

資料結構

# Fundamentals of Data Structures

1

## LECTURE 7 : TREE

授課教師：周兆龍

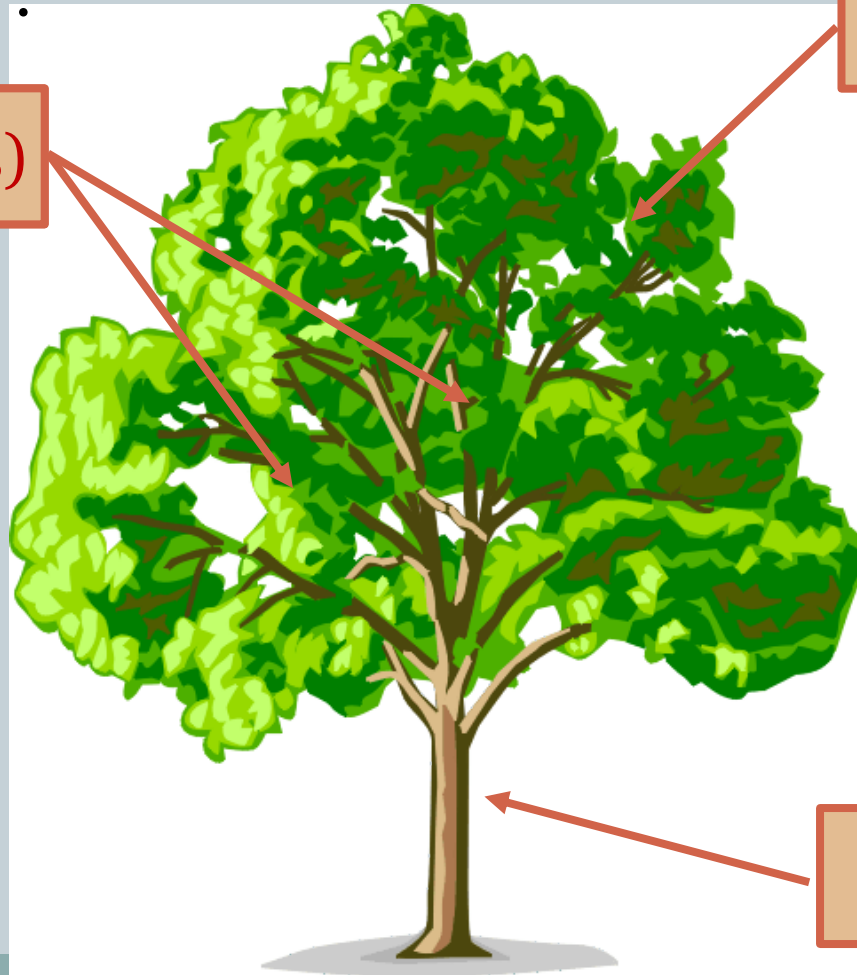
# 1 Basics

2

- 自然界的樹：

分枝(Branches)

樹葉(Leaves)

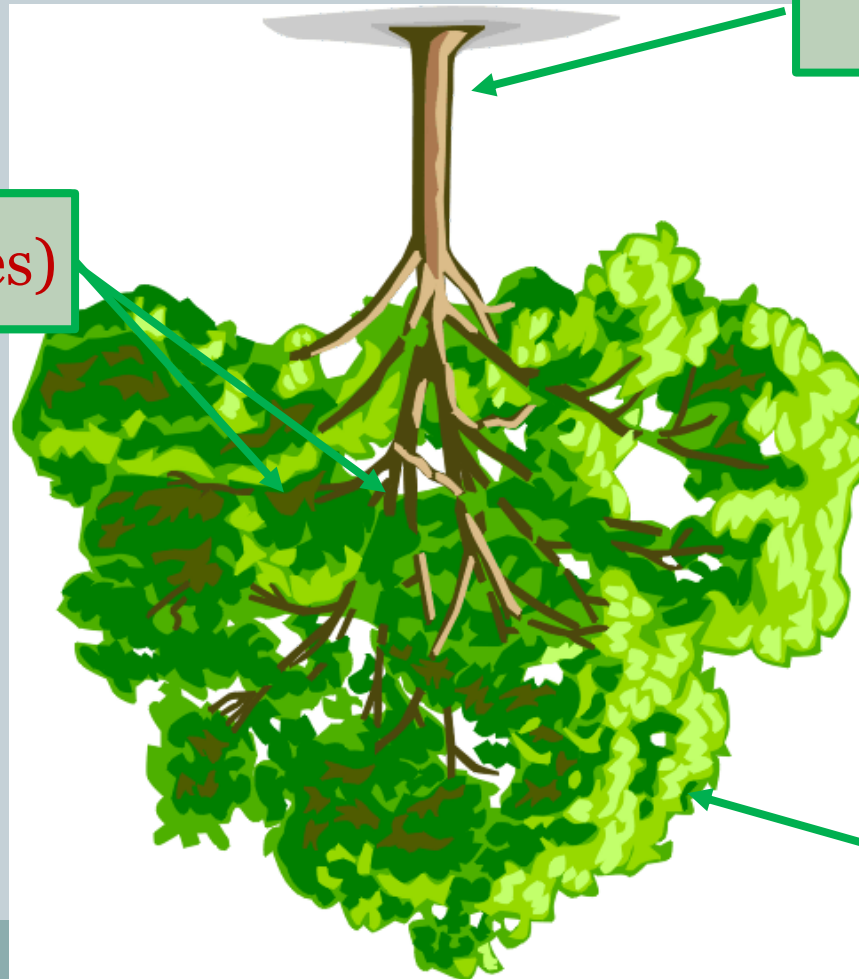


樹根(Root)

# 1 Basics

3

- 資料結構中的「樹」



樹根(Root)

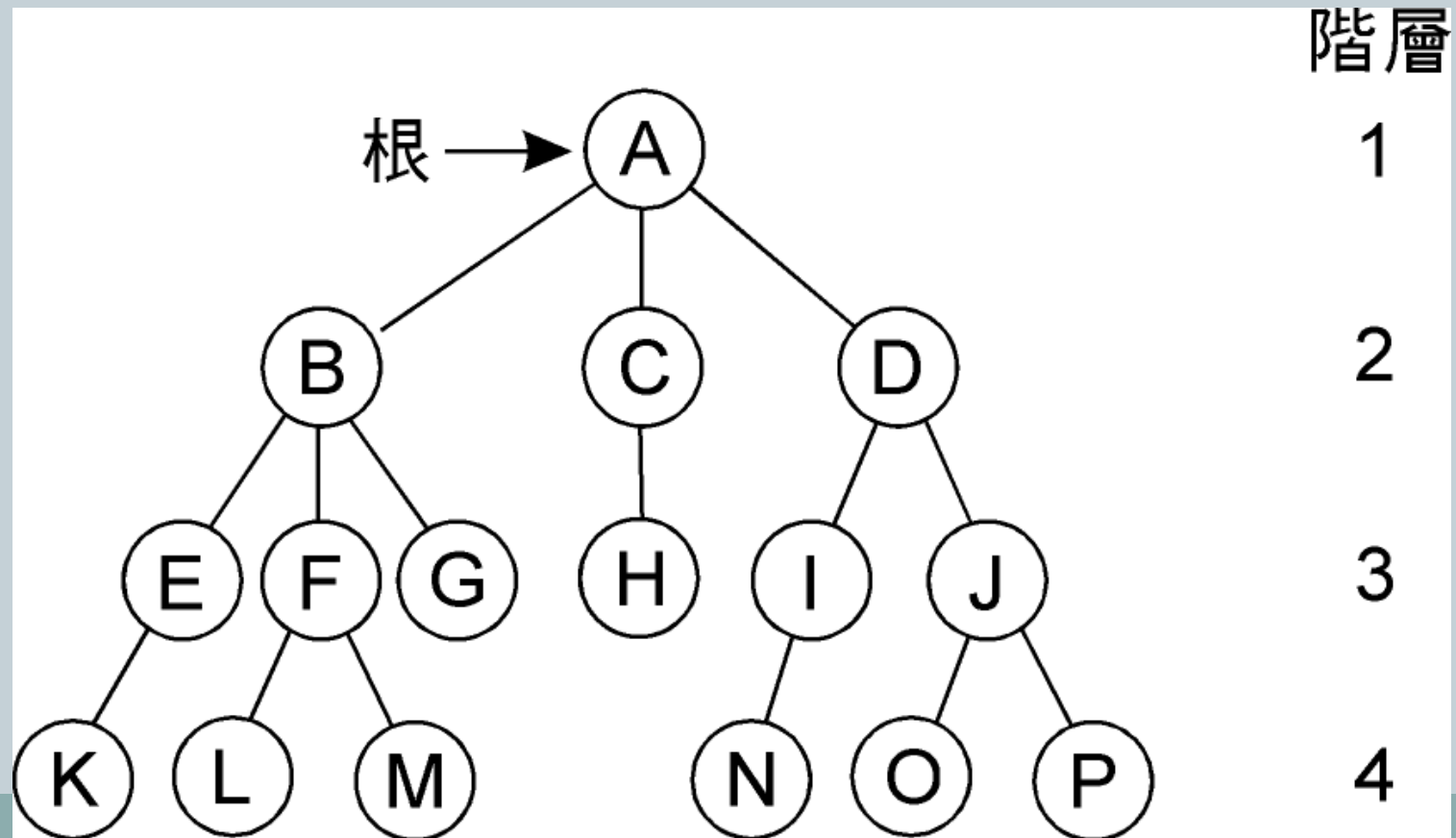
分枝(Branches)

樹葉(Leaves)

# 1 Basics

4

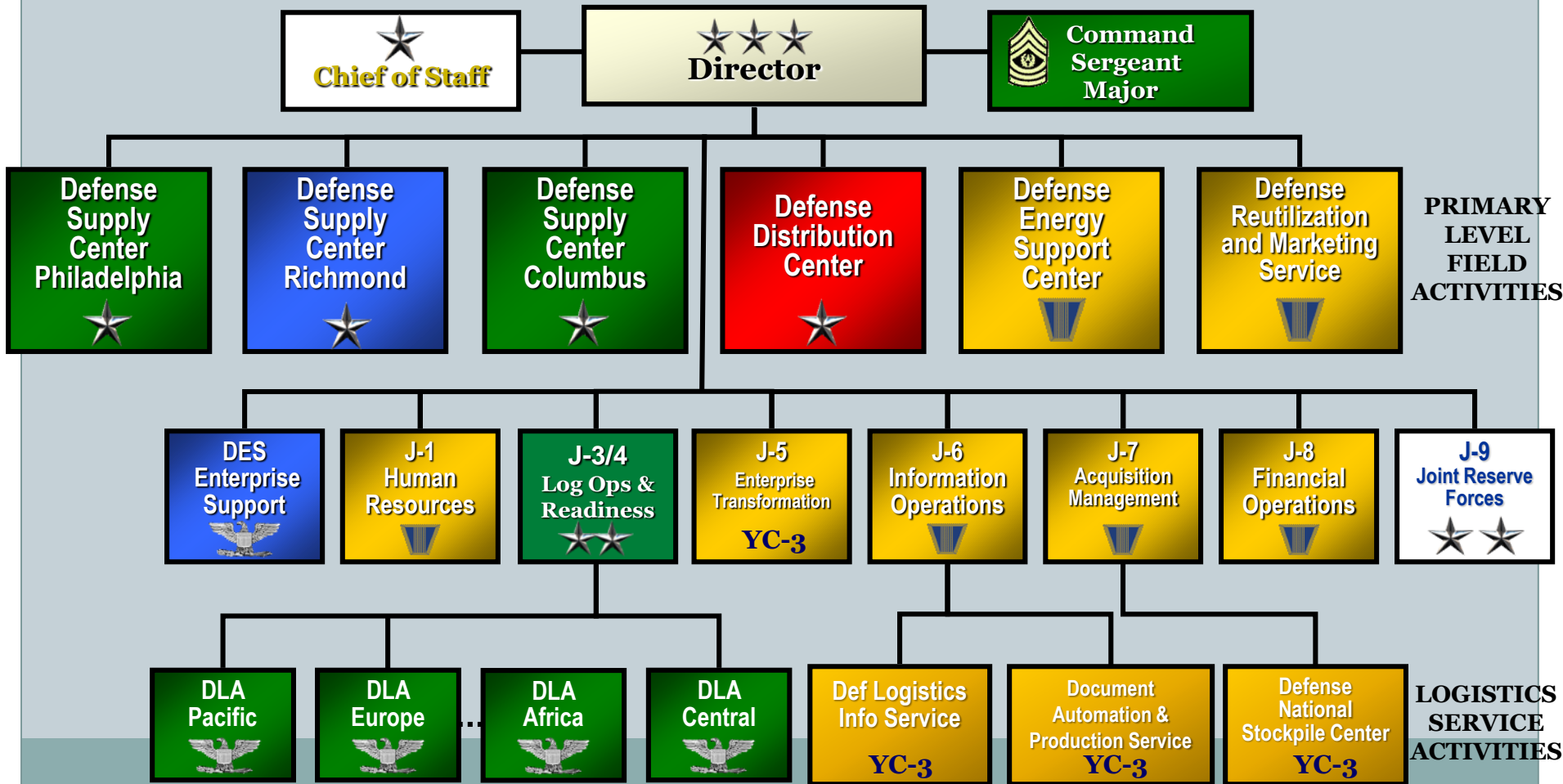
- 樹(tree) 是一種重要的離散結構 (discrete structure) ，它提供一種「具有層次關係」的概念來結構資料。



# 1 Basics

5

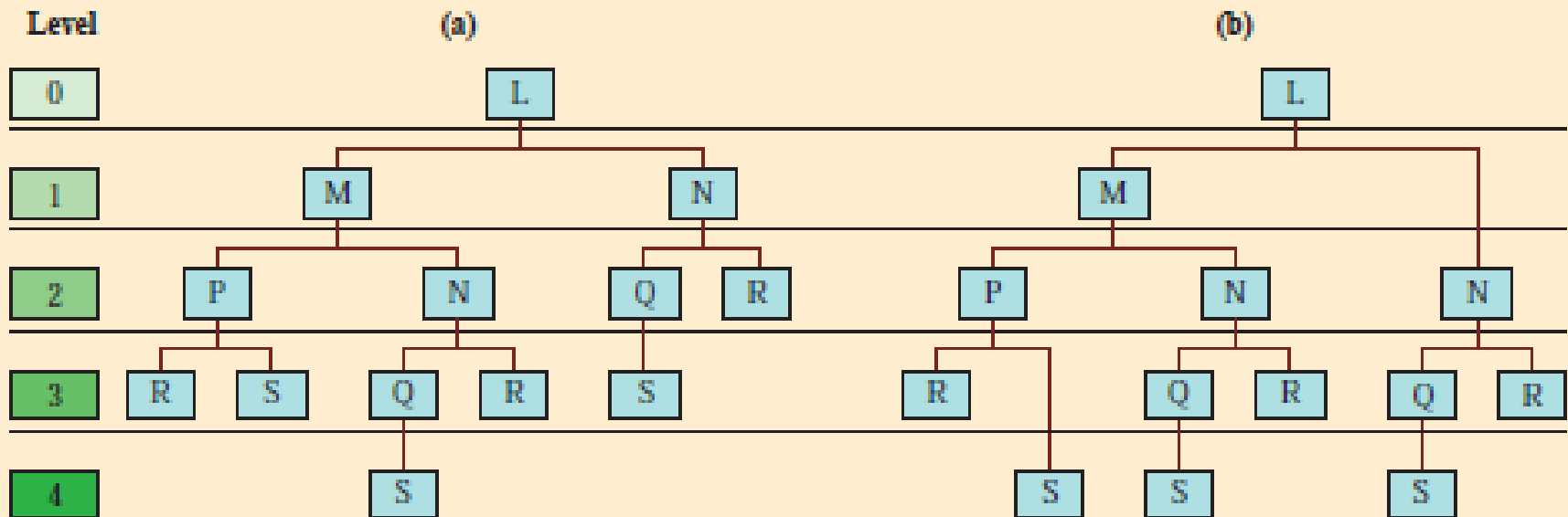
- 例如：組織圖



# 1 Basics

6

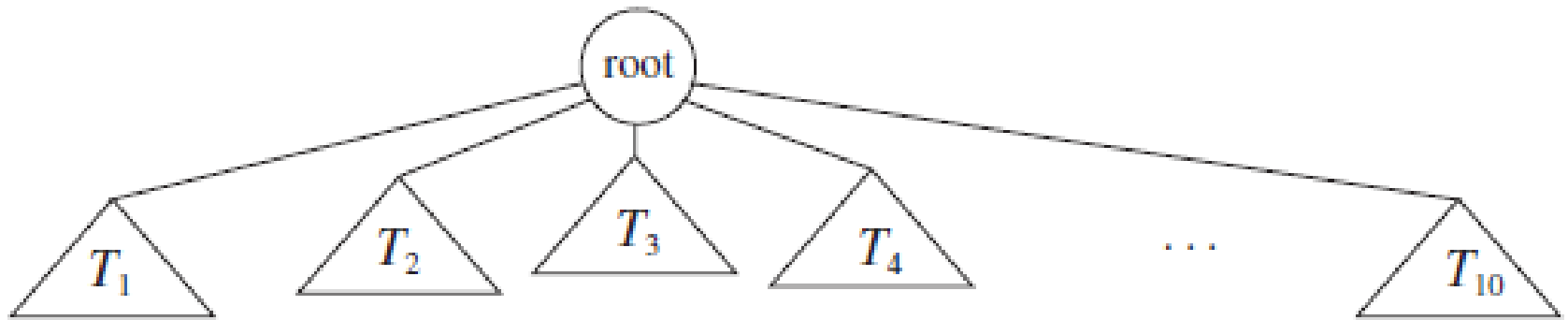
- 例如：產品成分圖(BOM)



# 1 Basics

7

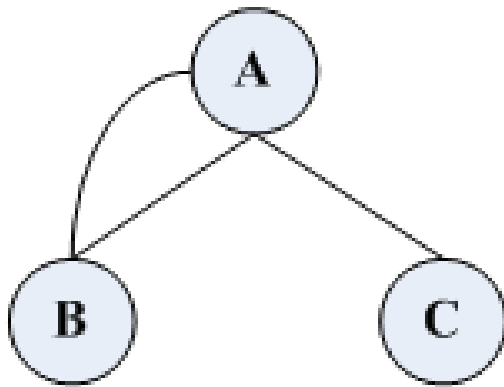
- 樹狀結構的定義：
  - 不可以為空，至少一個或多個節點的有限集合，並滿足以下條件：
    - ✦ (1) 存在一特殊節點  **$R$** ，稱為樹根(**root**)。
    - ✦ (2) 其它節點可分割成  **$n$**  個互斥集合： **$T_1, T_2, \dots, T_n$** ， **$n \geq 0$** ，而  **$T_1, T_2, \dots, T_n$**  皆為樹，稱其為  **$R$**  的子樹(**subtree**)。



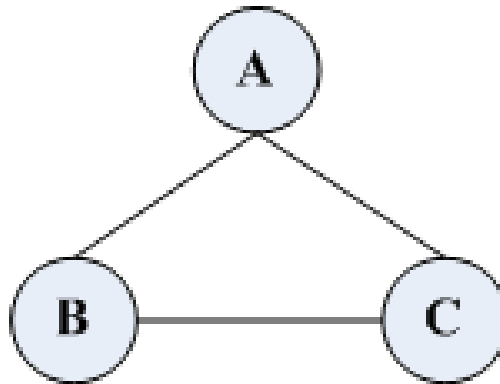
# 1 Basics

8

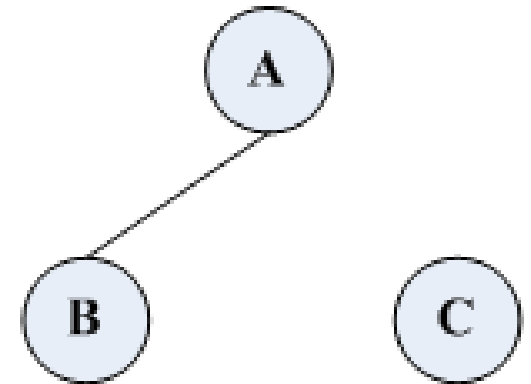
- 樹狀結構為**非線性(Non-linear)**的資料結構。
- 樹狀結構中**不可以**含有**迴圈**、**重邊**或**不相連**的情況。  
以下為非樹狀結構的例子。



