前 言

許多人一聽到「演算法」這個名詞時，覺得跟寫程式一樣要具備專業知識，想必門檻也很高吧。然而，演算法終歸只是計算的「步驟」，不需要用到電腦或程式語言的能力。

學習演算法的目的不在於「思考出新的演算法」，而是了解現有演算法的各個特點，並且能夠「在各種情況下分別正確地使用它們」。

當我們要實際運用時，需要先創建一個程式後並在電腦上執行，但只要掌握住箇中訣竅，可以比透過實機操作還能學到得更多。

近來出現了方便編輯的函式庫，程式設計師幾乎不再需要從零編寫演算法。世界上我們所使用的軟體都出自於這個由前人精心編列的函式庫。

重要的是要知道什麼情況使用哪種演算法有效。例如，查詢到站電車路線的服務、導航系統等軟體都有使用本書介紹的演算法。此外，企業使用的系統軟體裡有變更排列或是搜尋等功能，想必每個人都有使用過。知道幾個具代表性的手法，除了可以不用自己花時間從零編寫，若是有函式庫也做不到的複雜指令，或許添加一些獨創性就能夠達到快速運算的效果。

本書不僅只是介紹一般演算法教科書中會提到的教材，還介紹機器學習和密碼學中使用到的演算法。如果您想了解更多細節，請閱讀各領域的專業書籍。

2021 年 12 月 增井敏克

碁峰 www.gotop.com.tw

第 1 章

演算法的基本概念

～演算法的功能有哪些？～

碁峰
www.gotop.com.tw

快速準確的計算步驟

軟體的範圍

當我們啟動電腦時，單只靠鍵盤、滑鼠、CPU 或是硬碟的話無法完成任何事情，需要使用軟體才能夠運作這些硬體設備。

軟體包括 OS（系統軟體）和 application software（應用程式）。**這些不僅包括執行處理的程式，還包括程式運算的數據、以及如何使用程式的準則**（圖 1-1）。

設計程式的流程

編寫程式語言意指程式設計，有時也稱為系統開發或軟體開發。即使說是編寫一個程式，範圍可能是指圖 1-2 所示的一整個開發過程，也可能僅止於其中的單一過程。

確立主題過程會決定要開發的內容，以及該如何展開設計過程。執行檔稱作為編寫原始碼，有時也稱為編碼。開發的程式經過測試後，確實能正確運作的話才能提供使用，接著進入實際運用以及維護階段。

到底什麼是演算法呢？

程式會因**原始碼的編寫方式改變運算時間的長短**。正如我們身邊遇到的問題可以用不同方法解決一樣，編寫程式也有不同的編碼方法（圖 1-3）。這個步驟或是計算方法就稱為**演算法**。這時，選擇使用某種方式可以高效率、大幅減低運算的時間。因此開發人員必須懂得從多種演算法中選擇較好的方法運用。

碁峰 www.gotop.com.tw

圖 1-1 軟體的種類

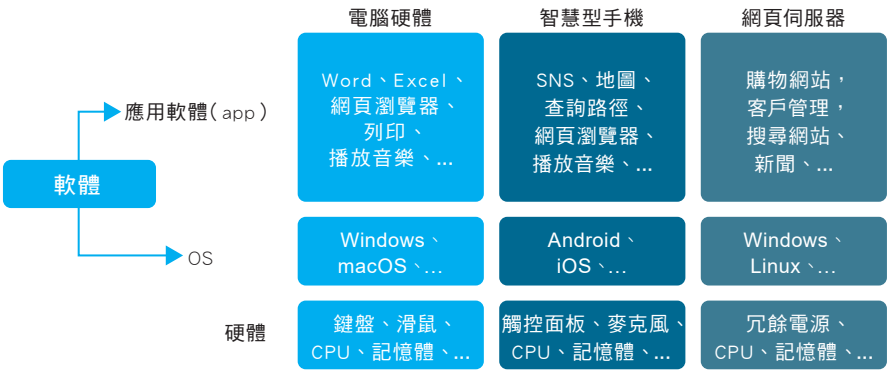


圖 1-2 開發流程

圖 1-3 有多種解題方式

105 × 95 = ?

