

關於本書

資料結構 (data structure) 是資訊科學領域中的基礎課程，多數資訊相關科系的研究所入學考試會將之列入考科，而諸如高考三級、地方特考、司法特考、鐵路考試中的資訊類別考試亦會將之列入考科，因此，選擇一本**觀念正確、條理清晰且掌握命題趨勢**的書籍是很重要的。

本書以實例演練為導向，漸進式的學習架構，詳盡解說各種資料結構，期能帶領讀者了解其中的精髓，進而具備開發大型程式的功力。此外，為了因應未來報考資訊相關科系的研究所或準備國家考試，本書精選豐富考題，並融入相關章節、隨堂練習與學習評量，建議讀者勤加練習。

接著來談談為何要在一本資料結構書籍中介紹 ChatGPT 吧！在 ChatGPT 橫空出世後，有不少人驚覺「寫程式」即將被 AI 工具取代，沒錯，AI 工具確實會寫程式，但這並不表示以後再也不用學習資料結構、程式設計等基礎課程，而是程式設計師必須要進化為 AI 工具的程式審查員或教 AI 學習的老師。

換句話說，您必須具備資料結構、程式設計等關鍵能力，才能有效率地跟 AI 工具溝通，讓它寫出您需要的程式，也才有辦法閱讀或審查 AI 工具所生成的程式，確保程式是正確的、有效率的、經過完整測試的，而想要具備這些能力，您所需要的正是一本好書。

我們把如何使用 ChatGPT 輔助學習資料結構的內容放在最後一章，目的是希望您以學會資料結構為主，使用 ChatGPT 為輔，與其將 ChatGPT 的操作技巧分散到各個章節，倒不如集中在一個章節，比較能夠有系統地學習，也不會干擾到上課的節奏。

本書內容

本書共有 11 章，以實例演練為導向，搭配題型多元的隨堂練習與學習評量。首先從第 1 章「**導論**」開始，舉例說明資料結構、演算法與程式的效能分析。

接著是第 2 章「**陣列**」、第 3 章「**鏈結串列**」、第 4 章「**堆疊**」、第 5 章「**佇列**」，逐一探討陣列 (array)、鏈結串列 (linked list)、堆疊 (stack)、佇列 (queue) 等基礎資料結構，包括其實作方式、相關的運算與應用。

繼續是第 6 章「樹狀結構」，內容涵蓋樹 (tree) 的定義與應用、二元樹、二元搜尋樹、運算式樹、霍夫曼樹、樹林、集合等；第 7 章「圖形」，內容涵蓋圖形 (graph) 的定義與應用、最小成本擴張樹、最短路徑、拓樸排序等。

在認識資料結構後，接著是第 8 章「排序」和第 9 章「搜尋」，介紹排序 (sort) 與搜尋 (search) 兩種常見的運算，前者包括選擇排序、插入排序、氣泡排序、謝耳排序、快速排序、合併排序、基數排序、二元樹排序、堆積排序等，而後者包括循序搜尋、二元搜尋、內插搜尋、雜湊法等；再來是第 10 章「樹狀搜尋結構」，透過實例解說 AVL 樹、2-3 樹、2-3-4 樹、m 元搜尋樹、B 樹等樹狀搜尋結構的操作方式，以及彼此之間的異同。

最後是第 11 章「資料結構×ChatGPT」，介紹使用 ChatGPT 輔助學習資料結構，例如查詢理論與實作、上傳圖片解題、出題練習、撰寫程式、修正、優化與除錯、與其它程式語言互相轉換等，這些技巧大多不限定於 ChatGPT，也可以靈活運用在 Copilot、Gemini 等 AI 助理。本書是以 C 語言進行實作，讀者可以自行使用 ChatGPT 轉換成 Python、C++、C#、Java 等程式語言。

線上下載

本書的線上下載 (<http://books.gotop.com.tw/download/AEE041100>) 提供「範例程式」和「考題觀摩」，前者是以 Visual Studio Code 做為開發環境，而後者是一些研究所入學考試與國家考試的精選考題。

