簡介圖論演算法

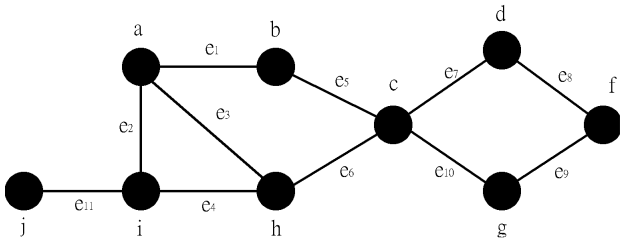
黃國卿

1736年, 瑞士數學家尤拉解決了七橋問題。從此開啓圖論這門學問。它與拓樸學、代數學、組合數學等學科關係極爲密切。其應用也極爲廣泛, 已經滲透到物理學、化學、電子學、生物學、經濟學、系統工程及計算機科學等學科領域。基於上述原因, 引起了越來多的人對圖論的興趣與重視。

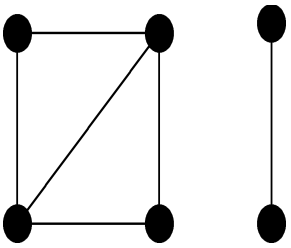
圖論的問題, 基本上有兩大類, 一是存在性問題, 一是最佳化問題。本文所討論的都是這兩類的例子。1920年, 當電腦被引進後, 尤其是演算法與圖論的結合, 更促進了圖論的蓬勃發展。我們將以一些例子說明如何利用演算法解決圖論的問題。

什麼是演算法呢? 粗略地說, 一個解決問題的方法就可稱之爲演算法 (Algorithm)。由於想借助電腦來幫我們計算, 所以它的敍述方式很像電腦程式。其結構大致分爲輸入、執行步驟、輸出三部份。

以下我們介紹本文會用到的一些圖論名詞。圖是由一些點 (有限或無窮), 及一些連接兩點的邊所形成的, 記作 G; 圖的點所形成的集合, 稱爲點集, 記作 V; 圖的邊所形成的集合, 稱爲邊集, 記作 E。所以一個圖可用 $G = (V, E)$ 來表示。如果限制兩點之間最多只連一個邊, 這樣的圖稱爲單圖。本文中的圖都是有限的單圖。如果圖 H 滿足 $V(H) \subseteq V(G)$ 且 $E(H) \subseteq E(G)$ 則稱 H 是 G 的一個子圖。如果 H 是 G 的子圖且 $V(H) = V(G)$, 則稱 H 是 G 的生成子圖。假設 v 是 G 中的一個點。點 v 所連的邊數稱爲 v 的秩。比較 G 中所有點的秩, 最大的秩數稱爲 G 的最大秩, 記作 $\Delta(G)$。圖 G 中的一條路徑是一個有限非空的點和邊的交錯序列, 其中的點兩兩不相同。如圖一, $P = ae_1be_5ce_7d$, 是一條 (a, d)- 路徑。點 a 稱爲起點, 點 d 稱爲終點。爲簡化起見, 可將它記爲 $P = abcd$。又 $Q = abchia$ 也是一條路徑, 但起點與終點相同, 這樣的路徑稱爲圈。路徑 (圈) 的長度是指其所含的邊數。如 P 的長度爲3。奇圈是指長度爲奇數的圈, 如 $Q = abchia$; 偶圈是指長度爲偶數的圈, 如 $R = cdfgc$。假設點 x、點 y 是圖 G 中的任意兩點, 若存在一條 (x, y)- 路徑, 則稱 G 是連通圖, 如圖一。反之, 則稱 G 爲非連通圖, 如圖二。

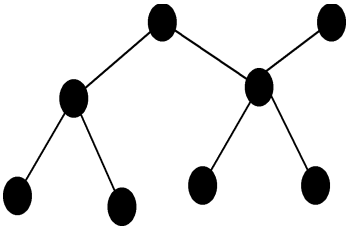


圖一



圖二

假設 G 是一個連通圖且 e 是 G 的一個邊。如果 G 去掉 e 這個邊後形成一個非連圖, 則稱 e 是 G 的一座橋, 如圖一中的 e_{11}。一個不含有圈的圖稱爲無圈圖, 而一個連通的無圈圖稱爲樹圖。如圖三。



圖三

關於樹圖, 有個簡單的定理: 一個樹圖的點數比邊數多一。對 G 中的每一個邊 e 賦以一個實數值 $W(e)$, 此值稱爲 e 的權值。賦了權的圖稱爲賦權圖。而一個子圖的權是指子圖上所有邊的權值加總。

本文所敍述的演算法, 我們只說明想法, 證明則予以省略。有興趣的讀者, 可自行試試看或閱讀相關的參考書籍。

一、圖的連通性

首先考慮如下的問題, 給定一個圖 G, 如何判斷 G 是否爲連通圖, 換句話說, 對 G 中的任意兩點 x, y, 是否存在 (x, y)-路徑? 如果能畫出 G, 就很容易知道 G 是否爲連通圖。如圖1.1。然而, G 的邊或點很多時, 想畫出 G 會有些困難, 有時也不經濟。不過, 可透過電腦記錄圖 G (如記錄 G 的點邊關係)。此時就需要用演算法來檢驗 G 的連通性。

