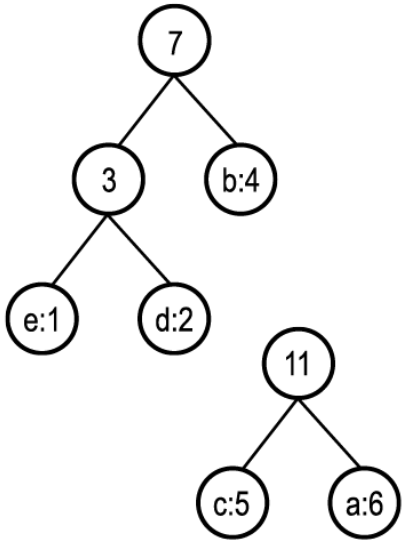
STEP 3

① 挑選頻率最小的c、a
作為左右節點

② 並加總頻率作為兩者樹根

③ 將此樹根列入頻率表排序

字元		
出現頻率	11	7

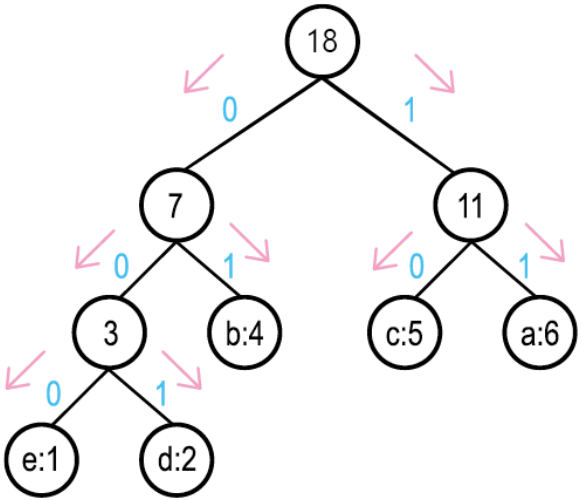


2. 開始根據出現頻率列表，逐步建立霍夫曼樹。

STEP 4

① 將最後的 7 、11 作為左右節點

② 加總頻率作為兩者樹根



3. 再以建立的霍夫曼二元搜尋樹，制定新的編碼規則。

字元	a	b	c	d	e
編碼	11	10	01	001	000
出現頻率	6	5	4	2	1

共需 2×6 + 2×5 + 2×4 + 3×2 + 3×1

= 39 位元

Question

（1） 比較上述例子，**ASCII 編碼**與**霍夫曼編碼**兩種方式編碼後，**何者較能節省儲存空間**？

Question

（ 2 ） 為何此種編碼方式較能節省儲存空間呢？

Question

（ 3 ） 除了如 JPEG 圖片檔案格式是使用霍夫曼編碼外，**還有哪些檔案格式也是運用這樣的技術呢**？

二元樹的走訪

二元樹的走訪（或稱追蹤），目的是為了造訪其每個節點，且依據造訪順序，可分成**三種走訪方式**：

01 前序走訪方式 Preorder Traversal

先拜訪 **樹根節點** → 而後拜訪 **左子樹** → 最後拜訪 **右子樹**

而每次拜訪的子樹又是依據相同追蹤方式，拜訪每一個節點。

二元樹的走訪

二元樹的走訪（或稱追蹤），目的是為了造訪其每個節點，且依據造訪順序，可分成**三種走訪方式**：

02 中序走訪方式 Inorder Traversal

先拜訪	而後拜訪	最後拜訪
左子樹	樹根節點	右子樹

而每次拜訪的子樹又是依據相同追蹤方式，拜訪每一個節點。

二元樹的走訪

二元樹的走訪（或稱追蹤），目的是為了造訪其每個節點，且依據造訪順序，可分成**三種走訪方式**：

03 後序走訪方式 Postorder Traversal

先拜訪 **左子樹** → 而後拜訪 **右子樹** → 最後拜訪 **樹根節點**

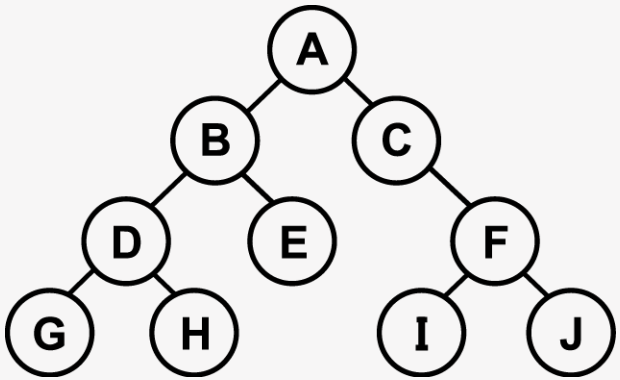
而每次拜訪的子樹又是依據相同追蹤方式，拜訪每一個節點。

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

1-3-15 / 二元樹範例



以此二元樹為例，
進行下列二元樹的走訪：

01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

二元樹的走訪

01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

① 由**樹根節點**優先輸出

② 而後輸出**左子節點**

③ 再輸出**右子節點**

樹根輸出順序在最前面，
故稱為前序追蹤。

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

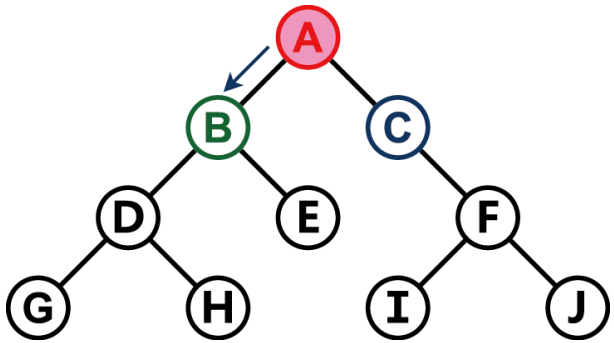
01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

STEP 1

先輸出樹根 A，而後拜訪左子節點，最後再拜訪右子節點。



目前輸出 A

：輸出值
：樹根節點
：左子節點
：右子節點
：已輸出節點

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

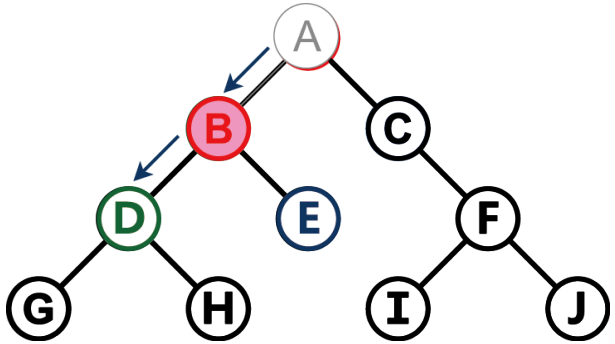
01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

