PRIORITY QUEUES

Michael Tsai

2010/12/28

Outline

- Dynamic hashing
- Priority Queue的種類
- DEPQ的用途
- Leftist Tree
- Binomial Heaps

Dynamic hashing

- 觀察: 當n/b比較大以後, O(1)就開始崩壞 (往O(n)方向移動)

- 應變: 所以要隨時觀察 n/b, 當它大過某一個threshold時就把hash table變大

- 觀察: 把hash table變大的時候,
- 需要把小hash table的東西通通倒出來,
- 算出每一個pair在大hash table的位置
- 然後重新放進大hash table

- 有個可憐鬼做insert正好碰到應該hash table rebuild的時候, 他就會等非常非常久. T_T

Dynamic hashing

- 目標: 重建的時候, 不要一次把所以重建的事情都做完
- 或許, 留一些之後慢慢做?
- 每個operation的時間都要合理

- 又叫做extendible hashing

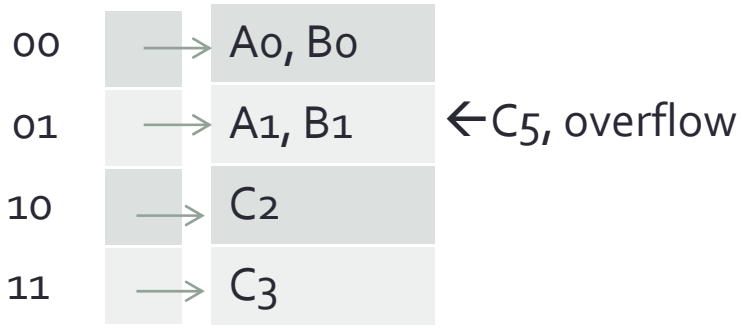
例子

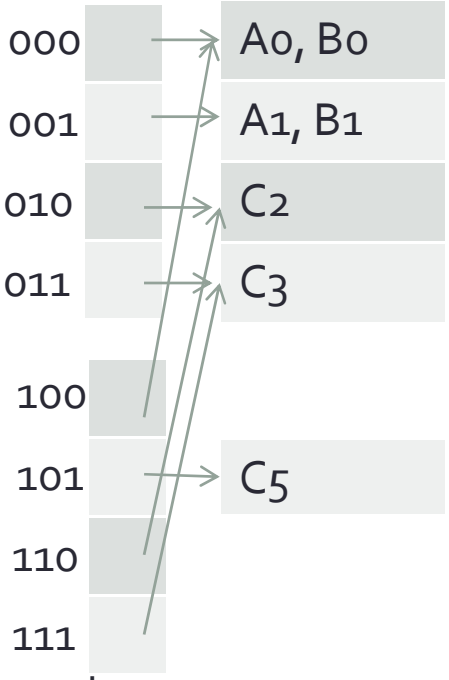
k	h(k)
A0	100 000
A1	100 001
B0	101 000
B1	101 001
C1	110 001
C2	110 010
C3	110 011
C5	110 101

h(k,i)=bits 0-i of h(k)
Example:
h(A0,1)=0
h(A1,3)=001=1
h(B1,4)=1001=9

Dynamic hashing using directories

directory depth=
number of bits of the index of the hash table





Insert C_5

$h(C_5, 2)=01=1$

we increase d by 1
until not all h(k,d) of the keys in the cell are the same

k	h(k)
A0	100 000
A1	100 001
B0	101 000
B1	101 001
C1	110 001
C2	110 010
C3	110 011
C5	110 101

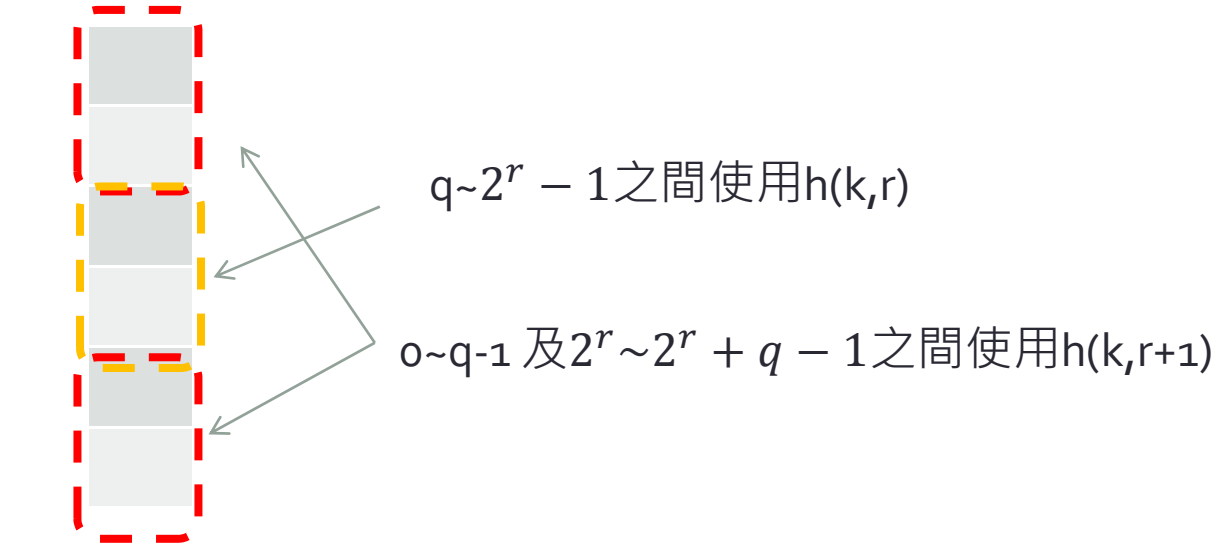
動腦時間:
如果原本的要加入C1呢?
如果第二步驟後加入A4呢?答案: p. 412-413

Dynamic hashing using directories

- 為什麼比較快?
- 只需要處理overflow的櫃子
- 如果把directory放在記憶體, 而櫃子資料放在硬碟
- 則
- search只需要讀一次硬碟
- insert最多需要讀一次硬碟(讀資料, 發現overflow了), 寫兩次硬碟(寫兩個新的櫃子)
- 當要把hash table變兩倍大時, 不需要碰硬碟(只有改directory)

Directoryless Dynamic hashing

- 假設hash table很大, 但是我們不想一開始就整個開來用(initialization會花很大)
- 用兩個變數來控制的hash table大小: r, q
- hash table開啟的地方為 0, $2^r + q - 1$之間

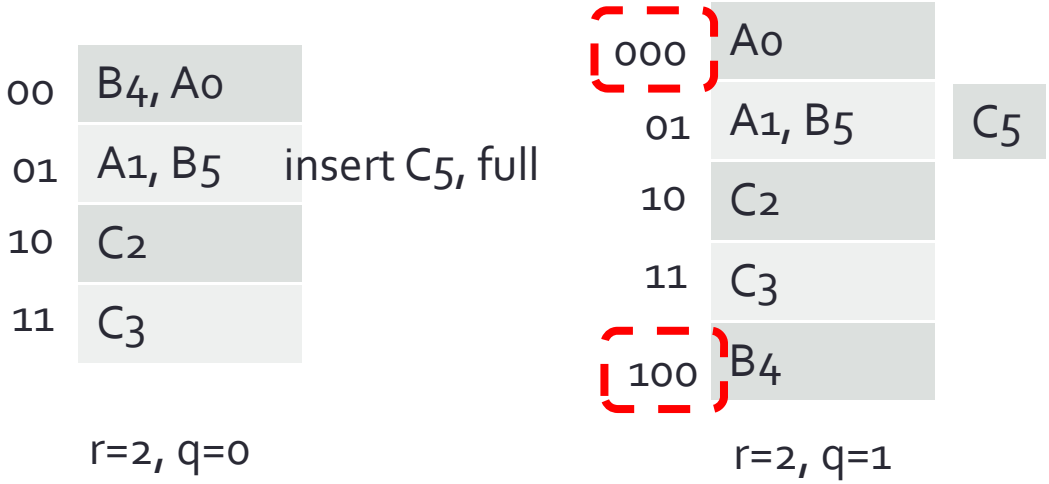


Directoryless Dynamic hashing

- 每次輸入的時候, 如果現在這個櫃子滿了
- 則開一個新的櫃子: $2^r + q$
- 原本q櫃子裡面的東西用
- h(k,r+1)分到q和$2^r + q$兩櫃子裡
- 注意有可能還是沒有解決問題
- 多出來的暫時用chain掛在櫃子下面

問:再加入C1呢? (課本p 415)

k	h(k)
A0	100 000
A1	100 001
B4	101 100
B5	101 101
C1	110 001
C2	110 010
C3	110 011
C5	110 101



複習: Priority queue的operations

- A. 看(找到)queue裡面priority最小(或最大)的element
- B. 插入一個element到queue裡面
- C. 拿掉(delete)queue裡面priority最小(或最大)的element
- 之前用max or min heap
- A=O(1)
- B=O(log n)
- C=O(log n)

