FIBONACCI HEAPS

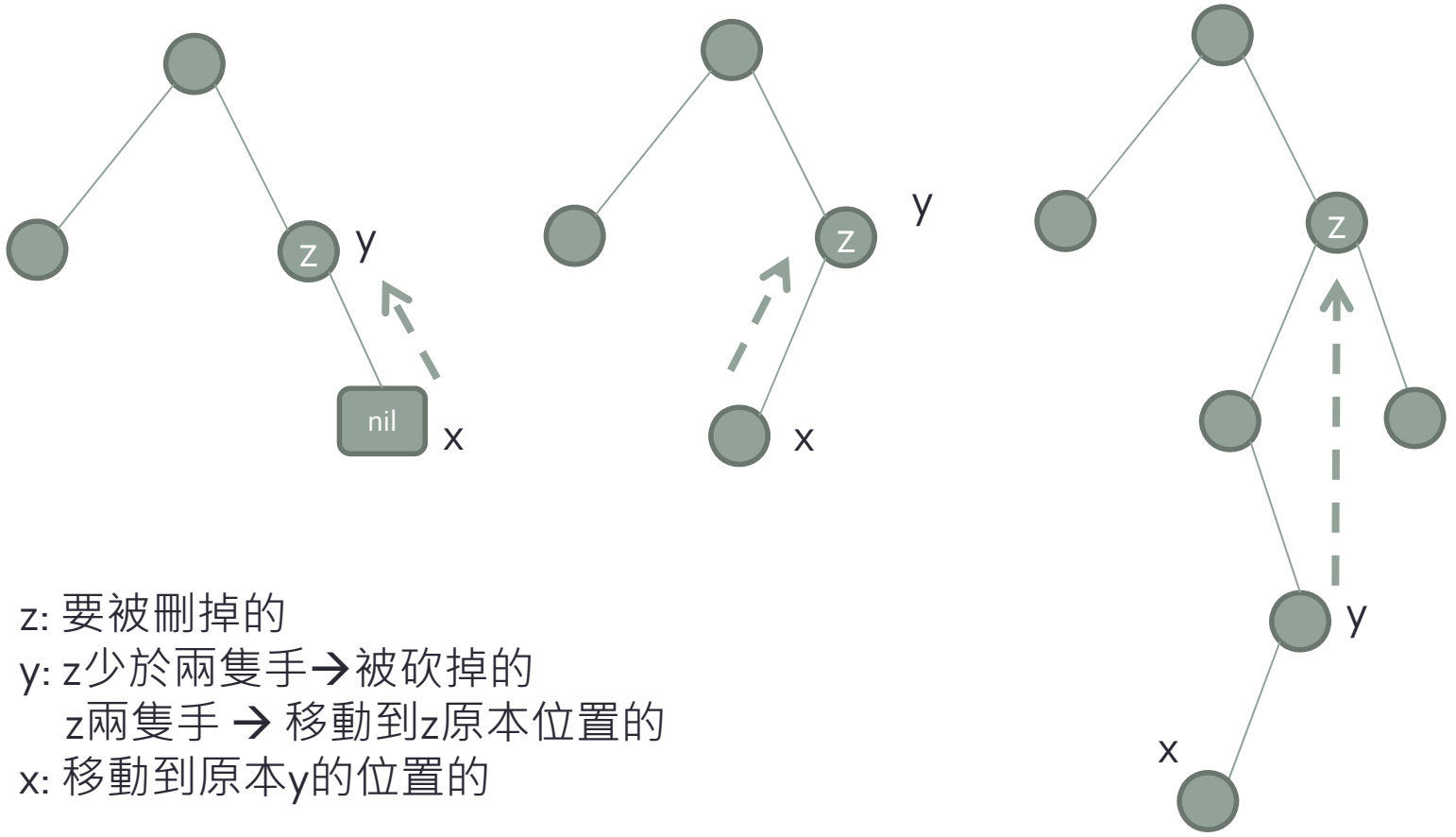
Michael Tsai

2011/1/6

Outline

- 本學期最後一堂課...有什麼話要講嗎?
- 考試方式
- Red-black tree複習
- Fibonacci Heaps

x，y，z的定義



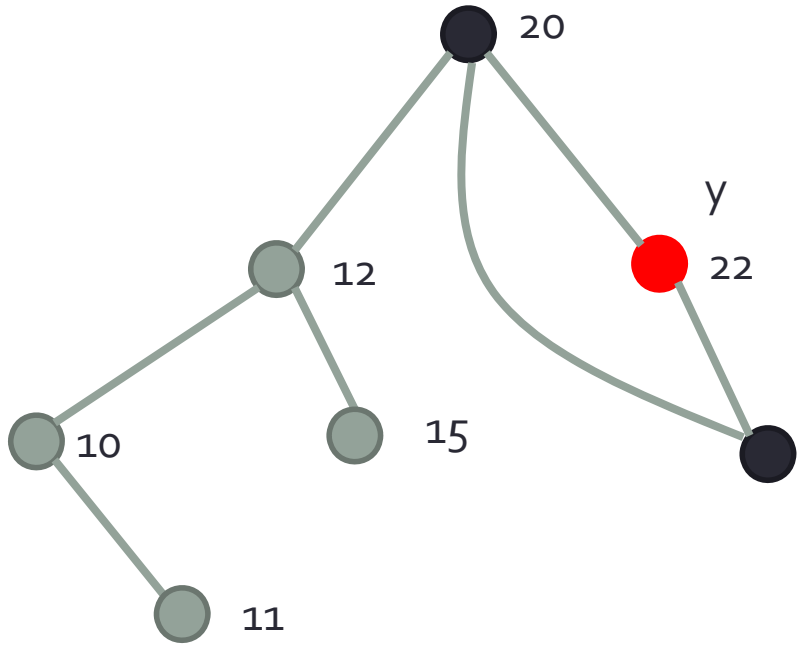
z: 要被刪掉的
y: z少於兩隻手→被砍掉的
　z兩隻手 → 移動到z原本位置的
x: 移動到原本y的位置的