解題 1

```
   105
 × 95
 ------
   525
  945
 ------
  9975
```

解題 2

$$
\begin{aligned}
105\times95 &= (100+5)\times(100-5)\\
&= 100^2-5^2\\
&= 10000-25\\
&= 9975
\end{aligned}
$$

Point

- 程式是軟體的一部分，創建程式稱為編寫程式
- 程式會因編碼方式不同而影響運算時間的長短，設計者必須要懂得選擇有效率的方法

碁峰 www.gotop.com.tw

方便處理的資料

方便使用的檔案會因人為和機器操作而異

當我們進行資料處理時，通常都會使用檔案。檔案大致分為**文字檔**和**二進位檔案**（圖 1-4）。

文字檔是**完全由字元所構成的檔案**。使用「記事本」等軟體開啟時因為顯示為文字，非常方便閱讀。

而二進位檔案是指**文字檔以外、如圖片和音樂等檔案格式**，需要用專用軟體才能讀取檔案，並且無法被轉換成文字。

電腦處理資料的類型

我們需要將檔案執行的具體含義編寫到檔案之中，才能執行此應用程式。二進位檔案可以儲存為方便程式執行的格式，但文字檔可以自訂編寫方式而無項目可言。

例如，在簡單幾個段落的筆記或日記中無法判別在哪裡以及寫了什麼，當需要尋找的時候就必須從頭搜尋來判讀是否吻合。這類資料稱為**非結構化資料**（圖 1-5）。

但若是像通訊錄等 CSV 文件，其姓名和地址等項目就相對明確。最近，也越來越重視在 HTML 文件之中建構標籤，而這種電腦易於處理的資料被稱為**結構化資料**。

不僅是檔案如此，在程式內部也是如此，**良好地保存檔案方便尋找，更能快速新增或刪除檔案**。因此，當我們要建立程式的時候，要同時思考演算法的**資料結構**。

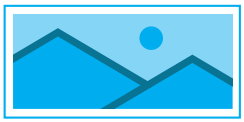
碁峰 www.gotop.com.tw

圖 1-4 文字檔與二進位檔案

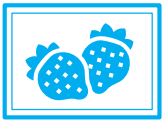
文字檔	二進位檔案
• 文章（txt、rtf）	• 圖片（bmp、png、jpeg、…）
• HTML、CSS	• 音樂（mp3、wma、…）
• CSV	• 影音（mov、mp4、…）
• JSON	• PDF
• XML	• 壓縮格式（zip、lzh、…）
• …	• …

圖 1-5 結構化資料、非結構化資料

非結構化資料



今天跟XX和XX出去玩。
早上天氣很好玩得很開心。
如果有機會要再去一次。



結構化資料

姓名	郵遞區號	地址	電話號碼
陳○迅	100206	臺北市博愛路一三一號	02-23226871
蘇○綠	100203	臺北市重慶南路一段一二四號	02-23713260
蕭○騰	106248	臺北市和平東路三段一巷一號	02-27027308
林○傑	111035	臺北市士林區士東路一九〇號	02-28312321

以列為單位，同一項目在同一列中

看標籤可知道內容

無法知道哪裡是名字、位置在哪裡

音樂、影音或圖片無法做搜尋

```
<html>
  <head>
    <title>○○</title>
  </head>
  <body>
    <header>
      <nav></nav>
    </header>
    <section>
      <h1>標題</h1>
      <article>
        文章的內容
      </article>
    </section>
    <footer>
    </footer>
  </body>
</html>
```

Point

- 非結構化資料不適合在電腦等硬體上做搜尋
- 結構化資料的格式可提升執行檔案的效率
- 在程式當中想要快速執行處理，除了演算法之外，資料結構也很重要

碁峰 www.gotop.com.tw

什麼是好的程式？

人們對電腦的需求

當我們使用軟體時，對於「好用」的標準因人而異。討喜的設計外觀，易懂的操作介面，也都可以算是一種基準；如果你是初學者，那麼手冊的易讀性對你來說可能就會很重要（圖 1-6）。