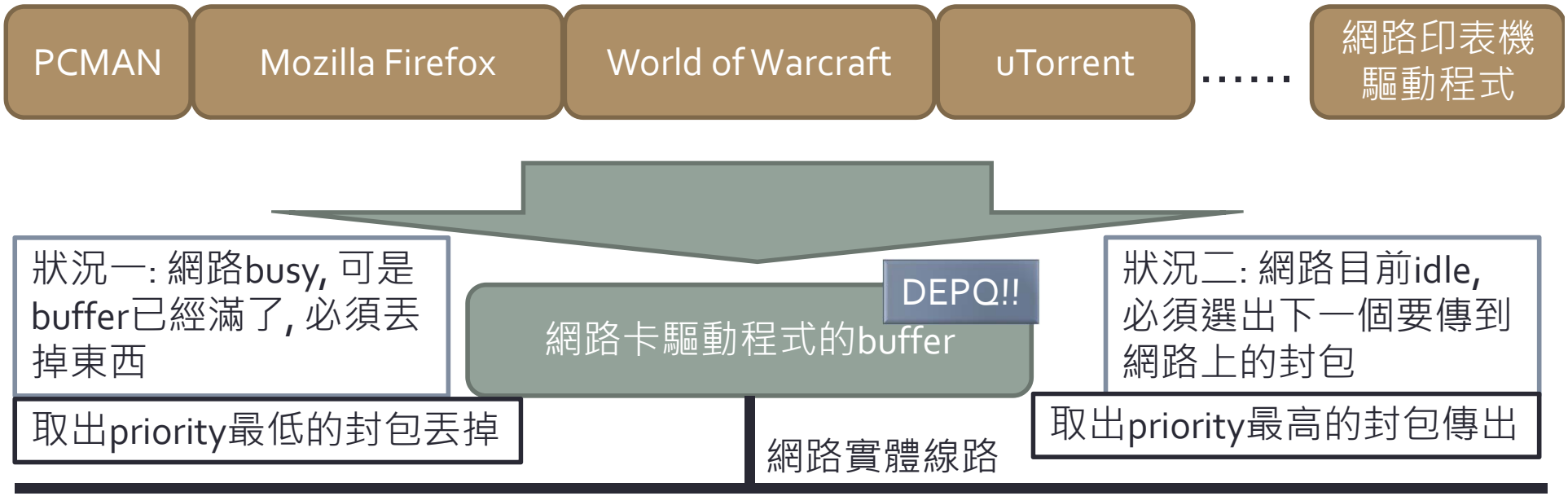
其他priority queue的種類

- Meldable priority queue:
  - 除了原本三個operation
  - 還支援一個operation可以把兩個priority queue合併
- 有什麼用途?

- Double-ended:
  可以拿最大也可以拿最小的 (原本的為single-ended)
- 支援下面的operations
- A. 看(找到)queue裡面priority最小的element
- B. 看(找到)queue裡面priority最大的element
- C. 插入一個element到queue裡面
- D. 拿掉(delete)queue裡面priority最小的element
- E. 拿掉(delete)queue裡面priority最大的element

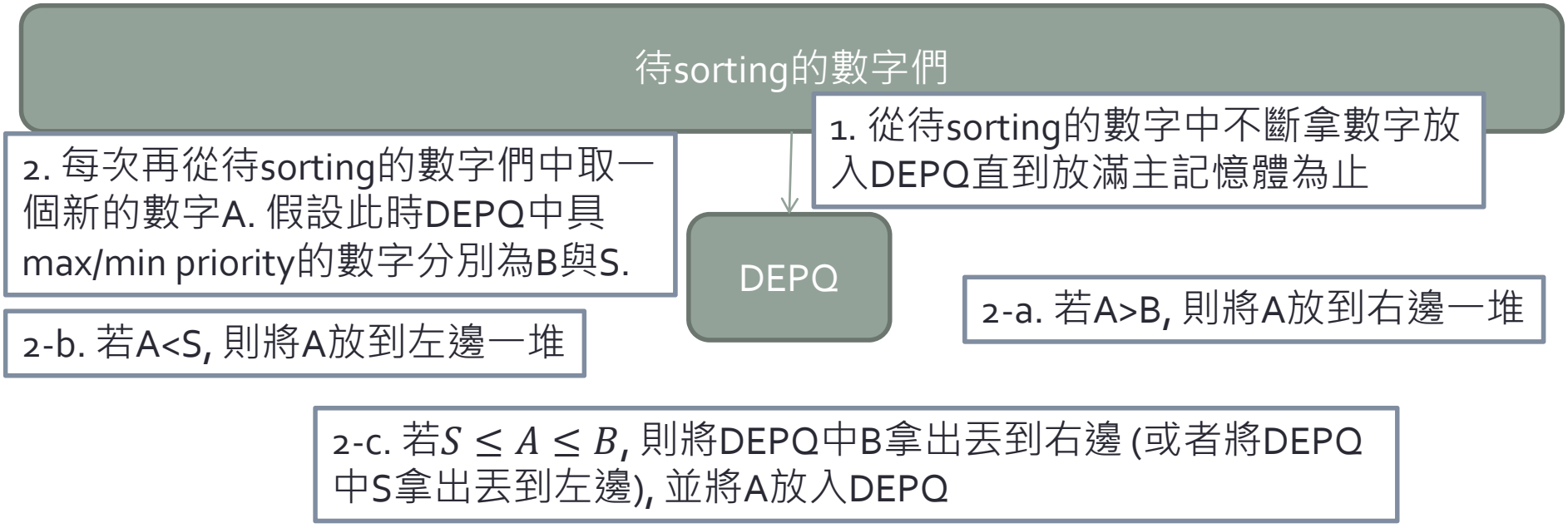
DEPQ的用途(1)

- DEPQ=Double-Ended Priority Queue
- 用來implement network buffer

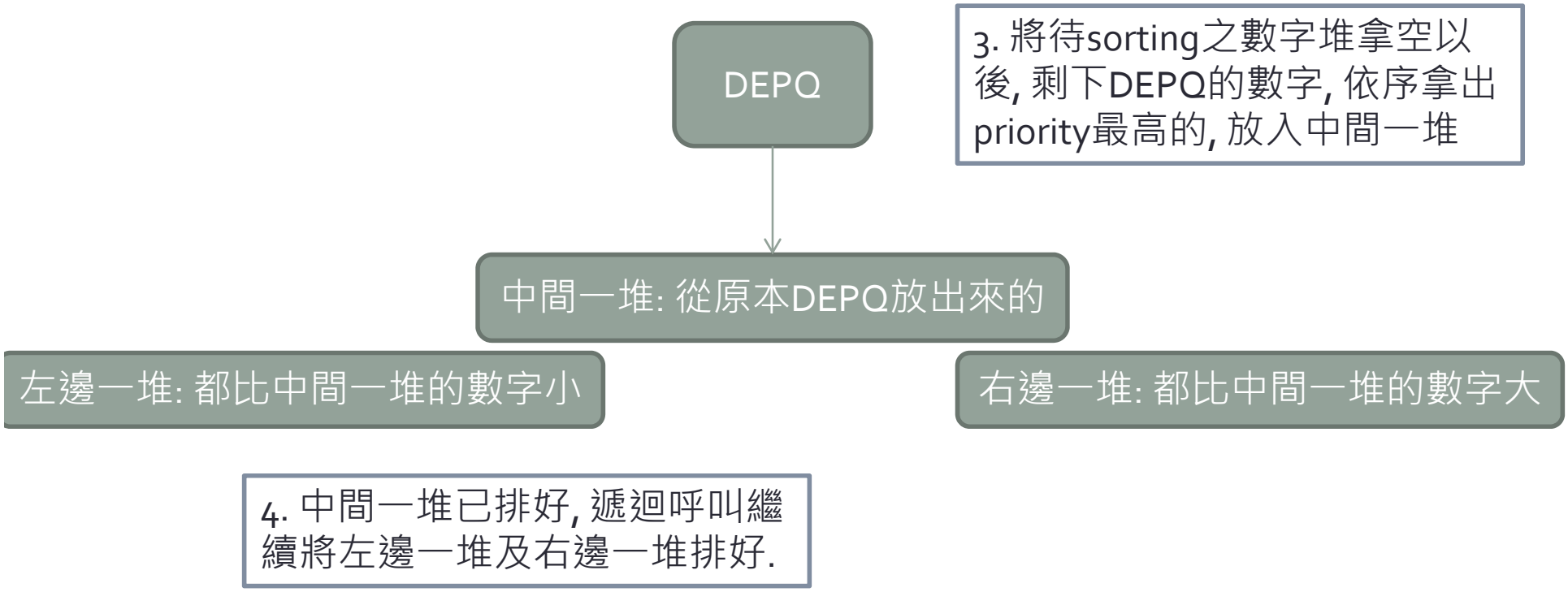


DEPQ的用途(2): External sorting

- External sorting的適用時機:
  無法全部丟入主記憶體(部分要放在慢速儲存媒體)
- 使用DEPQ來implement quick sort的變種