拿掉一個node，什麼時候會違反規則?

- 假設前述三種情形中,
- 移動(兩手都有)或是刪除(一隻手或沒有手)的node為y

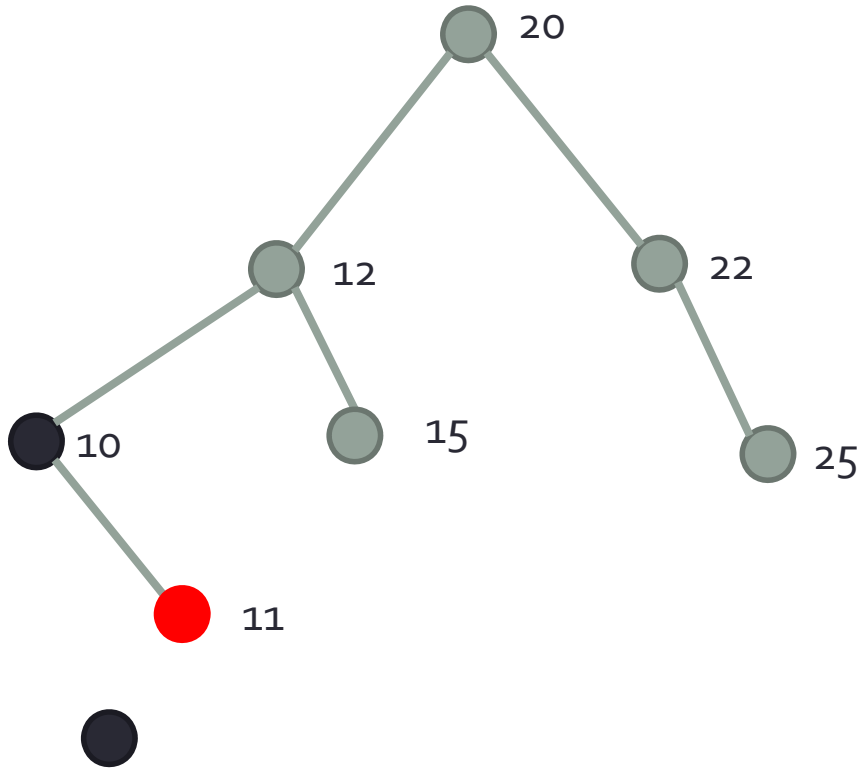
- 當y為紅色(原本的顏色), 移動或刪除會造成違反規則嗎?

- 1. black height不會改變, 因為y是紅色node
- 2. 會不會造成兩個紅色node是相鄰的呢? (父子)
- A. 如果y是被刪掉的, 因為它是紅的, 所以它的上下層都是黑的
- 不會有問題



拿掉一個node，什麼時候會違反規則?