迴圈



重邊



不相連

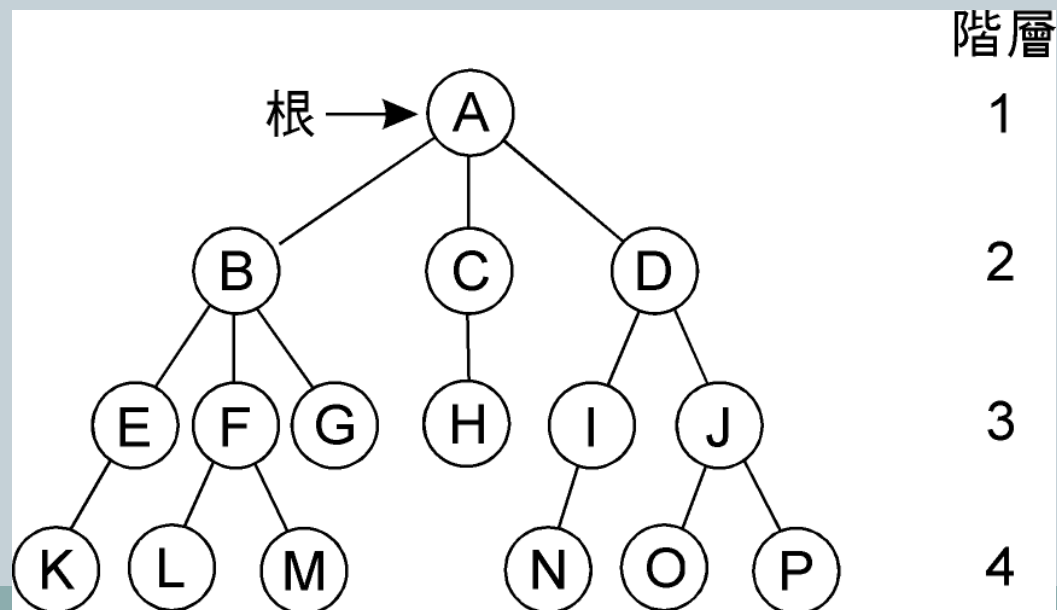


# 1 Basics

9

- 樹狀結構的專用術語：

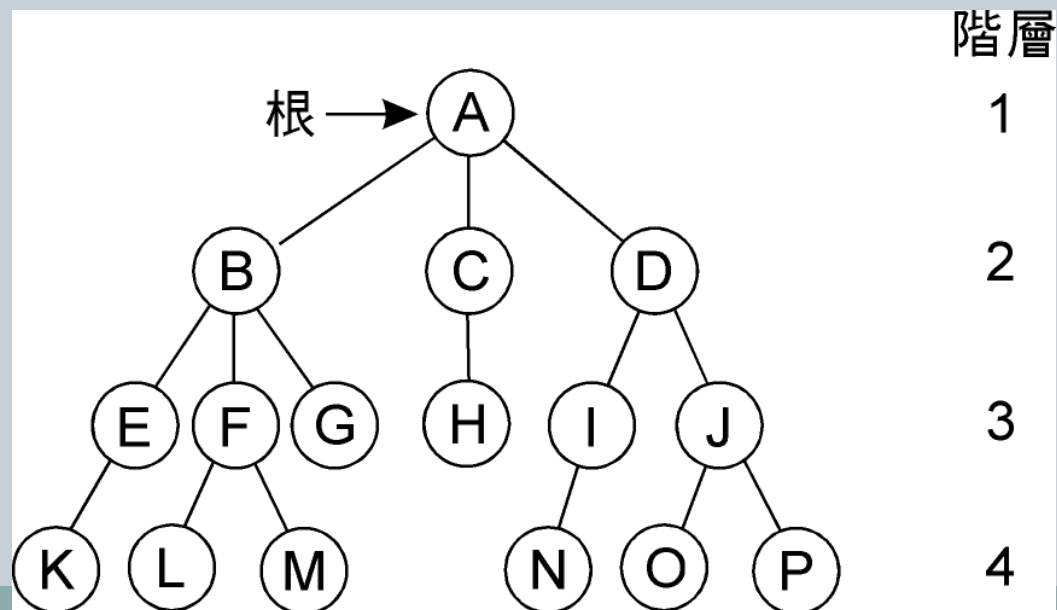
- 節點(node)：樹的節點代表著某項資料，A, B, C,...,O, P都是節點。
- 樹根(root)：每一棵樹的最上層的節點，稱為樹根，而A就是為這棵樹的樹根。



# 1 Basics

10

- 樹狀結構的專用術語：
  - 子樹(subtree)：把樹根A去掉之後，就剩下以B、C及D為三棵子樹。
  - 樹林(forest)：是由 $N \geq 0$ 個互斥子樹的集合。把樹根A去掉之後，剩下的部分( $B \cup C \cup D$ )就叫樹林。

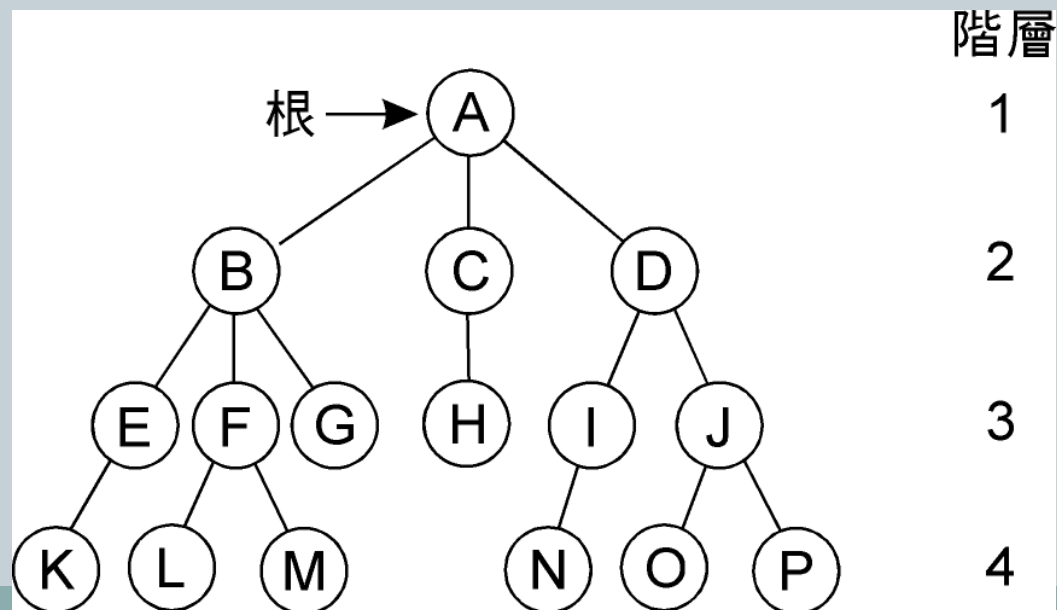


# 1 Basics

11

- 樹狀結構的專用術語：

- 階度(level)：表示節點的階層位置。樹根A的階度是1，B的階度是2，K的階度是4。
- 高度(height)或深度(depth)：一棵樹中節點的最大階度就是高度(深度)，例如此樹的高度(深度)為4。

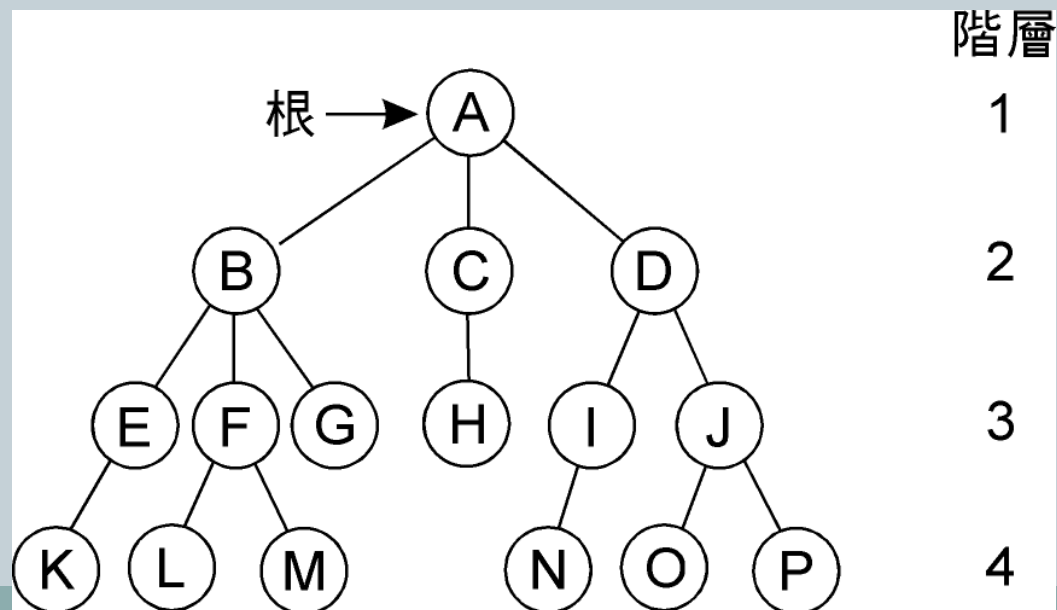


# 1 Basics

12

- 樹狀結構的專用術語：

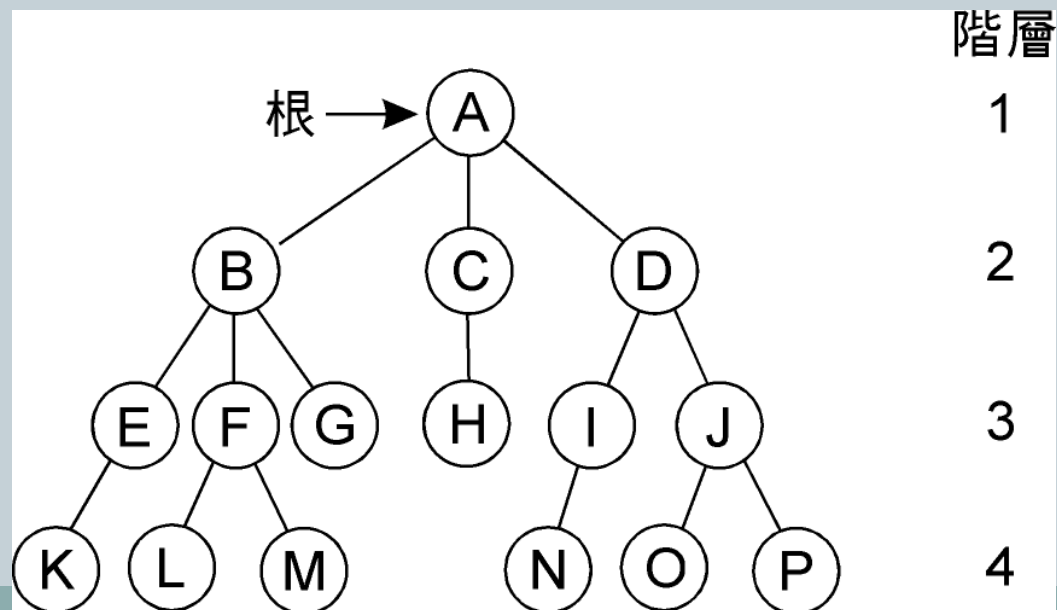
- 父節點(parent)：每一個節點的上一個節點就是父節點，B節點的父節點就是A。
- 子節點(children)：每一個節點的下一個節點就是子節點，D節點的子節點就是I、J。



# 1 Basics

13

- 樹狀結構的專用術語：
  - 兄弟(sibling)：有相同父節點的節點稱為兄弟，例如E、F、G為兄弟，B、C、D為兄弟。
  - 子孫(descendant)：由樹根往下至某個節點之前所經過的節點，稱為的下一個節點就是子節點，D節點的子節點就是I、J。

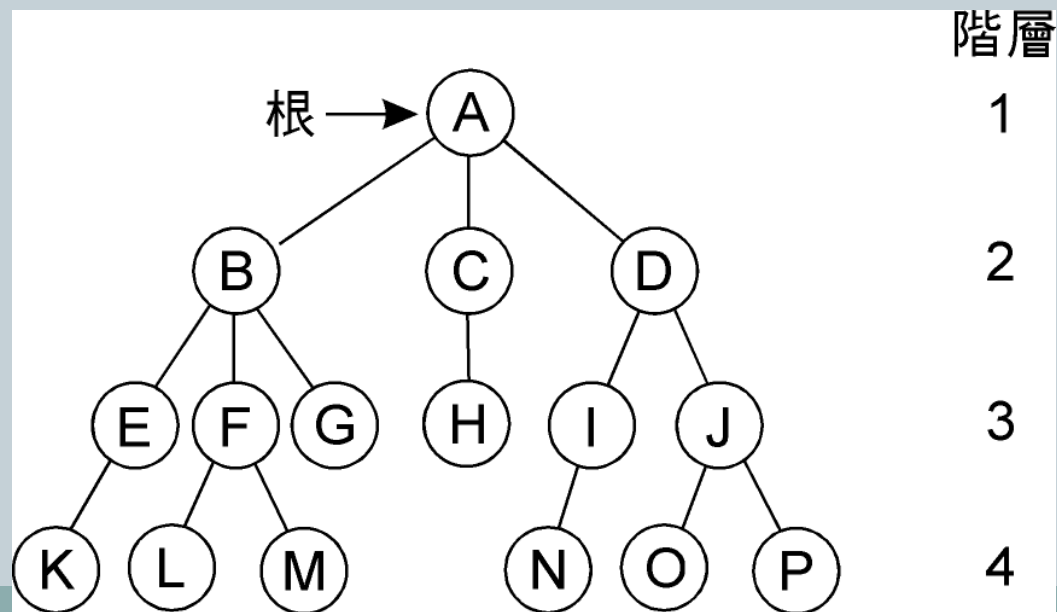


# 1 Basics

14

- 樹狀結構的專用術語：

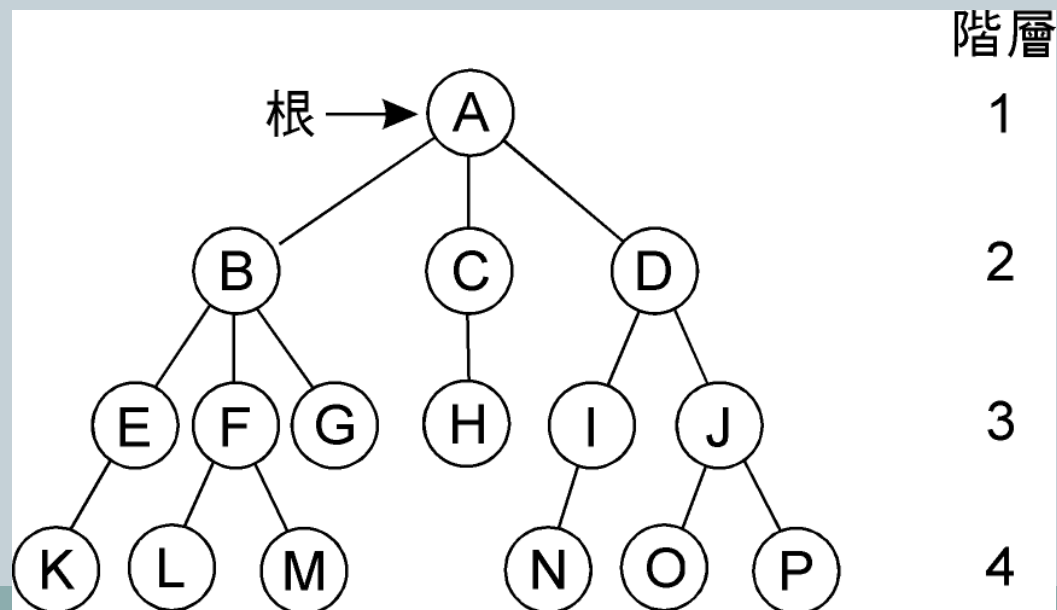
- 分支度(**degree**)：一個節點的子樹之個數稱為該節點的分支度。例如：A的分支度為**3**，D的分支度是**2**。
- 終端節點(**terminal node**)：分支度為**0**的節點，這棵樹的終端節點(又稱做**樹葉Leaf**)。共有**K, L, M, G, H, N, O, P**。



# 1 Basics

15

- 樹狀結構的專用術語：
  - 非終端節點(nonterminal node)：終端節點以外的節點，這棵樹的非終端節點共有A, B, E, F, C, D, I, J。
    - ✦ 樹的節點總數 = 終端節點數量 + 非終端節點數量

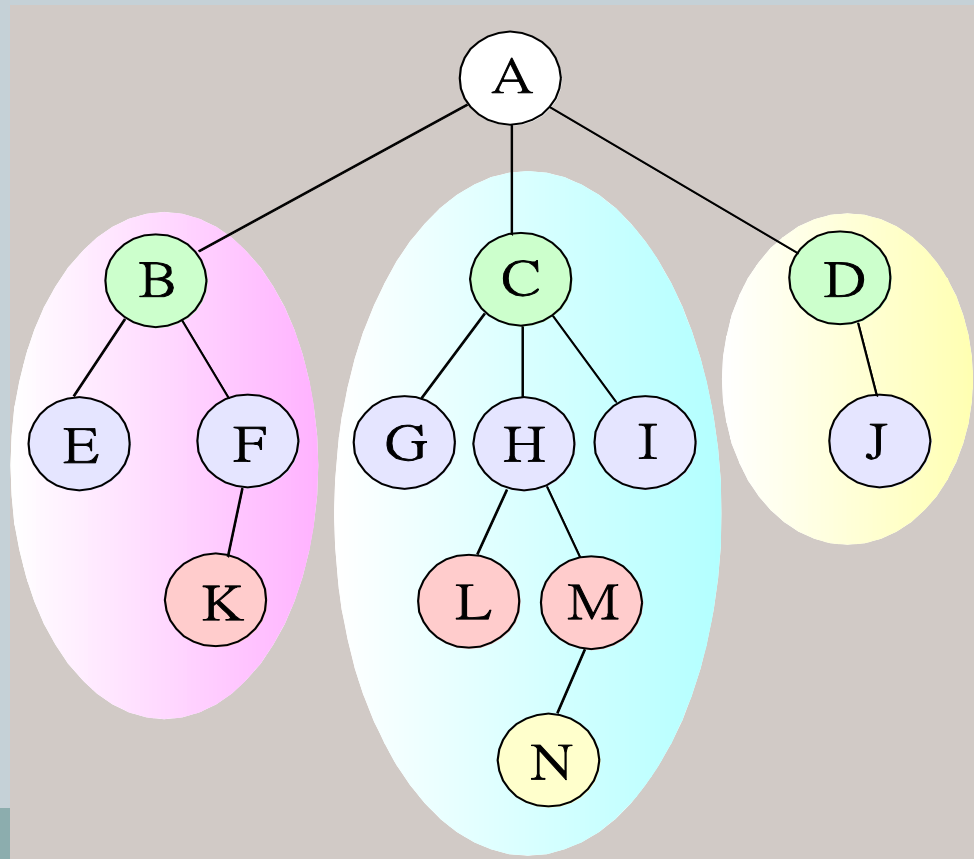


# 1 Basics

16

- 樹的表示方式：

(A(B(E, F(K)), C(G, H(L, M(N)), I), D(J)))

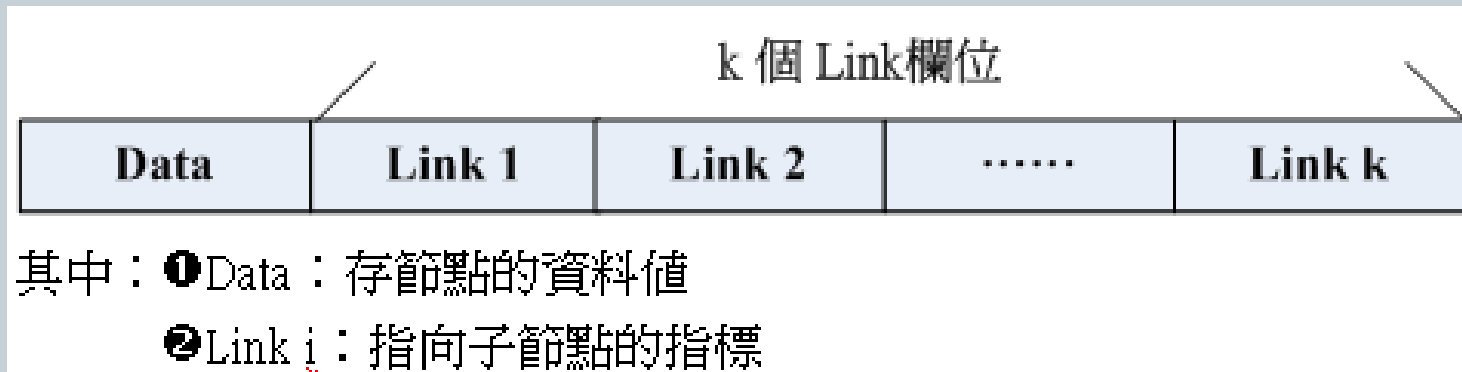




# 1 Basics

17

- 樹中節點的儲存方式：鏈結串列

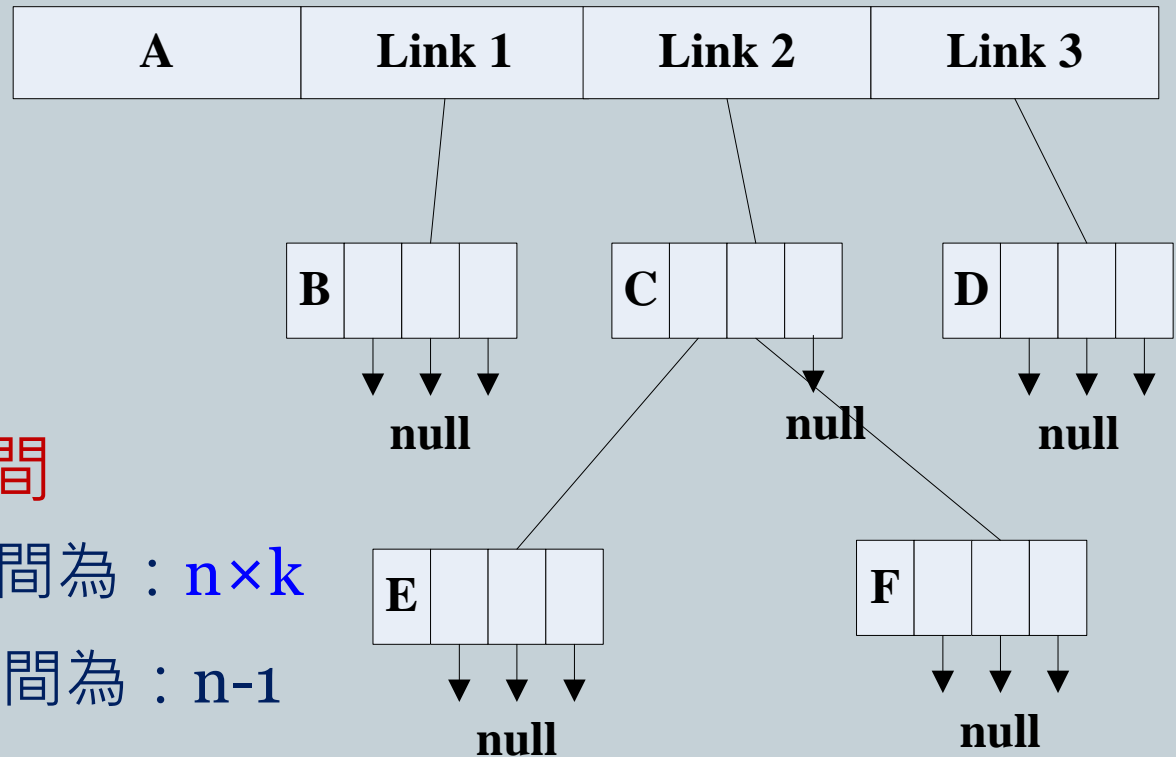
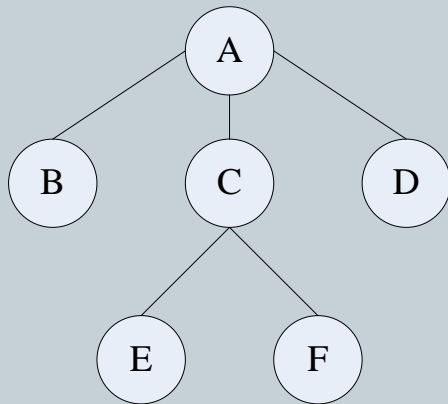


