Ch3. P 與 NP 問題

演算法裡有一個著名的「P 與 NP 問題」。

本章的介紹與討論，在「學術研究」或「考研究所」時較重要，實務上則較少用到。

本章大綱

A. 問題的分類
B. P 與 NP 問題
C. NP-hard
D. NP-complete（NPC）
E. 總結演算法問題

首先，會談一下在演算法的範疇中問題該怎麼區分，再來分別討論 P 與 NP 問題、NP-hard、NP-complete。

1. 演算法問題的分類

大抵而言，電腦科學中的「問題」可以分成兩種。

第一種是「決策問題 Decision Problem」，這類問題只需回答「是 Yes」或「不是 No」：若回答「是」，則隨後給出滿足要求的解；回答「不是」，則給出相對應的證明。

「100 到 110 是不是一個含有質數的區間」就是一個這樣的「決策問題」，如果回答「是」，就要給出這個區間中的任一質數（101、103、107、109），回答不是時，則要有相應的證明。

另外一種問題是「最佳化問題 Optimization Problem」，也就是在有限種候選答案之中選出最佳解。通常，最佳化問題比決策問題來得更難解決。

比如說，一家公司中有許多員工，每個員工各自有某幾天想上班、某幾天不想上班，這時候如何盡量滿足員工意願排出班表呢？這就是一個「最佳化問題」，

因為排班的方式是「有限的」，而在這有限個選項中選出最佳者，就是一個最佳化問題。

雖然要求候選答案數量必須是「有限的」，但數目仍可能遠遠超過想像，比如下圍棋時，想在符合規則的下法中得到一個最佳解，也屬於最佳化問題，然而解的總數高達 10^{171}，因此找到最佳解仍然相當困難。

另外，找出最佳交通路徑也是一個最佳化問題的例子，這與後續章節中「如何尋找最短路徑」的問題相關。

（1）典型的決策問題

A. 分割問題 Partition Problem

給定一個正整數的集合，是否可將其分成兩個子集合，並使兩子集合中的數字總和相等？

當 $S = \{1, 2, 4, 5, 6\}$ 時，答案為「可以」，因為此時可以將其分成兩集合 $S_1 = \{1, 2, 6\}$ 與 $S_2 = \{4, 5\}$。

因為對於一個給定的集合 S，只需根據問題要求回答「是」或「否」，因此這屬於決策問題。

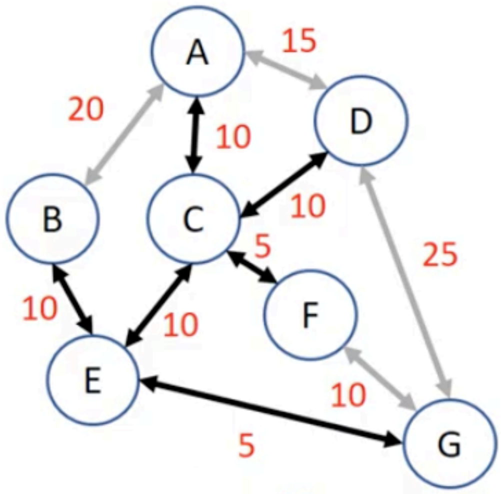
B. 部分集合的和問題 Sum of Subset Problem

給定一個正整數的集合，是否存在一子集合，其和為特定常數 C？

例如當 $S = \{1, 2, 4, 5, 6\}$、$C = 15$ 時，答案為「是」，因為可以找出一個 S 的子集合 $S_1 = \{4, 5, 6\}$ 滿足條件。