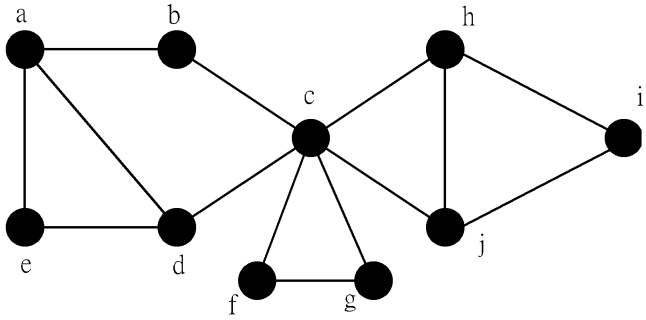


圖1.1 (a) 連通圖

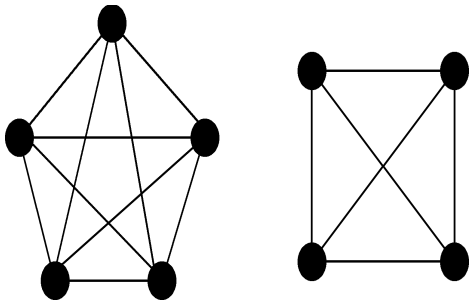


圖1.1 (b) 非連通圖

假設 G 有 n 個點, 檢驗 G 是否爲連通圖的想法如下。任選一點 v 且標爲1號。然後找出一個跟 v 相連且未標號的點, 如找到點 u。將點 u 標爲2號並記錄 uv 這個邊。從點 u 出發, 重覆前面的步驟, 直到找到一點 z 且跟 z 相連的點均已有標號。(此時, 已有標號的點形成一條 (v,z)-路徑。) 這時, 找出標號爲 z 的前一號的點, 記作 α, 點 α 稱爲點 z 的父親。再從 α 出發, 繼續搜尋的步驟。我們總是可以重覆"往下"搜尋或往回走再繼續搜尋的步驟, 直到所有標號 $1,2,3,\ldots,n$ 全部用完或回到標號爲1的點。如果所有標號均被使用過, 則 G 是一個連通圖, 如果還有點尚未被標號, 則 G 是一個非連通圖。

上述的想法就是所謂深度搜尋 (Depth-First Search) 的主要精神。在解決圖論的存在性或最佳化問題時, 深度搜尋是一個簡單且重要的演算法。後面還會有例子介紹。

依據深度搜尋的想法, 演算法敍述如下。

深度搜尋演算法

輸入: n 個點的圖 $G=(V,E)$

步驟1: 任選一點 v, 令 $k=1$。

$\ell(v)=k$ (點 v 的標號)

$S=\{v\}$ (收集已標過號的點)

$T=\phi$ (收集作記錄的邊)

$\alpha=v$ (當作父親也就是開始搜尋的點)

步驟2: 令 $k=k+1$。

(i) 如果 $S=V$, 則輸出結果。

(ii) 如果 $S\neq V$, 則考慮步驟2.1和步驟2.2兩種情形。

步驟2.1: 存在一點 $u\in V\backslash S$ 跟 α 相連,

令 $\ell(u)=k$, $S=S\cup\{u\}$,

$T=T\cup\{\alpha u\}$, $\alpha=u$,

重覆步驟2。

步驟2.2: 所有 $V\backslash S$ 中的點都不跟 α 相連。如果 $\ell(\alpha)=1$, 則輸出"G 是非連通圖"。

如果 $\ell(\alpha)>1$, 令 w 爲α 的父親且 $\alpha=w$,

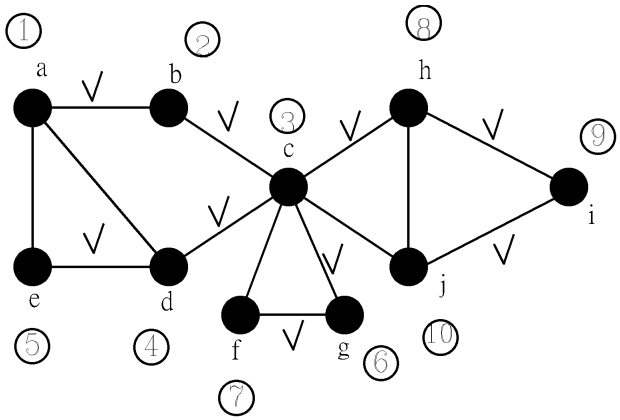
重覆步驟2.1。

輸出: 所有點的標號及 T。

我們以圖1.1來說明深度搜尋是如何執行的。假設從點 a 開始。點 a 標爲1號。點 b 與點 a 相連且未標號, 點 b 標爲2號且記錄 ab 這個邊。依此類推, 點 c、點 d、點 e 分別爲標爲3、4、5號且記錄 bc、cd、de 三個邊。此時, 跟點 e 相連的點都已標號, 於是回頭看點 d。同樣地, 跟點 d 相連的點也都已標號, 於是回頭看點 c。點 g 與點 c 相連且未標號, 點 g 標爲6號並記錄 cg 邊。重覆上述步驟, 直至點 j 標爲10號爲止。執行後的結果如圖1.2。其中圈號代表標號, 打勾代表所記錄的邊。

