# 第10章　資料結構

10-1 陣列

10-2 堆疊

10-3 佇列

10-4 鏈結串列

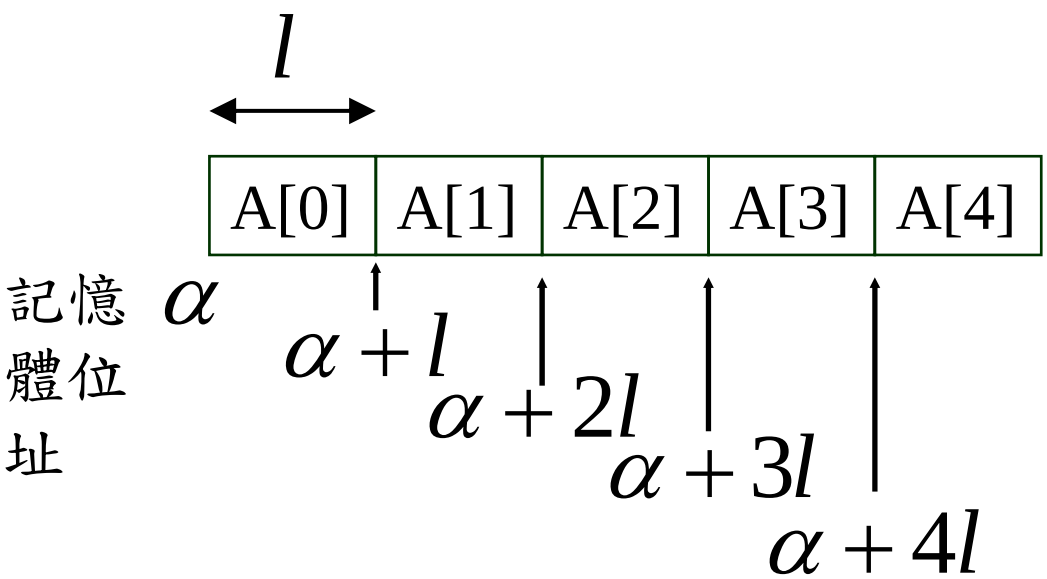
10-5 樹

# 10-1　陣列(array)

- 一個完整的資料結構（data structure）必須包含資料、相關運算的定義及函數。
- 陣列（array）和變數一樣是用來存放資料，不同的是陣列雖然只有一個名稱，卻能夠用來存放多個資料，這些資料叫做元素（element），陣列是透過索引（index）來區分各個元素。

# 一維陣列的記憶體配置

- 一維陣列A[0..4] ，每個元素A[0]~A[4]長度為皆為 $l$



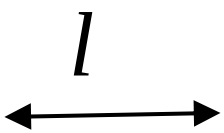

記憶體位址

| 元素 | 位址 |
|---|---|
| A[0] | $\alpha$ |
| A[1] | $\alpha+l$ |
| A[2] | $\alpha+2l$ |
| A[3] | $\alpha+3l$ |
| A[4] | $\alpha+4l$ |

- 記憶體位址公式 $A[i]=\alpha+i\cdot l$

# 二維陣列的記憶體配置

一維陣列A[0..1][0..2] ，每個元素A[0][0]~A[1][2]長度為皆為 $l$




| A[0][0] | A[0][1] | A[0][2] |
|---|---|---|
| A[1][0] | A[1][1] | A[1][2] |

記憶體位址

| 元素 | 位址 |
|---|---|
| A[0][0] | $\alpha$ |
| A[0][1] | $\alpha + l$ |
| A[0][2] | $\alpha + 2l$ |
| A[1][0] | $\alpha + 3l$ |
| A[1][1] | $\alpha + 4l$ |
| A[1][2] | $\alpha + 5l$ |

# 二維陣列的記憶體配置(續)

- 二維陣列A[0..$n_1$][0..$n_2$] ，每個元素A[0][0]~A[$n_1$][$n_2$]長度為皆為 $l$

- 記憶體位址公式

$$A[i][j] = \alpha + (i \cdot (n_2 + 1) + j) \cdot l$$

- 例如：二維陣列A[0..1][0..2]

$$A[1][2] = \alpha + (1 \cdot (2+1) + 2) \cdot l$$

$$= \alpha + 5l$$

# 三維陣列的記憶體配置

三維陣列A[0..1][0..1][0..2] ，每個元素A[0][0][0]~A[1][1][2]長度為皆為 $l$


| | | |
|---|---|---|
| A[0][0][0] | A[0][0][1] | A[0][0][2] |
| A[0][1][0] | A[0][1][1] | A[0][1][2] |
| A[1][0][0] | A[1][0][1] | A[1][0][2] |
| A[1][1][0] | A[1][1][1] | A[1][1][2] |

記憶體位址

| 元素 | 位址 |
|---|---|
| A[0][0][0] | $\alpha$ |
| A[0][0][1] | $\alpha + l$ |
| A[0][0][2] | $\alpha + 2l$ |
| A[0][1][0] | $\alpha + 3l$ |
| A[0][1][1] | $\alpha + 4l$ |
| A[0][1][2] | $\alpha + 5l$ |

# 三維陣列的記憶體配置(續)

三維陣列A[0..1][0..1][0..2] ，每個元素A[0][0][0]~A[1][1][2]長度為皆為 $l$


| | | |
|---|---|---|
| A[0][0][0] | A[0][0][1] | A[0][0][2] |
| A[0][1][0] | A[0][1][1] | A[0][1][2] |
| A[1][0][0] | A[1][0][1] | A[1][0][2] |
| A[1][1][0] | A[1][1][1] | A[1][1][2] |

記憶體位址

| 元素 | 位址 |
|---|---|
| A[1][0][0] | $\alpha+6l$ |
| A[1][0][1] | $\alpha+7l$ |
| A[1][0][2] | $\alpha+8l$ |
| A[1][1][0] | $\alpha+9l$ |
| A[1][1][1] | $\alpha+10l$ |
| A[1][1][2] | $\alpha+11l$ |

# 三維陣列的記憶體配置(續)

- 三維陣列A[0..$n_1$][0..$n_2$][0..$n_3$] ，每個元素A[0][0][0]~A[$n_1$][$n_2$][$n_3$]長度為皆為 $l$

- 記憶體位址公式

$$A[i][j][k] = \alpha + (i \cdot (n_2 + 1)(n_3 + 1) + j \cdot (n_3 + 1) + k) \cdot l$$

- 例如：三維陣列A[0..1][0..1][0..2]

$$A[1][1][2] = \alpha + (1 \cdot (1+1)(2+1) + 1 \cdot (2+1) + 2) \cdot l$$

$$= \alpha + 11l$$

| | 第 0 行 | 第 1 行 | 第 2 行 | …… | 第 n-1 行 |
|---|---|---|---|---|---|
| 第 0 列 | | 國語 | 自然 | …… | 數學 |
| 第 1 列 | 王小美 | 85 | 88 | …… | 77 |
| 第 2 列 | 孫大偉 | 99 | 86 | …… | 89 |
| …… | …… | …… | …… | …… | …… |
| 第 m-1 列 | 張婷婷 | 75 | 92 | …… | 86 |

圖 10.2
成績單

| | 第 0 行 | 第 1 行 | 第 2 行 | …… | 第 m-1 行 |
|---|---|---|---|---|---|
| 第 0 列 | [0][0] | [0][1] | [0][2] | …… | [0][n-1] |
| 第 1 列 | [1][0] | [1][1] | [1][2] | …… | [1][n-1] |
| 第 2 列 | [2][0] | [2][1] | [2][2] | …… | [2][n-1] |
| …… | …… | …… | …… | …… | …… |
| 第 m-1 列 | [m-1][0] | [m-1][1] | [m-1][2] | …… | [m-1][n-1] |

圖 10.3
以二維陣列表示成績單

# 陣列的應用

- 存放多項式

以 $8X^4 - 6X^2 + 3X^5 + 5$ 為例，我們先依照冪次由高至低排列寫出 $3X^5 + 8X^4 - 6X^2 + 5X^0$ ，

| Polynomial | [0] | [1] | [2] | [3] |
|---|---|---|---|---|
| coef | 3 | 8 | -6 | 5 |
| exp | 5 | 4 | 2 | 0 |

矩陣大小宣告　**Polynomial**[0..1][0..3]

| 元素 |
| --- |
| **Polynomial**[0][0] = 3 |
| **Polynomial**[0][1] = 8 |
| **Polynomial**[0][2] = –6 |
| **Polynomial**[0][3] = 5 |
| **Polynomial**[1][0] = 5 |
| **Polynomial**[1][1] = 4 |
| **Polynomial**[1][2] = 2 |
| **Polynomial**[1][3] = 0 |

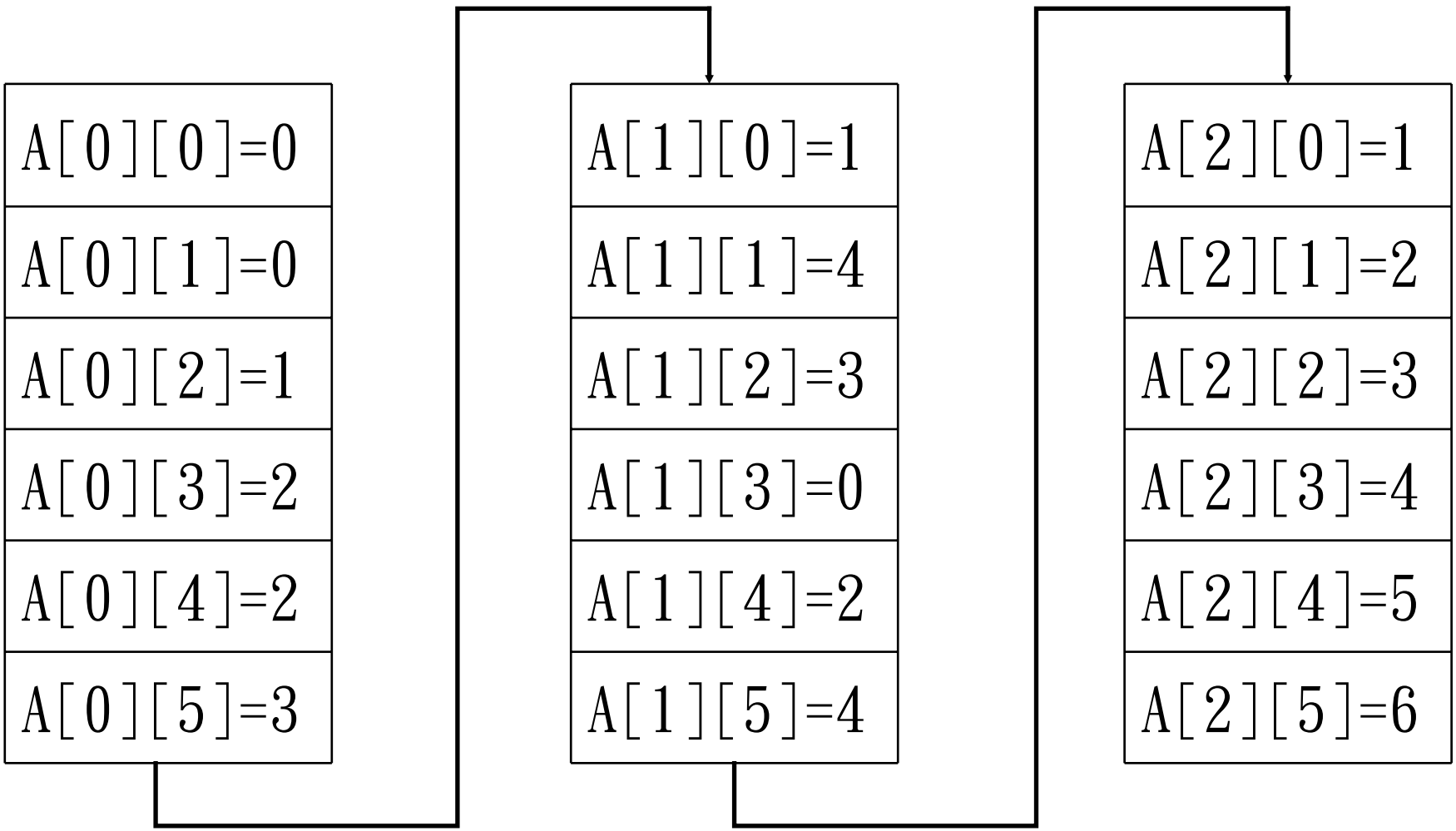
# ● 存放稀疏矩陣

| | col0 | col1 | col2 | col3 | col4 |
|---|---|---|---|---|---|
| Row0 | 0 | 1 | 0 | 0 | 2 |
| Row1 | 0 | 0 | 0 | 3 | 0 |
| Row2 | 4 | 0 | 5 | 0 | 0 |
| Row3 | 0 | 0 | 0 | 0 | 6 |

| SparseMatrix | [0] | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|---|
| row | 0 | 0 | 1 | 2 | 2 | 3 |
| col | 1 | 4 | 3 | 0 | 2 | 4 |
| value | 1 | 2 | 3 | 4 | 5 | 6 |

