

DISJOINT SETS

Michael Tsai

2012/04/10

Equivalence Relation

- Set: 一個group的elements, 沒有次序
- 假設S為包含所有元素的集合
- 兩個element a和b的relation R稱為**equivalence relation**, iff:
 1. Reflexive: 對每個element $a \in S$, $a R a$ is true.
 2. Symmetric: 對任兩個elements $a, b \in S$, if $a R b$ is true, then $b R a$ is true.
 3. Transitive: 對任三個elements $a, b, c \in S$, if $a R b$ and $b R c$ is true, then $a R c$ is true.
- 例: 道路連接性 (road connectivity)是equivalence relation

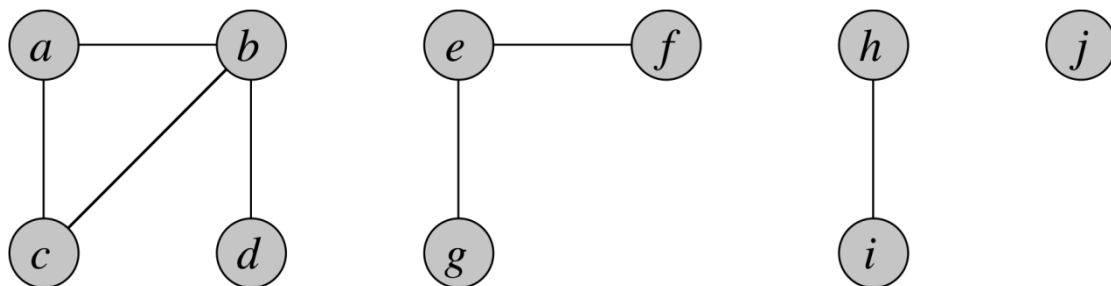
Equivalence Class

- The equivalence class of an element a :
一個包含S中所有和 a 有equivalence relation的elements的集合
- $\{\forall e \in S \text{ s.t. } e R a\}$
- 假設我們把S中所有元素分到不同的equivalence class, 則每個元素只會屬於一個equivalence class
 - 任兩個equivalence class S_i, S_j 都符合 $S_i \cap S_j = \phi$, if $i \neq j$. \rightarrow **Disjoint sets!**
 - Equivalence class把原本的S切(partition)成數個equivalence class
- 道路連接性的例子: 如果兩個城市有路連接, 則它們屬於同一個equivalence class

Operation on Disjoint Sets

- MAKESET(x):
做一個新的set, 只包含element x
- UNION(x,y):
將包含x的set和包含y的set合併成為一個新的set (原本包含x和包含y的兩個set刪掉)
- FIND(x):
找出包含x的set的“名字”(ID號碼)
- UNION之前通常要先用FIND確定兩個element屬於不同set
- 問: 如何表示Disjoint Sets, 使得這些operation可以快速地執行呢?

例子：尋找兩個城市是否連接



- 給一些城市, 及所有道路(每條道路連接兩個城市)

```

for each city C
    MAKESET(C)
for each road (x, y)
    if FIND(x) != FIND(y)
        UNION(x, y)
  
```

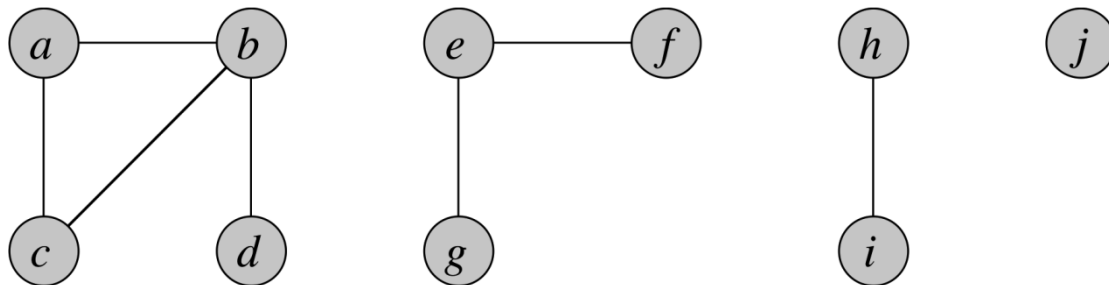
- 如何知道兩個城市是否連接?

```

Boolean CITY_CONNECTED(x, y) {
    if FIND(x) == FIND(y)
        return TRUE;
    else
        return FALSE;
}
  
