

Tarjan

進階教學組

August 7, 2022

# 目錄

|     |                      |   |
|-----|----------------------|---|
| 1   | DFS tree             | 1 |
| 1.1 | 概念 . . . . .         | 1 |
| 2   | 連通                   | 3 |
| 3   | 連通圖                  | 3 |
| 4   | 連通分量                 | 3 |
| 4.1 | 邊雙連通分量/橋 . . . . .   | 3 |
| 4.2 | 點雙連通分量/割點 . . . . .  | 3 |
| 5   | Tarjan               | 4 |
| 5.1 | 無向圖/BCC . . . . .    | 4 |
| 5.2 | 有向圖/SCC . . . . .    | 6 |
| 6   | Korasaju's algorithm | 6 |

# 1 DFS tree

## 1.1 概念

這並不是真正的一個 **tree**，DFS tree 是一個在有向或無向圖中，把有成功進行到 DFS 的邊當成樹邊，然後變成一棵樹 + 其他邊的形式，所以依據選的根結點的不同、DFS 順序不同，DFS tree 也會跟著改變。

在實作上，我們也不會特別的把他求出來，只是剛好在 DFS 的過程中他會出現而已，而且他的特性會比較方便之後的講解

```
1 func DFS_tree(node current) {  
2     if current.isvisited :  
3         return;  
4  
5     //edge.from = a, edge.to = b  
6     for edge in edge_from_current :  
7         if not edge.to.isvisited :  
8             edge.to.is_tree_node <- true  
9             DFS_tree( edge.to )  
10        else  
11            edge.to.is_tree_node <- false  
12 }
```

上文的虛擬碼是找出 DFS tree 的邊，因為實作不複雜而且概念也十分基本就不放完整程式碼。

可以發現，圖上其實會有些邊沒有被選到，於是為了方便分類，人們幫這些沒有被選到的邊分了類並取了名字

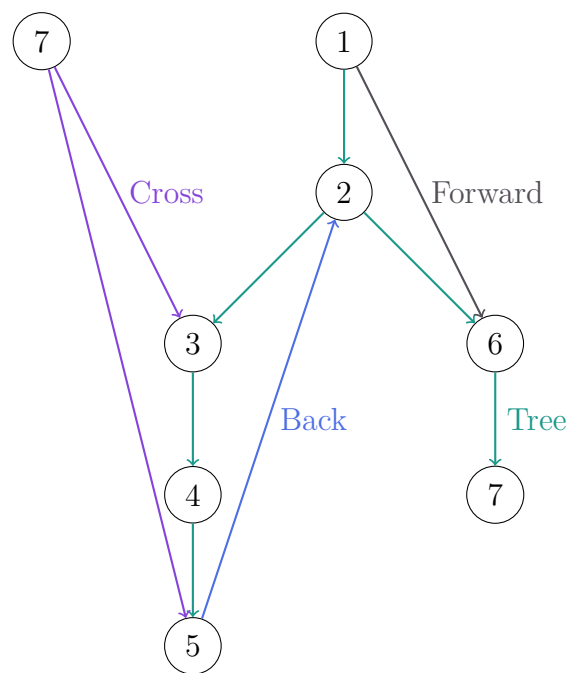
1. 樹邊 (Tree Edge): 在 DFS tree 上的邊
2. 回邊 (Back Edge): 從子節點連到父節點的邊

3. 前向邊 (Forward Edge): 從父節點連到子結點的子樹結點的邊
4. 交錯邊 (Cross Edge): 連到兩個非祖孫關係的邊

如果是無向圖的話就只會有 Tree Edge 跟 Back Edge

換個說法，你可以把 DFS tree 想像成一個族譜圖，把邊想像成認識的關係圖 (有向邊就是你認識他，但他不認識你，可能過年的時候很常出現)，以自身為節點的話如下。

1. 樹邊: 我認識我的孩子
2. 回邊: 我認識輩分比我大的直系血親
3. 前向邊: 我認識輩分比我低兩輩以上的血親
4. 交錯邊: 我認識我的旁系血親



(圖片原形 by 建中)

## 2 連通

### 嚴格定義

在一個無向圖中，若從點  $u$  到點  $v$  間有路徑，則稱他們是連通的。

在一個有向圖中，若從點  $u$  到點  $v$  間有路徑，且邊皆同向，則稱他們是連通的。

## 3 連通圖

### 定義

如果圖中任兩個點都是連通的我們稱他為連通圖，而有向圖還有分「強連通」跟「弱連通」，強連通是需保證  $u \rightarrow v$  跟  $v \rightarrow u$  皆有路徑且邊為同向，而弱連通只要滿足其一即可。不過弱連通圖好像較少討論，所以接下來講的都是強連通圖。

## 4 連通分量

### 4.1 邊雙連通分量/橋

#### 橋的定義

如果刪除一條邊時導致圖從**連通**  $\implies$  **不連通**我們則把這條邊稱為橋。亦即每刪除一個橋，連通塊的數量都會增加。在把所有的橋都移去之後，我們就可以得到邊雙連通分量。為什麼他要叫邊雙連通呢？因為想要讓圖的連通塊增加就必須刪除兩個邊以上。

