前言：

第一周的課程延續第一部份第四周的內容，提及更多有關圖論的經典演算法，如廣度優先搜尋 (BFS)、深度優先搜尋 (DFS) 以及拓樸順序 (Topological Sort) 等。作業是利用 Kosaraju's 算法求有向圖的強連通元件 (Strongly Connected Components)。

拓樸順序 (Topological Sort)：

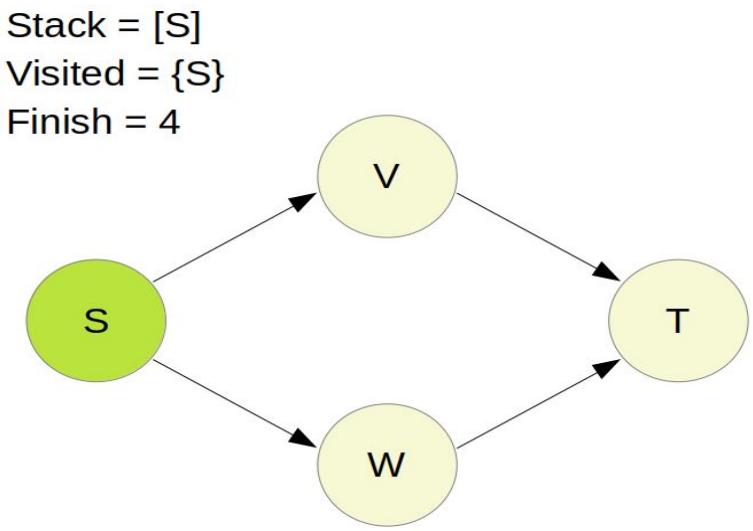
由於 Kosaraju's 是一種基於拓樸順序算法，所以必需先說明拓樸順序的算法邏輯。
在講解拓樸排序時，僅考慮有向圖的狀況 (有方向性才有順序)，一個有效的拓樸順序必須滿足以下兩個條件：

1. 順序中包含圖中每個節點，且只出現一次。
2. 若存在一條從 x 到 y 的路徑 (x, y)，則 x 一定排在 y 的前面；也可以理解成圖中不存在 y 至 x 的路徑。

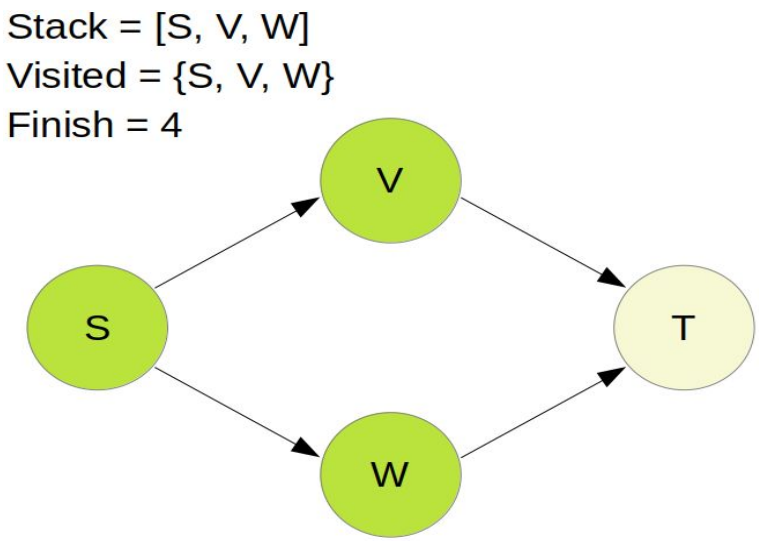
條件 1. 非常直觀，排順序時一個節點出現兩次本來就很奇怪，就像你要從新竹到台北，需要經過桃園，結果桃園經過兩次，這中間一定有什麼誤會。
條件 2. 要說明的是，只有有向無環圖 (DAG) 的拓樸排序才有意義，假如今天選課系統告訴你，要先修 [機率] 才能修 [統計]，修過 [統計] 才能修 [機器學習]，選了 [機率] 系統卻跟你說沒修過 [機器學習] 不能修 [機率]，這不是莫名其妙嘛？

拓樸排序計算的是一系列事件的前後關係，因此 DFS 這種一個腸子通到底的算法非常適合，且可以在 O(m+n)時間完成，其中 m 為圖的邊數、 n 則為圖的節點數，以下利用圖片簡單說明演算邏輯：