10.4
以陣列存放稀疏矩陣

矩陣大小宣告 **A**[0..2][0..5]

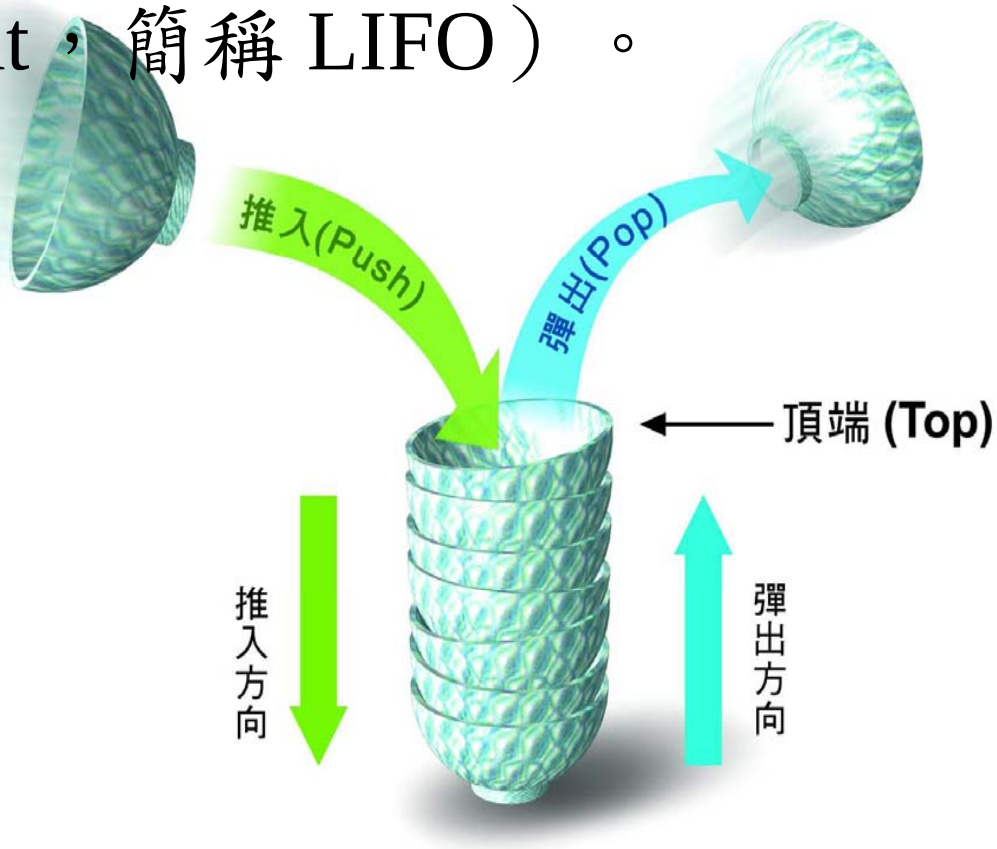


# 10-2 堆疊(stack)

- 堆疊是一個有順序的資料列，兩端分別稱為**頂端**與**底端**，新增或刪除資料，都必需從堆疊的**頂端**開始。
- 執行新增資料的動作，稱為**推入**(push)。
- 執行刪除資料的動作，稱為**彈出**(pop)。
- 由於越晚推入推疊的資料越早被彈出，故堆疊又稱為後進先出串列(last-in-first-out list, LIFO list)

# 堆疊結構

- 堆疊（Stack）是一種後進先出（Last In First Out，簡稱 LIFO）。

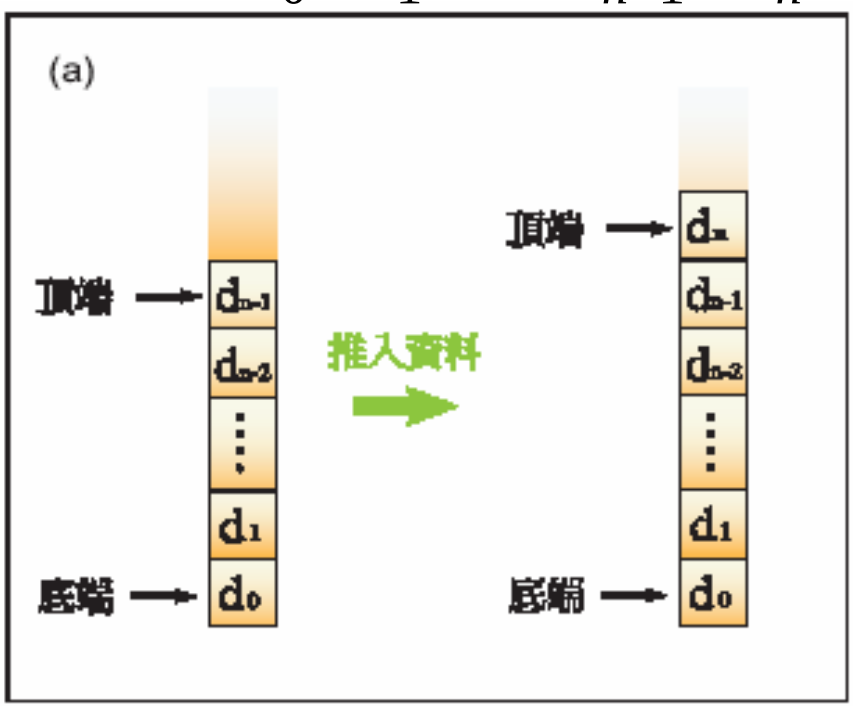


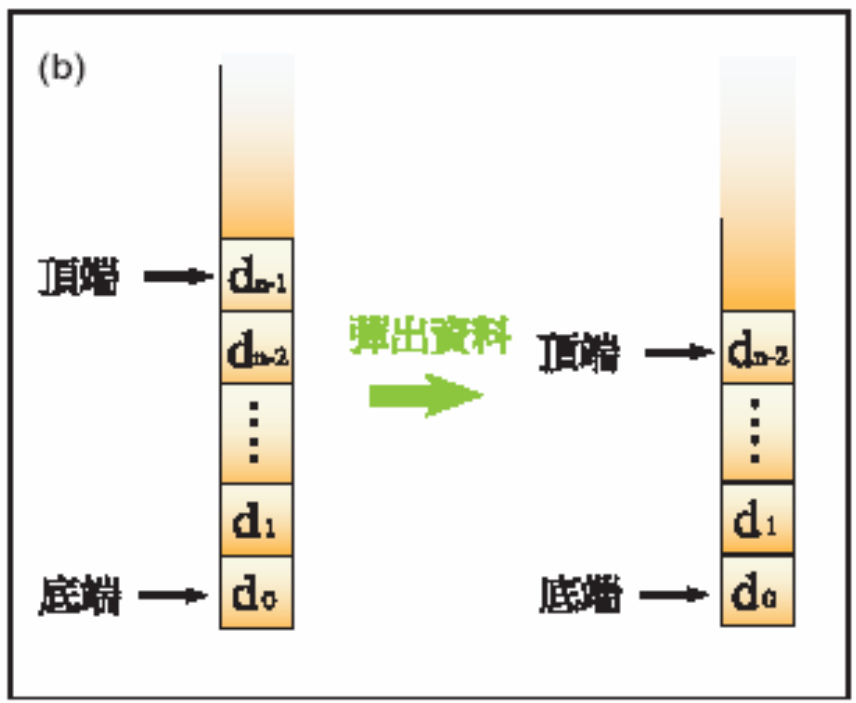
# 10-2 堆疊

- 例如：

$$S=(d_0,d_1,\cdots d_{n-1}) \xrightarrow{\text{push}} S=(d_0,d_1,\cdots d_{n-1},d_n)$$

$$S=(d_0,d_1,\cdots d_{n-1}) \xrightarrow{\text{pop}} S=(d_0,d_1,\cdots d_{n-2})$$



堆疊經常應用於下列幾個方面：

- 資料反轉（data reversing）
- 回溯（backtracking）
- 資料剖析（data parsing）
- 運算式表示法轉換
  - 中序表示法（infix） $A*(B+C)-D$
  - 前序表示法（prefix） $-*A+BCD$
  - 後序表示法（postfix） $ABC+*D-$

# 使用堆疊將運算式A * (B + C) - D由中序表示法轉換成後序表示法：

| 符號 | 堆疊 [0] [1] [2] | 頂端 | 輸出 | 說明 |
|---|---|---|---|---|
| A | | -1 | A | 當符號為運算元時，直接將它輸出。 |
| * | * | 0 | A | 當符號為運算子時，將它推入堆疊。 |
| ( | * ( | 1 | A | 當符號為左括弧時，將它推入堆疊。 |
| B | * ( | 1 | AB | 當符號為運算元時，直接將它輸出。 |
| + | * ( + | 2 | AB | 當符號為運算子時，將它推入堆疊。 |
| C | * ( + | 2 | ABC | 當符號為運算元時，直接將它輸出。 |
| ) | * | 0 | ABC+ | 當符號為右括弧時，彈出堆疊的資料直到左括弧。 |
| - | - | 0 | ABC+* | 當符號為優先順序較低的運算元時，先彈出堆疊的資料，再將它推入堆疊。 |
| D | - | 0 | ABC+*D | 當符號為運算元時，直接將它輸出。 |
| | | -1 | ABC+*D- | 符號處理完畢後彈出堆疊的資料，以得到結果。 |

# 轉換成後序表示法原則

- 以由左至右方式讀入
- 當符號為((左括弧)，則放入堆疊。
- 當符號為)(右括弧)，則將堆疊最上層的符號彈出。
- 當符號為運算子，其優先順序比堆疊最上層符號的優先低時，則最上層符號彈出。
- 當符號為運算子，其優先順序比堆疊最上層符號的優先高時，則將此符號推入。
- 當符號為運算元時，直接輸出。輸出方式由左至右。

# 例題

$A+B*C-D$

減號與加號優先權相同，但順序比加號低，因為加號在減號左邊，需先執行。所以將加號彈出，減號推入堆疊

| 符號 | 堆疊<br>[0][1][2] | 頂端 | 輸出 |
|---|---|---|---|
| A | | -1 | A |
| + | + | 0 | A |
| B | + | 0 | AB |
| * | +* | 1 | AB |
| C | +* | 1 | ABC |
| - | - | 0 | ABC*+ |
| D | - | 0 | ABC*+D |
| | | -1 | ABC*+D- |

# 轉換成前序表示法原則

- 以由右至左方式讀入。
- 當符號為)(右括弧），則放入堆疊。
- 當符號為（(左括弧），則將堆疊最上層的符號彈出。
- 當符號為運算子，其優先順序比堆疊最上層符號的優先低時，則最上層符號彈出。
- 當符號為運算子，其優先順序比堆疊最上層符號的優先高時，則將此符號推入。
- 當符號為運算元時，直接輸出。輸出方式由右至左。