```

例子：尋找兩個城市是否連接

Edge processed	Collection of disjoint sets									
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}



要怎麼表示集合呢？ 方法一

- 方法一: Array法 – Find很快, Union很慢

Index代表的是每個element的號碼

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	4	3	1	2	2	2

Array裡面的值紀錄的是該element所屬的set ID

- 上面的例子共有四個SET:

$$S_1 = \{3\}, S_2 = \{4,5,6\}, S_3 = \{0,2\}, S_4 = \{1\}$$

- FINDSET(x)?

- 直接看array的值

- 時間複雜度?

$O(1)$



- UNION(x,y)?

- 要把所有跟x同set的element都改set ID成跟y的set ID一樣

- 時間複雜度?

$O(n)$



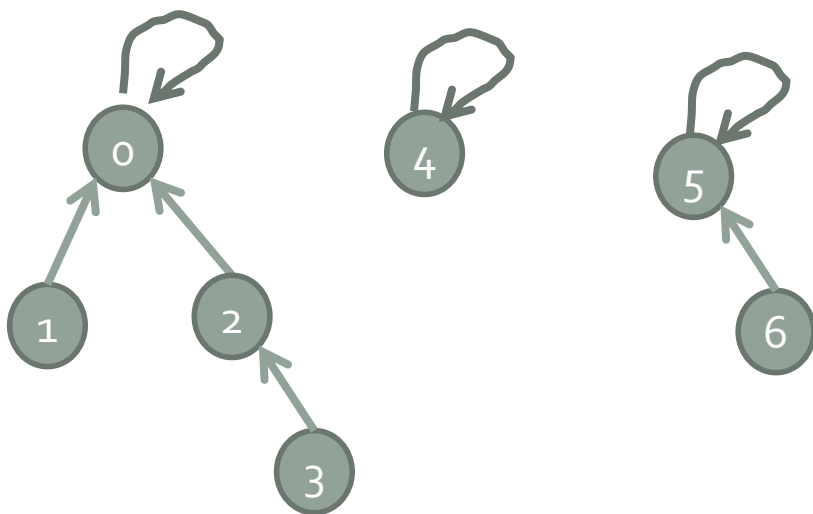
要怎麼表示集合呢？ 方法二

- 方法二: tree法 – Union很快, Find有點慢

Index代表的是每個element的號碼

[0]	[1]	[2]	[3]	[4]	[5]	[6]
0	0	0	2	4	5	5

Array裡面的值紀錄的是該element的“parent”

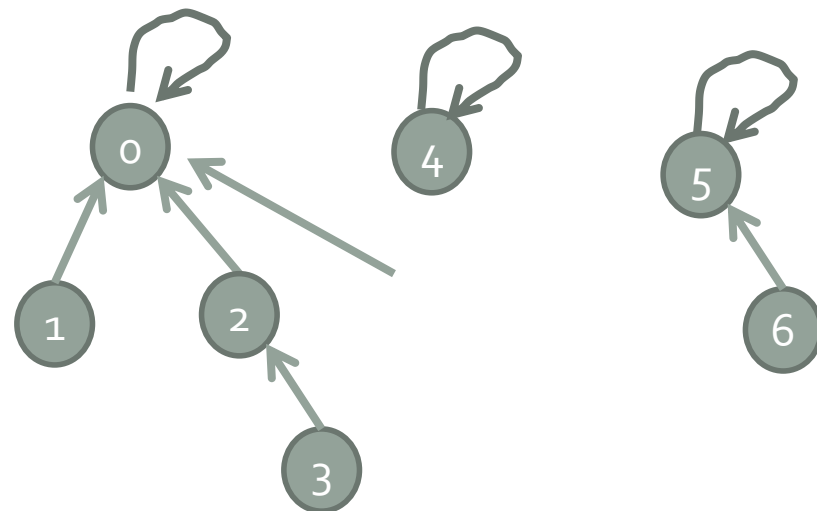


反正我們並不是那麼重視set ID or name, 只要可以做 $\text{FINDSET}(x) == \text{FINDSET}(y)$ 的比較!

