Complexity

by Chin-Huang Lin

Sprout

在之前的投影片中……

- 我們提到資訊之芽會教你……
  - 如何比較兩個演算法，哪一個比較「好」？
- 問題是：什麼叫做好？

對一個程式來說……

- 如果……，你會覺得好
  - 時間花得少 (運行速度)
    - ex. C_r_m_ vs I_t_rn_t Ex_lo_e_
  - 空間花得少
    - ex. Fi_ef_x vs Ch_o_e
  - 正確率較高
    - ex. B_td_f_nd_r vs M_Af_e
  - 相容性較高
    - ex. W_nd_ws vs L_n_x
  - 開發成本低
- ♪ 平衡是很重要的事情
  - ♪ 美國進行人口普查，在沒有電腦前每10年進行一次，完成一次的統計需要7.5年 → 等你查完小孩子都長大了
  - ♪ 如果今天只是要知道人口的分佈，其實抽查的實用性會高得多
  - ♪ 有了讀卡機（最早的電腦）之後，6周即可統計完成

怎麼衡量？

- 實測通常最精準！
- 但，怎麼預測？
- 相容性、開發成本通常仰賴實做方式與實做者評估
- 正確率有很多面向
  - 實做細節
  - Case 數量
  - 理論分析
- 時間、空間？

一些瓶頸

- 每種操作所需時間略有差異
  - ex. 除法比加法慢、記憶體存取根據區塊不同也有差異
- 小時候胖不是胖
  - ex. $3x^2 < 10x \text{ if } x < 4$
  - 通常輸入規模很小的時候，時間的差異很小！
  - 我們在意輸入規模很大的時候的情形

複雜度的概念

- 乘法比加法慢，但……
  - 老師出長期作業，有兩種方案
    1. 每天算 10 題乘法
    2. 第一天算 1 題加法，第二天算 2 題加法，第三天算 3 題加法……
  - 你會選哪一種作業？
  - 假設算一題加法需要 5 秒，算一題乘法需要 20 秒
    - 到了第 n 天，方案 1 花了 $10 * n * 20 = 200n$ 的時間寫作業
    - 到了第 n 天，方案 2 花了 $(n+1) * n/2 * 5 = 2.5(n^2 + n)$ 的時間寫作業
    - 如果 n 很小，例如 $n = 3$，那麼方案 2 比較好
    - 如果 n 很大，例如 $n = 365$ 呢？

  → 方案 2 需要的操作數的「量級」比較多 (複雜)，所以長期作業應該選方案 1 比較輕鬆

量級？

- 以動物為例……
- 簡單來說，他們「成長」的速度是不同等級的！
- 極限的概念 1
  - 對於兩個函數 $f(n)$ 與 $g(n)$，當 n 趨近於無窮大，誰會比較大？
- 超級比一比：
  - $3n^2 + n + 20$ vs $100n$
  - n^{100} vs 2^n
  - n^2 vs $10\text{nlog}\,n$
  - 100^n vs $n!$
  - $30 * 2^n$ vs 3^n
  - $100n$ vs $200n$

$100n$ *vs* $200n$?

- 極限的概念 2
  - 如果 $\frac{f(n)}{g(n)}$ 在 n 趨近於無限大時趨近於 0，那麼我們說 $f(n)$ 比 $g(n)$ 小
  - 如果 $\frac{f(n)}{g(n)}$ 在 n 趨近於無限大時趨近於無限大，那麼我們說 $f(n)$ 比 $g(n)$ 大
  - 否則我們說兩者一樣量級
- 再次超級比一比
  - $3n^2 + n + 20$ vs $100n$
  - n^{100} vs 2^n
  - n^2 vs $10\text{nlog}\,n$
  - 100^n vs $n!$
  - $30 * 2^n$ vs 3^n
  - $100n$ vs $200n$
- 量級的實務意義：科技的進步到底有沒有辦法彌補兩者的差異？

常見量級比較

- 在一台每秒可進行10億次運算的電腦上的話……

n	n	$n\log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	$.01\mu s$	$.03\mu s$	$.1\mu s$	$1\mu s$	$10\mu s$	$10s$	$1\mu s$
20	$.02\mu s$	$.09\mu s$	$.4\mu s$	$8\mu s$	$160\mu s$	$2.84h$	$1ms$
30	$.03\mu s$	$.15\mu s$	$.9\mu s$	$27\mu s$	$810\mu s$	$6.83d$	$1s$
40	$.04\mu s$	$.21\mu s$	$1.6\mu s$	$64\mu s$	$2.56ms$	$121d$	$18m$
50	$.05\mu s$	$.28\mu s$	$2.5\mu s$	$125\mu s$	$6.25ms$	$3.1y$	$13d$
100	$.10\mu s$	$.66\mu s$	$10\mu s$	$1ms$	$100ms$	$3171y$	$4 * 10^{13}y$
10^3	$1\mu s$	$9.96\mu s$	$1ms$	$1s$	$16.67m$	$3.17 * 10^{13}y$	$32 * 10^{283}y$
10^4	$10\mu s$	$130\mu s$	$100ms$	$16.67m$	$115.7d$	$3.17 * 10^{23}y$	
10^5	$100\mu s$	$1.66ms$	$10s$	$11.57d$	$3171y$	$3.17 * 10^{33}y$	
10^6	$1ms$	$19.92ms$	$16.67m$	$31.71y$	$3.17 * 10^{7}y$	$3.17 * 10^{43}y$	