STEP 2

先輸出樹根 B，而後拜訪左子節點，最後再拜訪右子節點。



目前輸出 A → B

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

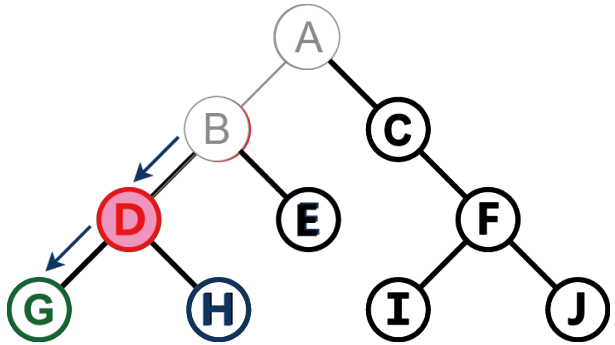
01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

STEP 3

先輸出樹根 D，而後拜訪左子節點，最後再拜訪右子節點。



目前輸出 A → B → D

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

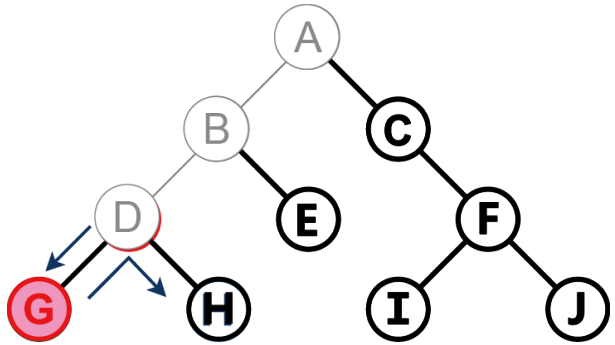
02 中序走訪方式

03 後序走訪方式

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

STEP 4

輸出 D 的左子節點 G。拜訪完 D 的左子樹後，再前往 D 的右子樹。



目前輸出 A → B → D → G

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

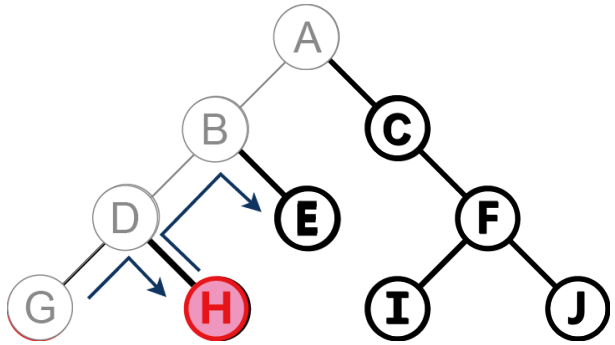
01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

STEP 5

輸出 D 的右子節點 H。當拜訪完 B 的左子樹後，再返回上一層，進入 B 的右子樹。



目前輸出 A → B → D → G → H

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

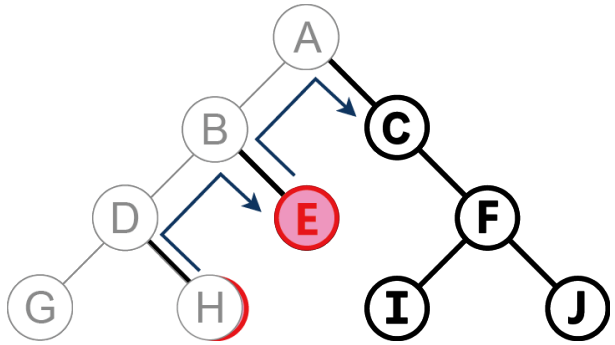
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 6

輸出 B 的右子節點 E。當拜訪完 A 的左子樹後，再返回上一層，進入 A 的右子樹。



目前輸出 A → B → D → G → H → E

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

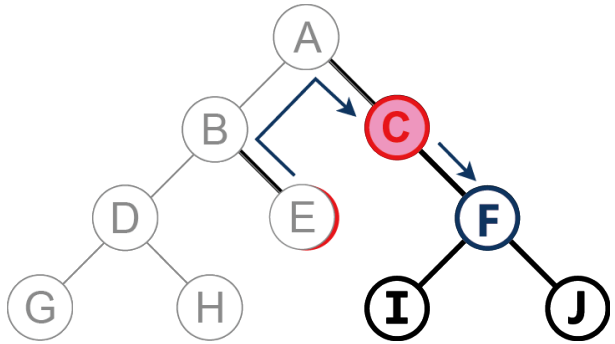
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 7

先輸出樹根 C，因 C 無左子樹，故直接拜訪 C 的右子樹節點。



目前輸出 A → B → D → G → H → E → C

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

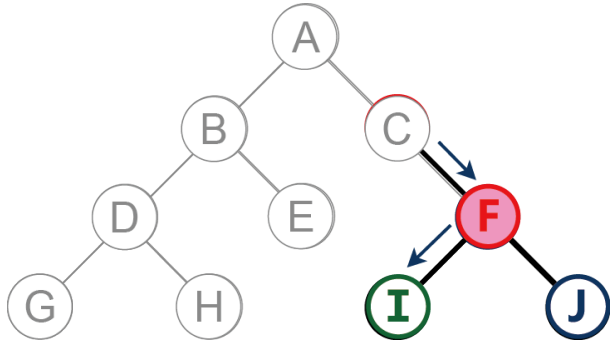
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 8

先輸出樹根F，而後拜訪左子節點，
最後再拜訪右子節點。



目前輸出 A → B → D → G → H → E → C → F

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

02 中序走訪方式

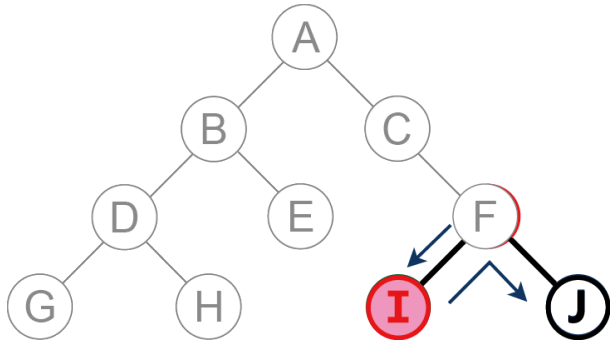
03 後序走訪方式

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

STEP 9

輸出 F 的左子節點 I。

拜訪完 F 的左子樹後，再前往 F 的右子樹。



目前輸出 A → B → D → G → H → E → C → F → I

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

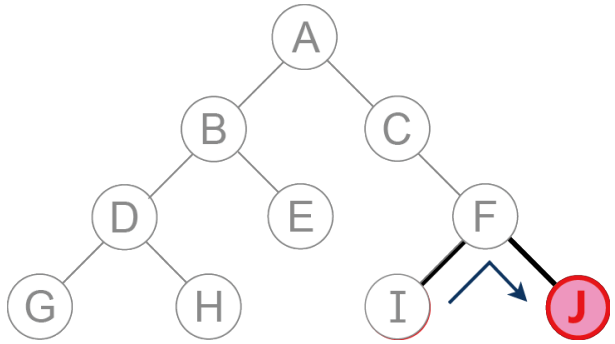
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 10