聯絡方式

如果讀者有建議或授課老師需要教學投影片，歡迎與我們洽詢：基峰資訊網站 <https://www.gotop.com.tw/>；國內學校業務處電話 (02)2788-2408。

參考書目

- ◀ Fundamentals of Data Structures in C/Horowitz, Sahni, Anderson-Freed
- ◀ Data Structures and Algorithms/Alfred Aho
- ◀ Algorithms/Robert Sedgewick, Kevin Wayne

不過，遞迴演算法不一定適合所有能夠以個個擊破方式來解決的問題，先決條件是分割後的小問題其特質與解法必須和原來的大問題相同才行，典型的例子有 $n!$ (n 階乘)、費伯納西數列、兩個自然數的最大公因數 (GCD) 等。

範例 1.8 [$n!$ (n 階乘)] 使用遞迴的方式計算 $n!$ ，其公式定義如下：

當 $n = 0$ 時， $f(n) = n! = 0! = 1$
當 $n > 0$ 時， $f(n) = n! = n \times f(n - 1)$

`\Ch01\factorial2.c`

```
int factorial(int n)
{
    if (n == 0) return 1;
    else return (n * factorial(n - 1));
}
```

這個 `factorial()` 函數屬於遞迴函數 (recursive function)，也就是呼叫自己本身的函數。任何遞迴函數都必須有終止條件，一旦終止條件成立，就會計算結果，不再呼叫自己本身，例如 `factorial(n)` 的終止條件為 n 等於 0。此外，遞迴函數的呼叫方式是一層層的巢狀結構，等最內層的終止條件成立時，再回到上一層執行，而等上一層結束執行後，再回到上上一層執行，直到回到最上層。下面是以 $n = 3$ 為例，模擬 `factorial(3)` 的執行過程 (圖 1.7)：

- ◀ $n = 3$ ，執行到 $3 * \text{factorial}(2)$ 時，呼叫 `factorial()` 計算 $n = 2$ 的結果。
- ◀ $n = 2$ ，執行到 $2 * \text{factorial}(1)$ 時，呼叫 `factorial()` 計算 $n = 1$ 的結果。
- ◀ $n = 1$ ，執行到 $1 * \text{factorial}(0)$ 時，呼叫 `factorial()` 計算 $n = 0$ 的結果。
- ◀ $n = 0$ ，遞迴函數的終止條件成立，直接傳回 1，得到 `factorial(0)` 為 1。
- ◀ 回到 $n = 1$ ，計算 $1 * \text{factorial}(0)$ 的結果為 $1 * 1 = 1$ ，得到 `factorial(1)` 為 1。
- ◀ 回到 $n = 2$ ，計算 $2 * \text{factorial}(1)$ 的結果為 $2 * 1 = 2$ ，得到 `factorial(2)` 為 2。
- ◀ 回到 $n = 3$ ，計算 $3 * \text{factorial}(2)$ 的結果為 $3 * 2 = 6$ ，得到 `factorial(3)` 為 6。