這些標準當然很重要，但是當**習慣使用某個軟體後，你會開始從「能不能更省時處理」的角度來重新審視**。就算再漂亮的設計，如果輸入之後需要花時間等待的話，就不會想再使用這個軟體了。還有，即使反應時間再快，但才處理一下就產生大量檔案使磁碟機容量馬上爆滿，也會妨礙工作效率。

不管是 CPU 跑得再快也好、增加再多記憶體或是硬碟等容量也罷，需要更有效的使用方式才有成效可言。

思考運算花費的時間和所需的容量

即使是很有效率的程式，也會因要執行的內容決定需花費時間長短。如果需要進行相當複雜的運算，則不可避免地會花費相對多的時間。

當效率的好壞用時間來做衡量時，就會思考**當數據量增加時會需要多少運算時間**。即使要處理的數據量很大也不會佔用太多時間的演算法，就是「好的演算法」。

但是，不能單看執行時間來判斷。例如，遇到需要花時間執行複雜運算的情況時，我們可以先存放所有的運算結果，再透過搜尋的方式取代重新運算的過程，加快作業的效率，但此方式需要有足夠的存放空間。因此，就必須要思考記憶體和磁碟機的使用量（圖 1-7）。

碁峰 www.gotop.com.tw

圖 1-6 可用性的指標（AC 尼爾森對易用性的定義）

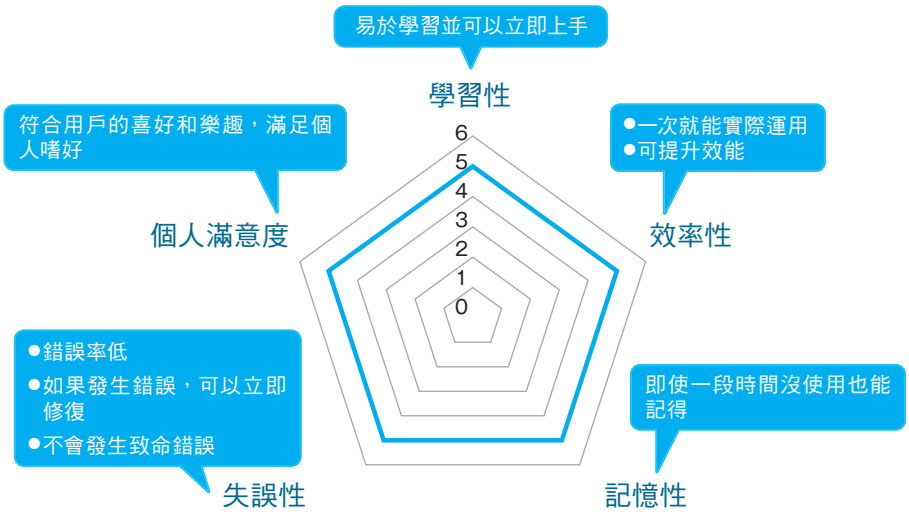
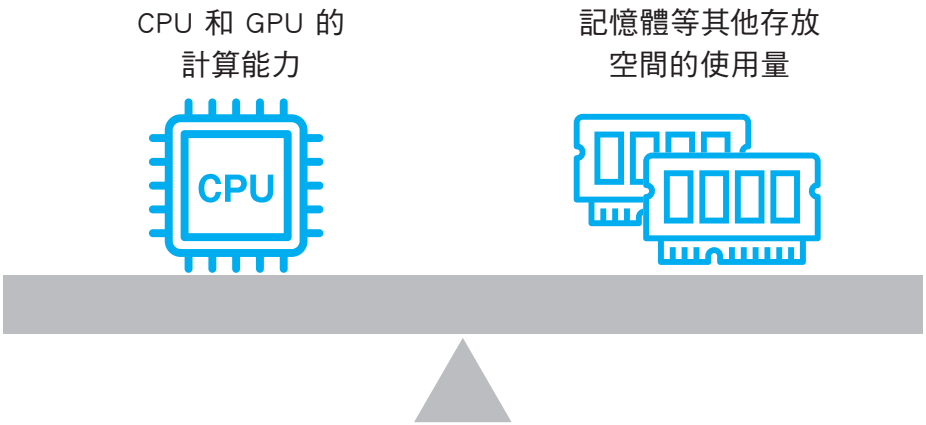


