

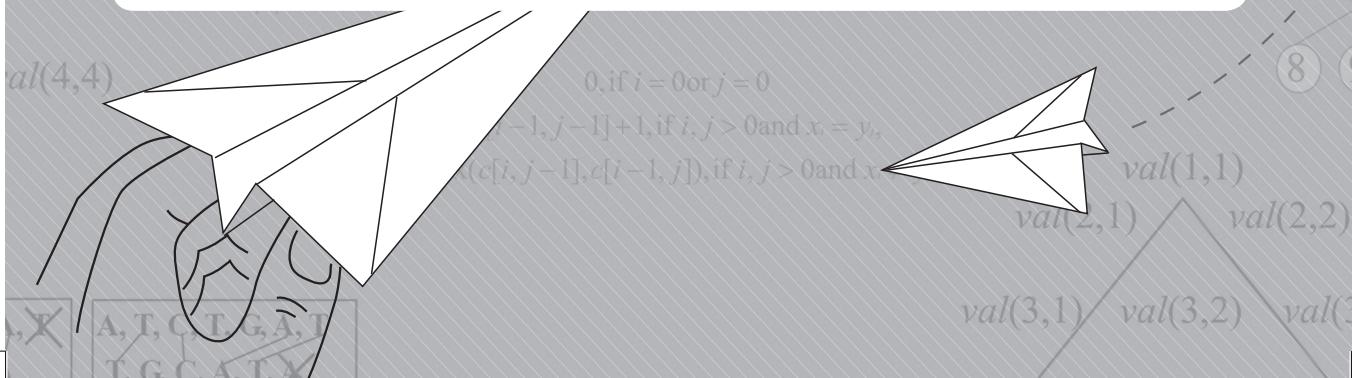
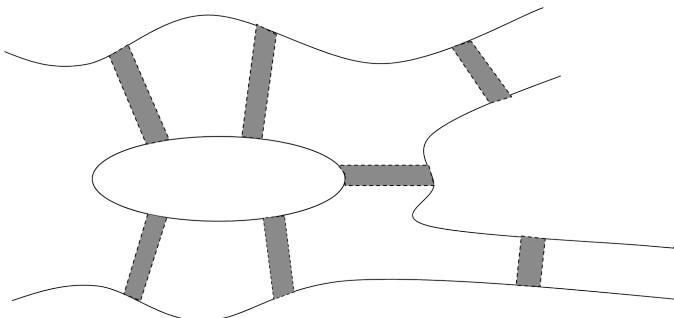


# 圖演算法

## 章節大綱

- 8.1 何謂圖？
- 8.2 連接組成
- 8.3 Dijkstra 最短路徑演算法
- 8.4 Bellman-Ford 最短路徑演算法
- 8.5 雙連接組成
- 8.6 圖演算法的技巧

在古代東普魯士(Eastern Prussia)柯尼斯堡(Koenigsberg)上有一條河流環繞 Keiphof 島，並將附近土地切割成四個區域。這些區域的居民藉著七座橋進行往來。當地的居民想自某一個區域剛好走過七座橋一次，且回到原出發地。你可不可以幫他們找出這條路徑？



## 8.1 何謂圖？

「什麼是圖 (graphs)？」

簡言之：「一個圖包含幾個點和幾條線，每一條線連接兩個點。」

圖演算法，就是將問題的輸入表達成一個圖，並在此圖上設計演算法，以找到問題的輸出(解)。

例如，當思考柯尼斯堡(Koenigsberg)問題時，把隔離的四個區域當作點(vertex)並將橋當作線(edge)，則柯尼斯堡的地圖可以用下面的圖來表示。

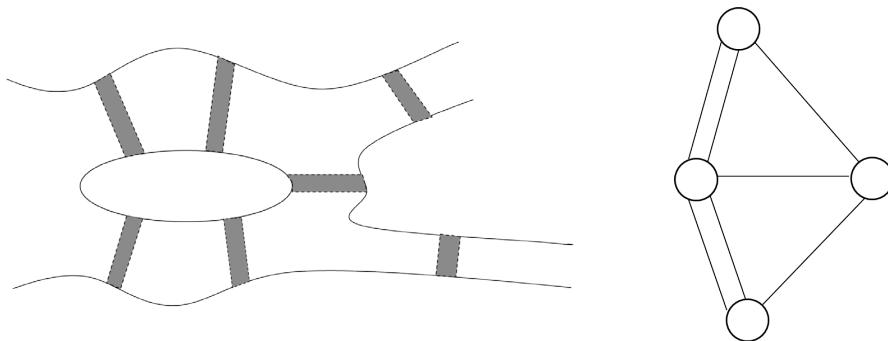


圖 8.1 用圖來表示柯尼斯堡問題

柯尼斯堡問題，就是在這個圖上找到一條路徑，使得剛好走過每一條線一次且回到原出發點。此問題，也被稱為尤拉路徑問題(Euler circuit)，有解的條件是：此圖是連接的(**connected**)，而且每一個點連接線的個數都是偶數。依據這個條件，我們可以判斷出「柯尼斯堡問題是無解的」，因為圖 8.1 雖然是連接的，但是每一個點連接線的個數分別是{3, 5, 3, 3}，並不全是偶數。

## 8.2 連接組成

第一個圖演算法的例子，是利用找出一個圖的連接組成(connected components)的技巧來解決問題。下表說明第一個例子。

表 8.1 同班同學

問題	有一個學校擁有 $N$ 位學生。有些學生彼此是同班同學。依據一個簡單法則 "我的同班同學的同班同學，也是我的同班同學"；也就是，如果 $A$ 和 $B$ 是同班而且 $B$ 和 $C$ 是同班，則 $A$ 和 $C$ 也是同班；換言之， $A$ 、 $B$ 、 $C$ 都是同一班的學生。當輸入所有學生之間的這種關係後，請計算全校共有幾個班級
輸入	輸入的第一行是整數 $N$ ，代表學校所有學生集合為 $\{1, 2, \dots, N\}$ 。第二行是整數 $M$ ，代表有 $M$ 對同班的關係。接下來， $M$ 行同班同學的關係列在其後 6 5 1 2 3 4 4 5 4 6 5 6
輸出	全校的班級個數 2

「如何利用圖演算法來解決同班同學的問題？」

「先試著將這個問題用一個圖來表示吧！」

「此問題是否提到，兩個不同個體之間的關係？」

「有啊！兩個學生是不是擁有同班同學的關係。」

「若利用一個點代表學生時，那麼一條線代表兩個學生之間的什麼關係？」

「如果兩位學生是同班，就用一條線連起來。」

利用這個方法，就可以將上述的輸入，表示成下面的圖。

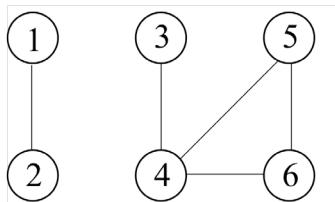


圖 8.2 用圖來表示同班同學的問題 (一個點代表一個學生，如果兩位學生是同班，就用一條線連起來)

「接下來，我們要在這個圖上找什麼？」「哪些學生是同一班的？及總共有多少班？」「從這個圖上，看得出總共有多少班嗎？」

「兩班，因為有兩團學生。」

「怎樣找到每一團的學生？」

「從一個點開始，尋找所有連接得到的點。」

「對一個圖進行搜尋的常見方法為何？」

「好像學過，是深度搜尋法嗎？」

---

直覺上，圖 8.2 中的每一團，就是此圖上的**連接組成**(connected component)－即此圖上最大可連接子圖。若能找到一個圖的連接組成的個數，就可以解決同班同學的問題。

根據以上的討論，我們可以**深度優先搜尋法**(depth-first search，簡稱**DFS**)展開每一個連接組成的搜尋。深度優先搜尋法的特點是：最晚被拜訪的點的鄰居，比其他較早被拜訪的點的鄰居，優先被搜尋的一種技巧。以圖 8.2 為例，可能的搜尋順序為 1 (起始點)，2 (1的鄰居)，發現第一個連接組成；3 (起始點)，4 (3的鄰居)，5 (4的鄰居)，6 (5的鄰居)，發現第二個連接組成 (圖 8.3)。詳細的演算法列於表 8.2。

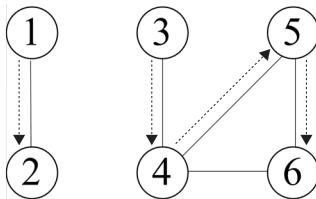


圖 8.3 利用深度優先搜尋法展開每一個連接組成的搜尋(虛線表示搜尋順序)

表 8.2 深度搜尋法

輸入	一個圖 $G=(V, E)$ 及搜尋起始點 $x$
輸出	自起始點 $x$ 開始，以深度搜尋法，拜訪所有 $V$ 中連接得到的點
步驟	<pre> Algorithm depth_first_search (x) /*令圖 G 有 n 個點，即   V   =n */ /*令矩陣 visited[1:n]全部為 false*/ { Step 1: 輸出 x 並且令 visited [x]:=true           /*拜訪點 x*/ Step 2: for 每一個 x 的鄰居 y do {     if visited [y]:=false     then depth_first_search (y)                  /*遞迴地拜訪點 y*/ } } </pre>

最後，利用深度搜尋法來逐一搜尋每一個連接組成。以下就是利用圖演算法，來解決同班同學問題的連接組成演算法。

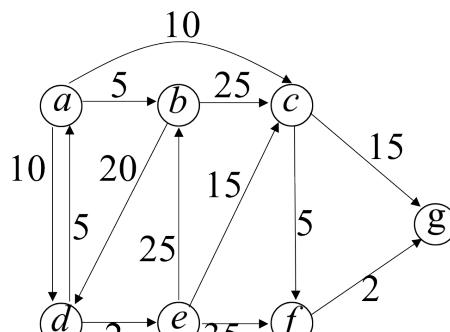
表 8.3 連接組成演算法

輸入	圖 $G=(V, E)$
輸出	圖 $G$ 的連接組成數目
步驟	<pre> Algorithm connected_component (G) /*令圖 G 有 n 個，即   V   =n */ /*令矩陣 visited[1:n]全部為 false*/ { Step 1: y:=0;                           /*連接組成的數目一開始為 0*/ Step 2: for i:=1 to n do         if visited [i]:=false then             {depth_first_search(i); /*遞迴地找出點 i 可連接到的一個連接組成*/             y:=y+1;                 /*增加一個連接組成*/             } Step 3: 輸出連接組成數目 y } </pre>

## 8.3 Dijkstra 最短路徑演算法

第二個例子是最短路徑(shortest paths)的問題。

表 8.4 最短路徑問題

問題	小王考慮從甲地開車前往其他城市，但不知道需要花費多少交通時間才可到某一個城市。現在有一張地圖記載了所有城市的位置，及兩兩城市之間所需要的交通時間請您利用此地圖，找到一條從甲城市出發，最終可以抵達(每一個)另一個城市的最短路徑(其所需要的交通時間總和為最短)
輸入	一個圖 $G=(V, E)$ 及起始點(出發地) $a$ 。在此圖中，用點代表城市，並且用一條線 $e=(v_1, v_2)$ 及線上的數字 $t[v_1, v_2]$ ，代表有一條路自城市 $v_1$ 連接另一個城市 $v_2$ 且開車需要 $t[v_1, v_2]$ 時間(此值非負數；即 $t[e] \geq 0$ )  <pre>graph LR; a((a)) -- 5 --&gt; b((b)); a((a)) -- 10 --&gt; d((d)); a((a)) -- 20 --&gt; e((e)); b((b)) -- 25 --&gt; c((c)); b((b)) -- 5 --&gt; d((d)); b((b)) -- 15 --&gt; e((e)); b((b)) -- 25 --&gt; f((f)); c((c)) -- 15 --&gt; g((g)); c((c)) -- 5 --&gt; f((f)); d((d)) -- 2 --&gt; e((e)); e((e)) -- 35 --&gt; f((f)); f((f)) -- 2 --&gt; g((g));</pre>
輸出	$a$ 到其他每一個點的最短路徑 $a \rightarrow a$ ，時間 0 $a \rightarrow b$ ，時間 5 $a \rightarrow c$ ，時間 10 $a \rightarrow d$ ，時間 10 $a \rightarrow d \rightarrow e$ ，時間 12 $a \rightarrow c \rightarrow f$ ，時間 15 $a \rightarrow c \rightarrow f \rightarrow g$ ，時間 17

若能將地圖上的交通資訊轉成一個圖，再找出此圖上，來源端到每一個點的最短路徑，就可以解決此問題。在此我們稱兩點的距離(distance)，為此兩點間的最短路徑上所有線上時間的總和。

### 8.3.1 直覺概念—Dijkstra 最短路徑演算法

將表 8.4 中的圖，自起始點到其他每一個點的最短路徑，依照距離長度由小到大列出如下：

$a \rightarrow a$ ，時間 0  
 $a \rightarrow b$ ，時間 5  
 $a \rightarrow c$ ，時間 10  
 $a \rightarrow d$ ，時間 10  
 $a \rightarrow d \rightarrow e$ ，時間 12  
 $a \rightarrow c \rightarrow f$ ，時間 15  
 $a \rightarrow c \rightarrow f \rightarrow g$ ，時間 17