- B. 如果y是被移動的, 假設y有children也是黑色的, 不會造成問題
- 3. y如果是紅色, 它不會是root. 所以也不會有造成違反root是黑色的規定
- 綜合以上三點, 只有當y為黑色時才會造成問題, 需要調整



可能違反規則的情形

- 當y是黑色的, 可能造成違反規則的有下列情形:
- 1. y是root. y刪掉以後, 它的一個children是紅色的, 變成了root
- 2. y原本的上下兩層都是紅色的, 移走或刪除以後變成兩個紅色相鄰
- 3. y刪掉或移走以後, 造成y的祖先們的black height不一致(因為y是黑的)

- y拿掉以後, y的黑色就被"趕到" x裡面了
- x是表示: 這邊要有一份黑色
- 但是x本身可能原本是紅或黑色

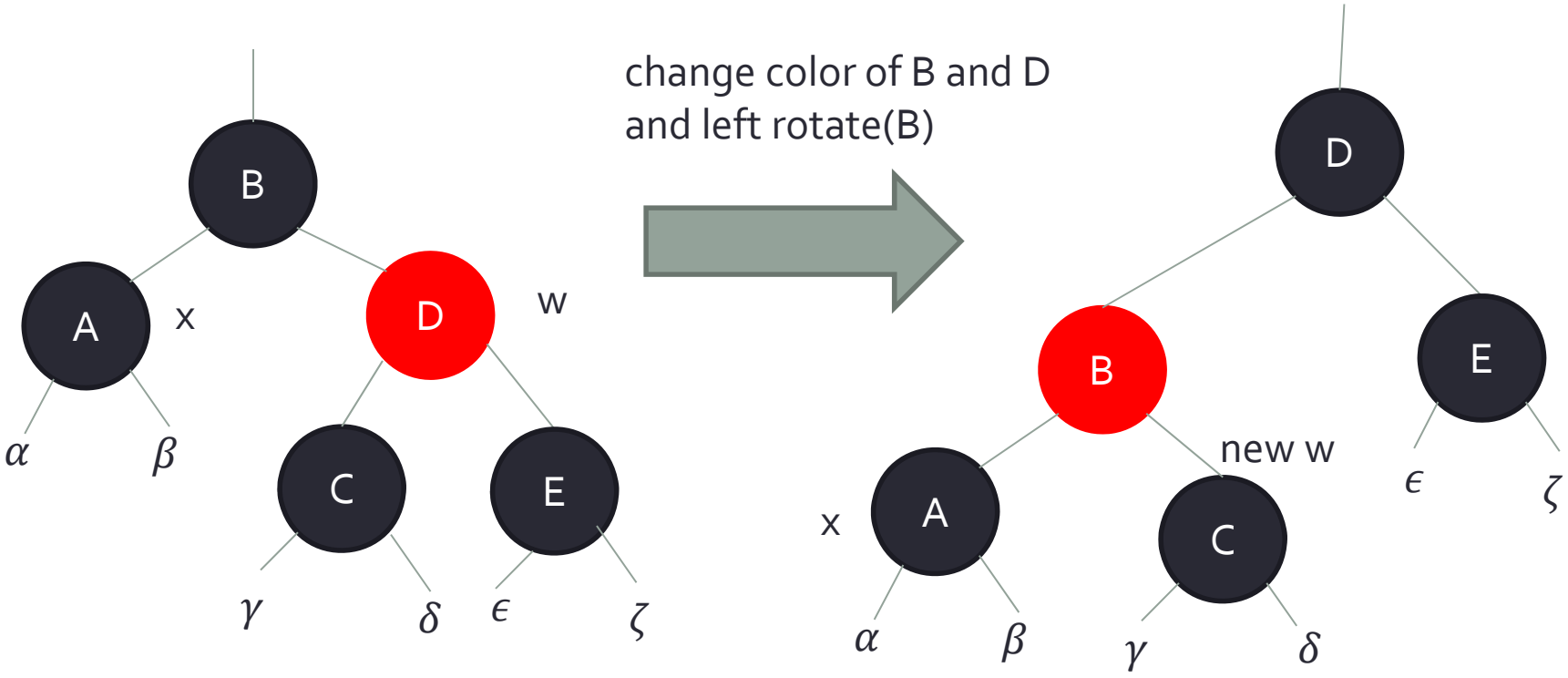
x是”紅與黑”或”黑與黑”?

- 1. x如果是”紅與黑”, 我們把它改成黑就可以了.
- (這邊要多一份黑色, 所以我就把紅色變成黑色就多一份了)

- 2. x如果是root且為”黑與黑”, 那麼直接就可以改成單純的黑色
- (反正已經到root了, 所以就算把一份黑色拿掉也不會有祖先會因此少算一份黑色的node)