圖 1-7 需要花費的處理時間和所需的使用容量



Point

- 評估軟體好壞時，除了滿意度和易學性之外，效能也是重要指標之一
- 運算的數據量很大也不會佔用太多時間的演算法，就是「好的演算法」
- 即使縮短運算的時間，卻需要占用巨大的存放空間也不具任何意義。因此必須要同時考慮運算所花費的時間和記憶體的使用量

碁峰 www.gotop.com.tw

比較演算法的基準

演算法執行速度的指標

請試著想像如果程式的數據量增加時，要怎麼知道執行時間的多寡。 可以先漸進式增加執行 10 筆、100 筆、1000 筆的數據量，同時量測實際花費的時間，就能觀察出執行時間的變化趨勢（圖 1-8）。

正因為不是經由如此實測的方式，就無法驗證該演算法的好壞。在**設計階段沒有採用正確的演算法而等到完成開發後才發現問題，有可能會因為沒有時間修正導致來不及如期完成**。

此外，執行時間也會因選用的電腦而異。程式開發者使用高效能的電腦用 1 秒時間就能執行完畢。而普通的電腦可能需要花費到 10 秒時間。

使用不同程式語言，也會發生同樣的事情。相同的演算法用 C 語言實作的執行效率高，但以 Python 類型的腳本語言來執行有可能會花費時間。

因此，不侷限於環境或語言當中，評估演算法的執行效率的指標是**複雜度**。

分析複雜度

當有多種演算法時，我們會思考如何比較個別的複雜度。忽略常數值內對整體執行時間影響不大的部分的符號，稱為 **order（等級）**、寫法為 **order notation**。表示 order 時會使用「O」符號，也稱為大 O 符號。

例如，輸入的數量為 n，等同於此 n 是以 $O(n)$、n 的 2 次方為 $O(n^2)$ 的表示法。也就是說，若評估 $O(n)$ 和 $O(n^2)$ 這兩種演算法時，可以判斷 $O(n)$ 演算法的執行效率較佳（圖 1-9）。

碁峰 www.gotop.com.tw

圖 1-8 處理大量資料時執行時間的變化

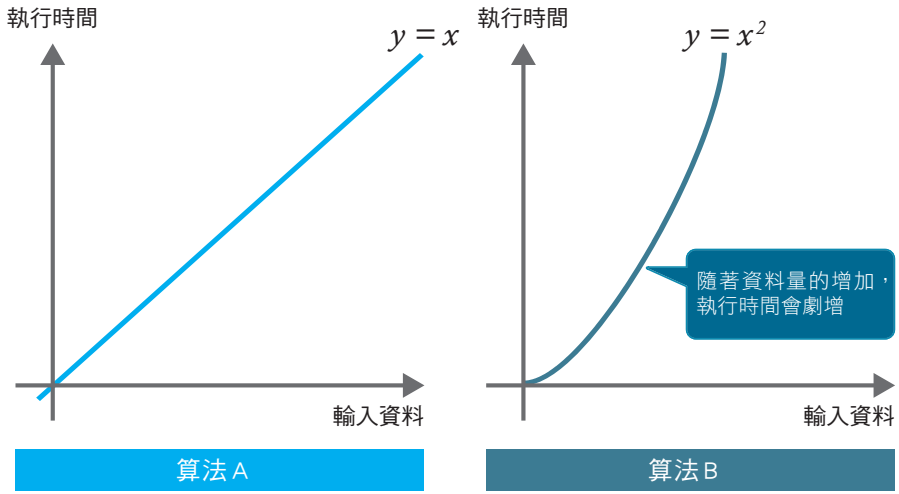


圖 1-9 比較 Order 的複雜度

執行時間	order	舉例
短	$O(1)$	存取陣列上的元素等
	$O(\log n)$	二分搜尋法等
	$O(n)$	線性搜尋法等
	$O(n\log n)$	合併排序、快速排序等
	$O(n^2)$	選擇排序、插入排序等
	$O(n^3)$	矩陣乘法等
	$O(2^n)$	背包問題等
長	$O(n!)$	旅行推銷員問題等