最後輸出 F 的右子節點 J，
完成二元樹的前序走訪。



目前輸出 A → B → D → G → H → E → C → F → I → J

二元樹的走訪

01 前序走訪方式

02 中序走訪方式 ▶

03 後序走訪方式

① 由**左子節點**優先輸出

② 而後輸出**樹根節點**

③ 再輸出**右子節點**

樹根輸出順序在中間，
故稱為中序追蹤。

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

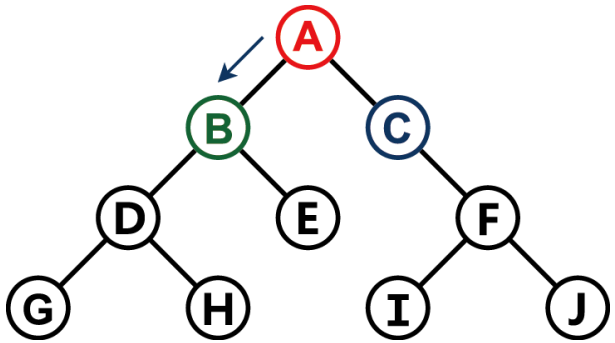
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 1

先拜訪 A 的左子樹，而後輸出樹根 A，最後再拜訪 A 的右子節點。



目前輸出 無

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

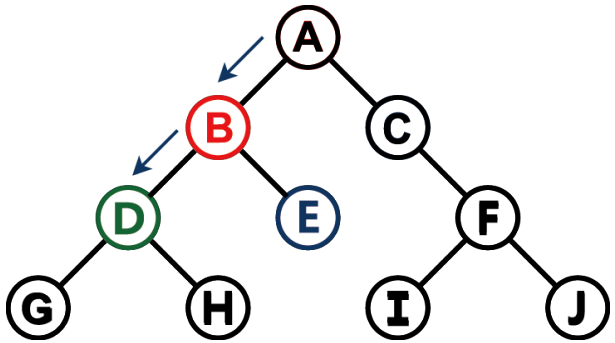
02 中序走訪方式

03 後序走訪方式

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

STEP 2

先拜訪 B 的左子樹，而後輸出樹根 B，最後再拜訪 B 的右子節點。



目前輸出 無

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

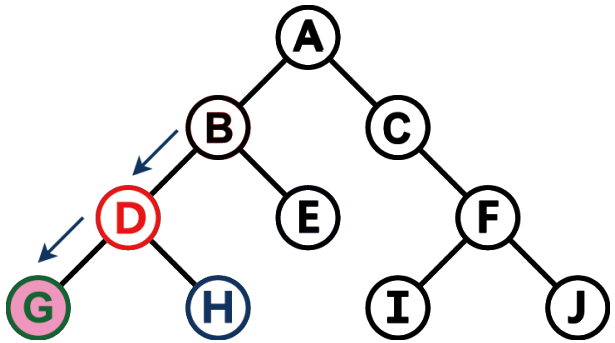
02 中序走訪方式

03 後序走訪方式

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

STEP 3

先拜訪 D 的左子樹，輸出左子節點 G。



目前輸出 G

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

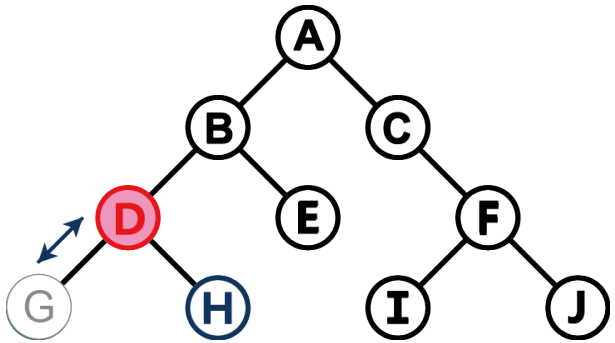
01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

STEP 4

而後輸出樹根節點 D。



目前輸出 G → D

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

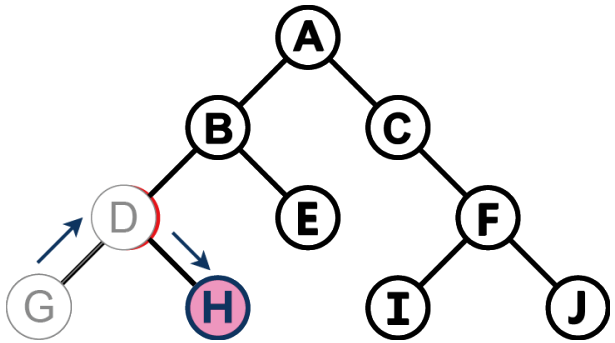
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 5

再拜訪 D 的右子樹，輸出右子節點 H。



目前輸出 G → D → H

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

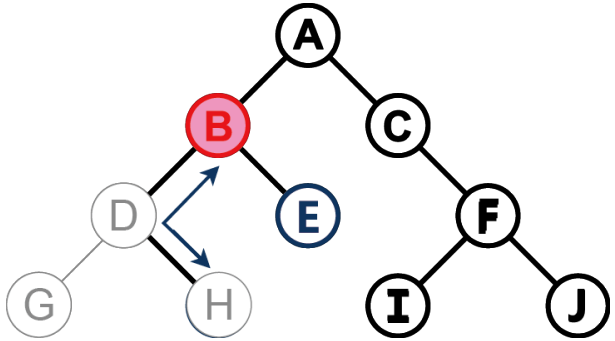
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 6

當拜訪完 B 的左子樹後，再返回上一層，輸出樹根節點 B。



目前輸出 G → D → H → B

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

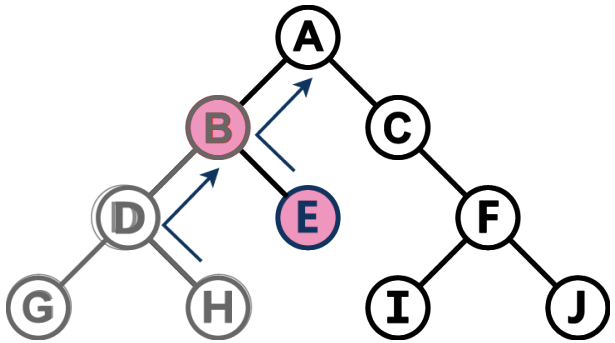
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 7

拜訪 B 的右子樹，輸出右子節點 E。
拜訪完 A 的左子樹後，返回上一層。



目前輸出 G → D → H → B → E

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

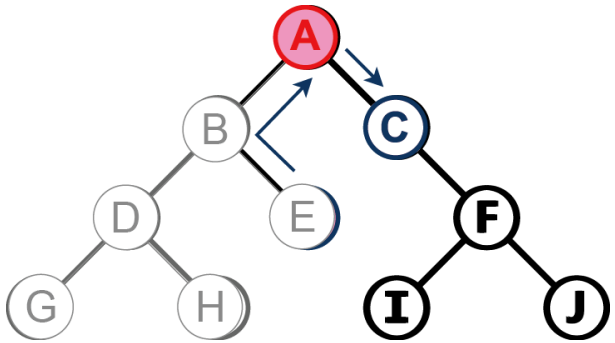
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 8