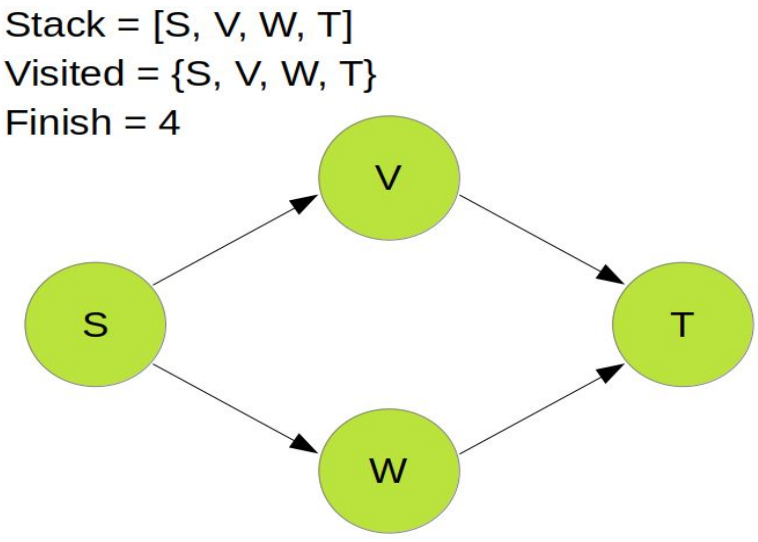
1. 一有向無環圖如下所示，假設我們以 S 作為起點 (選任一點都可以，僅方便說明)，排序 (finish time) 設為 4 (節點數)，並將 S 設定為已探索且加入 stack 中。



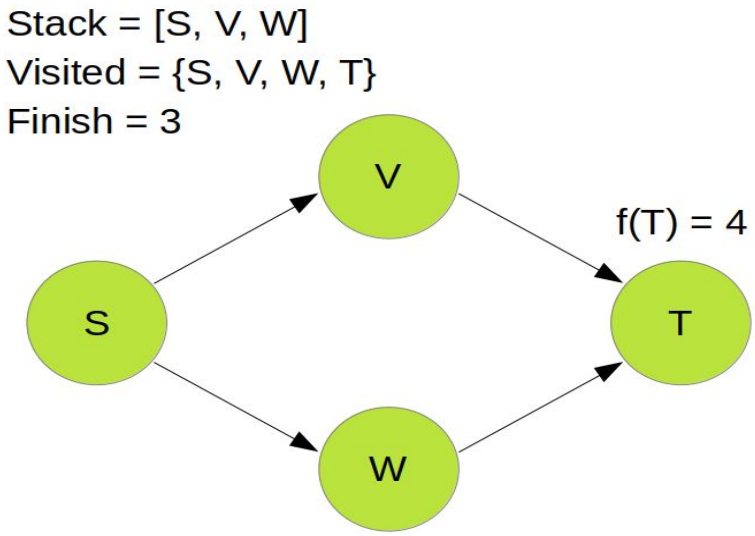
2. 遍歷 S 所有向外的邊，若該節點尚未搜尋過，則加入 stack 中，並設定已探索。注意必須確認從某點出發所有可探索的節點都探索過了，才可移除。