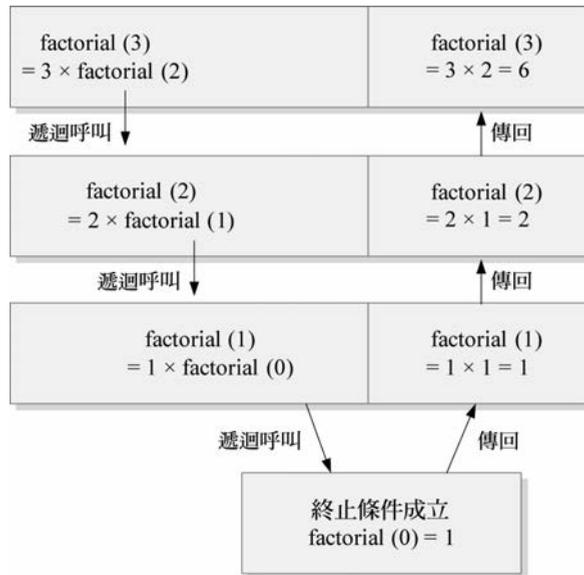


圖 1.7 以 $n = 3$ 為例，模擬 $\text{factorial}(3)$ 的執行過程

除了呼叫自己的函數，若函數 $f1()$ 呼叫函數 $f2()$ ，而函數 $f2()$ 又在某種情況下呼叫函數 $f1()$ ，那麼函數 $f1()$ 也可以算是一個遞迴函數。遞迴函數通常可以被諸如 **for**、**while** 等迴圈所取代，雖然遞迴函數的效率比不上迴圈，因為多了額外的函數呼叫，但是遞迴函數的邏輯性、可讀性及彈性均比迴圈好，所以在很多時候，尤其是要撰寫遞迴演算法，遞迴函數還是會被用來取代迴圈。

比方說，在 $n!$ (n 階乘) 的例子中，您或許會覺得使用迭代的方式比遞迴的方式來得直覺，這主要是因為人類筆算與電腦計算的差異，當我們以紙筆計算遞迴函數時，的確會比較繁複，但是對電腦來說，就沒有這樣的困擾。最後，我們簡單歸納出遞迴演算法的幾個條件：

- ◀ 遞迴函數的輸入和輸出必須定義清楚。
- ◀ 遞迴函數必須包含正確的指令並正確的呼叫自己。
- ◀ 遞迴函數的呼叫次數必須有限（即終止條件一定會在某個時候成立）。
- ◀ 當終止條件成立時，遞迴函數必須正確的計算結果。

6-2 二元樹

二元樹 (binary tree) 是每個節點最多有兩個子節點的樹，節點的左邊稱為左子樹 (left child)，節點的右邊稱為右子樹 (right child)，圖 6.4 即為一例。

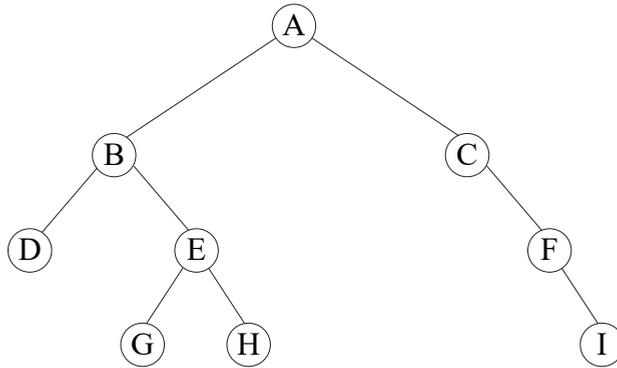


圖 6.4 二元樹

二元樹和樹的差別如下：

- ◀ 二元樹可以是空集合，而樹必須至少有一個樹根，不可以是空集合。
- ◀ 二元樹的節點分支度為 $0 \leq d \leq 2$ ，而樹的非終端節點分支度為 $d \neq 0$ 。
- ◀ 二元樹會以左右子樹或左右節點來區分順序，而樹無順序之分。

以圖 6.5(a)、(b) 為例，這雖然是兩棵相同的樹，卻是兩棵不同的二元樹，因為二元樹會以左右子樹或左右節點來區分順序。

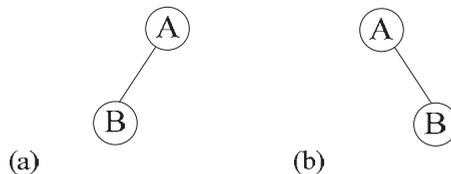


圖 6.5 兩棵不同的二元樹

範例 6.2 證明二元樹第 i 階的最多節點個數為 2^{i-1} ， $i \geq 1$ 。

解答：我們使用數學歸納法來加以證明：

1. 當 $i = 1$ 時，第 1 階最多只有樹根一個節點，故 $2^{i-1} = 2^{1-1} = 2^0 = 1$ 成立。
2. 假設當 $1 \leq i < k$ 時，第 i 階的最多節點個數為 2^{i-1} 成立，故第 $k - 1$ 階的最多節點個數為 2^{k-2} 。
3. 證明當 $i = k$ 時，第 i 階的最多節點個數為 2^{i-1} 亦成立。由於第 $k - 1$ 階的最多節點個數為 2^{k-2} ，而二元樹的每個節點最多有兩個子節點，故第 k 階的最多節點個數為 $2^{k-2} \times 2 = 2^{k-1}$ 。