→ 用每個set的root來代表那個set即可!

要怎麼表示集合呢? 方法二

- FIND(x)?
 - 必須找到該“tree”的root
- 時間複雜度?
- 跟樹的高度有關!
- Worst case: skew tree (一條龍)



- UNION(x,y)?
 - 把element x的set的root的parent (array的值)設成y (或反過來)
- 例如UNION(2,4)
- 時間複雜度?
- 如果不計算找root的部分 (通常需要先用FIND 檢查兩個是否為同set)



方法二：改良版 Weighted Union

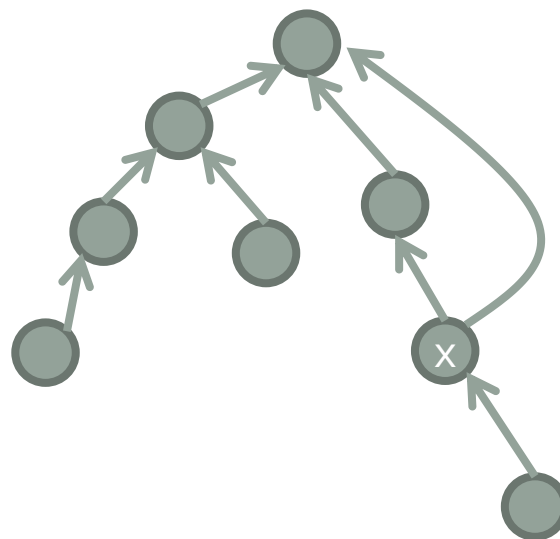
- 之前的問題在於, FIND的時候如果碰到skew tree就會變成 worst case, $O(n)$
- 如果從一開始(每一個set只有一個element)的時候, 每次 UNION的時候仔細選擇誰要當新的tree的root, 則可以避免這個問題!
- Weighted Union: Union的時候用某種“weight”來決定誰當 root
 1. **Union by size:** 每個set (tree)紀錄裡面有幾個node (element). Size大的set的root當合併之後的tree的root.
 2. **Union by height:** 每個set (tree)紀錄裡面tree的高度. 比較高的set的root當合併之後的tree的root.
- 使用兩者的執行時間相似, 下面使用Union by height舉例

Union by Height

- 考慮某個element x , 一開始它所屬的set只有1個element
- 跟別人union的時候, 如果加入別人(別人當root)就是比較小的
- 第一次union的時候, 如果是加入別人, 產生的set最少有兩個element
- 第二次union的時候, 如果是加入別人, 產生的set最少有四個element
- ...
- → 每次加入別人(高度增加)的時候, tree的size最少會變兩倍大
- 每個FIND最多只會花 $O(\log n)$ 
- UNION的部分不變! $O(1)$ 

方法二：開外掛加強版 Path Compression

- FIND還是太慢了
- 有沒有什麼方法可以加快?
- 從某一個node往上走的路上, 每一個parent都改指到root
- 時間複雜度還是一樣, constant變大而已
- 下一次**FIND**就快得多
- 此方法叫做path compression



Amortized Analysis

(多個operation一起考慮)

- 假設有m個 UNION 或 FIND operation, n個element

方法	FIND(x)	UNION(x,y)	m個UNION+FIND
方法一: array法			
方法二: tree法			
方法二: tree法 +Weighted Union			
方法二: tree法 +Weighted Union+Path Compression			

Today's Reading Assignment

- Karumanchi chapter 8
- (可參考Cormen chapter 21)