樹的最後一部分

Michael Tsai

2010/11/05

下週五期中考

作業評分有問題的話,
請務必找我or助教

記得帶自己的A4小抄一張
(可抄寫雙面)

今日菜單

- 計算邏輯運算式
- 分離集合的表示方法
- 三個問題, 一樣解答
- 複習時間

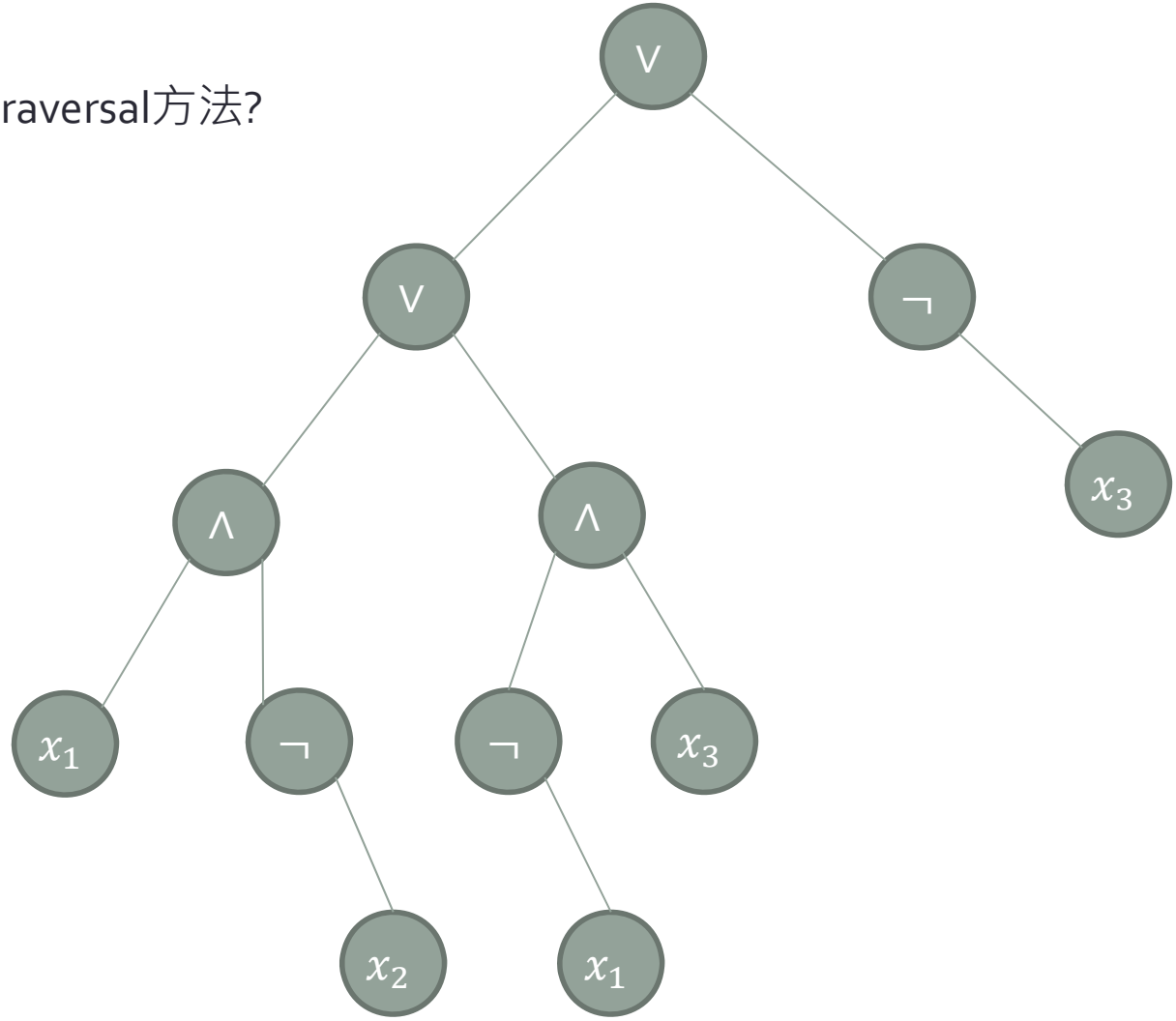
計算邏輯運算式

- 變數可為true or false
- 三種operator: $\neg\ (and)$,$\wedge\ (or)$,$\vee\ (and)$
- 可以使用括號
- 例如 $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3) \vee \neg x_3$
- 如何計算當$(x_1, x_2, x_3) = (t, t, f)$時的結果?
- 進階題: 如何找出所有組合使得結果為true?

計算邏輯運算式