如果圖 G 是連通圖, 觀察執行深度搜尋演算法所記錄的邊, 可以發現由這些邊所構成的圖會是 G 的一個生成樹。從而我們可證明下面的定理。

定理: 圖 G 是連通圖若且唯若 G 含有一個生成樹。



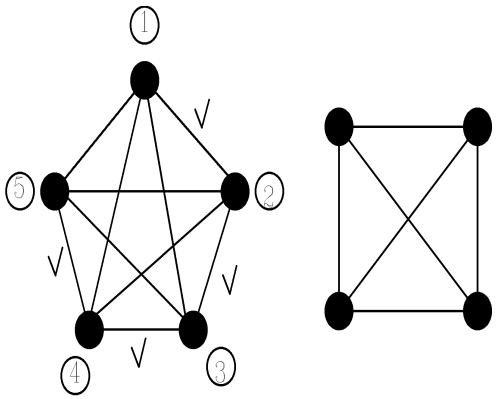


圖1.2

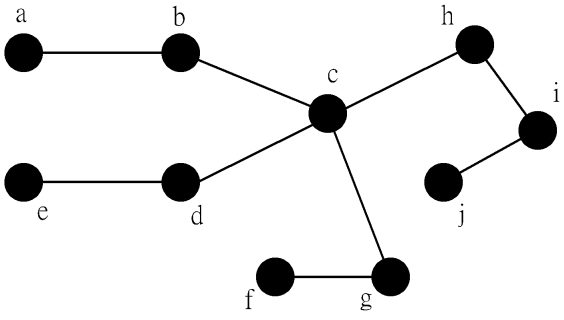


圖1.3

二、單行道問題

假設在某一城市中有一些重要的地點及連接這些地點的雙向街道。由於車輛的增加, 使得交通愈來愈擁塞且空氣污染亦愈形嚴重。於是有人建議將所有街道都改成單行道以改善交通狀況 (如台北市的公車逆向道就是一個例子)。由於從甲地到乙地的街道改成單向, 所以其他街道的行進方向就必須好好規劃才能使乙地也能到達甲地。於是問題就可簡化成如何給這些街道一個方向才能使任意兩個地點都能藉著這些單行道互相往返。以圖論的觀點來說, 給定一個連通圖, 是否能在每個邊都給予一個指向使得任何一點都能藉著邊的指向到達其他各點。如果能辦得到, 則稱此圖有一個強連通定向 (strongly connected orientation)。觀察圖 2.1, 任何其他地點都無法到達甲地, 而在乙地卻動彈不得! 所以街道的方向並不能隨意指定。觀察圖 2.2, e 這個邊是圖的一座橋。一旦給予 e 一個指向後, 如果

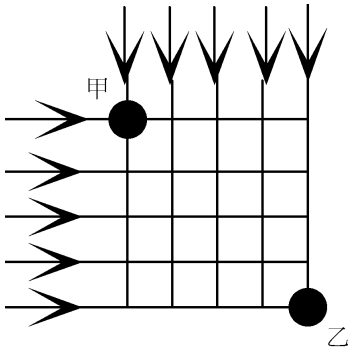


圖2.1

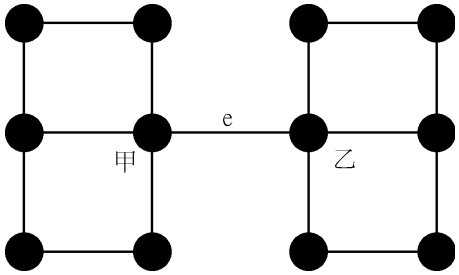


圖2.2

從甲地指向乙地, 則乙地不能到達甲地; 如果從乙地指向甲地, 則甲地不能到達乙地。於是有橋存在的圖一定沒有強連通定向。如果在圖2.2中加進一個邊, 如圖2.3, 則新的圖並沒有橋而且存在一個強連通定向。我們不禁會問, 一個沒有橋的連通圖是否有一個強連通定向呢? 這個答案是肯定的。1939年, Robbins 證明了如下的定理。