（2）典型的最佳化問題

A. 最小生成樹問題 Minimal Spanning Tree Problem

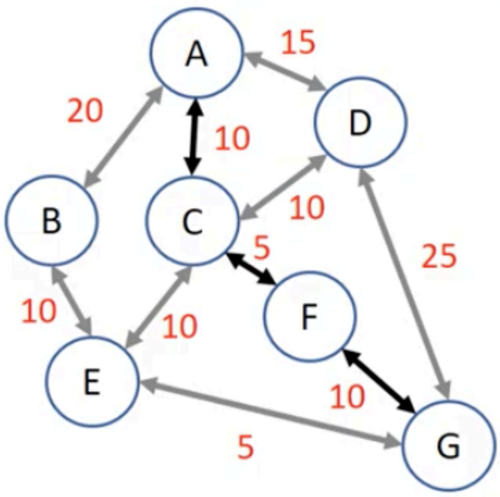


給定由頂點與邊構成的「圖」，試著從中取出部分邊與所有點讓其形成一棵樹（不會形成環），且使得該樹的權重和最小。

上面的圖中總共只有十個邊，因此選定部分邊的可能方式是有限的，也就是 C_6^{10} 種，接下來，就在這有限種的可能裡選出一個「最佳」者，這裡被選取到的邊權重和越小就越好。

後續章節會介紹相關演算法，可以得到解中的邊有 $\overline{BE}$、$\overline{EC}$、$\overline{AC}$、$\overline{CF}$、$\overline{EG}$、$\overline{CD}$。

B. 最短路徑問題 Shortest Path Problem



給定由頂點與邊構成的圖，找出兩點間的最短路徑。上圖中，A、G 間的最短路徑會依序經過 A、C、F、G。

（3）兩種問題的轉換

「最佳化問題」都可以被轉換成對應的「決策問題」。

比如最短路徑問題的最佳化版本是「給定由頂點與邊構成的圖，試著找出兩點間的最短路徑」，而決策版本如「給定由頂點與邊構成的圖，以及一個常數 C，能否找出一路徑使兩點間的最短路徑 $< C$？」，後者因為只需回答「能」或「不能」（並給出理由），所以屬於決策問題。

不過在此同時，決策問題則「不一定」可以找到與之對應的最佳化問題。

2. 問題的難度

要如何描述問題對「人」的難度呢？

有時候可能會說「這問題好難喔，沒有計算機我不會」，或者「這問題好難喔，不管有沒有計算機我都不會」，又或者「這問題好難喔，不只我，大家都不會」，但是在演算法中，有沒有更精準的敘述方式呢？