範例 6.3 證明高度為 h 之二元樹的最多節點個數為 $2^h - 1$ ， $h \geq 1$ 。

解答：由於二元樹第 i 階的最多節點個數為 2^{i-1} ，因此，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。

例如圖 6.6 是一個高度為 4 且全部存滿的二元樹，它的節點個數為 $2^4 - 1 = 15$ ，編號為 0 ~ 14，而且第 1、2、3、4 階的節點個數分別為 2^{1-1} 、 2^{2-1} 、 2^{3-1} 、 2^{4-1} 。事實上，這種全部存滿的二元樹就叫做**完滿二元樹** (full binary tree)。

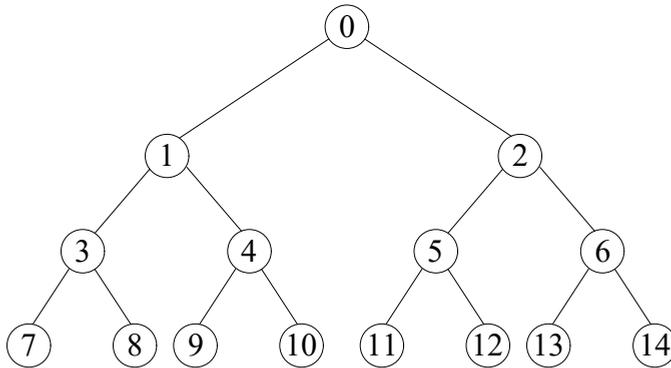
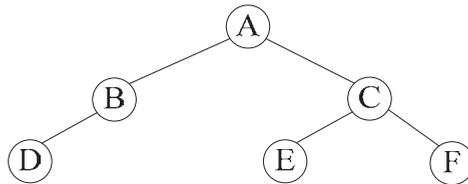


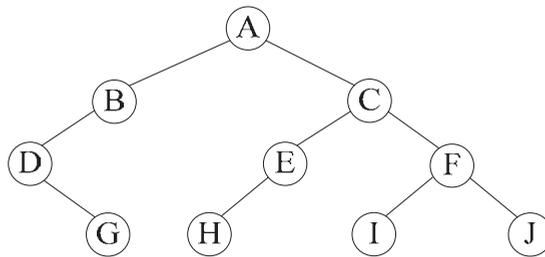
圖 6.6 高度為 4 的完滿二元樹

二、練習題

1. 簡單說明何謂樹？請您想想看，生活中有哪些例子屬於樹的應用呢？
2. 以鏈結串列表示下列二元樹。



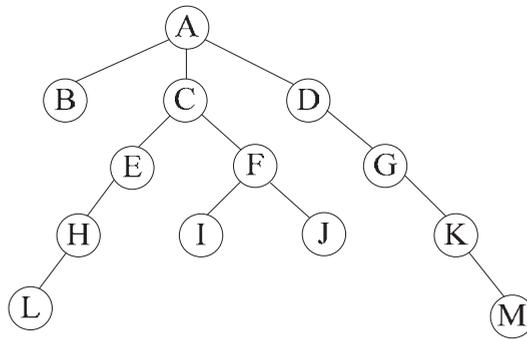
3. 我們可以從前序走訪結果或後序走訪結果決定唯一的二元樹嗎？
4. 寫出下列二元樹的中序、前序與後序走訪結果。



