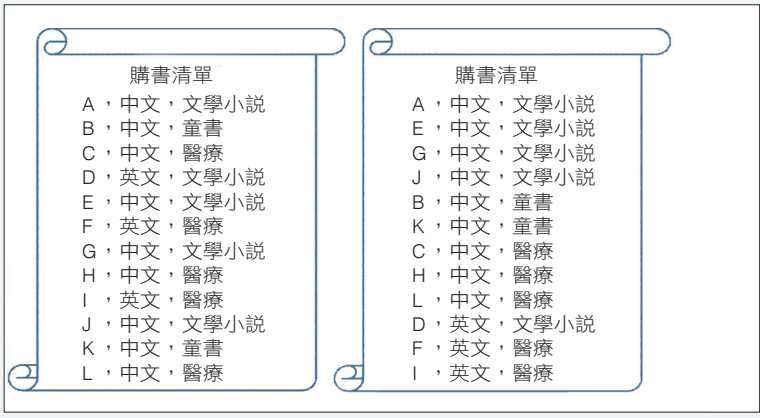
通俗演算法

謝孫源、李佳衛、洪綾珠

在日常生活中，演算法時時刻刻都在幫我們解決問題，
選用了合適的演算法，可以大幅提升生活品質，節省時間、空間或成本。

「演算法（algorithm）」在韋氏辭典的定義是「在有限步驟內解決數學問題的程序」；在計算機科學領域上，演算法則是一個計算的具體步驟，常用於計算、資料處理和自動推理。簡言之，演算法是為了解決某一問題所設計的一組有限運算規則的集合，其中包含精確的問題輸入與輸出。用白話來說，可以把演算法定義成「解決問題的方法」，它可以是利用文字敘述、流程圖或虛擬碼的方式來表示解決問題的步驟。



購書清單是否經排序，會影響到我們進出中文小說區的次數。

演算法有什麼用呢？它可以幫我們提升生活品質，例如節省時間、空間或成本等。既然演算法可以定義成解決問題的方法，我們又用了哪些演算法來解決哪些問題呢？

就以到書局買書為例吧！通常我們都會把欲購買的書籍列在清單上，到書局後就依照清單上的順序逐一去找，常用的方法就是演算法裡的「貪婪演算法（greedy algorithm）」。這個方法雖然可以找到所有的書，但是過程中可能需要在書局裡來回奔走，因為書局通常是依書籍的性質分類，例如小說、童書、醫療等。

倘若我們是依照清單上的順序，則有可能先在小說區找到書本 A，接下來到童書區找書本 B，然後到醫療區找書本 C，由於第四本書 D 是英文的文學小說，因此得先找到外文的小說區才能找到書本 D。而第五本書 E 是文學小說，我們又得回到中文的小說區。至於清單上的書本 G 和 J 都是中文的文學小說，因此又得進出中文小說區兩次才能找全所有書籍。

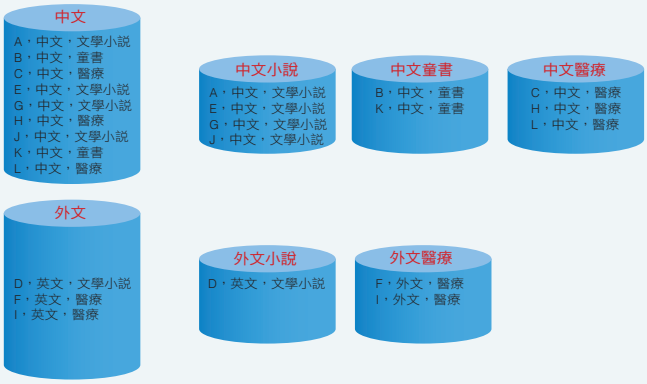
如果換個方式找書，先把清單上的書籍依性質分類，這樣一來，就可以先在中文小說區找到書本 A、E、G 和 J，接著再到中文童書區找書本 B 和 K，然後到中文的醫療區找書本 C、H 和 L，最後再到外文區尋找書本 D、F 和 I。這種程序避免了在書局裡來回奔走的狼狽情形。

上述依照書籍性質分類的方法，就是演算法裡的「桶子排序法（bucket sort）」。這方法想像我們有中文和外文兩個桶子，先依照書籍的語文分類，書本 A 是中文，因此放入中文的桶子中；書本 B 是中文，因此放入中文的桶子中；書本 C 是中文，也放入中文的桶子中；但書本 D 是英文，因此放入外文的桶子中；如此這般依序把其餘的書本各放入其歸屬的桶子中。