Point

- 評估演算法的執行效率的指標是複雜度，文字寫成 Order
- 使用 Order 時可以忽略對整體執行時間影響不大的部分，比較 Order 可以大致看出執行時間的增加方式

碁峰 www.gotop.com.tw

認識程式語言的差異

選擇程式語言

程式是由一連串的原始碼構成，而編寫原始碼所使用的語言稱為**程式語言**。與我們平時使用的日文和英文不同，**它是由電腦執行為前提下產生的語言**，號稱有成千上萬種程式語言存在於這個世界中。

多半都是依照自己的目標或是喜好選擇使用哪種程式語言居多。例如，Windows 應用軟體使用 C#，iOS 應用軟體使用 Swift，Android 應用軟體使用 Kotlin，網頁應用軟體使用 PHP 和 JavaScript 等。再來就是看公司或個人需求選用語言（圖 1-10）。

如何轉譯為程式

編寫好的原始碼不代表能夠運行程式，還必須將原始碼轉譯成程式的步驟，其方法有分為**編譯器**和**直譯器**（圖 1-11）。

編譯器是**事先將寫好的原始碼轉譯為程式，再執行完成轉譯後程式**。有如翻譯資料一樣，需要花時間轉換，但執行速度很快。

直譯器是**轉譯一行原始碼就立刻執行的方法**，它像是有翻譯人員同步進行翻譯的感覺。不需要事前的準備作業，但需要執行的處理時間。

近來也出現看起來像直譯器是逐次轉譯，但內部是與編譯器相同轉換方式的程式語言。稱為 **JIT（Just In Time）編譯器**，初次執行的時候需要時間，但是第二次以後的執行速度非常迅速。

碁峰 www.gotop.com.tw

圖 1-10 選擇符合需求的程式語言

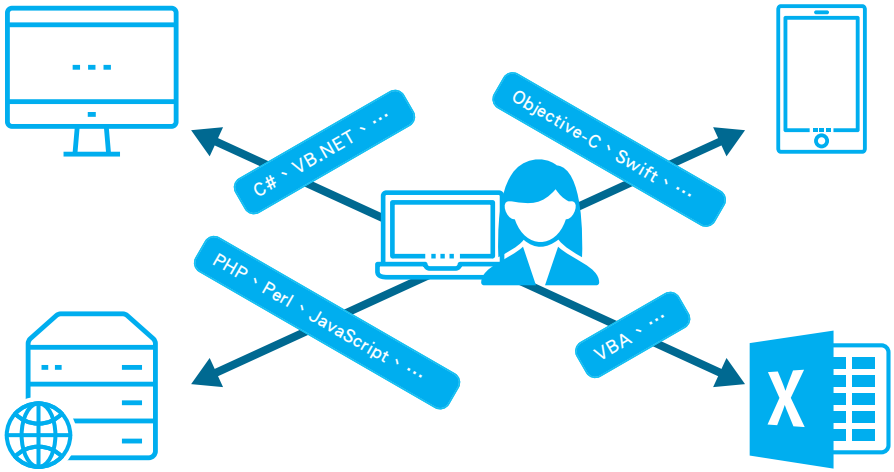
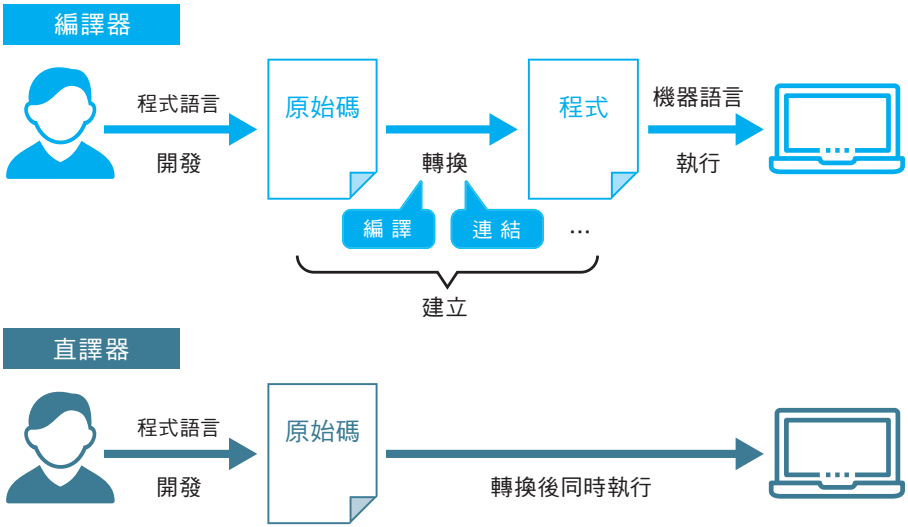