首先，要釐清問題的「難」是出在哪裡，其中一種是「問題已有一些解法，但想進一步找出最佳解法卻很困難」，另一種則是「問題本身很難找到簡單的解決方式」。

當然，可以用上一章介紹的複雜度來描述問題的難度，這樣，問題會被分為「易解的 Tractable」和「難解的 Intractable」兩種。

（1）難解的問題 Intractable Problems

難解的問題代表在最壞狀況 Worst-case 下，無法找到多項式時間的問題解法。不過目前還未找到，並不代表未來不會找到，因此一個問題是否「難解」會隨時間變動。

可以根據問題的解決難度、複雜度等定義幾種性質（注意不要把 NP 的意思誤解成「不是 P」)：

A. P（Polynomial Time）：

存在多項式時間複雜度的演算法來解決

B. NP（Nondeterministic Polynomial Time）：

存在多項式複雜度的演算法來「驗證」問題的解答是否正確

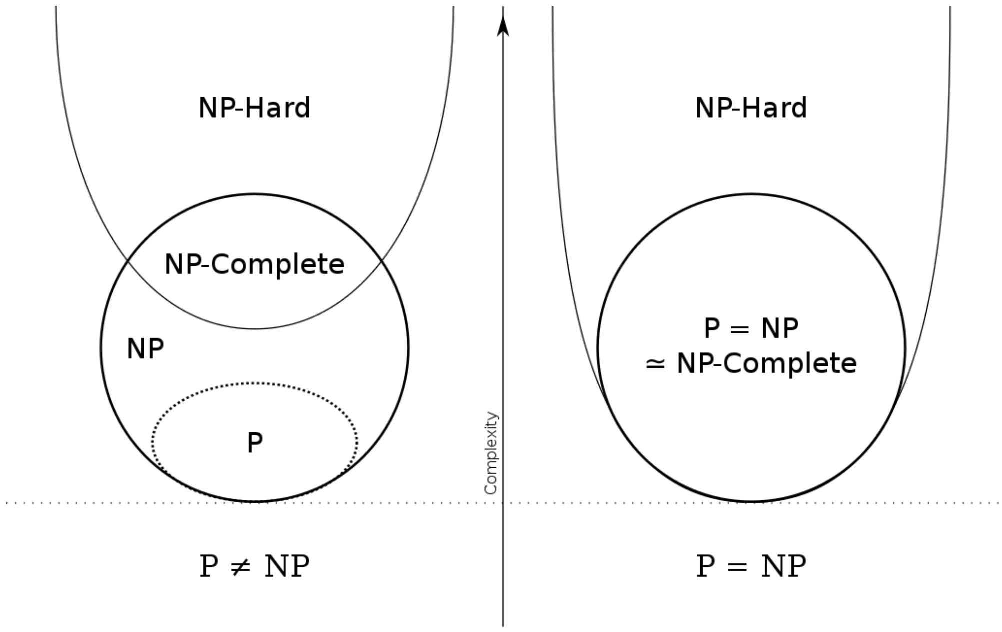
C. NP-hard：

目前還沒有找到多項式時間複雜度的算法，也不確定每一組解能否在多項式時間內被驗證

D. NP-complete（NPC）：

目前還沒有找到多項式時間複雜度的算法，但每一組解都可以被多項式時間的算法驗證

從定義上看，NP-Complete 問題是 NP-hard 問題中，屬於 NP 的那些。



因為這些性質的命名容易混淆，盡量不要從字面上理解意思，在繼續往下解釋之前，可以先用比較簡化的方式記憶這幾個性質：

A. P 問題：

容易解決的問題（所有容易解決的問題都容易驗證，所以 $P \subseteq NP$）

B. NP 問題：

答案容易驗證的問題

C. NP-hard 問題：是所有 NP 問題可以被歸約到的問題

「至少和所有 NP 問題一樣或更難」，而且可能會失去「答案容易驗證的性質」而不再屬於 NP 問題

D. NP-complete 問題：是 NP-hard 這些問題裡，答案仍然容易驗證的那些。

所以 NP-Complete（NPC） 問題全部都是 NP-hard 問題，而且也都還在 NP 範圍內

（2）比較 P 問題與 NP 問題

A. P 問題 Polynomial Time Problems

a. 問題在最壞狀況 Worst-Case 下可以被多項式時間內的算法解決

b. 可以視作「Easy to find」

B. NP 問題 Non-Deterministic Polynomial Time Problems

a. 不確定（可能可以，也可能不可以）在多項式時間內被解決

b. 最壞狀況 Worst-Case 下可以被多項式時間的算法驗證

c. 注意 NP 不是 non-polynomial time，也就是說，P 和 NP 問題並不互斥

d. 可以視作「Easy to check」

（3）P 和 NP 間的關係

每個 P 問題都一定是 NP 問題，也就是說，$P \subseteq NP$。每個可以簡單找到答案的問題，一定可以簡單的驗證，因為只要使用該種算法直接算出問題的答案，馬上可以驗證某個答案是否是對的。

也就是說，一個問題如果「可以在多項式時間內被解決」，就「可以在多項式時間內被驗證」。

	log_2n	n	$nlog_2n$	n^2	n^3	2^n	$n!$
10	3.32	10	33.22	10^2	10^3	$\sim 10^3$	3628800
10^2	6.64	10^2	664.39	10^4	10^6	$\sim 10^{30}$	$\sim 10^{158}$
10^3	9.97	10^3	9965.78	10^6	10^9	$\sim 10^{300}$	X
10^4	13.29	10^4	132877.12	10^8	10^{12}	$\sim 10^{3000}$	X

Polynomial　　Non-Polynomial

何謂多項式時間？過去在學校裡已經學過，形如 $a_0 + a_1n + a_2n^2 + a_3n^3 + \cdots + a_kn^k$ 的式子就是 n 的「多項式」。

