圖論演算法導論

完整教材 (依據 5CCS2FC2 Foundations of Computing II 第四週內容編寫)

BFS / DFS、最小生成樹、拓撲排序與強連通元件

整理自 Dr. Christopher Hampson(King's College London) 之投影片
並參考補充文獻擴充編寫

2026 年 6 月

本週學習目標

1. 認識**圖的記號**與圖演算法的時間度量 $T(|V|, |E|)$。
2. 掌握**廣度優先搜尋 (BFS)** 與**深度優先搜尋 (DFS)**, 並理解其 $O(|V| + |E|)$ 複雜度。
3. 理解**最小生成樹 (MST)**, 精通 **Kruskal** 與 **Prim** 兩個貪婪演算法及其正確性 (割性質/環性質)。
4. 認識**有向無環圖 (DAG)** 與**拓撲排序**。
5. 理解**強連通元件 (SCC)**、凝聚圖為 DAG 的證明, 以及 **Kosaraju** 兩趟 DFS 演算法。

Contents

1 圖論基礎與記號

定義 1.1 (圖)**.** 一個**圖** $G = (V, E)$ 由**頂點集** V 與**邊集** E 組成。若邊有方向則稱**有向圖**, 否則為**無向圖**。若每條邊 e 另附一個權重 $w(e)$, 則記為**加權圖** $G = (V, E, w)$。

分析圖演算法時, 執行時間同時取決於頂點數與邊數, 故把時間函數推廣為兩個參數:

$$T(|V|, |E|) = \text{在 } |V| \text{ 個頂點、 } |E| \text{ 條邊的圖上所需的基本運算次數.}$$

其中**基本運算 (primitive operations)** 指可在 $O(1)$ 步內完成的動作 (例如讀一個頂點、走一條邊、入隊/出隊)。

備註 1.2. 對一個簡單圖,$|E|$ 介於 0 與 $\binom{|V|}{2} = O(|V|^2)$ 之間。當 $|E| = \Theta(|V|^2)$ 稱為**稠密圖**; 當 $|E| = \Theta(|V|)$ 稱為**稀疏圖**。因此 $O(|V| + |E|)$ 這種「線性於圖大小」的複雜度, 是圖走訪能達到的最佳階。

我們用 $\mathrm{Adj}(u)$ 表示與 u 相鄰的頂點集合; 其大小 $|\mathrm{Adj}(u)|$ 稱為 u 的**度數**。重要的握手等式:

$$\sum_{u \in V} |\mathrm{Adj}(u)| = 2|E| \quad \text{(無向圖)}, \qquad \sum_{u \in V} |\mathrm{Adj}(u)| = |E| \quad \text{(有向圖)},$$

這是稍後證明走訪複雜度為 $O(|V| + |E|)$ 的關鍵。

2 廣度優先搜尋 (Breadth-First Search, BFS)

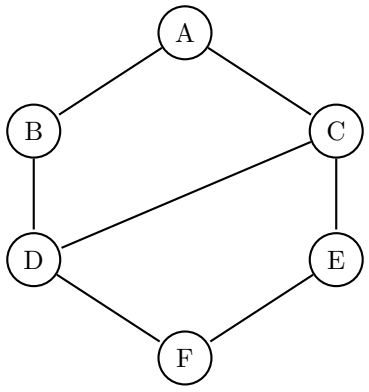
BFS 從根頂點出發, **一層一層**向外擴展: 先走訪所有距離為 1 的鄰居, 再走訪距離為 2 的, 以此類推。它的核心資料結構是**佇列 (Queue, 先進先出 FIFO)**。

Algorithm 1 BFS(G, r)

```
1: 選定根頂點 r 並加入佇列, 標記為已走訪
2: while 佇列非空 do
3:     檢視佇列最前端的元素 u
4:     將 u 所有未走訪的鄰居加入佇列 (並標記已走訪)
5:     將 u 從佇列移除
```

2.1 範例追蹤

考慮下圖, 以 A 為根。BFS 依序處理頂點, 佇列演變如表所示。



步驟	處理頂點 u	處理後的佇列
1	A	$[B, C]$
2	B	$[C, D]$
3	C	$[D, E]$
4	D	$[E, F]$
5	E	$[F]$
6	F	$[]$(空, 終止)

