資料結構

FUNDAMENTALS OF DATA STRUCTURES

第五版

林至翔 高憲敬 合譯

HOROWITZ
SAHNI 原著

松崗電腦圖書資料有限公司

資料結構

FUNDAMENTALS OF DATA STRUCTURES

第四版

林至翔 合譯
高憲敬

HOROWITZ 霍羅威茨
SAHNI 原著

松崗電腦圖書資料有限公司 印行

資料結構

FUNDAMENTALS OF DATA STRUCTURES

每本定價 250 元整

書號：510105

編著者：林至翔、高憲敬

發行人：吳 守 信

發行所：道明出版社

台北市仁愛路二段一一〇號三樓

總經銷：松崗電腦圖書資料有限公司

台北市仁愛路二段一一〇號三樓

電話：3930255 · 3930249

郵政劃撥：10930

印刷者：泉崗印刷設計股份有限公司

台北市仁愛路二段一一〇號三樓

電話：3930255 · 3930249

中華民國 七十 年 十 月 初 版
中華民國七十一年 六 月 第二版
中華民國七十二年 二 月 第三版
中華民國七十二年 八 月 第四版
中華民國七十三年 三 月 第五版

本出版社經行政院新聞局核准登記，

登記證號為局版台業字第一七二九號

序

資料結構這門課已經在計算機系被教了許多年，而且經常視爲計算機課程裏面的主要課程。看看這門課主要內容變遷的歷史，會給我們一些啓示。1960 年代中期，這門課不叫做資料結構，而可能稱爲字串處理語言（ List Processing Languages ）。其主要內容是一些語言系統，例如SLIP（ J.Weizenbaum ），IPL-V（A. Newell ，C. Shaw 和H.Simon ），LISP 1.5（J.McCatthy ）以及SNOBOL（D. Farber ，R. Griswold ，I. Polonsky ）。1968 年，D. Knuth 寫的 Art of Computer Programming 第一册出版。他的論點是，字串處理並不是只在特別設計的系統中才能完成的魔術。同樣的技巧可以用在幾乎所有的語言中，所以他的重點移到演譯邏輯的有效設計上。SLIP和IPL-V退出這個舞台，而LISP和SNOBOL 則放到程式語言課程中講授。新的課程重點在於如何利用一群連續的記憶位置建立某種表示法（如鏈結串列），以及如何用英文加上組合語言來描述演譯邏輯。

最重要的一個觀念是，我們必須分別資料結構的基本結構和在現成程式語言中實際結構的不同。以前的教科書大都忽略了這項不同，而只強調程式語言或者表示法技巧的重要。現在我們嘗試去區分資料結構的基本結構和實際結構，並且證明這兩種程序都可以成功的完成。在基本結構的階段上，我們只注意去描述資料結構的功能，而不考慮如何實際應用。這可以由英文和數學符號組成，但我們引入另一種程式符號，稱爲公理。這個與實際情形無關的基本結構有兩項價值：(1)對證明利用此種資料結構的程序是否正確很有幫助。(2)證明資料結構的某種實際結構正確。不用實際表示法，要描述資料結構就需要某種語法，這個語法可以在 1.1節末看到，同時也嚴格定義資料集和資料結構的符號。

本書也講到如何分析演譯邏輯，但是卻沒有複雜不當的數學。實際結構的價值主要是看資源利用的情形，也就是時間和記憶空間的利用。這暗示著學生必須有能力分析這些因數。文獻上雖然記載了許多這類分析，但我們覺得大部份學生都不願

意詳細分析自己的程式。資料結構這門課適時的給他們灌輸這些觀點。本書的每個演譯邏輯，我們都曾對其性能做簡單，但是嚴格的最壞情況分析。某些例子中甚至求出平均計算時間。

資料庫系統的發展已經給資料結構課程加進新的課題，那就是大檔案的資料結構。很多教授也喜歡講到分類和搜尋，因爲其中包含了大量資料結構及實際應用的例子。最後幾章選擇的材料可反映出這個趨勢。

