Graph Algorithm 2

Lecture by casperwang
Credit by baluteshih
Credit by qazwsxedcrfvtg14

Sprout

課程內容

- 無向圖連通性
  - 觀念複習
  - Tarjan 實作細節
  - 一些重要性質
- 有向圖連通性
  - Kosaraju Algorithm
  - 2-SAT 問題
- 仙人掌

無向圖連通性

關於 Tarjan

到底要怎麼念？？

※ 發信站: 批踢踢實業坊(ptt.cc), 來自: 222.251.19.60
※ 文章網址: http://www.ptt.cc/bbs/joke/M.1396921214.A.6E5.html

推	yeaaah:明明就不是念Tarjan	04/08 10:11
推	a775126:蠻好笑的	04/08 10:15
推	bomber0529:我都念Tarjan欸	04/08 10:17
推	asdfgh0920:常常有人念Tarjan 但其實應該要唸做Tarjan才是道地的發音	04/08 10:19
推	inglee:美國人都唸Tarjan 但台灣廣告都唸Tarjan	04/08 10:20
推	yjason:其實Tarjan跟Tarjan都有人念 但正統原音應該是念Tarjan	04/08 10:29
推	expectme:念Tarjan真的是通病，國外廣告一定都念Tarjan的	04/08 10:31
推	easonhp20010:我爸爸都念IKEA我是習慣念Tarjan	04/08 10:35
推	mitsurino:我也都唸Tarjan	04/08 10:36
推	yys310:我也唸Tarjan	04/08 10:39
推	heretomo:所以到底是念Tarjan還是念Tarjan?	04/08 10:39

關於 Tarjan

- Tarjan 症候群
- 第一期，你以為你會了 Tarjan
- 第二期，你發現你其實並不會 Tarjan
- 第三期，你覺得你真正了解了 Tarjan
- 第四期，你發現你還是不會 Tarjan
- ...

Robert Endre Tarjan

- 1986 年圖靈獎得主
- 橋、割點、雙連通分量、強連通分量
- Fibonacci heap（斐波那契堆）
- LCA（最近公共祖先）
- Splay Tree（伸展樹）
- ...

DFS Tree

- Tree Edge:真正在 DFS 樹上的邊，從父親連往小孩
- Back Edge:從子孫連回祖先的邊
- Forward Edge:連向沒有直接親子關係的子孫的邊
- Cross Edge:連向非直系血親的邊

Question:在無向圖上 DFS 時有哪些是不可能出現的？

例題 - 道路彩繪

- 給定一張簡單無向圖，保證每個點的度數不超過 3
- 請將所有邊上黑色或白色，使得同一種顏色的邊不會形成環 or 判斷無解
- 期望時間複雜度 O(V+E)

例題 - 道路彩繪

- 先考慮單一個連通塊就好
- 沒有環 → 生成樹
- 如果讓某種顏色的邊是生成樹呢？

例題 - 道路彩繪

- 先考慮單一個連通塊就好
- 沒有環 → 生成樹
- 如果讓某種顏色的邊是生成樹呢？
  - 直接對該連通塊隨便找生成樹塗成黑色
  - 如果白色的邊形成環，那麼這個環絕對不可能經過內節點
  - Why?

例題 - 道路彩繪

- 白色的環只有可能接在根或葉節點上
  - 葉節點跟葉節點之間的邊 → cross edge
    - 無向圖的 dfs tree 沒有 cross edge
  - 所以，環只可能接在
    - 不超過 1 個葉節點
    - 度數只有 1 的根節點
  - 但這樣只有兩個點，圖又是簡單的 → 沒有環
- 肯定有解