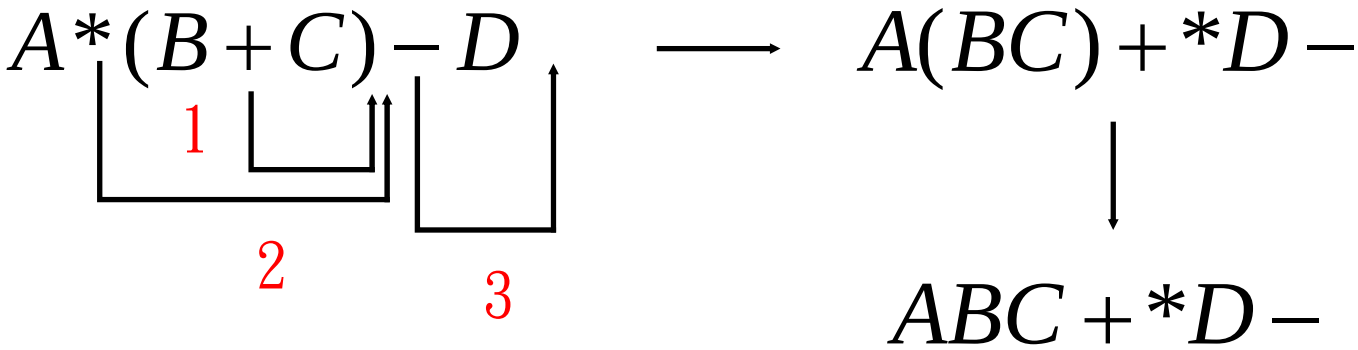
# 例題

$A+B*C-D$

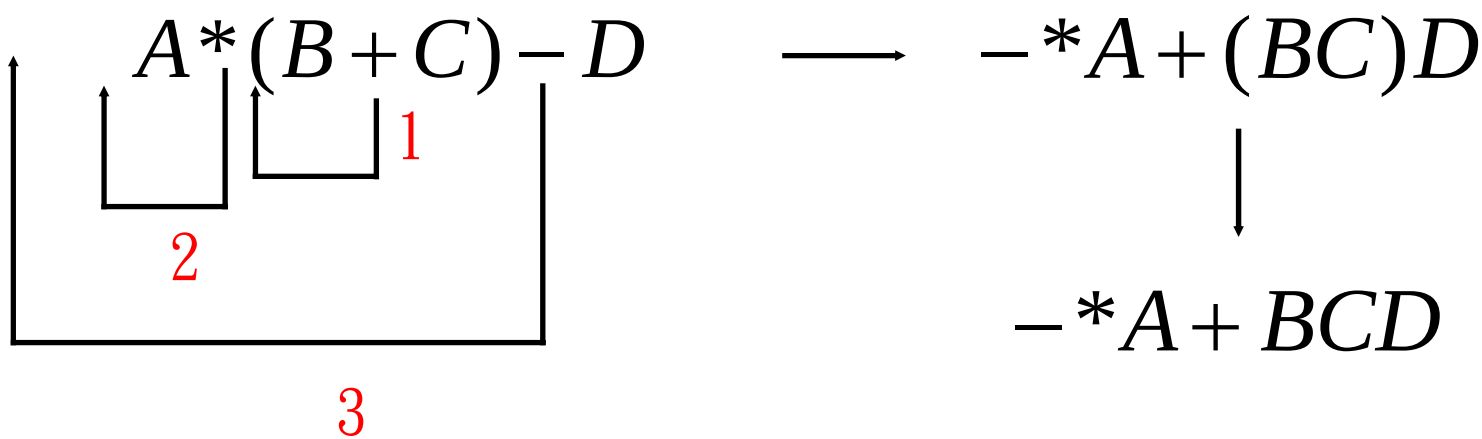
| 符號 | 堆疊<br>[0][1][2] | 頂端 | 輸出 |
|---|---|---|---|
| D | | -1 | D |
| - | - | 0 | D |
| C | - | 0 | CD |
| * | -* | 1 | CD |
| B | -* | 1 | BCD |
| + | -+ | 1 | *BCD |
| A | -+ | 1 | A*BCD |
| | | -1 | -+A*BCD |

# 快速方式

後序表示法




前序表示法

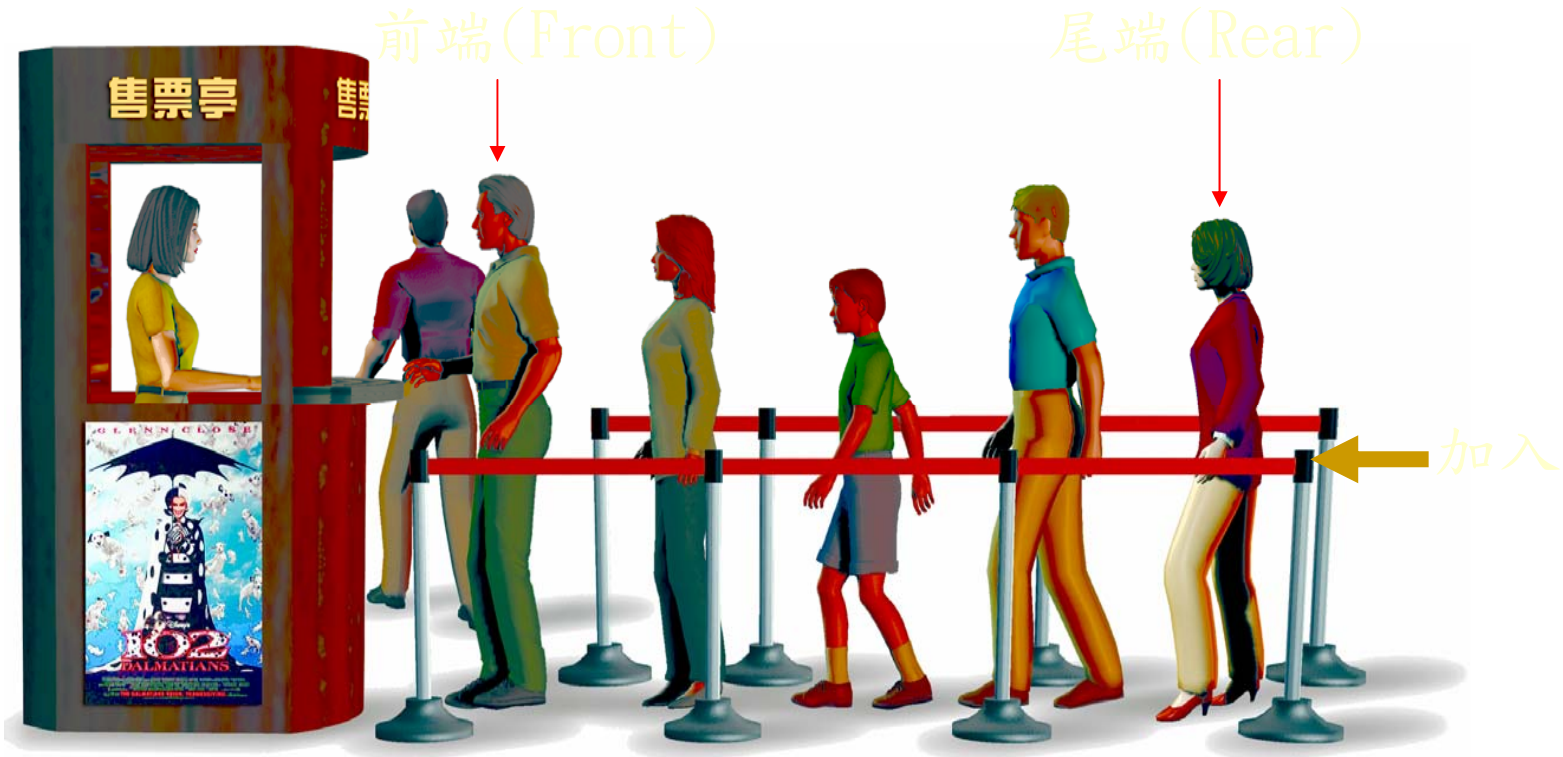


# 10-3 佇列(queue)

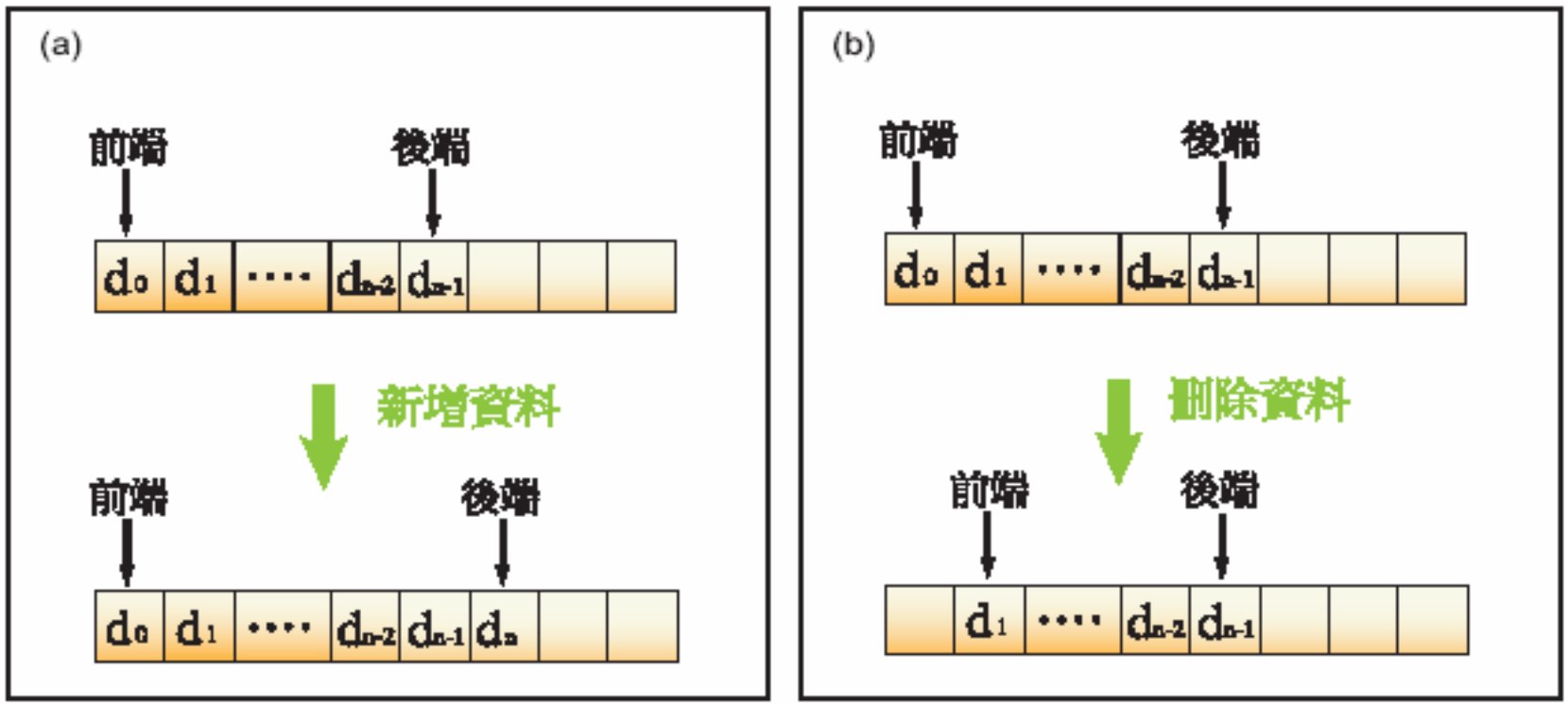
- 佇列是一個有順序的資料列，兩端分別稱為**前端**與**後端**。
- 新增資料，都必需從佇列的**後端**開始。
- 刪除資料，都必需從佇列的**前端**開始。
- 由於越早放入佇列的資料，越早被刪除，故佇列又稱為先進先出串列(first-in-first-out list, FIFO list) 。
- 經常應用於工作排程(job scheduling)

# 佇列結構

- 佇列（Queue）是一種先進先出（First In First Out，簡稱 FIFO）的資料結構。



# 10-3 佇列


(a)
前端
後端
d0 d1 ···· dn-2 dn-1
新增資料
前端
後端
d0 d1 ···· dn-2 dn-1 dn
(b)
前端
後端
d0 d1 ···· dn-2 dn-1
刪除資料
前端
後端
d1 ···· dn-2 dn-1

假設佇列的最大長度為 6，我們依序新增 J1 、J2 、J3 、J4 等四個工作，然後又依序刪除 J1 、J2 等兩個工作及新增 J5 、J6 、J7 等三個工作，那麼 front 、rear 的值與佇列的內容如下：