5. 假設運算式的中序表示法為 $a / b ** c * d - e$ ，試將它建構為運算式樹。
6. 假設二元樹的中序走訪結果為 AIBHCGDFE，後序走訪結果為 ABICHGDGEF，試問，其前序走訪結果為何？
7. 假設數字串列為 (14, 15, 5, 9, 8, 19, 2, 6, 16, 4, 20, 17, 10, 13, 7)，請回答下列問題：
 - (1) 將該數字串列建構為二元搜尋樹。
 - (2) 寫出該二元搜尋樹的後序走訪結果。
 - (3) 寫出該二元搜尋樹包含幾個樹葉與高度。
8. 假設二元樹的前序走訪結果為 ABCDE，中序走訪結果為 CBDAE，試據此推算出該二元樹。

9. 針對下圖的樹回答問題：

- (1) 寫出該樹的高度。
- (2) 寫出節點 B、H 的階度。
- (3) 寫出該樹的終端節點。
- (4) 寫出節點 A、K 的分支度。
- (5) 寫出節點 C 的兄弟與子節點。
- (6) 寫出節點 E 的祖先與子孫。



10. 假設二元樹的高度為 h ，試問，它最多包含幾個節點？
11. 假設二元樹的後序走訪結果為 **HAIECJDKFBG**，中序走訪結果為 **HACEIGJDBKF**，試據此推算出該二元樹。
12. 假設運算式的中序表示法為 $(A - B) / (C * D + E)$ ，試將它建構為運算式樹。
13. 簡單說明何謂二元樹？它和一般樹有何不同？
14. 簡單說明何謂二元搜尋樹？順序不同的資料仍會建構出相同的二元搜尋樹，對不對？哪種資料所建構出來的二元搜尋樹將為傾斜樹？
15. 根據下列字母的出現次數建構霍夫曼樹，並寫出霍夫曼碼。

字母	E	I	N	P	S	T
次數	29	5	7	12	4	8

7-1 認識圖形

圖形理論的運用最早可以追溯至 1736 年數學家尤拉 (Euler) 為了解決肯尼茲堡橋樑 (Koenigsberg Bridge) 問題所提出，這個問題是假設有 a、b、c、d、e、f、g 等七座橋樑連接 A、B、C、D 等四個區域，如圖 7.1，試問，是否存在一種走法可以從某個區域出發，然後經過每座橋各一次，再返回原先出發的區域呢？

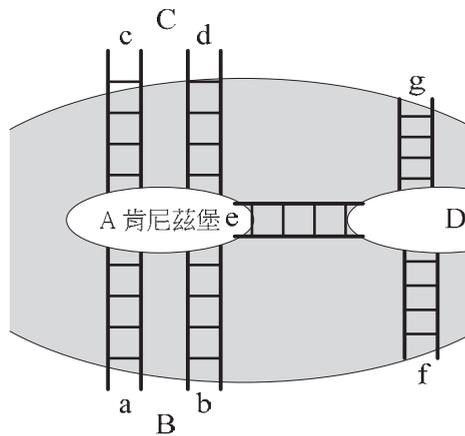


圖 7.1 Koenigsberg Bridge 問題 (又稱為七橋問題)

尤拉的答案是 No，因為他發現只有在圖形的每個頂點的分支度均為偶數時，才能從某個頂點出發，然後經過每個邊各一次，再返回原先出發的頂點，後人將這種走訪稱為尤拉迴路 (Eulerian walk、Eulerian cycle)。

頂點的分支度 (degree) 指的是有幾個邊連接到該頂點，我們可以將圖 7.1 描繪成如圖 7.2(a) 的圖形，顯然 A、B、C、D 等頂點的分支度均不為偶數，故尤拉迴路不存在。

若我們將圖 7.2(a) 改為圖 7.2(b)，令 A、B、C、D 等頂點的分支度均為偶數，就能找出尤拉迴路，例如 $A \rightarrow a \rightarrow B \rightarrow d \rightarrow D \rightarrow f \rightarrow B \rightarrow b \rightarrow A \rightarrow e \rightarrow D \rightarrow g \rightarrow C \rightarrow c \rightarrow A$ 。