複雜度的概念

- 我們假設所有的操作都需要花費一樣的時間！包括
  - 加減乘除
  - 取餘數
  - 位運算
  - 存取記憶體
  - 判斷運算子
  - 賦值運算子
  - ……
- 把所有操作需要的「次數」都計算出來
- 看看量級的大小（複雜度），並且評估執行時間
  - ex. $3n^2 + 2n + 7$，$n = 2000$，運行時間應該是 12004007 的常數倍
- 目前計算機的運算速度約為每秒 2000 萬~8000 萬次運算

一些符號

- 每次都要列出式子，很麻煩
- 常數其實不重要！
- 大歐符號 (Big-O)
  - 記為 $O(f(n))$
  - 其中 $f(n)$ 是複雜度量級的上界
  - 假設實際運行的時間為 $g(n)$，那麼在 n 趨近於無限大時，有 $\frac{f(n)}{g(n)}$ 趨近於某常數或者 $\frac{f(n)}{g(n)}$ 趨近於無限大
  - ex. $3n^3 + 5n^2 + 10n + 3 \in O(n^3)$
  - ex. $f(n) \in O(n^2)$，則 $f(n) \in O(n^3)$
  - ex. $1000 \in O(1)$
  - 通常會取最緊的上界

練習時間

- $3n^2 + n + 20$ vs $100n$
  - $O(n^2)$ vs $O(n)$
- n^{100} vs 2^n
  - $O(n^{100})$ vs $O(2^n)$
- n^2 vs $10\text{nlog}\,n$
  - $O(n^2)$ vs $O(\text{nlog}\,\text{n})$
- 100^n vs $n!$
  - $O(100^n)$ vs $O(n!)$
- $30 * 2^n$ vs 3^n
  - $O(2^n)$ vs $O(3^n)$
- $100n$ vs $200n$
  - $O(n)$ vs $O(n)$

其他符號

- 小歐符號 (Little-o)
  - 記為 $o(f(n))$
  - 其中 $f(n)$ 為複雜度量級的**嚴格**上界
  - 假設實際運行的時間為 $g(n)$，那麼在 n 趨近於無限大時，$\frac{f(n)}{g(n)}$ 趨近於無限大
  - ex. $3n^3 + 5n^2 + 10n + 3 \notin o(n^3)$
  - ex. $3n^3 + 5n^2 + 10n + 3 \in o(n^4)$
  - ex. $f(n) \in o(n^2)$，則 $f(n) \in o(n^3)$

其他符號

- Big-omega
  - 記為 $\Omega(f(n))$
  - 其中 $f(n)$ 為複雜度量級的下界
  - ex. $3n^3 + 5n^2 + 10n + 3 \in \Omega(n^3)$
  - ex. $f(n) \in \Omega(n^2)$，則 $f(n) \in \Omega(n)$
- Little-omega
  - 記為 $\omega(f(n))$
  - 其中 $f(n)$ 為複雜度量級的**嚴格**下界
  - ex. $3n^3 + 5n^2 + 10n + 3 \in \omega(n^2)$
  - ex. $3n^3 + 5n^2 + 10n + 3 \notin \omega(n^3)$

其他符號

- Big-theta
  - 記為 $\Theta(f(n))$
  - 其中 $f(n)$ 為複雜度量級的嚴格上下界（完全相同！）
  - ex. $3n^3 + 5n^2 + 10n + 3 \in \Theta(n^3)$
  - ex. $3n^3 + 5n^2 + 10n + 3 \notin \Theta(n^4)$
  - ex. $3n^3 + 5n^2 + 10n + 3 \notin \Theta(n^2)$
- 一般來說（至少在 2014 資訊之芽），我們最常用到 Big-O 和 Big-theta

如果時間不穩定？

- 通常我們不能夠容忍最壞情況非常糟，儘管出現機率不高
- 多數情況我們總是關注算法中最壞的 case
- ex. 插入排序法
  - 1. 從 $1\sim n$ 依序取出當前第 i 個元素
  - 2. 把當前元素往前移動到正確的位置
  - ex. 3 2 1 4
    - 1. 3 2 1 4
    - 2. 2 3 1 4
    - 3. 2 1 3 4
    - 4. 1 2 3 4
    - 5. 1 2 3 4
  - best case: 一開始就排好，$O(n)$
  - worst case: 完全逆序 $(n, n-1, \ldots, 1)$，$O(n^2)$

想一想.....

- 複雜度的全名實際上是「漸近複雜度」，你覺得是為什麼呢？(提示：跟漸近線有點關係)
- 複雜度評估有什麼關鍵的壞處？總是選擇複雜度低的方式就一定好嗎？
- 上次上課時介紹了質數的檢測方式以及優化。試著分析一下複雜度各是多少吧！(篩法除外，數學成份過重)
- $O\big(f(n)+g(n)\big)=O(\max\big(f(n),g(n)\big))$，其中 max 函數會取其中量級較大的函數。這是為什麼呢？
- 若 $g1(n)=O\big(f1(n)\big), g2(n)=O(f2(n))$，那麼 $g1(n)*g2(n)=O(f1(n)*f2(n))$，這是為什麼呢？
- 實務中，如果一個算法空間複雜度很高但時間複雜度較低，另一個算法時間複雜度很高但空間複雜度較低，哪一個算法在小範圍的時候比較實際？哪一個算法大範圍的時候比較實際？為什麼？