定理: 圖 G 有一個強連通定向若且唯若 G 是一個沒有橋的連通圖。

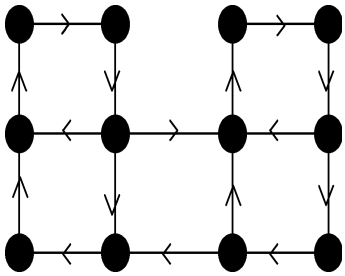


圖2.3

事實上, 如果 G 是非連通圖, 則一定沒有強連通定向。所以只需考慮連通圖即可。類似圖2.2的說明, 如果一個連通圖有橋, 則一定沒有強連通定向, 也就是說, 一個有強連通定向的圖不會有橋的存在。有待克服的是給定一個沒有橋的連通圖, 如何給予每個邊一個指向, 使之成爲一個強連通定向。這部份我們可以用深度搜尋演算法加以解決。首先對一個沒有橋的連通圖, 利用深度搜尋演算法將每一個點標號並且得到一個生成樹 T。其次, 給每個邊一個指向。任選一邊 $e = ij, i, j$ 爲標號且假設 $i < j$。如果 $e \in T$, 則由 i 指向 j; 如果 $e \notin T$, 則由 j 指向 i。將所有邊都給予指向後, 就得到圖的一個強連通定向。

再以圖1.1(a) 爲例, 執行上述的演算法, 所得結果如圖2.4。

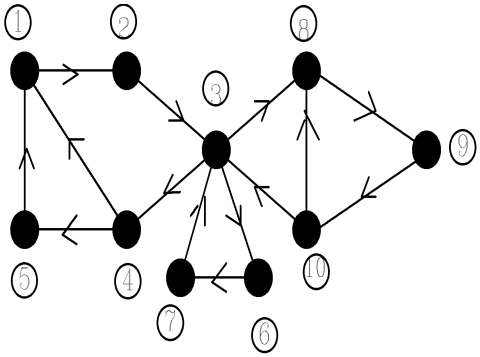


圖2.4

三、 圖的點著色 (Vertex Coloring)

圖的點著色是一個古典又有趣的課題, 它有極爲廣泛的應用, 目前仍有很多人在研究。什麼是圖的點著色呢? 簡單的說, 就是對圖的點著顏色。基於實際的理由 (如一個人不能在同一時間參加兩個會議), 還要要求任意有邊相連的兩點不可以著相同顏色。這樣的一個著色方式稱之爲圖的一個點著色法。如

圖3.1和圖3.2各用了三個顏色。如果每個點都著不一

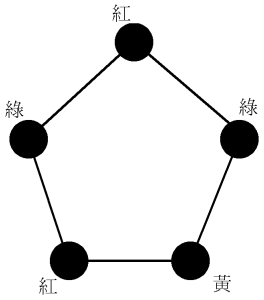


圖3.1

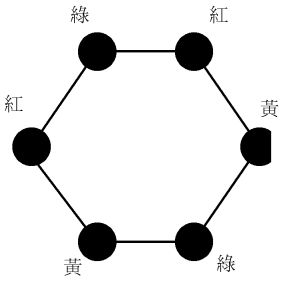


圖3.2

樣的顏色, 則一定是一個點著色法。問題是給定一個圖 G, 存在一個點著色法的最少顏色數是多少呢? 這個數目稱爲圖的點著色數, 記作 $\chi(G)$。求圖的點著色數是一個很難的問題, 到目前仍尚未完全解決。不過, 對一些特殊類的圖, 則有不少的結果。在此我們要討論點著色數爲2的圖, 這類圖通稱爲二部圖 (Bipartite graph)。

圖3.2用了三個顏色, 不過, 如果以紅綠兩個顏色依序對點著色就會是一個點著色法。因此圖3.2是一個二部圖。事實上, 所有的偶圈都是二部圖。而一個二部圖是否爲偶圈呢? 答案是否定的。如圖3.3。

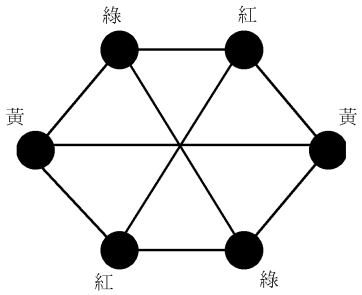


圖3.3

那麼如何判斷一個圖是否爲二部圖呢? 圖3.1是一個長度爲5的奇圈, 它的點著色數爲3。其實, 所有奇圈的點著色數都是3。因此, 一個含有奇圈的圖, 其點著色數必定大於或等於3。這保證此圖一定不是二部圖。換句話說, 一個二部圖一定不含有奇圈。可是, 一個不含奇圈的圖是否爲二部圖呢? 1936年, König證明了如下的定理。

定理: 一個圖 G 是二部圖若且唯若 G 不含有奇圈。

給定一個不含奇圈的圖, 要說明此圖是二部圖, 其實就是要找一個恰好用兩個顏色的點著色法。這並不難做到。任選一點 v, 著紅色。跟 v 相連的點著綠色。假設 u 與 v 相連, 則跟 u 相連的點都著紅色。依此類推, 直到所有的點都著了顏色。如圖3.4所示。