$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3) \vee \neg x_3$

用什麼traversal方法?



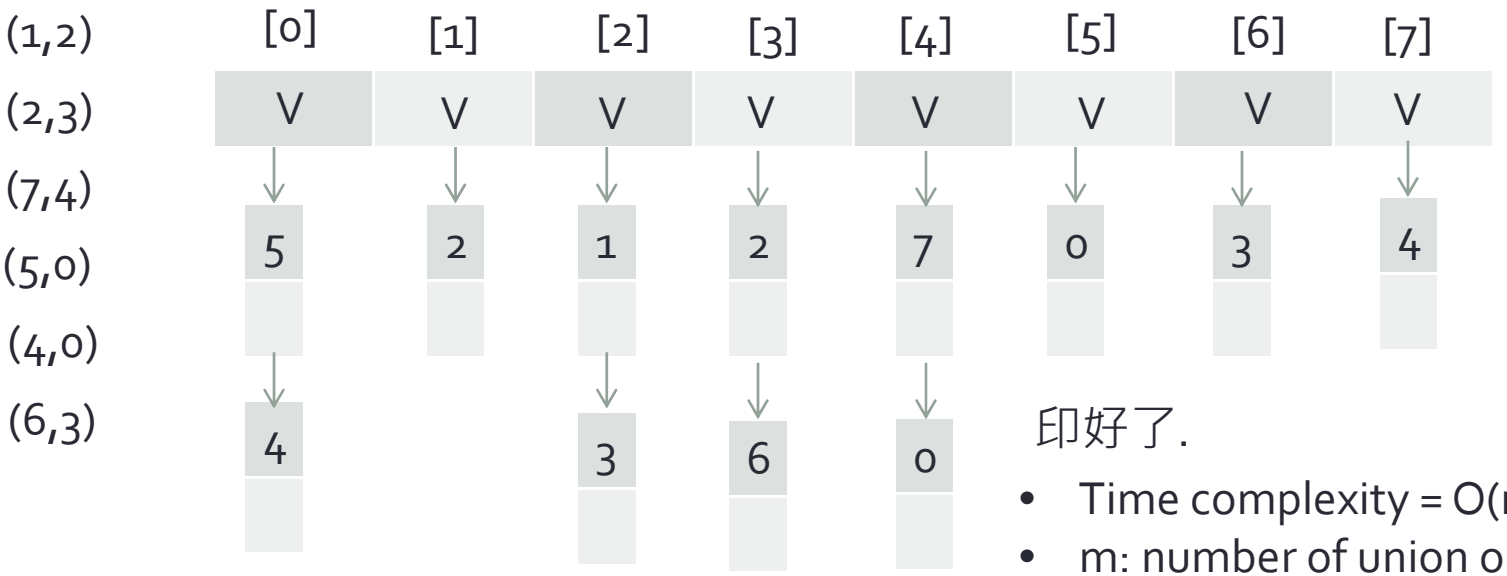
分離集合的表示方法
Representation of disjoint sets

- $S = \{S_1, S_2, \ldots, S_k\}$
- $S_i = \{x_1, x_2, \ldots\}$
- if $i \neq j$, then no elements can be in both S_i and S_j. (disjoint)

- Operations:
- Union(S_i, S_j): create a new set with all elements in S_i and S_j. (the original sets no longer exist)
- Find(x): find the set containing the element x.