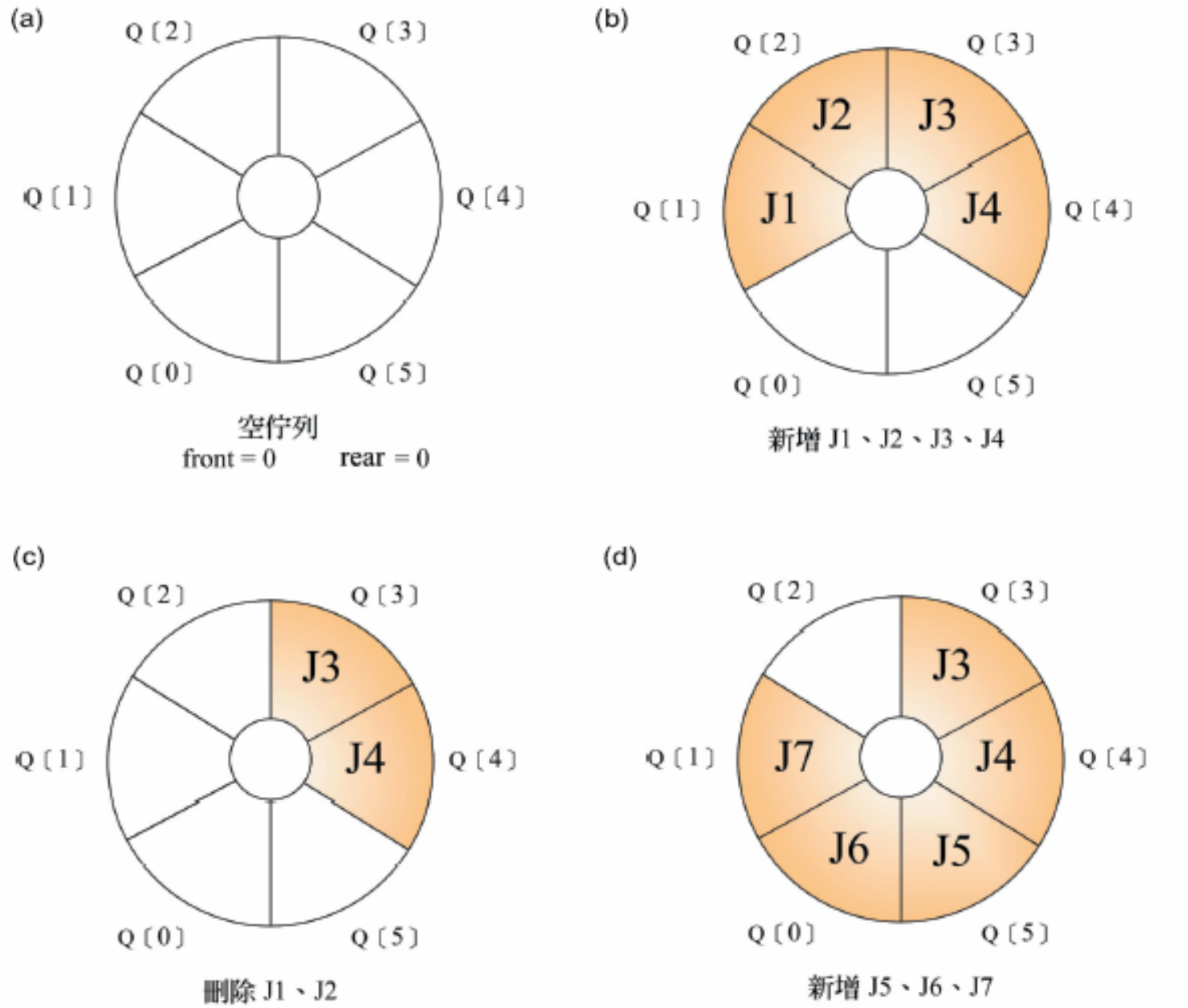
| 說明 | Q[0] | Q[1] | Q[2] | Q[3] | Q[4] | Q[5] | front | rear |
|---|---|---|---|---|---|---|---|---|
| 一開始為空佇列，front 等於 rear | | | | | | | -1 | -1 |
| 新增 J1 | J1 | | | | | | 0 | 0 |
| 新增 J2 | J1 | J2 | | | | | 0 | 1 |
| 新增 J3 | J1 | J2 | J3 | | | | 0 | 2 |
| 新增 J4 | J1 | J2 | J3 | J4 | | | 0 | 3 |
| 移除 J1 | | J2 | J3 | J4 | | | 1 | 3 |
| 移除 J2 | | | J3 | J4 | | | 2 | 3 |
| 新增 J5 | | | J3 | J4 | J5 | | 2 | 4 |
| 新增 J6 | | | J3 | J4 | J5 | J6 | 2 | 5 |
| 新增 J7 | rear 的值超過佇列的最大長度，雖然佇列的前端仍有位置可以存放資料，卻會顯示 Queue Full 。 | | | | | | | |

- 很明顯，經過多次新增或刪除後，佇列資料逐漸向後端移，待rear的值超過最大長度後，就會被判為Queue Full。
- 但實際上，佇列的前端卻可能有位置存放資料。
- 為此，便有環狀佇列(circular queue)方式的產生。

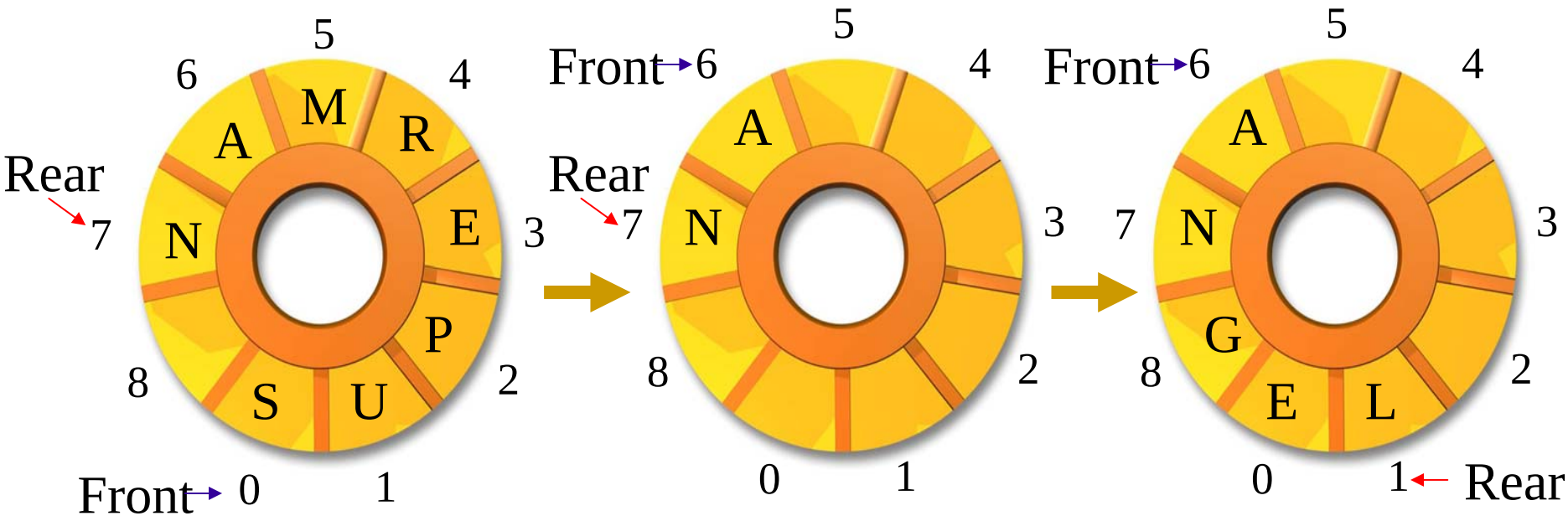
# 環狀佇列

前面的例子改成環狀佇列後， front 、 rear 的值與環狀佇列的內容如下：

| 說明 | Q[0] | Q[1] | Q[2] | Q[3] | Q[4] | Q[5] | front | rear |
|---|---|---|---|---|---|---|---|---|
| 一開始為空佇列，<br>front 等於 rear | | | | | | | 0 | 0 |
| 新增 J1 | | J1 | | | | | 1 | 1 |
| 新增 J2 | | J1 | J2 | | | | 1 | 2 |
| 新增 J3 | | J1 | J2 | J3 | | | 1 | 3 |
| 新增 J4 | | J1 | J2 | J3 | J4 | | 1 | 4 |
| 移除 J1 | | | J2 | J3 | J4 | | 2 | 4 |
| 移除 J2 | | | | J3 | J4 | | 3 | 4 |
| 新增 J5 | | | | J3 | J4 | J5 | 3 | 5 |
| 新增 J6 | J6 | | | J3 | J4 | J5 | 3 | 0 |
| 新增 J7 | J6 | J7 | | J3 | J4 | J5 | 3 | 1 |

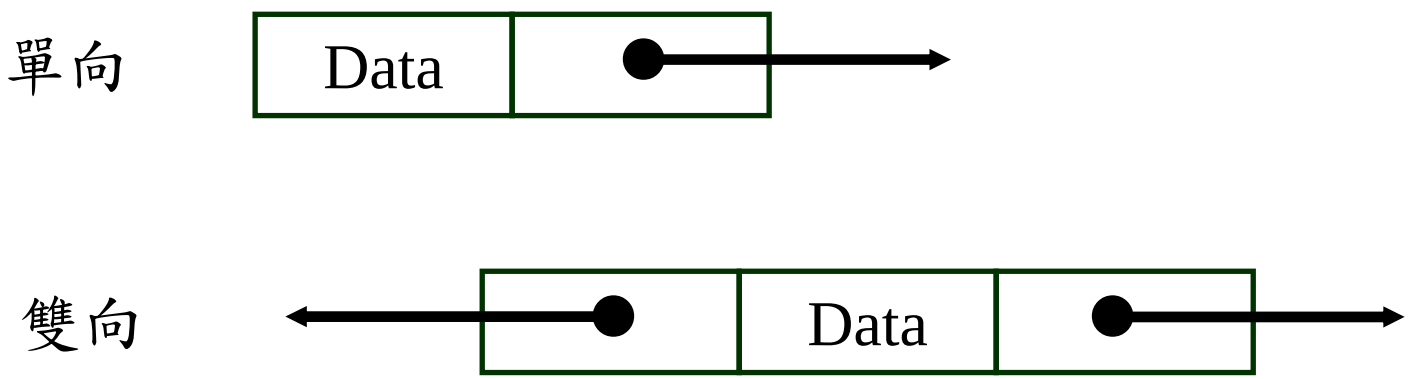

(a)
Q〔2〕
Q〔3〕
Q〔1〕
Q〔4〕
Q〔0〕
Q〔5〕
空佇列
front = 0
rear = 0
(b)
Q〔2〕
Q〔3〕
J2
J3
Q〔1〕
J1
J4
Q〔4〕
Q〔0〕
Q〔5〕
新增 J1、J2、J3、J4
(c)
Q〔2〕
Q〔3〕
J3
Q〔1〕
J4
Q〔4〕
Q〔0〕
Q〔5〕
刪除 J1、J2
(d)
Q〔2〕
Q〔3〕
J3
Q〔1〕
J7
J4
Q〔4〕
J6
J5
Q〔0〕
Q〔5〕
新增 J5、J6、J7