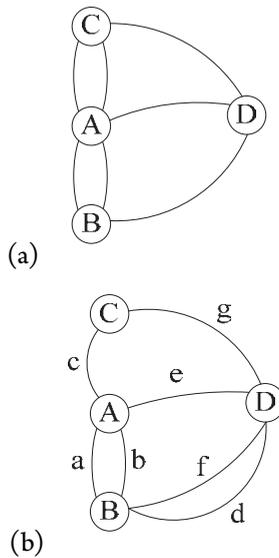


圖 7.2 (a) 頂點的分支度均不為偶數 (b) 頂點的分支度均為偶數

時至今日，圖形的應用已經相當廣泛，許多問題都能透過圖形來加以描述，然後找出問題的解答，常見的有找出兩點之間的最短路徑、電路分析、網路佈線、都市計畫分析、組織層次圖、交通路線圖等。舉例來說，我們可以將航空公司的航班路線表示成加權圖形，每個邊的權重是航班路線的距離或費用，然後由加權圖形求出兩個城市之間的最短航行路徑或最低航行費用。

7-1-1 圖形的定義

圖形 (graph) G 是由 $V(G)$ 和 $E(G)$ 所組成，其中 $V(G)$ 是一個有限且非空的集合，代表圖形的頂點 (vertex)， $E(G)$ 是一個有限的集合，代表圖形的邊 (edge)，我們通常寫成 $G = (V, E)$ 。

圖形又分成無向圖形 (undirected graph) 與有向圖形 (directed graph, digraph)，前者指的是沒有方向的圖形，而後者指的是有方向的圖形，例如圖 7.3(a) 的 G_1 是一個無向圖形， (A, B) 、 (B, A) 代表的都是連接頂點 A 和頂點 B 的邊，故 $E(G_1)$ 只要列出 (A, B) 即可。

範例 8.2 撰寫一個函數實作 [選擇排序] 並分析複雜度。

解答：首先，我們來討論 `selection_sort()` 函數的時間複雜度，無論是最佳情況、最差情況或平均情況，外部迴圈都會針對 $i = 0, 1, \dots, n - 2$ 重複執行，此時，內部迴圈的執行次數分別為 $n - 1, n - 2, \dots, 2, 1$ ，共 $n(n-1)/2$ 次，故時間複雜度為 $O(n^2)$ 。至於空間複雜度則為 $O(1)$ ，也就是需要一個額外的空間做為資料交換的緩衝區，即 `temp` 變數。

此外，選擇排序屬於不穩定排序，因為第 i 回合會將第 i 小的資料與第 i 個位置的資料交換，即使這兩個資料的大小相同，也會被交換。

\Ch08\select.c

```
#include <stdio.h>
#define N 8                                /*定義欲排序的資料個數為 8*/

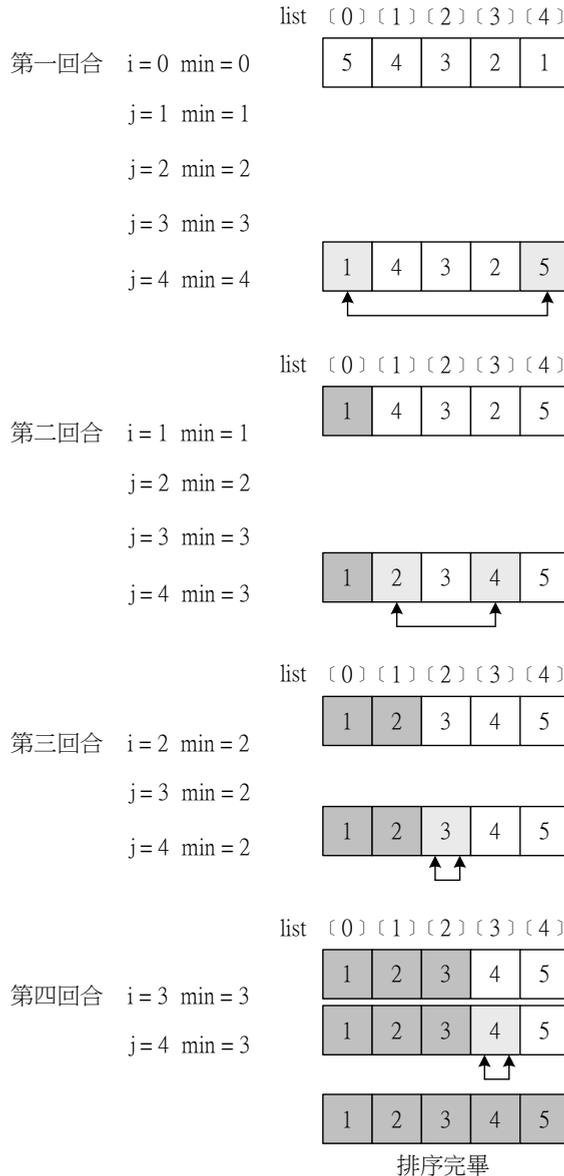
void selection_sort(int list[], int n)
{
    int i, j, min, temp;
    for (i = 0; i < n - 1; i++){
        min = i;
        for (j = i + 1; j < n; j++) /*在剩下的資料中找出最小的資料*/
            if (list[j] < list[min]) min = j;
        temp = list[min];          /*將第 i 小的資料與第 i 個位置的資料交換*/
        list[min] = list[i];
        list[i] = temp;
    }
}

int main()
{
    int list[N] = {8, 7, 6, 5, 4, 3, 2, 1};
    selection_sort(list, N);
    printf("排序結果為：");
    for (int i = 0; i < N; i++)
        printf("%d ", list[i]);
}
```

```
[Running] cd
"c:\Users\Jean\Documents\Samples\Ch08\" &&
gcc select.c -o select &&
"c:\Users\Jean\Documents\Samples\Ch08\"select
排序結果為：1 2 3 4 5 6 7 8
[Done] exited with code=0 in 1.739 seconds
```