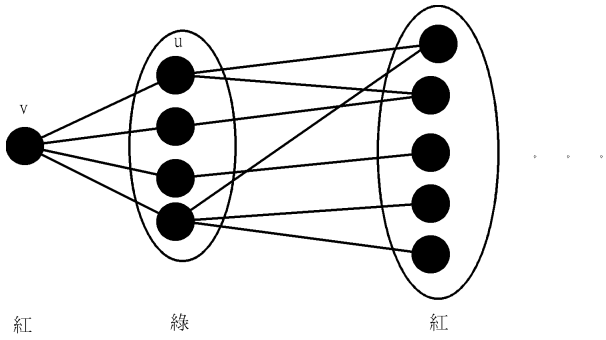


圖3.4

演算法要如何敍述呢? 因爲只用兩個顏色, 所以, 如果點 v 已著色, 則跟 v 相連的點一定要著另一個顏色。接下來, 該如何繼續呢? 將跟 v 相連的點著色, 然後將這些點依序放在一個叫做 queue 的地方, 記作 Q

(一個 queue 就像火車上放塑膠茶杯的管子, 先放進去的茶杯, 先拿出來使用)。執行演算法時, 先找出 Q 中的第一個點 v, 從 Q 中去掉點 v, 再找出所有跟 v 相連且未著色的點, 將這些點著跟 v 不同的顏色, 並依序放在 Q 的末端。重覆這樣的步驟直到所有的點都著了顏色。

以上的演算法就是所謂的廣度搜尋 (Breadth-First Search) 的主要精神。它也常用來解決圖論的問題 (找一個連通圖的生成樹也可以用廣度搜尋)。

依據上述的想法, 可將演算法敍述如下。

廣度搜尋演算法

輸入: 一個不含奇圈的連通圖。

步驟1: 令 $Q = \phi$。

步驟2: 任選一點 v, 將 v 著紅色並放進 Q 中。

步驟3: 令 u 是 Q 中的第一個點。從 Q 中去掉 u。

步驟4: 找出所有跟 u 相連且未著色的點。將這些點著綠色並把這些點依序放在 Q的末端。

步驟5: 如果所有點都已著色, 則輸出結果。否則, 重覆步驟3。

我們用圖 3.5 及表 3.1 來說明執行廣度搜尋演算法的過程和最後結果。

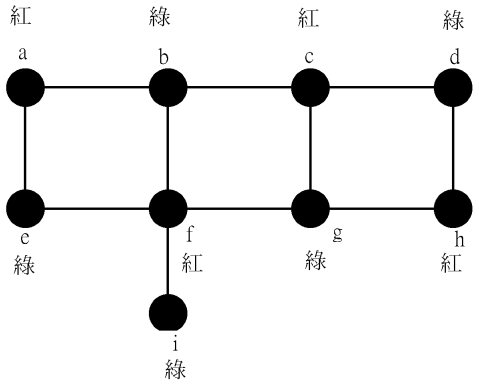


圖3.5

Q中的第一個點	點著色	Q
	a(紅)	a
a	b, e(綠)	b, e
b	c, f(紅)	e, c, f
e		c, f
c	d, g(綠)	f, d, g
f	i(綠)	d, g, i
d	h(紅)	g, i, h

表 3.1

四、最小生成樹 (Minimum Spanning Tree)

假設電信總局要在全省設立30家電信局及舖設通訊電纜。試問, 電信總局應如何舖設電纜才能使任兩家電信局都可互通訊息 (可透過轉接) 且經費花費最少? 如果不怕浪費的話, 任意兩家電信局之間都可舖設電纜, 問題就解決了! 只不過, 這樣做, 納稅人一定會抗議得沒完沒了! 有沒有更好的辦法呢? 針對問題, 我們可以構造一個賦權圖再

加以討論。令圖的點集是由所有的電信局所構成, 任意兩點連一個邊並在邊上給一個權值, 也就是造價 (造價太高的邊, 可以不予考慮)。這樣就構成了一個賦權圖。重點有兩項, 一是連通性, 要保留多少邊才能保持連通性; 一是經費, 在連通性的要求下, 要保留那些邊才能使經費花費最小。由圖論的定理得知, 一個 n 個點的連通圖至少要 $n-1$ 個邊, 而一個 n 個點的樹圖恰有 $n-1$ 個邊。再利用例子一的定理, 只要找到一個生成樹就能保持圖的連通性。把生成樹上的權值加總, 就是所需的經費。直覺上, 考慮所有可能的生成樹, 找出一個經費最少的生成樹, 問題就解決了。可不可行呢? 1857年, Cayley證明了如下的定理。

定理: 設 $n \geq 2$。則不一樣的 n 個點的樹圖共有 n^{n-2} 個。

由上面的定理得知, 需要考慮的生成樹共有 30^{28} 個! 這是一個天文數字, 用最快速的電腦也沒辦法檢驗。

我們的例子, 就是所謂的最小生成樹問題。1956年, Kruskal 解決了這個問題。主要的想法是從邊著手。給定一個 n 個點的連通賦權圖, 首先找一個權值最小的邊, 放進 T 中, (T 是用來記錄最小生成樹的邊)。在執行 Kruskal 演算法過程中, 假設 T 中已選了一些邊, 如果 T 中的邊數是 $n-1$, 則 T 就是一個最小生成樹; 如果 T 中的邊數小於 $n-1$, 則在 T 以外找一個權值最小的邊並且此邊加入 T 後不會產生圈 (因爲樹不含有圈)。將這個邊放進 T。重覆上述的步驟, 直到 T 含有 $n-1$ 個邊。爲了簡化找邊的步驟, 一開始就可將所有的邊依權值由小而大作排列 (若權值相同, 誰先誰後則無所謂), 接下來只要依序選邊並檢驗 T 是否含有圈即可。

