第 3 章　樹

愚者如我作詩，上帝造樹。

– Joyce Kilmer

3.1.　樹是簡單但重要圖

一個圖裡面如果有許多圈， 則點與點之間就會有各式各樣不同的路徑連接它們， 整個圖的結構因此就變得複雜。 反過來， 不含圈的圖， 結構就簡單許多， 大部份問題如果限制在這樣的條件下， 研究起來就特別容易。 繼 Euler 迴路有關圖論起源的介紹， 本書就從沒有迴圈的連通圖、 也就是樹[1]（tree） 開始。

在圖論的研究裡， 樹可以扮演一種特殊的角色， 那就是， 在探討一個問題的時候， 如果對一般圖無從下手， 可以先把問題限制在樹， 當作暖身開始。 如果問題在樹這類圖可以解， 再嘗試更一般的圖。 如果問題在樹這類圖都沒辦法解， 那可能表示這個問題可能很難。

第 1 章提到的 Ulam 猜想就是一個很好的例子， 這是一個看似容易， 至今還無人能解的難題， Kelly [11] 首先證明了 Ulam 猜想對於樹是會成立的， 後來陸續有人研究得出再各圖類的解答[2]， 但是對一般的圖還是沒有答案。

[1]這樣命名， 是因爲這種圖長得像樹木、 從根往上長出來又不形成迴圈的植物。 動植物相關的名詞經常在數學中出現， 例如， 代數幾何當中也有所謂的莖跟芽， 演算法中有種子， 圖論裡除了樹、葉之外，還有花、仙人掌、毛毛蟲等。

[2]Ulam 猜想有解的其他圖類含： 正則圖（regular graphs）、 非連通圖（disconnected graphs）、 單位區間圖圖（unit interval graphs）、 不含葉的可分離圖（separable graphs without end vertices）、極大平面圖（maximal planar graphs）、 極大外平面圖（maximal outerplanar graphs）、 外平面圖（outerplanar graphs）、 臨界區塊（critical blocks）。

圖論也有如下述， 限制在樹就已經是 NP-難的問題。 假設 A 是圖 G 的相鄰矩陣， **帶寬問題**（bandwidth problem）就是要將 A 的某些列調換、 同時也將對應的行同步調換， 使最後的矩陣中非 0 的元素盡量靠近對角線。 在這樣的結構下， 也可以用比較簡單、比較少的空間將圖儲存起來。 這個問題的另外一種描述方法是， 要將圖 G 的點排列成 $v_1, v_2, \ldots, v_n$， 使得

$$\max_{v_iv_j \in E(G)} |i-j|$$

的值越小越好。 舉例來說， n-圈 C_n 的點集是 $\{1, 2, \ldots, n\}$、 邊集是 $\{12, 23, \ldots, (n-1)n, n1\}$； 如果點排成 $1, 2, \ldots, n$ 則 $\max_{v_iv_j \in E(G)} |i-j| = n-1$ 很大； 但如果將點排成 $1, 2, n, 3, n-1, 4, n-2, \ldots$ 則 $\max_{v_iv_j \in E(G)} |i-j| = 2$ 達到最小可能。 圖的帶寬問題即使限制在樹都已經是 NP-難的問題， 對於一般的圖就更難了。

樹因爲結構簡單， 也成爲建模的一個好工具。 例如， 化學中比較簡單基本的分子， 其形狀都像樹。 計算同分異構物數目相關的問題， 就是問有 n 點的樹有多少個。 這種問題， 對於點名稱重要的**標號樹**（labelled trees）有解， 點名稱不重要的樹、 也就是同構的算同一顆樹、 沒有好的解。

在資訊領域裡， 樹更扮演多重的角色。 例如， 第 2 章用有根樹的結構表達集合， 得到聯集尋找問題的快速演算法。 設計圖的演算法時， 常用到的深度優先搜尋（depth-first search）和廣度優先搜尋（breadth-first search）， 對應的深度優先樹和廣度優樹， 在演算法裡扮演重要的角色。

在架設電腦網路時會遇到如下的問題， 可以轉化成圖論的最優化問題。 考慮由 n 個電腦工作站連接成的網路， 一種連接的方法是， 把任兩個工作站之間都牽線， 此時對應的圖連成 K_n 的樣子。 但是這樣做是沒有必要的， 更重要的是、障樣做花費很大， 而且牽線操作困難。 簡單的方法是， 只要對應的圖是連通的， 網路就能夠運作， 所以其實只要按照一個 K_n 的連通生成子圖來架設網路即可。 有個問題； 基於工作站所處的地點與條件的差異， 在各組不同的工作站之間架設網路線所需要的施工成本也不盡相同。 那麼， 我們應該如何架設， 才能使得網路既是連通的、 又是所有可能的架設方法當中成本最低的？ 這就是， 最小生成樹的問題。

其它利用樹解結問解的例子包括， 編碼理論的 Hoffman 樹， 也都是很實用的圖論最優化問題。

3.2. 樹的基本性質

一個無圈的連通圖稱爲**樹**（tree），各個連通部分都是樹的圖稱爲**林**（forest）；林跟**無圈圖**（acyclic graph）的意思是一樣的。圖 3.1 列出、同構是爲相同的、所有 11 棵含有 7 個點的樹。

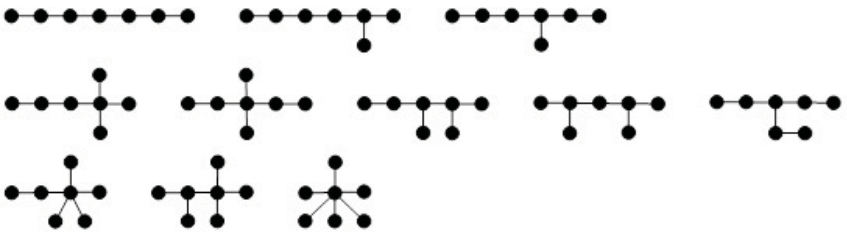

圖 3.1: 所有 11 棵恰有 7 點的樹。

樹除了上面的定義以外，還有許多不同的等價定義，在證明這些定義等價之前，先看幾個有用的性質。

性質 3.1. 若樹 T 最少含有兩點，則 T 最少有兩片葉。

證明： 從 T 當中選一條最長的路徑 P: $v_0, v_1, \ldots, v_r$。因爲 T 至少有一條邊，故 $r \geq 1$。注意到 v_0 和 v_r 一定是葉，因爲如果它們還有跟其他點相鄰，該點必不在 P 當中，不然就會出現圈，於是 P 就可以再延長，矛盾。 ■

性質 3.2. 若 v 是樹 T 中的葉，則 $T - v$ 也是樹。

證明： 顯然 $T - v$ 中沒有圈。對於任何 $x, y \in V(T - v)$，因爲它們也是 T 中的點，由 T 的連通性知道，在 T 中存在一條 x-y 路徑，而 v 不會在這條路徑上（因爲它在 T 中的度數是 1），所以該路徑也是 $T - v$ 中的 x-y 路徑；這樣就證明了 $T - v$ 是連通的，所以它是樹。 ■

性質 3.2 是很重要的基本事實，這在利用數學歸納法證明樹的一些定理的時候常用到。而在設計跟樹有關的演算法時，也常常一片葉一片葉地處理。意思是說，令 $T_1 = T$，而在 T_1 中取一片葉 v_1，然後令 $T_2 = T_1 - v_1$，並在 T_2 中取一片葉 v_2，如

此反覆下去， 直到最後 T_n 只剩一點 v_n 爲止。 也可以用下面的性質來描述： 一個圖 G 的**樹序**（tree ordering）是指將圖的點排爲 $v_1, v_2, \ldots, v_n$， 使得

對所有 $1 \leq i < n$，點 v_i 恰有一鄰居 v_j 滿足 $i < j$。

例如考慮圖 3.2 (a) 左邊的這個樹， 我們將它重新安排成右邊的這個樣子； 即選定中間的那個點當作**根**（root）， 而其他的枝條跟葉都是從這點逐一長出來的。 依照這種安排， 將樹從最底層開始逐層編號， 如圖所示。 易看出這個順序滿足樹序的條件。