「最短路徑之間有什麼關係？」

「最短路徑的前面部份，也是另一個最短路徑。例如， $a \rightarrow c \rightarrow f \rightarrow g$  之前面部份為  $a \rightarrow c \rightarrow f$  或  $a \rightarrow c \rightarrow f$  之前面部份為  $a \rightarrow c$  都是另一條最短路徑。」

「那一個距離較長？」

「包含較多點的路徑距離較長。」

「為什麼？」

「因為每兩點間的所需時間是正的。多走一段路需多一些時間。」

「如果由短到長找出所有的最短路徑，則這些最短路徑有何關係？」

「除了最後一點外，其他點的最短路徑，會比較早被找到。」

「如何選擇最短路徑的最後一點？」

「不知道耶！大概是找最短路徑吧。」

根據以上的討論，Dijkstra 最短路徑演算法，由近而遠地，找出起始點  $a$  到其他每一個點的最短路徑。此演算法一直紀錄目前找到最短路徑的點集合  $S$ ；並利用經過  $S$  的點向外跨一步，找到下一個距離起始點最近的最短路徑。

精準地說，Dijkstra 最短路徑演算法，嘗試找到自  $a$  點出發，並只經過  $S$  中的點，最後抵達  $S$  的外圍鄰居  $u$  的一條最短的路。注意  $u$  並不在  $S$  中，但是距離  $S$  其中的一點只有一步之遠。因此，為找到下一條最短路徑(及靠近  $a$  的點)，需計算  $a$  點(只經過  $S$  中的點)到每一個  $S$  的外圍鄰居  $u$  的路徑值，並將此值儲存於  $dist[u]$  中。此演算法會自所有可能的這種點  $u$  中，選擇一個擁有最小  $dist[u]$  值的點，而  $dist[u]$  即是其最短路徑。最後將點  $u$  納入集合  $S$  中。

Dijkstra 最短路徑演算法重複以上動作，直到所有點納入集合  $S$  中為止。

總結以上討論，下面的特性有助於了解 Dijkstra 最短路徑演算法：「令  $S$  集合包括目前已經找到距離最靠近  $a$  的點(含起始點  $a$ )。若下一個最靠近  $a$  的點為  $u$ ，自  $a$  走到  $u$  的最短路徑，所有中途經過的點都會在  $S$  中。」

直覺上，因為每一步都需要花額外的移動時間。所以越靠近起始點的點會被最先發現。如圖 8.4 所示，虛線內的點(代表  $S$ )是目前找到最靠近  $a$  的點。假設下一個最靠近  $a$  的點是自  $a$  走到  $u$  的路徑，則除了此路徑的最末點  $u$  外，不會經過虛線外的點。否則，假設此路徑中尚有另外一個點  $w$  落在  $S$  外，則自  $a$  走到  $w$  的路徑將會比自  $a$  走到  $u$  的路徑為短(因為自  $w$  走到  $u$  的路徑所需時間必為正數)(圖 8.4)。但是，如此又違反了「點  $u$  是下一個最靠近  $a$  的點」的假設。結論是此特性是正確的。

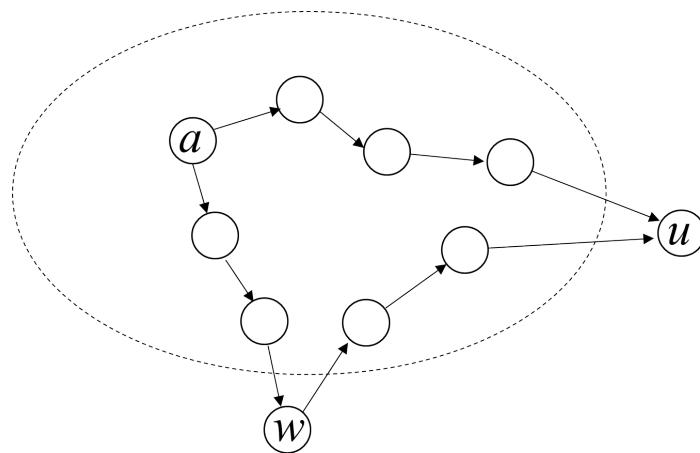


圖 8.4 下一個最靠近  $a$  的點，其路徑除了最後的一點外，不會經過虛線外的點

### 8.3.2 範例—Dijkstra 最短路徑演算法

以下將以表 8.4 中的圖為例子進行說明。

起初， $S$  只包含點  $a$ ，即  $S=\{a\}$ 。並計算  $a$  點直接走一步的最短距離。因為  $a$  點可連接到  $b$ (距離 5)，也可連接到  $c$ (距離 10)，或者連接到  $d$ (距離 10)，因此相對應的  $dist[]$  值可計算出，其餘的  $dist[]$  值暫時為  $\infty$ 。

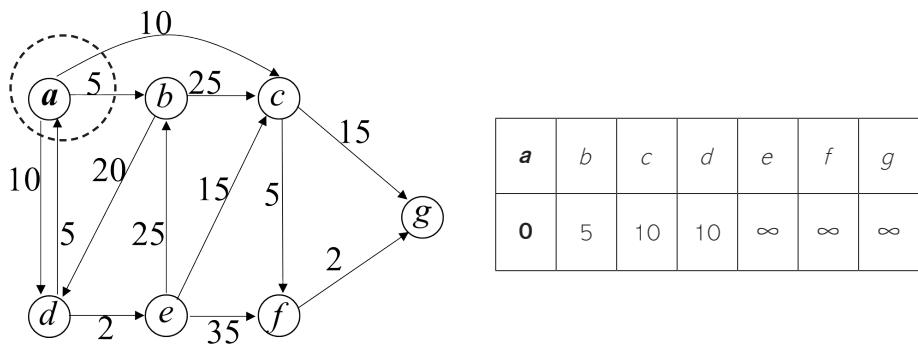


圖 8.5  $S=\{a\}$  及  $dist[]$

接下來，找出下一個最短路徑。自圖 8.5  $dist[]$  中，選擇不在  $S=\{a\}$  中且  $dist[]$  值最小的點。因為  $dist[b]=5$  最小，所以下一個最短路徑，是自點  $a$  到點  $b$  的路徑其最短距離為 5。

此刻將  $b$  加入  $S$  中，即  $S=\{a, b\}$ ；並重新計算， $a$  點到  $S$  的外圍鄰居的  $dist[]$  值。因為  $b$  是新加入  $S$  的點，故  $dist[]$  中有可能改變的是  $\{c, d\}$  (點  $b$  直接連接到的鄰居) (圖 8.6)。計算  $dist[b]+t[b, c]=5+25=30$ ，因為比目前的  $dist[c]=10$  還要大，故不更動  $dist[c]$ ；另外，計算  $dist[b]+t[b, d]=5+20=25$ ，因為也比目前的  $dist[c]=10$  還要大，故不更動  $dist[c]$  (圖 8.6)。

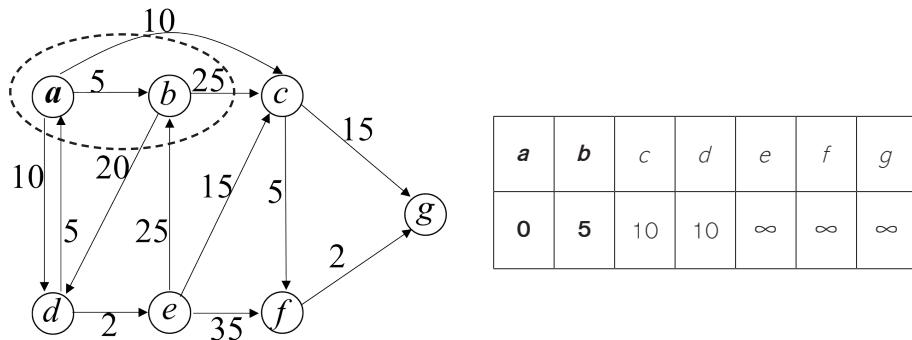
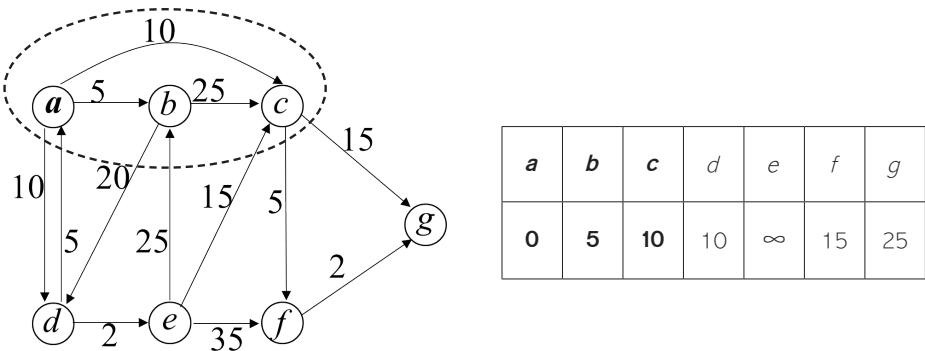


圖 8.6  $S=\{a, b\}$  及  $dist[]$

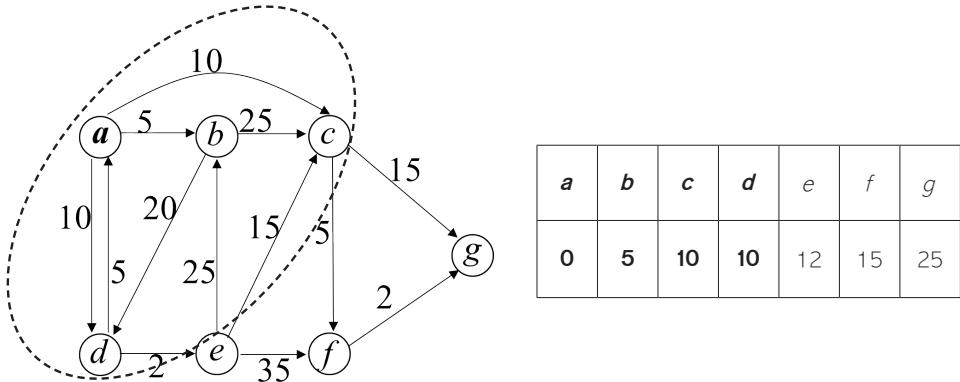
接下來，找出下一個最短路徑。自圖 8.6  $dist[]$  中，選擇不在  $S=\{a, b\}$  中且  $dist[]$  值最小的點(即點  $c$  或  $d$ ，因其  $dist[c]=dist[d]=10$ )。假設選擇  $c$ ，所以下一個最短路徑，是自點  $a$  到點  $c$  的路徑其最短距離為 10。

此刻將  $c$  加入  $S$  中，即  $S=\{a, b, c\}$  並重新計算， $a$  點到  $S$  的外圍鄰居  $\{f, g\}$ (點  $c$  直接連接到的鄰居)的最短路徑值  $dist[]$ 。計算  $dist[c]+t[c, f]=10+5=15$ ，因為比目前的  $dist[f]=\infty$  小，故更新  $dist[f]=15$ ；另外，計算  $dist[c]+t[c, g]=10+15=25$ ，因為也比目前的  $dist[g]=\infty$  小，故更動  $dist[g]=25$ (圖 8.7)。

圖 8.7  $S=\{a, b, c\}$  及  $dist[]$ 

接下來繼續完成後續的步驟。自圖 8.7  $dist[]$  中，選擇不在  $S=\{a, b, c\}$  中且  $dist[]$  值最小的點(即點  $d$ )。所以下一個最短路徑，是自點  $a$  到點  $d$  的路徑其最短距離為 10。

此刻將  $d$  加入  $S$  中，即  $S=\{a, b, c, d\}$  並重新計算， $a$  點到  $S$  的外圍鄰居 $\{e\}$ (點  $d$  直接連接到的鄰居)的最短路徑值  $dist[]$ 。計算  $dist[d]+t[d, e]=10+2=12$ ，因為比目前的  $dist[e]=\infty$  小，故更新  $dist[e]=12$  (圖 8.8)。

圖 8.8  $S=\{a, b, c, d\}$  及  $dist[]$

接下來，找尋下一個最短路徑。自圖 8.8  $dist[]$  中，選擇不在  $S=\{a, b, c\}$ ， $d$  中且  $dist[]$  值最小的點(即點  $e$ )。所以下一個最短路徑，是自點  $a$  到點  $e$  的路徑其最短距離為 12。

將  $e$  加入  $S$  中，即  $S=\{a, b, c, d, e\}$  並重新計算， $a$  點到  $S$  的外圍鄰居  $\{f\}$ (點  $e$  直接連接到的鄰居)的  $dist[]$ 。計算  $dist[e]+t[e, f]=12+35=47>dist[f]=15$ ，故不作更新(圖 8.9)。