要怎麼表示集合呢?

之前講Linked List的時候的方法:



0 5 4 7

1 2 3 6

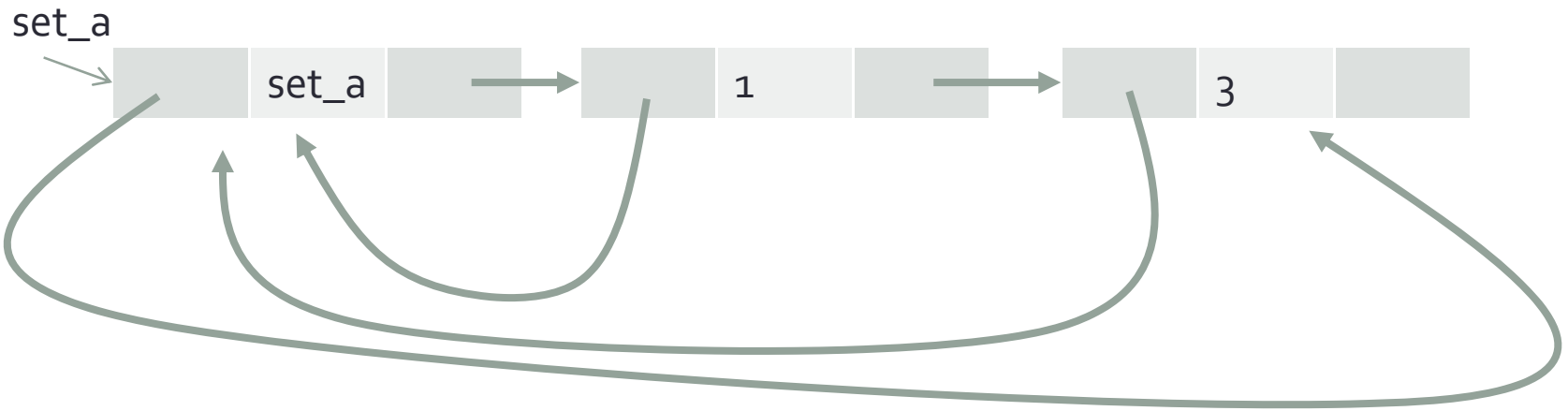
5 4 0 7 0 4

2 1 3 2 6 3

印好了.