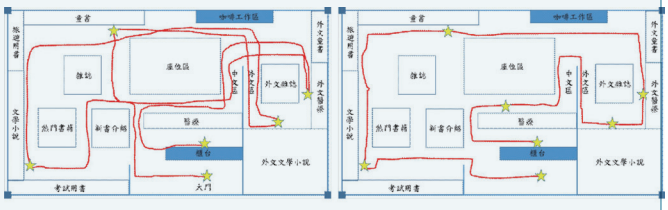
接下來，再針對桶子排序。想像有中文小說、中文童書和中文醫療 3 個桶子，書本 A 就該放入中文小說的桶子中；書本 B 則放入中文童書的桶子中；如此依序置放。接著再針對外文的桶子也做相同的排序。經過桶子排序法的整理之後，就可以在同一區塊內找到全部同一性質的書籍，也就避免來回奔走了。

由於每一間書局有各自的擺設格局，如果可以先知道各類書籍的擺放位置，應該可以找到一條最佳化的路徑，作最少路徑的奔波。這個尋找最佳化路徑的問題就是演算法中非常著名的「旅行推銷員問題（travelling salesperson problem）」。

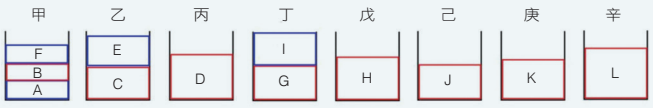
你終於把清單上的書找齊了，但是因數量太多，搬運並不方便，因此需要請書局宅配回家。然而書局所提供的箱子尺寸是固定的，如果裝箱越多，荷包會越傷，因此



以購書清單為例的桶子排序法



是否有先經演算法運算，所走的路徑差異可能很大。如進書局大門後，先至中文童書區找到書本 B 和 K，再至外文小說區找到書本 D，接著到中文小說區找到書本 A、E、G 和 J，再到外文醫療區找到書本 F 和 I，到中文醫療區找到書本 C、H 和 L，最後回到大門前的櫃台結帳，總共需要 24 個單位距離。然而若能找到最佳化的路徑，如走大門、中文小說區、中文童書區、外文醫療區、外文小說區、中文醫療區、大門的路徑，則只需 13 個單位距離。



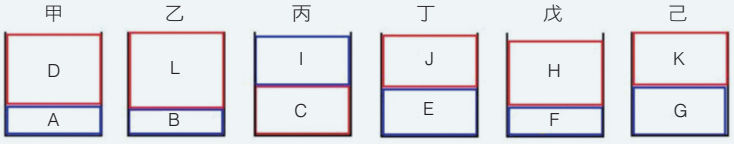
使用最先適配演算法，每一次找到適合的箱子，就把書本直接放進去。

箱子的數目越少越好。但如何使用最少的箱子，就是演算法中著名的「裝箱問題（bin packing problem）」。這裝箱問題在計算機科學中也是一個相當難的問題，以下介紹「最先適配演算法（first-fit algorithm）」來解決裝箱問題。

假設所購買的 12 本書長、寬大小都一致，只有書本厚度不同，而書局所提供的箱子，長、寬與書本的一致，且設箱子高度是 10 公分。最先適配演算法的概念就是先找到第一個可以放入書本的箱子。設書本 A 厚度 3.2 公分，且目前因還沒有用到任何箱子，因此就把書本 A 直接放入箱子甲。又書本 B 厚度 2.7 公分，而箱子甲還有 6.8 公分的空間可以使用，因此又把書本 B 放入箱子甲。然後是書本 C，厚度 4.3 公分，但是箱子甲只剩下 4.1 公分的空間，因此需要另外拿一個箱子乙來裝書本 C。

接著是書本 D 厚度 6.7 公分，箱子甲、乙所剩空間都不夠裝書本 D，因此另需箱子丙來裝書本 D。書本 E 厚度 5.3 公分，箱子甲只剩 4.1 公分的空間，而箱子乙還有 5.7 公分的空間，因此把書本 E 放入箱子乙。書本 F 厚度 2.5 公分，而箱子甲還有 4.1 公分的空間，因此把書本 F 放入箱子甲。如此這般，書本 G、H、I、J、K、L 終於找到其各自歸屬的箱子，算算眼前的箱山，總共使用了 8 個箱子來裝這 12 本書。

