前言：

第二周的課程進入最後一個主題 —— NP – Completeness。之前的課程，多在關注如何有效率地解決實務上的問題，但大多數的問題，其實無法找到一個快速求解的方法，甚至是電腦在有限時間中，都無法計算出來。那何謂快速？這類問題該如何處理？就是本周課程的重點

NP Complete (Non-deterministic Polynomial Complete)：

多項式時間可解 (Polynomial-Time Solvable)：

當一個問題可以在 $O(n^k)$ 以內的時間求解，則稱該問題為多項式時間可解，其中 k 為常數 (即便 k 大到 10,000 仍滿足此定義)，P 問題集就是所有多項式時間可解問題的集合。

若問題非多項式時間可解，例如找一陣列中，是否有多個元素相加等於 0，若以 brute-force 方法計算，需要 $O(N \times 2^N)$ ，假設今天只有 1000 個元素，就已經是天文數字了，因此這類問題即便可解，也不代表在有限時間內，能夠計算出來，這類問題我們稱作 NP 問題， 是 intractable problem (難解的問題)，更嚴謹的定義如下：

1. 輸入長度可以多項式表示。
2. 解是否正確，可在多項式時間內確認。

P = NP？

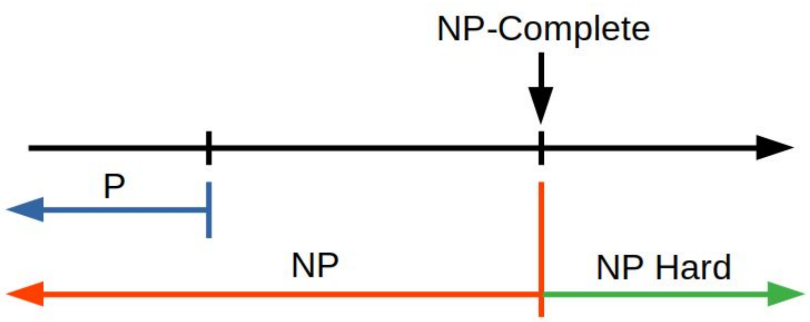
因此，NP 問題包含 P 問題，但 P = NP ? 這目前仍是數學界無法證明的難題，克雷數學研究所在 2000 年公布的七大難題之中 (成功解出者，可獲得 100 萬美金)，就包含 P = NP。那這個問題究竟重要在那？現今電腦的加密方式大多是 NP 問題 (如 RSA 加密利用大質數相乘的合數)，若今天有人證明出 P = NP，代表所有 NP 問題都可以化簡為某種 P 問題求解，這將大幅提昇我們的計算能力，解出目前人類無法處理的問題。

在數學家研究 P = NP 問題時，發現了一個重要的概念 —— NP complete。 1971 年 Stephen A. Cook 提出了一個關鍵的理論：任何一個 NP 問題都可以在多項式時間內，化約成 SAT 問題 (布林滿足性問題)，SAT 問題我們稱之為 NP Complete 問題，嚴謹的定義如下：

1. 這個問題是 NP 問題。
2. 所有 NP 問題都可以在多項式時間內，化簡成該問題。(亦即 NP-Complete 是 NP 問題中最困難的)

NP Complete 問題不僅僅有 SAT 問題，但重要的是，若我們今天能證明某個 NP Complete 問題是 P 問題，代表所有 NP 問題都可以在多項式時間內求解，亦即證明 P=NP。另外還有一種問題叫 NP Hard，若所有 NP 問題都可以在多項式時間內化簡為該問題，則稱作 NP Hard 問題，NP Complet 是其中的特例，因為 NP Hard 問題可不必為 NP 問題。

本節最後簡略的畫出 P、NP、NP Complete 及 NP Hard 的關係 (並非完全正確，但提供一個直觀的概念)