Low 函數

定義：

在不透過 Tree Edge 的情況下，最多經過一條 Back Edge 能夠到達深度最淺的祖先的深度

Low 函數的維護

假設現在要從 u 走到 v

- Case 1: visited[v] == false

  此時 v 是 u 在 DFS 樹上的小孩，u 有機會透過 v 走到祖先，所以先對 v 做 DFS，再將 low(u) 與 low(v) 取 min

- Case 2: visited[v] == true

  此時 v 會是 u 在 DFS 樹上的祖先 (why?)，那麼根據定義 low(u) 要與 v 被遍歷的 dfs 序取 min。

例題 - Cycling City

- 給定一張簡單無向圖，問是否存在一組點對 (a,b) 使得 a 可以有「三條」互不共用點且不共用邊的路徑走到 b，若存在則請輸出那三條路徑
- 期望時間複雜度 O(V+E)

例題 - Cycling City

- 考慮一個節點 a，如果他有兩條 Back Edge
- 這兩條 Back Edge 指向的節點一定都是他的祖先
- 令深度比較深的祖先為 b
- a，b 之間存在三條不共用點的路徑！

定義複習

- 橋？
- 邊雙連通分量？
- 關節點？
- 點雙連通分量？

How to 找橋

- 對於節點 v 以及他的父親 u，如果 low(v) > dfs(u) 的話，就代表 v 在不依靠 (u, v) 這條邊的情況下是永遠沒辦法走到 u 的
- (u, v) 是橋！

找橋ㄉ扣

```
void DFS(int u, int pa) {
  dfs[u] = low[u] = ++cnt;
  for (int v : G[u]) {
    if (v == pa) continue;
    if (!dfs[v]) {
      DFS(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] > dfs[u]) {
        // Edge(u, v) is a bridge.
      }
    } else if (dfs[v] < dfs[u]) {
      low[u] = min(low[u], dfs[v]);
    }
  }
}
```

How to 找邊雙連通分量

- Method 1:把橋都刪掉然後找連通塊
- Method 2:修改原本的 Tarjan
  - 在遞迴的過程中維護一個 stack
  - 在每次進入一個節點的時候把該節點推入堆疊中
  - 當我們在節點 u 確定他和自己的孩子 v 所連的邊 (u, v) 是橋的時候, 堆疊中從最上面一路取出, 一直到節點 v, 剛好就形成了 v 所在的邊雙連通分量！

找邊雙連通分量ㄉ扣

```
void DFS(int u, int pa) {
  dfs[u] = low[u] = ++cnt, stk.push(u);
  for (int v : G[u]) {
    if (v == pa) continue;
    if (!dfs[u]) {
      DFS(v, u), low[u] = min(low[u], low[v]);
      if (low[v] > dfs[u]) {
        bcc_cnt++;
        while (stk.top() != v)
          bcc[bcc_cnt].pb(stk.top()), stk.pop();
        bcc[bcc_cnt].pb(stk.top()), stk.pop();
      }
    } else if (dfs[v] < dfs[u])
      low[u] = min(low[u], dfs[v]);
  }
}
```

How to 找關節點

- 如果 v 的子孫們沒有除了透過 v 到達 v 的祖先的方法的話，那麼將 v 從圖上移除之後，v 的子孫們就再也無法走到 v 的祖先，也就代表圖變得不連通了
- 當 low(v) >= dfs(u) 的時候，u 就是一個關節點
- 特例是當 u 是根節點的時候
  - 此時 u 並沒有祖先
  - u 是關節點的條件變成在 dfs tree 上有超過一個小孩
  - Why?

找關節點ㄉ扣

```
void DFS(int u, int pa) {
  dfs[u] = low[u] = ++cnt;
  int chd_cnt = 0;
  for (int v : G[u]) {
    if (v == pa) continue;
    if (!dfs[u]) {
      chd_cnt++, DFS(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] >= dfs[u] && pa != -1)
        // u is a non-root cut-vertex
    } else if (dfs[v] < dfs[u])
      low[u] = min(low[u], dfs[v]);
  }
  if (pa == -1 && chd_cnt > 1)
    // the root of the tree is a cut-vertex
}
```

How to 找點雙連通分量

- Method 1:把關節點都刪掉然後找連通塊(???