# 環狀佇列循環使用儲存位置示意圖

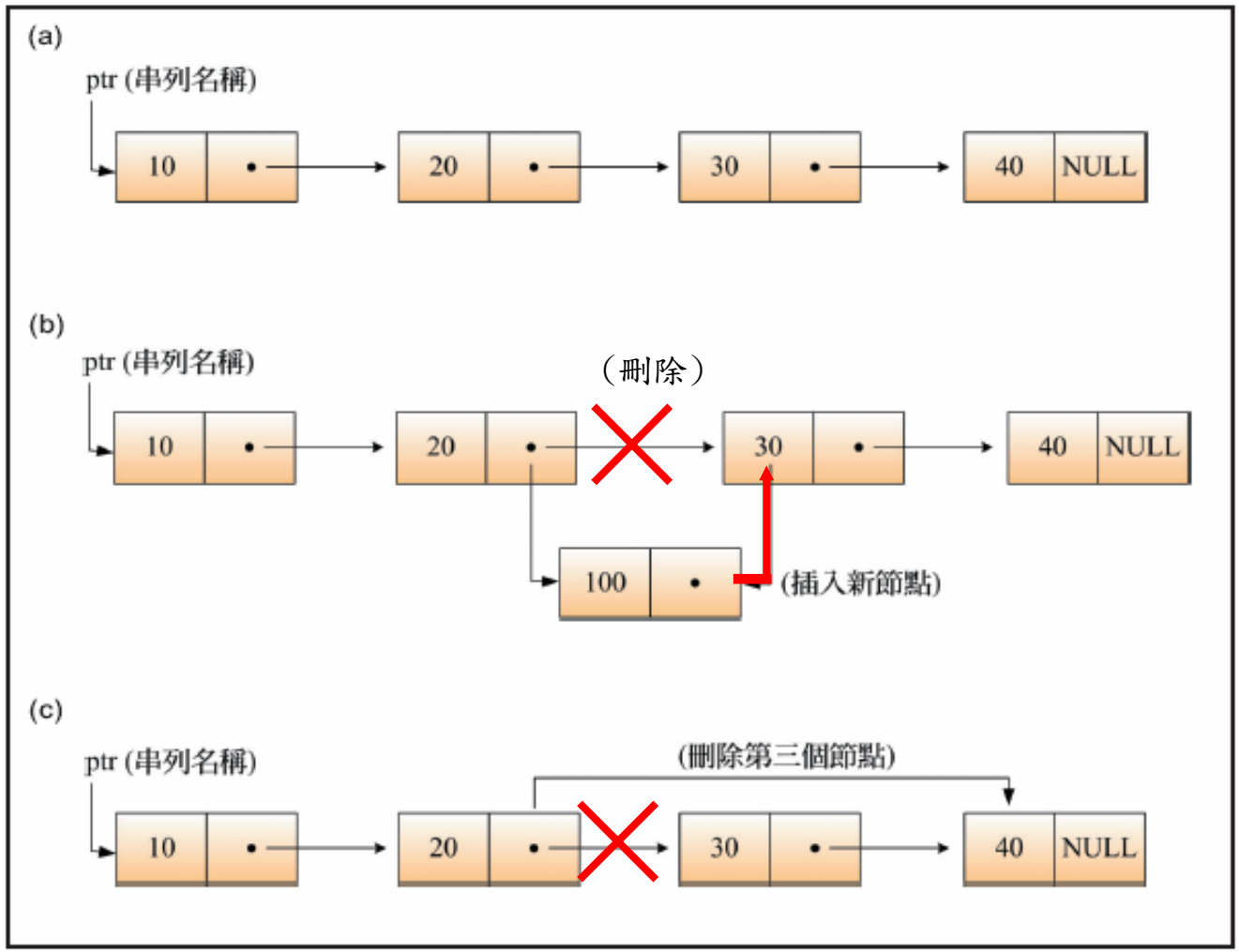


# 10-4 鏈結串列

- 鏈結串列有單向與雙向之分。
- 單向鏈結串列中，每個節點(node)有兩個欄位，分別存放資料(data)與右(或左)指標(pointer)。
- 雙向鏈結串列中，每個節點(node)有三個欄位，分別存放資料(data)與左右指標(pointer)。



# 10-4 鏈結串列




10.10
(a) 單向鏈結串列
(b) 插入新節點
(c) 刪除節點

# 鏈結串列存放佇列與堆疊

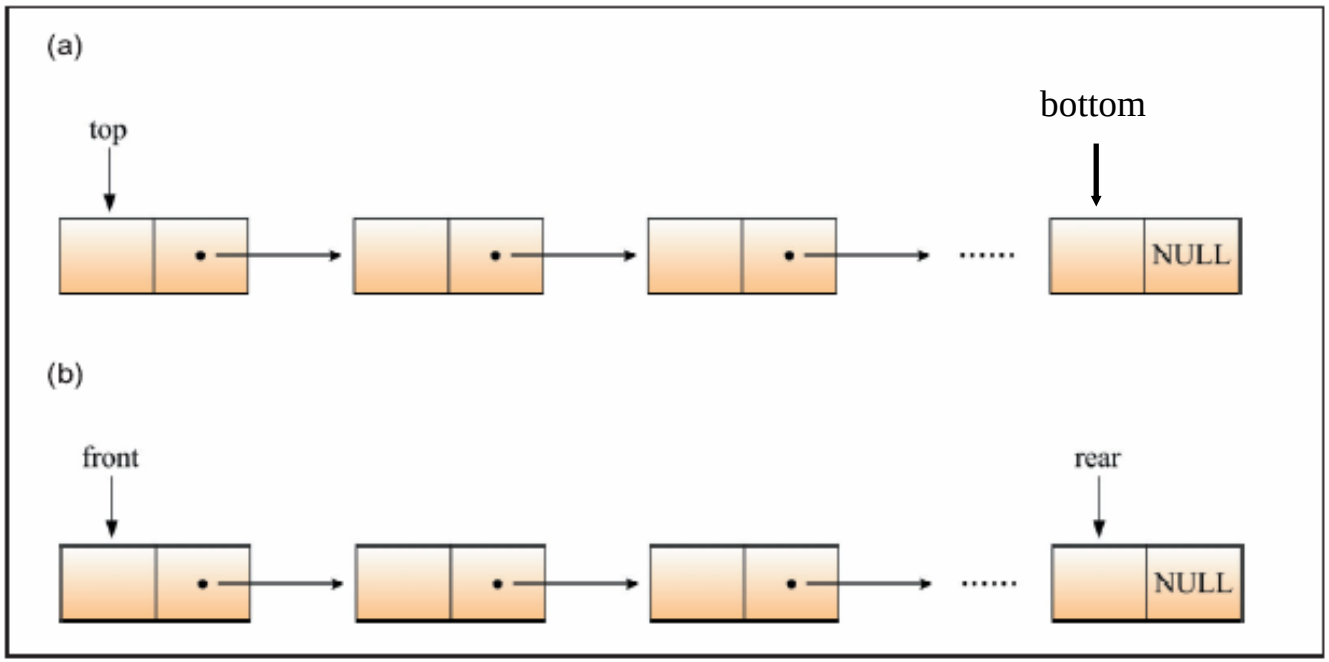



圖 10.13
(a) 以鏈結串列存放堆疊
(b) 以鏈結串列存放佇列

# 鏈結串列存放陣列

以$8X^4-6X^2+3X^5+5$為例

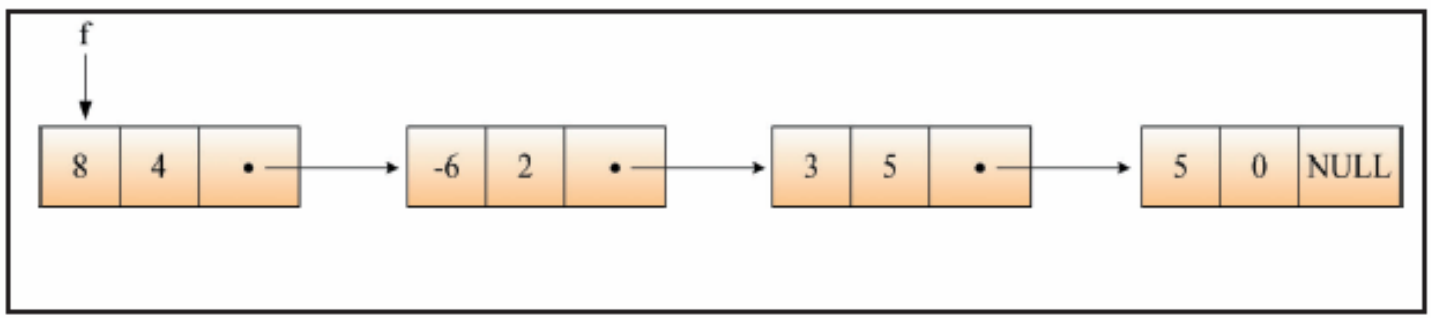
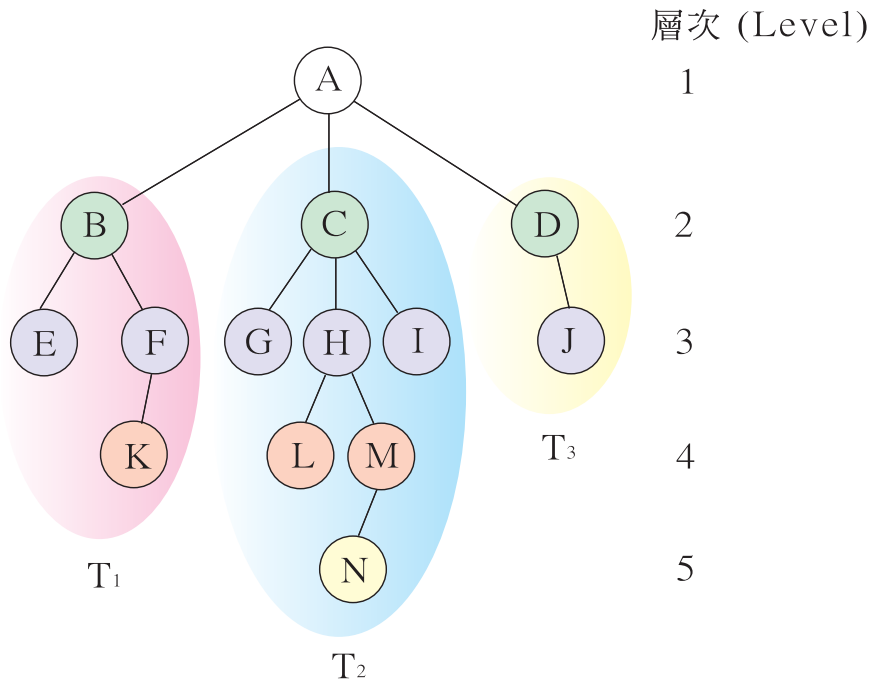



圖 10.14 以鏈結串列存放多項式

# 10-5 樹

- 樹是由一個或多個節點所組成的有限集合
- 祖先 (ancestor)或樹根(root)
- 子孫 (descendant)
- 父親 (parent)
- 孩子 (child)
- 兄弟 (sibling)
- 終端節點 (terminal node)
  或樹葉例如：E,K,G,L,N,I,J
- 內部節點 (internal node)
  不是樹根與樹葉的節點




- 階級 (degree) ：該節點有幾位子孫(子樹)，例如B的階級為2
- 層次 (level)
- 高度 (height) 或深度 (depth) ：最大的層次，本例高度為5

# 10-5-1　二元樹

對高度為h的二元樹來說，全部存滿的話，總共有$2^0 + 2^1 + \cdots + 2^{h-1} = 2^h - 1$個節點。

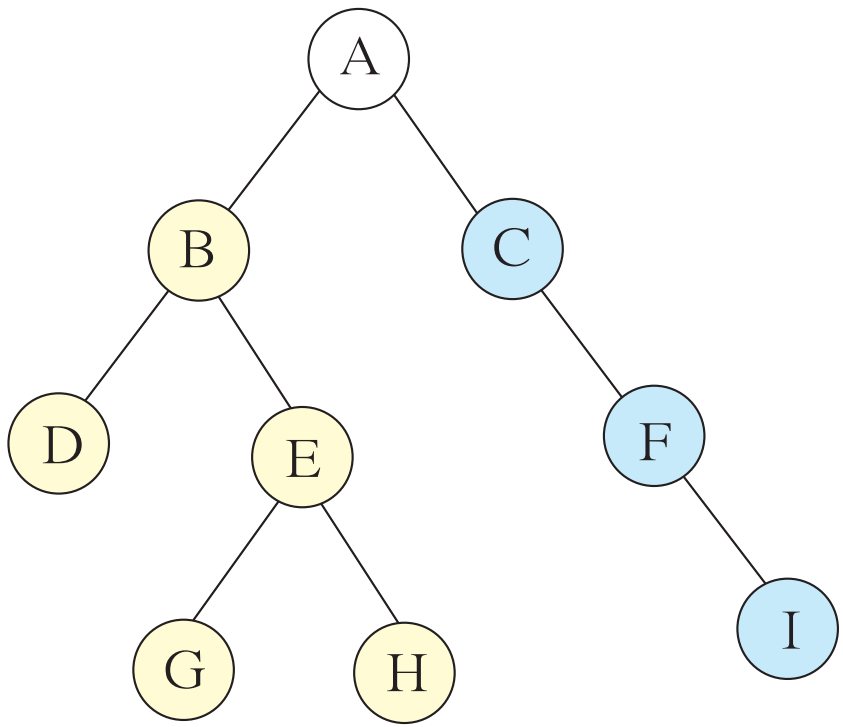



圖10.15(b)