我們特別要考慮的，是選擇適合描述演譯邏輯的語言。但是由於學生的背景不同以及各學校使用語言的分歧，不容易選擇適當的語言。最後我們決定用一種類似 Algol 的語法，但是並不限制用那種特定語言。這樣可以寫出可讀性高的程式，同時不會受制於某種特定語言的結構。可能的話，也參用英文敍述，使演譯邏輯的要點更清楚。如果有人不熟悉 IF-THEN-ELSE ，WHILE，REPEAT-UNTIL 以及其他具體指令，可以參考 1-2 節，那裏用流程圖定義這些指令的語意。對那些只能用 FORTRAN 的學生，附錄中有規則可以直接由演譯邏輯譯成 FORTRAN 程式，同時附上翻譯器（見附錄 A）。另一方面，我們不利用語言中可以自動完成複雜資料結構運算的特性。原因有下面幾點，其中之一是要使尚未開始學習的學生專力於一種語法，這種語法使本書特別不好閱讀。更重要的一點，這些自動的特性涵蓋了實際應用上的細節，而精通這些細節仍然是本課程的里程碑。

本書的主要對象是主修電腦科學並且上過一年以上的課程的學生，還有非電腦科系的研究所一年級學生。本書內容至少須要一個學期的教授時間，其中某些章節可以跳過不教。底下有兩種安排可供參考。

本書第一位作者曾用本書教授大學二年級生，他們已經上過一學期的 PL／I 和一學期的組合語言。授課內容包括第一章到第五章，中間跳過 2.2,2.3,3.2, 4.7,4.11 和 5.8 等節。剩下的時間就教第七章的分類。第二位作者用本書教三年級生，他們曾有一學季的 FORTRAN 或 PASCAL 和二個學季的入門課程，其中包括許多主題。在第一學季的資料結構課程，1 到 3 章稍微帶過，4 到 6 章則仔細研究。第二學季由第七章開始，前面幾章提到的技巧都在這一章用到，因此本章是很好的應用例。剩餘時間可以討論外部分類，符號表列和檔案等章。注意，第二章包含大量的數學，可以略述不教。

我們所用的講授方法是以一個問題開始每個新的專題，這些問題通常選自計算

機科學的範圍。問題定義後，就用高階語言提出解答，再把資料結構給予公理式的定義。由嚴密分析決定那些運算是主要的。然後實際應用資料結構，以表明資料結構的表示法和基本結構是相互一致的。最後討論書中演譯邏輯的正確性，分析有關的變數，用直接的方法導出計算時間的正確方式。

總之，我們希望向學生們強調下列數點：(1)用高度抽象的法則定義資料結構和演譯邏輯的能力；(2)設計資料結構的各種實際結構的能力；(3)正確編寫演譯邏輯的能力；以及(4)分析既有程式計算時間的能力。除此之外，有兩個比較不顯明的趨勢在本書各處都可發現。第一個趨勢是編寫結構性強的程式。本書所有的程式，我們都儘量的具結構性。我們希望學生藉閱讀這些好的程式養成良好的習慣。第二個趨勢表現在例子的選擇上，我們採用能充分說明定理特性的例子，以及那些可以應用到程式設計上，或在計算機科學裏有重要貢獻的例子。

Ellis Horowitz

Sartaj Sahni

譯者序

資料結構這一門學問所研究的事項爲計算機內部資料的各種儲存方式，這些資料安排的好壞對計算機作業產生一決定性的影響，資料安排得好，作業必能省時省力，否則必定事倍功半。因此對一個有志於學習計算機科學的人來說，資料結構應爲其入門的第一個課程。

本書的目的，在介紹各種常用的資料結構方式如堆疊（ stack ），串列（ list ），佇列（ quene ），樹（ tree ），圖（ graph ）等之理論與實際的應用，更難能可貴的是原著者對每一個演譯邏輯（ algorithm ）都能分析其時間因素及所需之記憶空間，這是坊間其它中英文本教科書所未見的。另外在書中還介紹了各種分類（ sorting ）及搜尋（ searching ）的技巧以及檔案（ file ）處理之理論，在內容上已將資料結構所應討論的事項完全包含在內，因此確爲一不可多得之好書。

本書在翻譯的過程中遇到許多計算機的專用名詞，這些名詞的中文譯釋都以儘量與其它中文本所用的名詞相統一爲原則，但有些名詞譯者覺得其它書籍的翻譯有商榷的必要，經再三考慮後以能最適切表達原意之名詞取代之。

最後本書完稿之後雖經數次修校，但難免仍有疏漏，若讀者發現有不當之處，尚祈不吝賜教。

譯者謹誌於台北

目 錄

第一章 概 論

1.1 緒論

“計算機科學”的領域是非常新的，因此我們覺得在著手這本書以前必須先提供一個定義給各位。我們通常引用定義而將計算機科學視爲一種“演繹邏輯的研究”（Study of algorithm），這種研究包含四種不同的範圍：