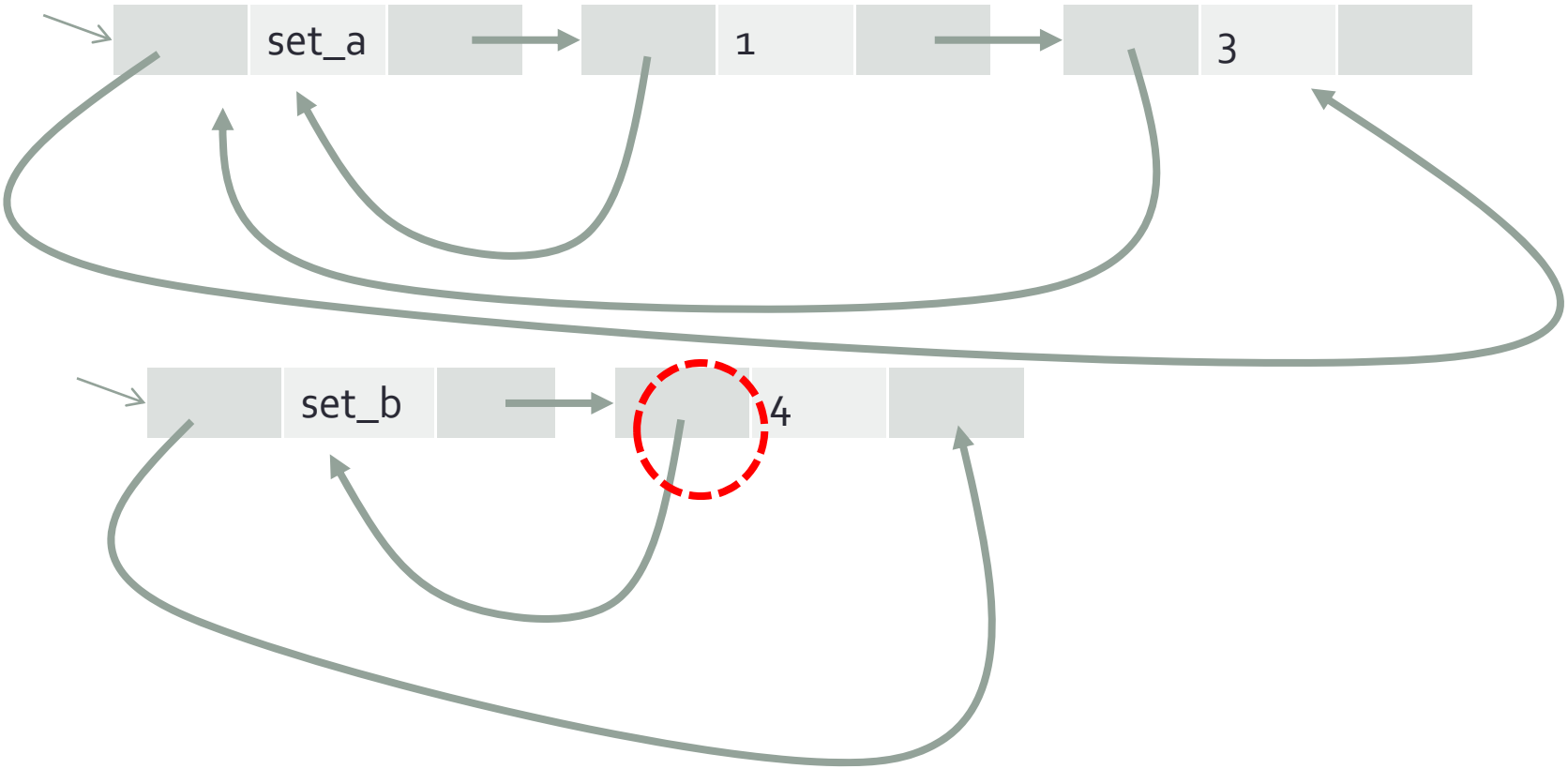
- Time complexity = O(m+n)
- m: number of union operations
- n: number of numbers in the sets
- 缺點:
- 無法直接找到2個element是不是同一個set

改良一下?



- m: number of operations
- n: number of elements
- 怎麼找某element是哪一個set?
- time complexity = O(??)
- 怎麼替兩個set做union? 誰加在誰裡面?
- m個operation的time complexity =O(??)

Weighted union



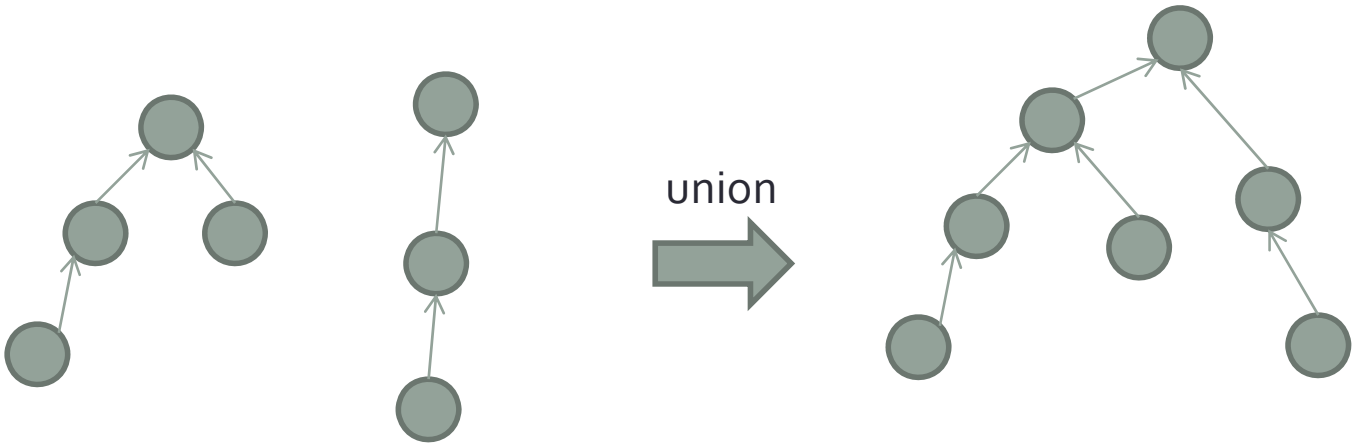
- 因為加到別人的set要改所有node指到頭頭的link
- 所以應該把比較少成員的set加到比較多成員的set
- m個operation的time complexity = O(??)