輸出樹根 A，再進入 A 的右子樹節點拜訪。



目前輸出 G → D → H → B → E → A

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

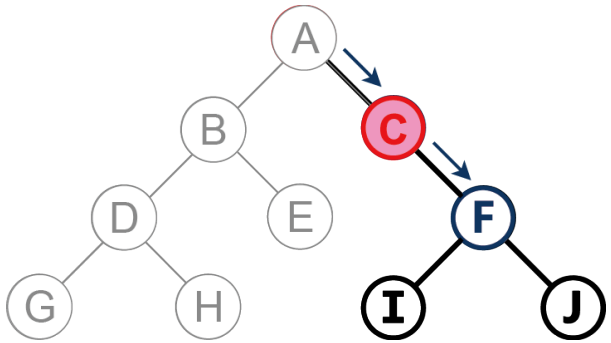
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 9

因 C 無左子樹，故先輸出樹根節點 C，再進入 C 的右子樹節點拜訪。



目前輸出 G → D → H → B → E → A → C

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

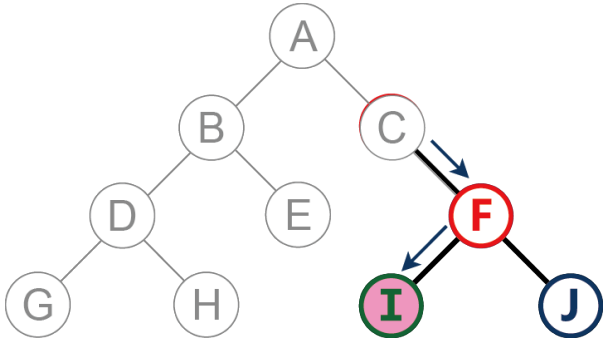
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 10

先拜訪 F 的左子樹，輸出左子節點 I。



目前輸出 G → D → H → B → E → A → C → I

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

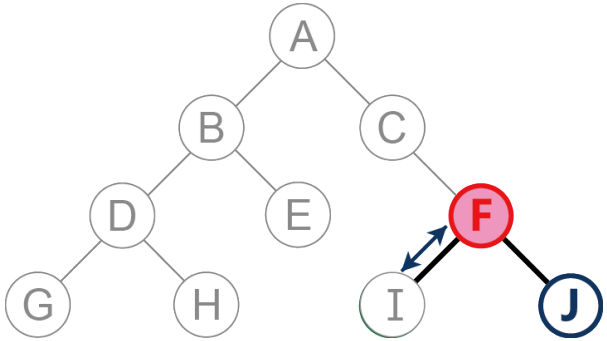
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 11

而後輸出樹根節點 F。



目前輸出 G → D → H → B → E → A → C → I → F

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

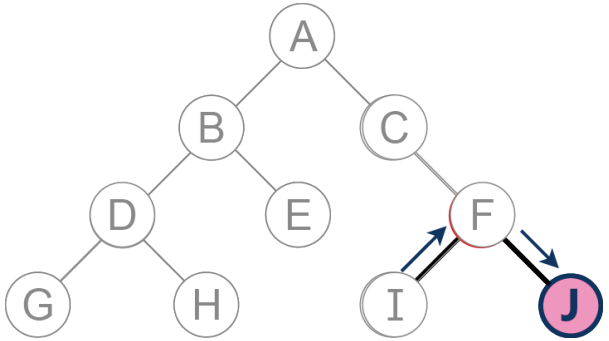
02 中序走訪方式

03 後序走訪方式

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

STEP 12

最後輸出 F 的右子節點 J，
完成二元樹的中序走訪。



目前輸出 G → D → H → B → E → A → C → I → F → J

二元樹的走訪

01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

① 由**左子節點**優先輸出

② 而後輸出**右子節點**

③ 再輸出**樹根節點**

樹根輸出順序在最後，
故稱為後序追蹤。

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

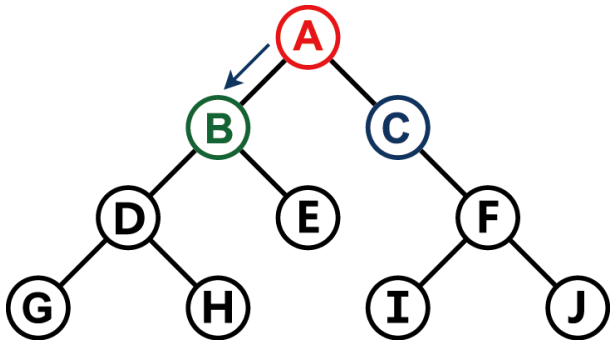
02 中序走訪方式

03 後序走訪方式

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

STEP 1

先拜訪 A 的左子節點，而後拜訪 A 的右子節點，最後再返回輸出樹根節點 A。



目前輸出 無

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

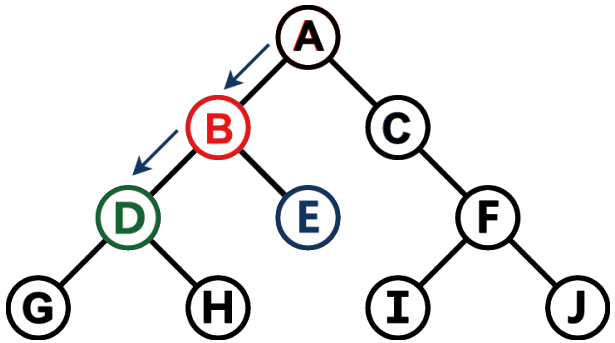
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 2

先拜訪 B 的左子節點，而後拜訪 B 的右子節點，最後再返回輸出樹根 B。



目前輸出 無

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

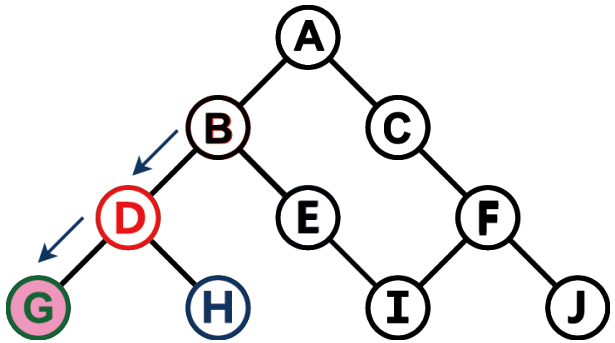
01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

STEP 3

先輸出 D 的左子節點 G。



目前輸出 G

：輸出值
：樹根節點
：左子節點
：右子節點
：已輸出節點

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

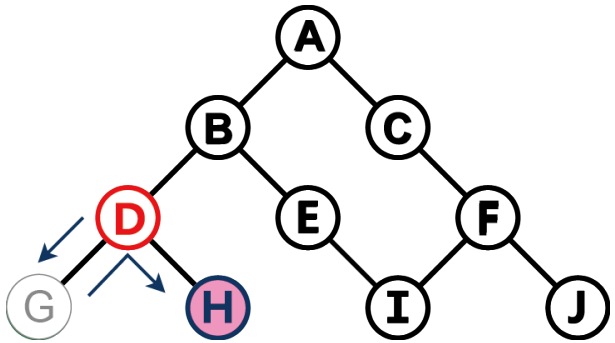
01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