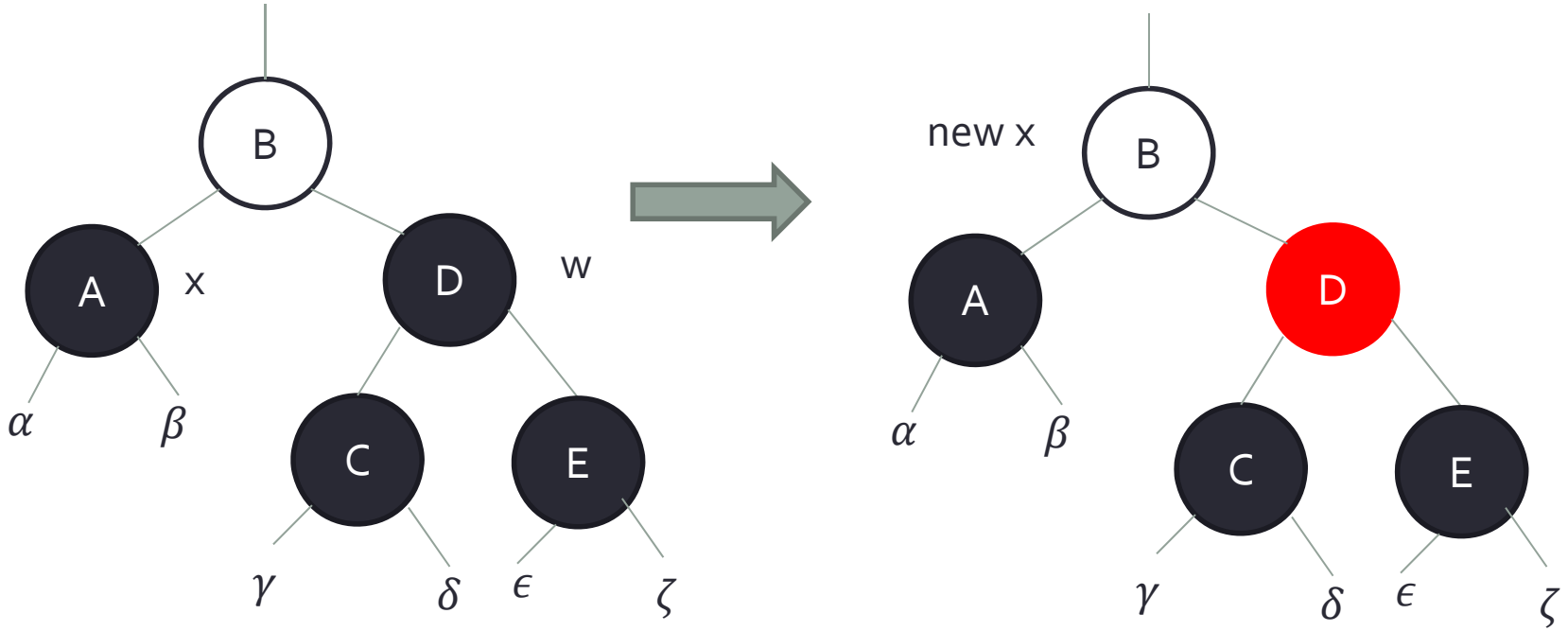
- 3. 其他的就用以下的四種case處理...