上述的最先適配演算法顯然沒能找到最佳解，以上面裝書的例子來說，其實只要使用 6 個箱子就可以了：即把書本 A 和 D 放入箱子甲；書本 B 和 L 放入箱子乙；書本 C 和 I 放入箱子丙；書本 E 和 J 放入箱子丁；書本 F 和 H 放入箱子戊；書本 G 和 K 放入箱子己而達陣。



關於裝箱問題的最佳解，當數量不大時，或許還可以用人工方式找到最佳解，然而當數量非常龐大時，就得仰賴演算法了。

前述例子由於數量並不大，用人工的方式或許就可以找到最佳解；但是若問題的數量非常龐大時，就非得仰賴演算法不可了。然而演算法並不是萬靈藥，無法確定在有限的時間與空間下可以找到最佳解。但即便如此，退一步想，縱然所得到的答案不是最佳解，離最佳解也不會太遠，這個概念就是演算法中的「近似演算法（approximation algorithm）」。

生活中還有許多運用演算法的例子。例如統一發票對獎，通常都是逐一核對每張發票上的數字，檢查其是否是中獎的號碼。而每核對一張發票時，就需要搜尋中獎號碼，這個過程就是演算法裡的「搜尋演算法（searching algorithm）」。假設有 100 張發票需要對獎，則至少需看過這 100 張發票上的末三碼，也就是 300 個數字。

然而，若能先依照最後一碼的數字把發票分類，則需要看的數字量就可以減少。

演算法並不是萬靈藥，無法確定在有限的時間與空間下可以找到最佳解，但即便所得到的答案不是最佳解，離最佳解也不會太遠。

因為若最後一碼的數字與中獎號碼的數字不同，即使前面 7 個數字都相同，依然是廢紙一張。而這樣的對獎方式，也就是前述的「桶子排序法」。由此可知，一個設計良善的演算法的確可以幫助我們提高工作效率。

在日常生活中，最常使用到的演算法又是什麼呢？答案是：排序演算法！諸如依照身高安排座位或成績排名等都需要使用這個演算法。以下介紹幾種依身高安排座位的排序演算法。在新生入學時，老師會先把全班學生依照高矮排序安排座位。若老師不知道每位學生確實的身高時，該如何排出高矮順序呢？這個問題就是演算法裡的「排序問題（sorting problem）」。

統一發票對獎的過程，若先經過桶子排序法就可增進工作效率。

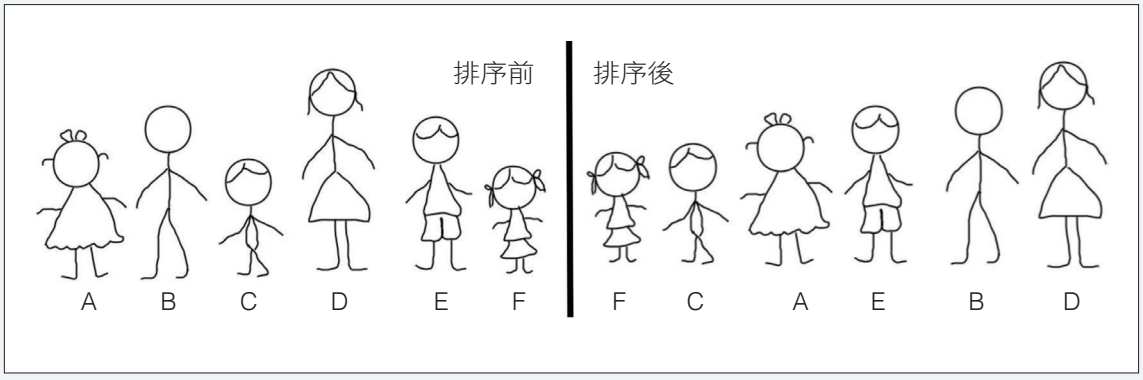
氣泡排序法

請學生排成一列，然後比較同學 A 和 B 的身高，較矮的同學 A 往前排在第一位，較高的同學排在第二位。接下來比較同學 C 的身高，因為 C 比同學 B 矮，則與同學 B 交換位置；也比同學 A 矮，再與同學 A 交換位置。接下來的同學 D 比排在第三位的同學 B 高，因此不需要更動排位。依序逐一比較，就可以排出高矮的順序。這個方法稱為「氣泡排序法（bubble sort）」，因為在過程中，較矮的同學就如氣泡般一直往上浮而得名。