- Link欄位的數量：取決於樹的分支度。

# 1 Basics

18

- 【實例】有一棵三個階度的樹狀結構，其鏈結串列表示法如下：



○ 缺點：浪費空間

(1) 總共的Link空間為： $n \times k$

(2) 有用的Link空間為： $n-1$

∴ 浪費的Link數為： $n \times k - (n-1)$

## 2 二元樹

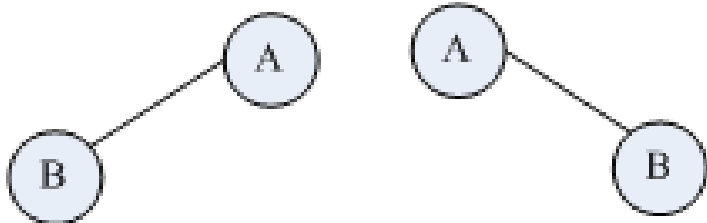
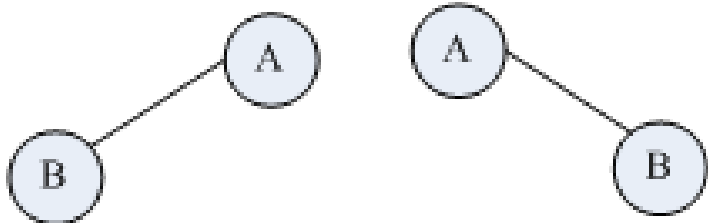
19

- 為了改進記空間浪費的缺點，我們將樹化成二元樹 (Binary tree) 的結構。
  - 二元樹所浪費的鏈結率最低。
- 二元樹之定義：
  - (1) 二元樹可以為空集合。
    - ✦ ※樹至少有一個樹根，不能為空集合。
  - (2) 具有根節點(Root)及左子樹和右子樹，有順序之分。
    - ✦ ※二元樹又稱為有序樹 (ordered tree)。
    - ✦ ※樹無順序之分。
  - (3) 二元樹節點的分支度為  $0 \leq d \leq 2$ 
    - ✦ ※二元樹即分支度為2的樹。
    - ✦ ※樹的分支度為  $0 < d$ 。

## 2 二元樹

20

- 二元樹與一般樹之比較：

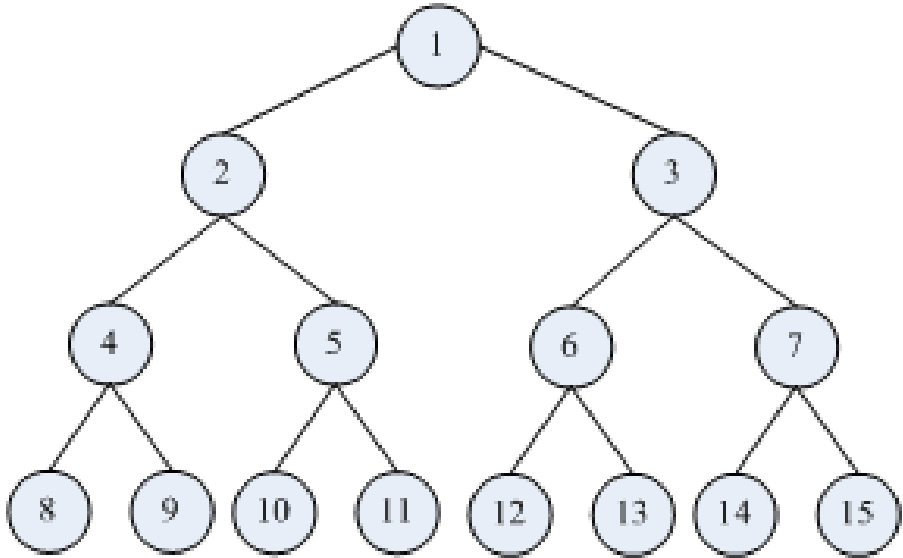
二元樹(binary tree)	樹(tree)
可以為空集合	樹不可為空集合
分支度為 $0 \leq d \leq 2$	分支度為 $d \geq 0$
左、右子樹有次序之分	左、右子樹無次序之分
 <p>不相同</p>	 <p>相同</p>

## 2 二元樹

21

- 二元樹的特性

- (1) 在第  $i$  階度(Level)上最多的節點個數為  $2^{i-1}$ ,  $i \geq 1$ 。

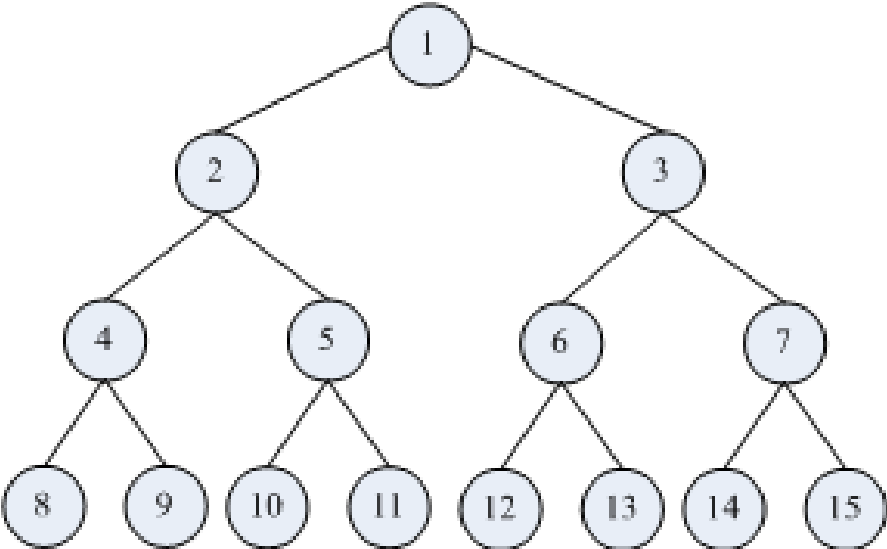
二元樹	Level	$2^{i-1}$
	1	$2^{1-1} = 2^0 = 1$
	2	$2^{2-1} = 2^1 = 2$
	3	$2^{3-1} = 2^2 = 4$
	4	$2^{4-1} = 2^3 = 8$

## 2 二元樹

22

- 二元樹的特性

- (2) 高度為 $h$ 的二元樹，其節點總數最多為  $2^h - 1$ ,  $h \geq 1$ 。

	Level	$2^h - 1$
	1	$2^1 - 1 = 1$
	2	$2^2 - 1 = 3$
	3	$2^3 - 1 = 7$
	4	$2^4 - 1 = 15$

## 2 二元樹

23

### ● 二元樹的特性

- (3) 若二元樹的總節點數(Node)為  $V$ ，總邊數(Edge)為  $E$ ，則  $V=E+1$ 。
- (4) 一棵二元樹，若  $n_0$  為樹葉節點的數量， $n_2$  為分支度為 2 的節點的數量則  $n_0 = n_2 + 1$

#### ✧ 證明範例

【證明】 假設  $n$ ：節點總數

$n_1$ ：Degree 為 1 的 Node 數

$E$ ：總邊數

$$n = n_0 + n_1 + n_2$$

$$n = E + 1$$

$$E = n_1 + 2n_2$$

$$n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$$

$$\Rightarrow n_0 = n_2 + 1$$

## 2.1 完滿二元樹V.S. 完整二元樹

24

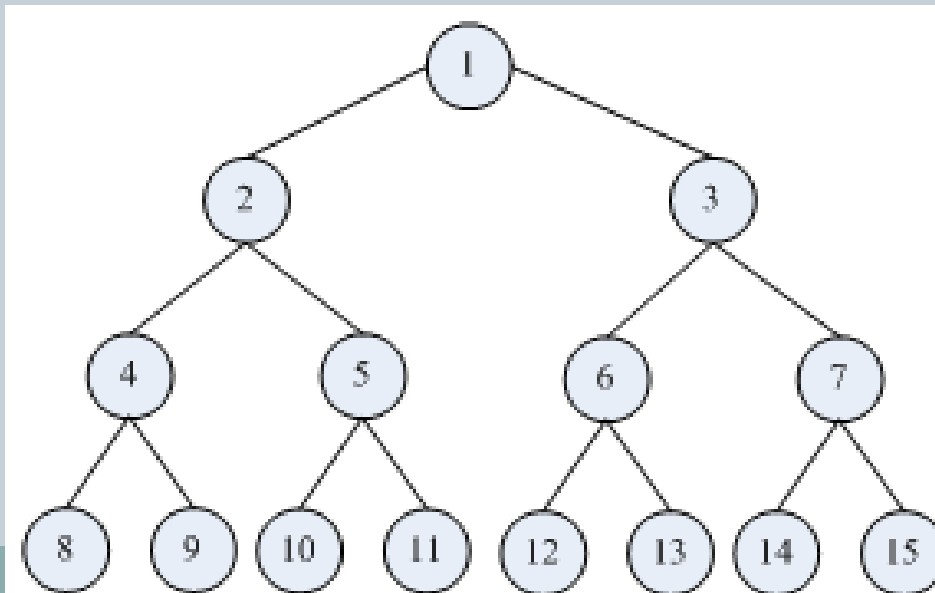
- 完滿二元樹(Full Binary Tree)

- 二元樹的特性-(2)

- ✦ 高度為 $h$ 的二元樹，其節點總數最多為  $2^h - 1$ ， $h \geq 1$ 。

- 完滿二元樹即是高度為  $h$  且具  $2^h - 1$  個節點， $h \geq 1$  的二元樹。

- Ex: 下圖為高度為4( $h=4$ )的完滿二元樹，共有 $2^4 - 1$ 個節點





## 2.1 完滿二元樹V.S. 完整二元樹

25

- 完整二元樹(Complete Binary Tree)

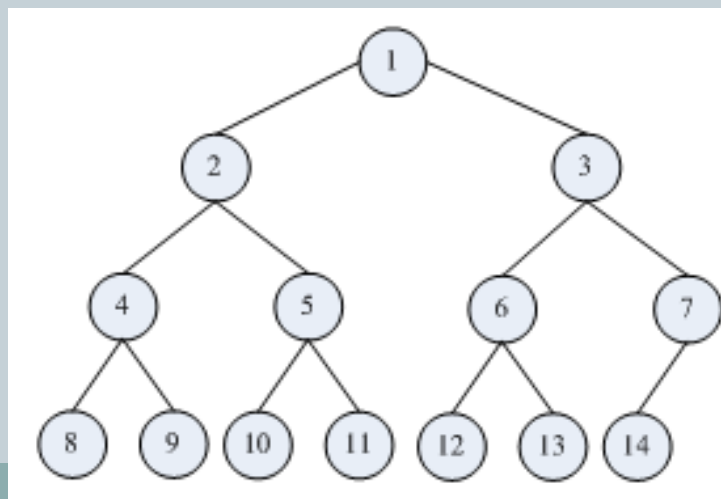
- 完滿二元樹

- ✦ 高度為  $h$  且具  $2^h - 1$  個節點,  $h \geq 1$  的二元樹。

- 完整二元樹

- ✦ 即是高度為  $h$  且具  $2^{h-1} - 1 \sim 2^h - 1$  個節點,  $h \geq 1$  的二元樹。

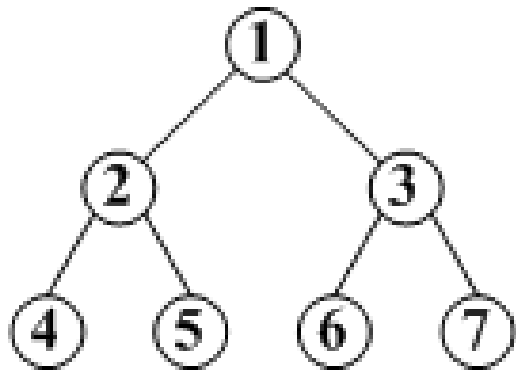
- Ex: 下圖為高度為4( $h=4$ )的完整二元樹(節點按順序排列)。



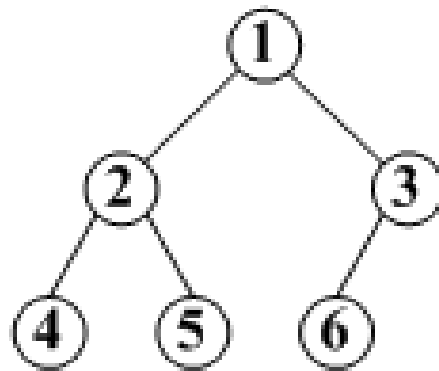
## 2.1 完滿二元樹V.S. 完整二元樹

26

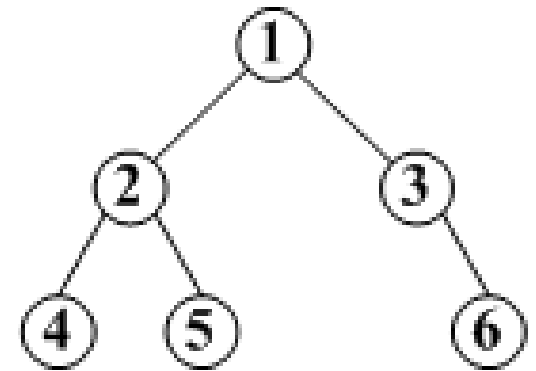
- 比較：



(a) 完滿二元樹



(b) 完整二元樹



(c) 非完整二元樹

## 2.2 二元樹的表示方式

27

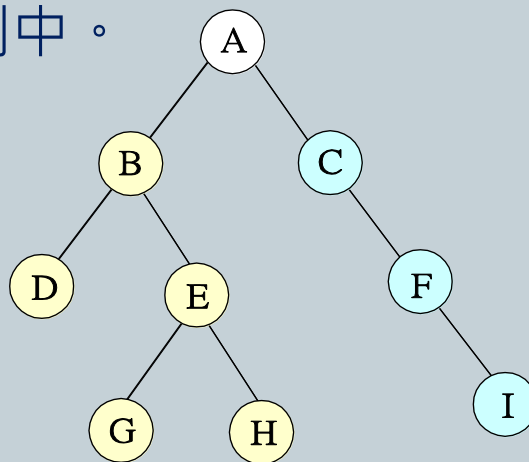
- 共有兩種方式：
- (1)以陣列(Array)表示
  - 適合完滿二元樹(Full Binary Tree)
  - 不適合傾斜樹(Skew Tree)
  - 容易追蹤(Traversal)
- (2)以鏈結串列(Linked List)表示
  - 適合處理傾斜樹。
  - 但Link空間約浪費一半。

## 2.2 二元樹的表示方式

28

- 以陣列(Array)表示：

- 首先將二元樹想像成一個完滿二元樹，並使用一維陣列建立二元樹的表示方法及索引值的配置。
  - ✦ 假設二元樹之高度為 $h$ ，則準備一維陣列 $A[1...2^h-1]$ ，並依照節點編號依序存入陣列中。

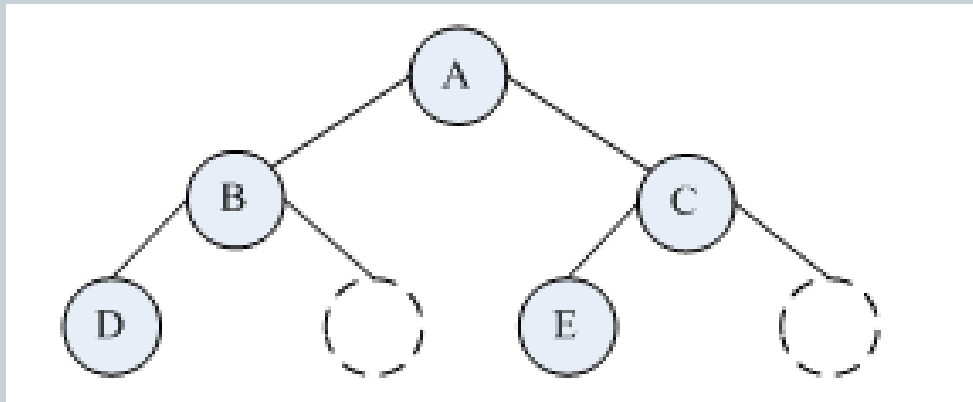


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
A	B	C	D	E	--	F	--	--	G	H	--	--	--	I

## 2.2 二元樹的表示方式

29

- 【隨堂練習】請將下列二元樹以陣列方式表示。
- PS. 根節點索引從1開始。



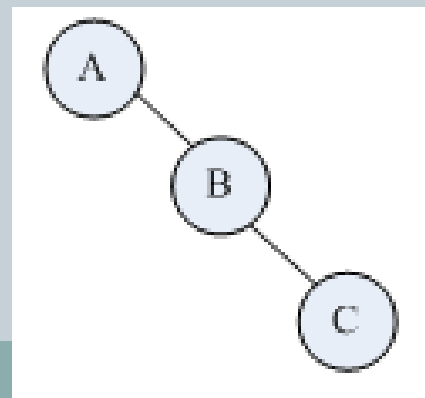
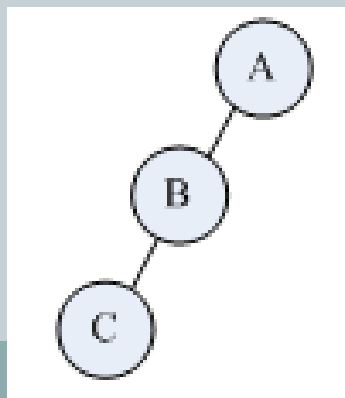
- 【解答】

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	—	<b>E</b>	—

## 2.2 二元樹的表示方式

30

- 以陣列(Array)表示：
- 優點：
  - 容易取得左、右子點及父節點。
  - 二元樹呈現左右平衡或完整時，不會浪費記憶體。
- 缺點：
  - 節點之插入與刪除較困難。
  - 稀疏或傾斜(Skewed)二元樹，易浪費記憶體。



## 2.2 二元樹的表示方式

31

- 陣列(Array)--左、右子點及父節點之關係：
- 假設完整二元樹的節點個數為 $n$ 且依照 $0 \sim n - 1$ 的順序編號，則其任意節點 $i$  ( $0 \leq i \leq n - 1$ ) 具有下列特質：
  - 節點 $i$ 的父節點為 $\text{floor}[(i - 1)/2]$ 。
    - ✦ 當 $i = 0$ ，表示節點 $i$ 為樹根。
  - 節點 $i$ 的左子節點為 $2i + 1$ 。
    - ✦ 若 $2i + 1 \geq n$ ，表示節點 $i$ 為樹葉，故沒有左子節點，
  - 節點 $i$ 的右子節點為 $2i + 2$ 。
    - ✦ 若 $2i + 2 \geq n$ ，表示節點 $i$ 為樹葉，故沒有右子節點，