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

圖 10.16 以陣列存放圖 10.15(b) 的二元樹

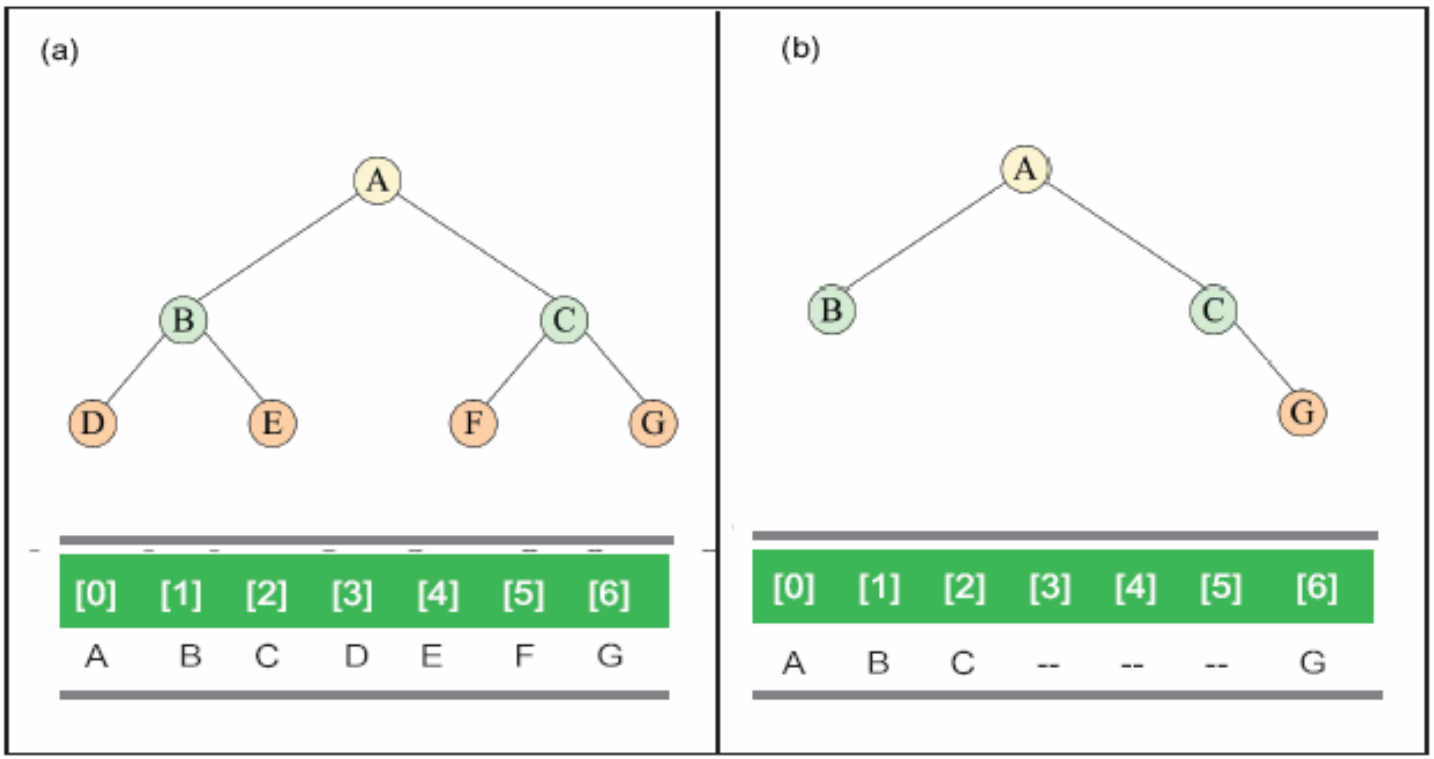



圖 10.17 (a) 左右平衡的二元樹較不浪費記憶體 (b) 左右不平衡的二元樹較浪費記憶體


- 以雙向鏈結串列存放二元樹(以圖10.15(b)為例)

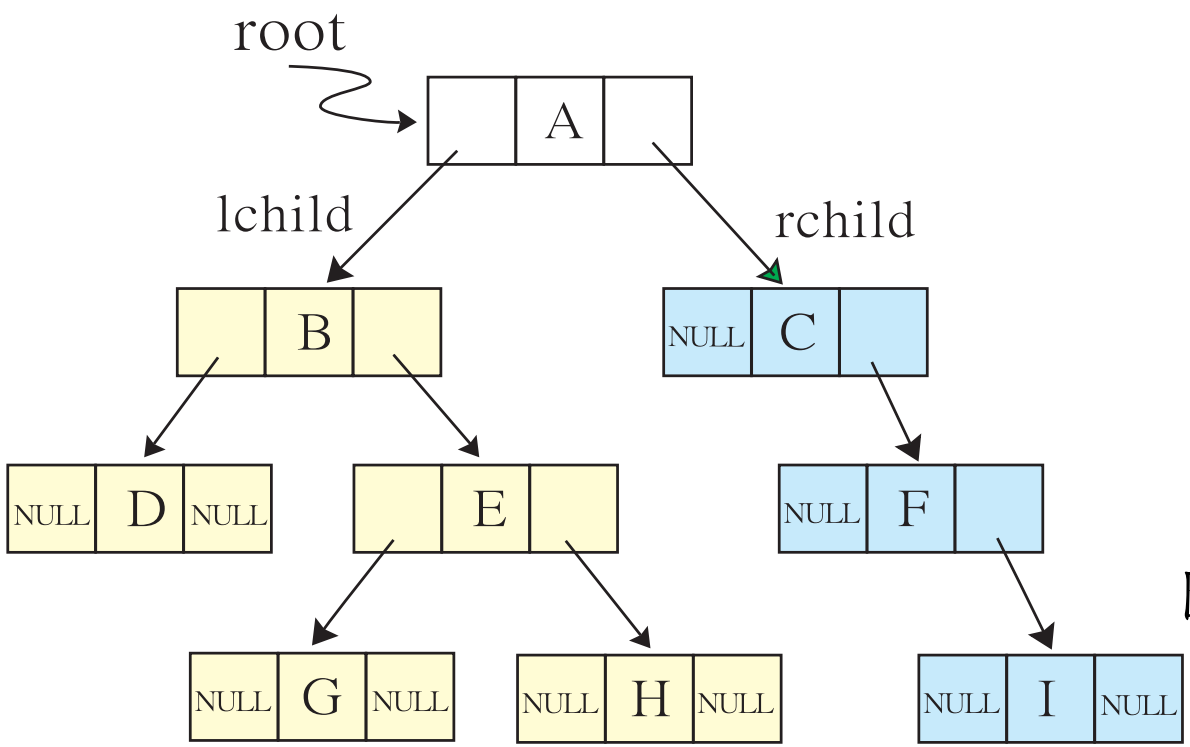
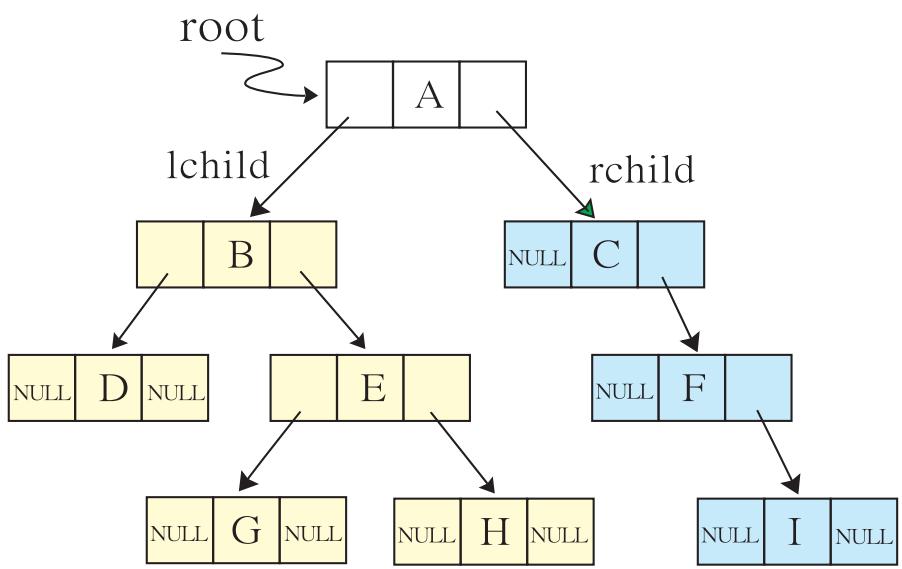



圖10.15(b)

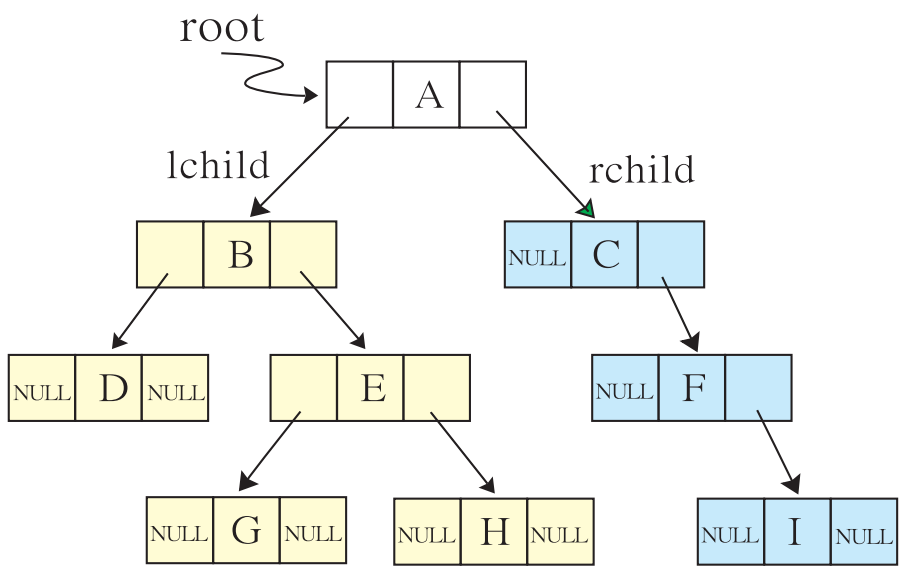
# 中序追蹤

- 窮盡左子樹，然後樹根，最後窮盡右子樹，不斷重複，直到所有節點都被拜訪後，工作才結束。
- 下圖的中序追蹤，其結果為DBGEHACFI。



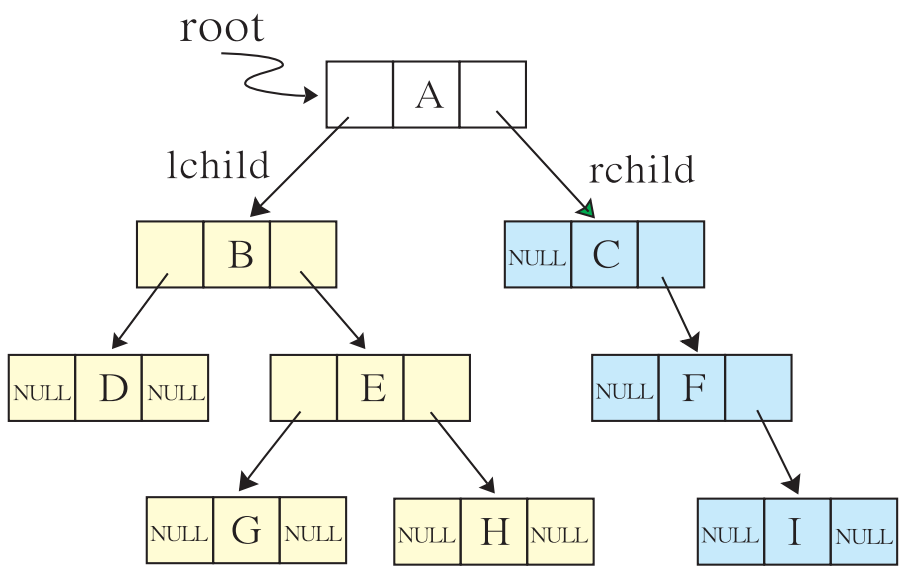
# 前序追蹤

- 先樹根，窮盡左子樹，最後窮盡右子樹，不斷重複，直到所有節點都被拜訪後，工作才結束。
- 下圖前序追蹤，其結果為ABDEGHCFI。



# 後序追蹤

- 窮盡左子樹，窮盡右子樹，最後樹根，不斷重複，直到所有節點都被拜訪後，工作才結束。
- 下圖後序追蹤，其結果為DGHEBIFCA。



# 10-5-2　二元搜尋樹

它必須滿足下列條件：

- 每個節點包含唯一的鍵（key）。
- 左右子樹亦為二元搜尋樹。
- 二元搜尋樹提供新增節點、刪除節點、追蹤等功能。
- 左子樹的鍵必須小於其樹根的鍵，右子樹的鍵必須大於其樹根的鍵。