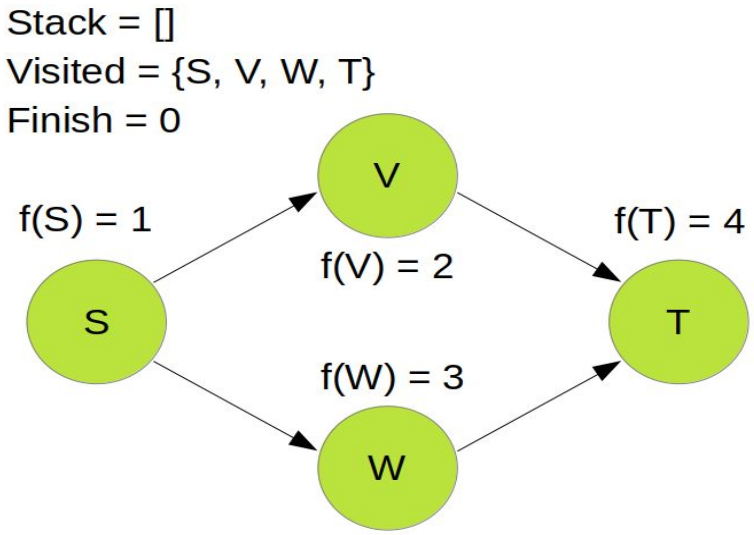
3. 遍歷 W 所有向外的邊，步驟同第 2. 步。



4. 發現 T 沒有任何向外的邊，代表走到盡頭，T 的排序為 4，並將 T 從 stack 中移除。

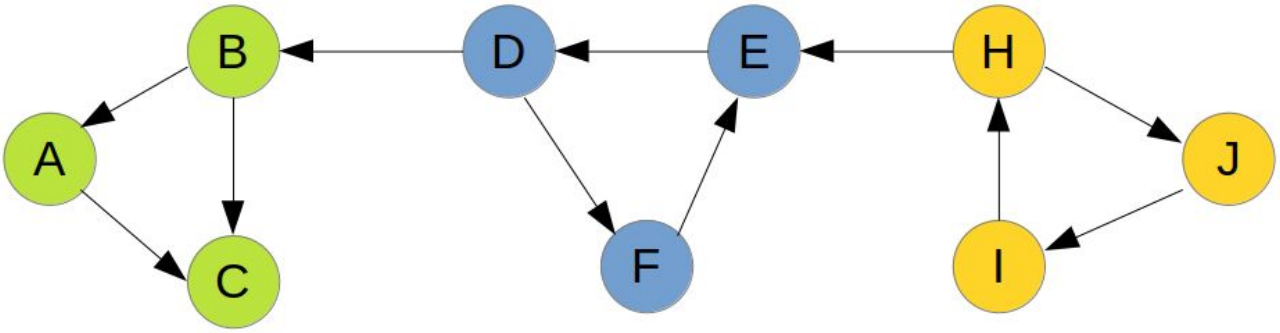


5. 接下來的步驟同 4. ， W、V 及 S 所有向外的邊都已探索過，依序從 stack 移除並設排序，最後所有點都遍歷過且完成排序 。



強連通元件 (Strongly Connected Components)：

強連通元件 (以下簡稱 SCC) 的定義為：對一個有向圖來說，一群節點組成的子圖中任取一節點對 x, y，同時存在 x 至 y 及 y 至 x 的路徑，也就是任意節點皆可以抵達子圖中的每一個節點，則稱該子圖為原圖的 SCC。一張圖可以有多個 SCC，如下圖就有三個。

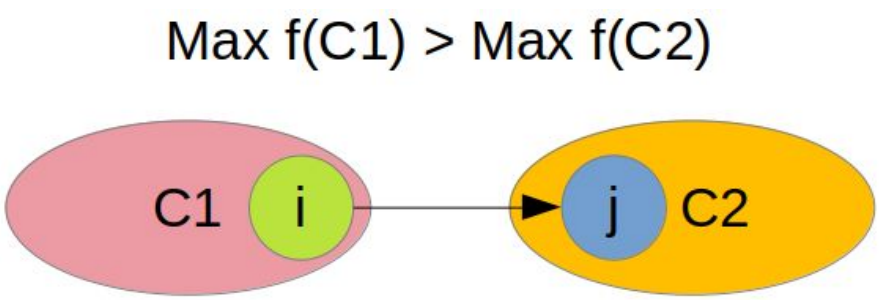


如何求 SCC ？

那該如何尋找 SCC ？一個直覺的想法是利用 DFS ，不過單純使用 DFS 會有問題。假設我們今天從 H 節點開始搜尋，可以找到 {H, J, I}這個 SCC，但如果接下來從 A 開始呢？由於可以由 B 節點走到 D 節點，代表 {A, B. C, D, E, F}會被考慮成同一個 SCC ，明顯是錯誤的！但是看起來，如果正確的選擇每次 DFS 的起始點，似乎就可以順利找到所有 SCC。