## 2.2 二元樹的表示方式

32

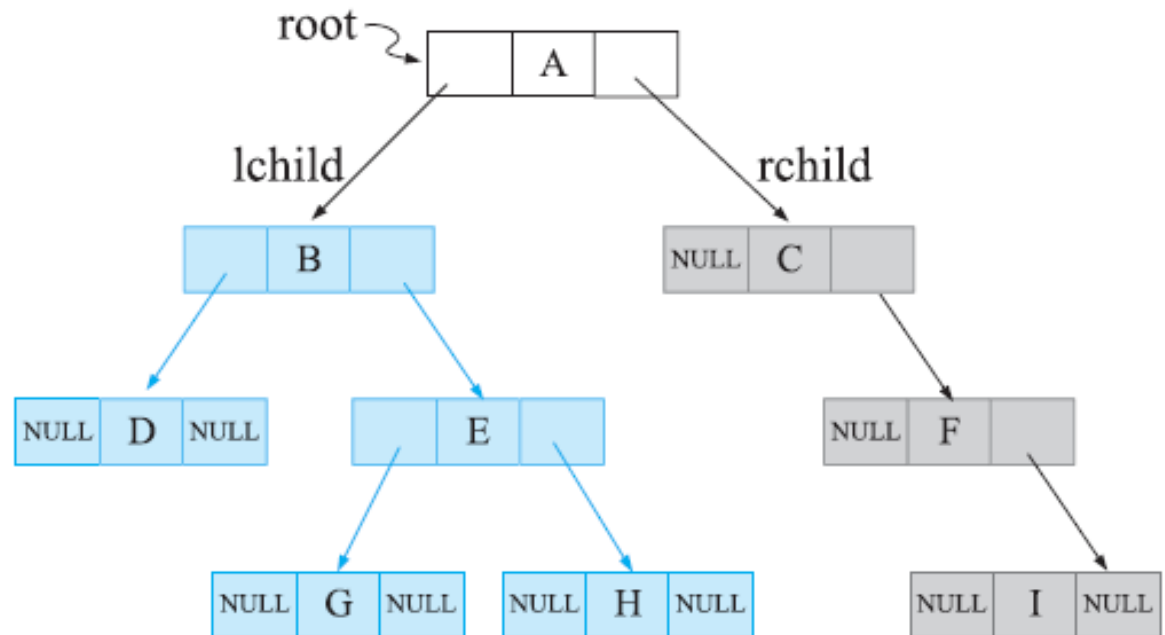
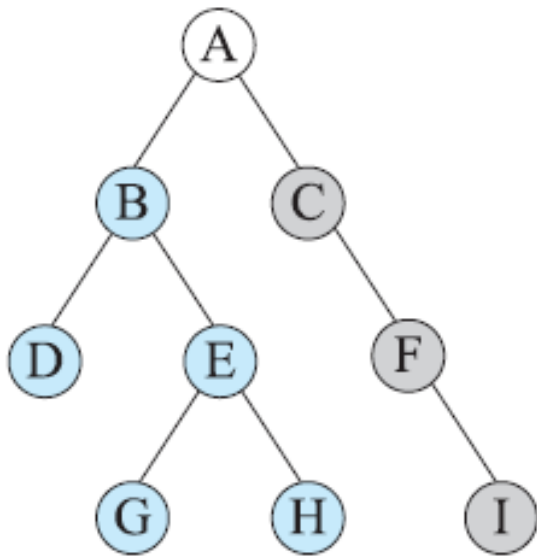
- 【辨析】
- 假設完整二元樹的節點個數為 $n$ 且依照 $1 \sim n$ 的順序編號，則其任意節點 $i$  ( $1 \leq i \leq n$ ) 具有下列特質：
  - 此樹之根節點 $i$ 為何？
    - ✧  $i = 1$ 。
  - 節點 $i$ 的父節點為何？
    - ✧  $\text{floor}[i/2]$ 。
  - 節點 $i$ 的左子節點為何？
    - ✧  $2i$ 。
  - 節點 $i$ 的右子節點為何？
    - ✧  $2i + 1$ 。
  - 若 $2i \geq n$ ，表示節點 $i$ 為何？
    - ✧ 樹葉。



## 2.2 二元樹的表示方式

33

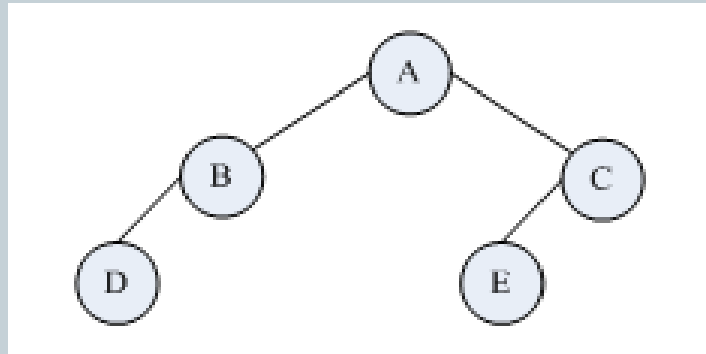
- 以鍵結串列(Linked List)表示
- 使用動態記憶體配置來建立二元樹，兩個指向左和右子樹的指標。



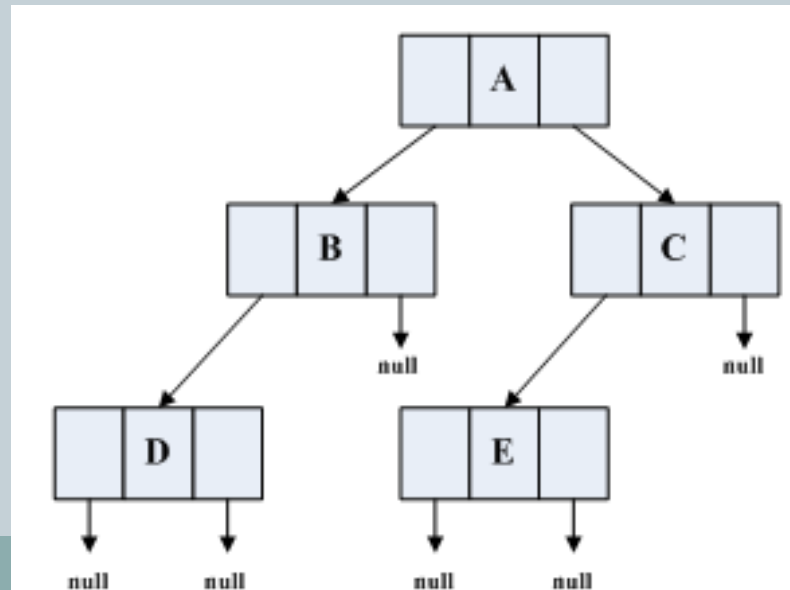
## 2.2 二元樹的表示方式

34

- 【隨堂練習】試以鏈結串列表示下列二元樹。



○ 【解答】



## 2.2 二元樹的表示方式

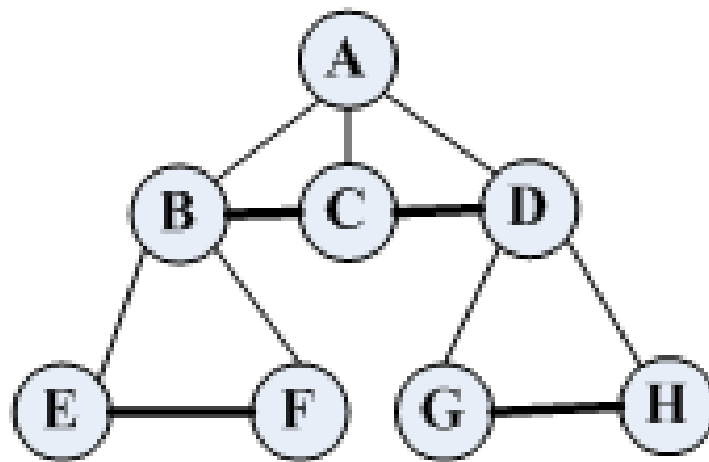
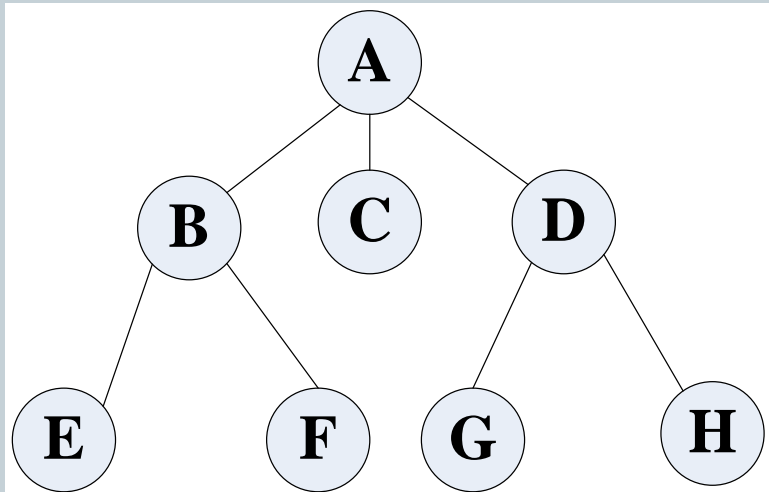
35

- 以鍵結串列(Linked List)表示：
- 優點：
  - 節點之插入與刪除較容易。
  - 稀疏或傾斜(Skewed)二元樹，較節省記憶體
- 缺點：
  - 不易找到父節點。
  - 需額外準備左、右指標的記憶體空間。

## 2.3 樹換為二元樹

36

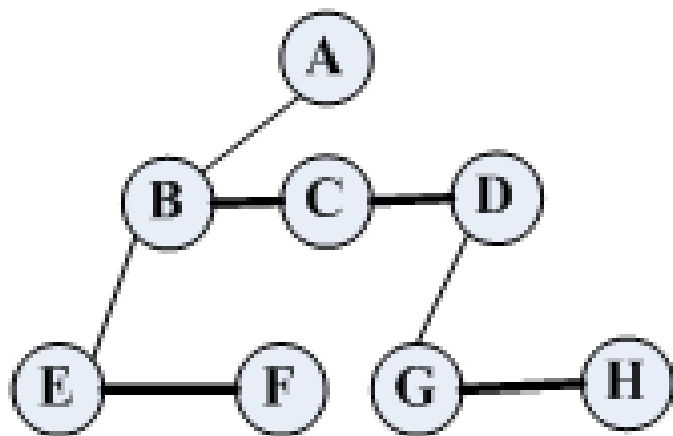
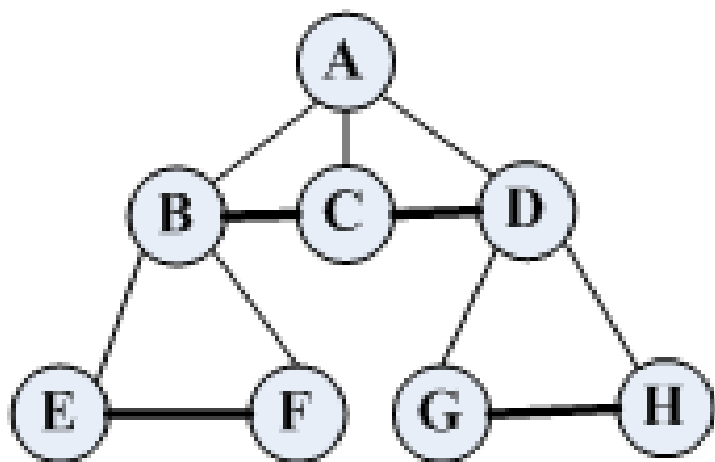
- 一般樹轉換成二元樹，其步驟如下：
  - 步驟1：將樹的各階層兄弟節點利用平行線連接起來。
  - 步驟2：刪掉所有右邊父子節點間的連結，只留最左邊的父子節點。
  - 步驟3：右邊的兄弟節點順時鐘轉45度。



## 2.3 樹換為二元樹

37

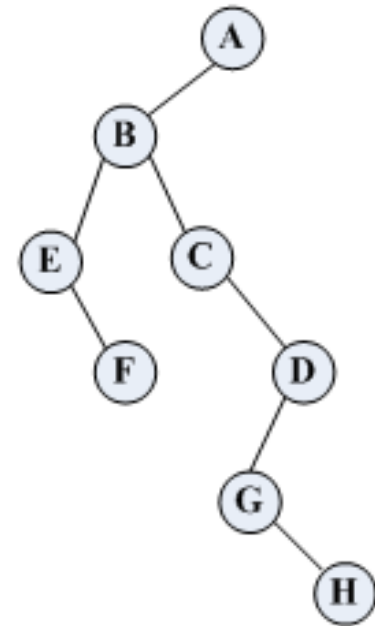
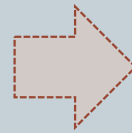
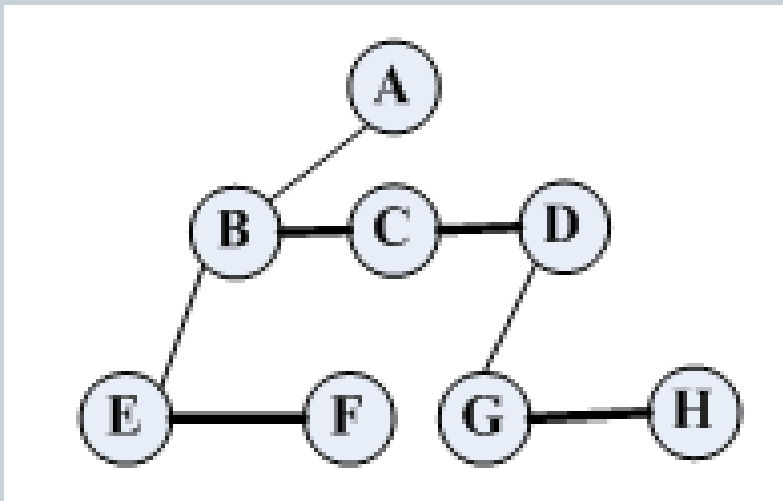
- 一般樹轉換成二元樹，其步驟如下：
  - 步驟1：將樹的各階層兄弟節點利用平行線連接起來。
  - 步驟2：刪掉所有右邊父子節點間的連結，只留最左邊的父子節點。
  - 步驟3：右邊的兄弟節點順時鐘轉45度。



## 2.3 樹換為二元樹

38

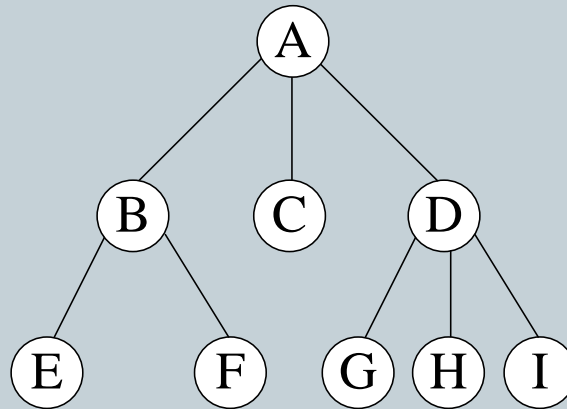
- 一般樹轉換成二元樹，其步驟如下：
  - 步驟1：將樹的各階層兄弟節點利用平行線連接起來。
  - 步驟2：刪掉所有右邊父子節點間的連結，只留最左邊的父子節點。
  - 步驟3：右邊的兄弟節點順時鐘轉45度。



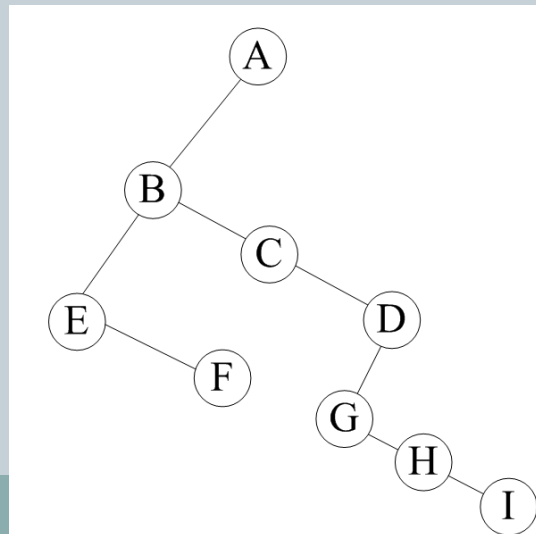
## 2.3 樹換為二元樹

39

- 【隨堂練習】試將下列樹轉換為二元樹。



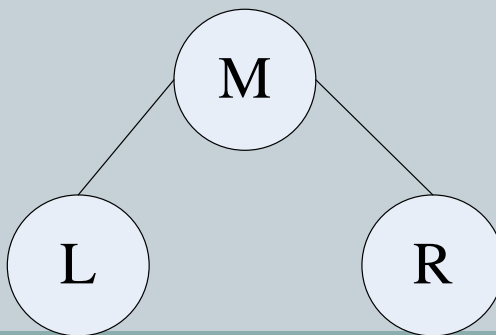
- 【解答】



### 3 二元樹的走訪

40

- 走訪(Traversal)：
  - 即走訪樹中的每一個節點，且每個節點恰好被走訪一次。
- 走訪的種類：
  - 中序 In-order ( 左、中、右 )
  - 前序 Pre-order ( 中、左、右 )
  - 後序 Post-order ( 左、右、中 )
  - ※限制條件：左子樹一定在右子樹之前走訪。





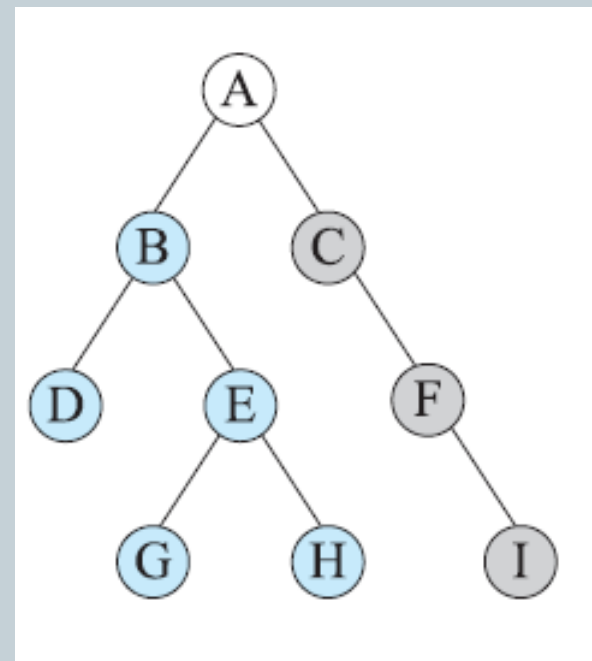
### 3 二元樹的走訪

41

- 中序走訪：

- 順序為：左子樹→樹根→右子樹。
- 沿樹的左子樹一直往下，直到無法前進，再後退回樹根(即父節點)，再往右子樹一直往下。

```
inorder(tree_pointer root)
{
    if (root){
        inorder(root->lchild);
        printf("%d ", root->data);
        inorder(root->rchild);
    }
}
```



- 右圖中序走訪的結果為DBGEHACFI。

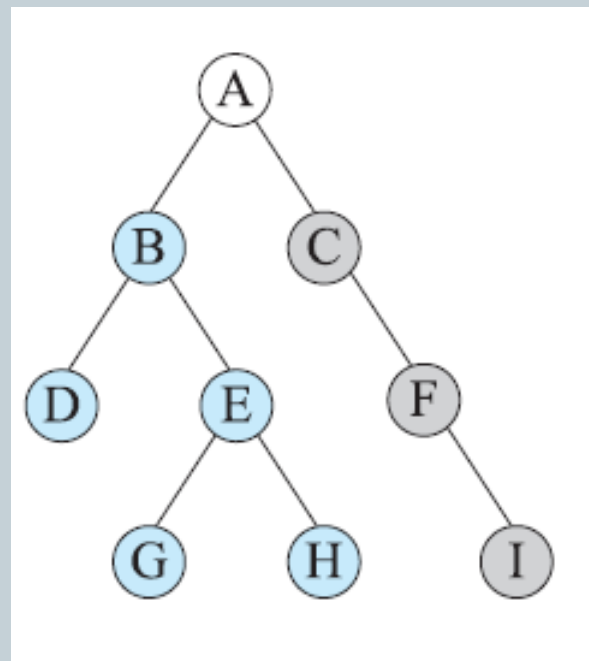
### 3 二元樹的走訪

42

- 前序走訪：

- 順序為：樹根→左子樹→右子樹。
- 從根節點開始處理，根節點處理完成之後，再往左子樹走，直到無法前進，再處理右子樹。

```
preorder(tree_pointer root)
{
    if (root){
        printf("%d ", root->data);
        preorder(root->lchild);
        preorder(root->rchild);
    }
}
```



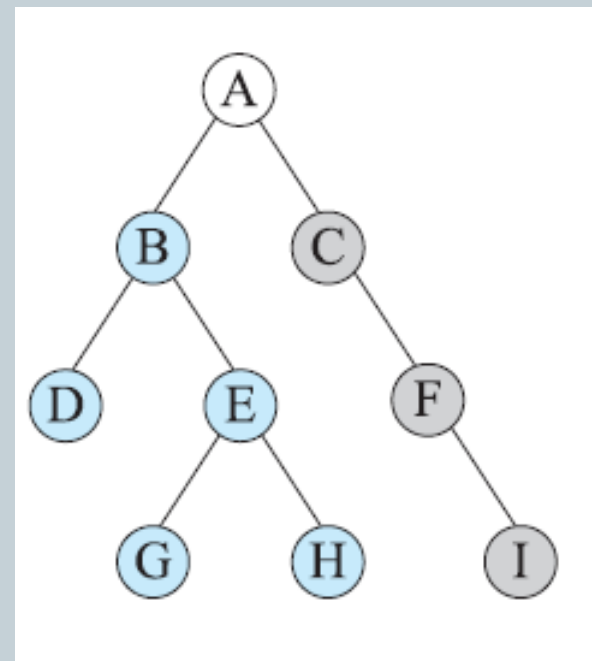
- 右圖中序走訪的結果為 **ABDEGHCFI**。

### 3 二元樹的走訪

43

- 後序走訪：
  - 順序為：左子樹→右子樹→樹根。
  - 沿樹的左子樹一直往下完成之後，再往右子樹一直往下，直到無法前進，再後退回樹根。

```
postorder(tree_pointer root)
{
    if (root){
        postorder(root->lchild);
        postorder(root->rchild);
        printf("%d", root->data);
    }
}
```