從複雜度的角度來看，有下列關係：

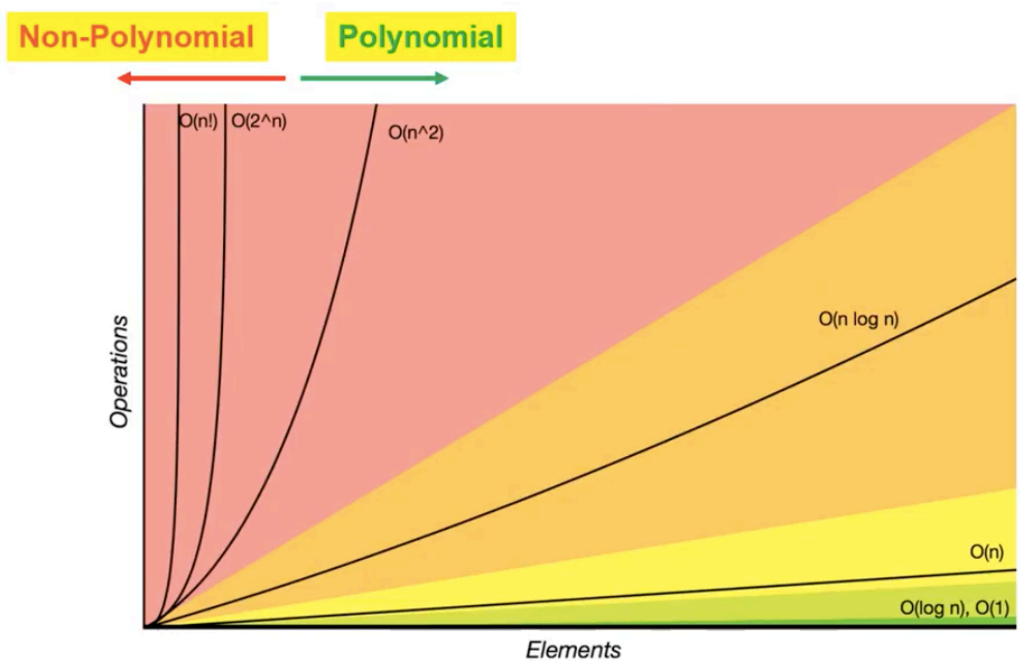
$$O(a_0 + a_1 n + a_2 n^2 + a_3 n^3 + \cdots + a_k n^k) = O(n^k)$$

不管 k 多大，只要能夠寫成 n^k 的形式，就還在多項式時間內。

為什麼 n 只能出現在「底數」，不能出現在「指數」呢？上表中，左邊的區塊（$\log_2 n$ 到 n^3 直行）屬於多項式時間 Polynomial Time，右邊兩個直行則屬於非多項式時間。

當 n 分別為 10、100、1,000、10,000 的時候，即使複雜度是 n^2 或 n^3，所需時間還勉強可以接受，但是一旦 n 出現在指數部分，總共需要的運算次數就陡增，如果是 $n!$ 則更甚。

這也就是說，如果一個問題無法在多項式時間內被解答，以目前的電腦硬體而言，就是「難解」的問題。



上面描述各種複雜度的圖中，一直到 n^2 為止都還屬於多項式時間，到了 2^n、$n!$ 這兩種常見的複雜度，就超出了多項式時間的範疇。

（4）容易驗證卻不容易得到解的問題

下面的問題就是一個這樣的問題：給定 400 位學生與一個有 100 個床位的宿舍，有一份代表「不相容」的名單，在這份名單上，學生兩兩成對，而每一對中的兩個學生都不能同時住宿在宿舍中，請給定一個分配宿舍的方式。

這個問題不屬於 P 問題，總共的分配方式有 C_{100}^{400} 種，如果名單很長，要得出符合答案的解就很困難。

但是給定任何一種解，只要按照名單依序檢查是否牴觸，就可以決定這個解是否符合條件，因此這是一個 NP 問題，也就是「容易驗證」的問題。

參考：Clay Mathematics Institute

（5）$P = NP$？

上面已經提過，所有的 P 問題都是 NP 問題，但是 P 問題的範圍是否與 NP 問題的範圍相等呢？也就是說，是否有一些問題，雖然容易驗證，但是絕對無法在多項式時間內解決呢（此時這個問題是 NP 問題且同時不是 P 問題，因此 $P \neq NP$）？