DEPQ的用途(2): External sorting

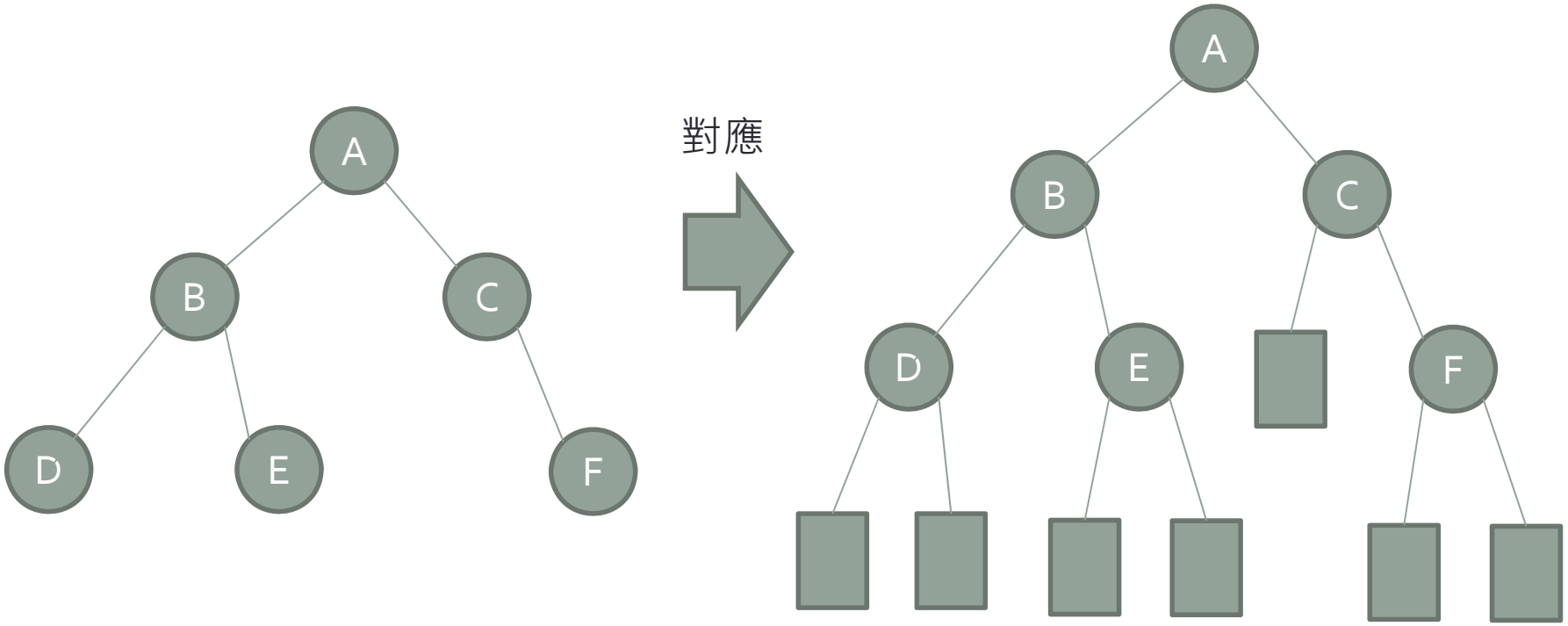


Meldable Priority Queue: Leftist tree

- Heap: 要花多少時間結合兩個heap?
- 把兩個heap都當作沒有排過直接重新建一個大heap
- 用從一個array建成一個heap的方法(課本section 7.6 heapsort) 需要O(n)

- 目標:
- meld, insert, delete=O(log n)
- find=O(1)

Extended Binary Tree



Binary Tree

Extended Binary Tree

: internal nodes

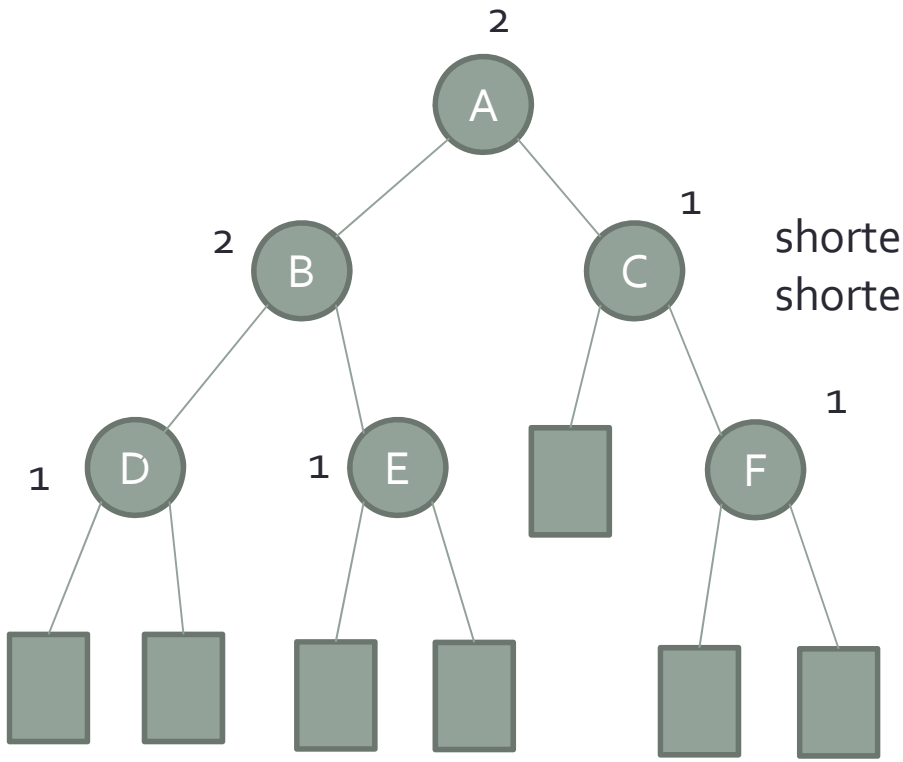
: external nodes

Leftist tree的一些定義

- Shortest(x): 從某個extended tree的node走到任一個external node的距離
- 可以用遞迴定義:
- Shortest(x)=
- a. 0, if x 是external node
- b. 1+ min{shortest(leftChild(x)), shortest(rightChild(x))}, if x 是 internal node

- 定義: A leftist tree is a binary tree such that if it is not empty, then shortest(leftChild(x))≥shortest(rightChild(x)), for every internal node x.

例子: 這是leftist tree嗎?



shortest(leftChild(C))=0
shortest(rightChild(C))=1

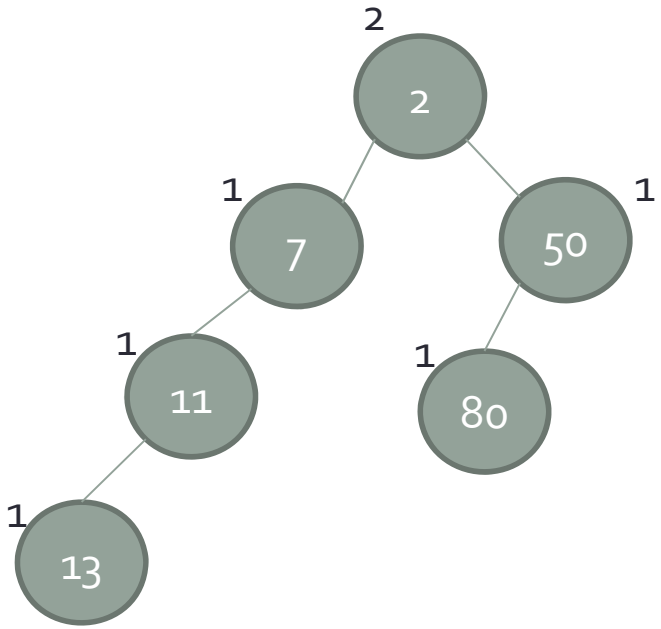
一些性質

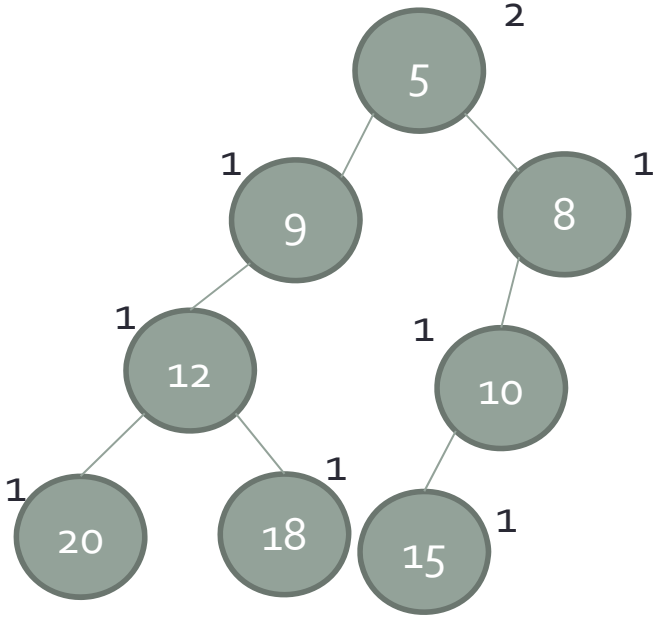
- 假設r為leftist tree的root, n為internal nodes個數
- 1. $n \geq 2^{shortest(r)} - 1$
- 2. 從root到任何一external node的路徑中最右邊的一條, 為最短. 其長度為shortest(r)≤ $\log_2(n+1)$

- 為什麼?
- 1. shortest(r)層內沒有external nodes.
- 2. 由定義得知左邊的路徑一定比右邊的路徑長.

Min leftist tree

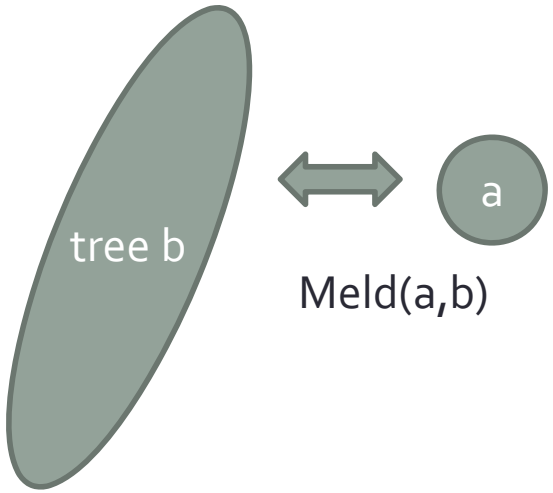
- Min leftist tree: 一個所有某node的key值永遠比它的小孩key值小的leftist tree

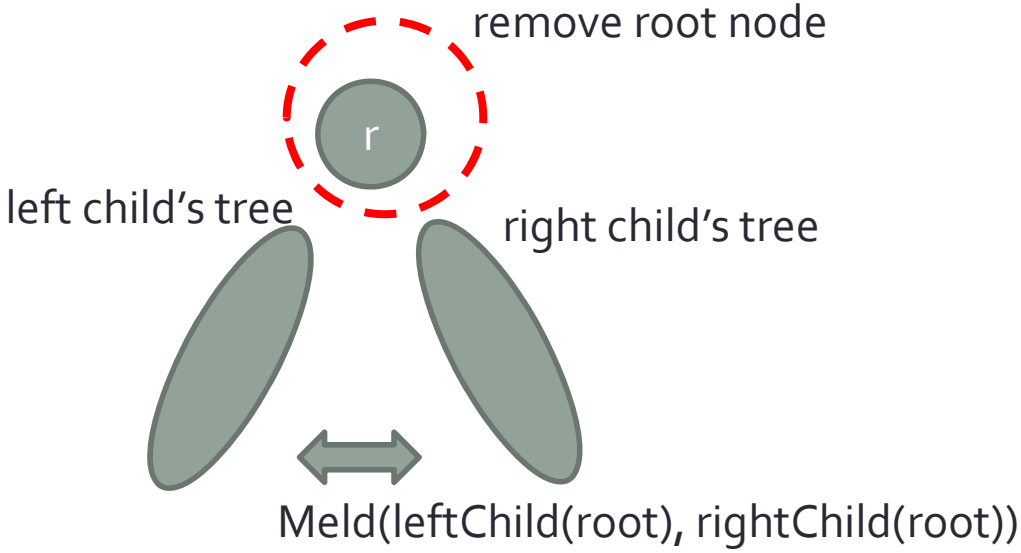




Insert & Delete by Melding

- Insert(a, tree b): 創造一個只有a的min leftist tree, 然後 meld(a,b).