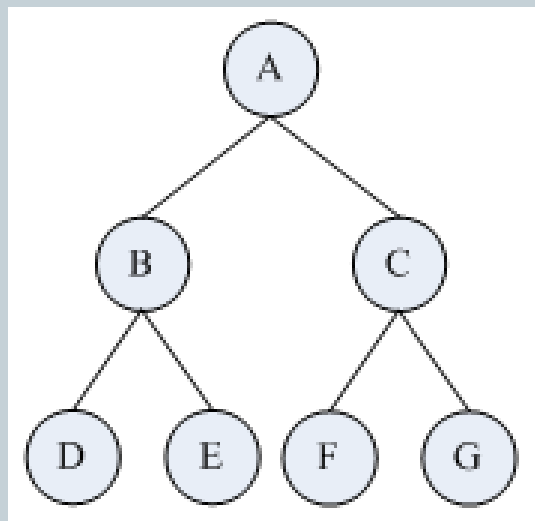


- 右圖中序走訪的結果為 **DGHEBIFCA**。

### 3 二元樹的走訪

44

- 【隨堂練習】試求下列二元樹的前序、中序及後序走訪。



【解答】❶ Preorder(前序追蹤)：ABDECFCG

❷ Inorder(中序追蹤)：DBEAFCCG

❸ Postorder(後序追蹤)：DEBFGCA

### 3 二元樹的走訪

45

- 決定唯一的二元樹：
  - 如果有中序與前序的走訪結果或者中序與後序的走訪結果，可由這些結果求得唯一的二元樹。
  - 不過如果只具備前序與後序的追蹤結果就無法決定唯一的二元樹。

### 3 二元樹的走訪

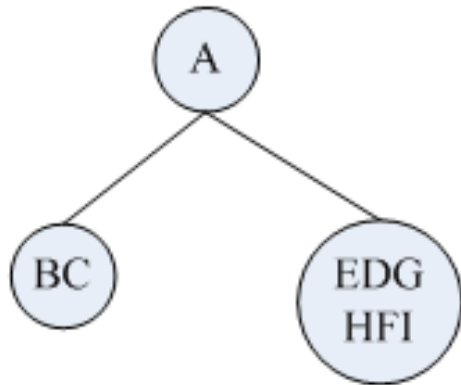
46

- 例1：二元樹的前序為ABCDEF GHI，  
中序為BCAEDG HFI，繪出此二元樹

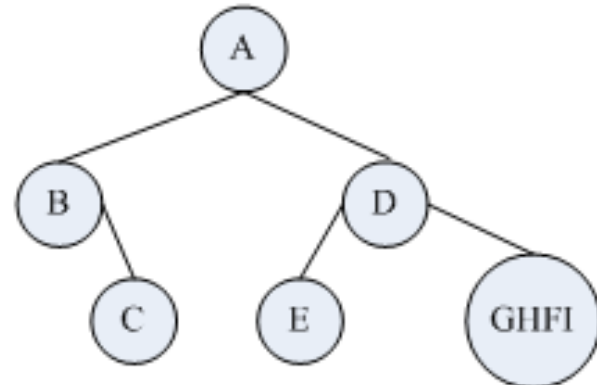
【解答】

解

步驟一：決定樹根，再左右分家



步驟二：決定第二層的樹根 B,D，再左右分家

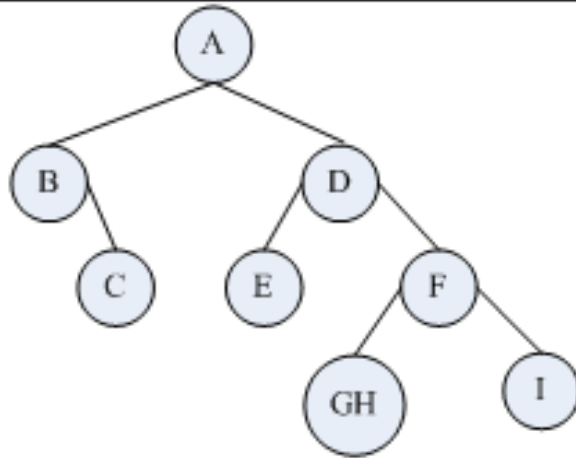


### 3 二元樹的走訪

47

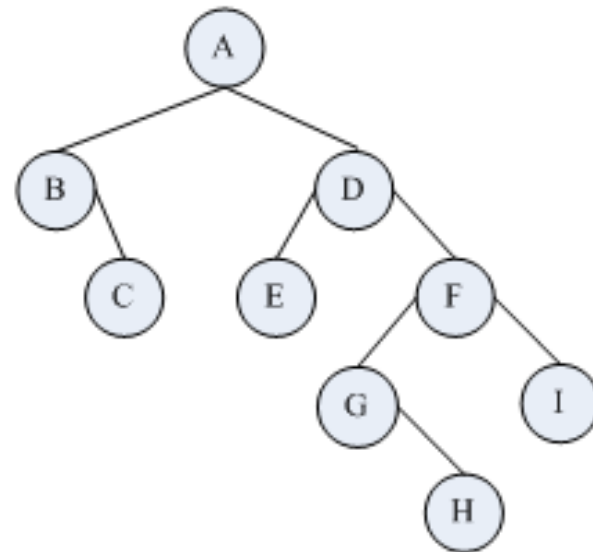
步驟三：決定第三層的樹根 F，再左

右分家



步驟四：決定第四層的樹根 G，再左

右分家



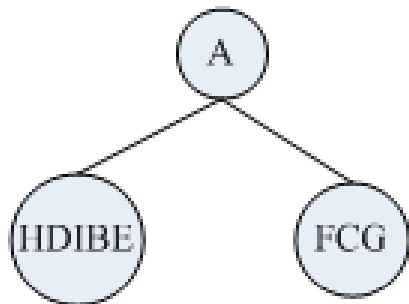
### 3 二元樹的走訪

48

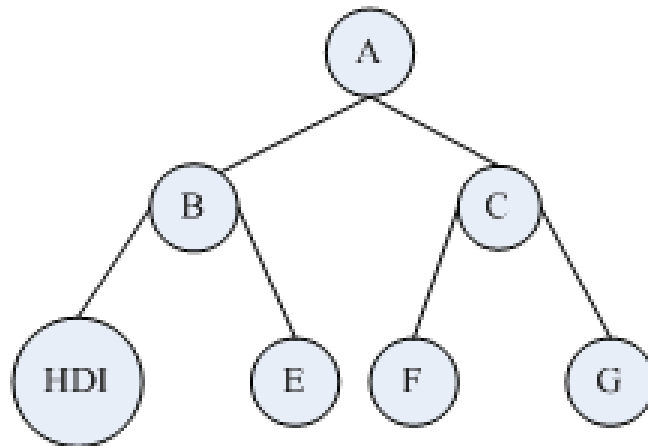
- 例2：二元樹的後序為HIDEBFGCA，  
中序為HDI~~B~~EAF~~C~~G，繪出此二元樹

【解答】

步驟一：決定樹根，再左右分家



步驟二：決定第二層的樹根 B,C，再左右分家

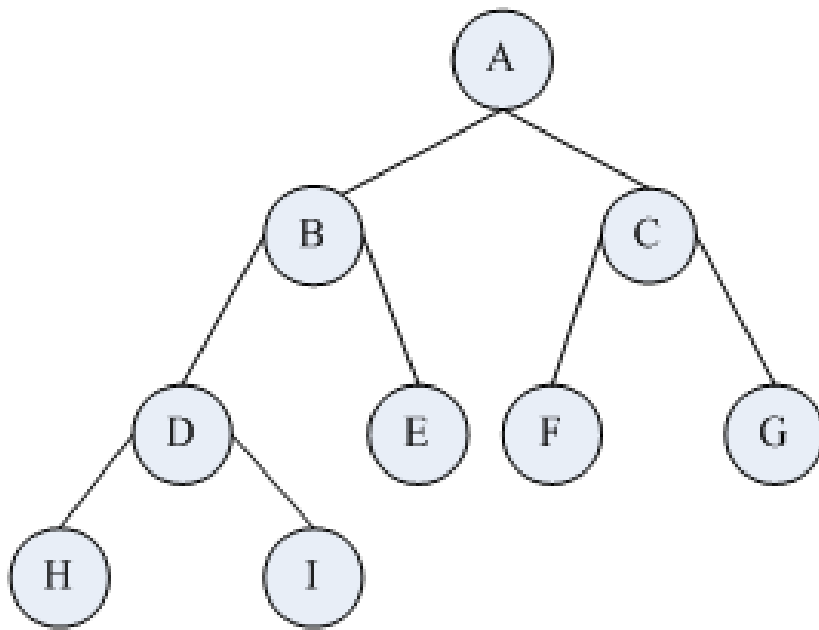




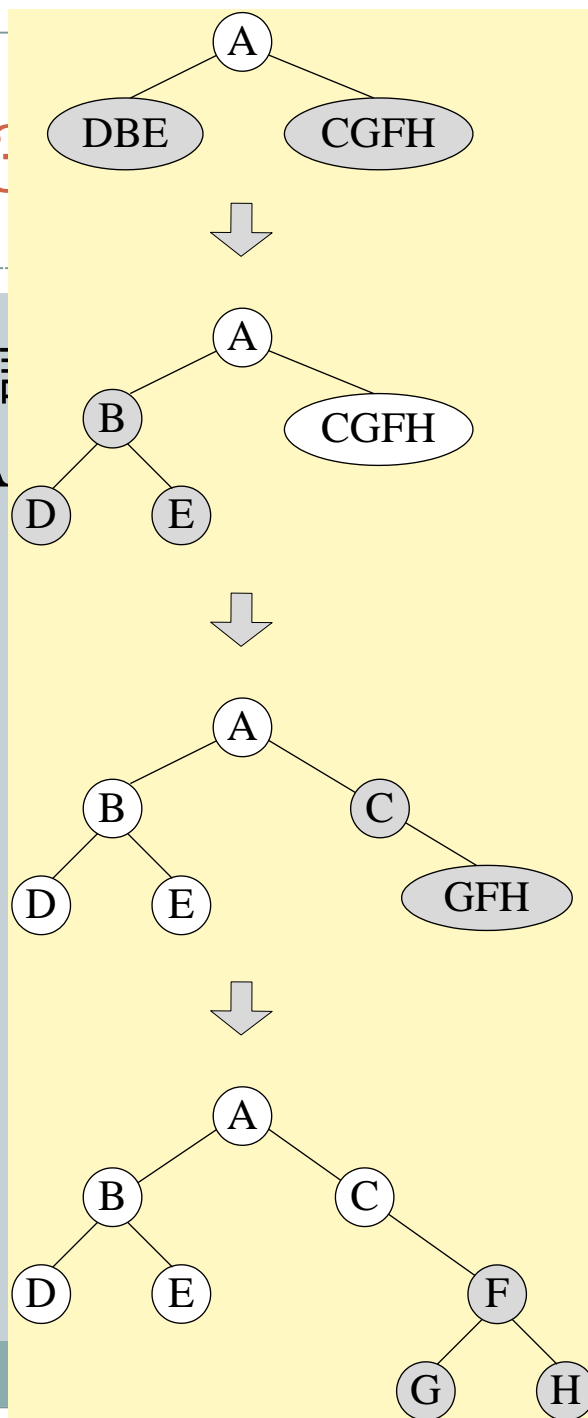
### 3 二元樹的走訪

49

步驟三：決定第三層的樹根 D，再左右分家



- 【隨堂練習】假設前序走訪分別為 DBEACGFH、A
- 【解答】

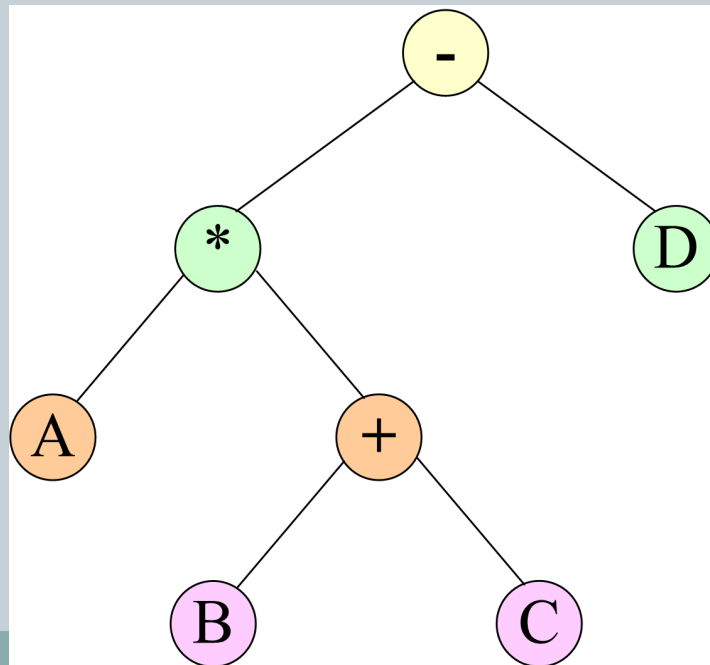


前序走訪分別為  
此二元樹。

## 4 運算式樹

51

- 運算式樹(Expression Tree) 是一種特殊二元樹，滿足以下條件：
  - 1. 終端節點為運算元；非終端節點為運算子。
  - 2. 子樹為子運算式且其樹根為運算子。



# 5 引線二元樹

52

- 二元樹若採用鏈結串列結構：
  - $n$ 個Nodes總共Link空間為 $2n$
  - 有用的Links空間為 $n-1 \rightarrow$  (二元樹的特性(3) ,  $V=E+1$ )
  - 無用的Links空間為 $2n-(n-1)=n+1 \rightarrow$  造成空間浪費

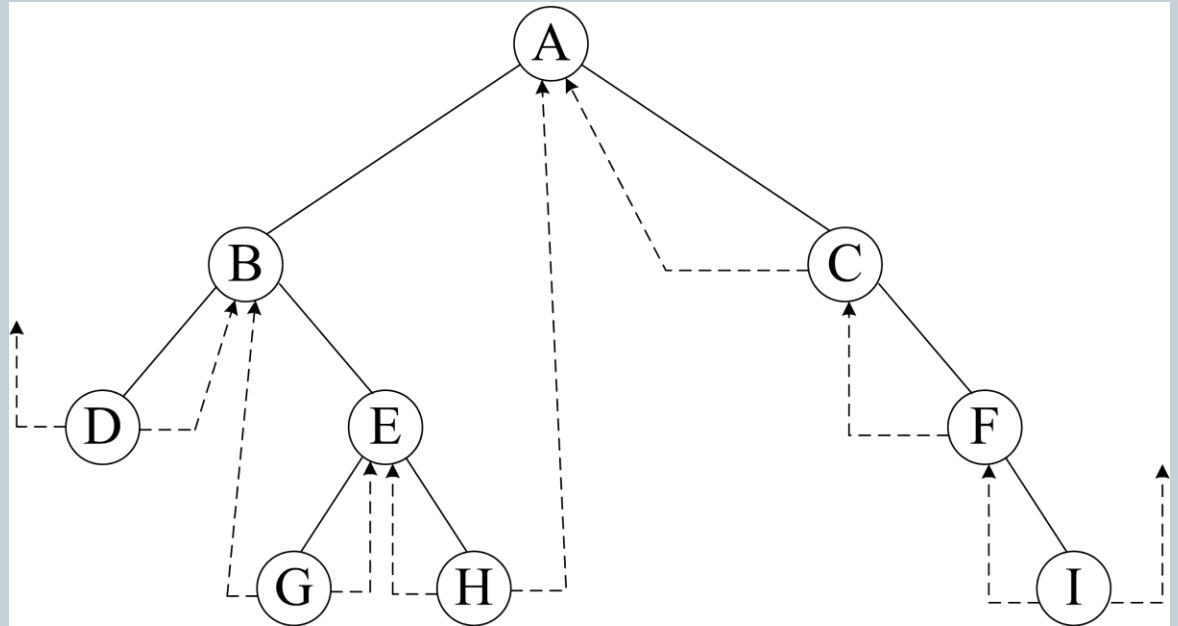
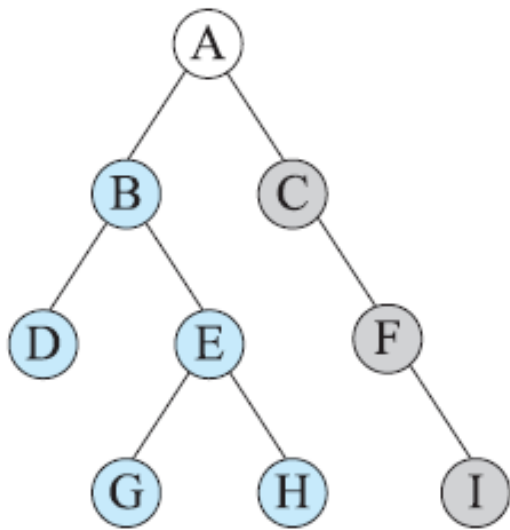
LChild	Data	RChild
--------	------	--------

- 把這些空的鏈結加以利用，指向樹的其他節點，而這些鏈結就稱為「引線」(thread)，而這種二元樹就稱為引線二元樹(Threaded Binary Tree)。

## 5 引線二元樹

53

- 引線二元樹的**左引線** (left thread) 指向節點本身的**中序前行者** (inorder predecessor)，**右引線** (left thread) 指向節點本身的**中序後繼者** (inorder successor)。



# 5 引線二元樹

54

- 引線二元樹的節點結構：

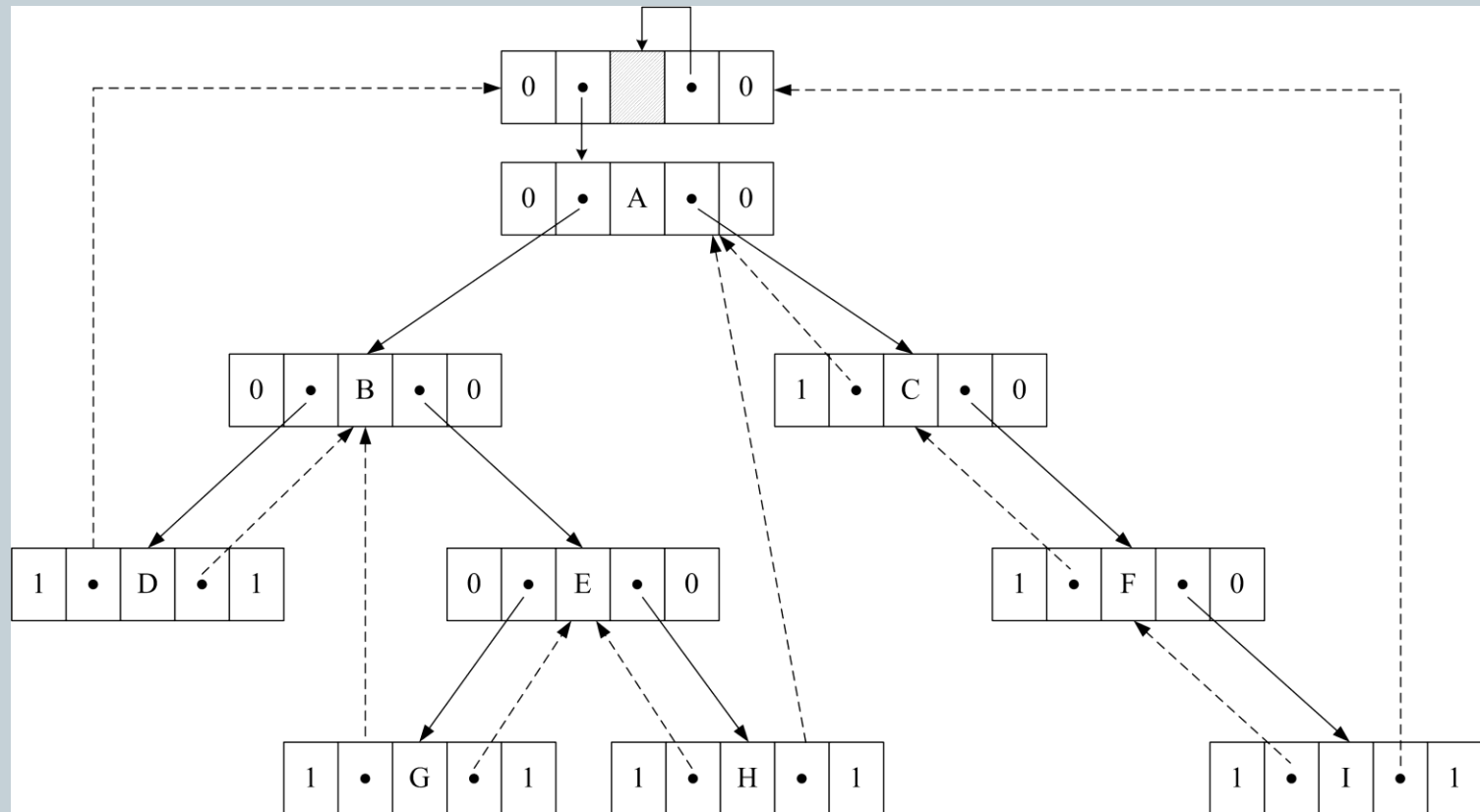
Left Thread	Lchild	Data	Rchild	Right Thread
-------------	--------	------	--------	--------------

- 額外加上兩個欄位，區分引線和實際指標的不同。
  - 1. Left Thread：
    - ✦ 當為True或1時，代表Lchild為引線，指向中序追蹤的前一個節點。
    - ✦ 當為False或0時，則代表Lchild為指標，指向左子樹的指標。
  - 2. Right Thread：
    - ✦ 當為True或1時，代表Rchild為引線，指向中序追蹤的後一個節點。
    - ✦ 當為False或0時，則代表Rchild為指標，指向右子樹的指標。