How to 找點雙連通分量

- ~~Method 1: 把關節點都刪掉然後找連通塊(???~~
- Method 2:
  - 一樣在 DFS 的時候維護一個 stack
  - 每次遇到一個新的點時便將它推入 stack
  - 當發現一個點是關節點時，便從 stack 頂端取出點直到遇到現在的這個關節點，則這些點就會剛好形成一個點雙連通分量。

找點雙連通分量ㄉ扣

```
void DFS(int u, int pa) {
  dfs[u] = low[u] = ++cnt, stk.push(u);
  for (int v : G[u]) {
    if (v == pa) continue;
    if (!dfs[u]) {
      DFS(v, u), low[u] = min(low[u], low[v]);
      if (low[v] >= dfs[u]) {
        bcc_cnt++;
        while (stk.top() != u)
          bcc[bcc_cnt].pb(stk.top()), stk.pop();
        bcc[bcc_cnt].pb(stk.top());
      }
    } else if (dfs[v] < dfs[u])
      low[u] = min(low[u], dfs[v]);
  }
  if (pa == -1) stk.pop();
}
```

點雙連通ㄉ性質 1

- 對於一個點雙連通分量內的任兩個節點 u 和 v，都存在至少兩條從 u 到 v 的路徑，且這兩條路徑沒有共用點
- 證明：

點雙連通ㄉ性質 1

- 對於一個點雙連通分量內的任兩個節點 u 和 v，都存在至少兩條從 u 到 v 的路徑，且這兩條路徑沒有共用點
- 證明：
  - 假設存在 u 和 v 使得 u 到 v 沒有兩條不相交的路徑
  - 那代表 u 到 v 的所有路徑一定會經過某條個點 w
  - 只要把這個點移除，u 就不能到 v 了，矛盾！

點雙連通ㄉ性質 2

- 對於一個超過三個點的點雙連通元件內的任三個節點 u,v,w, 一定存在一條從 u 到 v, 再從 v 到 w 的簡單路徑
- 證明:

點雙連通ㄉ性質 2

- 對於一個超過三個點的點雙連通元件內的任三個節點 u,v,w, 一定存在一條從 u 到 v, 再從 v 到 w 的簡單路徑
- 證明:
  - 考慮性質 1, u 到 v 給出的兩條不相交路徑形成的環
  - 若環上包含 w → 顯然有
  - 若環上不包含 w → 隨便從 w 找一條連到環上任意一個點的簡單路徑, u 從被連到的點的另一側繞到 v, 再繞到 w 即可！

點雙連通ㄉ性質 3

- 對於任兩條邊，他們可以在同一個簡單環上若且唯若他們同屬於同一個點雙連通分量
- 證明：

點雙連通ㄉ性質 3

- 對於任兩條邊，他們可以在同一個簡單環上若且唯若他們同屬於同一個點雙連通分量
- 證明：
  - 兩條邊可以在同一個簡單環 → 他們屬於同一個點雙連通分量
  - 兩條邊屬於同一個點雙連通分量 →
    - 兩條邊有共點 → 根據性質 1 顯然有環
    - 假設第一條邊連接著 a,b，第二條邊連接著 c,d
    - 根據性質 2 找得到一條 a->b->c 的簡單路徑，若 b->c 中間有經過 d，則不失一般性將 c,d 交換，因此有 a->b->c->d 的簡單路徑存在
    - 考慮從 a 出發的 dfs tree，樹邊一路恰好沿著 a->b->c->d 的路徑走，根據 low 函數的性質，low[d] <= low[a] 因為是同一個點雙連通分量
    - 因此可以從 d 往 dfs 子樹走一段路後直接用一條 back edge 跳回 a 來構出一個環

例題 - Simple Cycles Edges

- 給定一張 V 點 E 邊的圖，問你哪些邊恰好在一個簡單環上
- 期望時間複雜度 O(V+E)