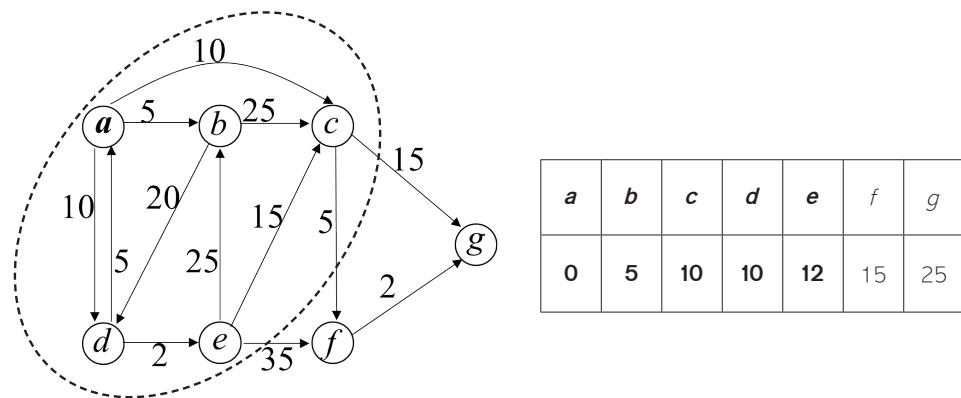
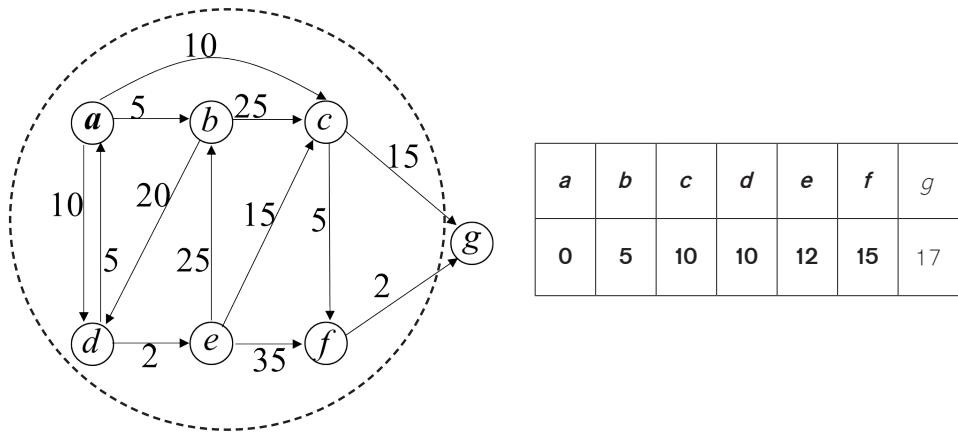


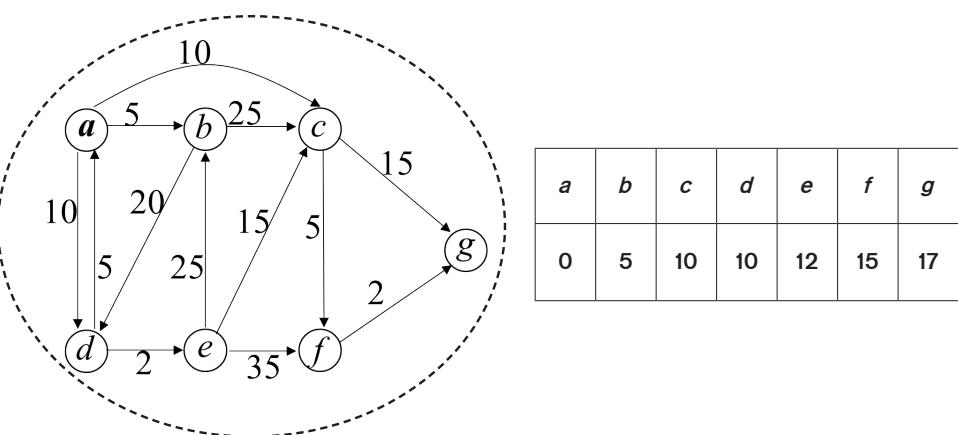
圖 8.9  $S=\{a, b, c, d, e\}$  及  $dist[]$

繼續找尋下一個最短路徑。自圖 8.9  $dist[]$  中，選擇不在  $S=\{a, b, c, d, e\}$  中且  $dist[]$  值最小的點(即點  $f$ )。所以下一個最短路徑，是自點  $a$  到點  $f$  的路徑其最短距離為 15。

將  $f$  加入  $S$  中，即  $S=\{a, b, c, d, e, f\}$  並重新計算， $a$  點到  $S$  的外圍鄰居  $\{g\}$ (點  $f$  直接連接到的鄰居)的  $dist[]$ 。因為  $dist[f]+t[f, g]=15+2=17<dist[g]=25$ ，故更新  $dist[g]=17$ (圖 8.10)。

圖 8.10  $S=\{a, b, c, d, e, f\}$  及  $dist[]$ 

最後，將  $g$  加入  $S$  中，即  $S=\{a, b, c, d, e, f, g\}$ ，並結束此演算法（圖 8.11）。

圖 8.11  $S=\{a, b, c, d, e, f, g\}$  及  $dist[]$ 

下頁的表 8.5 列出 Dijkstra 最短路徑演算法。

表 8.5 Dijkstra 最短路徑演算法

輸入	一個圖 $G=(V=\{1, 2, 3, \dots, n\}, E)$ ，起始點 $a$ ，每條線 $e=(i, j)$ 上的權重 $t[i, j]$ 。 (注意權重不可為負值，即 $t[i, j] \geq 0$ ；且 $i$ 不連接 $j$ 時，設 $t[i, j]=\infty$ 當 $1 \leq i, j \leq n$ )
輸出	$a$ 到其他每一個點的最短路徑長度；即 $dist[j]$ 儲存自 $a$ 到 $j$ 的最短路徑的長度
步驟	<pre> Algorithm Dijkstra_Shortest_Path (a, t[], dist[], n) {     /* 將起始點 a 選入 S 中，並設定其 dist[] 值 */     Step 1: S={a};         當 <math>1 \leq j \neq a \leq n</math> 時，令 <math>dist[j]=t[a, j]</math>;         當 <math>j=a</math> 時，令 <math>dist[j]=0</math>;      /* 決定自起始點 a 出發到達其他點的 <math>n-1</math> 條最短路徑 */     Step 2: 自 <math>i=2</math> 到 <math>n-1</math> 執行下列指令     {         /* 找到下一個最短路徑 */         自 <math>V-S</math> 中找出 <math>u</math> 使得其 <math>dist[u]</math> 值為最小;         將 <math>u</math> 加入 <math>S</math> 中;          /* 更新 dist[] */         針對 <math>V-S</math> 中的每一點 <math>w</math>，判斷如果 <math>dist[u]+t[u, w] &lt; dist[w]</math>，          則將 <math>dist[w]</math> 設為 <math>dist[u]+t[u, w]</math>;     } } </pre>

## 8.4 Bellman-Ford 最短路徑演算法

當輸入圖線上的值為負值時，Dijkstra 最短路徑演算法有可能找不到最短的路徑。如圖 8.12 所示，令  $a$  為起始點，起初  $S$  只包含點  $a$ (即  $S=\{a\}$ )。計算  $a$  點到  $S$  的外圍鄰居  $\{b, c\}$  的  $dist[]$  值(圖 8.12)。

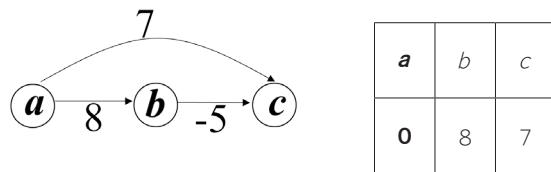


圖 8.12  $S=\{a\}$  及  $dist[]$

接著，選擇  $c$  加入  $S$ ，(即目前  $S=\{a, c\}$ )， $dist[]$  的值並未有改變；並且根據 Dijkstra 最短路徑演算法，當  $c$  加入  $S$  後， $dist[c]$  將不再被改變。此時  $dist[c]=7$  代表  $a$  點到  $c$  點的最短路徑為  $a \rightarrow c$ ，而此路徑值為 7；但是，在此圖中  $a$  點到  $c$  點的最短路徑應為  $a \rightarrow b \rightarrow c$ ，且其路徑值應為  $8-5=3$  才對。因此，當輸入圖線上的值為負值時，Dijkstra 最短路徑演算法有可能出錯。

此外，當輸入圖線上的值可為負時，若存在一個迴圈(cycle)其路徑值為負時，則可能存在負無限大值的最短路徑。如圖 8.13 中，路徑  $a \rightarrow b \rightarrow d \rightarrow a$  的路徑值為負，則自  $a$  點到  $c$  點的最短路徑可為  $a \rightarrow b \rightarrow d \rightarrow a \rightarrow b \rightarrow d \rightarrow a \rightarrow \dots \rightarrow b \rightarrow c$ 。當中間的負值迴圈不斷重覆時， $a$  點到  $c$  點的最短路徑值可為負無限大。

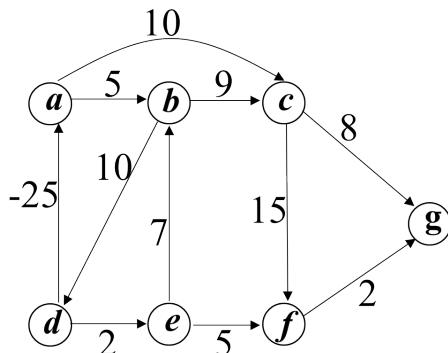


圖 8.13 回圈  $a \rightarrow b \rightarrow d \rightarrow a$  的路徑值為負的

當一個輸入圖沒有負值迴圈(但可能有負值的線)時，則圖中任兩點的最短路徑不會有迴圈；否則去掉此迴圈後，反而得到另一條更短的路徑。

#### 8.4.1 直覺概念—Bellman-Ford 最短路徑演算法

以下介紹 Bellman-Ford 最短路徑演算法，可以彌補 Dijkstra 最短路徑演算法不能處理負值線的缺憾。簡單地說，Bellman-Ford 最短路徑演算法的精神就是「動態規劃」。

首先，需要兩個符號：

$dist[u]$  代表自起始點  $a$  到點  $u$  的最短路徑值，

$dist_k[u]$  代表自起始點  $a$  到點  $u$  的最短路徑值，但是只經過最多  $k$  條線。

當輸入圖為  $G=(V=\{1, 2, 3, \dots, n\}, E)$ ，每條線  $e=(i, j)$  上的權重  $t[i, j]$  (注意此處權重可為負值，且當  $i$  不連接  $j$  時，設  $t[i, j]=\infty$ ， $1 \leq i, j \leq n$ )。此時， $dist_1[u]=t[a, u]$ ， $1 \leq u \leq n$ (因為初始時，只有一條線可經過)。其次，一條最短路徑最多只有  $n-1$  條線(否則會造成迴圈)， $dist_{n-1}[u]$  即等於自起始點  $a$  到點  $u$  的最短路徑值  $dist[u]$ 。

Bellman-Ford 最短路徑演算法，就是利用動態規劃的方式，計算出所有點的  $dist_{n-1}[u]$  值。如第三章所言，成功地使用動態規劃這個方法的關鍵，在於「找出大問題的解和小問題的解之間的關係」。

「當  $dist_k[u]$  是大問題的解，什麼是小問題的解？」

「會不會是  $dist_{k-1}[u]$ ？」

「如果是的話，他們之間的關係是什麼？」

「嗯！不知道。」

「 $dist_k[u]$  和  $dist_{k-1}[u]$  分別代表什麼？」

「 $dist_k[u]$  是自起始點  $a$  到點  $u$ ，只經過最多  $k$  條線的最短路徑值；而  $dist_{k-1}[u]$  是自起始點  $a$  到點  $u$ ，只經過最多  $k-1$  條線的最短路徑值。」

「他們之間的關係是什麼？他們之間有何不同？」

「 $dist_k[u]$  的路徑比  $dist_{k-1}[u]$  的路徑多一條線。」

「可以利用  $dist_{k-1}[u]$  找到  $dist_k[u]$  嗎？」

「好難想像呀！」

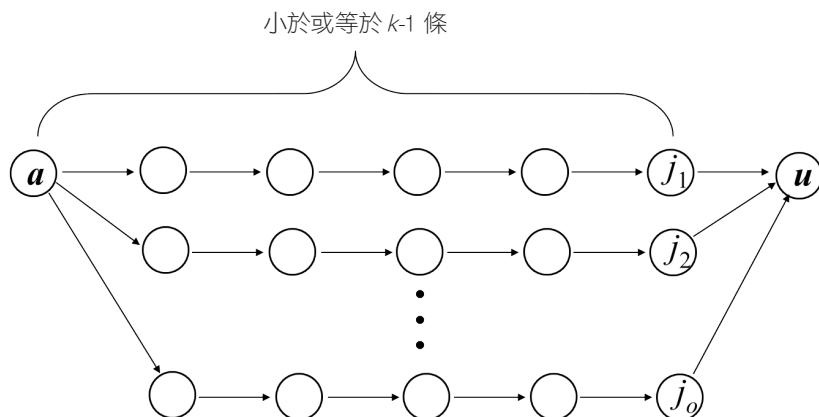
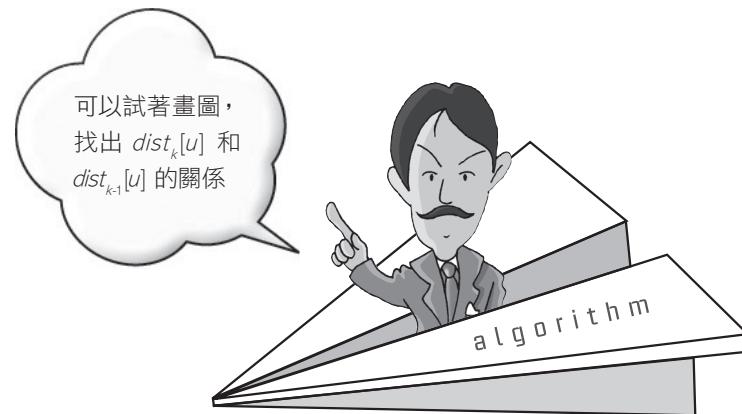