要注意的是，雖然顯然有一些問題是處於「目前容易驗證但是還沒找到容易解決的算法」，但是這些問題都還沒有被證明為「絕對不是 P 問題」，所以 P 是否與 NP 問題相等，至今仍未有定論。

如果證明 $P = NP$，這代表所有容易驗證的問題都必定可以被容易的解決，這會是個世紀大發現；相反的，如果證明 $P \neq NP$，那麼某些 NP 問題註定無法解決，同樣是非常重大的發現。

（6）千禧年世紀難題

千禧年世紀難題列出了 7 個問題，解決任何一個，都可以拿到 100 萬美金的獎金（不過對於能夠解決這些問題的人來說，100 萬美金應該是個小數目）。

「P 是否等於 NP」的問題就被列在這 7 個千禧年世紀問題之中：

A. P / NP 問題
B. 霍奇猜想
C. 黎曼猜想
D. 楊-米爾斯存在性與質量間隙
E. 納維-斯托克斯存在性與光滑性
F. 貝赫和斯維納通-戴爾猜想
G. 龐加萊猜想（已被解決）

3. NP-hard

講完 P 與 NP 問題後，再來講解 NP-hard 問題。

要瞭解 NP-hard 問題，需要先有「歸約 reduction」的概念：

（1）歸約 reduction

A. 某個計算問題 A 可以被轉換為另一個計算問題 B，寫成 $A \leq_P B$，這代表 A 問題比 B 問題簡單（或一樣難）

B. 若 B 問題有多項式時間解法，則 A 問題也同樣有多項式時間解法

歸約具有「傳遞性」，也就是說，如果問題 A 可以被歸約為問題 B，問題 B 又可以被歸約為問題 C，則問題 A 就可以被歸約為問題 C。

透過這樣的傳遞性，可以把所有的 NP 問題歸約為某幾種問題（不一定還屬於 NP 問題），這些就是所謂的 NP-hard 問題。如果這些歸約到的問題仍然具有容易驗證的特性的話，就仍然屬於 NP，而被叫做 NPC 問題。

（2）歸約的實例

如果問題 A 可以被歸約為問題 B，那麼問題 B「至少跟問題 A 一樣難」。

舉個例子：

問題 A：判斷一個正整數是否是質數
問題 B：找出一個正整數的所有因數

因為一旦解決問題 B，就可以馬上解決問題 A（只要判斷因數是否超過兩個），所以 A 的難度小於等於 B，記作 $A \leq_P B$。

從兩個問題的難度關係，可以推論出：

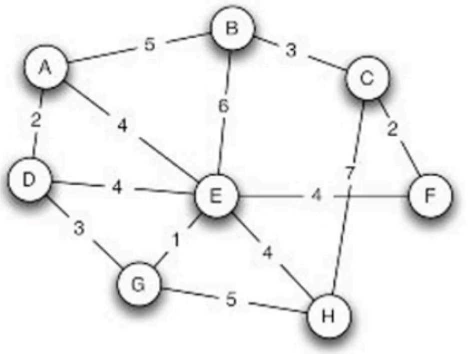
A. 如果「找出因數（問題 B）」有多項式時間的解法，則「判斷是否是質數（問題 A）」也會有多項式時間的解法

B. 如果「判斷是否是質數（問題 A）」沒有多項式時間的解法，那麼「找出因數（問題 B）」也不會有多項式時間的解法

為什麼要把問題歸約呢？這是因為演算法的問題實在太多，一一解決過於麻煩，如果所有問題都被歸約成單一問題，而又找到一個多項式時間的算法來解決它，這樣所有的演算法問題也就得到了快速的解決。