情形一：x的弟弟是紅的

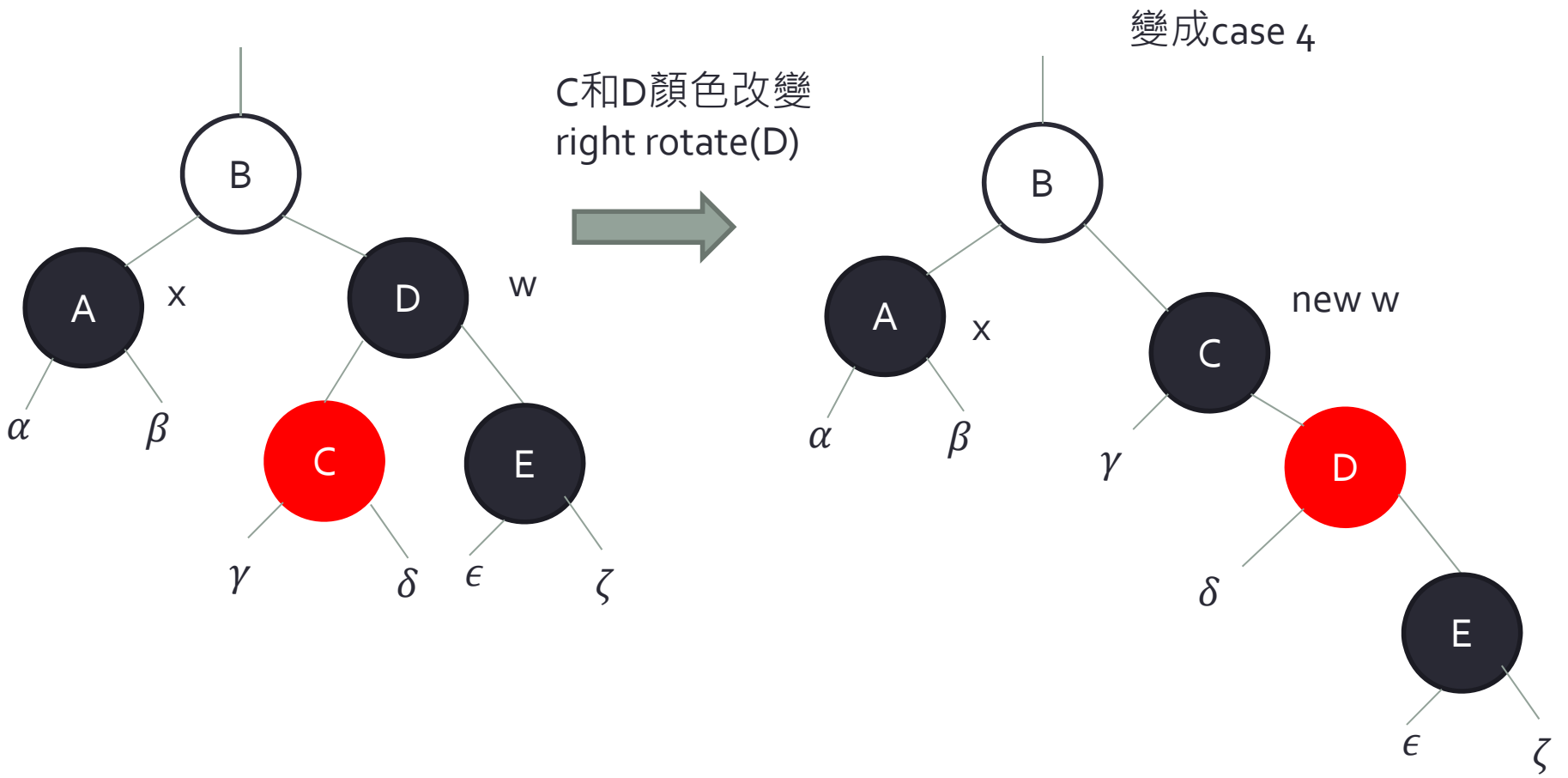


轉換成情形二、三、或四

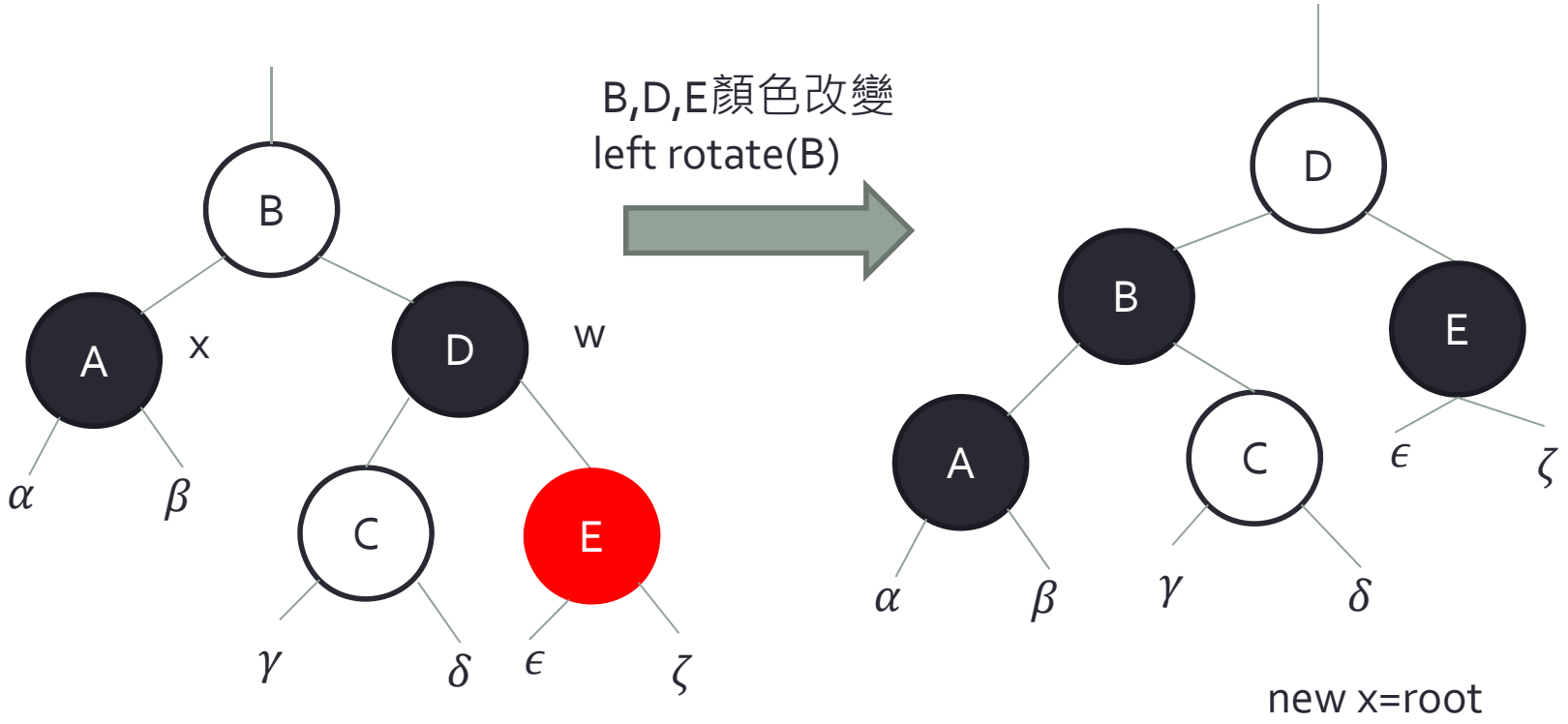
情形二：
x的弟弟是黑的&它的姪子們都是黑的



情形三：x的弟弟是黑的
它的姪子們都是左紅右黑



情形四：x的弟弟是黑的
它的姪子右邊那一個是紅的



Delete要花多少時間?

- 解決路徑圖:
- 1→ 2 → solved (從1到2的, 情形二中的圖中B原本一定是紅色, 所以可以直接解決)
- 1→3→4→solved
- 1→4→solved
- 3→4 solved
- 4 solved
- 2→解決(如果圖中B原本是紅色), or 轉到1 or 2 or 3 or 4 (x往上走一層) (如果圖中B原本是黑色)