圖 8.14 找尋  $dist_k[u]$  和  $dist_{k-1}[u]$  的關係

「自  $a$  到  $u$  只經過最多  $k$  條線的最短路徑中，有沒有藏著一個類似問題的解？」

「此最短路徑中自  $a$  到  $u$  的前面一個點  $j$  的路徑，應是最多使用  $k-1$  條線的最短路徑。」

「這個『點  $j$ 』會是哪一個點？」

「好像很多點都有可能。只要自  $a$  出發經過最多經過  $k-1$  條線，最後連接點  $j$ ，點  $j$  又可連接點  $u$  即可。」

基於上述的討論，這條自  $a$  到  $u$  的最多使用  $k$  條線的最短路徑，是自  $a$  到  $j$  的最短路徑(最多經過  $k-1$  條線)，再銜接( $j, u$ )這條線所構成的。

為了找到  $a$  到  $u$  的最短路徑，所以需要自所有可能的  $j$  點中，選擇其中最小的一條路徑。此步驟可表示成  $\min_j \{dist_{k-1}[j] + t[j, u]\}$ 。但是，自  $a$  到  $u$  的最短路徑，也有可能用不到第  $k$  條線(即只使用最多  $k-1$  條線)，所以大問題的解和小問題的解之間的關係，可以表示成：

$$dist_k[u] = \min\{dist_{k-1}[u], \min_j \{dist_{k-1}[j] + t[j, u]\}\}$$

利用此關係，Bellman-Ford 最短路徑演算法就可被設計出來。

#### 8.4.2 範例—Bellman-Ford 最短路徑演算法

以下以圖 8.15 為例子，來說明 Bellman-Ford 最短路徑演算法：

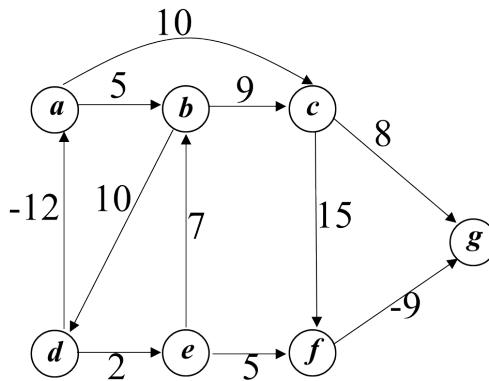


圖 8.15 Bellman-Ford 最短路徑演算法的範例

當  $k=1$  時，設定自起始點  $a$  僅利用一條線可抵達的點(即  $b, c$ )的路徑值；其他的點因利用一條線到不了，故設定其值為  $\infty$ 。

$$\begin{aligned} dist_1[a] &= t[a, a] = 0, \\ dist_1[b] &= t[a, b] = 5, \\ dist_1[c] &= t[a, c] = 10, \\ dist_1[d] &= t[a, d] = \infty, \\ dist_1[e] &= t[a, e] = \infty, \\ dist_1[f] &= t[a, f] = \infty, \\ dist_1[g] &= t[a, g] = \infty. \end{aligned}$$

當  $k=2$  時，利用  $dist_1[]$  來計算出  $dist_2[]$ 。

$$dist_2[a] = 0$$

$$dist_2[b] = \min\{dist_1[b], \min\{dist_1[a]+t[a,b], dist_1[e]+t[e,b]\}\} = \min\{5, \min\{0+5, \infty+7\}\} = 5$$

$$dist_2[c] = \min\{dist_1[c], \min\{dist_1[a]+t[a,c], dist_1[b]+t[b,c]\}\} = \min\{10, \min\{0+10, 5+9\}\} = 10$$

$$dist_2[d] = \min\{dist_1[d], \min\{dist_1[b]+t[b,d]\}\} = \min\{\infty, \min\{5+10\}\} = 15$$

$$dist_2[e] = \min\{dist_1[e], \min\{dist_1[d]+t[d,e]\}\} = \min\{\infty, \min\{\infty+2\}\} = \infty$$

$$dist_2[f] = \min\{dist_1[f], \min\{dist_1[c]+t[c,f], dist_1[e]+t[e,f]\}\} = \min\{\infty, \min\{10+15, \infty+5\}\} = 25$$

$$dist_2[g] = \min\{dist_1[g], \min\{dist_1[c]+t[c,g], dist_1[f]+t[f,g]\}\} = \min\{\infty, \min\{10+8, \infty-9\}\} = 18$$

當  $k=3$  時，利用  $dist_2[]$  來計算出  $dist_3[]$ 。

$$dist_3[a] = 0$$

$$dist_3[b] = \min\{dist_2[b], \min\{dist_2[a]+t[a,b], dist_2[e]+t[e,b]\}\} = \min\{5, \min\{0+5, \infty+7\}\} = 5$$

$$dist_3[c] = \min\{dist_2[c], \min\{dist_2[a]+t[a,c], dist_2[b]+t[b,c]\}\} = \min\{10, \min\{0+10, 5+9\}\} = 10$$

$$dist_3[d] = \min\{dist_2[d], \min\{dist_2[b]+t[b,d]\}\} = \min\{15, \min\{5+10\}\} = 15$$

$$dist_3[e] = \min\{dist_2[e], \min\{dist_2[d]+t[d,e]\}\} = \min\{\infty, \min\{15+2\}\} = 17$$

$$dist_3[f] = \min\{dist_2[f], \min\{dist_2[c]+t[c,f], dist_2[e]+t[e,f]\}\} = \min\{25, \min\{10+15, \infty+5\}\} = 25$$

$$dist_3[g] = \min\{dist_2[g], \min\{dist_2[c]+t[c,g], dist_2[f]+t[f,g]\}\} = \min\{18, \min\{10+8, 25-9\}\} = 16$$

當  $k=4$  時，利用  $dist_3[]$  來計算出  $dist_4[]$ 。

$$dist_4[a] = 0$$

$$dist_4[b] = \min\{dist_3[b], \min\{dist_3[a]+t[a,b], dist_3[e]+t[e,b]\}\} = \min\{5, \min\{0+5, 17+7\}\} = 5$$

$$dist_4[c] = \min\{dist_3[c], \min\{dist_3[a]+t[a,c], dist_3[b]+t[b,c]\}\} = \min\{10, \min\{0+10, 5+9\}\} = 10$$

$$dist_4[d] = \min\{dist_3[d], \min\{dist_3[b]+t[b,d]\}\} = \min\{15, \min\{5+10\}\} = 15$$

$$dist_4[e] = \min\{dist_3[e], \min\{dist_3[d]+t[d,e]\}\} = \min\{17, \min\{15+2\}\} = 17$$

$$dist_4[f] = \min\{dist_3[f], \min\{dist_3[c]+t[c,f], dist_3[e]+t[e,f]\}\} = \min\{25, \min\{10+15, 17+5\}\} = 22$$

$$dist_4[g] = \min\{dist_3[g], \min\{dist_3[c]+t[c,g], dist_3[f]+t[f,g]\}\} = \min\{16, \min\{10+8, 25-9\}\} = 16$$

當  $k=5$  時，利用  $dist_4[]$  來計算出  $dist_5[]$ 。

$$dist_5[a] = 0$$

$$dist_5[b] = \min\{dist_4[b], \min\{dist_4[a]+t[a,b], dist_4[e]+t[e,b]\}\} = \min\{5, \min\{0+5, 17+7\}\} = 5$$

$$dist_5[c] = \min\{dist_4[c], \min\{dist_4[a]+t[a,c], dist_4[b]+t[b,c]\}\} = \min\{10, \min\{0+10, 5+9\}\} = 10$$

$$dist_5[d] = \min\{dist_4[d], \min\{dist_4[b]+t[b,d]\}\} = \min\{15, \min\{5+10\}\} = 15$$

$$dist_5[e] = \min\{dist_4[e], \min\{dist_4[d]+t[d,e]\}\} = \min\{17, \min\{15+2\}\} = 17$$

$$dist_5[f] = \min\{dist_4[f], \min\{dist_4[c]+t[c,f], dist_4[e]+t[e,f]\}\} = \min\{22, \min\{10+15, 17+5\}\} = 22$$

$$dist_5[g] = \min\{dist_4[g], \min\{dist_4[c]+t[c,g], dist_4[f]+t[f,g]\}\} = \min\{16, \min\{10+8, 22-9\}\} = 13$$

當  $k=6$  時，利用  $dist_5[]$  來計算出  $dist_6[]$ 。

$$dist_6[a] = 0$$

$$dist_6[b] = \min\{dist_5[b], \min\{dist_5[a]+t[a,b], dist_5[e]+t[e,b]\}\} = \min\{5, \min\{0+5, 17+7\}\} = 5$$

$$dist_6[c] = \min\{dist_5[c], \min\{dist_5[a]+t[a,c], dist_5[b]+t[b,c]\}\} = \min\{10, \min\{0+10, 5+9\}\} = 10$$

$$dist_6[d] = \min\{dist_5[d], \min\{dist_5[b]+t[b,d]\}\} = \min\{15, \min\{5+10\}\} = 15$$

$$dist_6[e] = \min\{dist_5[e], \min\{dist_5[d]+t[d,e]\}\} = \min\{17, \min\{15+2\}\} = 17$$

$$dist_6[f] = \min\{dist_5[f], \min\{dist_5[c]+t[c,f], dist_5[e]+t[e,f]\}\} = \min\{22, \min\{10+15, 17+5\}\} = 22$$

$$dist_6[g] = \min\{dist_5[g], \min\{dist_5[c]+t[c,g], dist_5[f]+t[f,g]\}\} = \min\{13, \min\{10+8, 22-9\}\} = 13$$

最後，如表 8.6 所示， $dist_6[]$  中的每一個值，代表所有的最短路徑值。例如， $dist_6[g]$  代表點  $a$  到點  $g$  的最短路徑(即  $a \rightarrow b \rightarrow d \rightarrow e \rightarrow f \rightarrow g$ )值為 13。

表 8.6 Bellman-Ford 最短路徑演算法計算出的  $dist_k[]$  值

$k$	$dist_k[a..g]$						
	a	b	c	d	e	f	g
1	0	5	10	$\infty$	$\infty$	$\infty$	$\infty$
2	0	5	10	15	$\infty$	25	18
3	0	5	10	15	17	25	16
4	0	5	10	15	17	22	16
5	0	5	10	15	17	22	13
6	0	5	10	15	17	22	13

表 8.7 Bellman-Ford 最短路徑演算法

輸入	一個圖 $G=(V=\{1, 2, 3, \dots, n\}, E)$ ，起始點 $a$ ，每條線 $e=(i, j)$ 上的權重 $t[i, j]$ (注意此處權重可為負值，且當 $i$ 不連接 $j$ 時，設 $t[i, j]=\infty$ ， $1 \leq i, j \leq n$ )
輸出	$a$ 到其他每一個點的最短路徑長度；即 $dist[n-1, j]$ 儲存自 $a$ 到 $j$ 的最短路徑的長度
步驟	<pre> Algorithm Bellman_Ford_Shortest_Path (a, t[], dist[], n) {     /* 設定其 dist[1, j] 值 */     Step 1: 當 <math>1 \leq j \neq a \leq n</math> 時，令 <math>dist[1, j]=t[a, j]</math>；         當 <math>j=a</math> 時，令 <math>dist[1, j]=0</math>；      /* 決定自起始點 <math>a</math> 出發到達其他 <math>n-1</math> 點的最短路徑值 */     Step 2: 自 <math>i=2</math> 到 <math>n-1</math> 執行下列指令     {         2.1: 針對每一個 <math>a</math> 以外的點 <math>u</math> 執行下列指令         {             /* 注意此處的 <math>j</math> 需要考慮所有可直接連接到 <math>u</math> 的點 */             計算 <math>dist[i, u]=\min\{dist[i-1, u], \min_j\{dist[i-1, j]+t[j, u]\}\}</math>；         }     } } </pre>