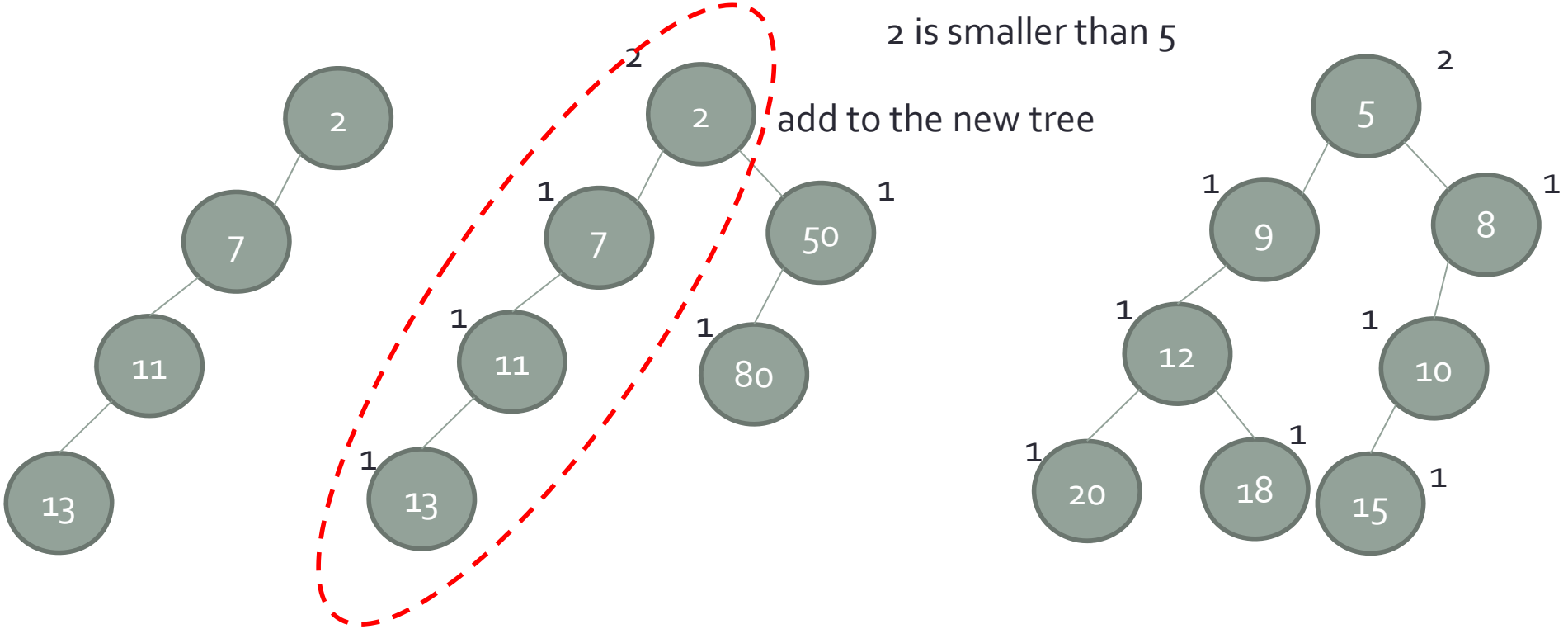




- Delete(tree b): Meld(leftChild(root(b)), rightChild(root(b)))

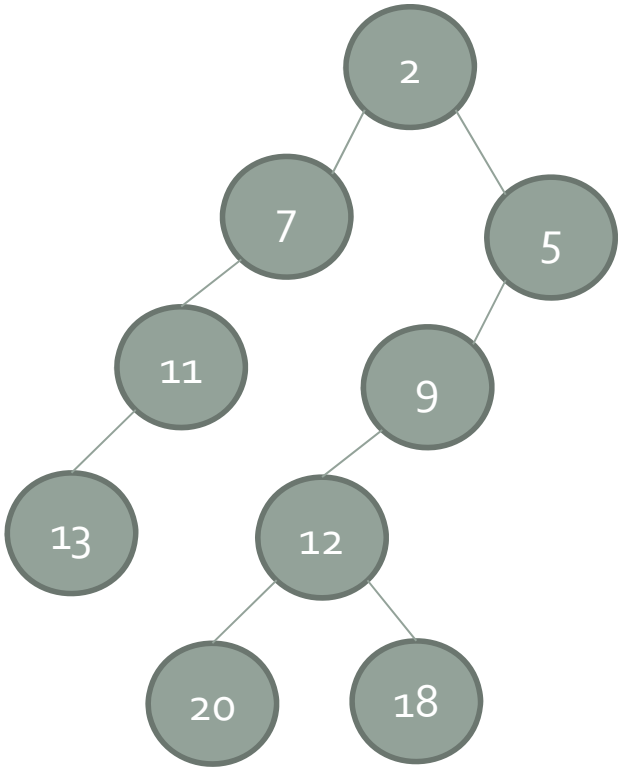
Melding process - Example

Step 1: 把兩棵樹合併並確定parent的key大於children的key



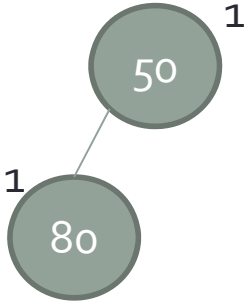
Melding process - Example

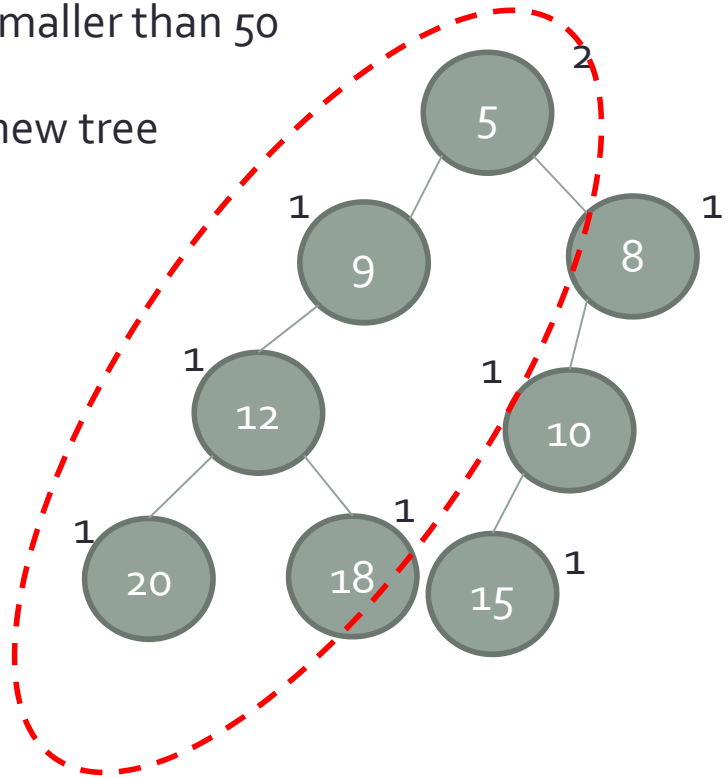
Step 1: 把兩棵樹合併並確定parent的key大於children的key



5 is smaller than 50

add to the new tree



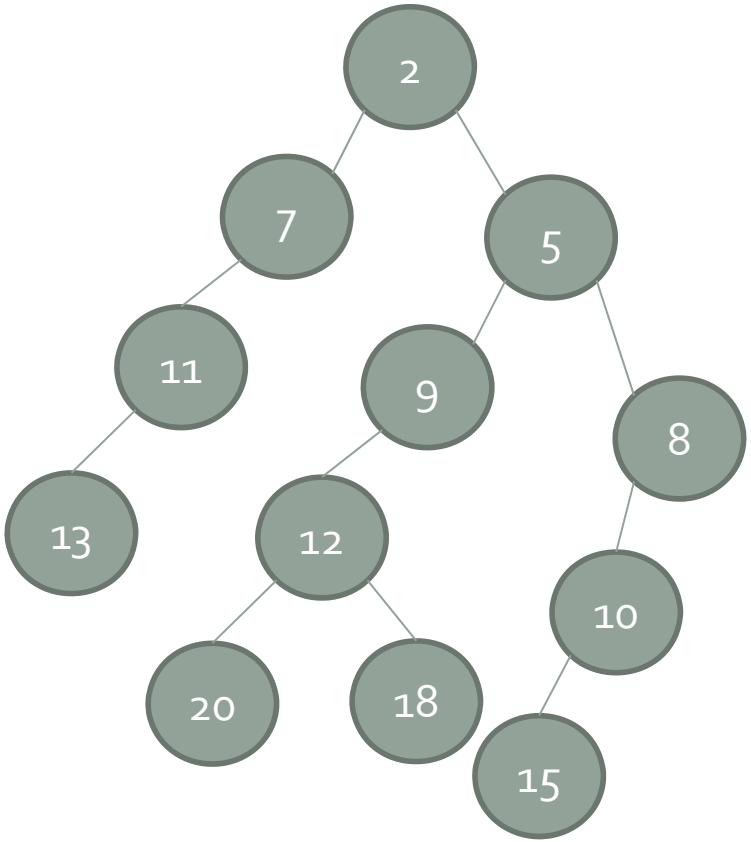


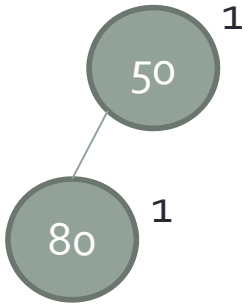
Melding process - Example

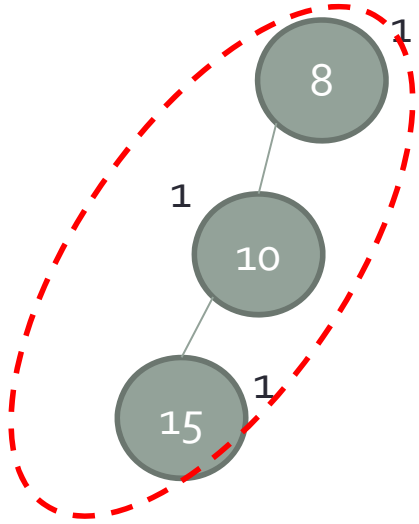
Step 1: 把兩棵樹合併並確定parent的key大於children的key