(1)“**執行演繹邏輯的機器**”——這個範圍包括從最小的手上型計算器到最大型的多用途數位計算機之間的所有事物，其目標爲研究各種不同型式的機器結構與組織以便使演繹邏輯能夠被有效的實現。

(2)“**敍述演繹邏輯的語言**”——這些語言能夠以一種連續的方式被存放著。一方面這些語言必須能夠與實際的機器相溝通而另一方面這些語言又是被設計來解決一些複雜的問題。我們通常將這個範圍分爲兩個面：語言的設計與語言的翻譯。前者尋求描述語法 Syntax 和語義 Semantics 的方法，後者則要求將語言翻譯成一些基本命令的方法。

(3)“**演繹邏輯的基礎**”——通常人們會提出並嘗試著去回答這種問題：“這件特定的工作能夠由一個計算元件來完成嗎？”，或者是“任何能夠完成某種特殊功能的演繹邏輯所須要的動作數目最小是多少？”因而發明出計算機的抽象模型，以便研究這些性質。

(4)“**演繹邏輯的分析**”——每次當一個演繹邏輯能夠被敍述時總是有理由去懷疑它的好壞。這點早在 1830 年 Charles Babbage ——計算機之父就已經知道了。一個演繹邏輯的動作模型或者說是性能輪廓可根據測量其計算時間以及演繹邏輯在處理時消耗的空間所決定。典型的問題包括最長及平均時間以及其發生次數的頻繁與否等。

我們可以了解計算機科學在這個定義下，“演繹邏輯”是一個最基本的概念。因此它值得我們給它一個精確的定義。字典上的定義：“任何機械性或者重覆性的

計算程序”並不足以完全描述它，因爲這些名詞還不是最基本的。

定　義：一個演繹邏輯是有限數目的一組指令，跟著它我們可以完成一件特定的工作。除此以外任何一個演繹邏輯必須合乎下列標準：

(1)輸入：沒有或者有一個或一個以上由外界供應的輸入。

(2)輸出：至少會產生一個輸出。

(3)清晰度：每個指令必須很清楚且不曖昧。

(4)限制性：假如我們順著一個演繹邏輯的指令去執行，那麼在所有的情況下這個演繹邏輯將在有限的步驟以後結束。

(5)有效性：每個指令必須足夠基本，大體上可以由一個人用一枝鉛筆和一張紙將之實現。每個指令不只是如(3)的被定義而已，它們還必須是可執行的。

在正式的計算機科學裏，我們將演繹邏輯與程式（Program）作一個辨別。一個程式不一定要合乎第(4)項。這種程式的一個重要例子如計算機的作業系統（Operating System），除了計算機損壞或是關閉它是永遠不終止的，而且一直在一個等待迴路（wait loop）直到有工作（job）進來爲止。在本書中我們將嚴格的只處理那些總是會結束的程式。因此這些術語將是通用的。

一個演繹邏輯可以很多種方法來描述。一種是自然的語言比如說英文能被應用在此，但是我們必須注意造成的指令必須是很清晰的（標準(3)）。英文的一種改進方式是將之與如流程圖之類的圖型記號合用。這種方式將每個處理步驟置於一個“方塊”中，並用箭頭指出下一個步驟。不同形狀的方塊代表不同種類的作業。這一切我們可以在圖 1.1 中一個自自動販賣機中購買可口可樂的流程圖中看出。主要的重點在於演繹邏輯能由許多普通的動作所發展出來。

你是否學過流程圖？那麼你可能已經了解那畢竟不是一種演繹邏輯，到底它還缺少那些特性呢？

回到我們早先對計算機科學的定義，我們發現那是非常不適合的，因爲它並沒有告訴我們爲什麼計算機可以對我們的社會產生一種革命或者爲什麼它已經使我們重新再檢討某些關於我們在宇宙是佔了何種角色的假設等。然而這點在定義上即使從技術性的觀點來看都是太過份的要求而且極不適合。這個定義對於演繹邏輯的觀念作了一個強調，但是並未提到“資料”這兩個字。假如一個計算機僅是一種從頭

到尾的方法，那麼這個方法可稱爲一個演繹邏輯，而其結果卻是一種資料的轉換，這也就是爲什麼我們時常聽到一個計算機被稱爲一個資料處理機器的原因。輸入原始的資料以及演繹邏輯可將之轉換爲有用而且精密的資料。所以除了說計算機科學是一種演繹邏輯的研究之外，也可以說計算機科學是一種“資料的研究”：