Krukal 演算法

輸入: 一個 n 個點、m 個邊的連通賦權圖。

步驟1: 設 $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_m)$

令 $T = \phi$ (T 用來記錄最小生成樹的邊)

$t = 0$ (t 用來計算 T 中的邊數)

$k = 0$ (檢驗第 k 個邊)

步驟2: 令 $k = k + 1$

步驟2.1:

(i) 如果 $T \cup \{e_k\}$ 不含圈,

令 $T = T \cup \{e_k\}$, $t = t + 1$。接步驟 2.2。

(ii) 如果 $T \cup \{e_k\}$ 含有圈, 則重覆步驟 2。

步驟2.2:

(i) 如果 $t = n - 1$, 則計算 $w(T)$ 並輸出結果。

(ii) 如果 $t < n - 1$, 則重覆步驟2。

輸出: T 及 $w(T)$。

我們以圖 4.1、 圖 4.2 及表 4.1 來說明 Kruskal 演算法的執行過程和最後結果。首先邊的排序爲 $ab, bc, de, fg, ad, be, dg, ce, bd, cf, eg, ef$。

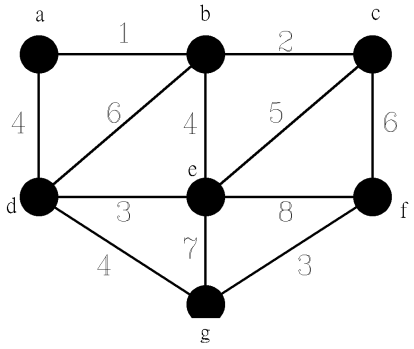


圖4.1

e_k	T	t
	ϕ	0
ab	$\{ab\}$	1
bc	$\{ab, bc\}$	2
de	$\{ab, bc, de\}$	3
fg	$\{ab, bc, de, fg\}$	4
ad	$\{ab, bc, de, fg, ad\}$	5
be	造成圈	5
dg	$\{ab, bc, de, fg, ad, dg\}$	6

表4.1

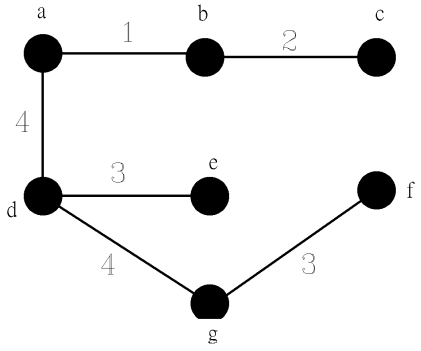


圖4.2 $w(T) = 17$

1957年, Prim 用另一種想法也解決了最小生成樹問題。主要的想法是由一點開始, 逐次加點、加邊, 使之"長成"最小生成樹。首先, 任選一點 v, 令 $B = \{v\}$, $T = \phi$。在執行 Prim 演算法過程中, 假設 B 中已有一些點, T 中已有一些邊。如果 $B = V$, 則輸出 T 及 $w(T)$。如果 $B \neq V$, 則找出一個權值最小的邊 uv 且 $u \notin B$, $v \in B$。將點 u 放進 B, 邊 uv 放進 T。重覆上述步驟, 直到 $B = V$。

Prim 演算法

輸入: 一個連通賦權圖 $G = (V, E)$。

步驟1: 令 $T = \phi$ (T 用來記錄最小生成樹的邊)。

任選一點 v, 令 $B = \{v\}$ (B 用來記錄T 中的點)。

步驟2:

(i) 如果 $B = V$, 則計算 $w(T)$ 並輸出結果。

(ii) 如果 $B \neq V$, 找出 $u \notin B$, $v \in B$ 且權值最小的邊vu。

令 $B = B \cup \{u\}$, $T = T \cup \{uv\}$。

重覆步驟2。

輸出: T 及 $w(T)$。

uv	B	T
	$\{a\}$	ϕ
ab	$\{a,b\}$	$\{ab\}$
bc	$\{a,b,c\}$	$\{ab,bc\}$
be	$\{a,b,c,e\}$	$\{ab,bc,be\}$
de	$\{a,b,c,d,e\}$	$\{ab,bc,be,de\}$
dg	$\{a,b,c,d,e,g\}$	$\{ab,bc,be,de,dg\}$
fg	$\{a,b,c,d,e,f,g\}$	$\{ab,bc,be,de,dg,fg\}$

