



Algorithm 筆記

Algorithm 快速上手

作者: sheng0603

時間: April 3, 2022

版本: 1.0



所有偉大的事，都是因為堅持才得以實現。—聖凱薩琳 Catherine of Siena

目錄

| | |
|---|-----------|
| 第 1 章 User Manual | 1 |
| 1.1 使用說明 | 1 |
| 1.2 購買筆記的好處 | 1 |
| 第 2 章 The Role of Algorithms in Computing | 3 |
| 2.1 演算法的定義 | 3 |
| 2.2 演算法對電腦硬體執行程式，效率上的重大影響 | 3 |
| 2.3 函數的執行時間比較 | 4 |
| 第 3 章 Getting Started | 5 |
| 3.1 先行知識 | 5 |
| 3.2 證明與分析演算法的正確性與運行時間的方法 | 6 |
| 3.2.1 證明與分析 Insertion-Sort 的正確性與運行時間 (running time) | 6 |
| 3.2.2 證明與分析 Merge-Sort 的正確性與運行時間 (running time) | 8 |
| 第 4 章 Growth of Functions | 13 |
| 4.1 Asymptotic notation (漸進符號) | 13 |
| 4.2 漸進符號的基礎知識與性質 | 13 |
| 第 5 章 Divide-and-Conquer | 15 |
| 5.1 基礎知識 | 15 |
| 5.2 代換法 (The substitution method) | 16 |
| 5.2.1 範例：證明 $T(n) = 2T(\lfloor n/2 \rfloor) + n$, where $T(1) = 1$ 的 $T(n) = O(n \lg n)$ | 17 |
| 5.2.2 範例：證明 $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$ 的 $T(n) = O(n)$ | 17 |
| 5.2.3 範例：證明 $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$ 的 $T(n) = O(\lg n \lg \lg n)$ | 17 |
| 5.3 遞迴樹法 (The recursion-tree method) | 18 |
| 5.3.1 範例：證明 $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$ 的 $T(n) = O(n^2)$ | 18 |
| 5.3.2 範例：證明 $T(n) = T(n/3) + T(2n/3) + O(n)$ 的 $T(n) = O(n \log_2 n)$ | 20 |
| 5.4 大師法 (The master method) | 21 |
| 5.4.1 範例： $T(n) = 9T(n/3) + n$, $T(n) = \Theta(?)$ | 21 |
| 5.4.2 範例： $T(n) = T(2n/3) + 1$, $T(n) = \Theta(?)$ | 21 |
| 5.4.3 範例： $T(n) = 3T(n/4) + n \log_2 n$, $T(n) = \Theta(?)$ | 22 |
| 5.4.4 範例： $T(n) = 2T(n/2) + n \log_2 n$, $T(n) = \Theta(?)$ | 22 |
| 5.4.5 範例： $T(n) = 2T(n/2) + \Theta(n)$, $T(n) = \Theta(?)$ | 22 |
| 5.4.6 範例： $T(n) = 8T(n/2) + \Theta(n)$, $T(n) = \Theta(?)$ | 22 |

| | |
|---|-----------|
| 5.4.7 範例: $T(n) = 7T(n/2) + \Theta(n)$, $T(n) = \Theta(?)$ | 22 |
| 第 6 章 Dynamic Programming | 23 |
| 6.1 先行知識 | 23 |
| 6.2 動態規劃 (Dynamic Programming) | 24 |
| 6.3 Rod-cutting Problem | 25 |
| 6.4 Matrix-chain Multiplication Problem | 28 |
| 6.5 Elements of dynamic programming | 32 |
| 6.6 Longest Common Subsequence | 33 |
| 6.7 Optimal Binary Search Trees Problem | 36 |
| 第 7 章 Greedy Algorithms | 39 |
| 7.1 基礎知識 | 39 |
| 7.2 An activity-selection problem | 39 |
| 7.3 Elements of the greedy strategy | 43 |
| 7.4 Huffman codes | 45 |
| 第 8 章 NP-Completeness | 47 |
| 8.1 先行知識 | 47 |
| 8.2 NP-completeness proofs 範例 | 49 |
| 8.2.1 $SAT \leq_p 3\text{-CNF-SAT}$ | 49 |
| 8.2.2 $3\text{-CNF-SAT} \leq_p \text{Clique}$ | 50 |
| 8.2.3 $\text{Clique} \leq_p \text{Vertex-Cover}$ | 51 |
| 參考文獻 | 54 |