## 8.5 雙連接組成

最後一個例子是雙連接組成(bi-connected component)問題。

表 8.8 雙連接組成問題

問題	在一個網路中，當一個節點的壞損，導致剩餘網路產生不連通的現象，此節點稱為關節點(articulation point)。小王想要自現存的網路中，找出一個儘可能大，但不存在關節點的子網路，以便進行網路攻擊測試 請您設計一個演算法，列出一個網路中的所有符合的子網路
輸入	一個圖 $G=(V, E)$ 代表現存的網路。每個點代表網路上的節點，而節點間的線，代表此節點間是有連接的 
輸出	所有不含關節點的最大子網路 {a, b} {b, c, f, g} {c, d} {e, f} {a, h, i, j, k}

以表 8.8 中輸入的網路為例，點 a 的壞損會導致剩餘網路不連通；明確地說，剩餘網路為兩個連接組成(connected component)，如圖 8.16(a)及(b)所示。

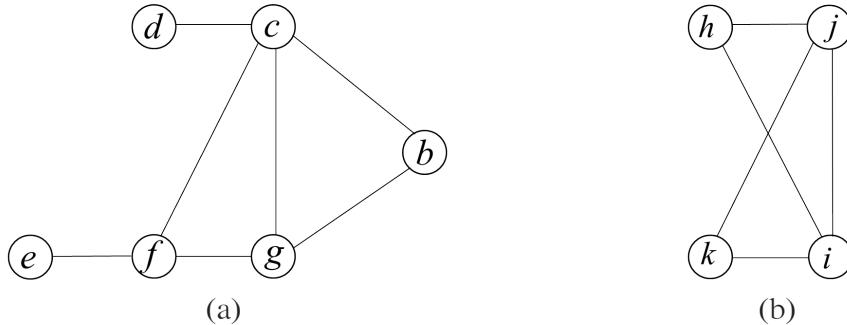


圖 8.16 當點  $a$  的壞損，導致剩餘網路不連通

因此，點  $a$  為一關節點，其他的關節點還包含點  $b, c, f$ 。一個連接圖 (connected graph)不含有關節點，則被稱為**雙連接圖**(bi-connected graph)。顯然地雙連接圖擁有較強的容錯能力。

圖 8.16(b)是一個雙連接圖，但是圖 8.16(a)不是一個雙連接圖。一個圖  $G$  的最大雙連接子圖(subgraph)  $H$ ，被稱為**雙連接組成**(bi-connected component)，代表在圖  $G$  中找不到比  $H$  更大的子圖，包含  $H$  而且是雙連接的。表 8.8 中的輸出 $\{a, b\}$ 、 $\{b, c, f, g\}$ 、 $\{c, d\}$ 、 $\{e, f\}$ 、 $\{a, h, i, j, k\}$ ，都是輸入圖的雙連接組成。雙連接組成問題，簡單地說，就是為一個圖找到其所有的雙連接組成。

### 8.5.1 雙連接組成的性質

想設計一個演算法，找到一個輸入圖的所有雙連接組成，似乎不容易。首先，將此範例的每個雙連接組成，用一個圈圈繪出，以方便找出其特點並進行討論。

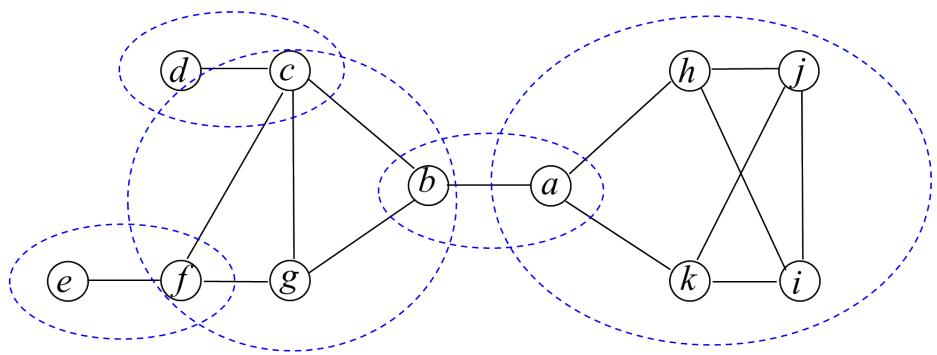


圖 8.17 每一個圈內的子圖代表一個雙連接組成

「雙連接組成之間有何關係？」

「只共用一個點。」

「這些共用的點有何性質？」

「都是關節點。」

「雙連接組成之間會共用一條線嗎？」

「好像不會。」

「有任何一條線沒有被任何一個雙連接組成所包含嗎？」

「好像也不會。」

「每一條線都被一個特定的雙連接組成所包含；因此，所有線被雙連接組成切割成不相交的集合？」

「對極了。」

「所有點被雙連接組成切割成不相交的集合？」

「不！因為有共用的現象。」

「這些共用的點有什麼特質？」

「好像割離出雙連接組成；好像是連接雙連接組成之間的橋或出入渡口。」

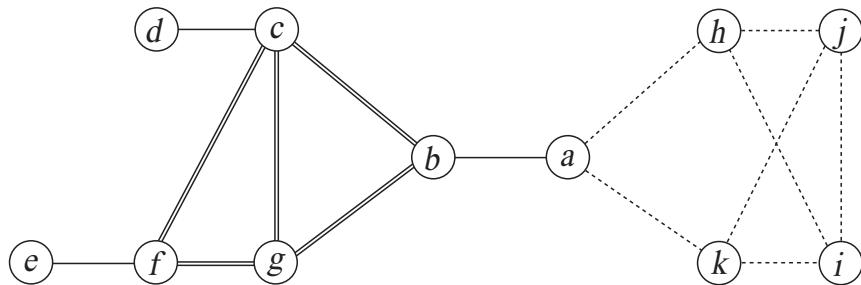


圖 8.18 每個雙連接組成內的線集合隱藏著一個神秘的關係

「哪些線會落在同一個雙連接組成？」

「不知道。」

「試著觀察存在同一個雙連接組成中的線之間的特性？」

「好像除了一條線之外，每個雙連接組成都是一團鄰近的線所組成。」

「這一團的線集合中藏著怎樣的關係？」

「不知道。」

「回到定義，雙連接組成的定義為何？」

「本身不存在關節點的最大連接子圖。」

「此網路不存在關節點，有何特性？」

「當此網路中任一個節點的壞損時，剩餘網路的任兩個節點還是連通的。也就是任一個節點的失效，無法破壞兩節點的連通。」

「因為原來網路是連通的，因此會有一條路徑連接任兩點。但是，為何當任一個節點壞損，導致這個路徑斷裂後，剩餘網路還是連通的？」

「會不會有另一條備份路徑(無共用中間點及線)，也連接此兩點？」

「這樣是不是暗示著，雙連接組成中的任兩點有何特性？」

「除非是一條線，雙連接組成中的任兩點，將存在兩條(無共用中間點及線)路徑，連接此兩點；即存在一條迴路(cycle)經過他們。」

---

以下將上述有關雙連接組成的特性列出：

1. 兩個雙連接組成最多共用一個節點，此節點為整個網路的關節點。
2. 每一條線被唯一的雙連接組成所包含。
3. 關節點將雙連接組成隔離並串接成整個網路。
4. 每個雙連接組成中的任兩條線，會被一個不重複節點的迴路所經過。

藉著上面的雙連接組成的特性，我們將有機會找出一個輸入圖的所有雙連接組成。

### 8.5.2 設計一個簡單的演算法

欲設計一個雙連接組成演算法，必須將所有圖中的線，分割成不同的雙連接組成。

目前已知，當兩條線被一個不重複節點的迴路經過，則此兩線屬於同一個雙連接組成。同理可得，所有在此迴路中的線，都屬於同一個雙連接組成，如圖 8.18 中的線 $[c, f]$ 、 $[f, g]$ 、 $[g, c]$ 屬於同一個雙連接組成。

更甚者，當有兩個迴路共用一些線，則此兩個迴路中的所有線，需與此共用線隸屬於同一個雙連接組成。如圖 8.19 中的迴路： $c-f-g-c$  和迴路  $b-c-g-b$  共用線 $[c, g]$ ，故 $[c, f]$ 、 $[f, g]$ 、 $[g, c]$ 、 $[b, c]$ 、 $[g, b]$ 應屬於同一個雙連接組成。又如迴路： $a-h-i-k-a$  和迴路  $i-k-j-i$  共用線 $[i, k]$ ，故 $[a, h]$ 、 $[h, i]$ 、 $[i, k]$ 、 $[k, a]$ 、 $[k, j]$ 、 $[j, i]$ 也應屬於同一個雙連接組成。

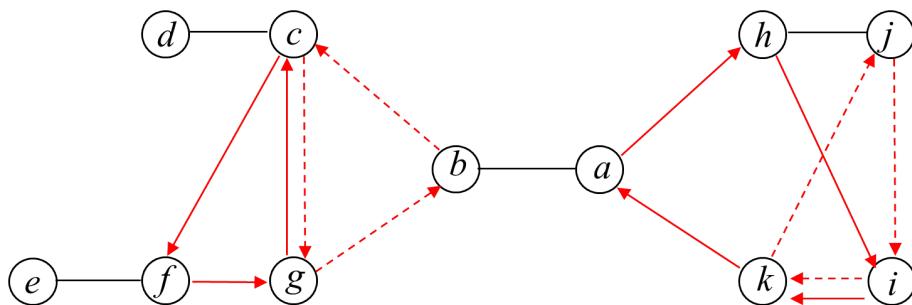
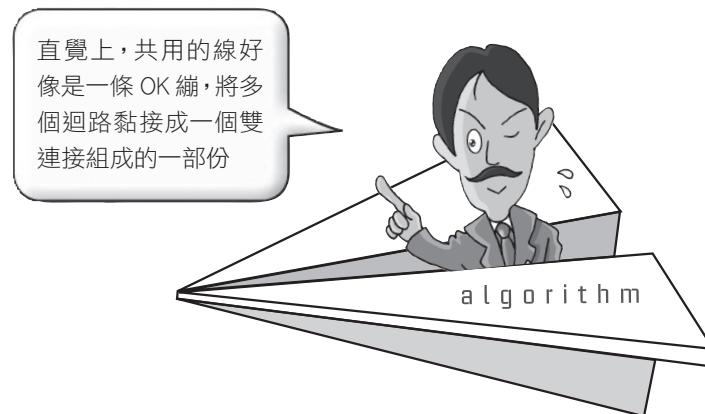


圖 8.19 共用線將多個迴路黏接成一個雙連接組成的一部份

為了找出所有的雙連接組成，目前最直接的方式是：將所有的迴路找到，並將一迴路和與其有共用線的迴路中的線，聯集成一個雙連接組成的一部份。但是，所有的迴路的數目有可能過大，導致此方法的效率不彰。

另外一種方法，是先找出所有關節點，因為關節點將整個網路隔離成多個雙連接組成。

欲找出所有關節點，可以試著將某一節點刪除，並測試剩下的圖是否依然連接，來判斷出所有關節點；但此法的效率顯然也不高。

### 8.5.3 深度優先搜尋生成樹的特性

另一種較有效率的方法是，利用深度優先搜尋法，找出所有的關節點及雙連接組成。

此法需要利用深度優先搜尋生成樹(depth-first-search spanning tree，簡稱 **DFS 生成樹**)的以下特性：

1. DFS 生成樹，再加上任一條不在此樹上的線，稱為退後線(back edge)，將形成唯一的迴路。
2. 任一條退後線，直接連接此樹中的祖先及其後代(如圖 8.20 虛線所示)。
3. 注意，圖中的所有線只有兩種：(1) DFS 生成樹上的線，和(2)連接 DFS 生成樹中祖先及其後代的退後線。我們所設計的演算法，需要將此兩種線，分類成不同的雙連接組成。

首先，討論因為加入一條退後線所牽連出的雙連接組成。注意，在圖 8.20 中，加入退後線[k, a]所形成的迴路  $a-h-i-j-k-a$ ，和加入退後線[k, i]所形成的迴路  $i-j-k-i$ ，和加入退後線[j, h]所形成的迴路  $h-i-j-h$ ，因為共用 DFS 生成樹上的線，導致此三條迴路隸屬於(黏接成)同一個雙連接組成。

相反地，若有 DFS 生成樹上的一條線，不落在任何這樣的迴路中，則它只好自己組成一個雙連接組成，如圖 8.20 中的[a, b]、[c, d]及[e, f]。以上的特性有助於找到每個雙連接組成中的線。