圖 1-11 編譯器和直譯器的差別



Point

- 程式語言的種類眾多，要依照需求目的選擇適用的方式
- 執行原始碼編寫出的程式語言需要先透過編譯器、直譯器轉譯處理

碁峰 www.gotop.com.tw

好用的演算法都在這

提供開發通用程式的功能

各個程式語言的許多常用功能在**函式庫**裡都找得到，例如發送電子郵件、日誌檔案、數學函數和影像處理，以及讀取保存檔案等都是。

有了函式庫**無須從頭實作就能夠輕鬆實現所需的功能**。此外，一個函式庫可分享給多個程式使用，進而有效地利用記憶體、硬碟空間（圖 1-12）。

快速實用的函式庫

多數程式語言都會內建常用到演算法的函式庫，程式設計師即使不清楚內部結構也能夠輕鬆實作這些演算法。

例如，在 Java 程式語言中，不僅提供處理日期時間、數學計算、影像處理和發送郵件等功能，搜尋字串和 sort（排序）等功能也都來自於函式庫。只要將它們讀取就簡單完成排序（圖 1-13）。

函式庫可以讓我們不再需要從頭實作排序等其他演算法。大部分的程式語言都是準備資料後，只要呼叫「sort」的指令就能快速執行功能。

或許這樣會讓你覺得沒必要花費太多心力在學習演算法上。當然就工作而言，要用到排序的時候可以從函式庫抓取，但如果沒有具備相關概念的話，有可能會除了排序以外其他都一概不通。內建函式庫裡沒有提供的功能我們就必須要想辦法自行解決問題，**處理類似問題時，光知不知道排序演算法就會影響你寫程式的效率**。

碁峰 www.gotop.com.tw

圖 1-12 函式庫

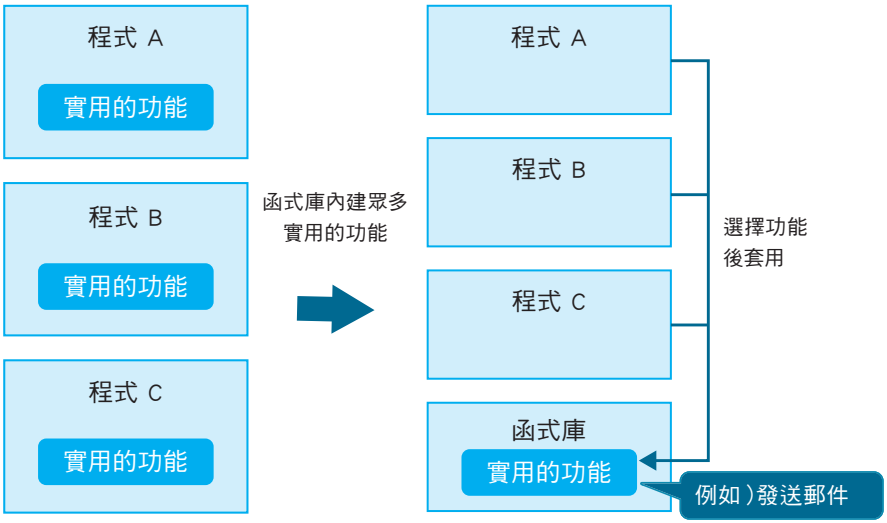


圖 1-13 使用函式庫的原始碼範例（Java 語言）

```
import java.util.Arrays;          ← 讀取函式庫

class Test
{
    public static void main (String[] args)
    {
        int[] a = {1, 8, 3, 7, 2, 4, 9, 5, 6};   ← 準備
        Arrays.sort(a);                          ← 執行排序
        System.out.println(Arrays.toString(a));  ← 輸出結果
    }
}
```