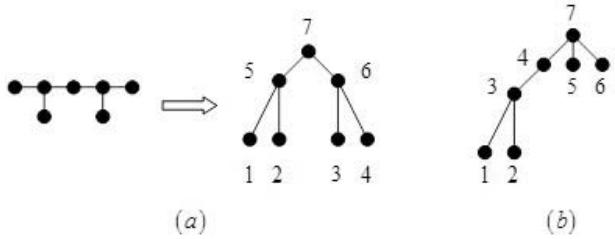

圖 3.2: 同一棵樹的兩種不同樹序。

一個圖的樹序不一定是唯一的， 例如圖 3.2 (b) 就是同一個圖的另外一種樹序， 其中我們選了另外一個點當作樹根。 基本上， 易看出樹中的任何一個點都可以當作樹根， 因而也給出了各種不同的樹序。

性質 3.3. 圖 G 有樹序，若且唯若 G 是樹。

證明： 從前面的構造過程中可以看出樹都有樹序。 反過來， 假如 G 有樹序， 根據條件， 對於所有的 v_i、$1 \leq i < n$， 它都與唯一的 v_j 相鄰， 其中 $i < j \leq n$。 因此易知每一點逐一的往編號比較大的點連、 最後就有一條路徑通往 v_n， 所以 G 是連通的。 接著， 假設 G 當中有圈 C， 則考慮這個圈在樹序當中排在最前面的那個點， 於是它必然與後面至少兩點相鄰， 矛盾； 所以就得到 G 是樹。 ■

於是， 在設計各種跟樹有關的演算法時， 我們經常會用迴圈「**for** $i = 1$ **to** n **do**」、 按照給定的樹序一個一個地處理 v_i。 此時當一個點被處理的時候， 如果不看那些已經處理過的點， 則它永遠會是一片葉， 這在設計演算法的時候會有幫助。 回憶起 G 的一個生成子圖是指包含 G 的所有點的子圖， 應用這個概念， 會有所謂的**生成樹**（spanning tree）。 有一些時候， 可以藉由研究 G 的生成樹來間接了解 G 的性質。

性質 3.4. 若連通圖 G 中有一個圈 C， 則對於 C 中的任一邊 e， $G-e$ 仍是連通的。

證明： 對於 $G-e$ 中的任意兩點 u 和 v， 它們也都是 G 中的點， 而因爲 G 是連通的， 它們之間存在一條 u-v 道路 W。 對於 W 當中任何有用到 e 的地方， 把它換成 $C-e$， 這將得到一條 $G-e$ 當中的 u-v 道路， 所以 $G-e$ 是連通的。 ■

推論 3.5. 任何連通圖都有一生成樹。

證明： 設 G 爲連通圖， 取其中一個邊數最少的連通生成子圖 H。 若 H 不是樹則 H 必有圈， 但這麼一來將這個圈的其中一條邊去掉， 根據性質 3.4 就得到一個邊數更少的連通生成子圖， 矛盾， 所以 H 必爲樹。 ■

最後， 來介紹一些樹的等價定義。 對於任何一類特殊的圖， 往往都不只一種方式可以刻畫它們， 而這些不同的刻畫方式往往會在不同的場合發揮作用， 所以多了解一些刻畫方式經常會有幫助。 底下介紹六種等價定義， 不過只證明前四種的等價性， 後面兩種留給讀者（參見習題 3.1）。 首先， 一個常用的性質是：

性質 3.6. 若樹 T 有 n 個點， 則 T 恰有 $n-1$ 條邊。

證明： 用歸納法證明。 當 $n=1$ 時顯然成立； 而假設 $n\geq 2$ 且敍述對 $n-1$ 成立， 由性質 3.1 我們可以找到一片葉 v， 由性質 3.2 可知 $T-v$ 是樹。 $T-v$ 恰有 $n-1$ 個點， 故由歸納假設它有 $n-2$ 條邊， 於是就得到 T 有 $n-1$ 條邊。 ■

定理 3.7. 設圖 G 有 n 個點， 則下列敍述等價：
(1) G 是樹。
(2) G 無圈且恰有 $n-1$ 條邊。
(3) G 連通且恰有 $n-1$ 條邊。
(4) G 中任意兩點之間恰有一路徑。
(5) G 無圈，且任意加入一條新的邊都必然使得 G 有圈。
(6) G 連通且任意刪除一條邊必使得 G不連通。

證明： (1) $\Rightarrow$ (2)： 根據定義， G 本來就沒有圈， 而性質 3.6 則保證 G 有 $n-1$ 條邊。

(2) ⇒ (3)： 只需證明 G 連通。 假設 G 恰有 r 個連通部分 $G_1, G_2, \ldots, G_r$， 且分別有 $n_1, n_2, \ldots, n_i$ 個點， 於是 $n = \sum_{i=1}^{r} n_i$。 根據定義， 每一個 G_i 都是樹， 故由性質 3.6 知道 G_i 皆恰有 $n_i - 1$ 條邊， 於是 $n - 1 = \sum_{i=1}^{r}(n_i - 1) = n - r$， 得到 $r = 1$。

(3) ⇒ (1)： 由推論 3.5， G 有生成樹 T， 由性質 3.6 知道 T 有 $n - 1$ 條邊。 但是 G 也只有 $n - 1$ 條邊， 所以 $G = T$ 是樹。

(1) ⇒ (4)： 由連通性， 任兩點之間都有一條路徑。 假設存在 u 和 v 使得其間有兩條相異路徑 P 和 Q， 不妨選取 u, v, P, Q 使得 P 和 Q 的長度和 s 是所有這樣的組合中最小的。 如果 P 和 Q 除了 u 跟 v 之外還有公共點 w， 則或者 u 和 w 之間有兩條相異路徑、 或者 w 和 v 之間有兩條相異路徑， 但無論哪一種情況， 都得到兩點使得它們之間有長度和小於 s 的相異路徑， 矛盾； 所以 P 跟 Q 沒有除了 u 跟 v 以外的共用點， 於是就形成了一個圈， 矛盾。

(4) ⇒ (1)： 根據定義， G 是連通的。 如果 G 中有圈， 則這個圈上的任意兩點之間沿著圈必有兩相異路徑， 矛盾。 ■

利用性質 3.1 和 3.2 證明性質 3.6 時所用的歸納法， 在很多跟樹有關的論證當中都會用到， 底下再舉一個例子說明。

性質 3.8. 設 T 是一個恰有 k 條邊的樹， 而圖 G 的最小度 $\delta(G) \geq k$， 則 G 有一個子圖與 T 同構。

證明： $k = 0$ 時顯然成立。 設 $k > 0$， 而且性質對於 $k - 1$ 成立。 找一個 T 中的葉 v、 並設它在 T 中的鄰居爲 u， 令 $T' = T - v$， 於是 T' 是有 $k - 1$ 條邊的樹。 由歸納法假設， G 中有一個子圖 G' 和 T' 同構， 假設 G' 中對應於 u 的頂點是 u'。 則因爲 $\deg_G(u') \geq \delta(G) \geq k = |V(G')|$， 所以 u' 在 G 中必有一個鄰居 v' 不在 $V(G')$， 於是將 G' 加上點 v' 與邊 $u'v'$ 就會是與 T 同構的子圖。 ■

Erdős 和 Sós 提出一個比上面敍述更強的猜想：

猜想 3.9. （Erdős-Sós 猜想 [5]） 如果圖 G 有 n 個點和 m 條邊、 而樹 T 有 k 條邊， 則在 $m > n(k-1)/2$ 的條件下， G 會有一個與 T 同構的子圖。

容易看出， 當 $\delta(G) \geq k$ 時 $m > n(k-1)/2$ 一定會成立， 但反之則不一定。

3.3. 樹的中心問題

圖 G 中兩點 u 和 v 之間的**距離**（distance）、記作 $d_G(u,v)$ 或 $d(u,v)$，是指從 u 到 v 的道路中長度最小者之長度。如果 u 到 v 之間沒有道路，方便起見規定 $d(u,v)=\infty$；如果 G 是連通圖，$d(u,v)$ 都會是有限的。在這個定義中，把「道路」換成「路徑」並沒有差別，因爲在性質 1.5 的證明當中已經知道，最短的道路一定是一條路徑。對於任一連通圖 $G=(V,E)$，距離函數 d 是定義在 V 上的一個**量度**（metric），即它滿足下面三個條件：