例題 - Simple Cycles Edges

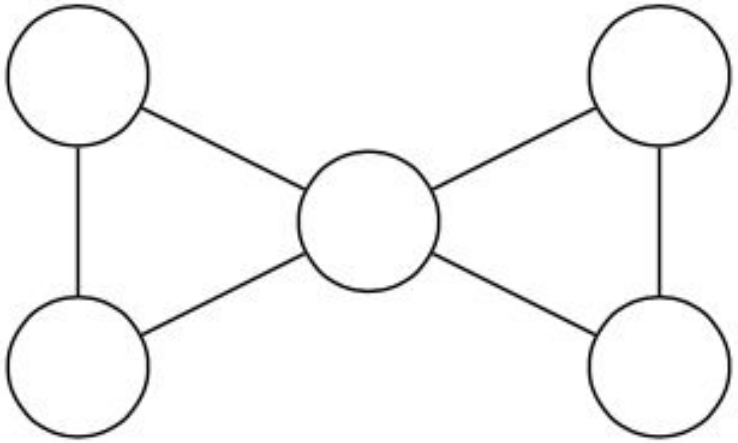
- 對於每條邊，他只需要在意自己所在的點雙連通分量即可
- 只要點雙連通分量內的點和邊一樣多，也就是形成一個整個環，那麼這些邊全都是答案
- Why?
- 點數大於邊數 → 沒環
- 點數小於邊數 → 考慮一條邊形成的一個簡單環，他又可以跟一個不在該環上的邊形成另一個簡單環(Why?)，他不是答案！

例題 - 最大最小生成樹

- 給定一張 V 點 E 邊的帶權無向圖
- 找出至少要修改幾條邊的邊權才能使得圖上的最大生成樹與最小生成樹的權值總和一樣
- 期望時間複雜度 O(V+E)

例題 - 最大最小生成樹

- 把所有邊都改成一樣的話肯定可以！
- 有沒有可能再少改一點邊？
  - 對於特定的圖會存在不把每條邊的邊權改到一樣就讓最大生成樹與最小生成樹的權值總和一樣的方法



有向圖連通性

強連通分量

- 「縮點」是強連通分量的一個重要應用
  - 將所有隸屬於同一個強連通元件的點縮成一塊
- 縮完後整張圖會變成一個 DAG
- 比起圖，DAG 上面可以做更多事情(ex. dp)

Kosaraju Algorithm

- 另一派的強連通分量演算法
- Tarjan 的強連通有點複雜，通常大家比較常用 Kosaraju

Kosaraju Algorithm

- 如果 u 能走到 v，那麼 u 跟 v 在同一個強連通元件上若且唯若 u 在反圖上也走得到 v
- 嘗試 dfs 一遍整張圖，把點依照離開順序由大到小排序
  - 請不要花 O(nlogn) 排序
- 若 u 能走到 v，v 不能走到 u，那麼 u 的離開順序肯定比較大 → 從反圖上離開順序最大的開始 dfs，肯定只會走到同一個強連通元件內的點！

Kosaraju Algorithm

- 簡單來說，就是在反圖上按照離開順序由大到小 dfs！

Kosaraju Algorithm

- 可以用 DAG 的例子來想像 Kosaraju 在做的事
- dfs 一遍照離開順序由大到小 → 拓撲順序
- 離開順序最大的點 → 入度 0 的點
- 反圖後這個點就變成出度 0 的點了，他走不到任何人，所以也就當成只會單純蒐集這一個點成一個強連通元件
- 所以我們只是在一直把反圖上出度 0 的點依序移除，你可以當成每個點其實就是一個強連通元件縮成一個點的樣子

例題 - 芽芽逛大街

- 給定一張有向圖，每個點有權重
- 問從一個點出發走一條路徑所能走到的最大權重和
- 期望時間複雜度 O(V+E)