8 is smaller than 50

add to the new tree

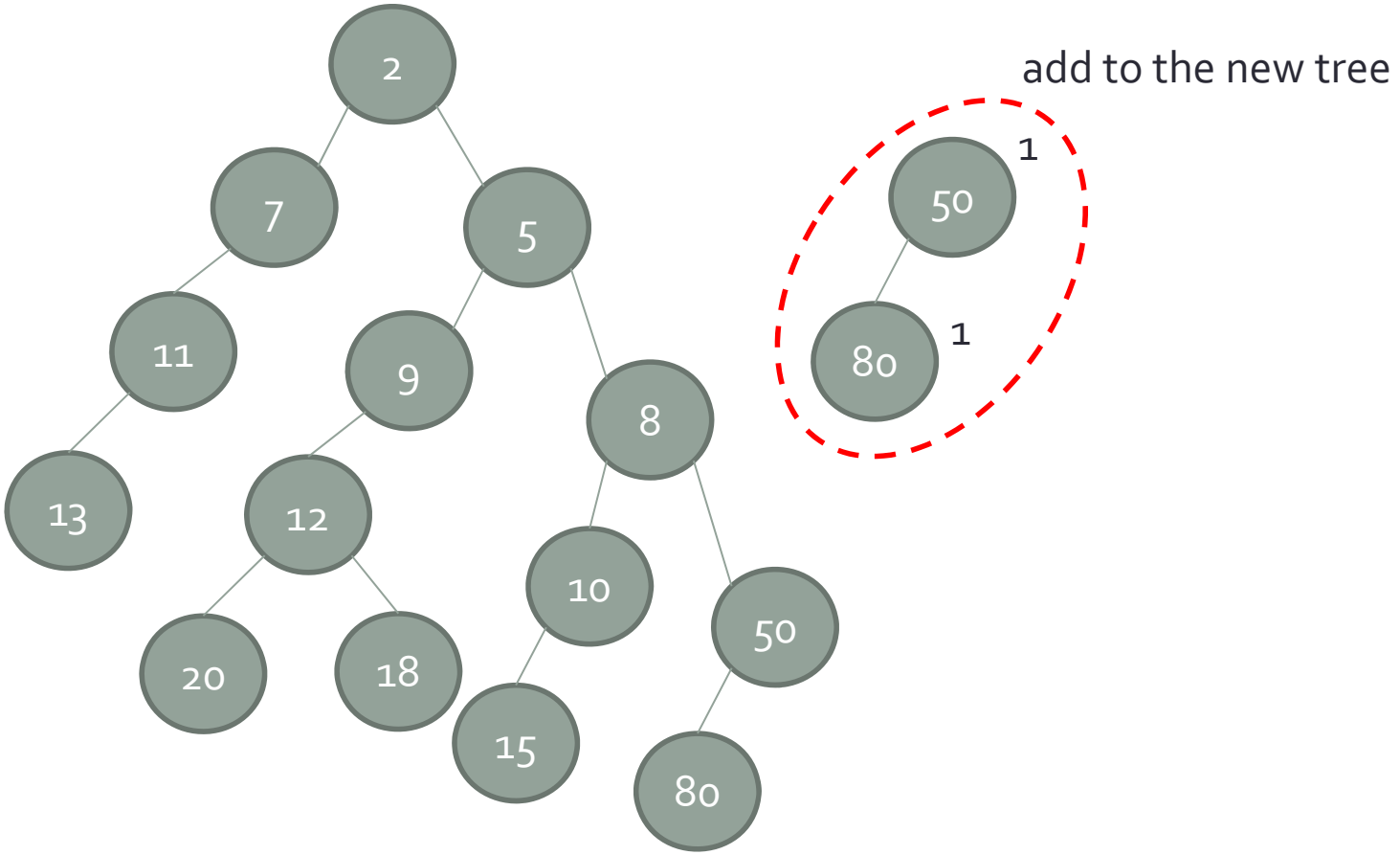






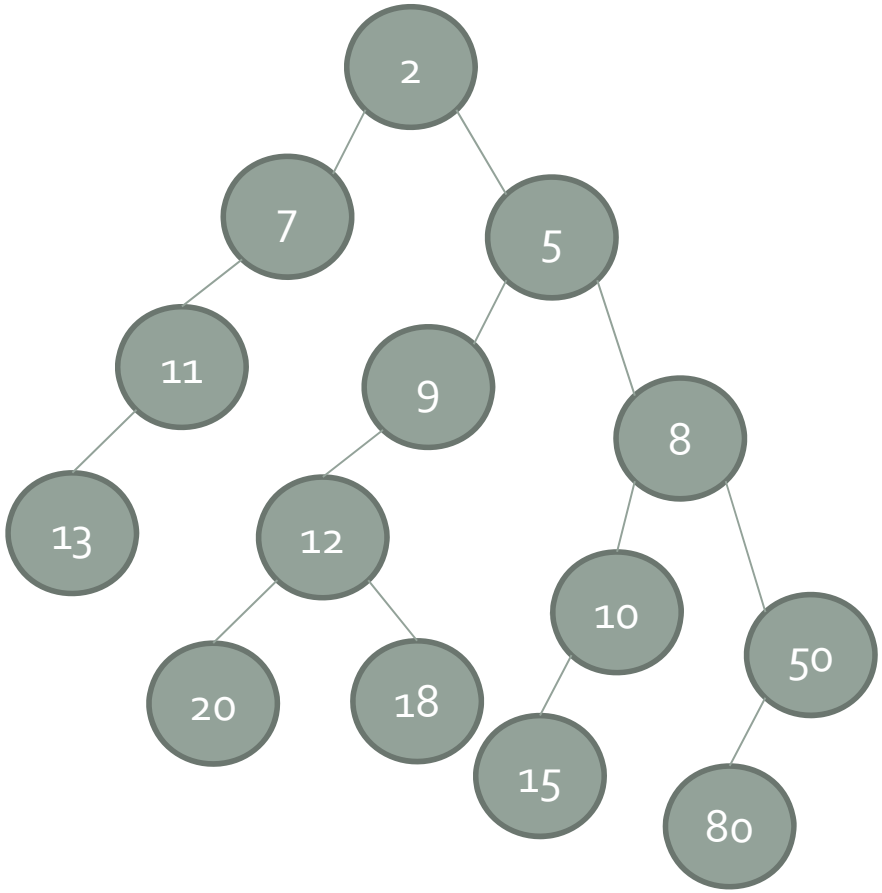
Melding process - Example

Step 1: 把兩棵樹合併並確定parent的key大於children的key



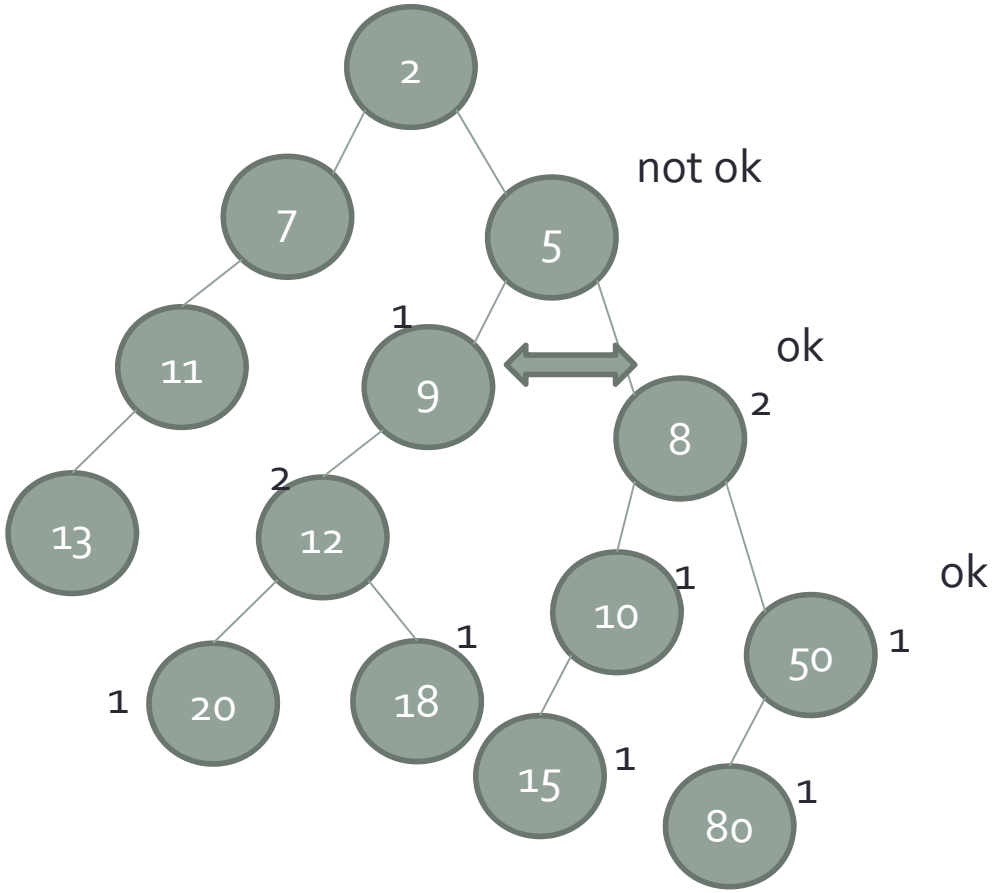
Melding process - Example

Step 1: 把兩棵樹合併並確定parent的key大於children的key



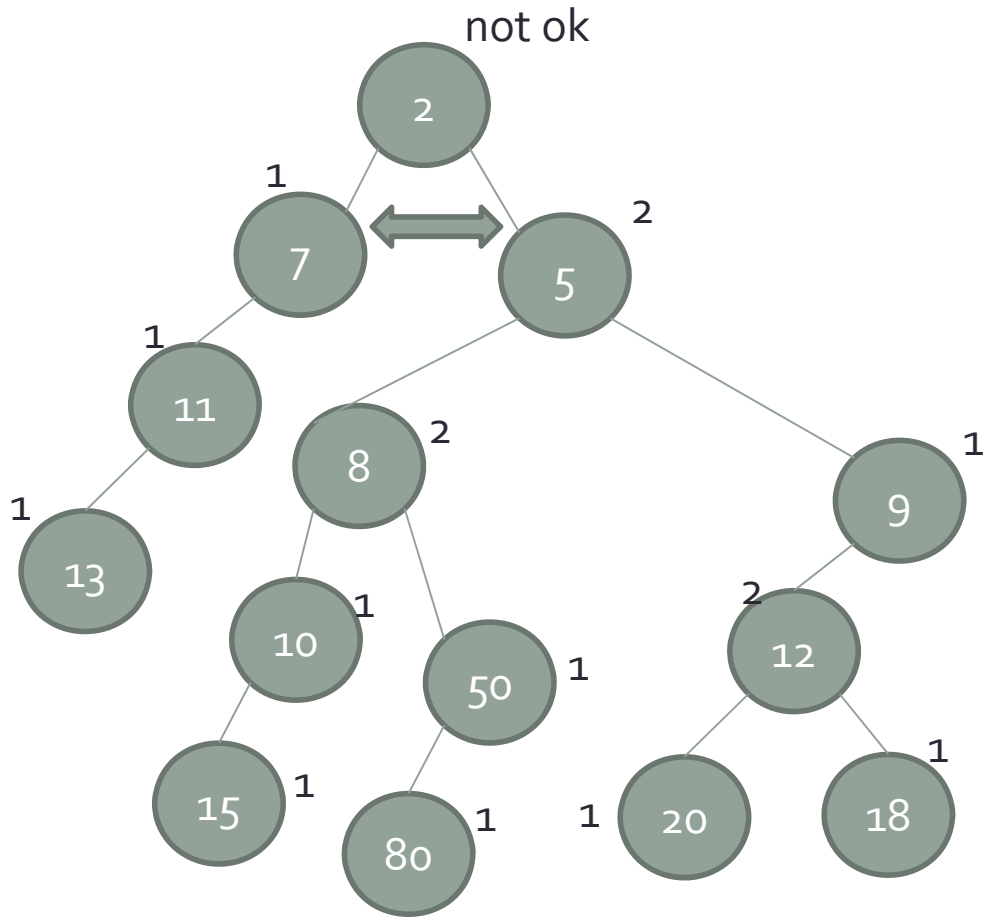
Melding process - Example

Step 2: 沿著剛剛右邊一條路線的root, 確定所有的shortest(left)≥shortest(right)



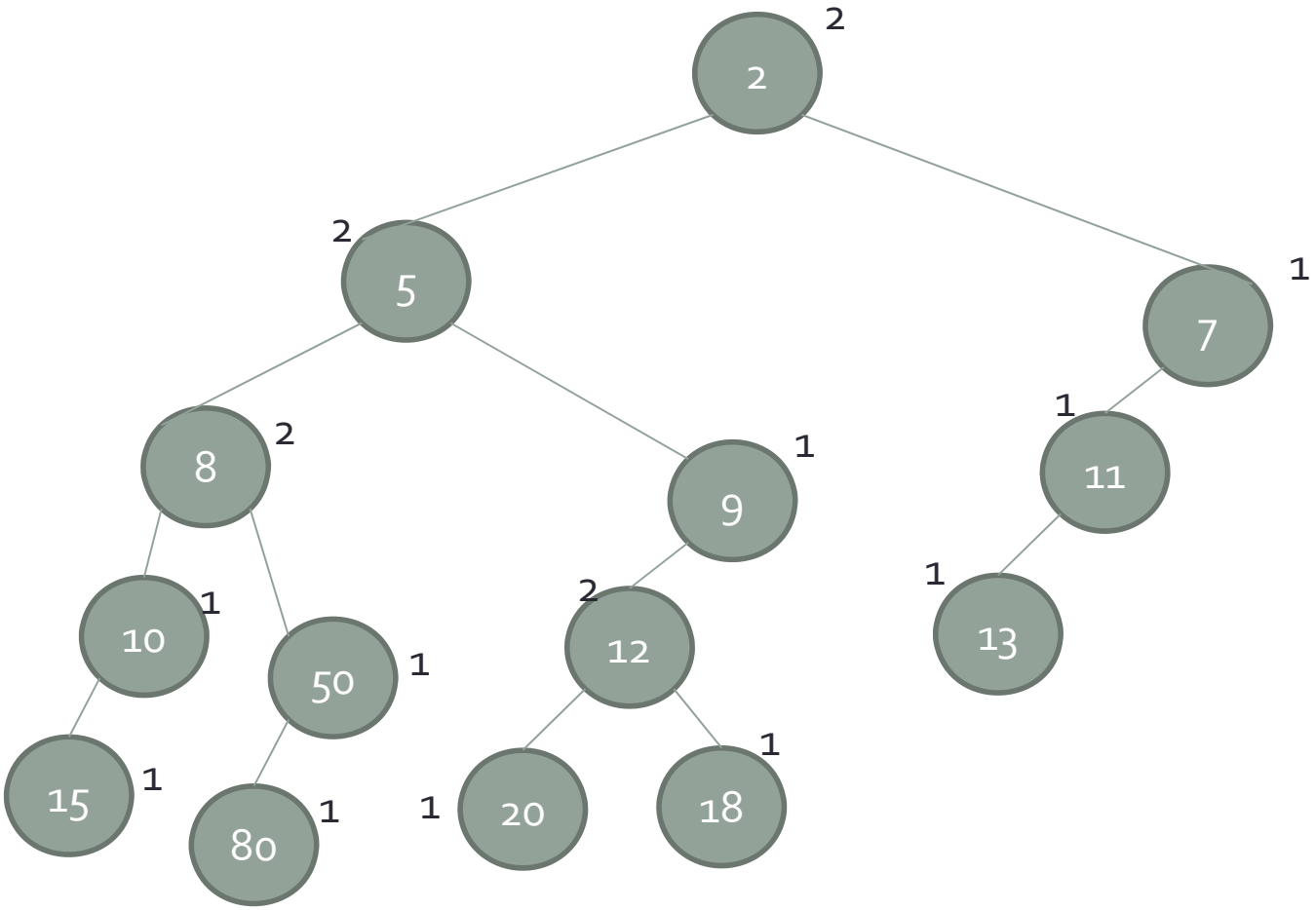
Melding process - Example

Step 2: 沿著剛剛右邊一條路線的root, 確定所有的shortest(left)≥shortest(right)



Melding process - Example

Step 2: 沿著剛剛右邊一條路線的root, 確定所有的shortest(left)≥shortest(right)



那要花多少時間呢?

- Meld operation = O(??)

- 兩個operation都只要花走右邊路徑長度的時間
- 右邊路徑長度都是最短的
- Worst case: balanced binary tree
- 所以所花時間為O(log n)

變形: Weight-based leftist tree

- w(x): node的weight
- w(x)=以x當作root的subtree裡面internal node的數目
- 如果x是external node, 則w(x)=0

- w(x)遞迴定義:
- w(x)=1+w(leftChild(x))+w(rightChild(x))

- Weight-based Leftist Tree定義:
- 對每一個internal node x, w(leftChild(x))≥w(rightChild(x)).
- 其中又可以有Max or min weight-based leftist tree
- (parent key值大於children key值)

Weight-based leftist tree

- 定理: 對weight-based leftist tree的任一internal node x, rightmost(x)為其至任一external node的路徑中最右邊一條的長度. 則rightmost(x)≤ $\log_2(w(x)+1)$
- 證明(使用歸納法)請見課本p.431