(M1) **正定性：** 恆有 $d(u,v)\geq 0$，且 $d(u,v)=0$ 若且唯若 $u=v$；

(M2) **對稱性：** $d(u,v)=d(v,u)$；

(M3) **三角不等式：** $d(u,v)+d(v,w)\geq d(u,w)$。

(M1) 顯然成立。(M2) 成立是因爲一條 u-v 路徑反向列出來就是一條 v-u 路徑。而要檢查 (M3) 成立，只要注意到任何的 u-v 道路和 v-w 道路串接起來都是 u-w 道路即可。

圖 G 中一個點 u 的**離心率**（eccentricity）$e_G(u)$、或是簡記爲 $e(u)$，是指從 u 到其他點的最大距離。而一個圖 G 的**半徑**（radius）$\mathrm{rad}(G)$ 是所有點的最小離心率，**直徑**（diameter）$\mathrm{diam}(G)$ 則是最大離心率。請特別注意，雖然直觀上圖的半徑跟直徑與這兩個名詞在幾何上的意義有一些類似之處，但是並不能互通[3]，例如圖的直徑不見得是其半徑的兩倍；這個等一下會有更仔細的討論。

如果 $e(u)=\mathrm{rad}(G)$，就說 u 是 G 的一個**中心點**（central vertex）；而**中心**（center）$C(G)$ 則是指所有中心點所構成的集合。有時爲了方便敍述，$C(G)$ 也有可能是指由 $C(G)$ 所導出的子圖。

性質 3.10. 任何圖 G 都是某一個圖的中心；也就是說，能夠構造出一個圖 H，使得 $C(H)$ 與 G 同構。

[3]其實，在平面上圓的半徑，不是圓上所有點到圓上各點最大距離的最小值、而是平面上所有點到圓上各點最大距離的最小值，這與圖的定義不同。只有將圓換成圓區域，才會得到與圖相同精神的定義。

證明： 令 H 是在 G 中加上四個新點 x, y, z, w、 以及 xy, yv, vz, zw（其中 v 過 G 的所有點）這些邊所成的新圖（見圖 3.3）。 易看出 $e(x) = e(w) = 4$、 $e(y) = e(z) = 3$， 且對所有 $v \in V(G)$ 有 $e(v) = 2$ ， 因此 G 就是 H 的中心。 ■

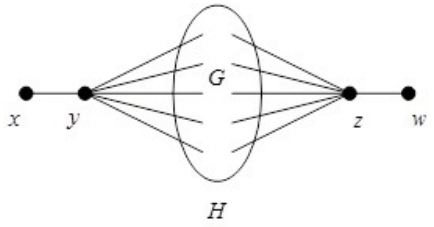

圖 3.3: 圖 H 是圖 G 再加上四點所成的圖。

由上面的性質可以知道， 一個圖的中心可以是任何樣子的圖； 然而有些特殊構造的圖的中心卻相當簡單， 其中樹就是一個例子。

定理 3.11.（Jordan [10]） 樹的中心或者恰爲一點、 或者恰含兩相鄰點。

證明： 這裡提供兩個證明， 一個較簡短， 一個有利於設計演算法求樹的中心。

證法一： 假設樹 T 有兩個不相鄰的中心 x 和 y， 以 $P(x, y)$ 表示 x 到 y 的唯一路徑。 取 $P(x, y)$ 中異於兩端點的一點 z。 令 w 是距離 z 最遠的一點、 也就是 $e(z) = d(z, w)$。 由樹中兩點間路徑唯一的性質， $P(z, w) \cap P(z, x) = \{z\}$ 或是 $P(z, w) \cap P(z, y) = \{z\}$， 不失一般性可假設前者成立， 此時 $P(y, z)$ 和 $P(z, w)$ 合成 $P(y, w)$， 所以

$$\mathrm{rad}(T) = e(y) \geq d(y, w) = d(y, z) + d(z, w) > d(z, w) = e(z)，$$

矛盾。定理得證。 □

證法二： 利用數學歸納法證明定理。 假設樹 $T = (V, E)$ 有 n 個頂點。 當 $n \leq 2$ 時定理顯然成立； 當 $n \geq 3$ 時， T 至少有一點不是葉 （否則， $n = \sum_{v \in V} \deg(v) = 2|E| = 2(n-1)$ 將導致 $n = 2$）， 此時令 L 爲 T 中所有葉的集合， 而 $T' = T - L$。 由性質 3.2， T' 是一棵樹。

對於任一點 $x \in L$， 若它在 T 中的鄰居是 y（此時必定 $y \notin L$）， 則因爲 T 中以 x 爲起點的道路都會通過 y， 所以 $e_T(x) = e_T(y) + 1$， 也就是說 $C(T) \cap L = \emptyset$。 底下對於任意的 $x \in V - L$， 將證明 $e_{T'}(x) = e_T(x) - 1$， 因此得到 $C(T') = C(T)$， 而由歸納法就可以得知 T 的中心不是一點就是相鄰兩點， 視其化簡到最後是一點還是兩點而定。

令 $r = e_T(x)$ 且 $r' = e_{T'}(x)$。 假設 $v_0, v_1, \ldots, v_r$ 是 T 中以 $x = v_0$ 爲出發點的一條最長路徑， 此時對於 $0 \le i \le r-1$ 恆有 $v_i \notin L$， 所以 $v_0, v_1, \ldots, v_{r-1}$ 是 T' 中的路徑， 由樹中兩點之間路徑的唯一性可知， $e_{T'}(x) \ge r - 1 = e_T(x) - 1$。 反過來， 令 $u_0, u_1, \ldots, u_{r'}$ 是 T' 中以 $x = u_0$ 爲出發點的一條最長路徑， 則 $u_{r'}$ 是 T' 中的葉或者是 T' 唯一的點， 不然 $u_{r'}$ 就會和 T' 中異於 $u_{r'-1}$ 的某點 u 相鄰， 因此 $u_0, u_1, \ldots, u_{r'}, u$ 是 T' 中的一條路徑， 這導致 $e_{T'}(x) \ge r' + 1$ 而矛盾。 既然 $u_{r'}$ 是 T' 中的葉或唯一點， 但又不在 L 中， 因此 $u_{r'}$ 必和某 $w \in L$ 相鄰， 得到 $e_T(x) \ge r + 1 = e_{T'}(x) + 1$。 合起來即得到 $e_{T'}(x) = e_T(x) - 1$。 □ ■

上述定理的證法二除了有利於設計演算法求樹的中心之外， 也可以用來推得 $\mathrm{rad}(T') = \mathrm{rad}(T) - 1$。 當 x 是 T 的葉而和 y 相鄰時， 不但如上述可以證得 $e_T(x) = e_T(y) + 1$， 若再多一點點論證， 其實也能說 $e_T(x) = e_{T'}(y) + 2$， 進一步推導出 $\mathrm{diam}(T') = \mathrm{diam}(T) - 2$。 所以， 其實不但可以說明樹的中心的長相， 也可以確定其直徑跟半徑之間的關係：

推論 3.12. 於任一樹 T， 或者 $C(T)$ 恰含一點， 此時 $\mathrm{diam}(T) = 2\,\mathrm{rad}(T)$； 或者 $C(T)$ 恰含相鄰兩點， 此時 $\mathrm{diam}(T) = 2\,\mathrm{rad}(T) - 1$。

對於一般的連通圖 G 而言， 我們只能說 $\mathrm{rad}(G) \le \mathrm{diam}(G) \le 2\,\mathrm{rad}(G)$。 而且， 一般來說， 若兩非負整數 a 和 b 滿足 $a \le b \le 2a$， 我們都有辦法找到一個圖 G 使得 $\mathrm{rad}(G) = a$ 且 $\mathrm{diam}(G) = b$（參見習題 3.7）。

Jordan 在他的文章中還提到了一個中心的變形概念。 定義一棵樹 T 在一點 u 的一個**分叉**（branch at u）是指一棵包含 u 爲葉的 T 之極大子樹， 換句話說， 就是 $T - u$ 的一個連通部分、 加上 u 以及 u 連到此部分的唯一邊。 u 的**分叉權重**（branch weight）$b(u)$ 是指 u 的分叉當中邊數最多者之邊數， 而 T 的諸點中分叉權重最小的點則稱爲是 T 的**質心點**（centroid vertex）。 參見圖 3.4 的例子。