舉例來說，圖 10.20 是將數字串列 25 、 30 、 24 、 58 、 45 、 26 、 12 、 14 建構為二元搜尋樹的過程。

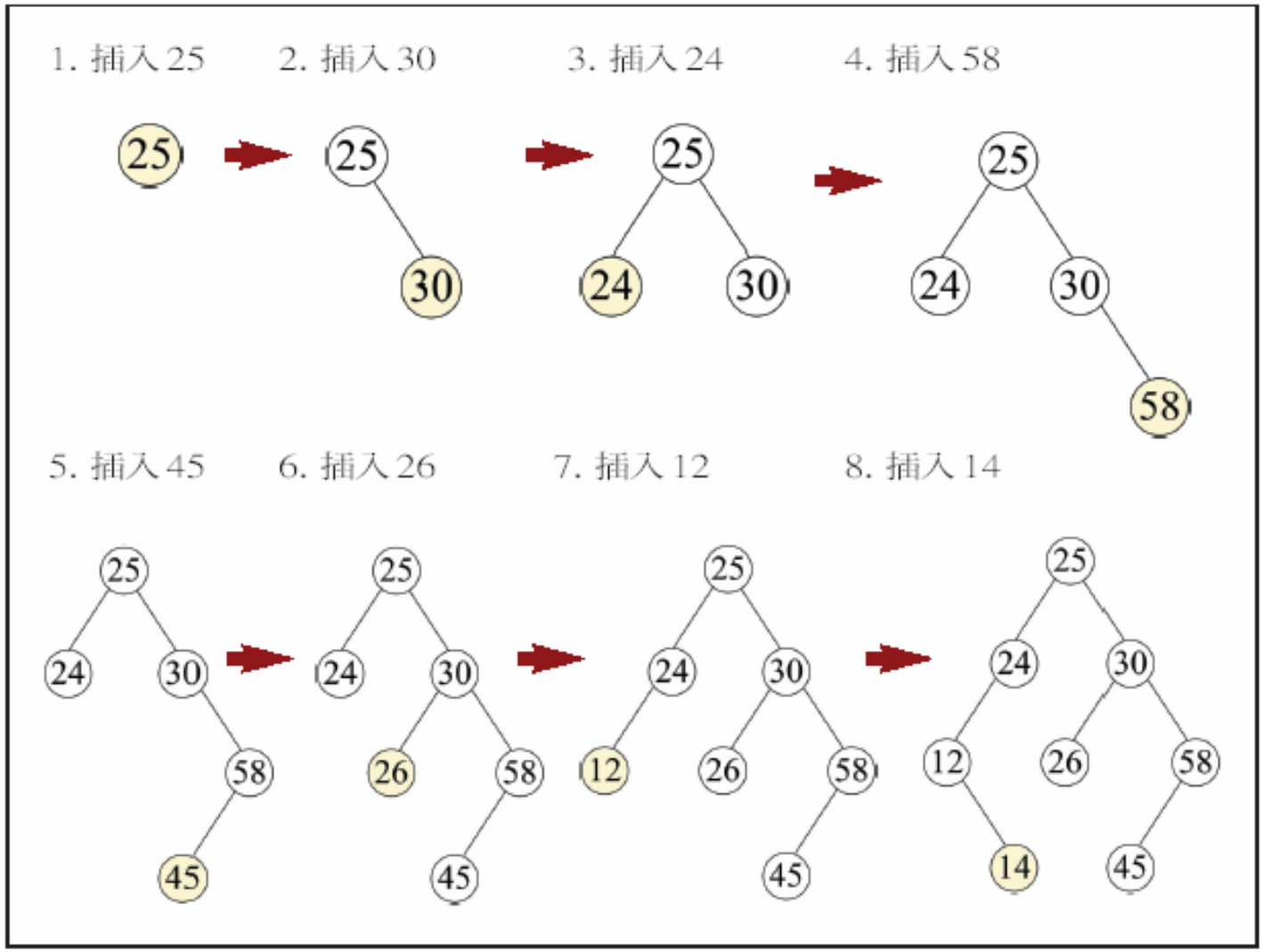



圖 10.20
建構二元搜尋樹

- 若為樹葉節點，直接刪除即可。
- 若為其他節點，除刪除外，以該節點的左子樹最大值，或右子樹最小值填入其位置，如下圖節點25刪除，以節點24取代原點

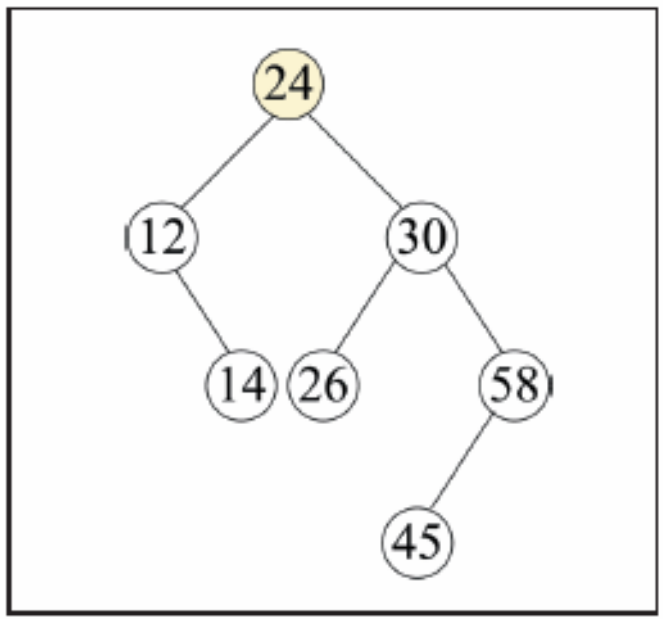



圖 10.21
刪除節點 25 的結果

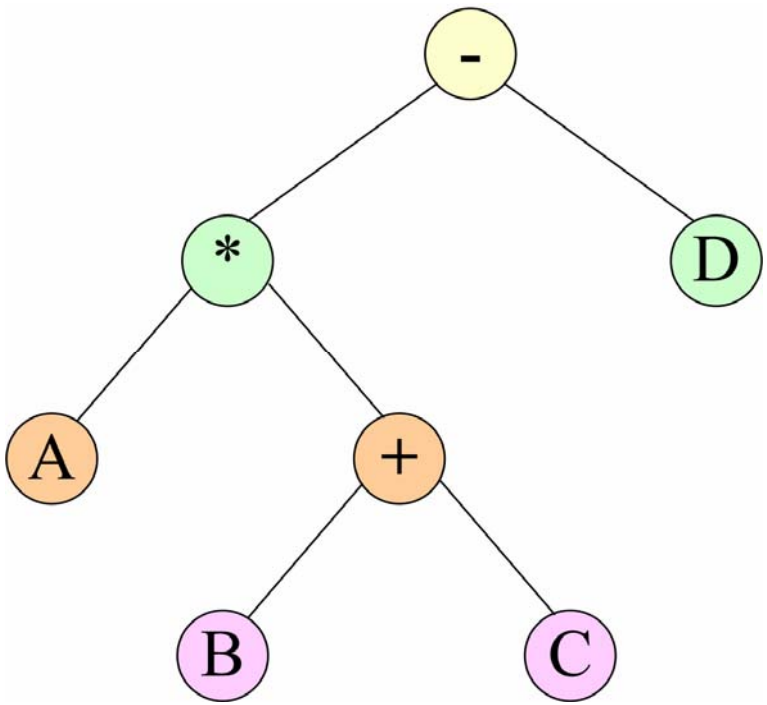
# 運算式樹

必須滿足下列條件：

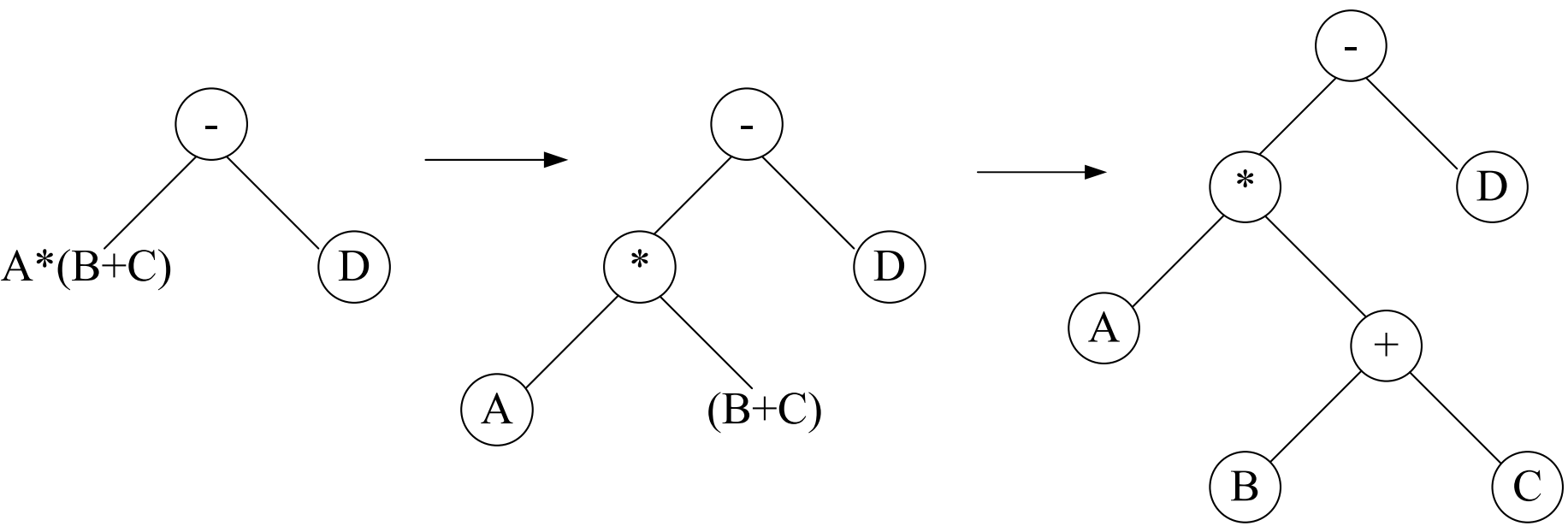
- 樹葉節點為運算元，而樹根與非樹葉節點為運算子。
- 子樹為子運算式且其樹根為運算子。

# 例題

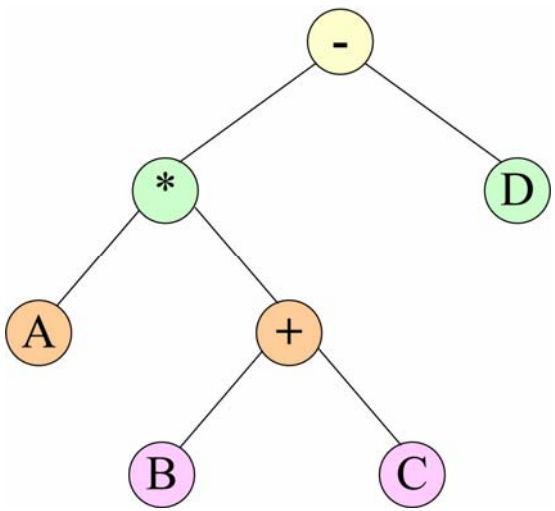
- A * (B + C) - D 的運算式樹



- 方式：由右至左



- 運算式樹的中序追蹤、前序追蹤與後序追蹤結果分別為運算式的中序表示法、前序表示法及後序表示法。




運算式樹的中序追蹤:
A*(B+C)-D

運算式樹的前序追蹤:

-*A+BCD

運算式樹的後序追蹤:
ABC+*D-

# 10-5-3 堆積

- 完全二元樹（full binary tree):全部存滿的二元樹。
- 完整二元樹（complete binary tree)：節點次序與編號完整。

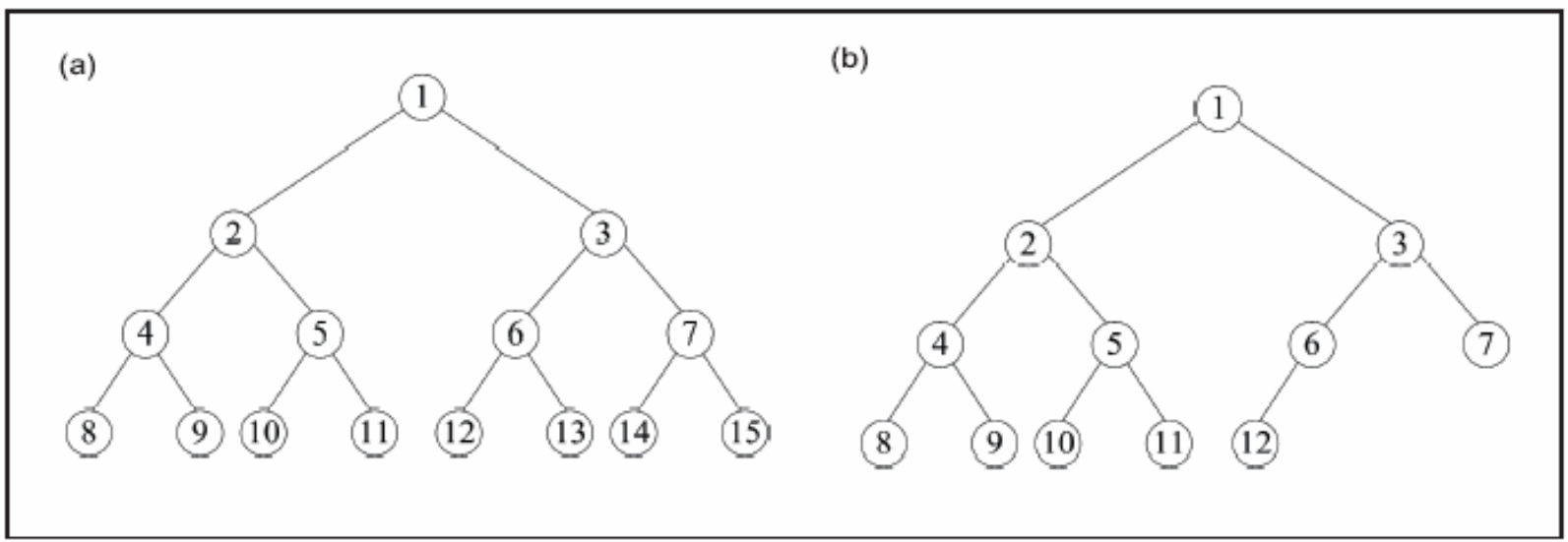



圖 10.22
(a) 完全二元樹 (b) 完整二元樹

堆積有下列兩種：

- 最大堆積（max heap):每節點不小於子節點。
- 最小堆積（min heap）：每節點不大於子節點。

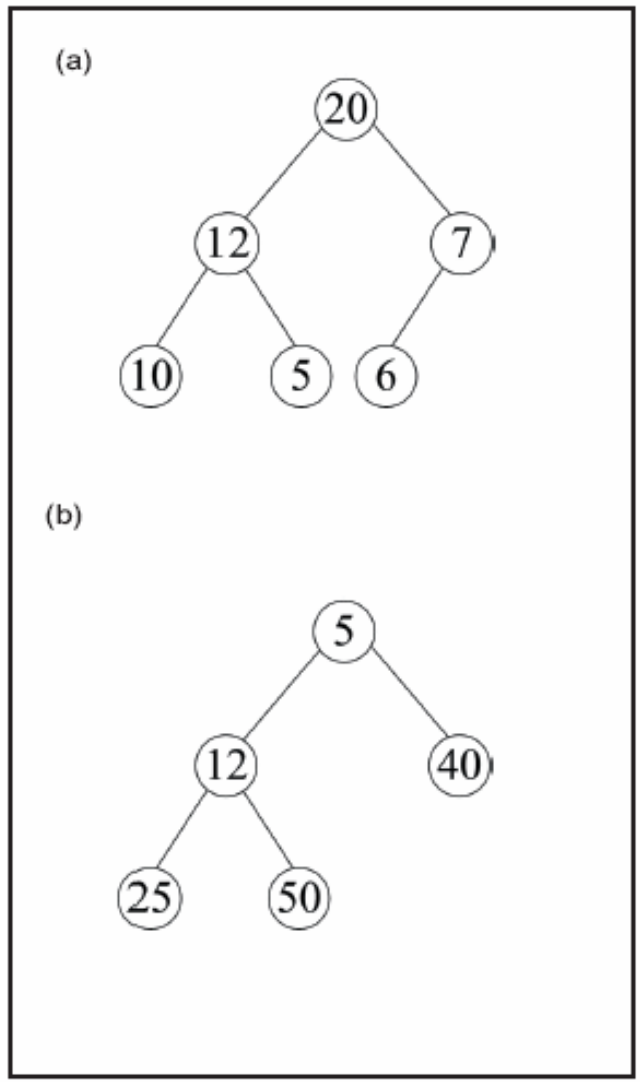
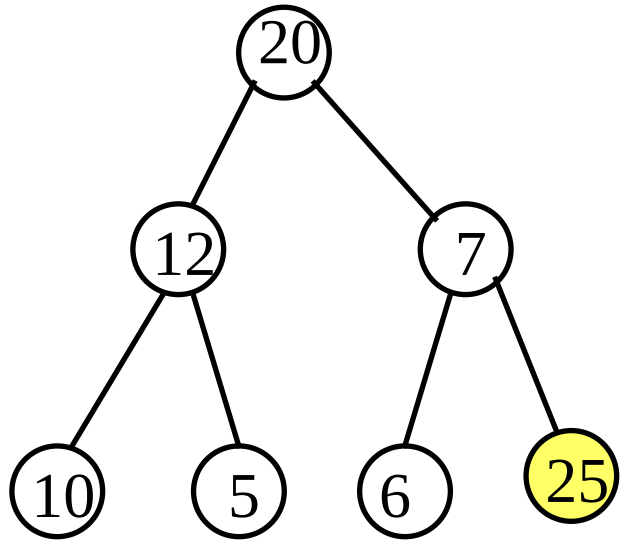


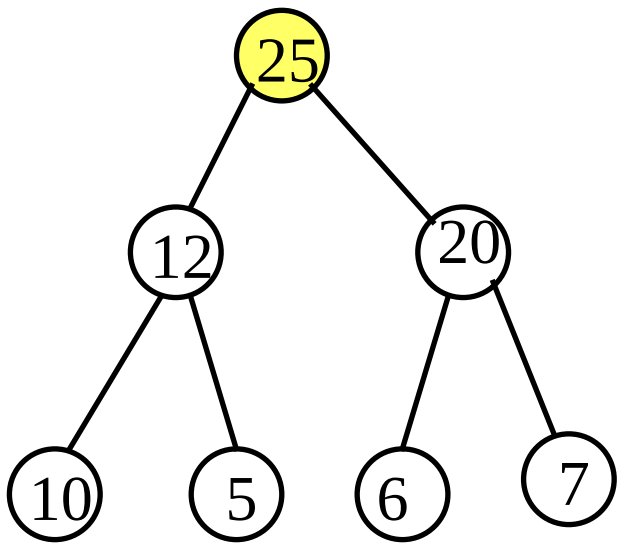

圖 10.23
(a) 最大堆積 (b) 最小堆積

# 最大堆積插入新節點

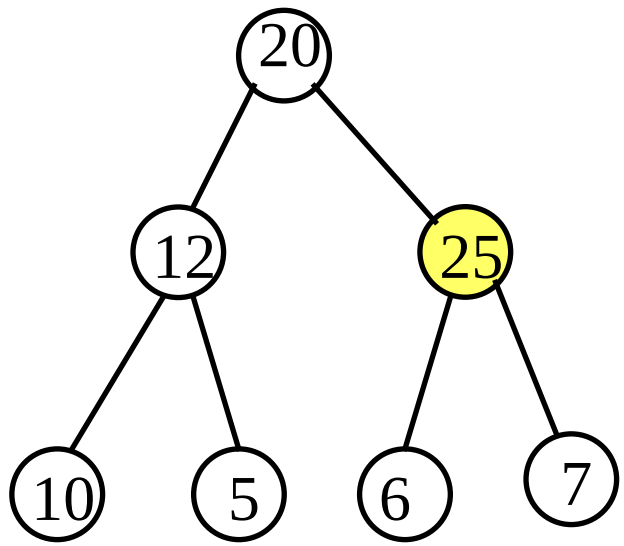
- 插入一空節點存放25
- 25比父節點7大，兩者交換。
- 25比父節點20大，兩者交換。




(a)



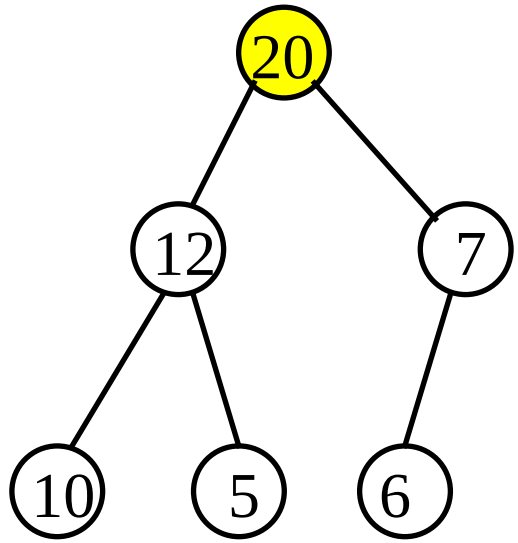

(c)



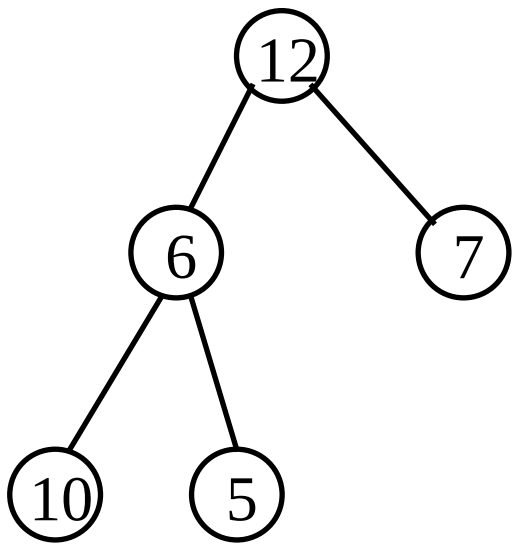

(b)

# 最大堆積刪除節點

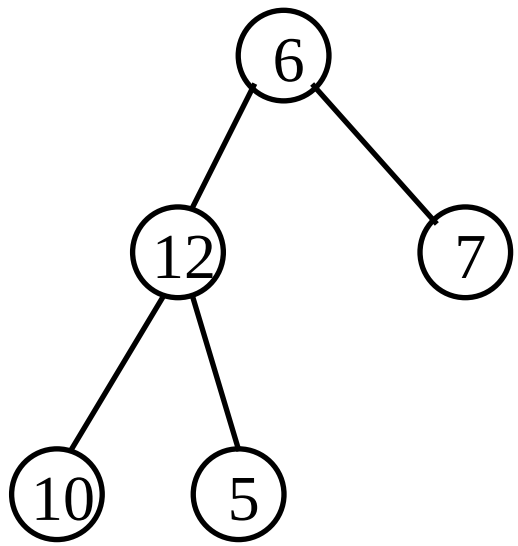
- 刪除節點一定從樹根開始。
- 最後一節點移到樹根。
- 6比子節點12小，兩者交換。




(a)



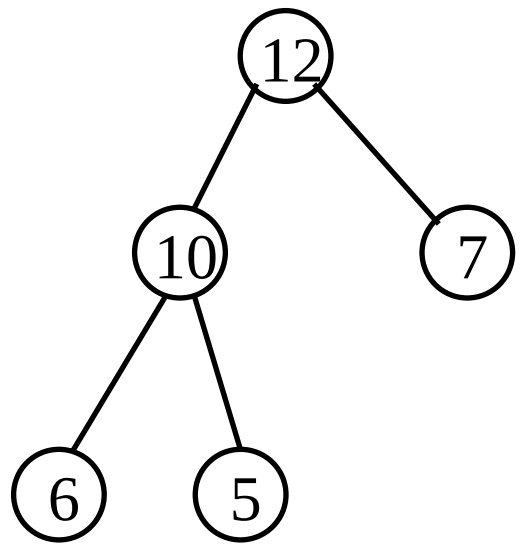

(c)




(b)

# 最大堆積刪除節點

- 6比子節點10小，兩者交換。
- 堆積可用於製作優先佇列，每個元素有不同優先權，若要取出最優權的最大元素，可用最大堆積。
- 反之，則用最小堆積。




(d)