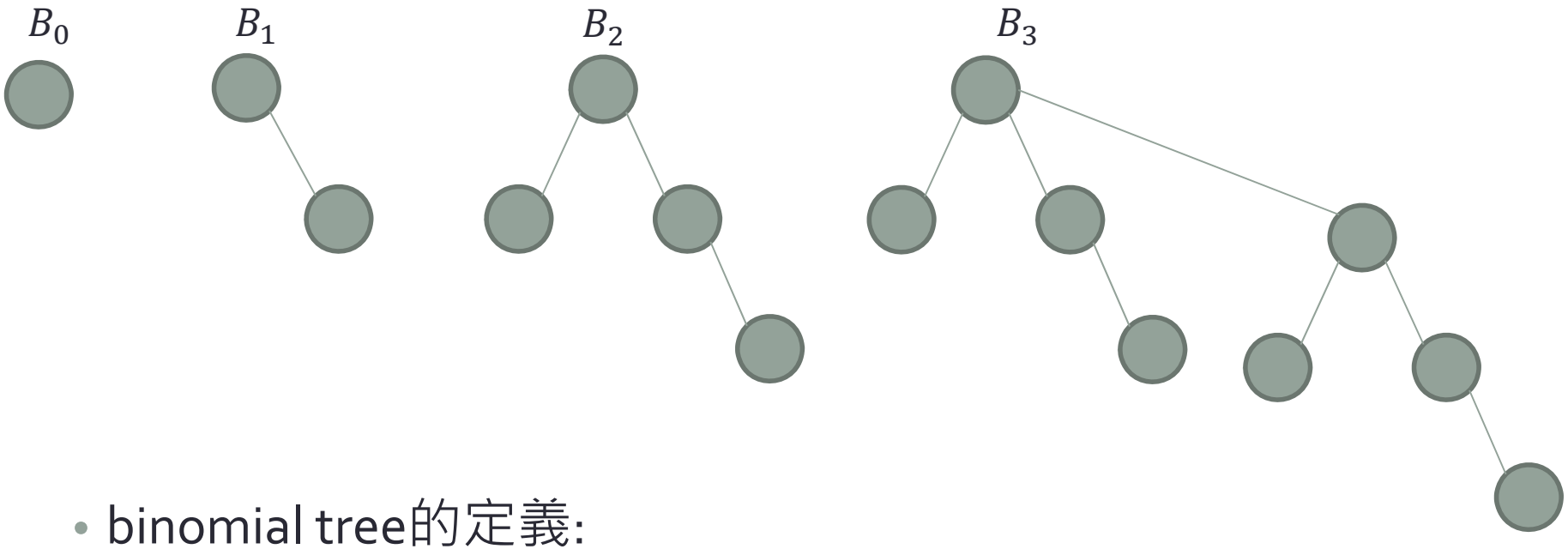
- meld, insert, delete的方法如同一般leftist tree
- 好處: 第二步驟不用再使用獨立一個pass即可完成

B-Heap (Binomial Heap)

- 目標:
- 1. 支援insert, delete (min), and meld的動作
- 2. “平均來說” 單一的operation可能可以到達O(1)或O(log n)

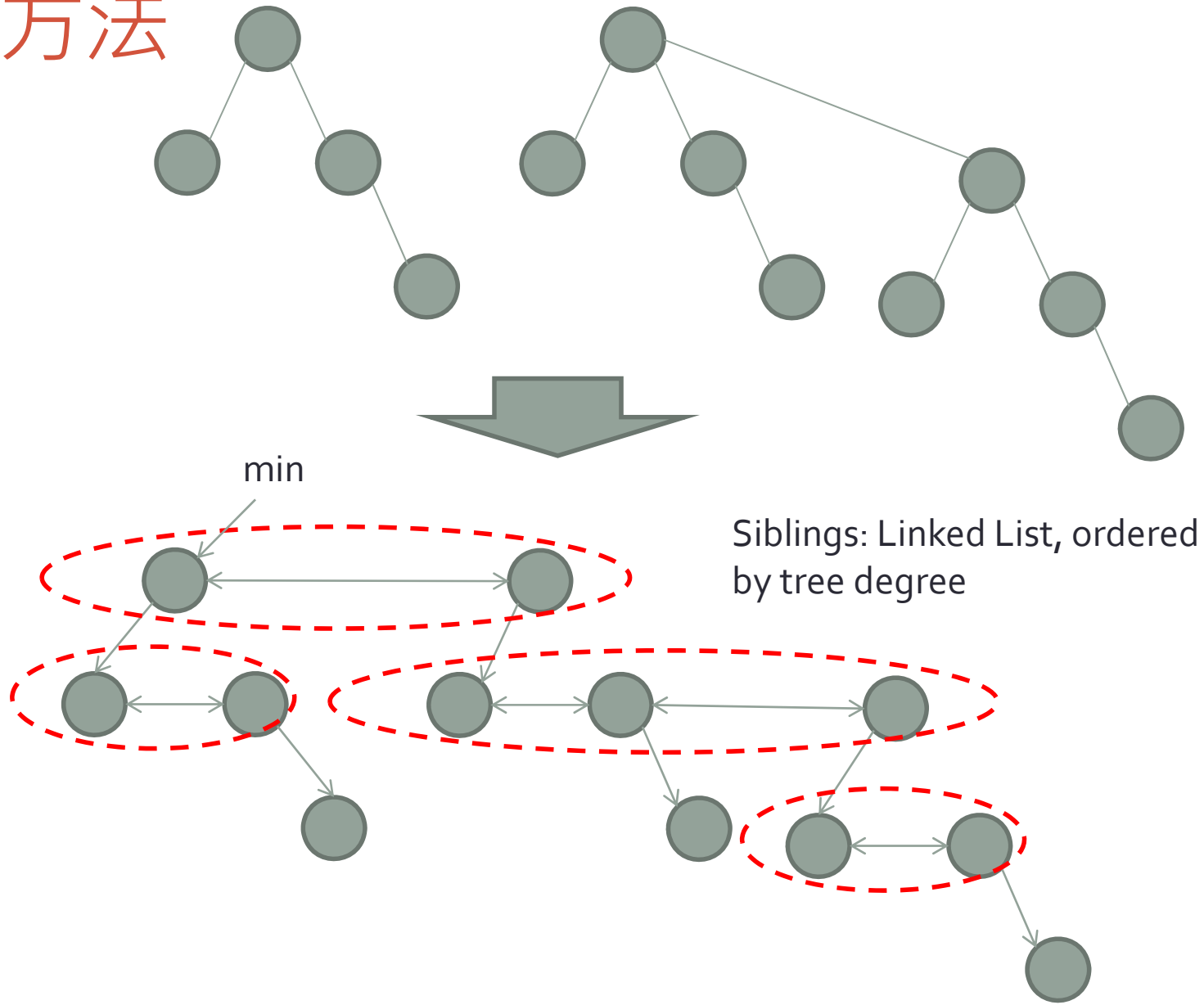
- 跟課本講得不太一樣
- 參考資料:
- http://en.wikipedia.org/wiki/Binomial_heap
- 很讚的動畫applet
- http://www.cse.yorku.ca/~aaw/Sotirios/BinomialHeap.html

Binomial Tree長什麼樣子



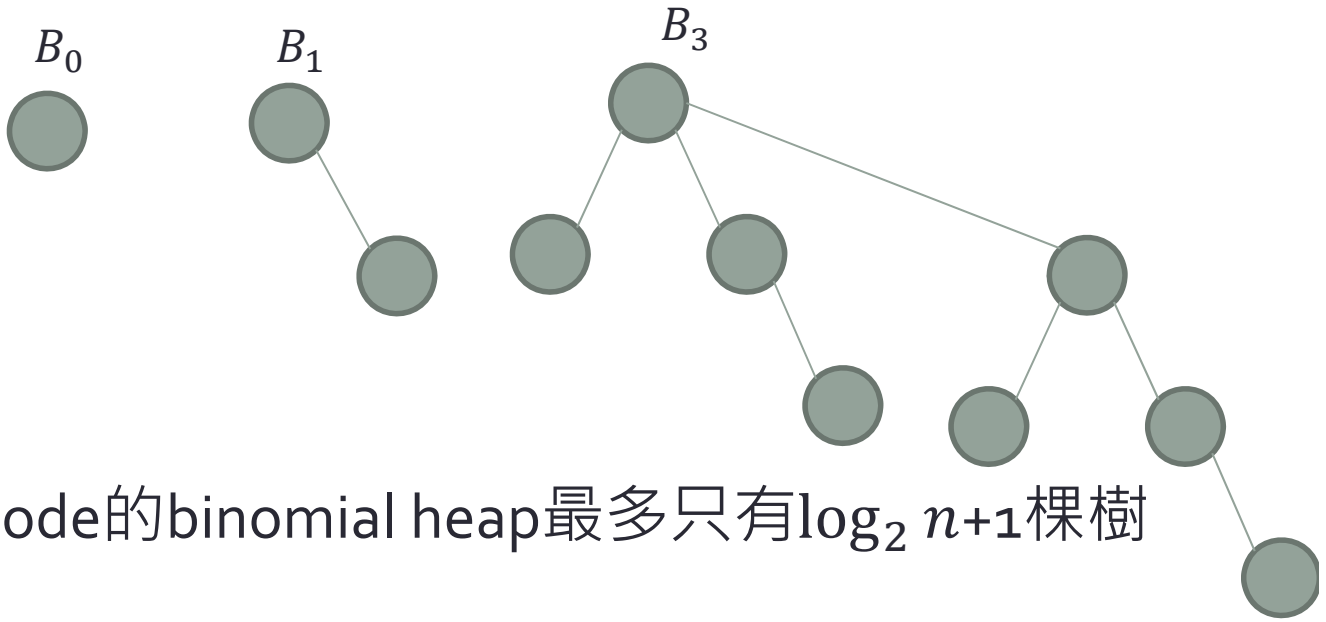
- binomial tree的定義:
- 1. binomial tree of order o →只有單一node
- 2. binomial tree of order k→一個root, 自己的children為order為k-1, k-2, …, o的binomial tree們
- node數目為n, $n = 2^{order}$