需注意的是， 一個點是否爲質心點和它是否爲中心點是無關的。 例如圖 3.4 中，權重爲 8 的點是質心點卻不是中心點； 而權重爲 9 的點是中心點卻不是質心點。

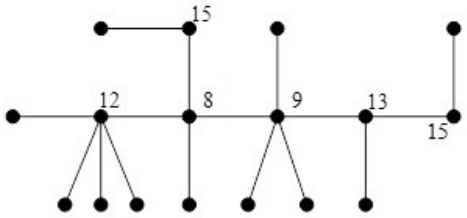

圖 3.4: 一棵樹以及其所有點的分叉權重（各葉的權重皆爲 16）。

Jordan 給出了下面這個和定理 3.11 類似的結論：

定理 3.13. （Jordan [10]） 任何樹或者恰有一質心點、 或者恰有兩相鄰質心點。

3.4. 建構生成樹

在設計圖論演算法的時候， 經常需要針對給定的連通圖 G 建立一個生成樹以了解該圖的基本架構。 推論 3.5 雖然已經告訴我們生成樹必定存在， 但卻沒有提供有一個有效構造生成樹的方法。 視演算法的需求我們會需要採取不同的方式來構造生成樹， 其中最主要有兩種方式， 分別是**深度優先搜尋**（depth-first search，簡稱 DFS）和**廣度優先搜尋**（breadth- first search，簡稱 BFS）。 利用這兩種方法產生的生成樹去處理一個圖，有些演算法會變得更容易理解、 執行得更快更有效率。

在這兩種方法當中， 都會先從 $V(G)$ 中選取一點 v 當作**樹根**（root）， 然後挑出一些邊以構成生成樹； 這些被挑出來的邊稱爲**樹邊**（tree edge）。 在生成樹架構當中，每一個點 u 和樹根 v 之間都有一條唯一的路徑， 這條路徑上的所有點（含 u 在內）通稱爲 u 的**長輩**（ancestor）， 其中與 u 相鄰的長輩則稱爲 u 的**父點**（parent 或 father）；反過來， 所有以 u 爲長輩的點（包括 u 自己）通常爲 u 的**晚輩**（descendant）， 而與 u 相鄰的晚輩則通稱爲 u 的**子點** （child 或 son）。 邊集 $E(G)$ 中沒有被選爲樹邊的邊可以分成兩種： 如果一條邊的兩個端點當中有其中一個是另一個的長輩， 就稱爲是**反向邊**（back edge）； 反之如果兩個點都不是對方的長輩， 那就稱爲**跨越邊**（cross edge）。

例如在圖 3.5 所示的生成樹中， 如果 v 爲樹根， 則 u 的長輩共有 u, v_1, v、 晚輩有 u, v_2, v_3, v_4、 父點爲 v_1、 子點有 v_2， 而 e_1 是一條跨越邊， e_2 是一條反向邊。

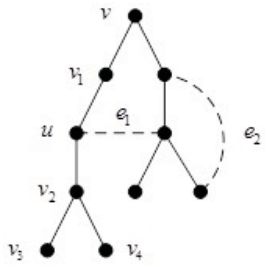

圖 3.5: 生成樹示意圖。

先來談深度優先搜尋。 DFS 的作法是， 先選擇一個點 a 做爲樹根， 然後前往 a 的一個鄰居 b、 並將 ab 選爲樹邊， 接著再前往 b 的一個還沒去過的鄰居 c（因此 $c \neq a$）、 將 bc 選爲樹邊， 依此類推， 每次都往一個還沒去過的鄰居前進、 並且把連接這兩點的邊選爲樹邊。 如此一來我們最終會來到一個點、 它所有的鄰居都已經到過了， 這個時候我們就回到最後一個尚有還沒走過的鄰居的點、 並繼續行走。 這跟第 1 章中用來尋找 Euler 迴路的演算法有點像， 當時是針對邊「能走就走、 不能走就回頭到一個較早的點， 從未走過的邊走完之後再插入」； 而 DFS 則是針對點「能走就走、 不能走就回頭到一個較早的點繼續走下去」。 容易知道如果圖是連通的， 那麼這個作法將會走過每一個點。 如果圖不連通， 也可以執行 DFS， 那就在每一個連通部分做一次 DFS， 事實上應該反過來說， 是每一次 DFS 構造出一個連通部分。 DFS 在有向圖一樣適用， 差別只在每次我們只能沿著順向邊前往下一個鄰居， 這時候就可能產生誇越邊。

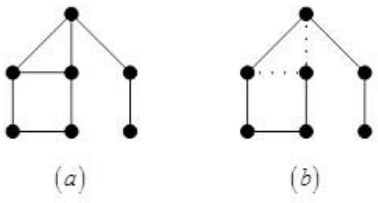

圖 3.6: (a) 原圖。(b) 以最上面的點爲樹根做 DFS 所產生的生成樹，虛線表示反向邊。

DFS 的一個特性是， 在無向圖的情況中， 邊就只有樹邊和反向邊兩種， 不會有跨

越邊出現。 這並不難理解， 因爲如果我們第一次來到 x 之後， 發現它有一個不是長輩卻是曾經到過的鄰居 y， 那麼當時在處理 y 的時候應該就已經走過 x 並把 xy 這條邊選爲樹邊才對， 這跟假設矛盾。 因此在無向圖的情況中， 我們也可以簡單地說不是樹邊的邊就是反向邊。

如果要實作 DFS 的話， 可以採用堆疊結構來記錄我們走過了哪些點、 好讓我們走到盡頭的時候可以方便倒退。 還有一個方法是利用遞迴程序（recursive procedure）來實作； 所謂的遞迴程序就是一個會呼叫自己的程序。 寫出來的話， 具體的形式會是如下所示。

演算法 3.14.（深度優先搜尋）

輸入：連通圖 $G = (V, E)$。

輸出：樹邊之集合 T（而 $E \backslash T$ 就是反向邊之集合）。

方法：

1. 將 V 中所有點 v 標爲「未到過」； $T \leftarrow \emptyset$；
2. 任選一個點 v 並執行 DFS(v)；

DFS(v)：

3. 將 v 標爲「已到過」；
4. 每當（v 有一個未到過的鄰居 u）{ $T \leftarrow T \cup \{uv\}$； 執行 DFS($u$)； }

對於一般圖 G， 將演演算法 2 行做適當修改如下， 每一次就可以得到一個連通部分。 這時候， T 是一個森林。

2. 每當（G 有一個未到過的點 v） 執行 DFS(v)；

接著我們來看廣度優先搜尋。 BFS 的作法是先盡可能地把同一個頂點的所有鄰居都走過， 再前往下一個頂點去搜尋其鄰居。 首先選一個點 a 當樹根， 然後把 a 的每一個鄰居 $b, c, \ldots$ 走一遍， 並且將 $ab, ac, \ldots$ 這些邊都選爲樹邊。 接著， 剛才繼 a 之後第一個被走過的點是 b， 我們就對 b 進行同樣的操作， 做完之後再繼續對下一個最早被走過的點、 也就是 c 繼續做同樣的操作。 有向圖的情況則是每次只能挑出順向邊作爲樹邊。 跟 DFS 相反地， 在無向圖的情況中， 只會出現跨越邊而不會出現反向邊， 道理跟之前的討論類似。

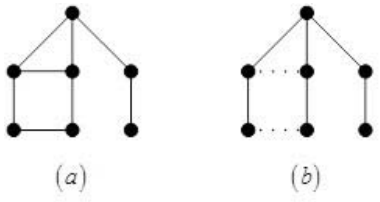

圖 3.7: (a) 原圖，(b) 以最上面的點爲樹根做 BFS 所產生的生成樹，虛線表示跨越邊。

爲了要實作這個演算法， 我們要利用到佇列的資料結構， 每次抓鄰居的時候就排在佇列後面， 而處理完一個點的時候則從佇列的前端移除。 BFS 的演算法如下所示。 與 DFS 一樣， 對於一般圖 G， 將演演算法 2 行做適當修改， 每一次就可以得到一個連通部分。 這時候， T 也是一個森林。