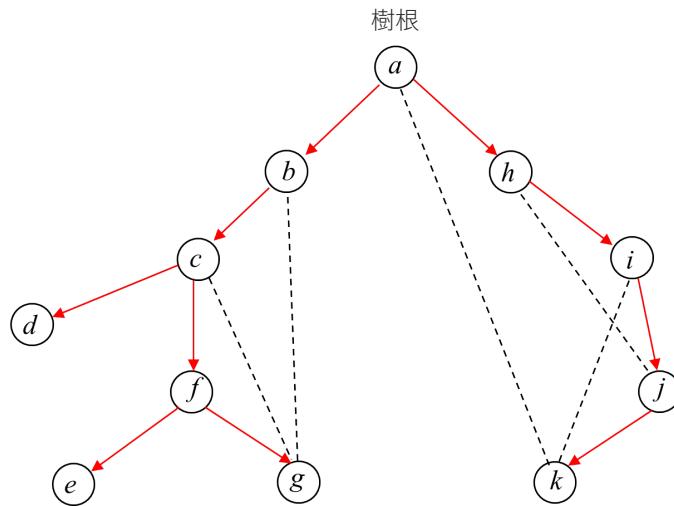


圖 8.20 深度優先搜尋的過程形成一個 DFS 生成樹(實線)

利用 DFS 生成樹，似乎可以有機會簡單地找出迴路，並且判斷兩迴路之間是否存有共用線，進而找到雙連接組成。

#### 8.5.4 利用深度優先搜尋法找出關節點

本節將利用 DFS 生成樹上的退後線所形成的迴路，來界定出雙連接組成及關節點的位置。注意關節點位居於雙連接組成的邊界，若能有效地辨認出圖中的關節點，將有助於設計出一個雙連接組成演算法。

首先，一個關節點的簡單性質說明如下。在此 DFS 生成樹上，一個樹根 (root)若是關節點，則最少有兩個以上的兒子(child) (因為除去此樹根後，此兩兒子所屬的圖將不會有路徑相連)；反之亦然。例如，圖 8.20 中的樹根  $a$  擁有兩個兒子，故為關節點。

當節點  $u$  不是樹根時，下列關節點的兩個特性是存在的：

- (1) 「如果節點  $u$  存在某一個兒子  $w$ ，利用  $w$  及其後代及一條退後線，無法連接到  $u$  的祖先，則節點  $u$  是一個關節點。」

原因是在一個連接圖中，除了 DFS 生成樹上的線，剩下的就是退後線了。刪除節點  $u$  後， $w$  和  $u$  的祖先將無法透過退後線連接。因此，節點  $u$  是一個關節點。

例如，圖 8.20 中的節點  $b$  有一個兒子  $c$ ，利用  $c$  及其後代及一條退後線，是無法連接到  $b$  的祖先(最多只連到  $b$ )。若除去  $b$ ，則  $c$  將無法連接到  $b$  的祖先。因此，節點  $b$  是關節點。請注意，節點  $c(f)$  也是關節點，因為此節點都存在一個兒子  $d(e)$ ，利用此兒子及其後代及一條退後線，不可以連接到  $c(f)$  的祖先。

- (2) 相反地，「如果自節點  $u$  的每一個兒子  $w$ ，利用  $w$  及其後代及一條退後線，都可以連接到  $u$  的祖先，則節點  $u$  不是一個關節點」。

原因是刪除節點  $u$  後，其所有兒子及其後代和樹上的其他節點，依然連接著。因此，節點  $u$  不是一個關節點。

例如，圖 8.20 中的節點  $i$ ，只有一個兒子  $j$ ，利用  $j$  及其後代及一條退後線，是可以連接到  $i$  的祖先  $h$  或  $a$ ，故節點  $i$  不是關節點。

若想利用上述的特性，來判斷某節點是否為關節點，需要知道利用此節點及其後代及一條退後線，可以連接到的祖先為哪一個節點。因此，我們將為每個點進行編號以方便辨識。

首先，我們依照深度優先搜尋拜訪的順序，將節點由小至大來進行編號(如圖 8.21)，此編號被稱為**深度優先編號**(depth first number, DFN)。如  $DFN(a)=1$ ,  $DFN(h)=8$ , 而  $DFN(i)=9$ 。在深度優先搜尋拜訪的過程中，因為祖先總是比後代先被拜訪到，祖先的 DFN 編號永遠比其後代為小。

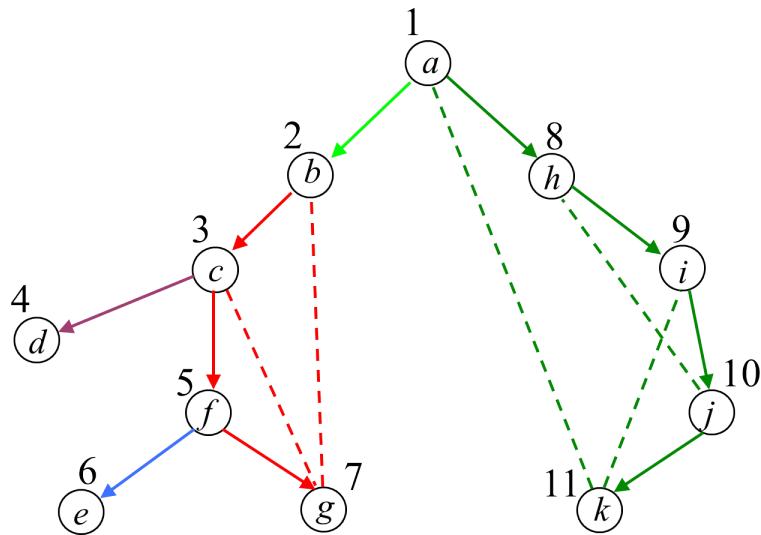


圖 8.21 在深度優先編號中，祖先的編號永遠比其後代為小

利用此特性，當執行深度優先搜尋法，拜訪到退後線時，紀錄此線連接到最老祖先的  $DFN$  編號，並且將之回傳給需要的節點，將可以有效率地藉著上述的兩個特性來判斷關節點。明確的作法為，針對每一個節點  $u$ ，我們定義一個函數

$$L(u) = \min \{ DFN(u), \min \{ L(w) \mid w \text{ 是 } u \text{ 的兒子} \}, \min \{ DFN(x) \mid (u, x) \text{ 是一條退後線} \} \}$$

$L(u)$  代表利用節點  $u$  及其後代所形成的路徑，和最多一條退後線可以連接到的最老祖先之  $DFN$  編號。

如先前討論，如果節點  $u$  (非樹根)，則

節點  $u$  是一個關節點

若且唯若

節點  $u$  存在某一個兒子  $w$ ，利用  $w$  及其後代及一條退後線，無法連接到  $u$  的祖先。

上述的條件若是利用  $DFN$  和函數  $L$  表示，則為

節點  $u$  是一個關節點  
若且唯若  
節點  $u$  有一個兒子  $w$  其  $L(w) \geq DFN(u)$ 。

例如，在圖 8.22 中，節點  $b$  有個兒子  $c$  其  $L(c)=2 \geq DFN(b)=2$ ，故節點  $b$  是一個關節點。又例如，節點  $c$  有個兒子  $d$  其  $L(d)=4 \geq DFN(c)=3$ ，故節點  $c$  是一個關節點。反之，節點  $i$  只有一個兒子  $j$  其  $L(j)=1 < DFN(i)=9$ ，故節點  $i$  不是一個關節點。計算函數  $L$  值可在作深度優先搜尋拜訪時，利用後序法(postorder)計算得出。有關計算  $DFN$  和函數  $L$  值的演算法，請參閱表 8.9。

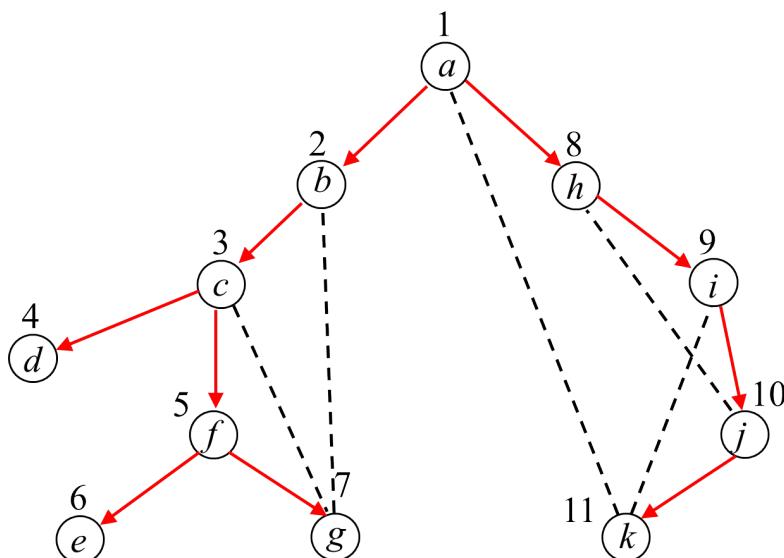


圖 8.22 計算每個節點的函數  $L$  值： $L(a)=1, L(b)=2, L(c)=2, L(d)=4, L(e)=6, L(f)=2, L(g)=2, L(h)=1, L(i)=1, L(j)=1, L(k)=1$

表 8.9 計算 DFN 和函數 L 值演算法

輸入	一個圖 $G=(V=\{1, 2, 3, \dots, n\}, E)$ ，起始節點 $u$ ，及節點 $v$ 是在深度優先搜尋生成樹中 $u$ 的父節點(如果存在的話)。設初始值 $DFN[] = 0$ 且 $num=1$
輸出	每個節點在深度優先搜尋生成樹中的 $DFN$ 和函數 $L$ 值
步驟	<pre> Algorithm Compute_DFN_L (u, v) {     /* 設定其起始節點的值 */     Step 1 : <math>DFN[u] = num</math>; <math>L[u] = num</math>; <math>num = num + 1</math>     Step 2 : 對每一個節點 <math>u</math> 的相鄰節點 <math>w</math>，執行下列指令     {         2.1 : if <math>DFN[w] = 0</math> then          /* 節點 <math>w</math> 未被拜訪過 */             {                 /* 繼續執行深度優先搜尋 */                 呼叫 Compute_DFN_L(w, u);                 <math>L[u] = \min(L[u], L[w])</math>;             }         else                                /* 節點 <math>w</math> 被拜訪過，則 <math>(u, w)</math> 是   一條退後線，執行 <math>L[u]</math> 計算 */             {                 if <math>w \neq v</math> then <math>L[u] = \min(L[u], DFN[w])</math>;             }     } } </pre>

計算  $DFN$  和函數  $L$  值演算法，顯然可以在  $O(n+e)$  的時間內完成(此處的  $n$  為圖的節點個數，而  $e$  為線的個數)，因此所有關節點也可在  $O(n+e)$  內找到。

### 8.5.5 利用深度優先搜尋法找出所有雙連接組成

至此，我們知道利用函數  $L()$  及  $DFN()$  可以辨認出關節點。接下來，我們將再找出所有雙連接組成。

一個雙連接組成內的線有兩個特色：(1)被關節點所分割，(2)在深度優先搜尋的過程中，除非被其他的雙連接組成的線所佔據，同一個雙連接組成內的線是連續地出現的。

例如，在圖 8.23 中，依照深度優先搜尋法對圖中的線之拜訪順序進行編號，則編號{9、10、11、12、13、14、15}等連續七條線是屬於同一個雙連接組成；但是，編號{2、4、6、7、8}不連續，卻是屬於同一個雙連接組成，原因是其中夾著兩個雙連接組成，即{3}及{5}。若是排除{3}及{5}不計算在內，則編號{2、4、6、7、8}的線可視為連續的。