Weighted union

- 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
- 改$\log k$次→表示所屬的set至少有k個element
- set有n個element的話→最多改$\log n$次
- 最多有n個set, 所以不可能改超過$n \log n$ 次→O($n \log n$)
- find(x)只要O(1)
- 所以m個operation time complexity為O($\mathrm{m} + n \log n$)

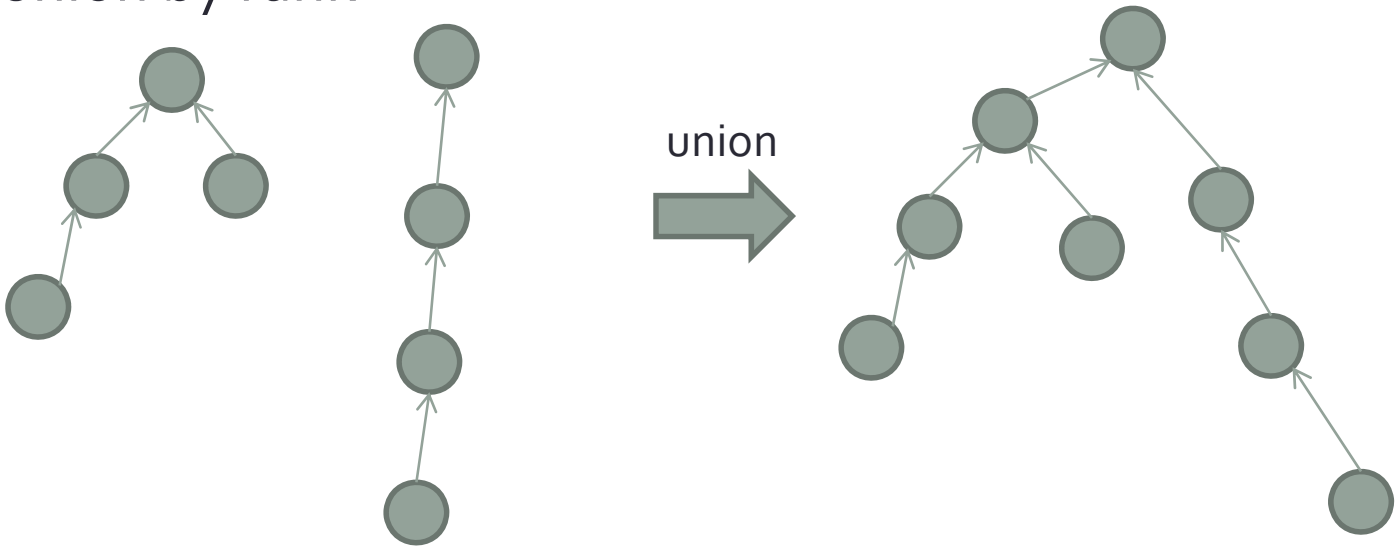
能不能更快?