走訪順序為 A, B, C, D, E, F。BFS 同時得到由根出發的**最短路徑層數**:A 在第 0 層,B, C 在第 1 層,D, E 在第 2 層,F 在第 3 層。

BFS 的關鍵性質

在**無權圖**中,BFS 找到的是從根到每個頂點的**最短路徑 (邊數最少)**。這是 BFS 最重要的應用: 無權最短路徑、判斷連通性、二分圖檢測等。

2.2 複雜度分析

定理 2.1. *BFS* 的終止時間為 $O(|V| + |E|)$。

Proof.

- 每個頂點**至多入隊與出隊一次** (入隊時即標記已走訪, 不會重複)。故步驟 3–5 對每個 $u \in V$ 各執行一次。
- 檢視 (步驟 3) 與移除 (步驟 5) 各為 $O(1)$。
- 步驟 4 在處理 u 時要掃過它的鄰接串列, 花 $O(|\mathrm{Adj}(u)|)$。
- 全部加總:

$$T(|V|, |E|) = \sum_{u \in V} \big(O(1) + O(|\mathrm{Adj}(u)|)\big) = O(|V|) + O\Big(\sum_{u \in V} |\mathrm{Adj}(u)|\Big) = O(|V|) + O(2|E|) = O(|V| + |E|).$$

□

3 深度優先搜尋 (Depth-First Search, DFS)

DFS 則是**一路走到底**: 沿著一條路徑盡可能深入, 走不下去才回溯。它的核心資料結構是**堆疊 (Stack, 後進先出 LIFO)**——把 BFS 的佇列換成堆疊即可。

Algorithm 2 DEPTH-FIRST-SEARCH(G, r)

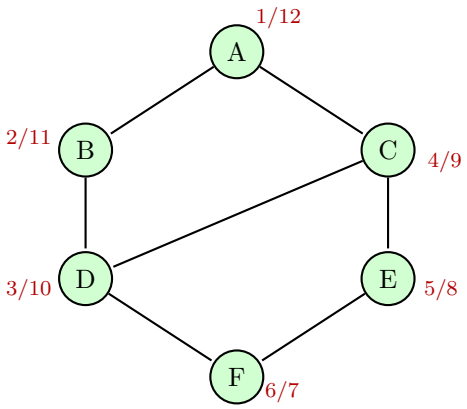
```
1: 選定根頂點 r 並推入堆疊
2: while 堆疊非空 do
3:     檢視堆疊頂端的元素 u
4:     將 u 任一未走訪的鄰居推入堆疊
5:     當 u 無未走訪鄰居時, 將 u 從堆疊彈出
```

3.1 發現時間與完成時間

DFS 常為每個頂點記錄兩個時間戳, 以一個全域時鐘 $(1, 2, 3, \dots)$ 度量:

- **發現時間** $d[u]$: 第一次走訪 (推入)u 的時刻。
- **完成時間** $f[u]$:u 所有後代都探索完畢、將 u 彈出的時刻。

以下用前述同一張圖 (根 A, 每次選字母序最小的未走訪鄰居) 示範, 標註為 d/f:



探索過程:$A(1) \to B(2) \to D(3) \to C(4) \to E(5) \to F(6)$, 然後依序回溯完成 $F(7), E(8), C(9), D(10), B(11), A(12)$。

括號定理 (Parenthesis Theorem)

對任意兩頂點 u, v, 區間 $[d[u], f[u]]$ 與 $[d[v], f[v]]$ 要嘛完全包含、要嘛完全不相交, 絕不會部分交疊。v 是 u 的後代 $\iff$ $[d[v], f[v]] \subseteq [d[u], f[u]]$。這正是 DFS「巢狀」結構的數學寫照, 像一組正確配對的括號。

3.2 複雜度分析

定理 3.1. *DFS* 的終止時間為 $O(|V| + |E|)$。

Proof. 證明與 BFS 幾乎相同, 只是把佇列換成堆疊: 每個頂點恰被推入與彈出一次 $(O(|V|))$, 而探索每個頂點的鄰居總共花 $\sum_u O(|\mathrm{Adj}(u)|) = O(|E|)$。合計 $O(|V| + |E|)$。 □