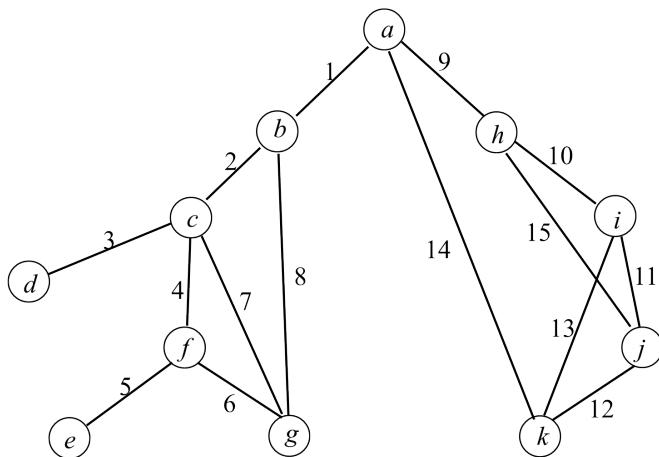


圖 8.23 依照深度優先搜尋法對所有的線進行編號

這個特性其實是：因為若將每一個雙連接組成視為一個節點，則整個連接圖就是一棵樹(tree) (圖 8.24)。

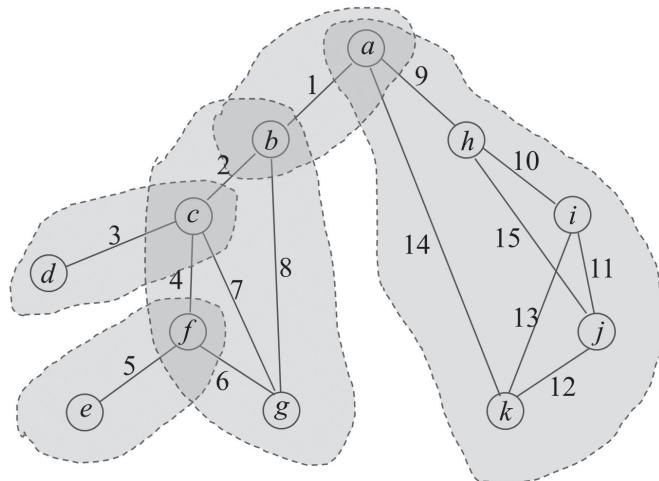
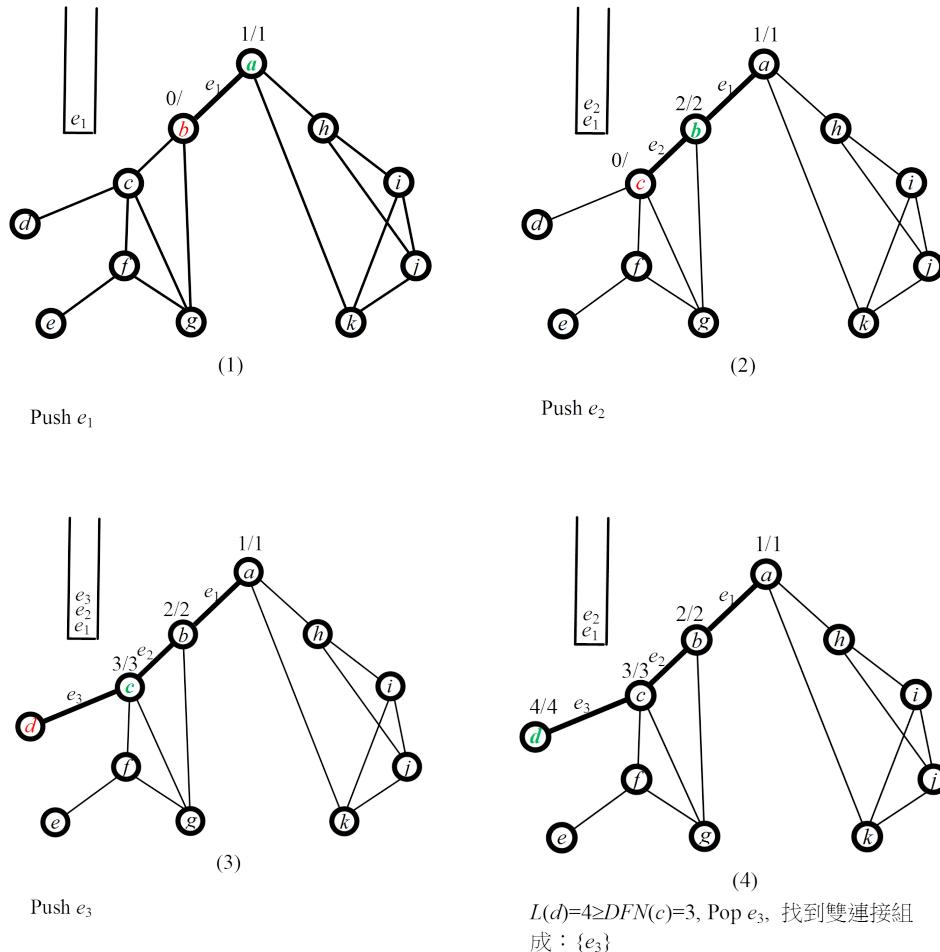
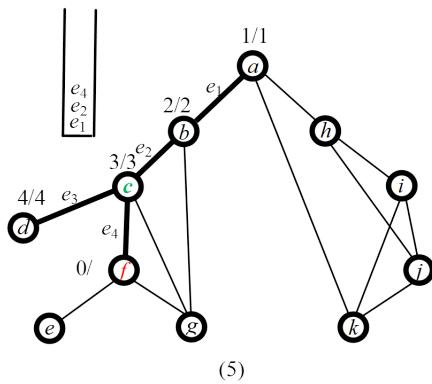


圖 8.24 每個雙連接組成可視為樹中的一個節點

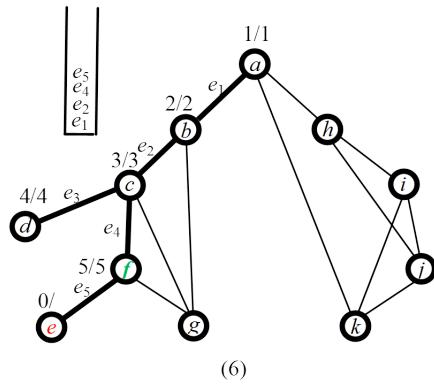
依照上述的特性，我們可以設計出一個有效率的雙連接組成演算法，其概念如下：

1. 利用深度優先搜尋法，依序拜訪輸入圖。
2. 在拜訪的過程中，將第一次遇到的線存到(push)一個堆疊(stack)中。
3. 當發現一個節點  $u$  是關節點時(即節點  $u$  的兒子  $w$  其  $L(w) \geq DFN(u)$ )，將堆疊中的線連續取出(pop)，直到取出的線為( $u, w$ )為止。圖 8.25 是一個範例。

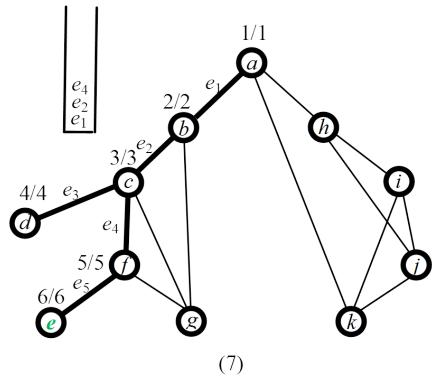




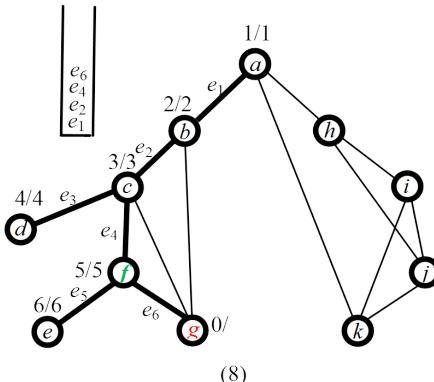
Push  $e_4$



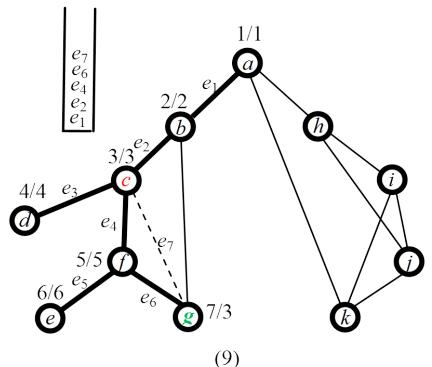
Push  $e_5$



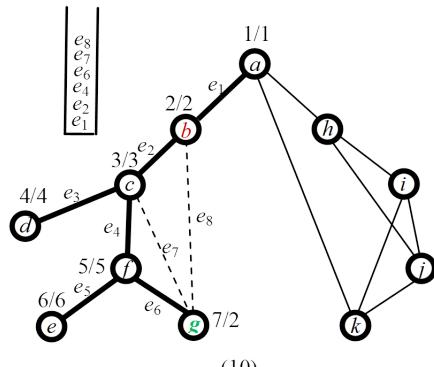
$L(e)=6 \geq DFN(f)=5$ , Pop  $e_5$ , 找到雙連接組成 :  $\{e_5\}$



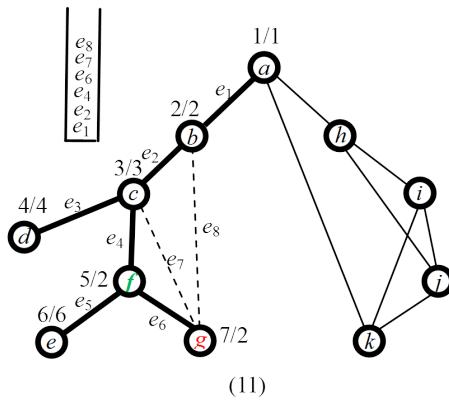
Push  $e_6$



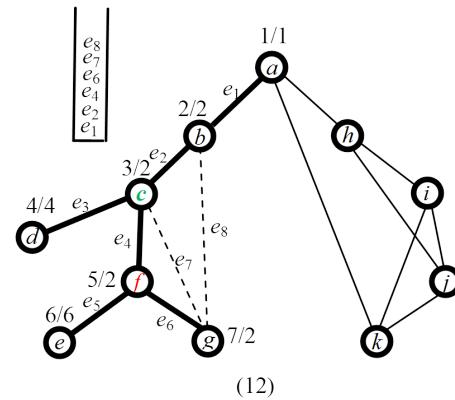
Push  $e_7$ ,  $L(g)=\min\{L(g)=7, DFN[c]=3\}=3$



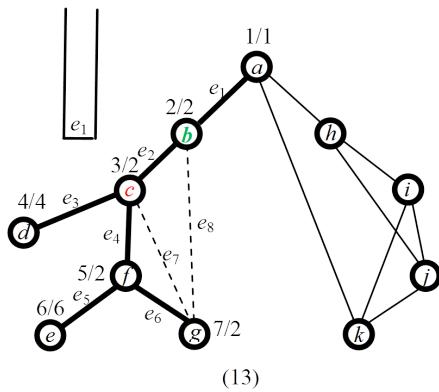
Push  $e_8$ ,  $L(g)=\min\{L(g)=3, DFN[b]=2\}=2$



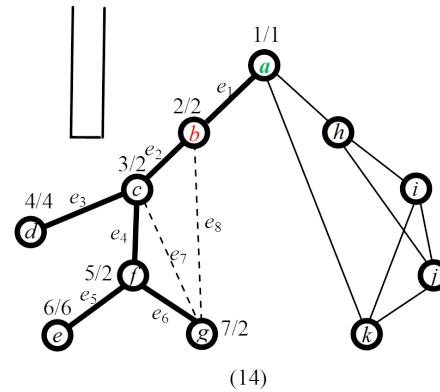
$L(f)=\min\{L(f)=5, L(g)=2\}=2$



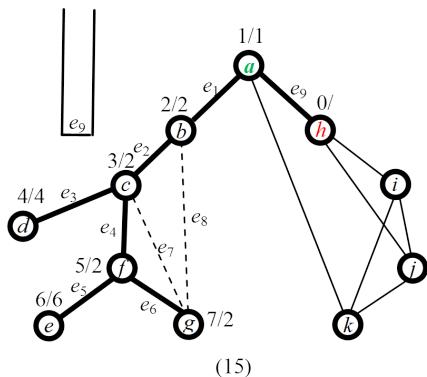
$L(c)=\min\{L(c)=3, L(f)=2\}=2$



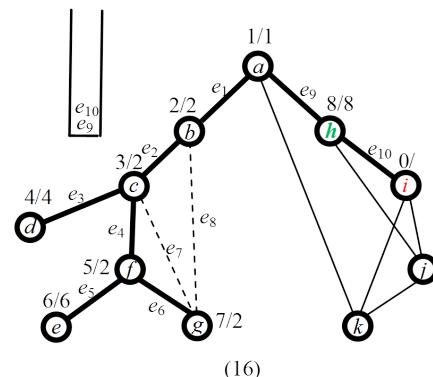
$L(c)=2 \geq DFN(b)=2$ , 連續 Pop  $e_8, e_7, e_6, e_4, e_2$ ,  
找到雙連接組成 :  $\{e_2, e_4, e_6, e_7, e_8\}$



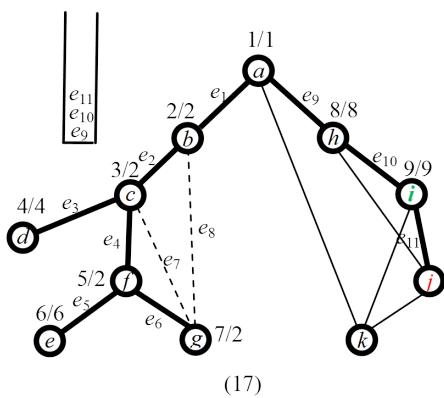
$L(b)=2 \geq DFN(a)=1$ , Pop  $e_1$ ,  
找到雙連接組成 :  $\{e_1\}$