演算法 3.15. （廣度優先搜尋）

輸入：連通圖 $G = (V, E)$。
輸出：樹邊之集合 T（而 $E \backslash T$ 就是誇越邊的集合）。
方法：

1. 將 V 中所有點 v 標爲「未曾佇列」； $T \leftarrow \emptyset$； Q 爲空佇列；
2. 任選一個點 v 加入 Q； 將 v 標爲「曾佇列」；
3. 每當（Q 非空）{ 將 Q 的第一個元素移出，並設其爲 u； 執行 BFS(u)； }

BFS(v)：

5. 對所有（v 未曾佇列的鄰居 u）
6. { $T \leftarrow \{vu\}$； 將 u 加入 Q； 將 u 標爲「曾佇列」； }

DFS 和 BFS 在很多圖論相關的演算法中都有應用， 往後我們會看到這樣的例子。 總結來說， DFS 可用來測試圖的平面性、 決定圖的各種連通度、 找有向圖的拓撲排序、 測試有向圖的圈； 而 BFS 則可用來解決最短路徑問題、 測試圖的無弦圈、 解決網路流問題等等。

3.5. 生成樹計數

考慮所有以 $[n] = \{1, 2, \ldots, n\}$ 爲點集的圖， 其中同構但是頂點名稱不同的圖視爲相異的； 我們想要知道的是， 其中到底有幾種是樹？ 更一般而言， 給定一個圖、 甚至是重圖， 則它有多少生成樹？

第一個問題相當於是在後者當中取圖爲 K_n 的特例， 這個問題首先由 Cayley [4] 給出答案。 他原來的證明方式是用生成函數的技巧[4]， 在這裡我們準備採用 Prüfer 的證明。

對於任一棵有 $n \geq 2$ 點的樹 $T = (V, E)$， 爲了方便可假設 $V \subseteq \mathbb{N}$。 考慮 $V^{n-2} = \{(a_1, a_2, \ldots, a_{n-2}) : a_i \in V\}$ 這個集合， 並考慮透過底下的演算法定義映射 $f : \mathcal{T} \to V^{n-2}$（其中 $\mathcal{T}$ 表示所有這種樹構成的集合）：

令 $T_1 \leftarrow T$， 每次取 T_i 中編號最小的葉 b_i、 其鄰居爲 a_i、 並令 $T_{i+1} \leftarrow T_i - b_i$。

也就是說， 每次都拔掉一片葉， 並且把跟該葉相鄰的點紀錄起來。 當 $i \leq n-2$ 時， a_i 不是 T_i 的葉， 更不會是 T 的葉。 這樣得到的序列 $f(T)$ 稱爲是樹 T 的 **Prüfer 碼**（Prüfer code）。 由於 $|V^{n-2}| = n^{n-2}$， 如果我們能說明 f 是一個**對射**（bijection，即一對一映成函數）， 那就可以得到：

定理 3.16.（Cayley [4]） 恰有 n^{n-2} 棵樹其點集 $V \subseteq \mathbb{N}$ 且 $|V| = n$。

證明： 定理當 $n = 1$ 時顯然成立。 對於 $n \geq 2$ 的情況， 我們準備用歸納法證明上面定義的 f 是一個對射。 這當 $n = 2$ 時成立， 因爲只有一棵兩點的樹， 而 V^0 當中也只有空序列 ( ) 而已。

接著假設定理對 $n = k$ 成立， 則當 $n = k+1$ 時， 對任何 $a = (a_1, a_2, \ldots, a_{n-2}) \in V^{n-2}$， 要求 $f(T) = a$ 的解。 注意到因爲 $n > 2$， 表示 $a_1, a_2, \ldots, a_{n-2}$ 都不會是 T 的葉。 同時， 這樣的 T 當中編號最小的葉一定是 V 當中沒有出現在 a 裡面的最小數（不妨稱爲 x）， 所以 T 一定有一個邊爲 xa_1。 考慮 $a' = (a_2, a_3, \ldots, a_{n-2}) \in (V \backslash \{x\})^{n-2}$， 由歸納假設， 恰存在一個不包含 x 的樹 T' 使得 $f(T') = a'$， 而將 T' 加回 x 與邊 xa_1 就得到 $f(T) = a$ 的唯一解。 ■

[4]用生成函數的求法也相當簡潔但須略多預備知識， 有興趣的讀者可以參考 [6]。 利用生成函數的手法， 可以有效地處理非常多不同種類關於樹的計數問題， 該書中同樣對這些技巧有詳細的討論。

一般而言， 對於任何近圖 G 我們可以定義其生成樹的個數爲 $\tau(G)$。 而 Cayley 定理便是在說明 $\tau(K_n) = n^{n-2}$。 爲了計算一般的 $\tau(G)$， 可以引入「**收縮**（contract）」的概念。

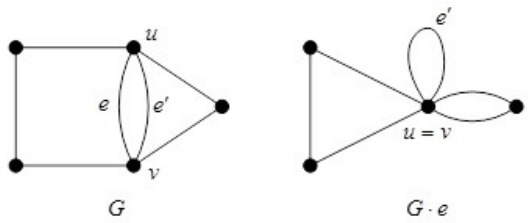

圖 3.8: 圖 G 及 $G \cdot e$。

給定近圖 G 當中的一個非迴邊 e， 將該邊與其兩端點「合成」爲一點， 得到的新近圖稱爲 G 的**收縮圖**（contraction）， 以 $G \cdot e$ 表示（或者某些書上會用 G/e）。 例如圖 3.8 就是一個收縮的例子； 在新圖 $G \cdot e$ 裡面， 將 u 跟 v 變成同一點， 讓 e 消失， 並且將本來其他的連邊關係繼續維持。 注意到收縮一條邊的結果可能會產生重邊與迴邊。

性質 3.17. 若 e 是近圖 G 中一條不是迴邊的邊， 則 $\tau(G) = \tau(G - e) + \tau(G \cdot e)$。

證明： 任何 G 的生成樹都可以分成兩種， 一種是不包含 e 的、 一種是包含 e 的。 第一種的數目即爲 $\tau(G - e)$； 而第二種可以和 $G \cdot e$ 的生成樹一一對應。 考慮 G 的一個包含 e 的生成樹 T， $T \cdot e$ 就會是 $G \cdot e$ 的生成樹； 反之對於 T' 的生成樹 $u = v$， 將 $u = v$ 那一點「擴張」成兩點並加回一條邊， 就會得到 G 的生成樹了。 ■

因爲 $G - e$ 與 $G \cdot e$ 都比原圖小， 上面固然是給出了一個遞迴關係， 但用這個方法計算是不理想的， 甚至可以說是很糟糕的， 因爲每次做遞迴時圖的個數都變成兩倍， 而且可以預期地， 經常會需要經過非常多次遞迴關係才能得到便於計算的小圖， 這使得計算量跟資料儲存需求都大得驚人。 而底下應用矩陣來解決這個問題則會是一個比較好的方法。 以下的內容會用到一些線性代數的知識。

將近圖 G 當中的迴邊去掉並不會影響 $\tau(G)$ 的值， 所以我們只需討論重圖 G 的 $\tau(G)$ 就夠了。 對於任意的重圖 G， 設其點集爲 $\{v_1, v_2, \ldots, v_n\}$， 考慮 $n \times n$ 矩陣

$Q=(a_{i,j})$ 定義如下： 當 $i=j$ 時， $a_{i,j}=\deg_G(v_i)$， 而當 $i\neq j$ 時， v_i 和 v_j 之間恰有 $-a_{i,j}$ 條重邊。 以 K_n 爲例， 其矩陣就會型如：

$$\begin{bmatrix} n-1 & -1 & -1 & \cdots & -1 \\ -1 & n-1 & -1 & \cdots & -1 \\ -1 & -1 & n-1 & \cdots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & -1 & \cdots & n-1 \end{bmatrix}_{n\times n}$$

定理 3.18.（矩陣-樹定理，Matrix-Tree Theorem） 若將 Q 的第 s 列與第 t 行去掉後得到 Q^*， 則 $\tau(G)=(-1)^{s+t}\det(Q^*)$。

我們先來看例子。 仍以 K_n 爲例， 若我們取 $s=t=1$， 則矩陣的長相跟本來是一樣的， 只是階數降了一階而已。 利用列（行）運算操作如下： 首先將第 1 列以後的每一列減去第 1 列， 會得到：

$$\begin{bmatrix} n-1 & -1 & -1 & \cdots & -1 \\ -n & n & 0 & \cdots & 0 \\ -n & 0 & n & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -n & 0 & 0 & \cdots & n \end{bmatrix}_{(n-1)\times(n-1)}$$