# 5 引線二元樹

55

- 加入首節點的引線二元樹

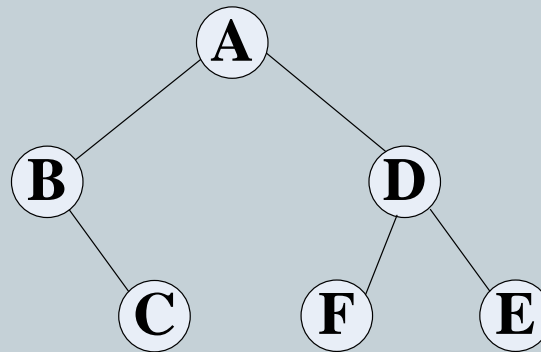


○ 【觀察】 樹葉的左、右鏈結一定為引線。

## 5 引線二元樹

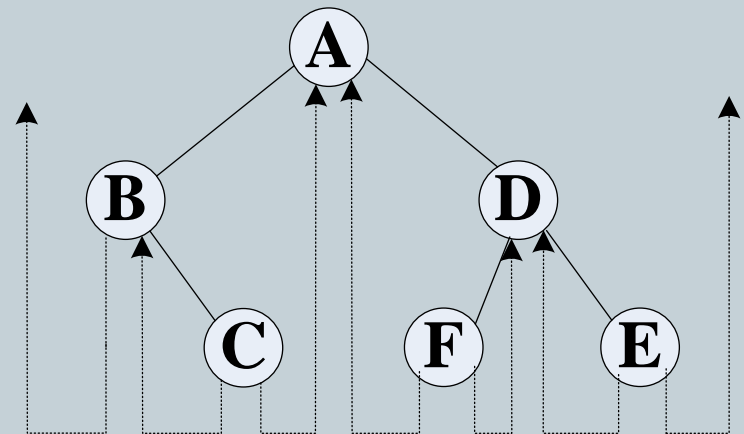
56

- 【隨堂練習】在下圖中給予一個二元樹，請繪出其引線二元樹。



- 【解答】

- 1. 先寫出中序式: B C A F D E
- 2. 引線二元樹表示如右：

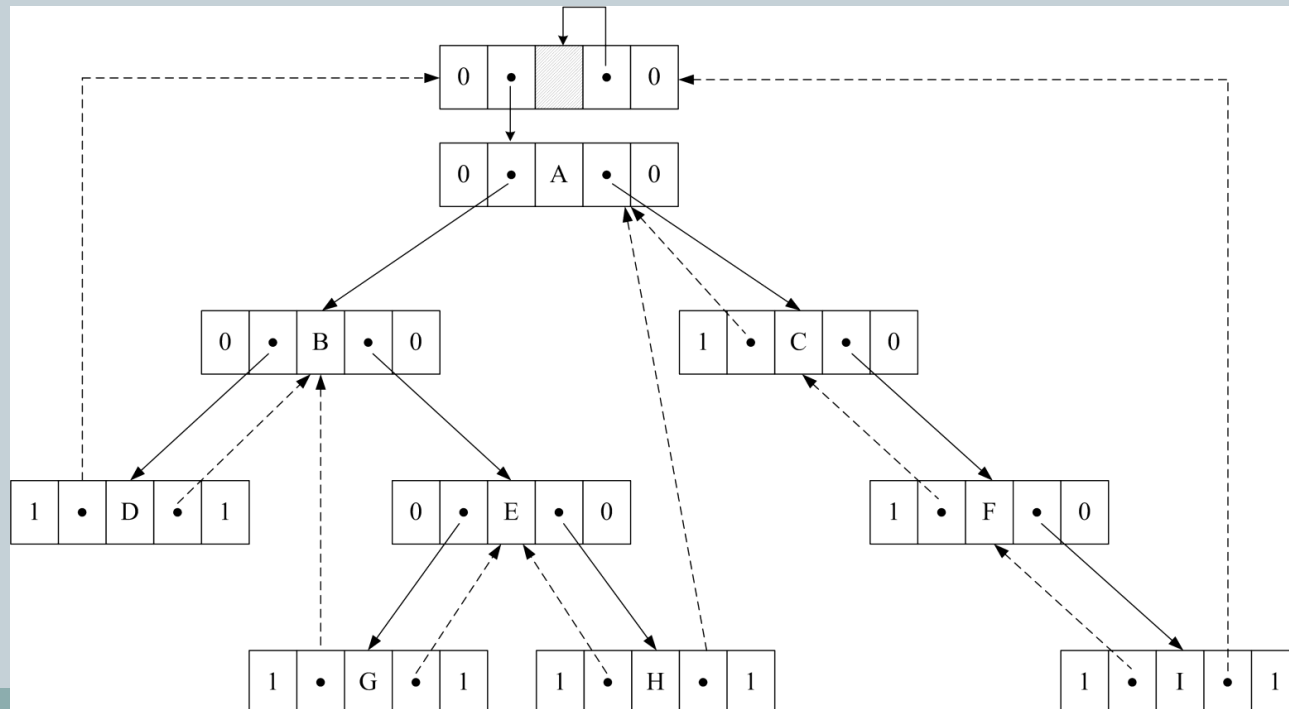




# 5 引線二元樹

57

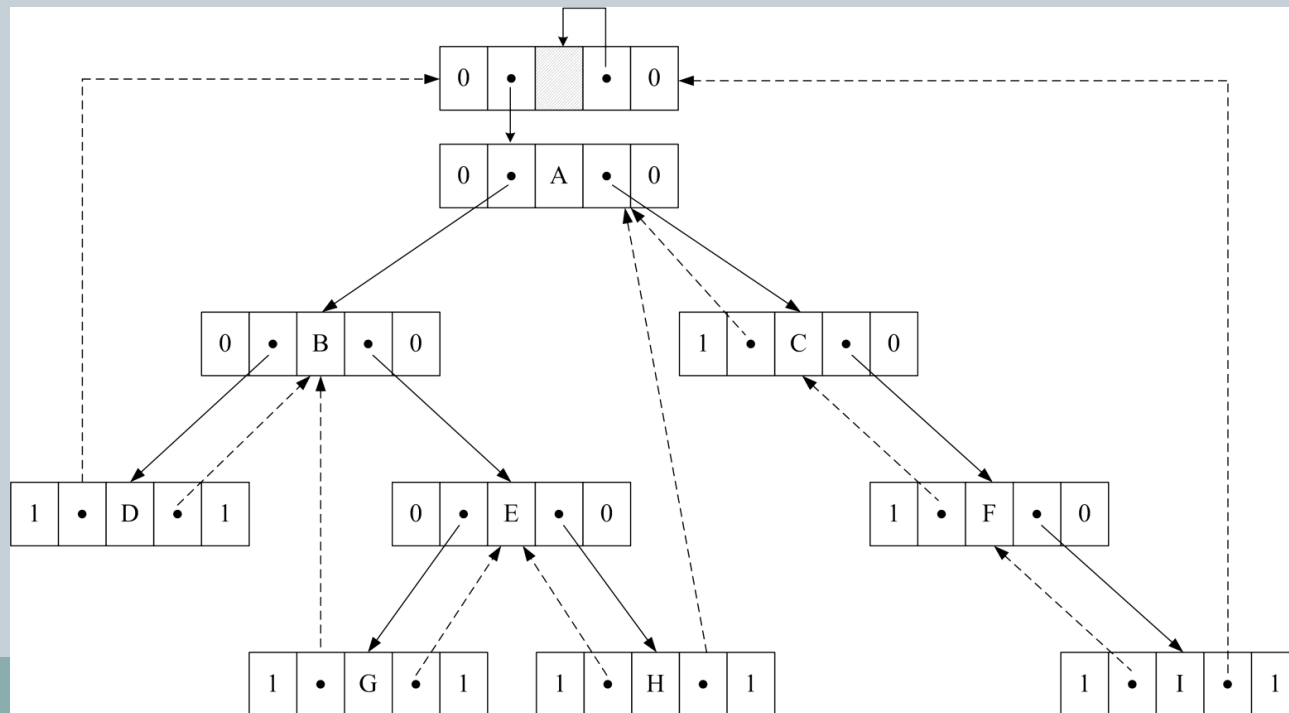
- 如何進行引線二元樹中序走訪，**找出中序後繼者**。
  - 若節點右鏈結為**引線**，則所指到的節點即為中序後繼者。
    - ✦ Ex. H的右引線指向A，故H的中序後繼者為A。



# 5 引線二元樹

58

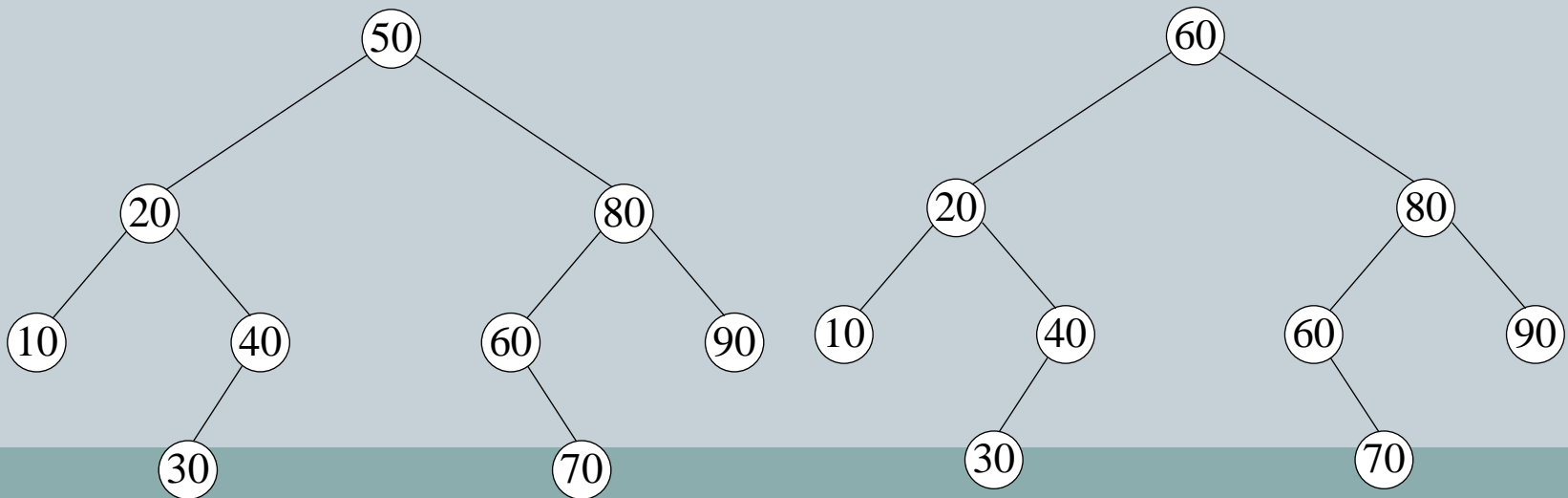
- 如何進行引線二元樹中序走訪，**找出中序後繼者**。
  - 若節點右鏈結為**指標**，則移往右子樹，沿著左鏈結找到左鏈結為引線者，該節點即為中序後繼者。
    - ✦ Ex. B的右鏈結為指標，移往右子樹，找到G左鏈結為引線。



# 6 二元搜尋樹

59

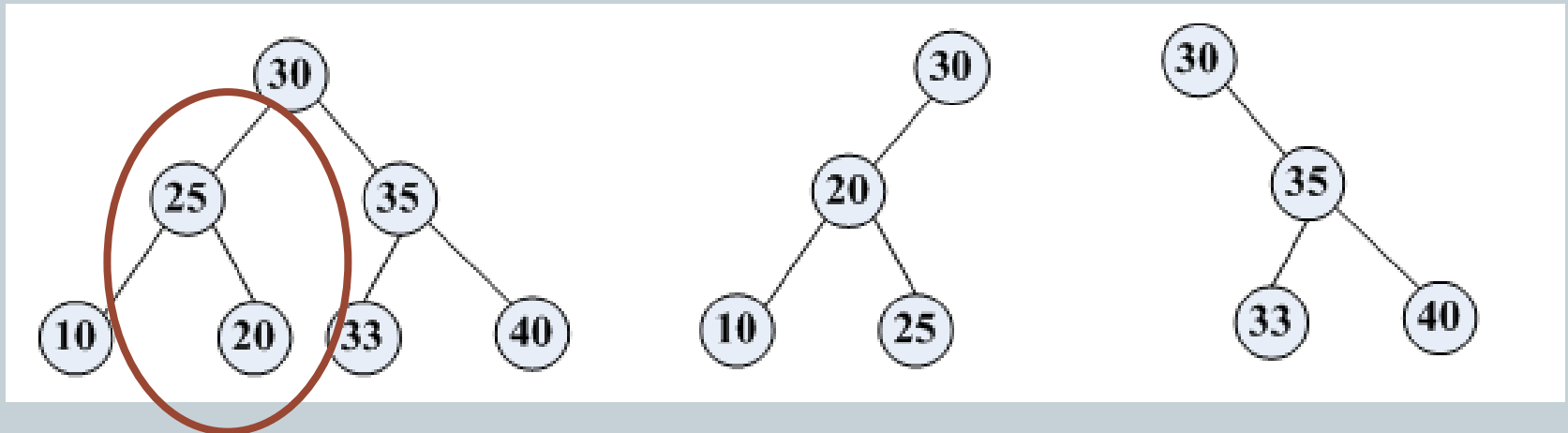
- 二元搜尋樹是一種特殊的二元樹，它可以為空，若不為空，則必須要滿足以下條件：
  - 1. 節點的鍵值唯一 (不能重複)
  - 2. 左子樹的鍵值均須要小於樹根的鍵值。
  - 3. 右子樹的鍵值均須要大於樹根的鍵值。
  - 4. 左子樹與右子樹亦為二元搜尋樹。



## 6 二元搜尋樹

60

- 【隨堂練習】請判斷以下二元樹是否為二元搜尋樹。



- 【解答】

No

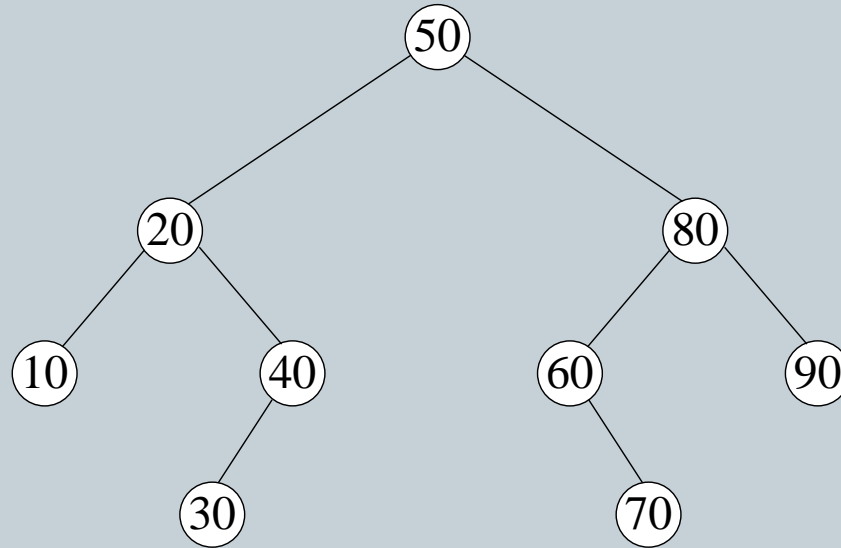
Yes

Yes

## 6 二元搜尋樹

61

- 二元搜尋樹的中序走訪剛好會使得資料由小到大排序 (10, 20, 30, 40, 50, 60, 70, 80, 90)，因此只要將資料整理成二元搜尋樹，就能快速找到資料。



- 高度為 $h$ 的二元搜尋樹，最多搜尋 $h$ 次，亦即 $O(h)$ 。

# 6 二元搜尋樹

62

- 建立二元搜尋樹：
  - 依序插入新節點。
    - ✦ ※第1個節點即為根節點。
  - 若新節點小於樹根，就移往樹根的左子樹；
  - 若新節點大於樹根，就移往樹根的右子樹，直到抵達子樹的尾端，再將新節點加入子樹的尾端。
- 相同資料排列順序不同，會建構出不同之二元搜尋樹。
- 當資料本身已經過排序時，會建構出一棵傾斜樹。

## 6 二元搜尋樹

63

- 例：將(25, 30, 24, 58, 45, 26, 12, 14) 建構為二元搜尋樹。

1. 插入 25

25



2. 插入 30

25

30



3. 插入 24

25

24

30



4. 插入 58

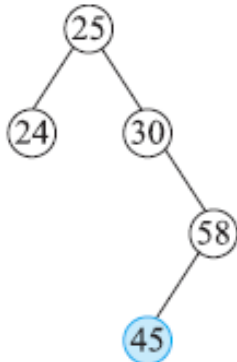
25

24

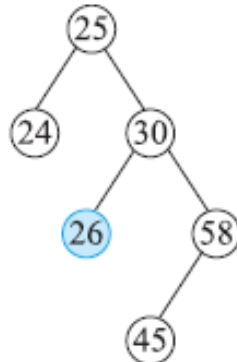
30

58

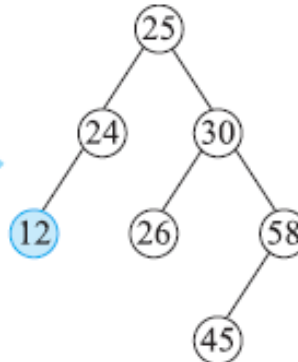
5. 插入 45



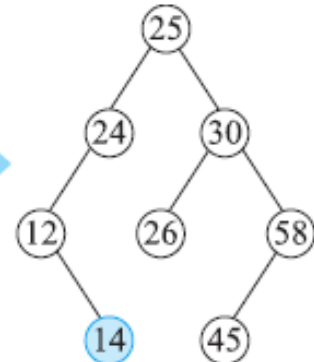
6. 插入 26



7. 插入 12



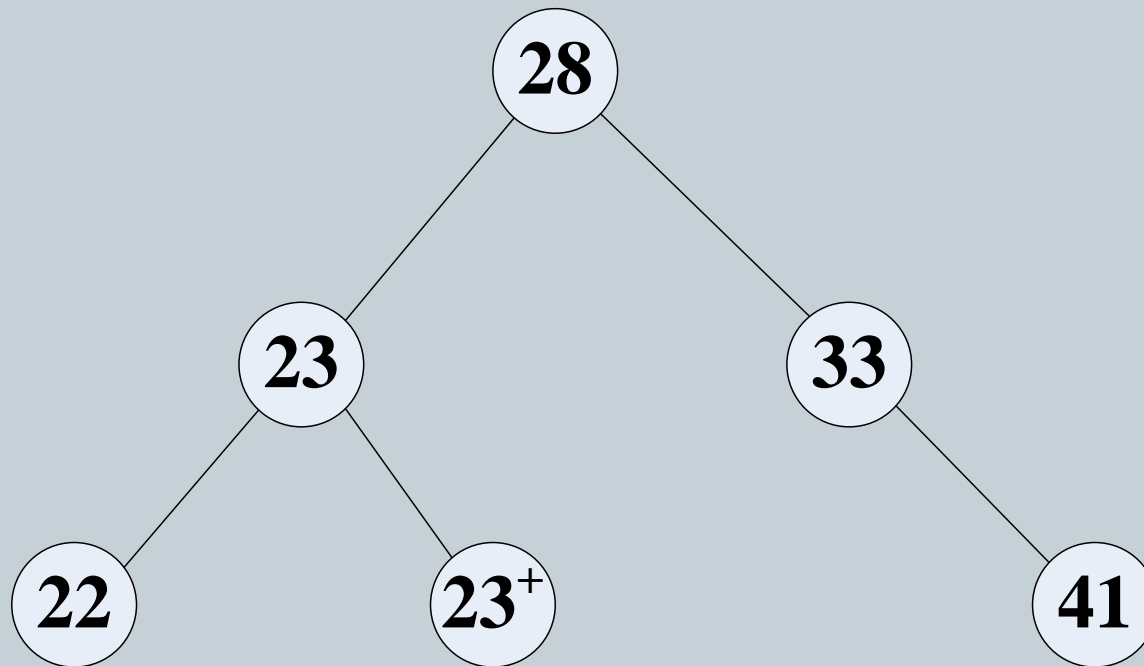
8. 插入 14



## 6 二元搜尋樹

64

- 【隨堂練習】請依以下順序建構二元搜尋樹：28, 23, 33, 41, 22, 23<sup>+</sup>。(註：23<sup>+</sup>與23視為不同的資料)
- 【解答】

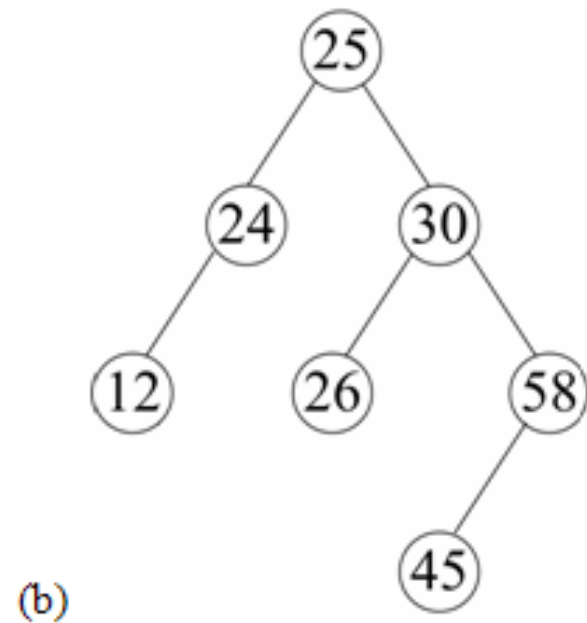
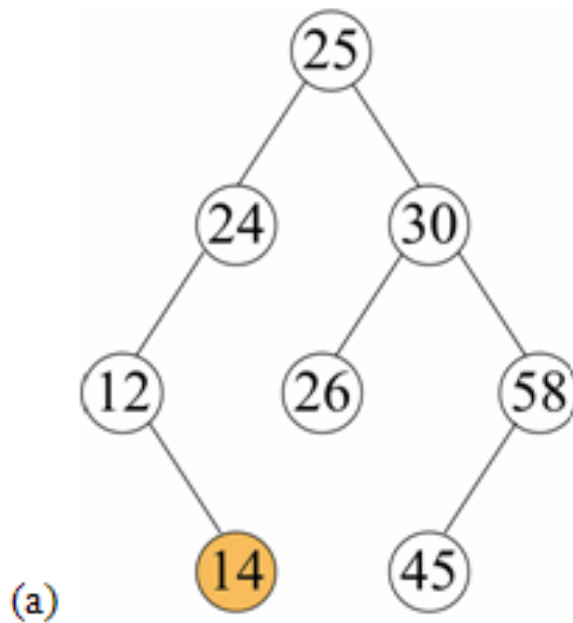