(1)擁有資料的機器。

(2)敍述資料處理的語言。

(3)敍述何種有用的資料可由原始資料產生的基礎。

(4)表示資料的一種結構。

資料的結構與演繹邏輯的合成之間有一種密切的關係。事實上資料結構與演繹邏輯可被視爲一體，少掉一個另一個將變得沒有意義。舉例來說假設我們有一個串列（ list ）包含 n 對姓名和電話號碼（ a_1,b_1 ）（ a_2,b_2 ），……（ a_n,b_n ），且我們想寫一個程式，當給它任一個姓名時即會印出那個人的電話號碼。這件工作叫做搜尋（ Searching ）。我們所寫的這個演繹邏輯與姓名與電話號碼貯存的方式或結構正具有典型的關係。一種演繹邏輯可能只是一個一個的檢查姓名，a_1,a_2,a_3，……等等，直到相符的姓名被找到。這種方法在 Oshkosh 可能很理想，但是在 Los Angeles 成千上萬的姓名中這種方法是不實際的。然而假設我們知道資料是根據英文字母的次序排列的，那麼將可以做得好一點。我們可以列出第二個串列將每個姓名依第一個字母的次序排列出來。對一個姓名比如說以 S 開頭，我們可以避免搜尋其它不以 S 開頭的姓名。因爲這個新的結構，一種截然不同的演繹邏輯是可能的。其他各種演繹邏輯的想法在了解可將資料按照我們的想法組織以後是可能的。我們將在第七章和第九章中討論更多搜尋的策略。

因此，計算機科學可以被定義爲資料的研究，並將之描述與轉換至一數位計算機上。本書的目標在探索許多不同種類的資料集（ data objects ）。對各種資料集，我們考慮各種從事的作業以及表示這個資料集使之能有效實現的方法。這暗示著兩種精練的技巧：發明資料其他表示方式的能力以及分析在此結構下作業的演繹邏輯的能力。我們選擇的教學方式是考慮一些經常在計算機應用上發生的問題。對於每一個問題我們將描述其資料集以及到底是要完成什麼等。當我們決定了由何種方式表示我們的資料集以後，我們將提供一個完整的演繹邏輯，並分析其所需的計算時間。讀通了六、七個這種例子以後你將有足夠的自信自己試著做一個。