表4.2

再以圖4.1爲例, 我們用表4.2及圖4.3來說明 Prim 演算法的執行過程和結果。

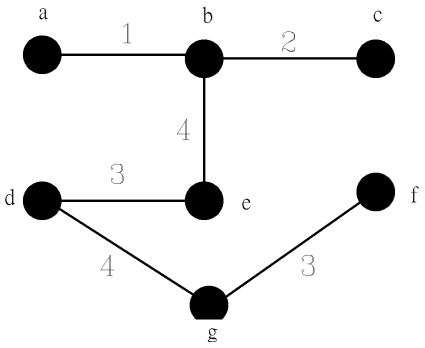


圖4.3 $w(T)=17$

在 Prim 演算法中, 因爲是由小樹長成大樹, 所以少了檢驗 T 是否有圈的步驟, 不過卻多了選邊的步驟。在 Kruskal 演算法中, 有檢驗 T 是否有圈的步驟, 但因爲邊已依序排好, 所以少了選邊的步驟。兩個演算法各有所長。選擇上, 如果邊數多, 就用 Prim 演算法; 如果邊數少, 就用 Kruskal 演算法。(Prim 演算法早在1930年便由 Jarnik 所發現, 當時是發表在捷克的雜誌上, 因此很少人注意到。)

五、最短路徑

「從台北到巴黎, 怎樣的行程最省時?」「從台南到花蓮, 怎樣的行程最省錢?」「從台大到中研院, 怎樣走最快?」等等, 都是最短路徑問題。以第一個問題爲例, 將地球上的城市當成點, 若兩個城市間有飛機直航, 就連一個邊, 並註明飛航時間, 於是就構成了一個賦權圖。假設 P 是一條從甲城市到乙城市的一條路徑, 定義 P 的長度爲 P 上的所有權值的和, 記作 $\ell(P)$。而從甲城市到乙城市的最短路徑長度記作 d(甲, 乙)。那麼我們就是要求 d(台北, 巴黎) 的值。

以圖5.1爲例, 若 $P=xaby$, 則 $\ell(P)=7$。若 $Q=xcdby$, 則 $\ell(Q)=6$。其實 Q 是一條最短 (x,y)-路徑, 所以 $d(x,y)=6$。當然, $xedby$, $xcby$ 也都是最短的 (x,y)-路徑。

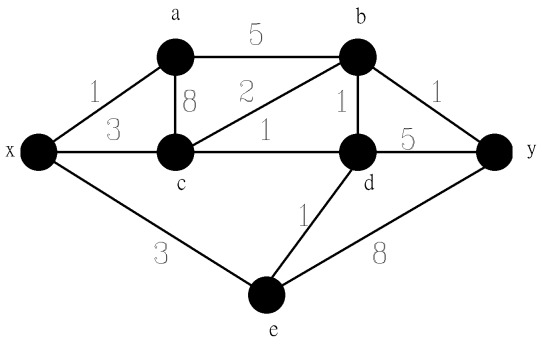


圖5.1

一般而言, 一個連通賦權圖的最短路徑問題有三類:

(1) 給定兩點 u,v, 求 $d(u,v)$。

(2) 給定一點 v, 求點 v 到其他點的最短路徑。

(3) 求任兩點之間的最短路徑。

實際上, 只要考慮問題 (2) 即可。因爲問題 (1) 是問題 (2) 的特例, 而問題 (3) 只要重覆利用解決問題 (2) 的方法即可。Dijkstra 在1959年解決了問題 (2)。主要想法如下。

假設 $S \subsetneq V$ 且 $x \in S$。令 $\overline{S} = V \backslash S$。如果 $P = x \cdots uv$ 是一條從 x 到 $\overline{S}$ 的最短路徑, 則 $u \in S$ 且 $x \cdots u$ 是一條從 x 到 u 的最短路徑。於是 $d(x,v) = d(x,u) + w(uv)$。因此

$$d(x, \overline{S}) = \min_{\substack{u \in S \\ v \in \overline{S}}} \{d(x,u) + w(uv)\} \quad (5.1)$$

Dijkstra演算法就是由式子 (5.1) 發展而得。首先, 令 $S_0 = \{x\}$。由式子(5.1) 得知,

$$\begin{aligned} d(x, \overline{S}_0) &= \min_{\substack{u \in S_0 \\ v \in \overline{S}_0}} \{d(x,u) + w(u,v)\} \\ &= \min_{v \in \overline{S}_0} \{w(xv)\} \\ &= \min_{xv \in E} \{w(xv)\} \end{aligned}$$

於是 $d(x, \overline{S}_0)$ 很快就可得到。假設 $u_1 \in \overline{S}_0$ 且 $d(x,u_1) = d(x, \overline{S}_0)$。令 $S_1 = \{x, u_1\}$, $P_1 = xu_1$。很明顯地, P_1 是一個最短的 (x,u_1)-路徑。重覆上述的步驟, 假設 $S_k = \{x, u_1, u_2, \ldots, u_k\}$ 且對應的最短路徑爲 $P_1, P_2, \ldots, P_k$。計算 $d(x, \overline{S}_k)$。從 $\overline{S}_k$ 中找到一點 u_{k+1}, 使得

$$d(x,u_{k+1}) = d(x, \overline{S}_k) = d(x,u_j) + w(u_j u_{k+1}),$$

其中 $u_j \in S_k$。而最短的 (x,u_{k+1})-路徑 P_{k+1} 則可由 P_j 再延伸邊 $u_j u_{k+1}$ 即可。重覆上述步驟, 直到找出所有以點 x 爲起點的最短路徑。