表示方法



Binomial Heap長什麼樣子

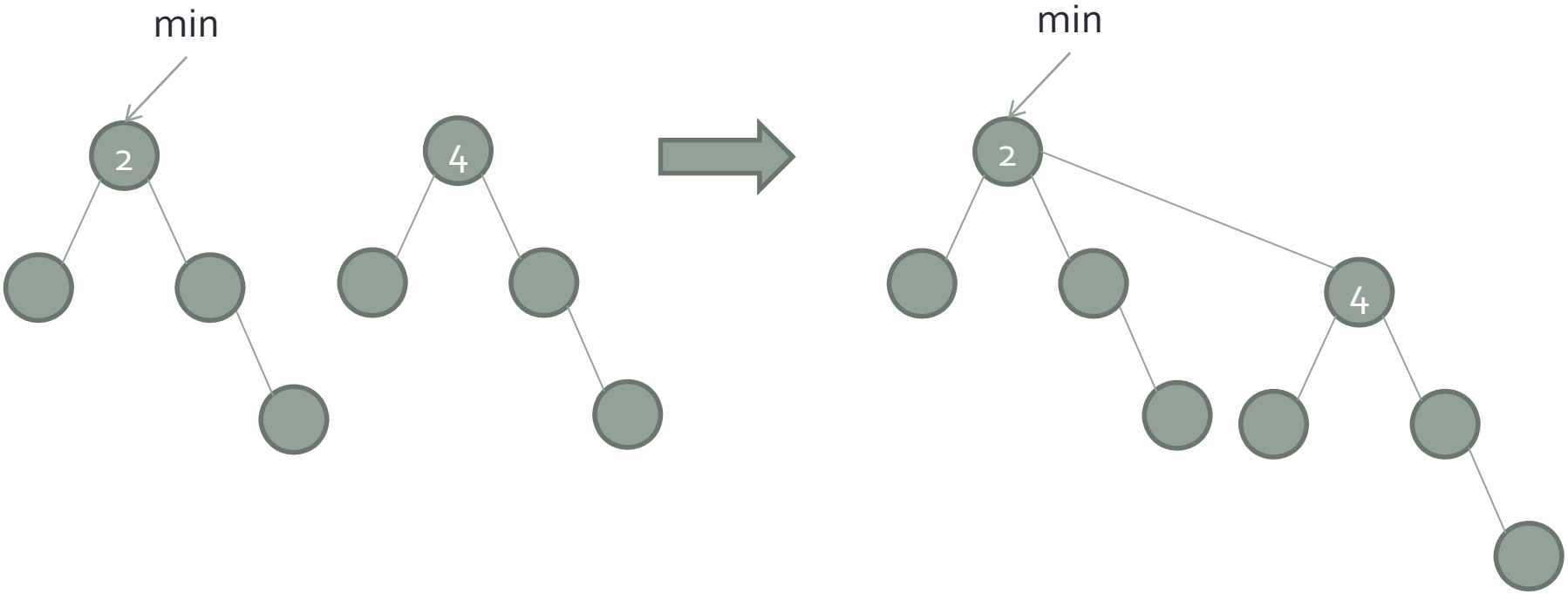
- 是binomial tree組成的forest
- 1. 每個tree的任何一parent node的key大(小)於child node的key
- 2. 每個order的binomial tree只能在這個forest裡面有一顆或零棵



- 條件2. 使得n個node的binomial heap最多只有$\log_2 n+1$棵樹

Merge兩棵order一樣的樹

- 看root key誰比較小
- 小的當新的樹的root
- 大的接在新的樹的root下面



Merge兩個heap (Meld)

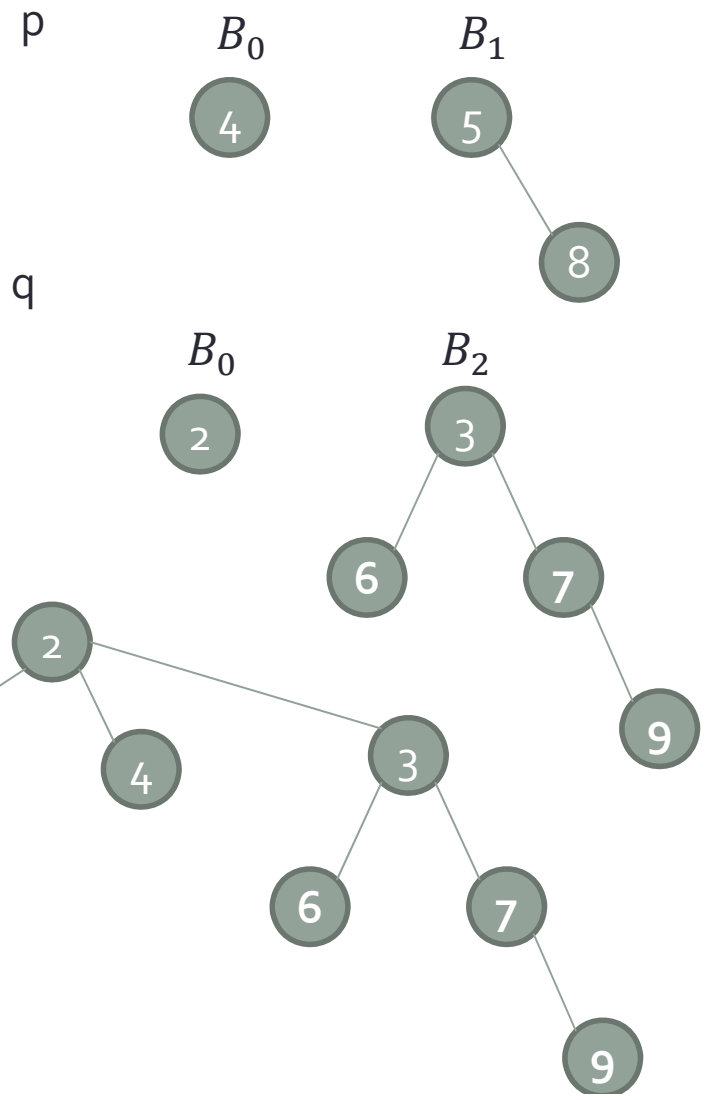
- 假設要merge p和q兩個heap
- p和q各有order 0-K的binomial tree
- 則開始看每個order (假設是i)
- 如果p和q中只有一個有order i的
- 則直接丟到merged的heap
- 否則的話, 就把兩個tree合併
- 合併後的tree, 還有可能跟後面order比較大的tree再合併

- 黑板舉另外一個例子☺
- O(log n)

order 0: 都有

order 1: 只有一個, 但merge過後的也有

order 2: 只有一個, 但merge過後的也有



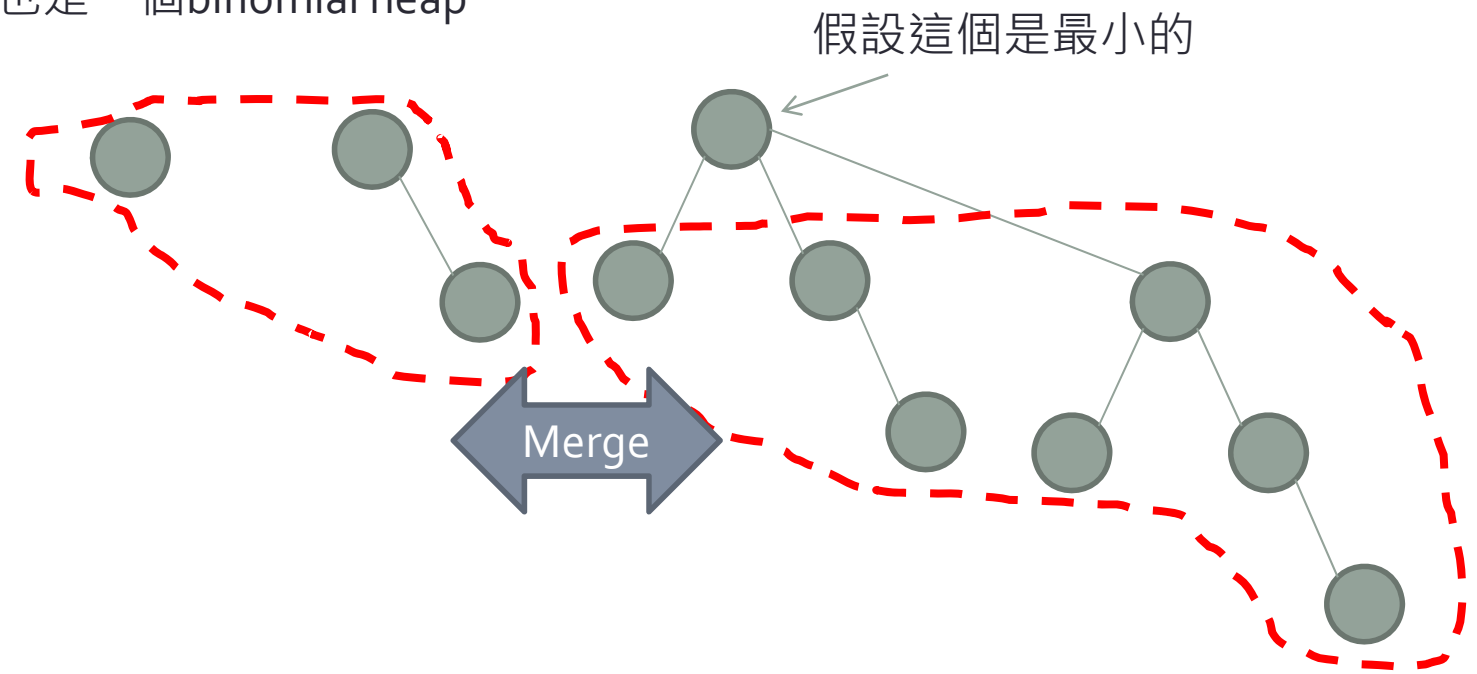
Insert & Find minimum

- Insert:
- 使用merge來implement
- 當作merge原本的heap和一個order 0的tree(只有一個node)就好
- O(log n)
- "平均來講"可以達到O(1)!

- Find minimum:
- 把所有樹的root key都檢查一遍找到最小的
- O(log n)
- 或O(1) 如果有一個pointer每次都指到最小的
- (把找的動作在其他operation做了)

Delete minimum

除了有動到的tree之外, 其他的
sub tree也是一個binomial heap



拿掉之後, 底下的sub tree也是
binomial heap

O(log n)