### 4.2 點雙連通分量/割點

大致上和邊雙連通分量差不多，只是由刪除邊變成刪除點而已，而那個點稱為割點，如果把所有的割點都去掉之後我們就可以得到點雙連通分量。

## 5 Tarjan

### 5.1 無向圖/BCC

Tarjan 的無向圖比較簡單理解一些，因為無向圖有如下幾種性質

1. 沒有 Cross Edge 和 Forward Edge

#### 性質解釋

1. 因為無向圖在 DFS 時，任兩個相鄰的點  $u, v$  的  $t_i, n$  差距應該為 1 顯然 Cross Edge 並不符合。

如果知道了以上的性質，我們要如何找出所有的橋呢？首先，我們可以發現，

對於無向圖，我們只需要去考慮 Tree Edge 跟 Back Edge。先不要一次想太多，那我們不妨從只有 Tree Edge 開始，之後再一個個的把 Back Edge 加回去。

一開始，對於每一個 Tree Edge 都是橋，只要刪掉任何一個邊都會多出一個連通塊，此時，你會發現其實對於無向圖來說，**最終有大小大於 2 的連通塊 = 有環**。因為沒有環就等於只有 Tree Edge。而對於一個 Tree 來說，增加 Back Edge 就會出現**環**，讓我們來觀察一下**環**的性質，你可以發現，**環**會是 Back Edge 中間的那一條線。

那要如何去維護呢，我們可以對每個點  $u$  紀錄一個  $low$  值，即他「從  $u$  或  $u$  的子孫最多走一次回邊 (可以不走)，最高可以到達的點」。接下來講結論：「對於一條 Tree Edge  $e = (a, b)$  而言，如果  $low(b) = b$  則代表  $e$  是橋」。

因為把  $e$  移除掉之後。因為  $low(b) = b$ ，代表其  $a$  的子孫最高只能走回  $b$ ，所以如果這時把  $e$  移除掉的話，其子孫就沒有其他方式可以走回  $a$  和其祖先了，表示  $a$  的子孫和原圖變得不連通了。

實作時主要是用 DFS+ **進入戳記**。在實作時需要知道幾個觀念

1. DFS 會先把所有子孫處理完才會繼續處理當前節點
2. DFS 中，在沒有遇到分岔的情況下，遍歷的順序總是一條條線。
3. 只有互為**直系親屬** 才可以使用進入戳記來判斷層樹高低，不然如果先遞迴到的子樹其進入戳記會比後遞迴到的子樹小。

這部分可以自行隨意劃一個 DFS tree 實際手做遞迴一次。

## Algorithm: Tarjan BCC

```
1 Time_count = 0;
2 Stack node_save;
3 func Tarjan(node current) {
4     current.time_in, current.low <- Time_count
5     Time_count <- Time_count+1
6     node_save.push( current)
7
8     for edge in edge_from_current
9         if( not edge.to.isvisited )
10             Tarjan( edge.to )
11         else
12             current.low = min( current.low, edge.low)
13
14     if( current.time_in == current.low )
15         node temp;
16         do {
17             temp = node_save.top()
18             bcc[ temp.id ] = current.id
19             node_save.pop()
20         }
21         while(temp != current);
22 }
```

上面的 code 是求邊雙連通分量的，如果想要求點雙連通分量的話還需要多紀錄一個 father 值。和求邊雙連通分量的程式碼差不多，只不過在對於  $e = (a, b)$  且  $low(b) = b$  時，這時的割點會是  $a$ 。

```
1 func Tarjan( node current, node father)
```

## 5.2 有向圖/SCC

有向圖大部分都和無向圖差不多，不過有向邊多了 Cross Edge，所以**概念3**的部分可能會出現錯誤，因為 Cross Edge 會碰到沒有直系親屬關係的節點。那我們要如何解決呢？

十分的簡潔，在從結點  $u$  走到 low 函數更低的節點  $v$  時，我們可以先判斷  $v$  在不在 Stack 內。因為根據**觀念1**如果  $v$  是祖先，他一定還沒被踢出 Stack，而且根據**觀念2**，如果  $v$  在別的子樹且還未被踢出 Stack 外的話，代表  $[v, LCA(u, v))$  以下的節點都已經被拜訪完了，也就是  $u, v$  會有共同的連通塊

Tarjan 的複雜度只需要一次 DFS，所以為  $O(V + E)$ ， $V$  是點的個數， $E$  是邊的個數。

## 6 Korasaju's algorithm

這個演算法相對簡單一些，他妥善的使用了強連通圖的定義「從  $u \rightarrow v$  和  $v \rightarrow u$  都有一條路徑且邊皆同向」。可以發現對於 SCC 來說，把邊翻轉並不會影響他的答案，所以如果我們有一個圖  $G$ ，我們只需要把所有邊翻轉變成  $G'$ 。然後對  $G$  和  $G'$  都做一次 DFS 判斷連通性，最後取交集即可。

那無向圖呢？無向圖的邊沒有辦法翻轉啊？其實也是一樣的，只要把 Back Edge 當成有向的即可，反轉的時候會變成 Forward Edge，之後的作法就和有向圖相同