第 1 章

User Manual

本章學習重點

□ 使用說明

1.1 使用說明

- 筆記是根據『台灣大學電資學院』的演算法用書(書名: Introduction to algorithms, 作者: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein), 所寫成的筆記。
- 筆記目的: 幫助快速恢復演算法到熟悉狀態。(遺忘是正常的!)
- 筆記章節: 參照 Introduction to algorithms 的章節名稱。
- 重新排版演算法, 增加清晰度, 避免不必要的誤解。(參考圖 1.1 與圖 1.2。)

1.2 購買筆記的好處

- 精美的排版。(參考圖 1.3, 圖 1.4, 圖 1.5。)
- 詳細的推導過程。(參考圖 1.6, 圖 1.7。)
- 終生保固。(只要購買一次, 之後如有更新或修正內容, 都可透過 e-mail 免費獲得更新內容。)

```
PRINT-OPTIMAL-PARENS( $s, i, j$ )
1  if  $i == j$ 
2    print " $A_i$ "
3  else print "("
4    PRINT-OPTIMAL-PARENS( $s, i, s[i, j]$ )
5    PRINT-OPTIMAL-PARENS( $s, s[i, j] + 1, j$ )
6    print ")"
```

圖 1.1: 書本排版。

Algorithm 6.8: Print-Optimal-Parens(s, i, j)

```
1 Print-Optimal-Parens( $s, i, j$ )
2 if  $i == j$  then
3   print " $A_i$ ";
4 else
5   print "(";
6   Print-Optimal-Parens( $s, i, s[i, j]$ );
7   Print-Optimal-Parens( $s, s[i, j] + 1, j$ );
8   print ")";
9 end
```

圖 1.2: 筆記排版。

定義 2.1 (演算法定義)

什麼是演算法: (What are algorithms?)

定義 1. 演算法是將輸入轉換為輸出的一系列有序計算步驟。

(原文: An algorithm is a sequence of computational steps that transform the input into the output.)

定義 2. 演算法是任何定義明確的計算過程，它將某個值或一組值作為輸入，並產生一些值或一組值作為輸出。

(原文: An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.)

補充說明 1. 考試，使用哪個定義都可以，但通常只有 2 分，建議使用簡潔的定義 1，即
可拿分，多的時間攻略大分題。

補充說明 2. 英文好的同學，也可直接寫英文定義。

圖 1.3: 精美排版: 定義

定理 5.1 (大師法 (Master theorem))

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$.

Then $T(n)$ has the following asymptotic bounds:

case 1. If $f(n) = O(n^{(\log_b a)-\epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

快速記法：即 $f(n) < n^{\log_b a}$ 時套用。

case 2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$.

快速記法：即 $f(n) = n^{\log_b a}$ 時套用。

case 3. If $f(n) = \Omega(n^{(\log_b a)+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$.

快速記法：即 $f(n) > n^{\log_b a}$ 時套用。

◆ 頓外補充： $f(n) = \Theta(n^{\log_b a} \log_2^k n)$, where $k \geq 0$, 則 $T(n) = \Theta(n^{\log_b a} \log_2^{k+1} n)$.

圖 1.4: 精美排版: 定裡

題型 6.2 (Matrix-Chain Multiplication Problem)

給定一個矩陣序列(鏈) (A_1, A_2, \dots, A_n) ，包含 n 個矩陣，其中 $i = 1, 2, \dots, n$ ，矩陣 A_i 的維度為 $p_{i-1} \times p_i$ ，經某種括號方式獲得 $A_1 A_2 \dots A_n$ 乘積時，可以最小化使用乘法的次數。

(原文) Given a sequence (chain) (A_1, A_2, \dots, A_n) of n matrices, where for $i = 1, 2, \dots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \dots A_n$ in a way that minimizes the number of scalar multiplications.)