- 所以worst case為2,2,2,...一直到root為止
- O(log n)

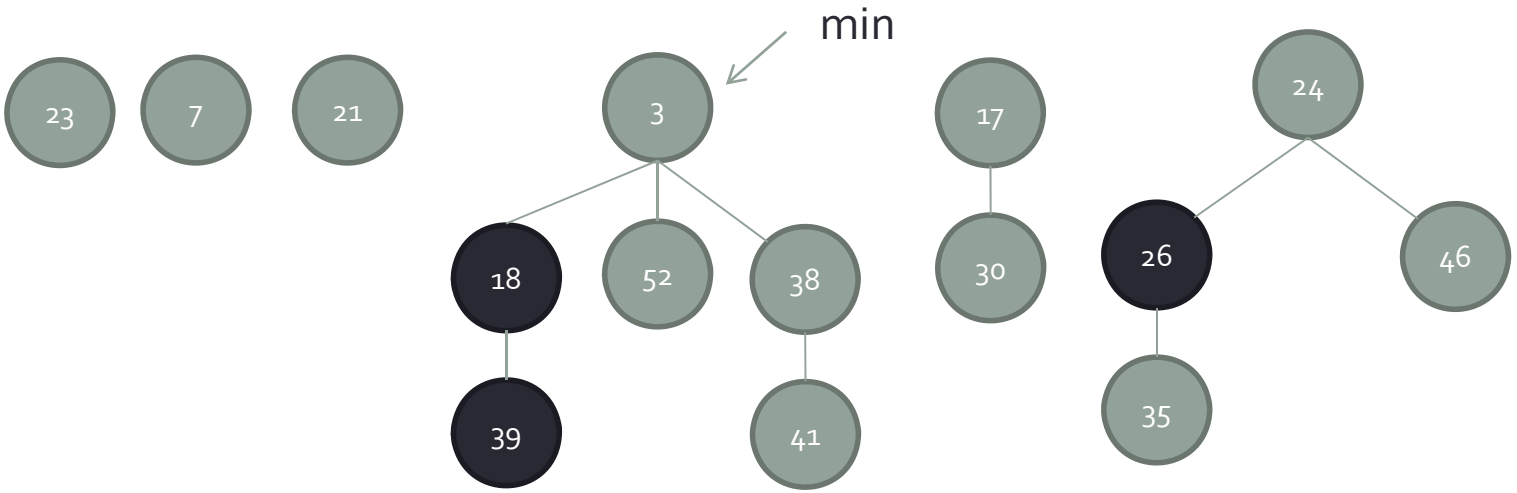
Fibonacci heaps

- 類似於Binomial heaps
- 也是forest (min tree or max tree組成)
- 各層的sibilings使用circular doubly linked list

- 有以下的operations:
- insert
- find min
- delete min
- decrease
- merge
- 除了delete min外, 其他都可以"平均"達到O(1)!
- 還可以做以下的operation
- decrease
- delete