範例 8.3 以紙筆模擬使用前述的 `selection_sort()` 函數，將 `list[] = {5, 4, 3, 2, 1}` 進行排序的過程。

解答：



10-1-1 LL 型

LL 型指的是將新節點 N 插入到節點 A 的左兒子的左子樹，而節點 A 是離新節點 N 最近且平衡係數由 ± 1 變成 ± 2 的祖先節點。以圖 10.2(a) 為例，這是一棵 AVL 樹的子樹，節點 A 的平衡係數為 1，但在將新節點 N 插入到節點 A 的左兒子的左子樹後，節點 A 的平衡係數變成 2，如圖 10.2(b)，這樣將失去平衡，此時，我們可以做 LL 型旋轉，將節點 A 往順時針方向旋轉，成為節點 B 的右兒子，而節點 B 原本的右子樹 B_R 則移到節點 A 的左子樹，如圖 10.2(c)。

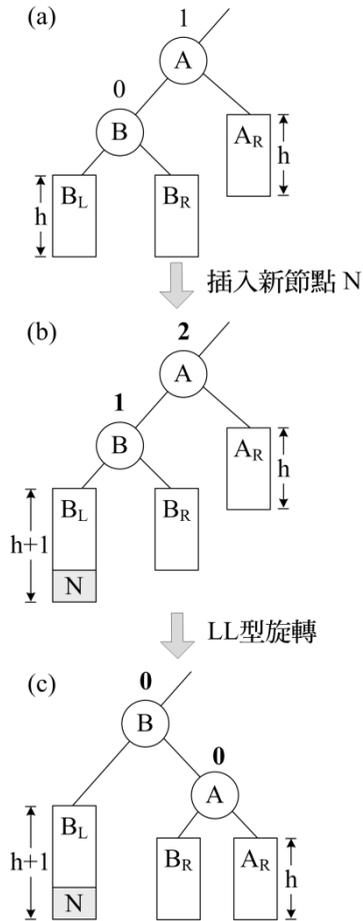
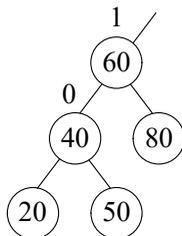
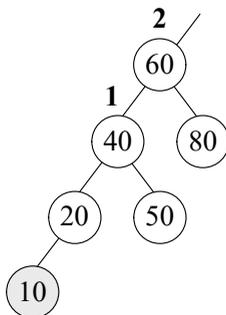