STEP 4

而後輸出 D 的右子節點 H。



目前輸出 G → H

- ：輸出值
- ：樹根節點
- ：左子節點
- ：右子節點
- ：已輸出節點

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

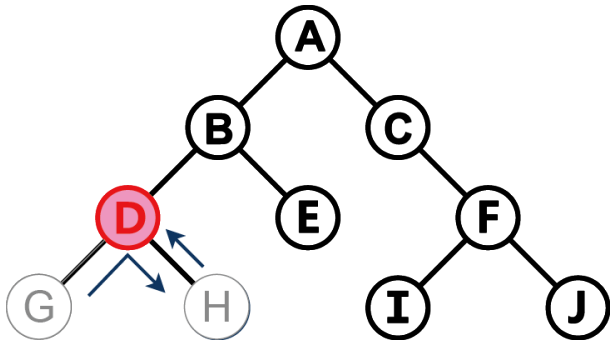
01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

STEP 5

輸出樹根節點 D。



目前輸出 G → H → D

：輸出值
：樹根節點
：左子節點
：右子節點
：已輸出節點

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

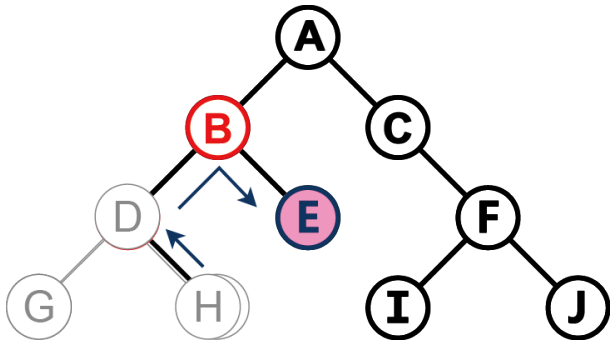
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 6

當拜訪完 B 的左子樹後，再返回上一層，輸出 B 的右子節點 E。



目前輸出 G → H → D → E

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

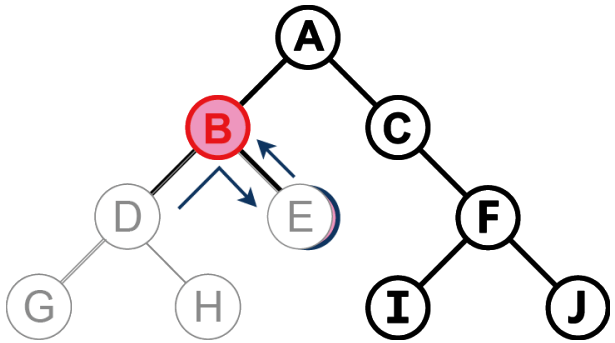
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 7

最後再輸出樹根節點 B。



目前輸出 G → H → D → E → B

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

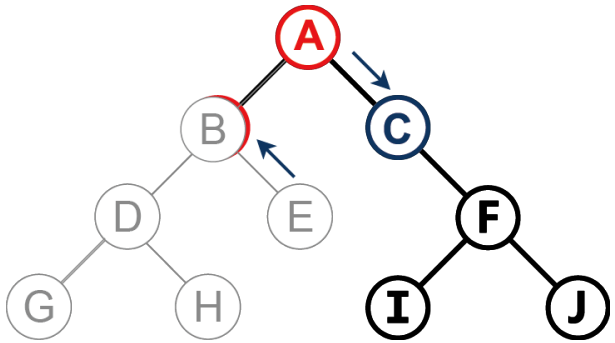
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 8

當拜訪完 A 的左子樹後，再返回上一層，拜訪 A 的右子節點。



目前輸出 G → H → D → E → B

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

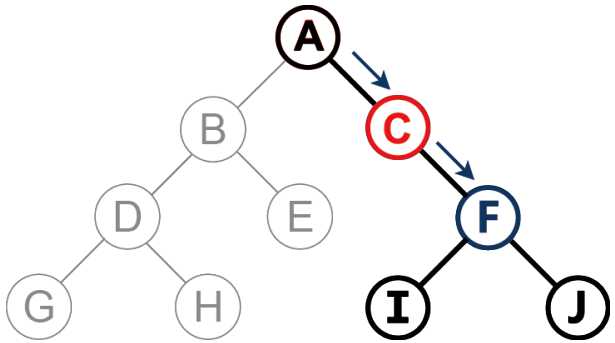
01 前序走訪方式

02 中序走訪方式

03 後序走訪方式

STEP 9

因 C 無左子樹，先進入 C 的右子節點拜訪。



目前輸出 G → H → D → E → B

●：輸出值
○：樹根節點
○：左子節點
○：右子節點
○：已輸出節點

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

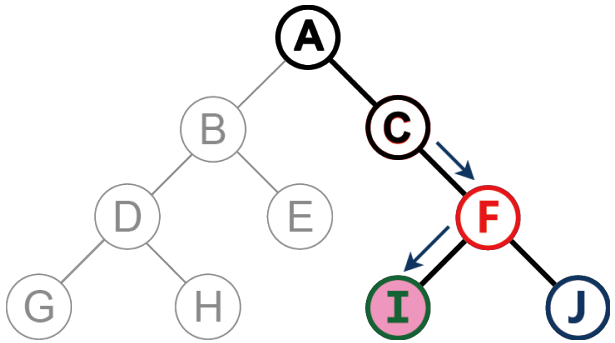
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 10

先輸出 F 的左子節點 I。



目前輸出 G → H → D → E → B → I

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

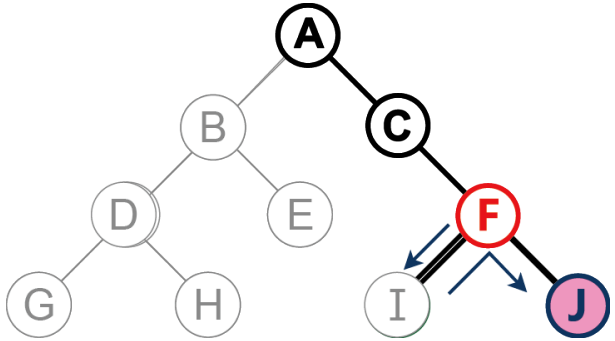
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 11

而後輸出 F 的右子節點 J。



目前輸出 G → H → D → E → B → I → J

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

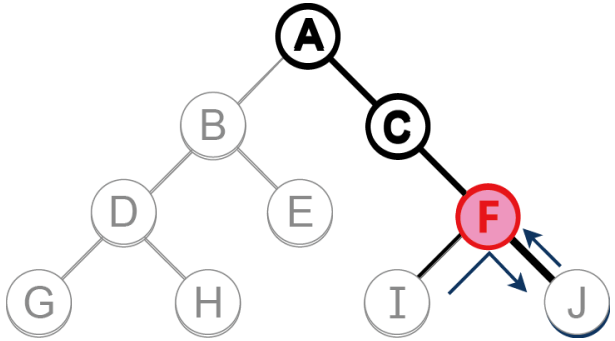
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 12