例子：

$$\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \end{bmatrix} \right) \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 & 1 \cdot 2 \\ 2 \cdot 1 & 2 \cdot 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 3 & 1 \cdot 2 + 2 \cdot 4 \\ 2 \cdot 1 + 4 \cdot 3 & 2 \cdot 2 + 4 \cdot 3 \end{bmatrix}$$

使用 12 個乘法，但

$$\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix} \times \left(\begin{bmatrix} 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right) \right) = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 1 \cdot 1 + 2 \cdot 3 & 1 \cdot 2 + 2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 7 & 1 \cdot 10 \\ 2 \cdot 7 & 2 \cdot 10 \end{bmatrix}$$

只使用 8 個乘法。

圖 1.5: 精美排版: 題型

$$\therefore T(n) \leq dn \log_2 n$$

$$\therefore T(n) = T(n/3) + T(2n/3) + O(n)$$

$$\leq T(n/3) + T(2n/3) + cn$$

$$\leq d(n/3) \log_2(n/3) + d(2n/3) \log_2(2n/3) + cn$$

$$= (dn/3)(\log_2 n - \log_2 3 + 2 \log_2 2n - 2 \log_2 3)$$

$$= (dn/3)(\log_2 n - \log_2 3 + 2 \log_2 2 + 2 \log_2 n - 2 \log_2 3)$$

$$= (dn/3)(3 \log_2 n - 3 \log_2 3 + 2) + cn$$

$$= (dn)(\log_2 n - \log_2 3 + 2/3) + cn$$

$$\leq dn \log_2 n, \text{ 取 } d \geq c/(\log_2 3 - 2/3), \text{ 故得證}$$

$$\therefore T(n) = 3T(n/4) + cn^2, c > 0$$

$$= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2cn^2 + \dots + (\frac{3}{16})^{\log_4 n-1}cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4 n-1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$< \sum_{i=0}^{\infty} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{1}{1 - (3/16)} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2), \text{ 故得證}$$

圖 1.6: 詳細的推導過程。

圖 1.7: 詳細的推導過程。

第 2 章

The Role of Algorithms in Computing

本章學習重點

- 演算法的定義
- 了解演算法對電腦硬體執行程式，效率上的重大影響

2.1 演算法的定義

定義 2.1 (演算法定義)

什麼是演算法: (What are algorithms?)

定義 1. 演算法是將輸入轉換為輸出的一系列有序計算步驟。

(原文: An algorithm is a sequence of computational steps that transform the input into the output.)

定義 2. 演算法是任何定義明確的計算過程，它將某個值或一組值作為輸入，並產生一些值或一組值作為輸出。

(原文: An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.)

補充說明 1. 考試，使用哪個定義都可以，但通常只有 2 分，建議使用簡潔的定義 1，即可拿分，多的時間攻略大分題。

補充說明 2. 英文好的同學，也可直接寫英文定義。



2.2 演算法對電腦硬體執行程式，效率上的重大影響

為解決同一個問題，而設計的不同演算法，在效率上往往差異很大。這些差異可能比硬體和軟體造成的差異更為顯著。

表 2.1: 電腦 A 與電腦 B，的硬體參數與所使用的演算法

| 名稱 | input size | 硬體效能 | 使用的演算法 | 時間複雜度 |
|------|------------------------|--------------------------------|----------------|----------------|
| 電腦 A | 10 million ($=10^7$) | 10 billion ($=10^{10}$) 指令/秒 | insertion-sort | $2n^2$ |
| 電腦 B | 10 million ($=10^7$) | 10 million ($=10^7$) 指令/秒 | merge-sort | $50n \log_2 n$ |