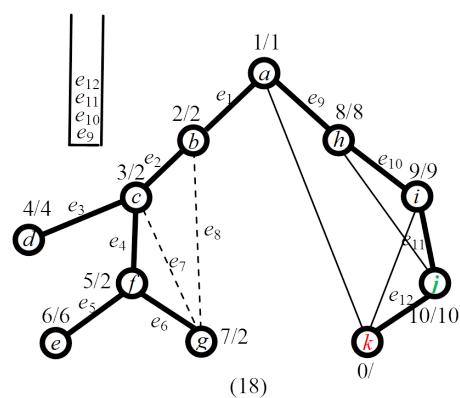
Push  $e_9$



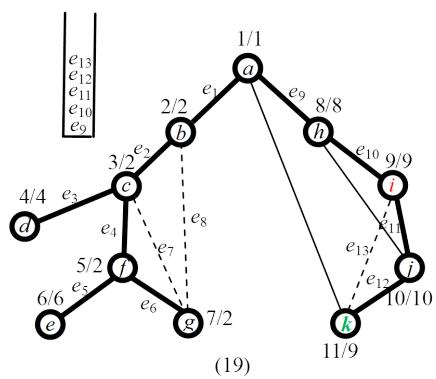
Push  $e_{10}$



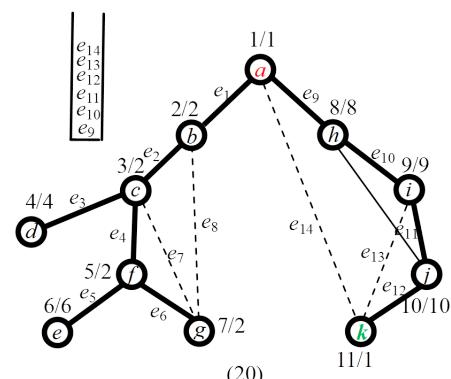
Push  $e_{11}$



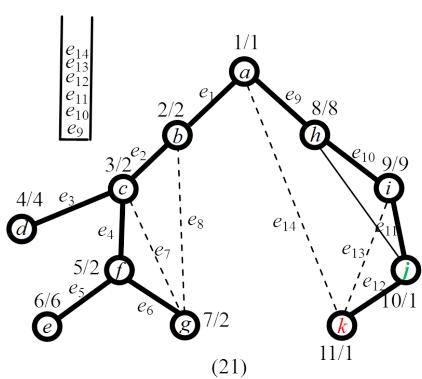
Push  $e_{12}$



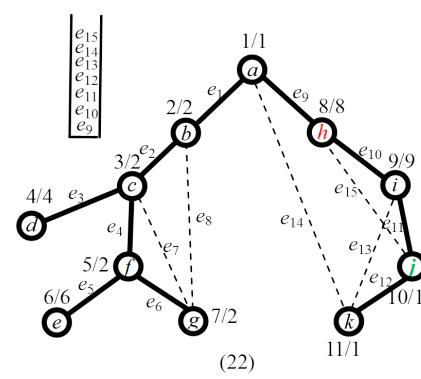
Push  $e_{13}$ ,  $L(k)=\min\{L(k)=11, DFN[i]=9\}=9$



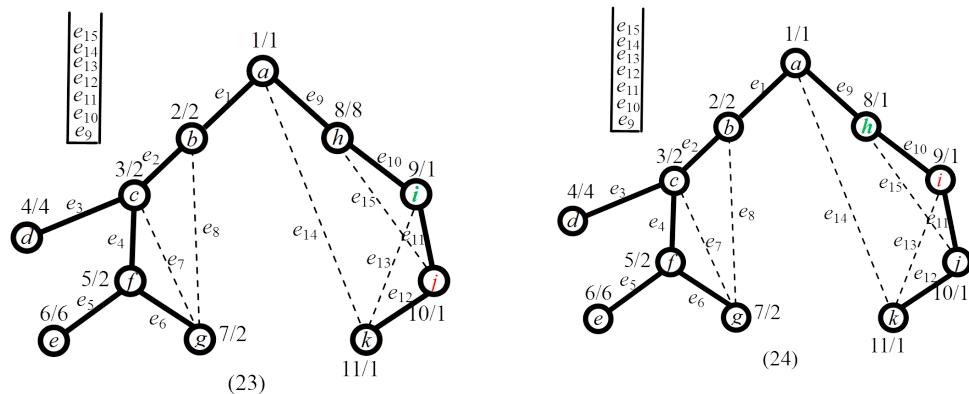
Push  $e_{14}$ ,  $L(k)=\min\{L(k)=9, DFN[a]=1\}=1$



$L(j)=\min\{L(j)=10, L(k)=1\}=1$



Push  $e_{15}$ ,  $L(j)=\min\{L(j)=1, DFN[h]=8\}=1$



$$L(i) = \min\{L(i)=9, L(j)=1\} = 1$$

$$L(h) = \min\{L(h)=8, L(i)=1\} = 1$$

圖 8.25 雙連接組成演算法之範例。每個節點  $u$  旁的兩個整數為  $DFN(u)/L(u)$

根據以上的討論，我們可以修改計算  $DFN$  和函數  $L$  值演算法(表 8.9)後，得到雙連接組成演算法(表 8.10)。注意，此雙連接組成演算法的時間複雜度，仍為  $O(n+c)$  (此處的  $n$  為圖的節點個數，而  $c$  為線的個數)。

表 8.10 雙連接組成演算法

輸入	一個圖 $G=(V=\{1, 2, 3, \dots, n\}, E)$ ，起始節點 $u$ ，及節點 $v$ 為節點 $u$ 在深度優先搜尋生成樹中的父節點(如果存在的話)。設初始值 $DFN[] = 0$ 且 $num=1$
輸出	所有的雙連接組成
步驟	<pre>Algorithm bi-connected_component (u, v) {     /*設定其起始節點的值*/     Step 1: DFN[u]=num; L[u]=num; num =num+1     Step 2: 針對每一個節點 u 的相鄰節點 w，執行下列指令     {         2.1: if w≠v 而且 DFN[w]&lt;DFN(u) then 將線(u, w) push             到堆疊 S 中         2.2: if DFN[w]=0 then 呼叫 bi-connected_component (w, u);             /*節點 w 未被拜訪過*/         {             L[u]=min (L[u], L[w]);             if L[w]≥DFN[u] then 輸出 "以下是一個新的雙連接組成";                 /*發現節點 u 是一個關節點輸出一個雙連接組成*/         }         重複自堆疊 S 中 pop 一條線(並將此線輸出)，直到此線為(u, w)或         (w, u)為止;     } } else           /*節點 w 被拜訪過*/ {     if w≠v then L[u]=min (L[u], DFN[w]); }</pre>

## 8.6 圖演算法的技巧

「怎樣的問題可以利用圖來解決呢？」

「不知道。」

「甚麼是圖？」

「幾個節點利用幾條線連接起來。」

「如果每個節點代表一個體，則連接兩節點的一條線，代表兩個體之間的什麼？」

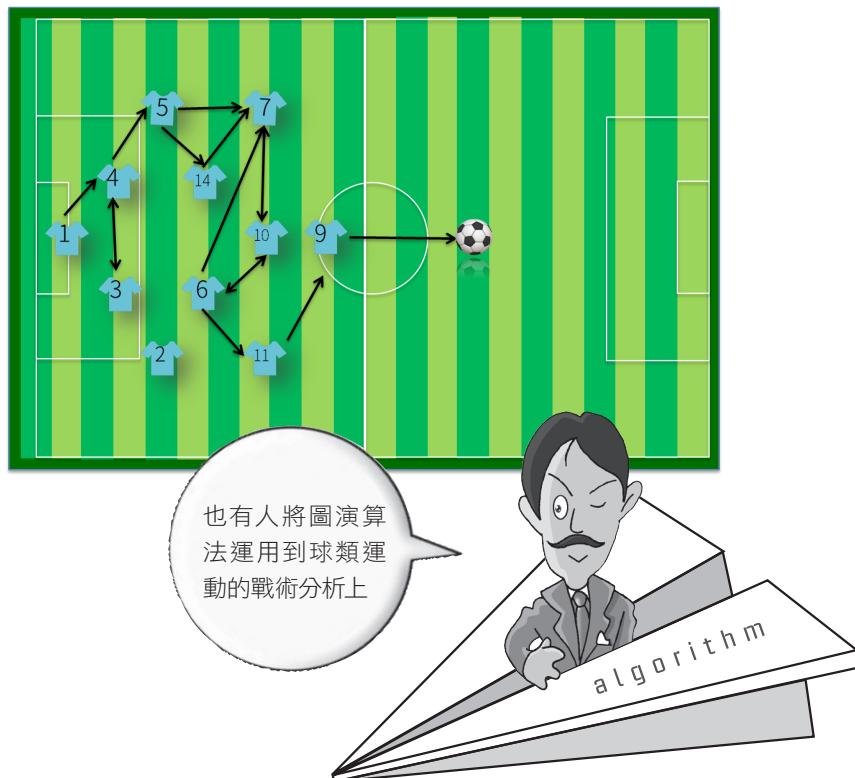
「會不會是此兩個體之間的一種關係？」

「因此，怎樣的問題可以利用圖來解決呢？」

「當一個問題牽扯到若干個體之間的某種關係時，可能可以利用圖來解決。」

圖演算法的技巧，是將一個問題表達成一個圖之後，在此圖上設計演算法，以找到問題的解。

一般而言，當一個問題是探討有關離散個體之間的二元關係(binary relation)時，此問題常可以被抽象成圖上的問題。若設計出的演算法能充分地利用到此圖的特性，則將更有效率地解決此問題。



## 學習評量

1. **關節點(articulation points)**：令圖  $G=(V, E)$  是一個無迴圈(loop-free)且連接的(connected)無方向圖。一個在集合  $V$  中的點  $v$  被稱為關節點，則圖  $G$  除去  $v$  後，是不連接的。請寫一個程式找出所有輸入圖的關節點。

### 輸入

6 (點的個數，並且以 1, 2, 3, …編號)  
8 (線的個數)  
1 2 (以下為線的資料)  
2 3  
2 4  
2 5  
3 4  
3 5  
4 5  
5 6

### 輸出

2 5 (所有關節點)

2. **四海之內皆兄弟**：一個村子有  $N$  位村民，已經知道那些人彼此是朋友。俗話說：四海之內皆兄弟；也就是 如果  $A$  和  $B$  是朋友 且  $B$  和  $C$  是朋友 則  $A$  和  $C$  也是朋友。請寫個程式，計算此村內共有多少群村民彼此擁有朋友關係。

### 輸入

7 (村民的個數  $N$ ，並且以 1, 2, 3, …編號)  
7 (朋友關係的個數)  
1 5 (以下為關係的資料)  
2 3  
2 4  
4 3  
1 6  
5 1  
7 1

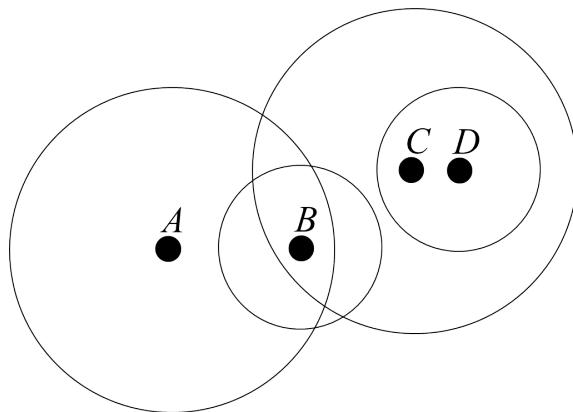
### 輸出

2 (擁有朋友關係的群體數目)

3. 收集器(sinks)：無線感測網路(wireless sensor networks)是由大量、小尺寸、低成本的感測器所組成。每個感測器可偵測各種訊號(包含溫度、溼度、照度或聲音)，並將收集到的訊號，透過鄰近的感測器傳送到網路上的收集器。在一個無限感測網路中，選擇一個特定的感測器來當作收集器，是一個十分基本的問題。

當兩個感測器的距離小於或等於  $R_A$  單位長度時，則感測器  $A$  可以直接傳封包給感測器  $B$ 。我們稱感測器  $A$  的通訊半徑是  $R_A$  單位長度。明確地說，當 $(x_1, y_1)$ 和 $(x_2, y_2)$ 分別是  $A$  和  $B$  的座標時，如果 $(x_1-x_2)^2+(y_1-y_2)^2 \leq R_A^2$  則感測器  $A$  可以直接傳封包給感測器  $B$ 。注意，因為每個感測器的通訊半徑可能不同，當感測器  $A$  可以直接傳封包給感測器  $B$  時，感測器  $B$  不必然可以直接傳封包給感測器  $A$ 。

假設每個感測器的通訊範圍是一個圓圈，而其中心點就是感測器的位置。



在上圖中有四個感測器  $A$ 、 $B$ 、 $C$ 、 $D$  佈署在平面。令感測器  $A$ 、 $C$  的半徑是 4 個單位長度(即  $R_A=R_C=4$ )，而感測器  $B$ 、 $D$  的半徑是 2 個單位長度(即  $R_B=R_D=2$ )。因為  $A(C)$  圓圈包含  $B$  的中心點， $A(C)$  可以直接和  $B$  通訊。但是  $B$  却不能直接和  $A(C)$  通訊，因為  $B$  的通訊範圍並沒有包含  $A(C)$  的中心點。同樣地， $D$  可以直接和  $C$  通訊，但是  $D$  不能直接送封包給  $B$  或  $A$ 。在上圖中，明顯地只有感測器  $B$  可以擔任收集器的工作。

請寫一個程式，計算一個輸入的感測網路中，最多有幾個感測器可當作收集器。

**輸入**

4  
1 6 4  
4 6 4  
7 8 4  
8 8 4

(感測器的總數)  
(以下是感測器的  $x, y$  座標及半徑)

**輸出**

4  
(收集器的個數)