Insert

- 把要加入的element自己當作一棵樹 (只有一個node)
- 放到heap裡面 (當作最高層的一棵樹)
- 檢查指著最小的element的指標是不是正確的
- (不用檢查是否要把同樣degree的tree merge起來, 和 binomial heap不同)
- (把工作拖到需要做再來做→可能之後某一些根本不需要做?)
- O(1)



Merge

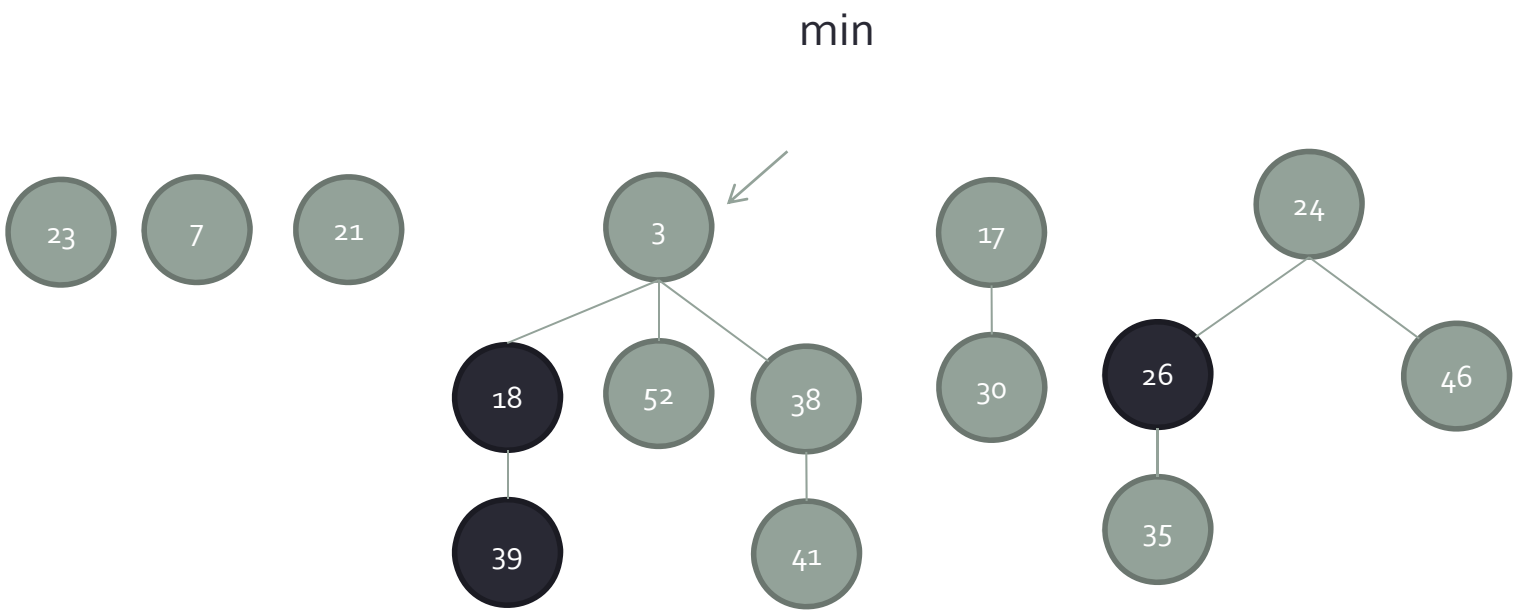
- 把兩個forest直接變成同一個forest
- 也就是把兩個circular linked list合併
- (不用檢查是否要把同樣degree的tree merge起來, 和binomial heap不同)
- O(1)

重頭戲: delete min

- 剛剛欠的現在都要要回來了
- 跟上次binomial heaps裡面merge的情形幾乎一樣
- 1. 先把min指到的element底下的sub tree都放到最頂層
- 2. min可以從最頂層拿掉了
- 3. 依序檢查是否有一樣degree的樹, 有的話就把兩個合併, 然後繼續尋找下去

