DISJOINT SETS

Michael Tsai

2017/05/09

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 $a \in S$, $a \mathbb{R} a$ is true.
  2. Symmetric: 對任兩個elements $a, b \in S$,
     if $a \mathbb{R} b$ is true, then $b \mathbb{R} a$ is true.
  3. Transitive: 對任三個elements $a, b, c \in S$,
     if $a \mathbb{R} b$ and $b \mathbb{R} c$ is true, then $a \mathbb{R} c$ 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
- $\{\forall e \in S\ s.t.\ e\ \mathbb{R}\ a\}$

- 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 $i \neq j$. → **Disjoint sets!**
  - Equivalence classes 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

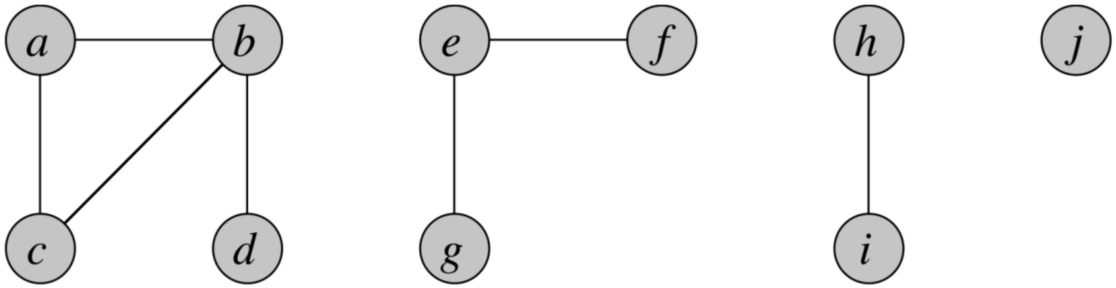
- MAKE-SET(x):
  做一個新的set, 只包含element x

- 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-SET(x):
  找出包含x的set的"名字"(ID號碼) or 代表號碼