最後再輸出樹根節點 F。



目前輸出 G → H → D → E → B → I → J → F

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

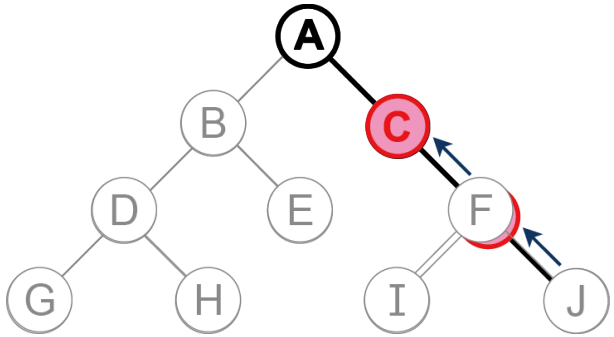
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 13

當拜訪完 C 的右子樹後， 再返回上一層，輸出樹根節點 C。



目前輸出 G → H → D → E → B → I → J → F → C

1-3 Tree

樹 ▸ 1-3-2 二元樹 (Binary Tree)

二元樹的走訪

01 前序走訪方式

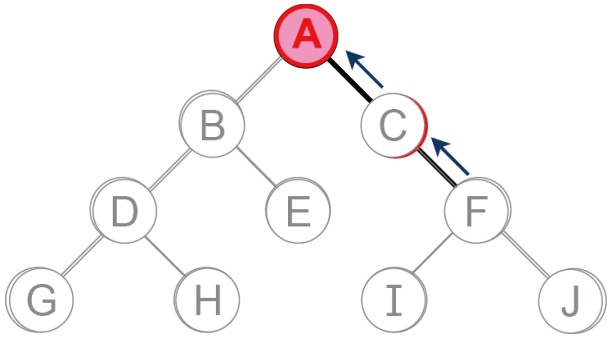
02 中序走訪方式

03 後序走訪方式

● ：輸出值
○ ：樹根節點
○ ：左子節點
○ ：右子節點
○ ：已輸出節點

STEP 14

當拜訪完 A 的右子樹後，再返回上一層，輸出樹根節點 A，完成二元樹的後序走訪。



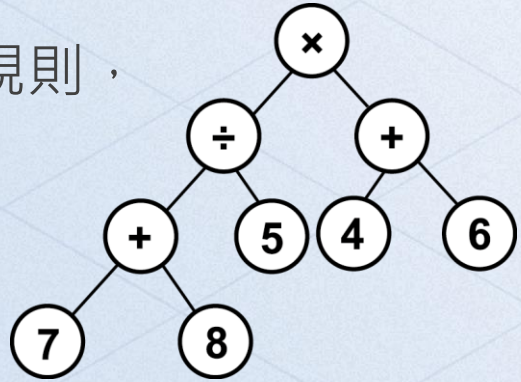
目前輸出 G → H → D → E → B → I → J → F → C → A

資料結構的日常應用(02_46)

Question

請同學依據「**中序追蹤**」的規則，
計算此圖走訪後的結果。

（將全部節點的數值與運算符號走訪完畢後，
再將算式依四則運算規則計算結果。）



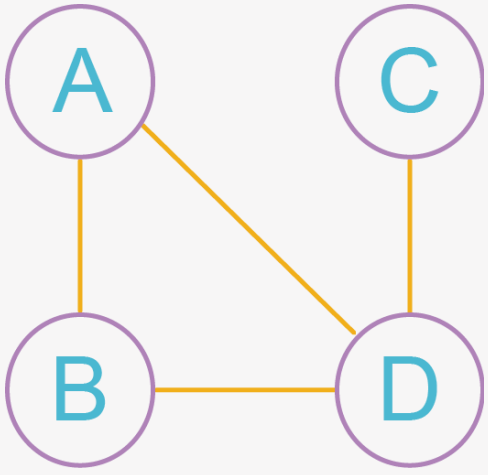
1-4 Graph

圖

由「頂點（Vertex，或稱節點）」和「邊（Edge）」構成。

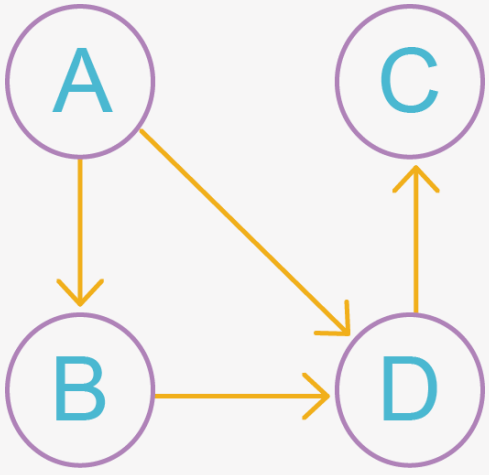
依**方向性**分為

無向圖 邊不存在方向性



1-4-1

有向圖 邊存在方向性

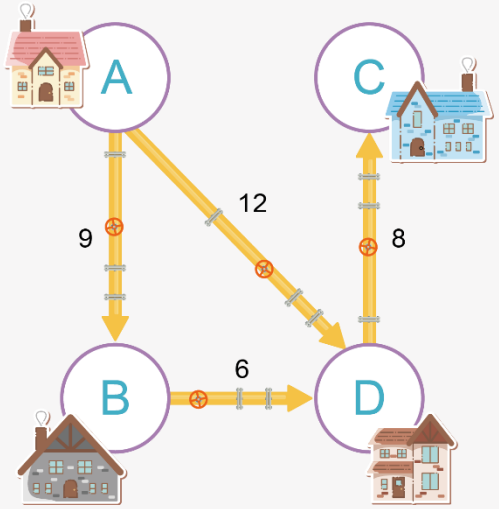


1-4-2

1-4 Graph 圖

而邊也可加上**距離**、**時間**、**成本**、**數量**等數值作為**權重**（Weight）以表示相鄰兩節點的關係。

1-4-3 / 邊可以加上權重



- A、B、C、D 四座村莊，各村莊間架設數條自來水管線。

「**有向邊**」跟「**權重**」就可以用來表示管線水流的方向與距離。

1-4 Graph

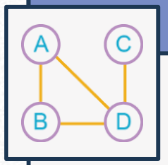
圖

透過下列二維陣列，記錄與儲存無向圖和有向圖的資料。

若有 n 個節點，則我們需 n×n 的二維陣列錄資料。

1-4 Graph

表 1-1 / 將圖 1-4-1 的無向圖轉為矩陣方式記錄



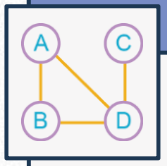
	A	B	C	D
A	**0**	1	0	1
B	1	**0**	0	1
C	0	0	**0**	1
D	1	1	1	**0**

以左側的行，作為節點的出發

上方欄位的節點名稱作為終點

1-4 Graph

表 1-1 / 將圖 1-4-1 的無向圖轉為矩陣方式記錄



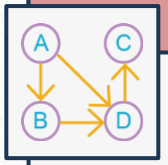
	A	B	C	D
A	**0**	1	0	1
B	1	**0**	0	1
C	0	0	**0**	1
D	1	1	1	**0**

以 A 行來說

- 若存在有 $\overline{AB}$、$\overline{AD}$，則在該列對應的欄位表格中，**填入數值 1**
- 若**不存在**有邊的關係，則**填入數值 0**。