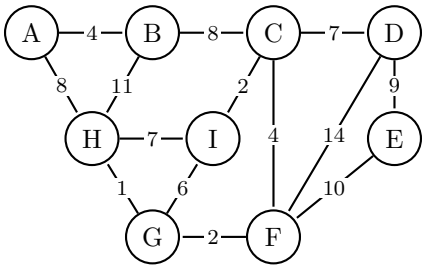
3.3 BFS 與 DFS 的比較

	BFS	**DFS**
資料結構	佇列 (FIFO)	堆疊 (LIFO)/遞迴
走訪方式	逐層由近到遠	一路深入再回溯
複雜度	$O(\|V\| + \|E\|)$	$O(\|V\| + \|E\|)$
典型應用	無權最短路徑、連通分量、二分圖檢測	拓撲排序、強連通元件、找環、迷宮回溯

4 最小生成樹 (Minimum Spanning Tree, MST)

定義 4.1 (生成樹)**.** 加權連通圖 $G = (V, E, w)$ 的**生成樹**是一棵樹 $T = (V, E')$, 其中 $E' \subseteq E$、連接了所有頂點且不含環。生成樹恰有 $|V| - 1$ 條邊。

定義 4.2 (最小生成樹)**. 最小生成樹 (MST)** 是所有生成樹中總權重最小者:$w(T) = \sum_{e \in E'} w(e)$ 最小。



此圖 (經典範例) 的 MST 由邊 $\{G–H(1),\ C–I(2),\ F–G(2),\ A–B(4),\ C–F(4),\ C–D(7),\ A–H(8),\ D–E(9)\}$ 組成, 總權重 $= 37$。

4.1 兩個基本性質 (MST 正確性的根基)

幾乎所有 MST 演算法都是**貪婪**的, 其正確性都奠基於以下兩個性質。為方便敘述, 假設所有邊權互異 (MST 唯一)。

定義 4.3 (割)**.** 圖的一個**割** $(S, V \setminus S)$ 是把頂點分成兩個非空集合的劃分。一條邊若兩端點分屬兩側, 稱為**跨越邊 (crossing edge)**。

性質 4.4 (割性質 Cut Property)**.** 對任意割 $(S, V \setminus S)$, 其最小權重的跨越邊 e 必屬於 MST。

證明 *(交換論證).* 設 MST 為 T^* 且 $e \notin T^*$。把 e 加入 T^* 會形成唯一一個環 C; 此環必定再一次跨越割, 即環上另有一條跨越邊 $f \neq e$。因 e 是最小跨越邊,$w(e) < w(f)$。令 $T' = T^* \cup \{e\} \setminus \{f\}$, 它仍是生成樹, 但 $w(T') = w(T^*) + w(e) - w(f) < w(T^*)$, 與 T^* 是 MST 矛盾。故 $e \in T^*$。 □

性質 4.5 (環性質 Cycle Property)**.** 對任意環 C, 其最大權重的邊 f 必不屬於 MST。

證明 *(交換論證).* 設 $f \in T^*$。從 T^* 刪去 f 會把樹斷成兩個連通塊; 環 C 上必有另一條邊 e 橫跨這兩塊。因 f 是環上最大邊,$w(e) < w(f)$。令 $T' = T^* \cup \{e\} \setminus \{f\}$ 仍為生成樹且 $w(T') < w(T^*)$, 矛盾。故 $f \notin T^*$。 □

5 Kruskal 最小生成樹演算法

Kruskal 的策略: 把所有邊**由小到大**排序, 逐一檢視, 只要加入後不形成環就納入。

Algorithm 3 KRUSKAL-MST(G)

```
依權重 (由小到大) 排序邊串列 E
初始化 T ← ∅
while 排序後的 E 非空 do
    從 E 取出最短邊 (u, v)
    if T ∪ {(u, v)} 無環 then
        T ← T ∪ {(u, v)}
return 樹 T
```

5.1 Union-Find: 如何高效檢查「是否成環」

關鍵在第 5 行: 加入 (u, v) 是否成環? $\iff$ u **與** v **是否已經連通**。我們用**並查集 (Union-Find / Disjoint-Set)** 維護「目前各連通分量」:

- 初始時每個頂點自成一個集合 $\{A\}, \{B\}, \ldots$
- $\textsc{Find}(u)$: 查詢 u 所屬集合的代表元。
- 若 $\textsc{Find}(u) = \textsc{Find}(v) \Rightarrow$ 已連通, 加入會成環, **捨棄** (環性質)。
- 否則加入此邊, 並 $\textsc{Union}$ 合併兩集合 (割性質)。

範例: 連通分量的合併過程

沿用上頁的圖, 依序處理邊 (權重遞增)。集合的演變 (只列出合併的關鍵步驟):

$$\begin{aligned}
&\{A\}\{B\}\{C\}\{D\}\{E\}\{F\}\{G\}\{H\}\{I\} \\
&\xrightarrow{GH(1)} \{G,H\} \ \cdots \ \xrightarrow{CI(2)} \{C,I\} \ \xrightarrow{FG(2)} \{F,G,H\} \\
&\xrightarrow{AB(4)} \{A,B\} \ \xrightarrow{CF(4)} \{C,I,F,G,H\} \ \xrightarrow{CD(7)} \{C,D,I,F,G,H\} \\
&\xrightarrow{AH(8)} \{A,B,C,D,F,G,H,I\} \ \xrightarrow{DE(9)} \{A,B,C,D,E,F,G,H,I\}
\end{aligned}$$

注意 $BC(8)$、$BH(11)$ 等邊在處理時兩端已同集合, 會被捨棄。最終 8 條邊、總權重 37。

5.2 複雜度與正確性

定理 5.1. *Kruskal* 演算法的終止時間為 $O(|E| \log |E|)$。

Proof.
- **步驟 1(排序)**: 用合併排序等高效演算法, 排序 E 花 $O(|E| \log |E|)$。
- **步驟 4、6(取邊、加邊)**: 每條邊 $O(1)$, 共 $\sum_E (O(1) + O(1)) = O(|E|)$。
- **步驟 5(檢查連通)**: 以 Union-Find 實作, 全部操作合計 $O((|V| + |E|)\, \alpha(|V|))$, 其中 α 是反阿克曼函數, 成長極慢, 實務上可視為常數。
- 合計

$$T(|V|, |E|) = O(|E| \log |E|) + O(|E|) + O((|V| + |E|)\alpha(|V|)) = O(|E| \log |E|).$$

□

備註 5.2 (正確性). Kruskal 每次加入的邊, 都是連接「兩個不同分量」的目前最短邊——正是某個割的最小跨越邊, 由**割性質** (性質 4.4) 保證屬於 MST; 被捨棄的邊都是某環上的最大邊, 由**環性質** (性質 4.5) 保證不屬於 MST。故輸出必為 MST。

6 Prim 最小生成樹演算法

Prim 的策略: 從一個根頂點出發, 讓樹像「滾雪球」般**逐步長大**, 每次加入連接「樹」與「樹外」的最便宜邊。它用**優先佇列 (priority queue)** 維護候選邊。

Algorithm 4 PRIM-MST(G)

選根 $r \in V$, 把所有與 r 相鄰的邊加入優先佇列 Q
初始化 $T \leftarrow \varnothing$
while Q 非空 **do**
    從 Q 取出最短邊 (u, v)
    if $T \cup \{(u, v)\}$ 無環 **then**
        $T \leftarrow T \cup \{(u, v)\}$
        把與 u 或 v 新相鄰的邊加入 Q
return 樹 T

範例: 從 A 出發的成長過程

沿用同一張圖, 根 $= A$。每步「樹」新增一個頂點 (括號為被選中的最便宜跨越邊):

$$A \xrightarrow{(A,B)\,4} B \xrightarrow{(B,C)\,8} C \xrightarrow{(C,I)\,2} I \xrightarrow{(C,F)\,4} F \xrightarrow{(F,G)\,2} G \xrightarrow{(G,H)\,1} H \xrightarrow{(C,D)\,7} D \xrightarrow{(D,E)\,9} E.$$