## 6 二元搜尋樹

65

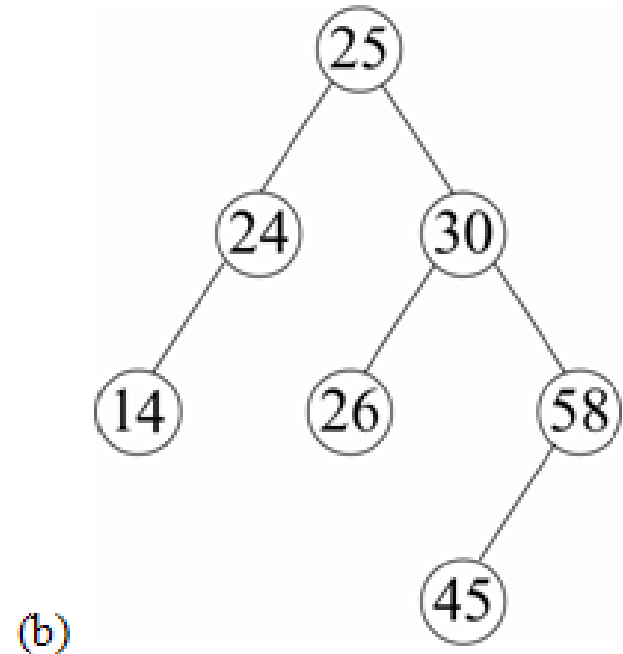
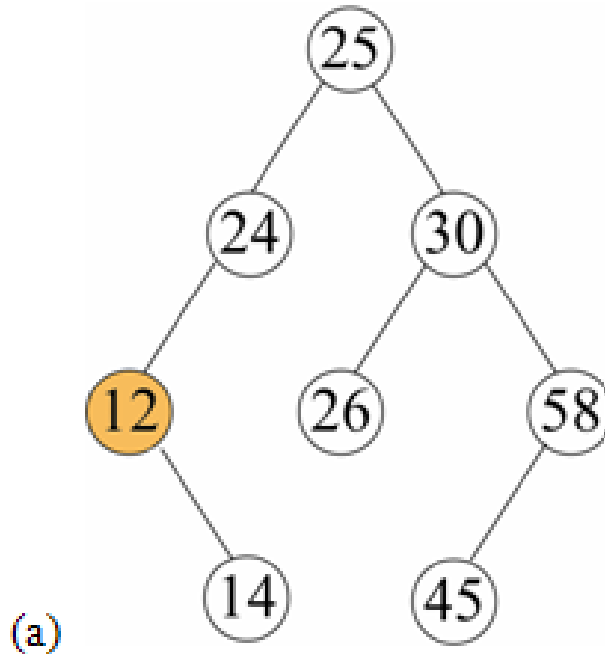
- 刪除節點：
- 情況一：當欲刪除的節點是樹葉時，直接刪除該節點即可。



## 6 二元搜尋樹

66

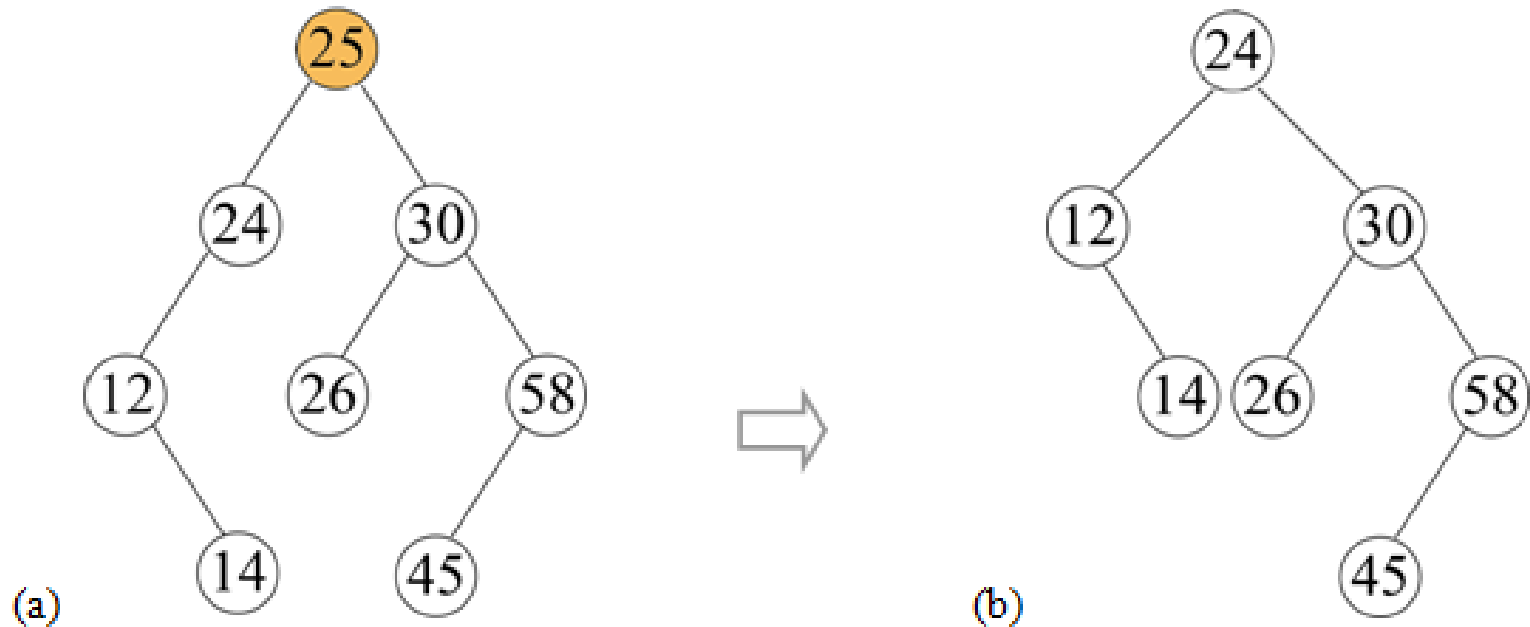
- 情況二：當欲刪除的節點有一棵子樹時，將其父節點的指標改指向其子節點，然後刪除該節點即可。



## 6 二元搜尋樹

67

- 情況三：當欲刪除的節點有兩棵子樹時，以該節點的左子樹中最大節點或右子樹中最小節點填入其位置，然後刪除該節點即可。



# 7 霍夫曼樹

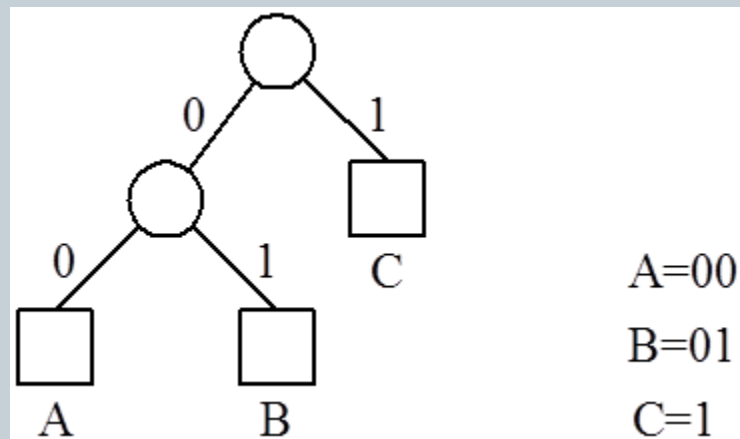
68

- 霍夫曼樹(Huffman Tree)是二元搜尋樹的延伸應用，它是依據霍夫曼碼(Huffman Coding)所建構的編碼樹。
- 霍夫曼碼是一種不固定長度的編碼技術，符號的編碼長度與出現頻率成反比。
- 特性：
  - 根據資料出現頻率多寡來建造的二元樹。
  - 霍夫曼樹的樹葉節點用以儲存資料元素。
  - 霍夫曼樹可用來編碼，也可用來解碼以達成資料壓縮目的。
  - 若該元素出現頻率愈高則從根節點到該元素所經過的節點路徑愈短，反之則愈長。

# 7 霍夫曼樹

69

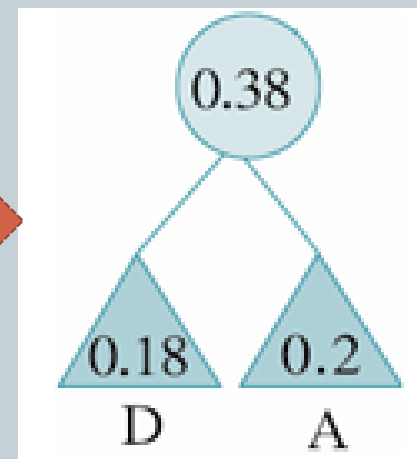
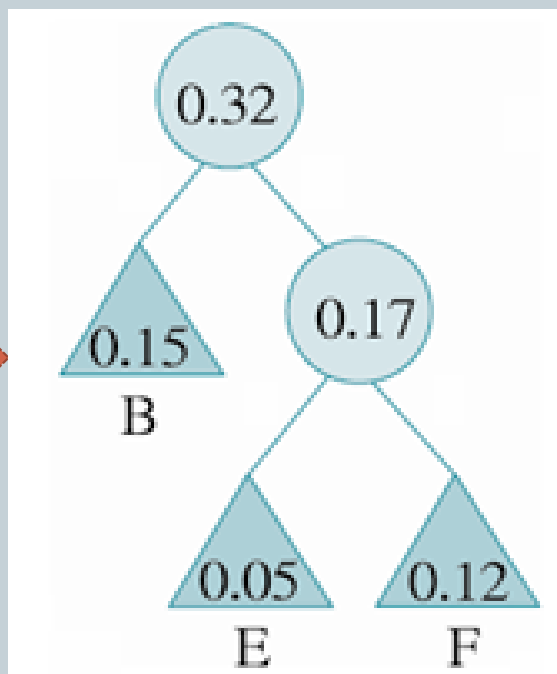
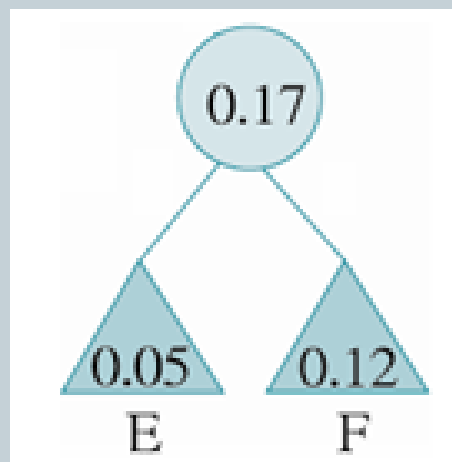
- 霍夫曼碼的編碼步驟如下：
  1. 找出所有符號的出現頻率。
  2. 將頻率最低的兩者相加得出另一個頻率。
  3. 重覆步驟2.，持續將頻率最低的兩者相加，直到剩下一個頻率為止。
  4. 根據合併的關係配置0與1，進而形成一棵編碼樹。



## 7 霍夫曼樹

70

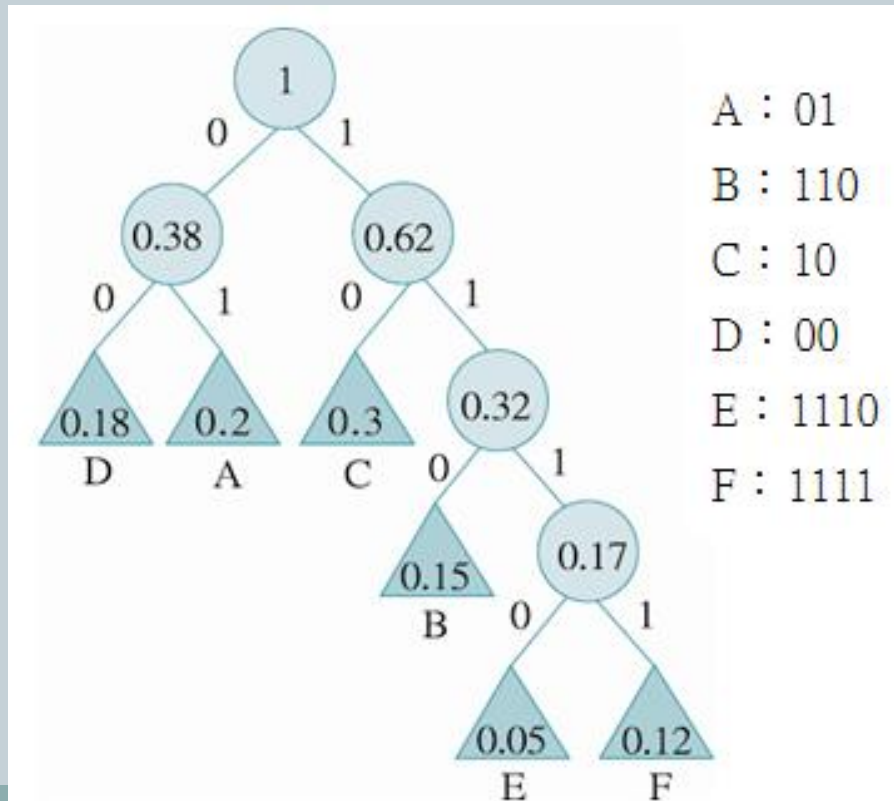
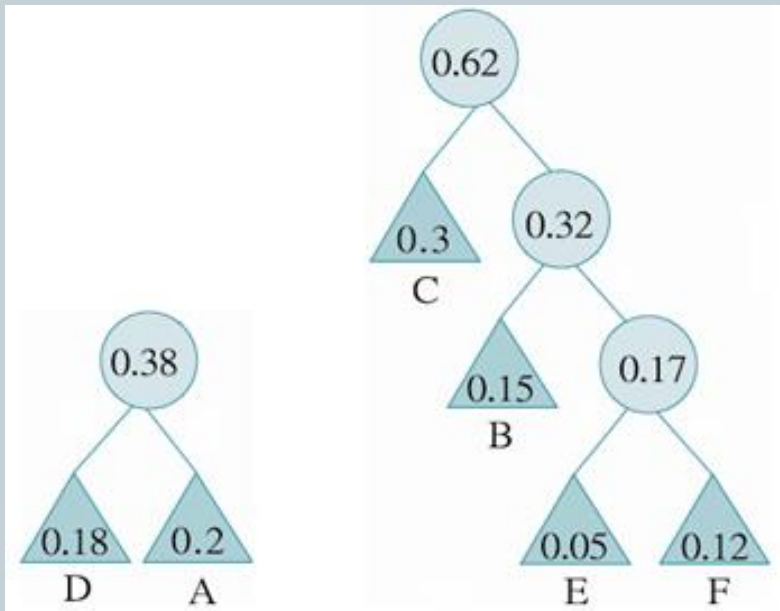
- 假設A、B、C、D、E、F等六個符號的出現頻率為0.2、0.15、0.3、0.18、0.05、0.12，試據此建構霍夫曼樹並寫出各個符號的編碼。



# 7 霍夫曼樹

71

- 假設A、B、C、D、E、F等六個符號的出現頻率為0.2、0.15、0.3、0.18、0.05、0.12，試據此建構霍夫曼樹並寫出各個符號的編碼。



## 7 霍夫曼樹

72

- [霍夫曼碼解碼] 根據上一個範例所設計的霍夫曼碼，將111110010000110進行解碼。
- 由於A、B、C、D、E、F等六個符號分別被編碼為01、110、10、00、1110、1111，故將111110010000110解碼後會得到FCADDB。



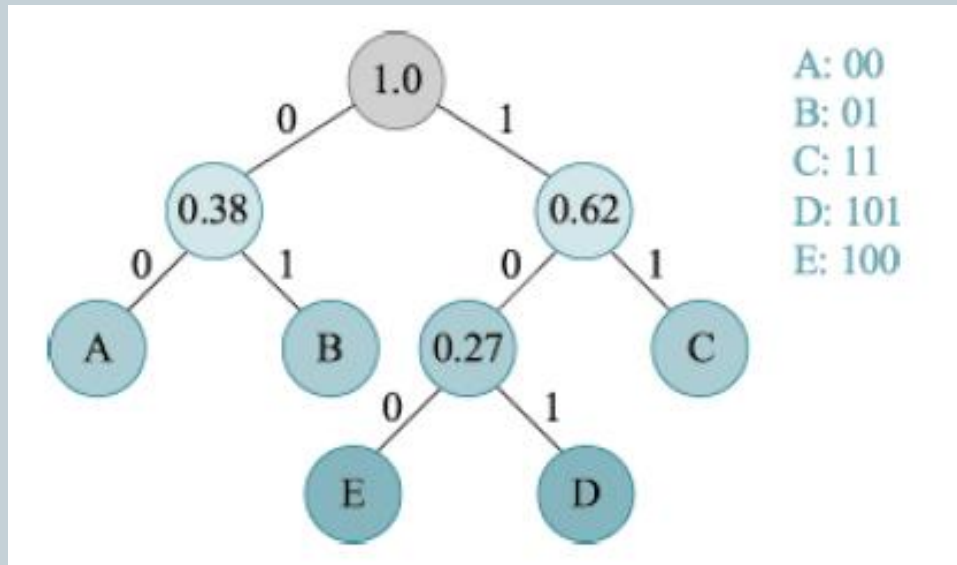
# 7 霍夫曼樹

73

## 【課堂練習】

- 1. 假設A、B、C、D、E出現頻率為0.18、0.20、0.35、0.15、0.12，試據此建構霍夫曼樹，並寫出各符號的編碼。
- 2. 根據此一霍夫曼樹，將1111101000101100101解碼。

【解答】1.



2. CCDABBED

# 7 霍夫曼樹

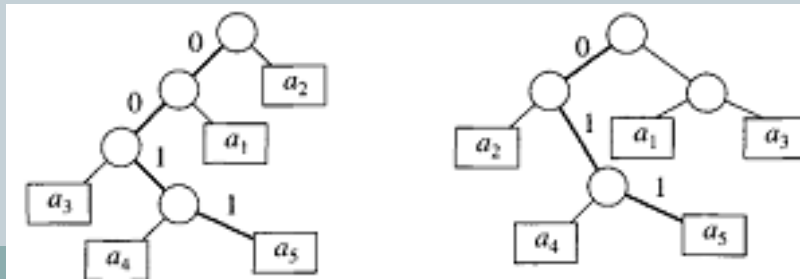


- When more than two symbols in a Huffman tree *have the same probability*, **different merge orders produce different Huffman codes.**

Symbol	Step 1	Step 2	Step 3	Step 4	Codeword
$a_2$	0.4	0.4	0.4	0.6 0	00
$a_1$	0.2	0.2	0.4 } 0	0.4 1	10
$a_3$	0.2	0.2 } 0	0.2 } 1		11
$a_4$	0.1 } 0	0.2 } 1			010
$a_5$	0.1 } 1				011

The average codeword length is still 2.2 bits/symbol. But variances are different!

- Two code trees with same symbol probabilities.



We prefer a code with smaller length-variance, Why?

# 7 霍夫曼樹



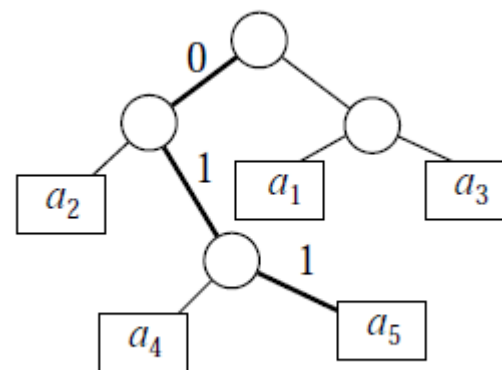
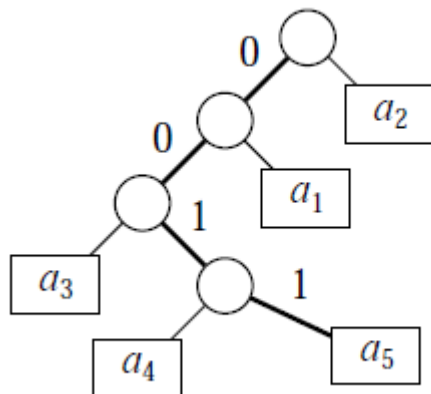
- To obtain the Huffman code with minimum variance, we always put the *combined letter* as high in the list as possible.