- 用樹
- disjoint sets → disjoint-set forests
- 怎麼union?
- time complexity = O(??)
- 那怎麼find?
- time complexity = O(??)



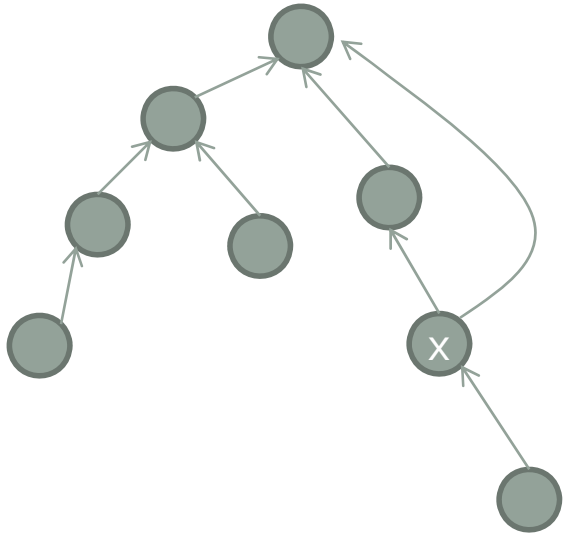
同樣的問題

- 誰加入誰?
- 類似的解決方法: 比大小
- 小的加入大的
- 怎麼知道誰大誰小?
- union的時候記住. (怎麼算union之後的大小?)
- 本方法叫做union by rank



加強版

- find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link都改指到root
- complexity還是一樣, constant變大而已
- 下一次就快得多
- 本方法叫做path compression



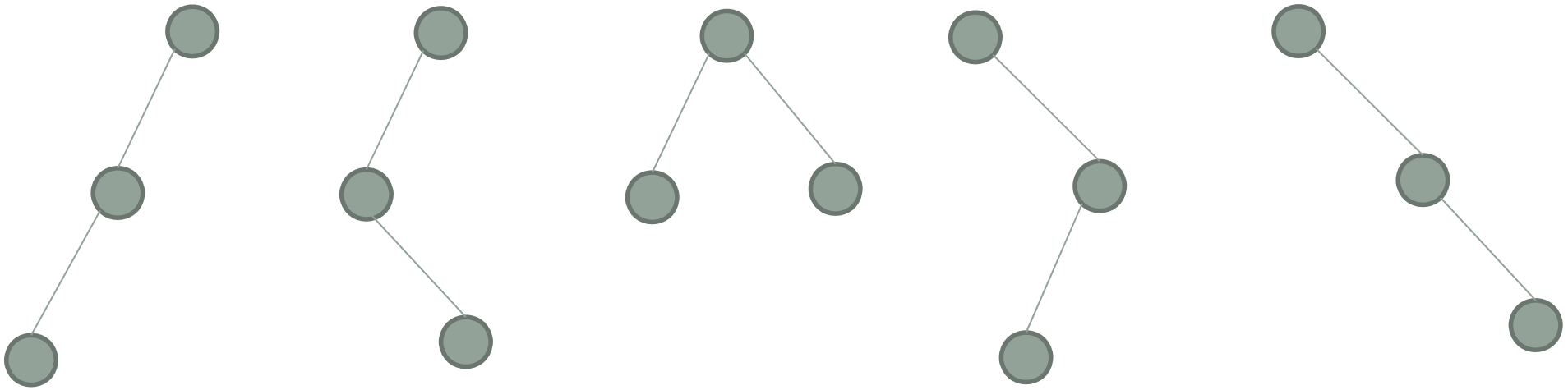
Time complexity?

- 當有m個operation的時候
- time complexity為O($m\ \alpha(n)$)
- $\alpha(n)$是一個長得很慢的function
- Ackermann's function的反函式 (textbook p.255)
- for all reasonable n and m,$\alpha(n) \leq 4$
- 因此可以看成是O(m) → linear
- !!