最終得到與 Kruskal 相同的 MST(總權重 37)——MST 唯一時, 兩法殊途同歸。

定理 6.1. *Prim* 演算法的終止時間為 $O(|E| + |V| \log |V|)$。

Proof. 複雜度取決於優先佇列的實作。每條邊至多被處理一次 (共 $O(|E|)$ 次「減少鍵值/插入」), 每個頂點被取出一次 (共 $|V|$ 次「取最小」)。採用**費氏堆 (Fibonacci Heap)** 時,DECREASE-KEY 攤還 $O(1)$、EXTRACT-MIN 攤還 $O(\log |V|)$, 合計 $O(|E| + |V| \log |V|)$。 □

備註 6.2 (正確性). 在任一時刻令 $S =$「已在樹中的頂點」,Prim 取的正是割 $(S, V \setminus S)$ 的最小跨越邊, 由**割性質**保證屬於 MST。對稀疏圖,Kruskal 的 $O(|E| \log |E|)$ 與 Prim(費氏堆) 的 $O(|E| + |V| \log |V|)$ 都相當高效。

7 有向無環圖與拓撲排序

定義 7.1 (有向無環圖 DAG)**.** 有向圖 $G = (V, E)$ 若**無反身邊**且**不含長度 ≥ 2 的有向環**, 即

$$\text{若存在路徑 } u \rightsquigarrow v, \text{ 則不存在路徑 } v \rightsquigarrow u,$$

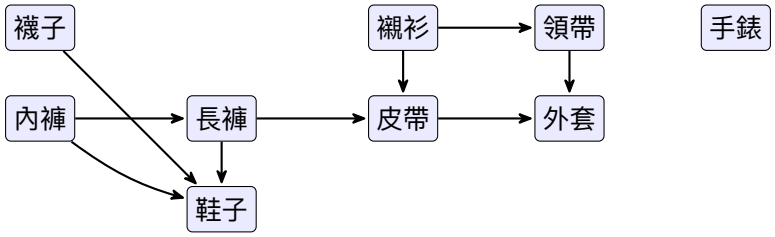
則稱為**有向無環圖 (DAG)**。DAG 是描述「依賴關係」「先後次序」的自然模型 (如軟體相依、課程先修、工作排程)。

定義 7.2 (拓撲排序)**.** DAG $G = (V, E)$ 的**拓撲排序**是頂點的線性排列 $L = \langle v_1, \ldots, v_n \rangle$, 使得

$$\text{若 } v_i \rightsquigarrow v_j \text{ 則 } i < j,$$

亦即所有箭頭都從「前面」指向「後面」(由上游流向下游)。

例 7.3 (穿衣服的順序)**.** 把「必須先穿 A 才能穿 B」畫成有向邊 $A \to B$:



一個合法的拓撲排序為:

$$\langle \text{襪子},\ \text{內褲},\ \text{長褲},\ \text{鞋子},\ \text{襯衫},\ \text{皮帶},\ \text{領帶},\ \text{外套},\ \text{手錶} \rangle.$$

(「手錶」與其他無依賴關係, 可放在任何位置。)

以 DFS 求拓撲排序　對 DAG 做 DFS, 並在每個頂點**完成** (彈出) 時把它壓入結果堆疊; 最後把堆疊由頂到底讀出, 即為拓撲排序。直覺: 一個頂點完成, 代表它能到達的下游全部探索完畢, 故它應排在所有下游之前——即「完成時間遞減」就是拓撲序。複雜度 $O(|V| + |E|)$。

8　強連通元件 (Strongly Connected Components, SCC)

定義 8.1 (強連通關係)**.** 在有向圖 G 上定義二元關係 $\sim \subseteq V \times V$:

$$u \sim v \iff u = v \text{ 或 (存在路徑 } u \rightsquigarrow v \text{ 且 } v \rightsquigarrow u).$$