舉個例子，如果某個人相信「世界上所有的問題都只因貧富差距而生」，那麼只要解決了貧富差距，世界上就再也沒有任何問題了。

（3）NP-hard 問題的例子：旅行業務員問題 Travelling Salesman Problem



有一個業務員需要不斷地拜訪各個城市，在每次出差中，每個城市都只能經過一次，而在拜訪完所有的城市後，他還必須回到最開始的城市。給定每兩個城市間單程所需花費的時間後，找出一個路徑讓整趟旅程花費的時間最短。

當城市總數比較小時，可能的解的數目還不多，比如有 3 個城市時，正好是一個三角形，只有一個可能的解（除了起點外的另兩個城市中，先往哪個走，花費的時間都一樣），而有 5 個城市時，有 12 組可能的解，當有 10 個城市時，就有 $\frac{9!}{2} = 181{,}440$ 組可能的解了。

計算可能的解的總數：從起點往外走到第一個城市時，有 $n-1$ 種選擇，再拜訪第二個城市時，因為不經過重複的城市，所以只有 $n-2$ 種選擇，依此類推。

不過每個路徑都會變成一個環，$A->B->C->D->A$ 和 $A->D->C->B->A$ 的時間是一樣的。當有 n 個城市時，就有 $\frac{(n-1)!}{2}$ 組可能的解，若一一去計算這些解，一定無法在多項式時間內解決（$n!$ 不是 n 的多項式）。

若有 $a\sim z$ 共 26 個城市，則有 $\frac{25!}{2}$ 條路徑可供選擇，大約等於 1.55×10^{25}。

假設每秒可以計算一百萬（10^6）條路徑，一年有 3.15×10^7 秒，則所需時間為

$$\frac{1.55\times 10^{25}}{10^6\times 3.15\times 10^7}\approx 5\times 10^{11}\text{（年）},$$

也就是五千億年，這代表非多項式時間的算法常常是並不實用的。

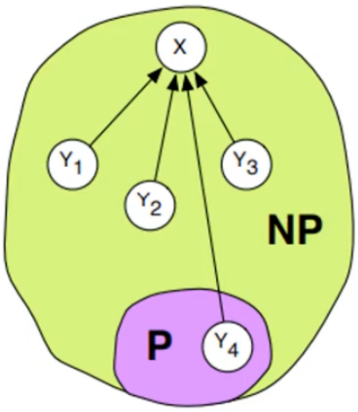
4. NP-complete（NPC）

本章最後，來看一下 NP-complete 問題。

（1）最難解決的 NP 問題

$P = NP$ 和 $P \neq NP$ 何者是對的，目前還沒有定論，不過共識上 $P = NP$ 問題「應該」不成立，也就是說，應該至少有一個問題雖然容易驗證（屬於 NP），但是不容易解決（不屬於 P）。

NP-complete 問題是 NP-hard 問題中，仍然屬於 NP 的那些，也就是 NP 問題歸約後得到的最難的問題裡，還是容易驗證的那些。NP-complete 問題因為還在 NP 的範疇內，又比所有其他 NP 問題來得困難，所以可以看作是 NP 問題中「最難」解決的問題。



用歸約的方式來解釋，如果 Y_i 都是 NP 問題，所有的 Y_i 都可以被歸約成某個問題 X（NP-complete / NPC 問題），那麼有 $Y \leq_P X$，代表問題 X 是 NP 問題中最難的。

（2）Cook-Levin 理論

1971 年 Stephen A. Cook 提出了 Cook-Levin 理論：任一個 NP 決策問題都可以在多項式時間內被轉換成同一個問題「某個布林方程式是否存在解」，這也就是說，如果找到一個方法來決定「某個布林方程式是否存在解」，就代表任意的

其他 NP 問題都可以在多項式時間內解決，也證明了 $P = NP$。

「布林方程式是否存在解」是第一個被找出的 NPC 問題（它是 NP-hard 問題，也仍然容易驗證，因此同時屬於 NP），隨後，又有上百個 NPC 問題陸續被發現，只要找到其中任何一個的多項式時間解法，就可以用這個解法來在多項式時間內解決所有的其他 NP 問題，代表 $P = NP$。