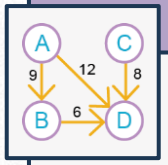
1-4 Graph

表 1-2 / 將圖 1-4-2 的有向圖轉為矩陣方式記錄。



	A	B	C	D
A	**0**	1	0	1
B	0	**0**	0	1
C	0	0	**0**	0
D	0	0	1	**0**

表 1-3 / 將圖 1-4-3 的有向圖與權值轉為矩陣方式記錄。



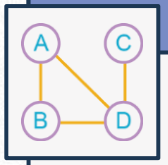
	A	B	C	D
A	**0**	1	0	1
B	0	**0**	0	1
C	0	0	**0**	0
D	0	0	1	**0**

透過相同方式，我們將**圖 1-4-2** 和**圖 1-4-3** 的，分別轉為二維陣列的**表 1-2** 和**表 1-3** 儲存。

1-4 Graph 圖

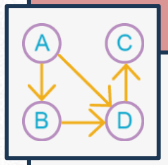
透過表 1-1 與表 1-2 的比較 ：

表 1-1 / 將圖 1-4-1 的無向圖轉為矩陣方式記錄



	A	B	C	D
A	0	1	0	1
B	1	0	0	1
C	0	0	0	1
D	1	1	1	0

表 1-2 / 將圖 1-4-2 的有向圖轉為矩陣方式記錄。



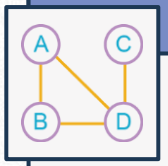
	A	B	C	D
A	0	1	0	1
B	0	0	0	1
C	0	0	0	0
D	0	0	1	0

無向圖的數值在陣列中的 **45 度軸線上呈現對稱**，我們將其稱為**對稱矩陣**。

1-4 Graph 圖

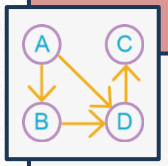
透過表 1-1 與表 1-2 的比較 ：

表 1-1 / 將圖 1-4-1 的無向圖轉為矩陣方式記錄



	A	B	C	D
A	0	1	0	1
B	1	0	0	1
C	0	0	0	1
D	1	1	1	0

表 1-2 / 將圖 1-4-2 的有向圖轉為矩陣方式記錄。



	A	B	C	D
A	0	1	0	1
B	0	0	0	1
C	0	0	0	0
D	0	0	1	0

有向圖則因**邊存在方向性**，不一定會呈現對稱矩陣。
如表所示，只有 $\overrightarrow{AB}$ ，並沒有 $\overrightarrow{BA}$ 。

1-4 Graph

導航軟體路線規劃

透過地圖起點與終點，加入各種路線規劃。

兩地（點）間可呈現不同**路線**（邊），及其所需**時間和里程**（權重）。

1-4 Graph

圖

捷運的站間時間

以點和邊加上權重的方式，**以圖來呈現**。

或是**將圖轉化為矩陣方式**（陣列）來表示或儲存。

表 1-4 / 以矩陣
表示站間的行駛時間

車站編號	R3	R4	R4A	R5
R3	0	2	5	8
R4	2	0	3	6
R4A	5	3	0	3
R5	8	6	3	0

六度分隔理論(02_19)