然後再把除了第 1 行以外的每一行都加到第 1 行， 就會得到如下的矩陣。 其行列式值顯然爲 n^{n-2}， 因此 $\tau(K_n)=n^{n-2}$。 這樣就重新驗證了 Cayley 定理。

$$\begin{bmatrix} 1 & -1 & -1 & \cdots & -1 \\ 0 & n & 0 & \cdots & 0 \\ 0 & 0 & n & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & n \end{bmatrix}_{(n-1)\times(n-1)}$$

矩陣-樹定理的證明需要一些預備知識。 首先， 在第 2 章（以及其習題中）曾談過如何用相鄰矩陣和相連矩陣來表示一個圖。 對於重圖的情況中這兩種表示法一樣適用， 只是這次相鄰矩陣的元素變成是指兩個頂點之間的邊數， 而相連矩陣可能會有重複的行而已。

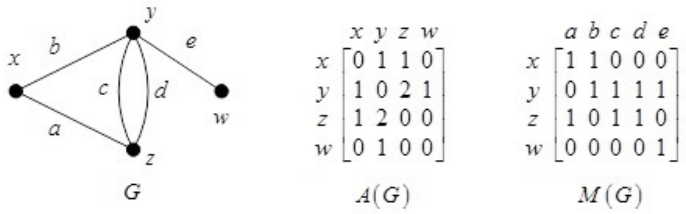

圖 3.9: 圖 G、其相鄰矩陣 $A(G)$ 與 相連矩陣 $M(G)$ 。

對於一個無向圖 G， 我們可以任意地指定邊的方向而得到一個有向圖 G'， 而任何一種這樣的 G' 都稱爲是 G 上的一個**定向**（orientation）。 例如圖 3.10 中的 G' 就是圖 3.9 中的 G 的一個定向。

對於有向圖， 相鄰矩陣與相連矩陣的定義略有不同。 相鄰矩陣仍舊是以點爲行列，不過矩陣中第 i, j 元的數字是表示從 v_i 到 v_j 有多少條邊； 相連矩陣當中， 如果有向邊是從該點出發則記爲 1， 如果是在該點結束則記爲 -1。

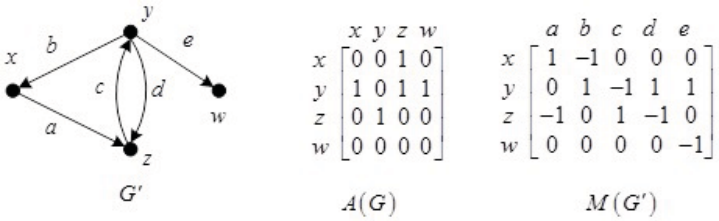

圖 3.10: 圖 G 的一個定向 G'、其相鄰矩陣 $A(G')$ 與 相連矩陣 $M(G')$。

現在回到先前的重圖 G 以及 $n \times n$ 矩陣 Q。 首先需要一些引理。

引理 3.19. 設 D 爲重圖 G 的任何一種定向， M 是 D 的相連矩陣， 則 $Q = MM^{\mathrm{T}}$。

證明： MM^{T} 的 i, j 元表示 M 的第 i 列和 M 的第 j 列的內積。 當 $i = j$ 時， 這個內積等於第 i 列非 0 的項數、 也就是 $\deg_G(v_i)$； 當 $i \neq j$ 時， 這個內積等於 v_i 和 v_j 之間邊數的負值。 這都跟 Q 的情況相符。 ■

引理 3.20. 設 B 爲 M 的某個 $(n-1) \times (n-1)$ 子矩陣， 則 $\det B = \pm 1$ 若 B 的行對應的 $n-1$ 條邊構成 G 的生成樹； 不然 $\det B = 0$。

證明： 如果B 的行對應的 $n-1$ 條邊構成 G 的生成樹， 我們對 n 做歸納法。 當 $n=1$ 時， 根據通例， 空矩陣的行列式值爲 1。 設引理對 $n=k$ 成立， 則當 $n=k+1$ 時， 令 T 爲B 的行對應的邊所構成的生成樹， 則因爲 T 至少有兩片葉， B 裡面就至少有一列是對應於 T 中的葉（不妨稱爲 x）。 因爲 x 是 T 的葉， 所以 B 的 x 列當中只有一個非零元素。 如果我們對 B 的 x 列做降階以計算行列式值， 則得到的矩陣 B' 就對應於 $G-x$ 的生成樹（即 $T-x$）， 於是由歸納法假設得知 $\det B' = \pm 1$， 因此 $\det B = \pm 1$。

如果 B 的行對應的 $n-1$ 條邊不構成 G 的生成樹， 則由定理 3.7 等價關係 (2)， 這些邊會包含有一個圈 C。 如果我們把那些構成 C 的邊對應的那幾行挑出來， 把順向邊乘以 1、 逆向邊乘以 -1 加起來的話就會得到零（因爲頭尾互相抵銷）， 表示這幾行是線性相關的， 於是 $\det B = 0$。 ■

引理 3.21.（Binet-Cauchy 公式） 設 A 爲 $n\times m$ 矩陣， B 爲 $m\times n$ 矩陣。 給定 $S\subseteq[m]$ 使得 $|S|=n$， 令 A_S 爲那些由 S 所標示的行所構成的 $n\times n$ 矩陣， 而 B_S 爲那些由 S 所標示的列所構成的 $n\times n$ 矩陣。 若 $C=AB$， 則 $\det C = \sum_S (\det A_S)(\det B_S)$， 其中 S 過 $[m]$ 的所有 n-子集。

證明： 考慮恆等式

$$\begin{bmatrix} I_m & 0 \\ A & I_n \end{bmatrix}\begin{bmatrix} -I_m & B \\ A & 0 \end{bmatrix} = \begin{bmatrix} -I_m & B \\ 0 & AB \end{bmatrix},$$

注意到 $\begin{vmatrix} -I_m & B \\ 0 & AB \end{vmatrix} = (-1)^m \det C$ 以及 $\begin{vmatrix} I_m & 0 \\ A & I_n \end{vmatrix} = 1$， 同時不難看出

$$\begin{vmatrix} -I_m & B \\ A & 0 \end{vmatrix} = (-1)^m \sum_S (\det A_S)(\det B_S),$$

於是就得到了結論。 ■

定理 3.18 的證明： 在這裡只證明 $s=t$ 的情況， 而一般的結論可以由線性代數中的定理 （如果矩陣的所有列向量總和爲零向量， 則每一行當中的各個餘因子都相等）推出來。

假設 M^* 是把 M 的第 t 列去掉後得到的矩陣， 則由引理 3.19， 容易看出有 $Q^* = M^*(M^*)^{\mathsf{T}}$。 假如 $m < n-1$ 的話， 則由 Binet-Cauchy 公式知道 $\det Q^* = 0$，

而另一方面， 確實 G 沒有生成樹， 因爲 G 根本就不連通。 因此我們可以假設 $m \geq n-1$。 再次根據 Binet-Cauchy 公式， 可見 $\det Q^* = \sum_B (\det B)^2$， 其中 B 過 M^* 的所有 $n-1$ 階子方陣。 由引理 3.20， $(\det B)^2 = 1$ 若且唯若對應的邊是生成樹， 而 B 又通過所有可能的組合， 因此就計算到了所有的生成樹恰一次。 ■

3.6. 最小生成樹

一個圖的生成樹無論是點數或邊數都是固定的， 所以在這個觀點下每一個生成樹都是一樣大的； 不過， 在實際的應用當中， 每條邊代表的意義可能會略有不同。 例如在本章一開始的網路架設問題當中， 每條網路線所對應的架設費用就不太一樣， 而目標是要找一個總值加起來最小的連通生成子圖。 這種時候， 我們很自然地會尋求生成樹， 因爲這是連通生成子圖當中邊最少的。

一般而言， 假設 G 是一個連通圖， 而且每條邊 e 上面都有賦予一個**權重**（weight）$w(e)$， 而希望求一個 G 的生成樹 T 使得總重 $w(T) = \sum_{e \in E(T)} w(e)$ 最小。 $w(e)$ 通常是一個實數， 有的時候也會加上非負的條件。 在上面的例子當中， $w(e)$ 就是在 e 上面架設一條線路的成本； 此時恆有 $w(e) \geq 0$， 有可能會等於零（例如某兩個工作站之間本來就已經有網路線了）。 假如權重有可能是負值的話， 那最小生成樹就不見得會是總重最小的連通生成子圖， 但這邊先不處理這種情況。