**TABLE 3.2 The reduced four-letter alphabet.**

Letter	Probability	Codeword
$a_2$	0.4	$c(a_2)$
$a_1$	0.2	$c(a_1)$
$a_3$	0.2	$c(a_3)$
$a'_4$	0.2	$\alpha_1$

**TABLE 3.6 Reduced four-letter alphabet.**

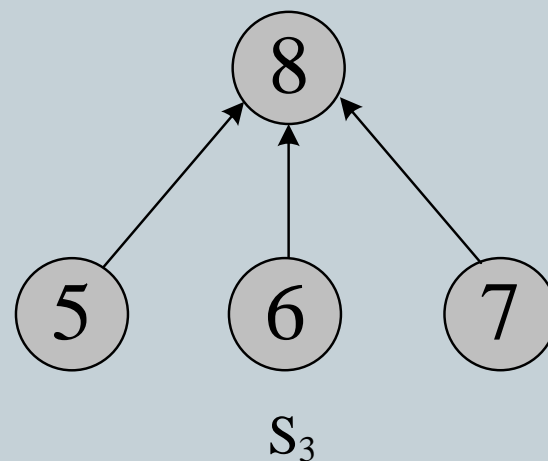
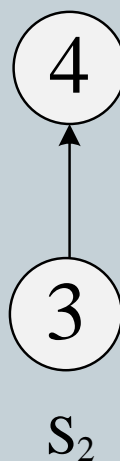
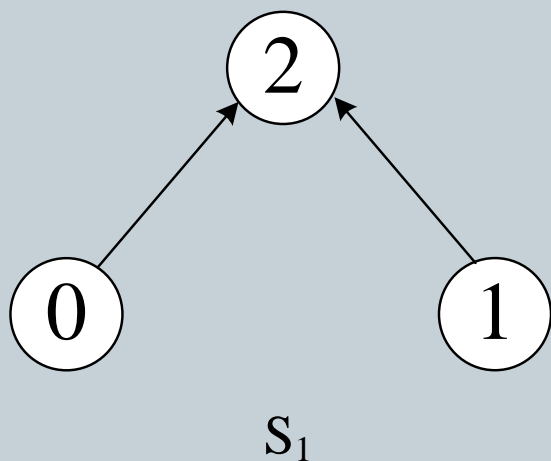
Letter	Probability	Codeword
$a_2$	0.4	$c(a_2)$
$a'_4$	0.2	$\alpha_1$
$a_1$	0.2	$c(a_1)$
$a_3$	0.2	$c(a_3)$



# 8 集合

76

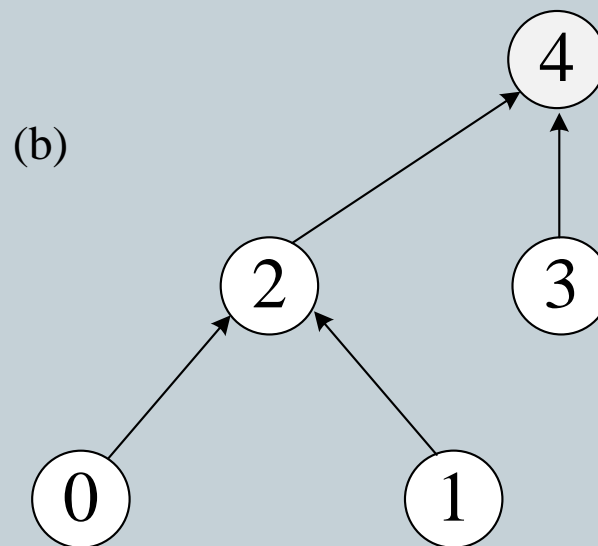
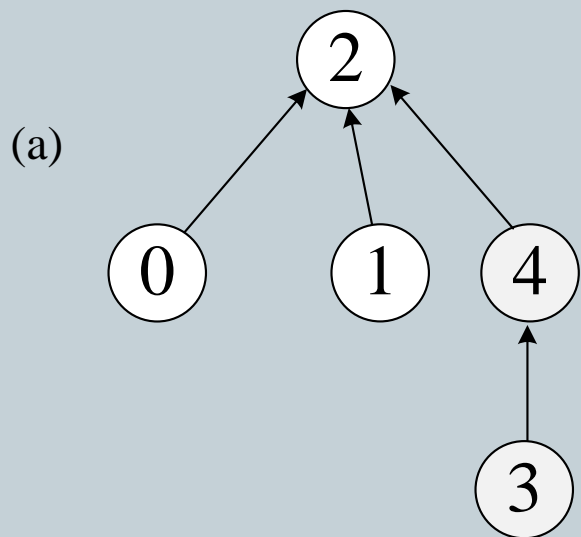
- 集合 (set) 是一群相同類型、**沒有順序之分**的元素。
- **互斥集合 (disjoin set)** 是沒有相同元素的集合。



# 8 集合

77

- 互斥集合常見的運算有下列兩種：
  - 聯集 (union)
  - 搜尋 (find)

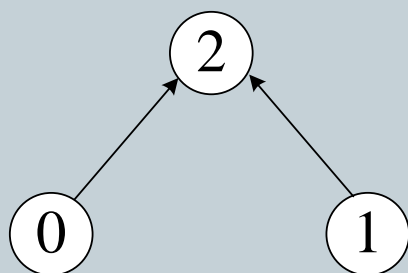


(a)  $S_1 \cup S_2$  (b)  $S_2 \cup S_1$

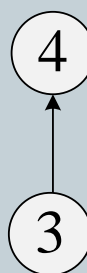
# 8 集合

78

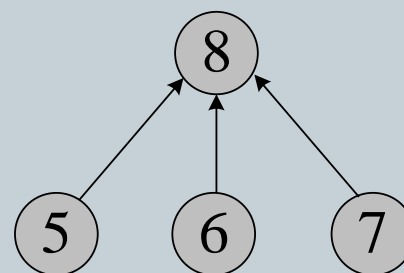
- 使用陣列存放互斥集合：
  - $i$  表示為節點索引值。
  - $\text{parent}[i]$  表示  $i$  的父節點。
  - 當  $\text{parent}[i] = -1$  時，表示為樹根。



$S_1$



$S_2$



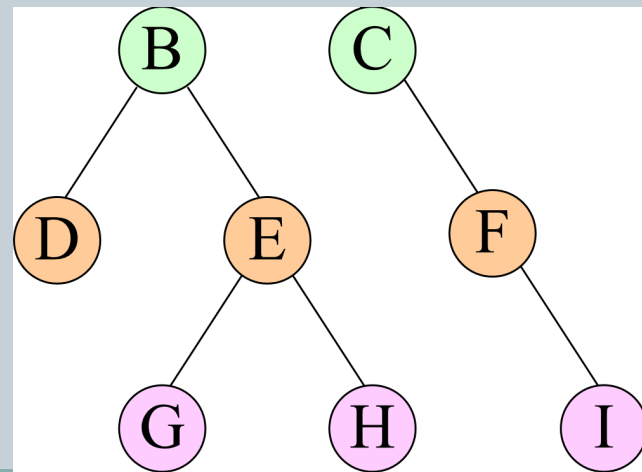
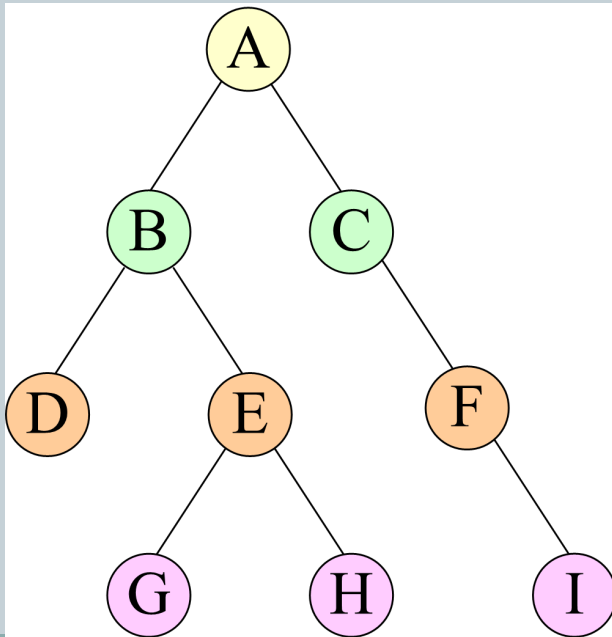
$S_3$

$i$	0	1	2	3	4	5	6	7	8
$\text{parent}[i]$	2	2	-1	4	-1	8	8	8	-1

# 9 樹林

79

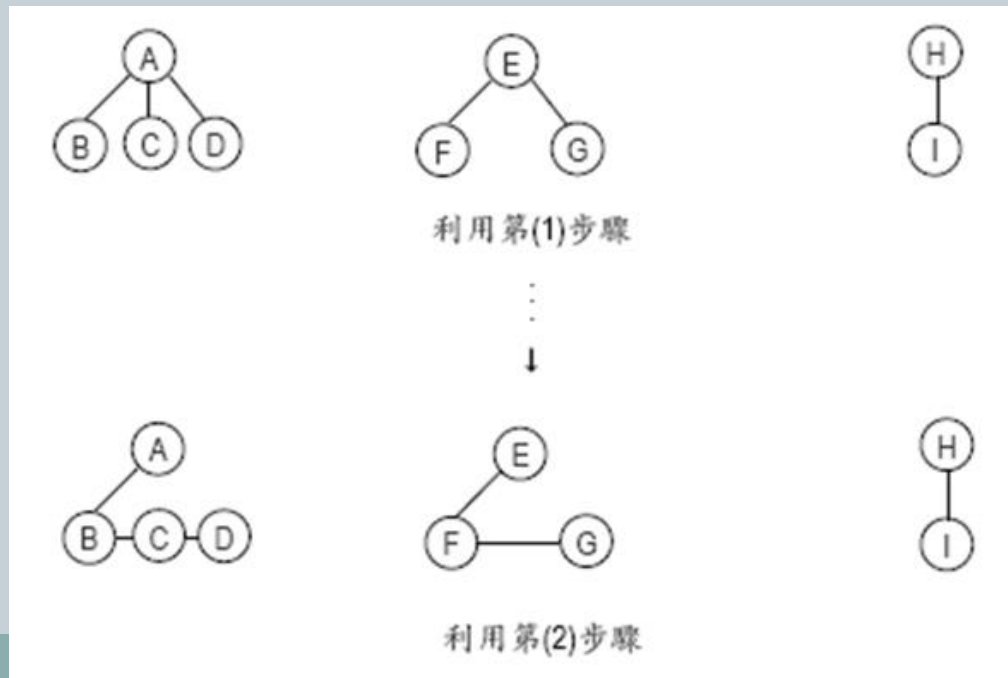
- 樹林 (forest) :
  - 是由 $n$ 棵互斥樹所組成的集合 ( $n \geq 0$ )，適合用來表示互斥集合 (disjoin set)。
  - 只要把樹根去掉，剩下的便是樹林。



# 9 樹林

80

- 樹林轉為二元樹，其步驟如下：
  - Step 1. 先將樹林中的每棵樹化為二元樹(不旋轉45度)。
  - Step 2. 把所有二元樹利用樹根節點全部鏈結在一起。
  - Step 3. 旋轉45度。

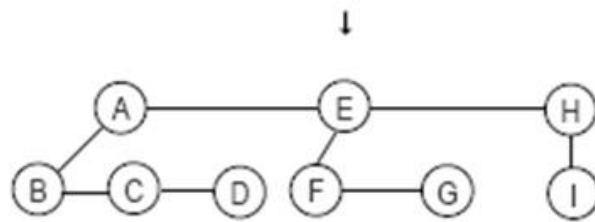




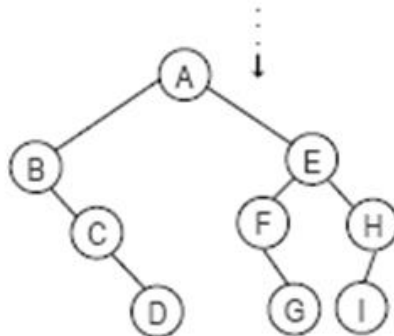
# 9 樹林

81

- 樹林轉為二元樹，其步驟如下：
  - Step 1. 先將樹林中的每棵樹化為二元樹(不旋轉45度)。
  - Step 2. 把所有二元樹利用樹根節點全部鏈結在一起。
  - Step 3. 旋轉45度。



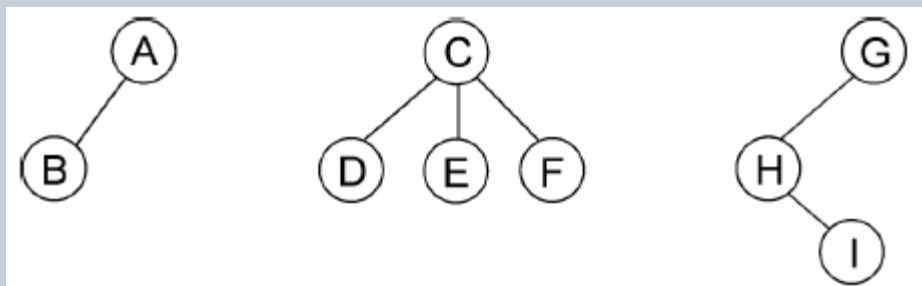
利用第(3)步驟



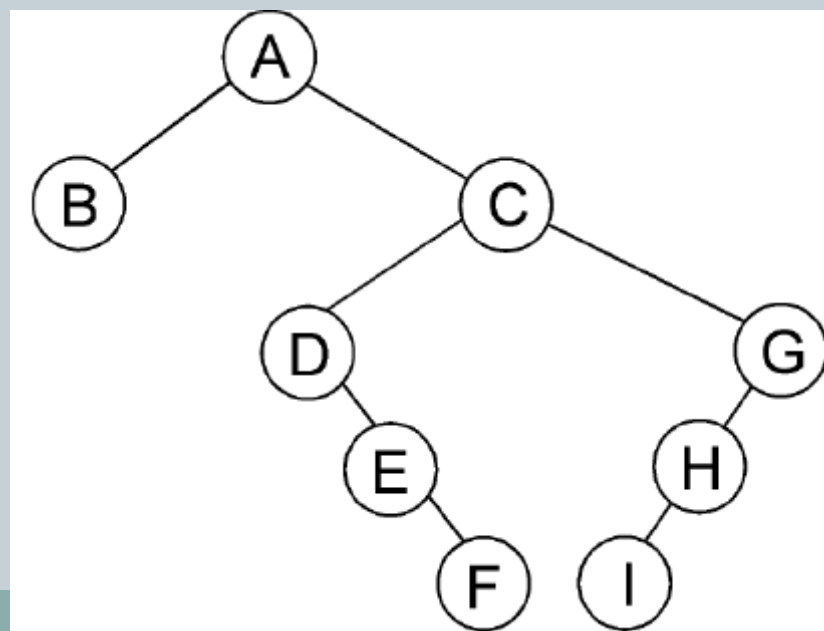
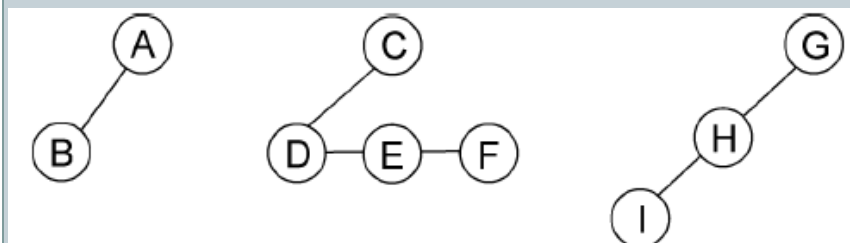
## 9 樹林

82

- 【隨堂練習】請將下圖樹林轉換成二元樹。



- 【解答】



# 10 Tree STL

83

- The Standard Template Library does not provide any templates with Tree in their name.
- However, some of its containers — the **set**, **map**, **multiset**, and **multimap** templates — are generally built using a special kind of self-balancing binary search tree called a **red-black**

# 10 Tree STL

84

## • set

- 方便快捷找尋資料是否已經存在，不允許資料重複。
- 插入資料時，以二元搜尋樹(binary search tree)實作set，讓set呈現已排序狀態，讓搜尋、插入與刪除資料有較好的效率。
- 插入資料時，會檢查是否已經有相同資料存在集合內，若資料相同則不會進行插入該元素，會回傳相同資料在set所在位置的指位器(iterator)。
-

# 10 Tree STL

85

- Set之基本動作：
  - insert(x): 插入元素x到set，會檢查是否x已經在集合內
  - erase(it): 刪除set的指位器it位置的元素
  - erase(x): 刪除set中數值為x的元素
  - find(x): 找出數值為x的元素，會檢查是否x已經在集合內，若x已經存在則回傳x所在的指位器，否則會回傳end位置的指位器。
  - count(x): 計算set中數值為x的個數，若x不在set中，則回傳0，否則回傳1。
  - empty()
  - size()

# 10 Tree STL

86

- 範例:
  - 實作一個程式將數字1,3,5,7,9依序加入set，再插入8。

```
#include <iostream>
#include <set>
using namespace std;
int main (){
    set<int> mset;
    set<int>::iterator it;
    for (int i=1; i<=9; i=i+2)
        mset.insert(i);
    mset.insert(8);
    for (it=mset.begin(); it!=mset.end(); it++)
        cout << *it << ' ';
    cout << endl; }
```

# 10 Tree STL

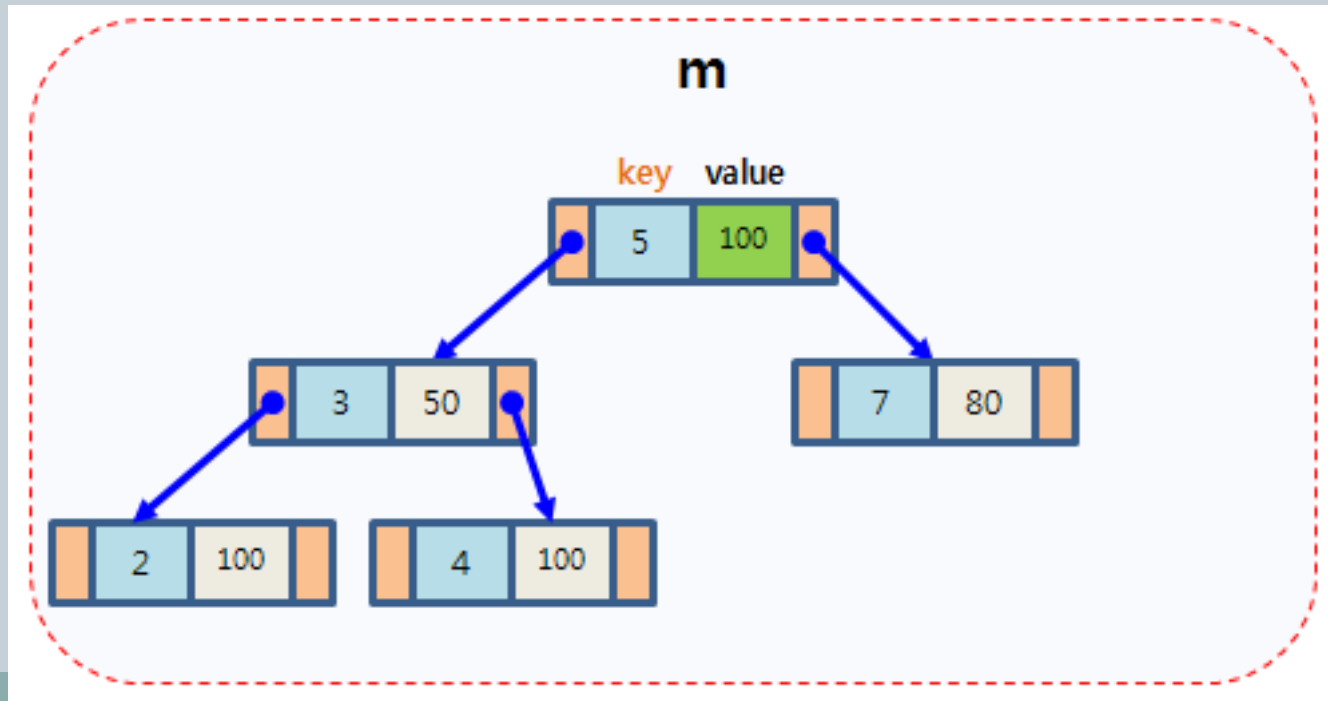
87

- **Map** 是 C++ 標準程式庫中的一個 class，為眾多容器(container)之一，使用前須包含 **map** 函式庫
- 它提供**搜尋和插入**的資料結構，並具有一對一 mapping 功能：
  - **map(key, value)**
  - 第一個為關鍵字 (**key**)。
  - 第二個為該關鍵字的值 (**value**)。
    - ✦ 可以修改 value 值、不能修改 **key** 值。

# 10 Tree STL

88

- map 資料結構為一顆紅黑樹 (red-black tree)：
  - 內部是**有排序**的資料。
  - 搜尋和插入演算法的效率  **$O(\log n)$** 。





# 10 Tree STL



- 例: 利用array加入key

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

int main()
{
    map<int, string> Employees;

    Employees[1234] = "Mike C.";
    Employees[1235] = "Charlie M.";
    Employees[1236] = "David D.";

    cout << "Employees[1234]= " <<
    Employees[1234] << endl << endl;
```

```
    cout << "Map size: " <<
    Employees.size() << endl;

    for(map<int, string>::iterator
    i=Employees.begin();
    i!=Employees.end(); i++)
        cout << (*i).first << ": " <<
        (*i).second << endl;

    return 0; }
```

# 10 Tree STL



- 例: 利用insert加入key

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

int main()
{
    map<int, string> Employees;

    Employees[1234] = "Mike C.";
    Employees[1235] = "Charlie M.";
    Employees[1236] = "David D.";

    Employees.insert(map<int,
string>::value_type(1237, "John
A."));
```

```
cout << "Employees[1234]= " <<
Employees[1234] << endl << endl;

cout << "Map size: " <<
Employees.size() << endl;

for(map<int, string>::iterator
i=Employees.begin();
i!=Employees.end(); i++)
    cout << (*i).first << ": " <<
(*i).second << endl;

return 0; }
```

# 11 期末實作(1/2)

91

- 請實作一套完整的霍夫曼編碼/解碼系統。
  - 主要功能：
    - ✦ 從螢幕讀取英文字母(A~Z, a~z)，需區分大小寫。
    - ✦ 針對英文字母進行**編碼**，並將**編碼結果**另存檔案。
    - ✦ 從檔案讀取**編碼**，進行**解碼**。
    - ✦ 輸出解碼結果(英文字母)。
  - 作業繳交：
    - ✦ WORD報告 + 程式碼
    - ✦ **12/7 23:59**前上傳至數位學習平台【期末實作1】

本章結束