圖 10.2 (a) AVL 樹的子樹 (b) 失去平衡 (c) 重新調整為 AVL 樹

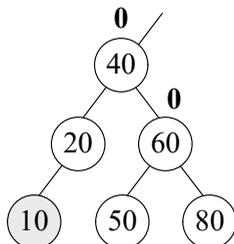
範例 10.1 [LL 型] 下圖是一棵二元搜尋樹的子樹，同時亦為 AVL 樹，請在該子樹插入新節點 10，然後重新調整為 AVL 樹。



解答：首先，在該子樹插入新節點 10，同時要保持二元搜尋樹的形式，如下圖。



此時，該子樹已經失去平衡，不再是 AVL 樹，由於新節點 10 是插入到節點 60 的左兒子的左子樹，屬於 LL 型，因此，我們可以做 LL 型旋轉，將它重新調整為 AVL 樹，如下圖，得到的結果不僅為 AVL 樹，同時亦保持二元搜尋樹的形式。



11-3 上傳圖片解題

在上一節中，我們示範了如何以文字提問的方式，請 ChatGPT 回答資料結構的問題，其實它還有一個實用的功能，就是接受用戶上傳圖片，然後根據圖片作答，下面是一個例子。

1. 點取對話框左側的迴紋針圖示。



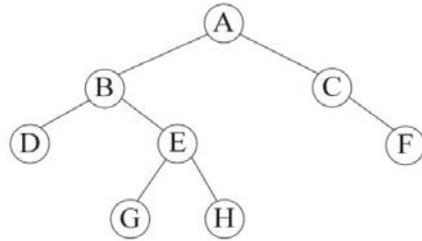
2. 選取要上傳的圖片，這是一棵二元樹，然後按 [開啟] (您可以在本書範例程式中找到此圖片 `tree.jpg` 做練習)。



3. 輸入提示詞，例如「寫出這棵二元樹的中序走訪結果」，然後按 。



4. 生成中序走訪的步驟與結果 (DBGEHACF)。同樣的，ChatGPT 不一定會一次就答對，如果答錯了，您可以告訴它是哪個步驟做錯了，請它進行修正，如果您也不是很確定錯在哪，可以請它重新作答。



寫出這棵二元樹的中序走訪結果

- 要寫出這棵二元樹的中序走訪結果，我們需要遵循中序遍歷的規則：先遍歷左子樹，然後訪問根節點，最後遍歷右子樹。

根據圖片中的二元樹結構，我們可以如下進行中序遍歷：

```

mathematica
graph TD
    A --- B
    A --- C
    B --- D
    B --- E
    E --- G
    E --- H
    C --- F
  
```

按照中序遍歷的步驟：

1. 從根節點 `A` 開始，先遍歷 `A` 的左子樹。
2. 到 `B`，再遍歷 `B` 的左子樹。
3. 到 `D`，因為 `D` 沒有左子樹，所以訪問 `D`，然後遍歷 `D` 的右子樹（沒有右子樹）。
4. 回到 `B`，訪問 `B`，然後遍歷 `B` 的右子樹。
5. 到 `E`，先遍歷 `E` 的左子樹。
6. 到 `G`，因為 `G` 沒有左子樹，所以訪問 `G`，然後遍歷 `G` 的右子樹（沒有右子樹）。
7. 回到 `E`，訪問 `E`，然後遍歷 `E` 的右子樹。
8. 到 `H`，因為 `H` 沒有左子樹，所以訪問 `H`，然後遍歷 `H` 的右子樹（沒有右子樹）。
9. 回到 `A`，訪問 `A`，然後遍歷 `A` 的右子樹。
10. 到 `C`，因為 `C` 沒有左子樹，所以訪問 `C`，然後遍歷 `C` 的右子樹。
11. 到 `F`，因為 `F` 沒有左子樹，所以訪問 `F`，然後遍歷 `F` 的右子樹（沒有右子樹）。

最終的中序遍歷結果是：

```

mathematica
DBGEHACF
  
```