其實, Dijkstra 演算法的第一步是找所有以點 x 爲起點的路徑中長度最短的路徑, 假設找到 P_1。則 u_1 就是 P_1 的終點。令 $S_1 = \{x_1, u_1\}$。接下來找出以點 x 爲起點, 以 $\overline{S}_1$ 中的點爲終點的路徑中長度最短的路徑, 假設找到 P_2。則 u_2 就是 P_2 的終點。依此類推。由於每次找到的終點都不一樣, 所以就可求出點 x 到其他點的最短路徑。在找終點的過程中, 可以利用已找到的最短路徑, 以節省時間。另一方面, 它並不是固定某一點當終點再找最短路徑, 而是以路徑的長度來考量該選那一點當終點。這是所謂 greedy 演算法的主要精神所在。順便一提, Kruskal演算法和 Prim 演算法都是 greedy 演算法。

Dijkstra 演算法

輸入: 一個 n 個點的連通賦權圖 $G = (V,E)$及點 x。

步驟1: 令 $k = 0$ (用來表示已找到了 k 個點) $d(x,x) = 0$, $S_k = \{x\}$。(S_k 用來記錄已找到的點)。
如果 $xv \notin E$, 令 $d(x,v) = \infty$。

步驟2: 如果 $k = n-1$, 則輸出結果。
如果 $k < n-1$, 考慮所有不在 S_k 中的點 v。
令 $d(x,v)$
$= \min\{d(x,v), d(x,u_k) + w(u_k v)\}$ 找出點 u_{k+1} 使得
$d(x,u_{k+1}) = \min_{v \notin S_k} \{d(x,v)\}$
令 $S_{k+1} = S_k \cup \{u_{k+1}\}$, $k = k+1$。
重覆步驟2

輸出: 所有的 $d(x,v)$。

我們以圖5.2來說明 Dijkstra 演算法的執行過程。

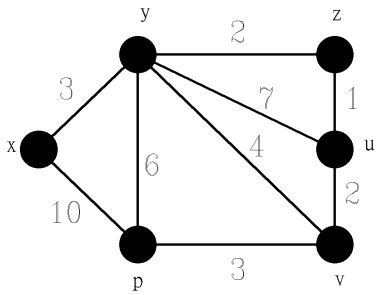


圖5.2

1. $S = \{x\}$。$d(x,x) = 0$, $d(x,y) = 3$, $d(x,z) = d(x,u) = d(x,v) = \infty$, $d(x,p) = 10$, 將點 y 放進 S。
2. $S = \{x,y\}$。

$$\begin{aligned} d(x,y) &= 3, \\ d(x,z) &= \min\{\infty, 3+2\} = 5 \\ d(x,u) &= \min\{\infty, 3+7\} = 10, \\ d(x,v) &= \min\{\infty, 3+4\} = 7, \\ d(x,p) &= \min\{10, 3+6\} = 9 \end{aligned}$$

將點z放進S。

3. $S = \{x,y,z\}$。$d(x,z) = 5$, $d(x,u) = \min\{10, 5+1\} = 6$, $d(x,v) = \min\{7, 5+\infty\} = 7$, $d(x,p) = \min\{9, 5+\infty\} = 9$, 將點 u 放進 S。
4. $S = \{x,y,z,u\}$。$d(x,u) = 6$, $d(x,v) = \min\{7, 6+2\} = 7$, $d(x,p) = \min\{9, 6+\infty\} = 9$, 將點 v 放進 S。
5. $S = \{x,y,z,u,v\}$。$d(x,v) = 7$, $d(x,p) = \min\{9, 7+3\} = 9$, 將點 p 放進 S。

最後結果爲 $d(x,x) = 0$, $d(x,y) = 3$, $d(x,z) = 5$, $d(x,u) = 6$, $d(x,v) = 7$, 最短路徑如圖5.3所示。

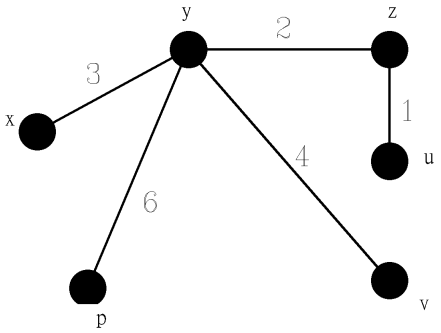


圖5.3

參考資料

1. J. A. Bondy and U.S.R Murty, "Graph Theory with Applications", 1976.
2. G. Brassard and P. Bratley, "Algorithmics", 1988.
3. G. Chartrand and C. R. Oellermann, "Applied and Algorithmic Graph Theorey", 1993.
4. M. Gondran and M. Minoux, "Graphs and Algorithms", 1984.
5. R. P. Grimaldi, "Discrete and Combinatorial Mathematics", 1994.
6. F. S. Roberts, "Applied Combinatorics", 1984.

—本文作者任教於靜宜大學應用數學系—