NP Complete 問題的處理方式：

一般來說，若碰到 NP Complete 問題，有三種策略可以參考：

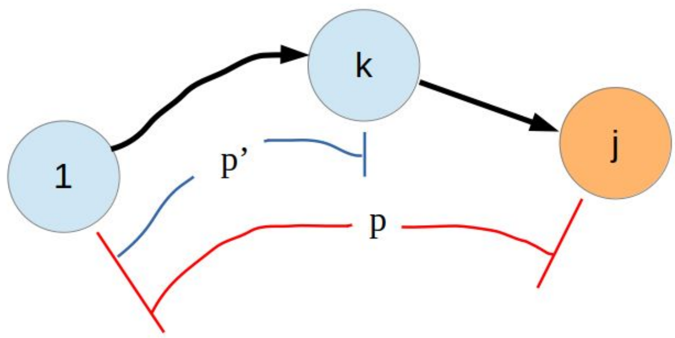
1. 找出可在多項式時間內處理的特例，如最後一週的 2 SAT 問題。
2. Heuristic (啟發式搜尋)，這類辦法通常無法提供最佳解，但能在有效時間內提出不錯的解，如利用貪婪演算法搭配動態規劃的包裝問題。
3. 以 exponential time 求解，但仍比 brute-force 快的方法，如接下來要談的 TSP 問題。

Traveling Salesman Problem (TSP):

TSP 問題給定一系列的節點，我們知道節點間的距離，計算通過所有節點並回到起點的最短路徑。若我們以 brute-force 方法計算，計算所有可能的排列組合，則時間複雜度為 O(n!)，但如果我們利用動態規劃，則時間複雜度可降至 $O(n^2 \times 2^n)$，在 n ＝ 30 附近可解。

TSP 問題可以拆成子問題：對**目標節點** j ∈ {1, 2,..., n}，及子集 S ⊆ {1, 2,..., n} 且包含節點 1 及 j，我們定義 Ls,j 為經過 S 中所有節點一次的 (1, j) 路徑的最小距離。假設我們有一條由 1 到 j 的路徑 p 通過 S 中所有節點，且最後一段為 (k, j)，若 p 為最短路徑，則 p’ = (1, k) 亦為通過 S-{j} 的最短路徑。寫成遞迴式如下：

$$L_{i,j} = \min_{\substack{k \in S \\ k \neq j}} \left(L_{s-j,k} + C_{kj} \right)$$



我們令 A 為一個 2-D 陣列，索引值分別為子集 S 以及目標 j ∈ {1, 2,..., n}，則初始狀況為：A[{j} , j] = distance(1, j)

動態規劃的虛擬碼如下：

for s := 2 to n-1:

for all sets S ⊆ {1, 2,..., n} of size m contains 1:

for each j ∈ S, j ≠ 1:

$$A[S, j] = \min_{\substack{k \in S \\ k \neq j}} \left(A[S-j, k] + C_{kj} \right)$$

return $\min_{j=2}^{n} \{ A[\{1, 2,..., n\}, j] + C_{kj} \}$

時間複雜度：

總共約 2^n 個子問題，每個子問題需以 S 中的某點作為終點，再掃過剩餘的點計算新的 A[S, j]值，複雜度為 $O(n^2)$ ，整體的時間複雜度為 $O(2^n \times n^2)$ 。

作業——以 Deterministic 方法求解旅行推銷員問題 (TSP)：

問題描述：

txt 檔中每一行代表一個城市的位置，第一欄為 x 座標，第二欄為 y 座標，兩城市間的距離以歐幾里得距離定義，試求通過這 25 個城市最後回到出發點的最短距離為何？

解題方法：

本題麻煩的地方在於，城市的排列組合非常多，例如 C25 取 10 就有高達 300 萬種可能性，這還只是其中一組，若直接以 tuple 容器類型儲存城市組合，勢必使用大量的記憶體。(空 List - 64 Byte、空 Tuple - 48 Byte；內有 25 個整數的 List - 264 Byte、 Tuple - 248 Byte)

為了節省記憶體空間，比較好的方式是使用整數，利用二進位的概念，25 個位數中，每一個位置都代表一個城市，如城市 5 = 00.....10000 = 2^4 = 16；城市 2、4、7 的組合＝00......1001010 = 2^6 + 2^3 + 2 = 74。過程中會用到的最大整數不大於 2^25，佔 28 Byte，比起一般容器，記憶體用量小非常多。

有了以上的概念後，剩下就是位元運算，整數與位元的互換，以及動態規劃的程式碼。
(bitmasks 參考：https://diego.assencio.com/?index=50ed9dcd9009dd70c3a2f8822271e4c7)