Point

- 使用函式庫可輕鬆實現許多程式功能
- 多數程式語言都有內建許多常用演算法的函式庫

碁峰 www.gotop.com.tw

演算法的權限

保護軟體的制度

軟體與工業產品不同，有易於被他人複製的特性。當你要開發新的軟體時，請想好該如何維護自身權益（圖 1-14）。

一項新的發明依專利法提出申請，經核准後便可獲得**專利權**。如此一來等於獨家擁有該項技術的使用權，如果有第三者未經同意使用該專利的話，有權利要求賠償損失。

而軟體有所謂的**軟體專利**，許多加密技術的手法已經被申請專利，也有因該專利權引起糾紛的訴訟案件。**演算法可以申請發明專利，缺點在於一旦申請該項專利就是將相關技術公諸於世**。還有程式語言不屬於發明的一種，並無專利可言。

原始碼的著作權

與專利權相同能夠保護作者權益的還有**著作權**，歸屬文字和音樂創作類型。與專利權不同的是不需要透過申請，只要能證明自己是原創者即享有著作權。

程式的原始碼也受著作權保護，**不能複製他人創建的原始碼用在自己的軟體當中**。由事業單位所開發的原始碼著作權基本上是歸屬於事業單位。以及程式語言和演算法不適用於著作權規範。

近來越來越常見以**公開資源**的形式對外發佈原始碼的平台。有需要時只要符合授權之利用條款，使用者便可以自由地取用、修改和分享。有些會要求公開修改的部分原始碼作為授權條件，請確認授權條款的內容再評估使用（圖 1-15）。

碁峰 www.gotop.com.tw

圖 1-14 知識財產權的類別

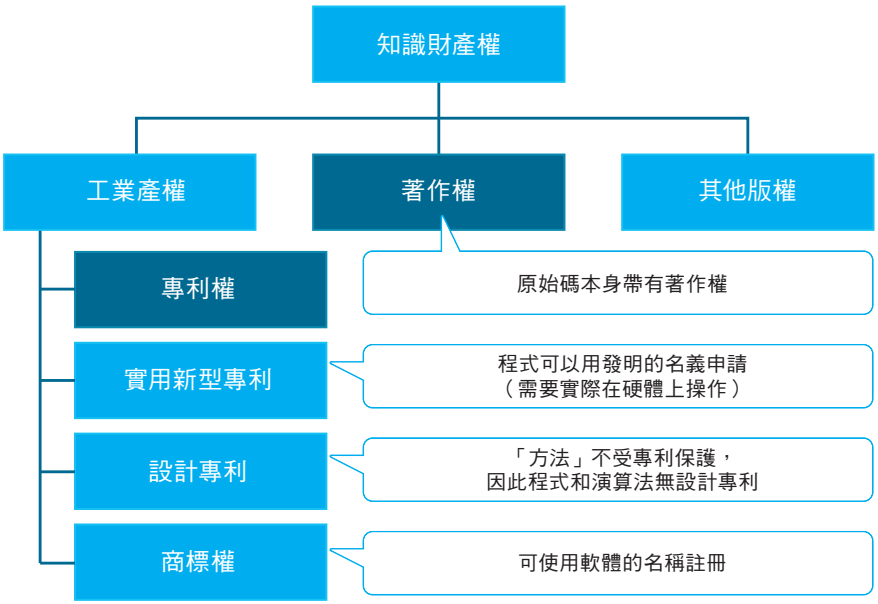


圖 1-15 公開軟體的授權條款

分類/類型	授權案例	公開修改過的原始碼	公開其他軟體原始碼
COPYLEFT類型	GPL、AGPLv3、EUPL等	需要	需要
半COPYLEFT類型	MPL、LGPLv3等	需要	不需要
非COPYLEFT類型	BSD License、Apache 2.0 License、MIT License等。	不需要	不需要