如何求最小生成樹的問題有很多種解法， 其中最爲人所知的就是 Kruskal 的**貪求法**[5]（greedy algorithm）。 這個方法是從 G 的無邊生成子圖開始， 將 G 中的邊依權重由小到大逐一嘗試加入子圖， 如果加入之後不會產生圈就將邊加入， 否則就跳過， 繼續看下一條邊。 寫成演算法的話， 會是下面的型式：

[5]「貪求法」這個名詞並不是專指這個 Kruskal 的演算法， 這個名稱是用來稱呼一類型的演算法， 其共同特徵是這些演算法都企圖在每一個步驟皆達到對當時來說最好的結果。 舉一個例子來說， 馬拉松比賽的貪求法就相當於是打從一開始就全力衝刺、 並且在每一個時間點都一直試圖維持著自己當時能夠達到的最高跑步速度。 由這個例子就可以了解到， 貪求法不見得能夠很有效地解決問題， 因爲任何稍微有點常識的人都知道在長跑比賽當中應該要適度保留體力， 才能讓整體的成績最好。 雖然貪求法具有這種會因爲著眼於局部而忽略全域性質的弱點， 但是它卻是最直觀的一類演算法， 而且有許多問題都是只要用貪求法就可以簡單地解決（包括這邊的最小生成樹問題在內）。 在本書後面的章節也會有其他應用在別的問題上的貪求法的例子。

演算法 3.22.（Kruskal 的貪求法 [13]）

做排序 $w(e_1) \le w(e_2) \le \ldots \le w(e_m)$；$T \leftarrow \emptyset$；
讓 i 從 1 到 m { 若 $T + e_i$ 沒有圈則 $T \leftarrow T + e_i$； }
輸出 T；

上面的演算法輸出的只有邊， 不過我們可以想像輸出的圖中也包含了所有的點。

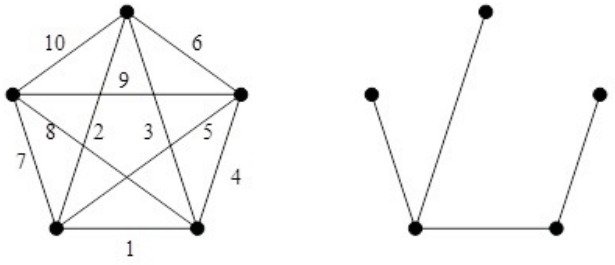

圖 3.11: 五個工作站的網路架設示意圖。

圖 3.11 慮一個有五個工作站的狀況， 在每條邊上架設網路線所需要的經費分別如其左圖所示。 其中最左邊的工作站因爲要穿越河流， 因此通往該工作站架設成本都比較高一些。

依照貪求法， 我們先將成本爲 1 的最底下那條邊加入， 然後是 2； 但是 3 就不能加入， 因爲加入 3 的話會構成圈， 所以跳過， 加入 4； 5 跟 6 也都不能加入， 所以加入 7， 在那之後都沒有其他邊可以加入了， 所以演算法最後會輸出右邊的圖。 這確實是生成樹， 而且如果讀者願意， 可以驗證這確實是最小的生成樹。

對於任何演算法， 一方面需要確定其正確性， 另一方面也需要探究其時間複雜度， 這裡先針對第一點討論。 要證明這個演算法是正確的， 需要用到下面這個跟生成樹有關的性質：

性質 3.23. 若 T 跟 T' 都是 G 的生成樹， 則對於任一條邊 $e \in E(T) \backslash E(T')$， 恆存在 $e' \in E(T') \backslash E(T)$ 使得 $T' + e - e'$ 是 G 的生成樹。

證明： 由於 T' 是生成樹， 根據定理 3.7 的等價關係 (5)， $T' + e$ 一定有一個圈 C。 但是 T 沒有圈， 因此 C 中一定有一條邊 e' 是 T 沒有的。 考慮 $T'' = T' + e - e'$ 這個

圖， 根據性質 3.4， T'' 是連通的。 又因爲 T'' 恰有 $n-1$ 條邊， 因此根據定理 3.7 的等價關係 (3)， T'' 是樹， 而且用到了 G 的每一點， 因此是 G 的生成樹。 ■

定理 3.24. *Kruskal* 的貪求法可以找到賦權連通圖的一個最小生成樹。

證明： 首先要確定這個演算法輸出的 T 是一個生成樹。 演算法從頭到尾都沒有加入任何一條使得 T 會有圈的邊， 因此 T 是無圈的。 如果 T 不是 G 的生成樹， 由定理 3.7 的等價關係 (5) 可知， 我們可以在 T 中加入一條邊 e 使得它仍然無圈； 然而， 在貪求法的過程中因此會將 e 加入 T 中， 矛盾。 故 T 是 G 的生成樹。

令 T^* 是 G 的某個最小生成樹， 不失一般性可以假設 T^* 在 G 的最小生成樹當中與 T 有最多的共用邊。 假如 $T = T^*$ 則我們已經做完了； 不然， 選取一條權重最小的邊 $e \in E(T)\backslash E(T^*)$， 於是根據性質 3.23， 我們可以選取 $e' \in E(T^*)\backslash E(T)$ 使得 $T' = T^* + e - e'$ 是 G 的生成樹。 由於 e' 以及 T 當中所有權重比 e 小的邊都在 T^* 中、 它們不含圈， 所以演算法進行到 e 的時候 e 跟 e' 這兩條邊都還可以用， 因此有 $w(e) \leq w(e')$， 於是 $w(T') \leq w(T^*)$， 但這表示 T' 也是一個最小生成樹； 然而 T' 跟 T 從頭算起有著更多相同的邊， 這與 T^* 的選取法矛盾。 ■

接下來要談這個演算法的時間複雜度。 跟一般在討論時間複雜度問題時相同， 想問的是整個演算法的需時跟輸入資料大小之間的關連如何； 而在這裡用的是邊數（即 m）當輸入大小。 基本上， 整個演算法分成兩個部分， 一個是排序， 另一個則是不斷地判斷每次加入邊之後有沒有圈產生。 排序的部分在演算法學當中另有深入的探討， 這邊不多說； 例如如果採用**堆積排序法**（heap sort）， 則排序需要的時間會是 $O(m \log m)$。

至於第二個階段， 這邊遇到最主要的問題是， 怎麼判斷一個圖有沒有圈？ 如果有了這樣的判斷方法， 那麼只要連續執行 m 次， 就可以完成第二階段的工作了。 不過， 其實並不需要這樣做。 相當有趣地， 「判斷一個圖有沒有圈」這個問題本身其實跟「每次加入一條邊看看有沒有圈出現」是一樣的， 換句話說， 思考方式應該是環繞著後者， 而不是前者。 具體來說， 可以這樣做。 一開始， 在還沒有加入任何邊之前， 整個圖是完全不連通的 n 個點， 於是就先建立 n 個集合來記錄這些連通部分（即每個集合裡各放一個點）。 每當加入一條邊之後， 就把這條邊兩端連接的連通部分之點集做聯集， 以維持記錄連通部分的長相。 在這個想法之下， 當準備要加入一條邊時， 要是發現它的兩端點是屬於同一個連通部分， 那就意味著這條邊的加入必然會造成圈了。 於是， 在

這種概念之下， 第 2 章提到的互斥集聯集法恰好可以派上用場。 比較簡單的方法需時 $O(m \log m)$， 高速互斥集聯集法就更快了、 極近線性時間。 這樣就得到：

定理 3.25. *Kruskal* 的貪求法所需時間爲 $O(m \log m)$。

3.7. 習題

3.1. 試證明定理 3.7 的 (5) 與 (6) 和其他四個敍述等價。

3.2. 證明每一顆樹 T 至少有 $\Delta(T)$ 片葉。 什麼樣的樹會恰有 $\Delta(T)$ 片葉？

3.3. 若圖 G 有 $n \geq 3$ 點， 且從 G 中去掉任一點均成樹， 試求 G 的邊數， 並藉此求 G。

3.4. 若樹 T 中任一與葉相鄰的點其度至少爲 3， 證明 T 中至少存在兩片葉有共同鄰居。

3.5. 試證當 $n \geq 2$ 時正整數序列 $d_1, d_2, \ldots, d_n$ 是某棵樹的度序列之充分必要條件爲 $\sum_{i=1}^{n} d_i = 2n - 2$。