以表 2.1 為例子，電腦 A 的硬體效能，每秒能執行 10^{10} 個指令，電腦 B 的硬體效能，每秒能執行 10^7 個指令，所以硬體效能方面，電腦 A 比電腦 B 快 1000 倍。

2.3 函數的執行時間比較

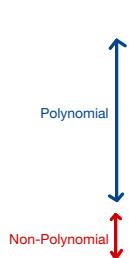
但電腦 A 採用 insertion-sort 演算法，所以電腦 A 花費的時間 = $\frac{\text{執行 insertion-sort 所需指令個數(即時間複雜度)}}{\text{電腦 A 的硬體效能}}$
 $= \frac{2n^2 \text{指令}}{10^{10} \text{指令/秒}} = \frac{2(10^7)^2 \text{指令}}{10^{10} \text{指令/秒}} = 20,000 \text{秒(約 5.5 個小時)}$ 。電腦 B 雖然較電腦 A 慢，但採用 merge-sort 演算法，電腦 B 花費的時間 = $\frac{\text{執行 merge-sort 所需指令個數(即時間複雜度)}}{\text{電腦 B 的硬體效能}} = \frac{50n \log_2 n \text{指令}}{10^7 \text{指令/秒}} = \frac{50 \cdot 10^7 \log_2 10^7 \text{指令}}{10^7 \text{指令/秒}} = 1163 \text{秒(約 20 個分鐘)}$ 。(其中， $n = 10^7$ 為 input size，即所需排序的個數。)

由此觀之，雖然電腦 B 的硬體效能雖然比電腦 A 慢 1000 倍，但採用較優秀的演算法，最後，執行時間却反比電腦 A 快 17 倍，由此可見優秀的演算法對電腦硬體的執行效率，著實有重大影響。

尤其，當 input size $n = 100$ million numbers，採用 insertion-sort 演算法的電腦 A 需花費 23 天，但採用 merge-sort 演算法的電腦 B 僅需花費 4 個小時。再次得證，演算法對電腦硬體執行效率的影響，不可謂之不甚。

2.3 函數的執行時間比較

- $\lg n < \sqrt{n} < n < n \lg n < n^2 < n^3 < 2^n < n!.$ (參考圖 2.1)
- Non-Polynomial functions (ex: 2^n or $n!$) 只要 n 稍微大一點 (ex: 51 or 17)，就需要花費執行一個世紀時間。
- 由此可知，設計演算法的執行時間是 Polynomial time (ex: $n \lg n$) 會更有效率。



| | 1 Second | 1 Minute | 1 Hour | 1 Day | 1 Month | 1 Year | 1 Century |
|------------|---------------------|----------------------|-----------------------|---------------------------|----------------------------|-----------------------------|------------------------------|
| $\lg n$ | $2^{1 \times 10^6}$ | $2^{6 \times 10^7}$ | $2^{3.6 \times 10^9}$ | $2^{8.64 \times 10^{10}}$ | $2^{2.592 \times 10^{12}}$ | $2^{3.1536 \times 10^{13}}$ | $2^{3.15576 \times 10^{15}}$ |
| \sqrt{n} | 1×10^{12} | 3.6×10^{15} | 1.29×10^{19} | 7.46×10^{21} | 6.72×10^{24} | 9.95×10^{26} | 9.96×10^{30} |
| n | 1×10^6 | 6×10^7 | 3.6×10^9 | 8.64×10^{10} | 2.59×10^{12} | 3.15×10^{13} | 3.16×10^{15} |
| $n \lg n$ | 62746 | 2801417 | 133378058 | 2755147513 | 71870856404 | 797633893349 | 6.86×10^{13} |
| n^2 | 1000 | 7745 | 60000 | 293938 | 1609968 | 5615692 | 56176151 |
| n^3 | 100 | 391 | 1532 | 4420 | 13736 | 31593 | 146679 |
| 2^n | 19 | 25 | 31 | 36 | 41 | 44 | 51 |
| $n!$ | 9 | 11 | 12 | 13 | 15 | 16 | 17 |

圖 2.1: Comparison of running times