- UNION之前通常要先用FIND-SET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
       MAKE-SET(C)
for each road (x,y)
       if FIND-SET(x)!=FIND-SET(y)
              UNION(x,y)
```

- 如何知道兩個城市是否連接?

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

Running Time Analysis

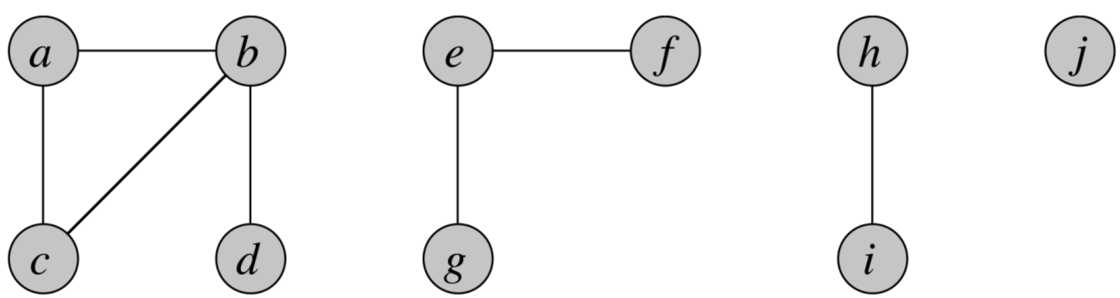
- 定義:
- MAKE-SET的執行次數: n (也就是到目前為止有幾個set)
- MAKE-SET, UNION, FIND-SET的總執行次數:m

- UNION最多n-1次
- $m \geq n$ (因為m包含了n次的MAKE-SET)

- 除了看單一個operation花多少時間, 有時候也會看m個operation 總共花了多少時間.

例子：尋找兩個城市是否連接

Edge processed	Collection of disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$



要怎麼表示集合呢? 方法一

- **方法一: Array法 – Find-Set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:

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

- FIND-SET(x)?
  - 直接看array的值
- 時間複雜度? $\boxed{O(1)}$

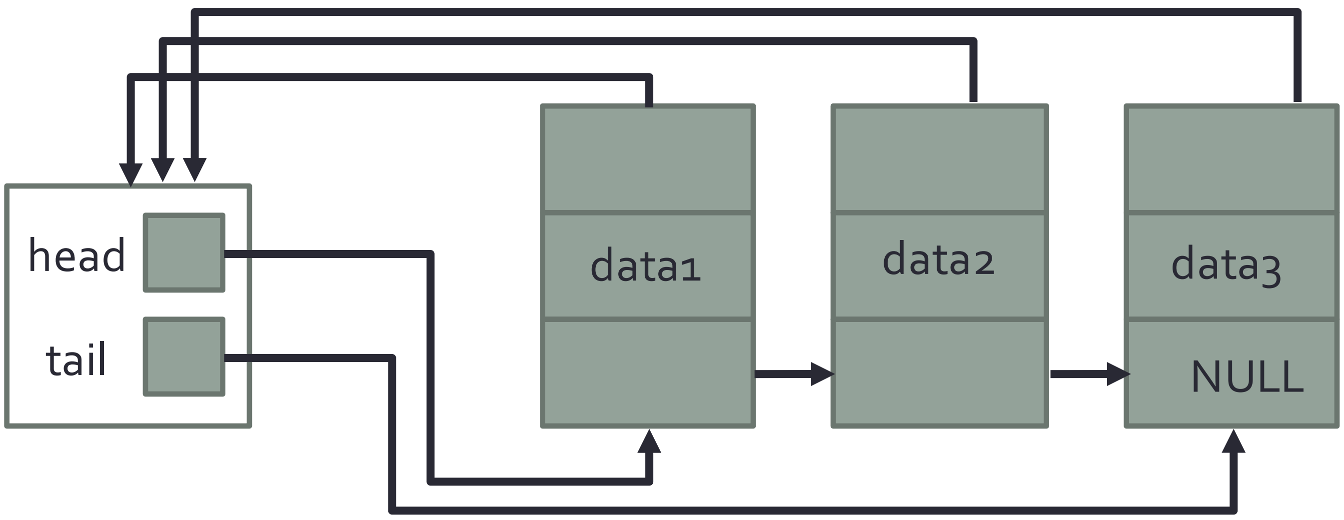
- UNION(x,y)?
  - 要把所有跟x同set的element都改set ID成跟y的set ID一樣
- 時間複雜度?

$\boxed{O(n)}$

要怎麼表示集合呢？方法二

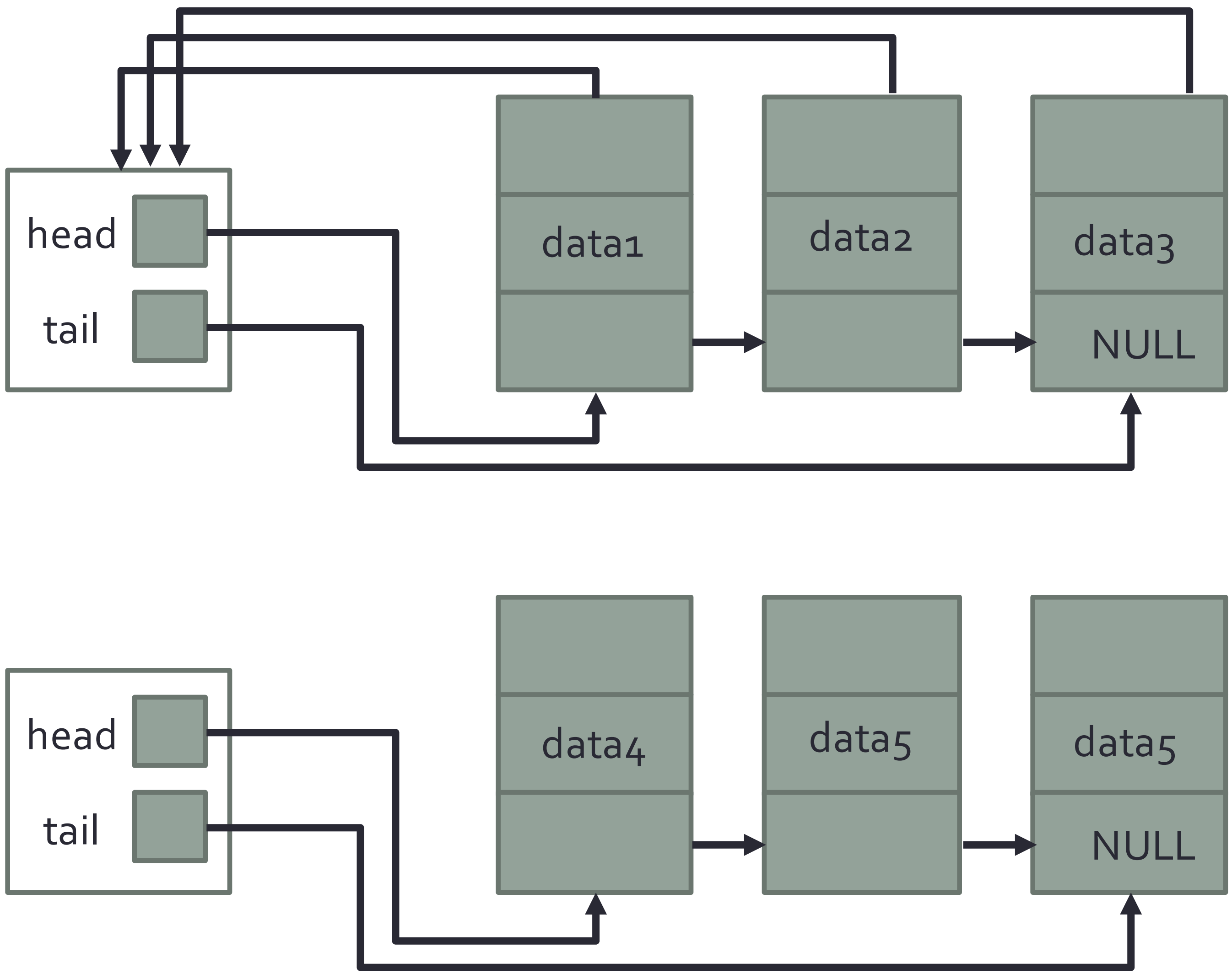
- **方法二: Array法 – Union很快, Find-Set有點慢**
- 如何表示? Hint: Tree

方法三 Linked-list Representation



- 使用上面的資料結構代表一個disjoint set
- MAKE-SET要花多少時間?
- FIND-SET要花多少時間?

方法三 Linked-list Representation



如何union?

花多少時間?

方法三 Linked-list Representation

- Worst-case?
- MAKE-SET for n items
- Union(s2, s1) // now s2 has two items {1,2}
- Union(s3, s2) // now s3 has three items {1,2,3}
- …
- Union (sn, sn-1) // now sn has n items

- Total time: $O(n^2)$

改良版 Weighted Union

- 之前的問題在於, 結合的時候沒有仔細選誰併入誰