參考：《比較OSS授權的使用趨勢與糾紛調查》（日本信息技術振興機構）

Point

- 可以申請專利保護新發明的軟體
- 原始碼本身帶有著作權
- 如果需要使用公開資源請記得確認授權條款，唯有遵照條款使用才能夠自由地取用、修改和分享

碁峰 www.gotop.com.tw

圖解演算法

與他人建立共識所用的流程圖

即使懂得如何寫程式，要看得懂別人寫的原始碼也是件難事，就算有用中文寫出備註項目，也是要把每一行執行的內容都看過一遍。這個時候，如果有圖示的流程說明的話是不是會更易於理解呢？

因此用於表示「處理流程」的圖表我們稱做為**流程圖**。它是由 JIS（日本工業標準）規定的標準規格，不僅用在說明程式執行上，還有行政作業的流程等也以圖表的方式說明各項作業流程。

程式可藉由圖 1-16 的搭配方式來實現基本的執行項目，將這些符號如圖 1-17 排列出的圖表稱為流程圖，如果**希望跟所有人都能夠達成共識，切記要使用標準符號進行繪製。**

流程圖的重要性

最近大家在建立程式的時候，不管是在設計階段還是實作階段都越來越少畫流程圖了。這是因為大部分認為先建立好程式並實際確認運作，比畫圖表來得更有效率。有時候因應客戶要求提供資料時，也是先建立完程式之後再來繪製流程圖。

另一方面，在構想物件導向程式設計或是物件導向設計時，常使用一種叫做 UML 的圖表。即便寫出或要看懂原始碼不是件簡單事，但透過繪製流程圖、UML 等圖表能幫助人們方便理解。

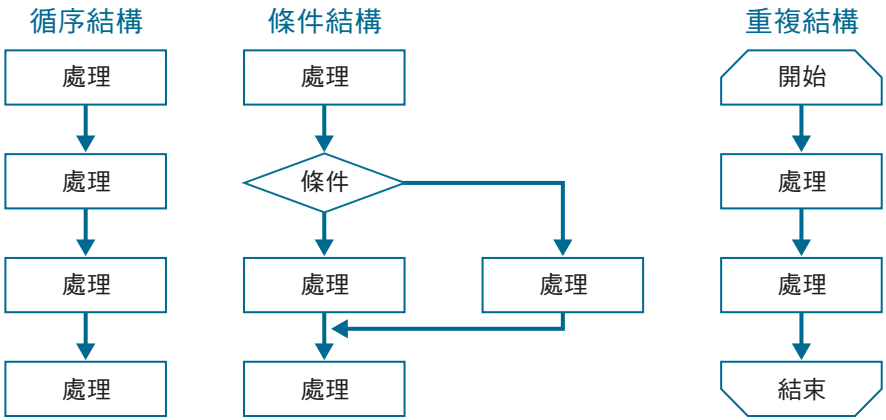
而 UML 中有叫做活動圖的圖表也類似於流程圖的結構。到現在流程圖的構想仍然被認為是有效的。

碁峰 www.gotop.com.tw

圖 1-16 流程圖常用的標準符號

意思	符號	詳細
起止符號		代表流程圖的開始和結束
處理符號		表示即將執行的一些動作
決策判斷符號		表示對某一條件做出判斷 記號內寫入其條件
迴圈符號		表示重複動作 設定變數的初始值（上）和終止值（下）
輸入符號		表示需要用戶輸入
副程式符號		表示一群已經定義流程的組合

圖 1-17 基本的流程控制結構



Point

- 流程圖是用來說明處理流程的圖表
- 程式中有依序執行、決策判斷、迴圈的搭配方式，可以用來說明多種情形
- 雖然程式設計的過程中，不一定要繪製流程圖，但需要向他人說明時，流程圖仍然是一個有效的工具

碁峰 www.gotop.com.tw