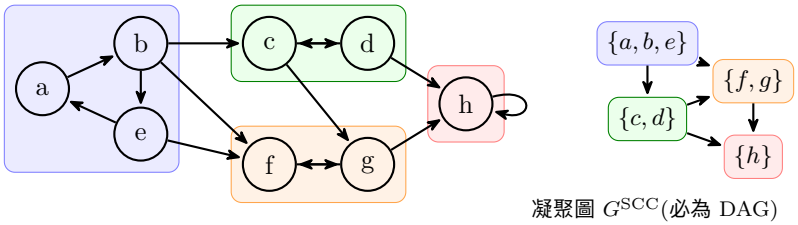
$\sim$ 是**等價關係** (自反、對稱、遞移)。

定義 8.2 (強連通元件)**.** 一個**強連通元件 (SCC)** 是極大的子集 $C \subseteq V$, 使得對所有 $u, v \in C$ 皆有 $u \sim v$。亦即 $C = [u]_\sim = \{v \in V : v \sim u\}$, 為 $\sim$ 的等價類。

定義 8.3 (凝聚圖 / 元件圖)**.** G 的**凝聚圖 (component graph)** $G^{\mathrm{SCC}} = (V^{\mathrm{SCC}}, E^{\mathrm{SCC}})$ 定義為:

$$V^{\mathrm{SCC}} = \{G \text{ 的所有 SCC}\}, \qquad (A, B) \in E^{\mathrm{SCC}} \iff \exists a \in A,\ b \in B,\ (a, b) \in E\ (A \neq B).$$

也就是把每個 SCC「縮成一個點」後得到的圖。



凝聚圖 G^{SCC}(必為 DAG)

8.1　凝聚圖必為 DAG

定理 8.4. 對任意有向圖 G, 其凝聚圖 G^{SCC} 都是有向無環圖。

證明 *(反證法)*. **步驟 1.** 假設 G^{SCC} 含一個環 $\langle C_1, C_2, \ldots, C_n, C_1 \rangle$(其中各 C_i 為相異 SCC)。

步驟 2. 任取 $a \in C_i$ 與 $b \in C_j (C_i \neq C_j$, 在此環上)。

步驟 3. 由凝聚圖的環, 沿著 $C_i \to \cdots \to C_j$ 與 $C_j \to \cdots \to C_i$ 各能得到 G 中的實際路徑, 故同時有 $a \rightsquigarrow b$ 與 $b \rightsquigarrow a$, 即 $a \sim b$。

步驟 4. 但 $a \sim b$ 表示 a, b 屬於同一個 SCC, 與 $C_i \neq C_j$ 矛盾。故 G^{SCC} 不可能有環。　□

備註 8.5. 這個定理是 SCC 演算法的理論基礎: 雖然原圖可能纏滿了環, 但「把每個 SCC 縮成一點」後得到的凝聚圖一定是 DAG, 於是可以拓撲排序、可以分層處理。

8.2 Kosaraju 演算法: 兩趟 DFS

Algorithm 5 SCC(G)(Kosaraju–Sharir)

```
對 G 做 DFS, 依完成時間遞減產生頂點順序串列 L(等同對 G 拓撲排序)
while L 非空 do
    取 L 中第一個 (完成時間最大的) 未指派頂點 u
    在轉置圖 G^T 上, 從 u 做 DFS
    把這趟 DFS 走訪到的所有頂點劃為一個新元件 S
    從 L 移除這些頂點
return 所有找到的元件
```

其中**轉置圖** $G^T = (V, E^T)$ 是把 G 的每條邊反向:$E^T = \{(v, u) : (u, v) \in E\}$。關鍵事實是 G 與 G^T 有完全相同的 SCC。

為什麼正確?(直覺)

1. **完成時間的性質**: 若凝聚圖中 $C \to C'$ 有邊, 則 C 中最大的完成時間 $> C'$ 中最大的完成時間。因此「按完成時間遞減」處理, 等於按凝聚圖的拓撲序由源頭往下處理。
2. **轉置的妙用**: 在 G^T 上, 所有跨元件的邊都反向了。從目前完成時間最大的頂點 u 出發在 G^T 做 DFS, 因為通往「下游元件」的邊已反向、通往「上游元件」的頂點已被移除, 這趟 DFS 會**剛好被困在 u 自己的 SCC 內**——走到的頂點正好構成一個 SCC。

定理 8.6. *Kosaraju 演算法在 $O(|V| + |E|)$ 時間內正確算出所有 SCC。*

複雜度. 第一趟 DFS:$O(|V| + |E|)$; 建轉置圖 G^T:$O(|V| + |E|)$; 第二趟 DFS:$O(|V| + |E|)$。三者皆線性, 合計 $O(|V| + |E|)$。 □