六度分隔理論

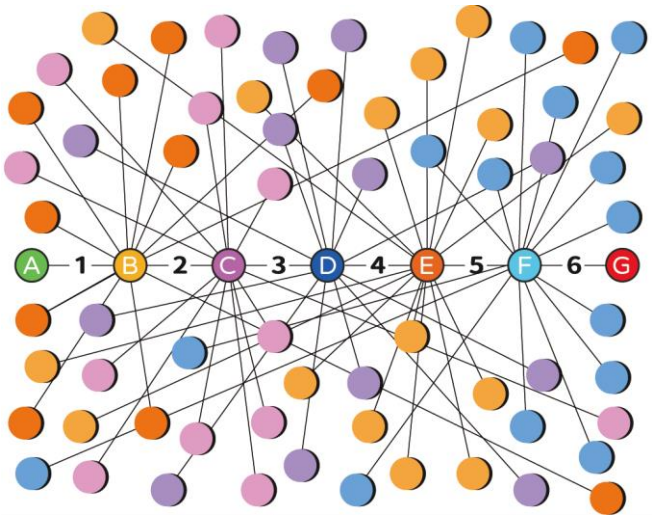
六度分隔理論又稱「**小世界現象**」或「**小世界效應**」。

其概念由哈佛大學的社會心理學教授**史丹利．米爾格蘭**在 1967 年所提出

- 指世上任何原先不相識的兩個人，透過少數朋友作為中介的連接橋梁，就可以互相認識對方。

六度分隔理論

然而，隨著近年來社群網路的蓬勃發展，
使得人與人的關係因為網路變得更加密集且錯綜。



1-4-6

運用圖的點與邊的關係，呈現人（節點）與人（節點）之間的連結（邊）關係。

如互不相識的 A 和 G 兩者間的連結，存在有 5 位好友（節點 B、C、D、E、F）、6 個空間（邊）的距離，可呈現出六度分隔的概念。

六度分隔理論

只要按下「加朋友」，即可串起彼此在虛擬世界的關係。

Facebook 更在其社群媒體上發現好友之間的關係**平均只需要 3.57 人的距離**。

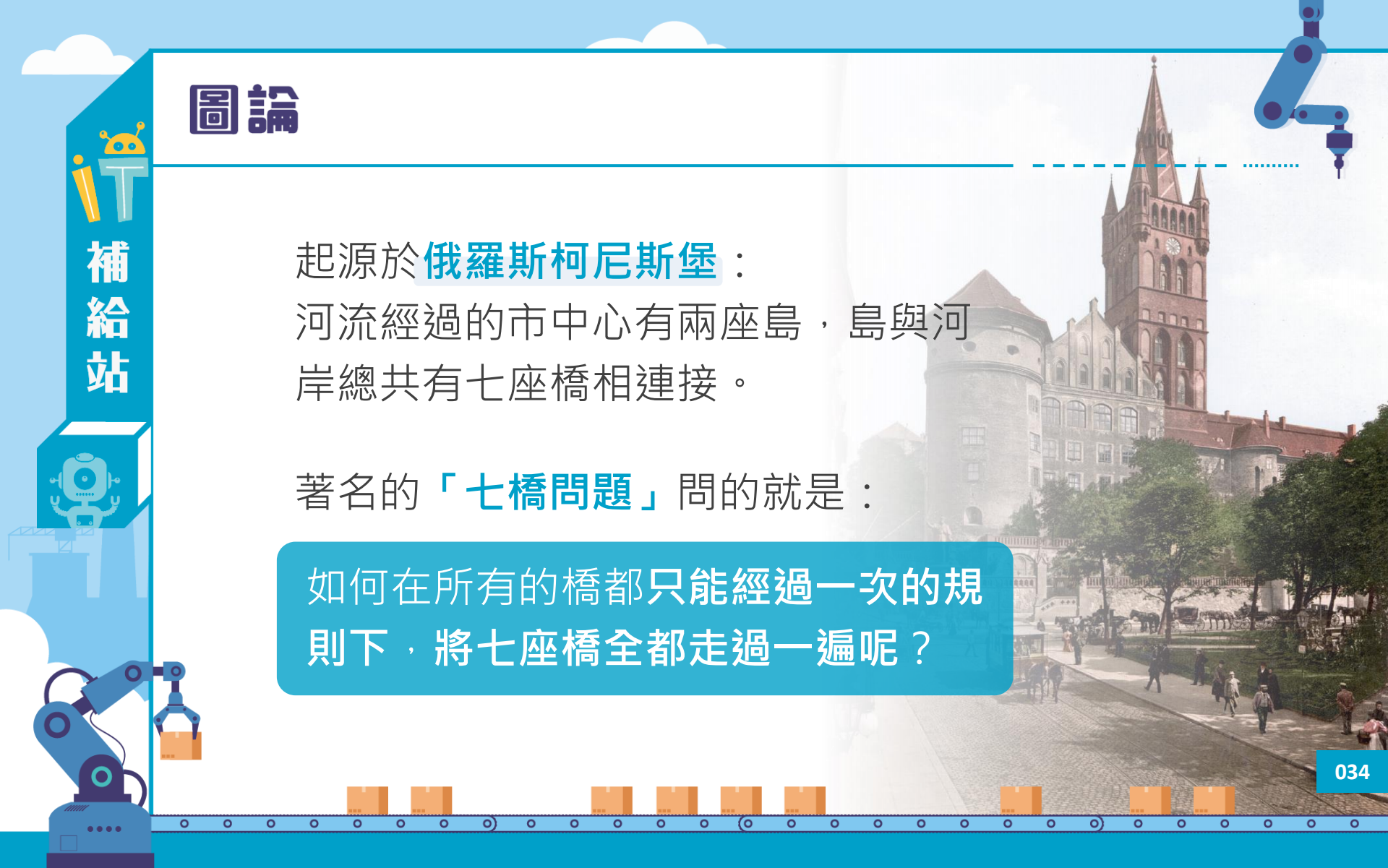
同學們是否覺得螢光幕上的偶像因此而離我們很近呢？

圖論

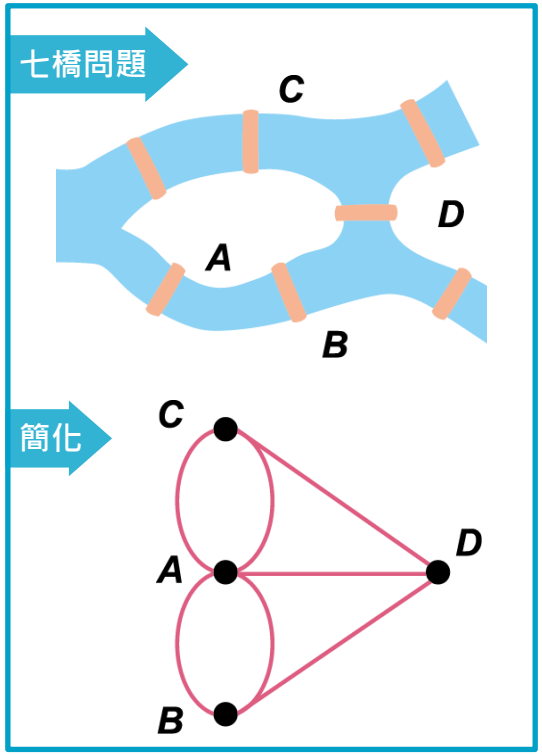
起源於**俄羅斯柯尼斯堡**：
河流經過的市中心有兩座島，島與河岸總共有七座橋相連接。

著名的**「七橋問題」**問的就是：

如何在所有的橋都**只能經過一次的規則下，將七座橋全都走過一遍呢？**



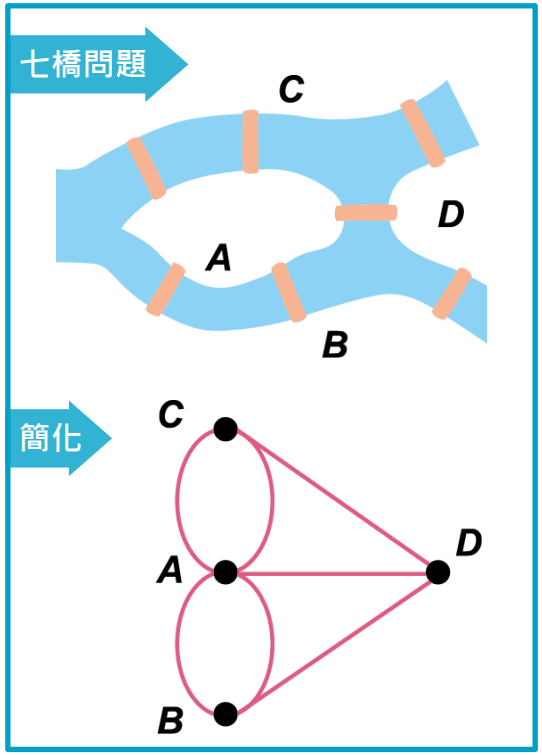
圖論



著名的瑞士數學家**尤拉**找到了要在**不重複**走訪橋（邊）的前提下，**將所有橋都走過一遍的兩項條件**：

① **每個節點都有偶數個邊。**
（起點和終點必相同）

② **只存在兩個節點是有奇數個邊，其餘節點皆是偶數個邊。**
（起點和終點必不同）



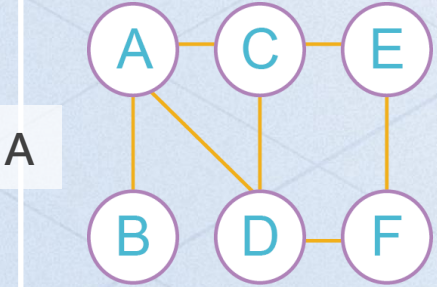
只要滿足上述的任一項條件，就可以在不重複走訪每一條邊的前提下，造訪所有的邊。

七橋問題在這兩項條件上都不符合，因此「在不重複走訪每一條邊的前提下，造訪所有的橋」的可能性並不存在。

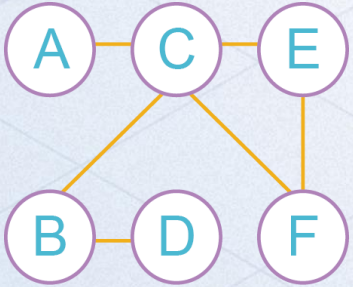
Question

試問下列圖形，哪些能**以一筆畫**（每一邊僅能走訪一次）的方式，走訪所有節點？

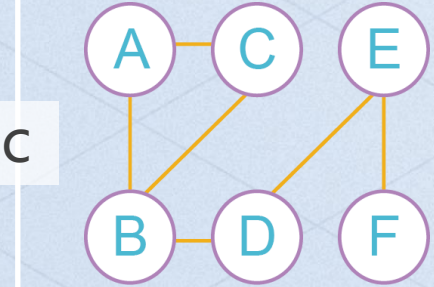
A



B



C



D