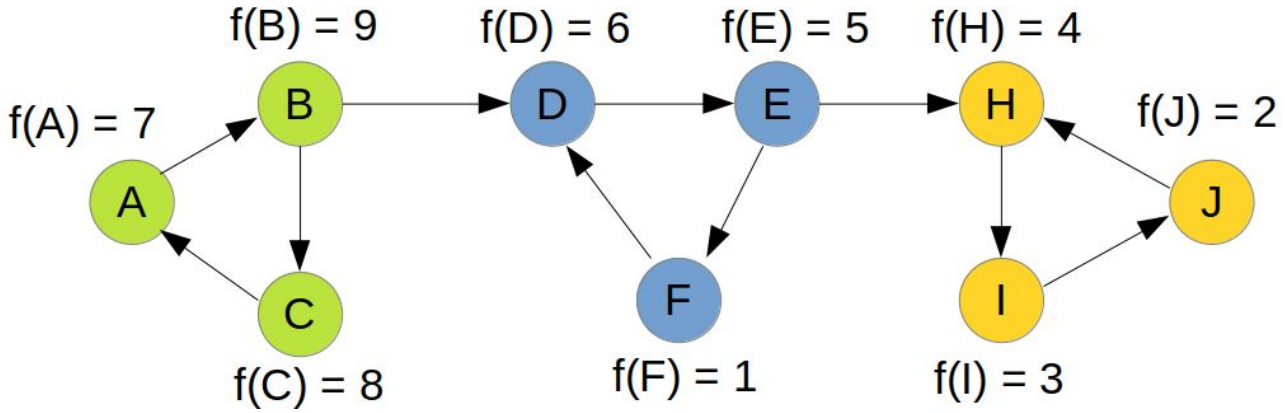
Kosaraju's Two-Pass 算法 ：

我們可以由上圖觀察到，若所有 SCC 都縮成一個超級節點，則會形成一個有向無環圖，這適用於所有 SCC，因為如果形成有環圖，代表兩個 SCC 之間有節點連通，這違反了 SCC 的定義。現在考慮兩個相鄰的 SCC C1 & C2，其中 C1 的 i 節點可以走到 C2 的 j 節點。

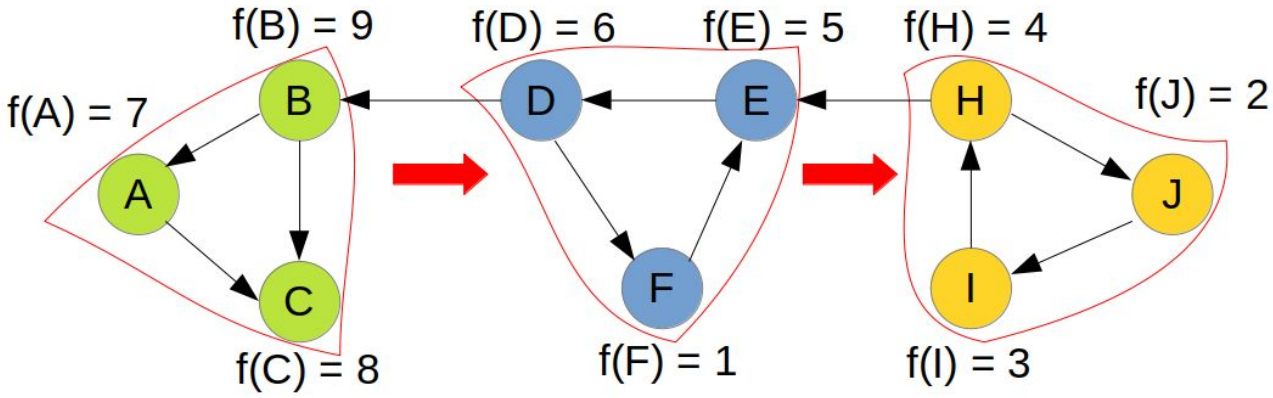


若 f(v) 為拓樸順序的完成時間 (這邊完成時間不是順序，而是何時被標記為搜尋完成)，因為 C1 有路徑通往 C2，依 DFS 的特性，C1 中的某些節點一定會等到 C2 所有節點都標記完成後，才能標記；也就是說，C1 中最大的完成時間，一定大於 C2 中最大的完成時間。利用上述特性，若我們先將圖反向 (想像上圖為反向後的樣子)，以拓樸排序計算各節點的完成時間，因 C1 的最大完成時間一定大於 C2，利用這個順序，對原圖進行 DFS (此時 j 指向 i)，代表我們每次都可以從當下為槽 (sink) 的 SCC 開始進行搜尋，而不會走到屬於其他 SCC (source) 的節點。

由之前 9 個節點的圖當例子，我們先將圖反向，並假設由 D 節點開始搜尋，下圖為可能的拓樸完成時間。



接著我們將完成時間套回原圖，若每次都從當下完成時間最大的節點進行搜尋，且無法走到其他 SCC 節點，如我們所預期，良好地利用拓樸順序，以及 SCC 縮成一個節點時，形成 DAG 的特性。



作業──計算前五大 Strongly Connected Components 中元素的個數：

問題描述：

檔案為一有向圖，節點編號從 1 ~ 875714，每一行都代表一個邊，第一欄的節點為邊的尾巴 (tail)、第二欄的節點為邊的頭 (head)，例如 "2 47646" 代表節點 2 有一個向外邊到節點 47646。本次作業請利用 Kosaraju 算法找出前五大 SCC 的元素個數。

解題方法：

本題給定的圖非常龐大，有 80 多萬個節點以及多達 500 多萬條邊，因此不適合以遞迴的方式處理 DFS。若以 stack 資料結構處理，那這題就變得簡單很多，也無須考慮遞迴或記憶體限制的問題。