插入排序法

挑選同學 A 排在第一位，接著再挑選同學 B 與排在第一位的同學 A 比較身高，由於同學 B 比同學 A 高，因此把同學 B 排在第二位；再挑選同學 C 與目前排在序列中的同學比較，由於同學 C 比目前排在第一位的同學 A 矮，因此把同學 C 排在第一位，原本序列中的同學每人都往後退一個位置。

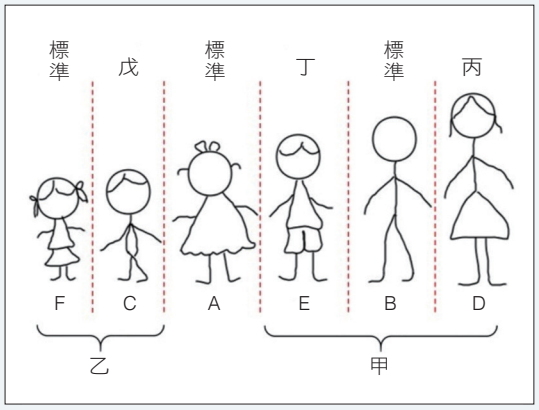
接著再挑選同學 D，再次與目前序列中的同學比較，由於同學 D 比目前的每一位同學都高，因此把同學 D 排在第四位。如此依序挑選同學與目前序列中的每一位同學比身高，再把這位同學插入適當的位置，就可完成身高排列的順序。這個方法就是「插入排序法（insertion sort）」，因為是在過程中逐一找到位置，再把同學插入這個位置而得名。



日常生活中最常用的演算法就是排序演算法，例如老師為了排座位所做的身高排序。

選擇排序法

利用目測的方式找出身高最矮的同學 F，把同學 F 排在第一位；接著從剩餘的同學中，再次目測找出最矮的同學 C，把他排在第二位；在剩餘的同學中持續找出身高最矮的同學，把他排到順序之中，直到所有同學都排進隊伍中。這個方法就是「選擇排序法（selection sort）」，因為在過程中，每一次都從尚未排入隊伍的同學中選擇最矮的同學來排入而得名。



快速排序法的例子

快速排序法

選擇同學 A 當作標準身高，其他同學都須與 A 比較身高，把比同學 A 高的同學 B、D 和 E 分到甲區，比同學 A 矮的同學 C 和 F 分到乙區；接著在甲區的同學中，挑選同學 B 當作標準身高，把比同學 B 高的同學 D 分到丙區，比同學 B 矮的同學 E 分到丁區；在乙區的同學中，挑選同學 F 當作標準身高，把比同學 F 高的同學 C 分到戊區。如此一來，每個小區域內只有一位同學，而兩個小區域之間有一位被當作標準的同學。最後，把每個小區域及標準身高的同學合併起來，這個方法就是演算法中的「快速排序法（quick sort）」。

在日常生活中，其實我們時時刻刻都在使用演算法解決問題，只是不曉得那個演算法的名稱，也不曉得是否可以得到最好的答案，甚至不曉得那個演算法是否有很好的效率可以省下很多時間。一般最常使用的是貪婪演算法，這方法雖然可以得到答案，但通常不是最好的，效率也不見

得理想。如果能夠學習到更多的演算法，明瞭該採取什麼方法以解決日常生活上的問題，就可以有效地節省時間、空間與成本，並提升生活品質。

演算法在有限的時間與空間內，可以有效率地解決問題，因此一直是資訊工程研究中非常重要的議題。例如在設計導航系統時，希望可以提供一條最佳化的行進路徑（時間最少或距離最短），這時需設計一個搜尋路徑的演算法，而 Dijkstra 演算法（Dijkstra algorithm）就可以幫助我們找到最佳解。又如常用的搜尋引擎，我們希望可以迅速地獲得最準確的搜尋結果，這時就需要設計一個有效率的字串比對演算法。

演算法的應用十分廣泛，而設計演算法能讓我們更有效率地解決各式各樣的問題，因此演算法在資訊工程領域中扮演著非常重要的角色，也一直是值得投入的研究方向。

謝孫源、李佳衛、洪綾珠
成功大學資訊工程學系