三個問題，一樣解答

- 問題一
- n個node的binary tree, 總共有幾種?

- 例如: n=3的話有五種



三個問題，一樣解答

- 問題二
- 矩陣乘法: $M_1 * M_2 * \cdots * M_n$
- matrix乘法是associative →意思就是:
- $(M_1 * M_2) * M_3 = M_1 * (M_2 * M_3)$
- 問: 總共有幾種乘法?
- 例如: n=4的時候, 有五種
- $((M_1 * M_2) * M_3) * M_4$
- $(M_1 * (M_2 * M_3)) * M_4$
- $(M_1 * M_2) * (M_3 * M_4)$
- $M_1 * ((M_2 * M_3) * M_4)$
- $M_1 * (M_2 * (M_3 * M_4))$

三個問題，一樣解答

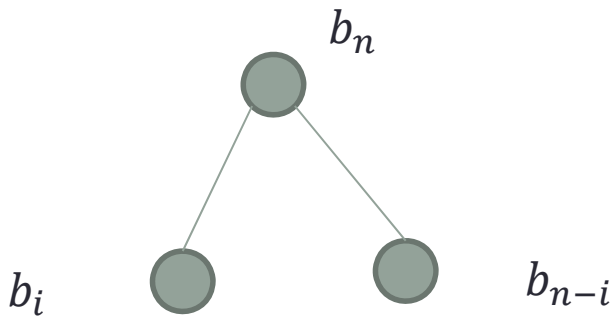
- 問題三
- 使用一個stack, 把1到n的整數照順序push進去, 但是中間可以夾雜任意次數的pop, 請問output有幾種?
- 假設n=3
- 則有五種
- (push 1, pop, push 2, pop, push 3, pop) → 123
- (push 1, push 2, pop, pop, push 3, pop) → 213
- (push 1, pop, push 2, push 3, pop, pop) → 132
- (push 1, push 2, pop, push 3, pop, pop) → 231
- (push 1, push 2, push 3, pop, pop, pop) → 321

來解問題二

- 其他其實是一模一樣的問題(幾乎)
- 最強的方法: recursive!
- 其實總共的數目, 會等於
- $(M_1) * (M_2 * \cdots * M_n)$
  1個矩陣可以排的數目* n-1個矩陣可以排的數目
- $(M_1 * M_2) * (M_3 * \cdots * M_n)$
  2個矩陣可以排的數目* n-2個矩陣可以排的數目
- …
- $(M_1 * \cdots * M_{n-1}) * (M_n)$
  n-1個矩陣可以排的數目*1個矩陣可以排的數目
- 通通加起來
- $b_n = \sum_{i=1}^{n-1} b_i b_{n-i}, n > 1, b_0 = 1$ (recursive)

來解問題一

- $b_n = \sum_{i=0}^{n-1} b_i b_{n-i}, n > 1, b_0 = 1$ (recursive)



最後的推倒(誤)

- $b_n = \sum_{i=0}^{n-1} b_i b_{n-i}, n > 1, b_0 = 1$ (recursive)
- $B(x) = \sum_{i \geq 0} b_i x^i$
- $xB^2(x) = B(x) - 1$
- $B(0) = 1$
- $B(x) = \frac{1-\sqrt{1-4x}}{2x} = \frac{1}{2x}\left(1 - \sum_{n \geq 0}\binom{1/2}{n}(-4x)^n\right)$

$$= \sum_{m \geq 0} \binom{\frac{1}{2}}{m+1} (-1)^m 2^{2m+1} x^m$$

- So, $b_n = \sum_{n \geq 0} \binom{\frac{1}{2}}{n+1} (-1)^n 2^{2n+1}$

複習時間

- 各位想要複習什麼?
- 我覺得重要的:
  - Big-oh, Sigma, and Theta
  - Failure function and KMP algorithm
  - Prefix, infix, and postfix expression evaluation/conversion
  - Binary search tree
  - Heap (priority queue)
  - Stack & Queue