- 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頭， 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. (必須記錄這些weight)
  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. (例如方法二可以適用)
- 使用兩者的執行時間相似, 下面使用Union by size舉例

Union by size

- 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
- ...
- →每次加入別人的時候, set的size最少會變兩倍大
- 如果某個item x的pointer被update $\lceil \log k \rceil$次, 則他所屬的set至少有k個東西.
- 最大的set只會有n個東西, 因此每個item最多update $\log n$ 次.
- 因此所有union的operation(包含在m次 operation中)最多花 $\boxed{O(\log n)}$

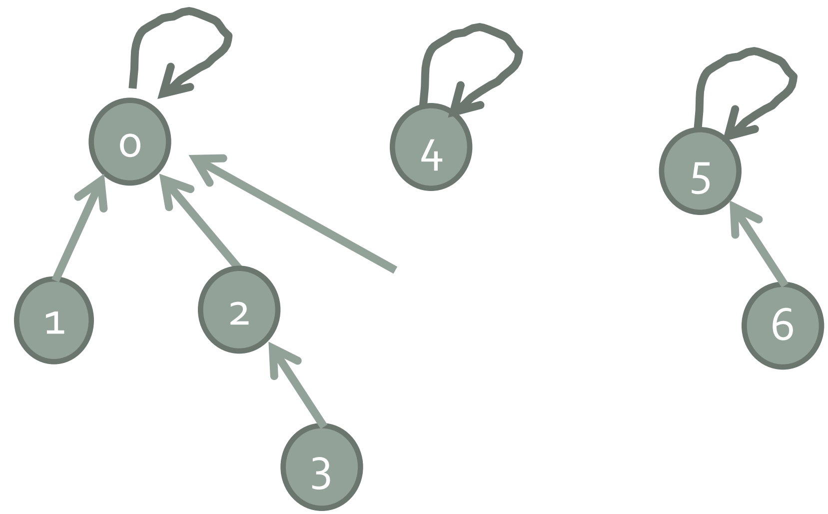
回到方法二

- FIND-SET(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)

$\boxed{O(n)}$

- 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)

$\boxed{O(1)}$

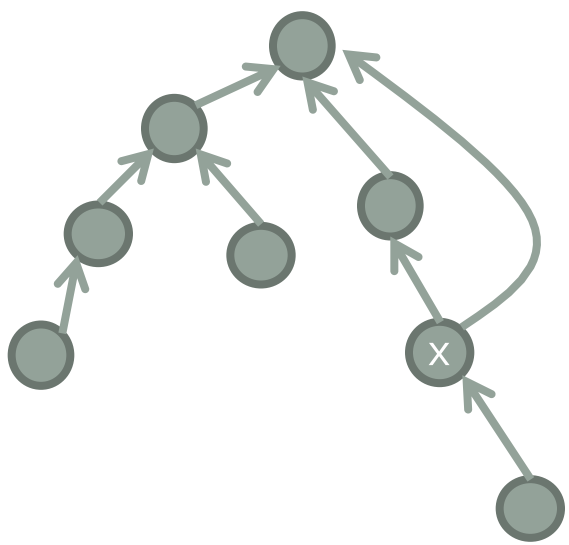


開外掛加強版1: Union by Height

- 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 $\boxed{O(\log n)}$ 
- UNION的部分不變! $\boxed{O(1)}$ 

開外掛加強版2 Path Compression

- FIND-SET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-SET**就快得多
- 此方法叫做path compression



Amortized Analysis (Averge)

方法	FIND(x)	UNION(x,y)	m個MAKET-SET+ UNION+FIND-SET
方法一: array法	O(1)	O(n)	O(m+n^2)
方法三: linked list	O(1)	O(n)	O(m+n^2)
方法三: linked list+ Weighted Union	O(1)	O(log n)	O(m+n log n)
方法二: array (tree)法	O(n)	O(1)	O(m+n^2)
方法二: tree法 +Weighted Union	O(log n)	O(1)	O(m log n)
方法二: tree法 +Weighted Union+Path Compression	O(log n)	O(1)	O($m\ \alpha(n)$)≈ $O(m)$ $\alpha(n)$是一個長得很慢的function Ackermann's function的反函式，大部分情形$\alpha(n)\ \leq 4$ (Cormen 21.4)

Related Course Book Chapter

- Cormen chapter 21