例題 - 芽芽逛大街

- 如果是 DAG 呢？
  - DAG 上最長路徑可以直接 dp
- 先把他縮成 DAG 就好了！

2CNF-SAT

Definition 6 *CNF (Conjunctive normal form)* 的定義如下：

- 一個布林變數 (Boolean variable) v 只可能為兩種值：True(T) 或 False(F)。若 $v = \mathtt{T}$，則 $\neg v = \mathtt{F}$；若 $v = \mathtt{F}$，則 $\neg v = \mathtt{T}$。
- 一個文字 (literal) l 形如 v 或 $\neg v$
- 一個子句 (clause) c_i 形如 $(l_1 \vee l_2 \vee \cdots \vee l_{m_i})$
- 一個合取範式 (CNF) 的布林算式 (Boolean formula) 形如 $c_1 \wedge c_2 \wedge \cdots \wedge c_n$

(注：$\vee$ 為布林運算中的「或 (OR)」，$\wedge$ 為布林運算中的「且 (AND)」)

舉例來說，算式 $(a \vee b) \wedge (\neg b \vee c)$ 是 CNF，而算式 $(a \wedge b) \vee (\neg b \wedge c)$ 則不是 CNF。

2CNF-SAT

- 給你一段形如 (a or b) and (c or d) and ... 的式子
- 詢問你是否有一組合法的解

2CNF-SAT

- 假設有 N 個變數 X1~XN，建一個 2N 個點的有向圖，代表 X1~XN 和 ¬X1~¬XN
- 每個 clause 都形如 (ai v bi)，因此對以下條件建邊
  - ¬ai -> bi
  - ¬bi -> ai
- 建一條邊的意義是什麼？
  - 若圖內有 x 走得到 y，那麼「若 x 是 true，y 也得是 true」

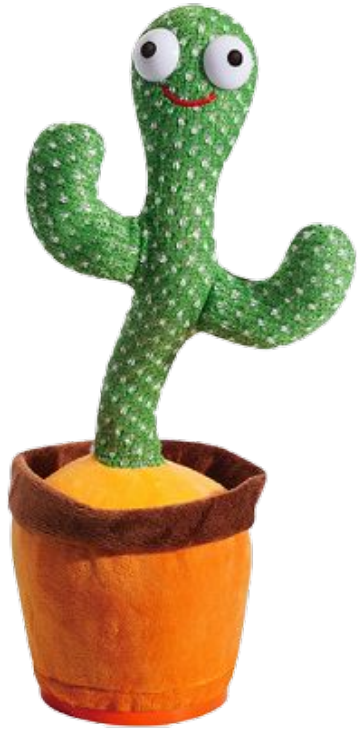
2SAT

- 直接對該圖做強連通分量縮點！
- 如果 Xi 和 ¬Xi 在同一個強連通分量？
- 若有解的話，可以直接用拓撲順序構出來

仙人掌

定義

- 一張無向連通圖可以被稱為仙人掌若且唯若這張圖裡頭的每條邊最多只在一個環內



仙人掌ㄉ酷性質

- 仙人掌圖有一個古老的名字叫「Husimi tree」，後來才漸漸 被改稱為「cactus」
- 在仙人掌中的任意兩個環至多只會有一個共同節點
- 一個仙人掌可以被稱為「聖誕仙人掌」若且唯若它的每個節點都只會在至多兩個環內

仙人掌有什麼優點ㄇ

- 最常見的仙人掌處理方式是將其用點雙連通分量縮點，建成「圓方樹」
- 因為仙人掌的每條邊最多只在一個環內，圓方樹上的方點代表的就是一個環，而環上的資訊維護和一般正常的圖縮成 Block-Cut tree 後會出現的點雙連通分量相比是相對容易的

例題 - How Dare You

- 給定一棵包含 N 個節點 M 條邊的仙人掌，接下來有 Q 筆詢問，每次詢問從 si 走到 ti 的最短距離以及有幾種可以達成最短距離的走法

例題 - How Dare You

- 發現 s 走到 t 的最短路徑其實只跟每次經過環的時候順 or 逆時針繞有關
- 對於每個環維護關節點之間的距離(可以用前綴和維護)
- 將仙人掌問題回歸到樹上路徑詢問後就可以套上倍增法，AC~