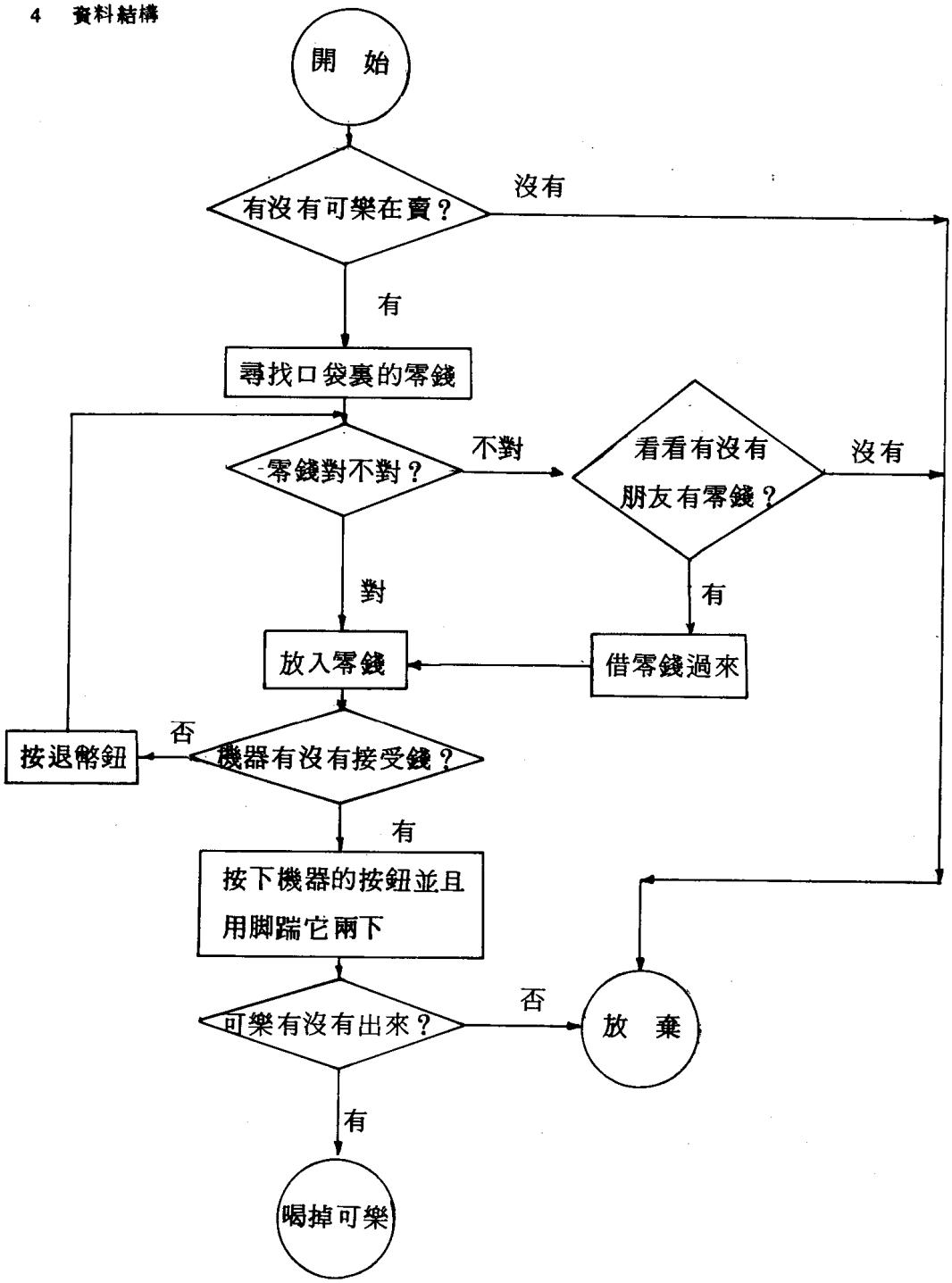

圖 1.1 買可口可樂的流程圖

在我們著手以前有一些名詞必須先小心的被定義出來。這些名詞包括資料結構，資料集，資料型式以及資料表示等。這四個名詞在計算機科學界並沒有標準的定義，而且它們經常可以交互轉換使用。

“資料型式（data type）”是一個與某些變數在程式語言中可以“不變”（hold）的資料有關的名詞。在FORTRAN中資料型式是INTEGER，REAL，LOGICAL，COMPLEX，以及DOUBLE PRECISION。在PL/I中資料型式是CHARACTER。SNOBOL的基本資料型式是字母串而在LISP中則爲串列（或是S—表示法）。各種程式語言都有一組內藏的資料型式。這表示語言允許變數被稱爲那種型式的資料並提供一組運算以便處理這些變數。某些資料型式可以很容易的供應，因爲它們已經被設定在計算機的機械語言（machine language）指令群之中。整數與實數的數學運算即爲一例。其他的資料型式須要較多努力方能實現。某些語言允許我們建立內藏資料型式的組合。在COBOL和PL/I中這種特性稱爲STRUCTURE而在PASCAL中稱爲RECORD。無論如何並不需要眞的具有這種構造。我們可以在此看到所有的資料結構都可以適度的以一種傳統的程式語言建立起來。

資料集D就是指構成D的元素所成的集合。舉例來說資料集“整數”，即指D＝{0，±1，±2，…}而言。資料集“英文字母字串且長度小於三十一”則代表D＝(″，'A'，'B'，…，'Z'，'AA'，…}。因此D可以是有限或是無限的。若D很大的話我們可能須要發展一種特殊的方法以便在我們的計算機中代表它的各個元素。

資料結構的觀念與資料集的差別在於我們不只是想敍述各個資料集而已，且更要表示出它們之間的關係。換句話說，我們想敍述一組運算，利用它我們可以合理的用在資料集的各個元素上。另一方面，我們必須說明各組運算並顯示它們如何工作。對整數而言我們具有一些算術運算如＋，－，＊，／，以及其他如“模（mod）”，“階乘”，“大於”，“小於”等很嚴密的構成了一個資料結構的定義。

詳細一點來說，我們可以看看一個適當的例子。假設我們要定義一個正整數的資料結構（簡稱NATNO）其中NATNO＝{0，1，2，3，…}包含兩個運算：測驗是否爲零或者加法。我們可以使用下列的表示法：