3.6. 證明推論 3.12。

3.7. (a) 證明對於任意連通圖 G 恆有 $\mathrm{rad}(G) \leq \mathrm{diam}(G) \leq 2\mathrm{rad}(G)$。 (b) 若非負整數 r 和 d 滿足 $r \leq d \leq 2r$， 試求一圖使得其半徑爲 r、直徑爲 d。

3.8. 證明若 $\mathrm{rad}(G) \geq 3$， 則 $\mathrm{rad}(\overline{G}) \leq 2$。

3.9. 證明定理 3.13。

3.10. 對 $x \in V(G)$， 令 $s(x) = \sum_{y \in V(G)} d(x, y)$。 圖 G 的**中段**（median）是指由 $s(x)$ 最小的所有 x 所構成的集合。 證明樹 T 的中段恰含一點或恰含相鄰兩點。

3.11. 設 G 是有 n 點的連通圖， 定義一個新圖 G'， 其點集爲 G 的所有生成樹所成的集合， 而兩生成樹相鄰若且唯若它們在 G 中有 $n-2$ 條共用邊。 證明 G' 是連通圖， 並決定 G' 的直徑。

3.12. 在 DFS 或 BFS 產生的生成樹中， 對每個點 v 定義其**層次**（level）爲它到樹根的距離。 試證明， 在 BFS 當中， 無論是樹邊還是跨越邊， 它所連結的兩點之層次至多差 1。

3.13. 針對下面各個問題， 設計出對應的演算法。

(1) 求出一個連通圖中兩點 x 和 y 的距離。

(2) 判斷一個圖是否爲二分圖。

3.14. 求圖 3.12 中有 16 點 16 邊的圖的生成樹個數。

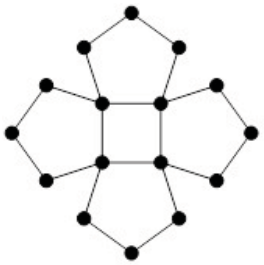

圖 3.12: 有 16 點 16 邊的圖。

3.15. 求 $K_n - e$ 的生成樹個數， 其中 e 是 K_n 的任意一條邊。

3.16. 令 G_n 是如圖 3.13 所示具有 $2n$ 點和 $3n-2$ 邊的圖。 證明當 $n \geq 3$ 時， $\tau(G_n) = 4\tau(G_{n-1}) - \tau(G_{n-2})$。 當 $n \geq 1$ 時，求 $\tau(G_n)$。

圖 3.13: 圖 G_n。

3.17. 令 H_n 表示加一點連到 P_n 的所有點所成的圖， 如圖 3.14 所示。 試證明當 $n \geq 3$ 時， $\tau(H_n) = 3\tau(H_{n-1}) - \tau(H_{n-2})$。 當 $n \geq 1$ 時，求 $\tau(H_n)$。

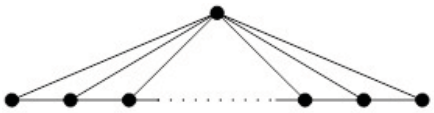

圖 3.14: 圖 H_n。

3.18. 試求 $\tau(K_{m,n})$。

3.19. 若 T 和 T' 是連通圖 G 的兩生成樹， 對任一 $e \in E(T)\backslash E(T')$， 證明存在 $e' \in E(T')\backslash E(T)$ 使得 $T - e + e'$ 和 $T' + e - e'$ 都是 G 的生成樹。

3.20. 假如 G 是一個邊賦權連通圖， 其各邊的權重都相異。 證明 G 恰有一最小生成樹。

3.21. Prim 演算法以下面方法產生一邊賦權連通圖 G 的最小生成樹： 最初先選定一點 v_0， 令 $S = \{v_0\}$、 $E = \emptyset$ 並逐漸擴大直到 $S = V(G)$ 爲止； 擴大方法是， 每次由 S 和 $\overline{S}$ 之間找出一條權重最小的邊 xy、 其中 $x \in S$ 而 $y \in \overline{S}$， 將 y 加入 S 並將 xy 加入 E。 試證明當演算法結束時， E 是 G 的最小生成樹。

3.22. 對一給定的邊賦權連通圖 G 中的一個生成樹 T， 令 $m(T)$ 表示 T 中的最大權重。 若 x 表示所有生成樹中 $m(T)$ 的最小值， 且 T^* 是一最小生成樹， 試證明 $m(T^*) = x$。 這種滿足 $m(T) = x$ 的生成樹 T 稱爲是 G 的**瓶頸**（bottleneck）生成樹。

3.8. 參考文獻

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley, Reading, Mass., 1975.

[2] B. Bollobás, Almost every graph has reconstruction number three, *J. Graph Theory*, vol. 14 (1990), pp. 1–4.

[3] J. A. Bondy, On Ulam's conjecture for separable graphs, *Pacific J. Math.*, vol. 31 (1969), pp. 281–288.

[4] A. Cayley, A theorem on trees, *Quart. J. Math.*, vol. 23 (1889), pp. 376-378.

[5] P. Erdős, Extremal problems in graph theory, in: *Theory of Graphs and its Applications*, (M. Fiedler, eds.), Academic Press, 1965, pp. 29-36.

[6] I. P. Goulden and D. M. Jackson, *Combinatorial Enumeration*, Dover Publications, Inc., Mineola, NY, 2004. pp. 174.

[7] F. Harary, On the reconstruction of a graph from a collection of subgraphs, in: *Theory of Graphs and its Applications*, (*Proc. Sympos. Smolenice*, 1963). Publ. House Czechoslovak Acad. Sci., Prague, 1964, pp. 47–52.

[8] F. Harary, A survey of the reconstruction conjecture, *Graphs and Combinatorics*, *Lecture Notes in Mathematics* 406, Springer, 1974, pp. 18-28.

[9] F. Harary and E. Palmer, On the problem of reconstructing a tournament from sub-tournaments, *Monatsh. Math.*, vol. 71 (1967), pp. 14–23.

[10] C. Jordan, Sur les assemblages de lignes, *J. Reine Angew. Math.*, vol. 70 (1869), pp. 185-190.

[11] P. J. Kelly, A congruence theorem for trees, *Pacific J. Math.*, vol. 7 (1957), pp. 961-968.

[12] W. L. Kocay, A family of nonreconstructible hypergraphs, *J. Combin. Theory Ser. B*, vol. 42 (1987), pp. 46–63.

[13] J. B. Kruskal, Jr., On the shortest spanning subtree of a graph and the traveling salesman problem, *Proc. Amer. Math. Soc.*, vol. 7 (1956), pp. 48-50.

[14] B. D. McKay, Small graphs are reconstructible, *Australian J. Combin.*, vol. 15 (1997), pp. 123–126.

[15] C. St. J. A. Nash-Williams, The Reconstruction Problem, in: *Selected Topics in Graph Theory*, 1978, pp. 205–236.

[16] P. V. O'Neil, Ulam's conjecture and graph reconstructions, *Amer. Math. Monthly*, vol. 77 (1970), pp. 35-43.

[17] H. Prüfer, Neuer beweis eines satzes über permutationen, *Arch. Math. Plys.*, vol. 27 (1918), pp. 742-744.

[18] P. K. Stockmeyer, The falsity of the reconstruction conjecture for tournaments, *J. Graph Theory*, vol. 1 (1977), pp. 19–25.

[19] P. K. Stockmeyer, A census of non-reconstructable digraphs, I: six related families, *J. Combin. Theory Ser. B*, vol. 31 (1981), pp. 232–239.

[20] S. M. Ulam, *A collection of Mathematical Problems*, Wiley, New York, 1960, pp. 29.

[21] M. von Rimscha, Reconstructibility and perfect graphs, *Discrete Math.*, vol. 47 (1983), pp. 283–291.

[22] Y. Yang, The reconstruction conjecture is true if all 2-connected graphs are reconstructible, *J. Graph Theory*, vol. 12 (1988), pp. 237–243.