9 本週重點整理

演算法	重點	複雜度
BFS	佇列、逐層走訪、無權最短路徑	$O(\|V\| + \|E\|)$
DFS	堆疊/遞迴、發現/完成時間、回溯	$O(\|V\| + \|E\|)$
Kruskal MST	邊排序 + Union-Find 避環 (割/環性質)	$O(\|E\| \log \|E\|)$
Prim MST	優先佇列由根成長 (費氏堆)	$O(\|E\| + \|V\| \log \|V\|)$
拓撲排序	DAG 線性化;DFS 完成時間遞減	$O(\|V\| + \|E\|)$
SCC(Kosaraju)	兩趟 DFS + 轉置圖; 凝聚圖為 DAG	$O(\|V\| + \|E\|)$

10 練習題 (附解答)

練習 1. 在一個 $|V| = n$ 頂點的圖上,BFS 與 DFS 的複雜度皆為 $O(|V| + |E|)$。對**稠密圖** ($|E| = \Theta(n^2)$) 與**稀疏圖** ($|E| = \Theta(n)$), 這分別簡化為何?

解答

稠密圖:$O(n + n^2) = O(n^2)$; 稀疏圖:$O(n + n) = O(n)$。可見走訪成本主要由邊數決定。

練習 2. 為何 Kruskal 用**割性質**解釋「加入的邊」、用**環性質**解釋「捨棄的邊」?

解答

被加入的邊 (u, v) 連接兩個目前不同的分量, 是跨越「該分量 vs. 其餘」這個割的最小跨越邊, 故由割性質屬於 MST; 被捨棄的邊兩端已連通, 加入會成環, 且它是此環中最後 (最大) 被考慮的邊, 由環性質不屬於 MST。

練習 3. 給一個 5 頂點的有向圖, 使其只有 1 個 SCC; 再給一個使其有 5 個 SCC。

解答

單一 SCC: 有向環 $v_1 \to v_2 \to v_3 \to v_4 \to v_5 \to v_1$(每點互達)。
五個 SCC: 有向鏈 $v_1 \to v_2 \to v_3 \to v_4 \to v_5$(無任何回邊, 每點自成一 SCC, 凝聚圖即此鏈)。

練習 4. 對只有一條有向鏈 $v_1 \to v_2 \to \cdots \to v_n$ 的 DAG, 以「DFS 完成時間遞減」求拓撲序, 結果為何?

解答

從 v_1 做 DFS 會一路深入到 v_n, 完成順序為 $v_n, v_{n-1}, \ldots, v_1$(完成時間遞增)。按完成時間遞減讀出即 $v_1, v_2, \ldots, v_n$, 正是唯一的拓撲序。

練習 5 (思考題)**.** 為什麼 Kosaraju 演算法必須在**轉置圖** G^T 上做第二趟 DFS? 若直接在原圖 G 上做會發生什麼?

解答

若在原圖 G 上, 從完成時間最大的頂點 u 做 DFS, 會沿著下游邊走出 u 所在的 SCC, 把好幾個 SCC 混在一起。轉置後跨元件邊全部反向, 通往下游的路被切斷,DFS 便被「困」在 u 的 SCC 內, 恰好圈出單一元件。這正是演算法正確性的核心。

參考資料

1. C. Hampson, *5CCS2FC2 Foundations of Computing II, Week 4* 投影片 (bfsdfs / mst / scc),King's College London.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 3rd/4th ed., MIT Press.(第 20–23 章: 圖走訪、MST、SCC)
3. J. Kleinberg, É. Tardos, *Algorithm Design*, Pearson, 2006.(第 3、4 章: 圖走訪與貪婪/MST)
4. R. Sedgewick, K. Wayne, *Algorithms*, 4th ed., Addison-Wesley.(MST、割性質投影片)

5. Wikipedia: *Minimum spanning tree*; *Kruskal's algorithm*; *Prim's algorithm*; *Kosaraju's algorithm*; *Topological sorting*.
6. CP-Algorithms: *Strongly Connected Components and Condensation Graph*.