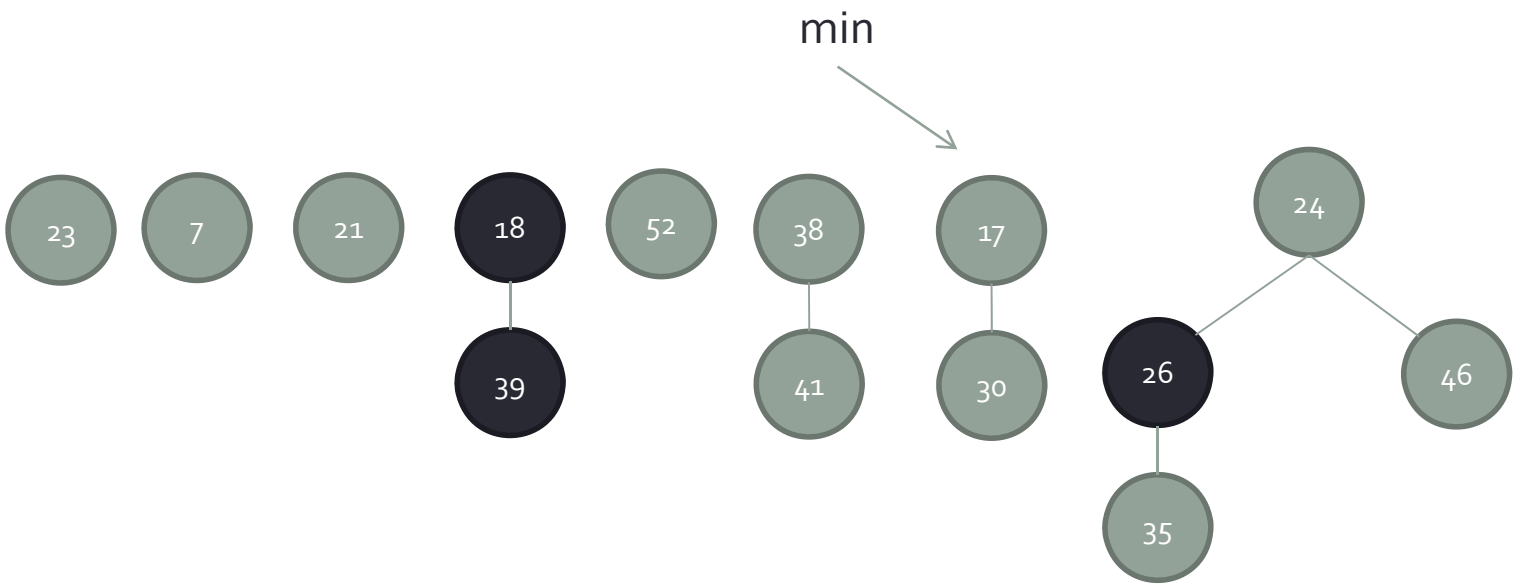
- 黑色的node是被標記, 在被跟現在的parent連結之後, 有砍掉過一個小孩的
- →所以如果某個node跟別人合併成一棵, 則要把黑色消掉

delete min例子



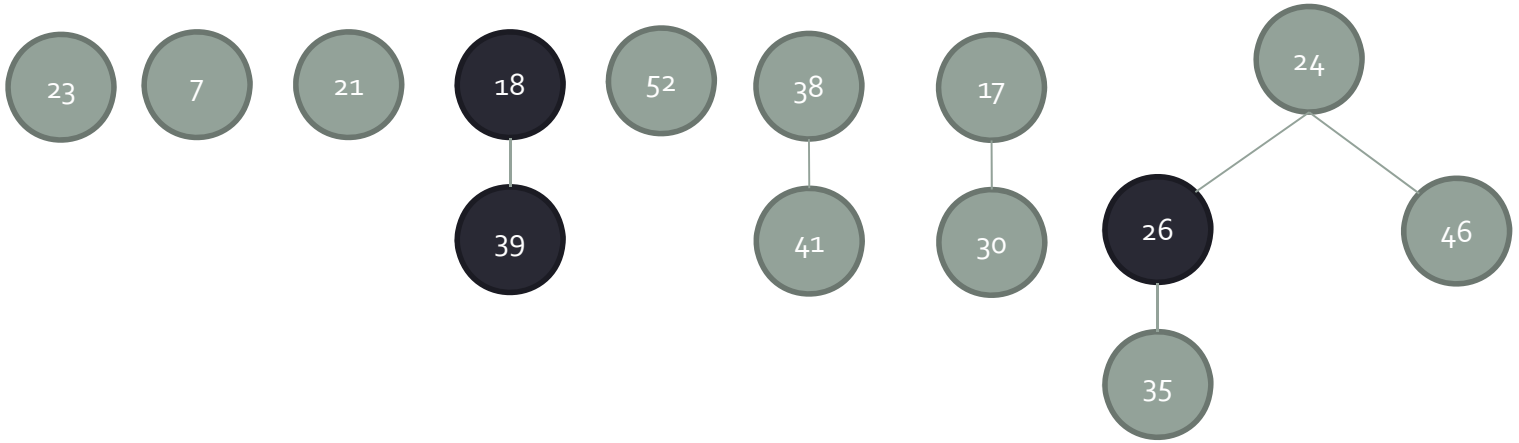
delete min例子

0	1	2	3



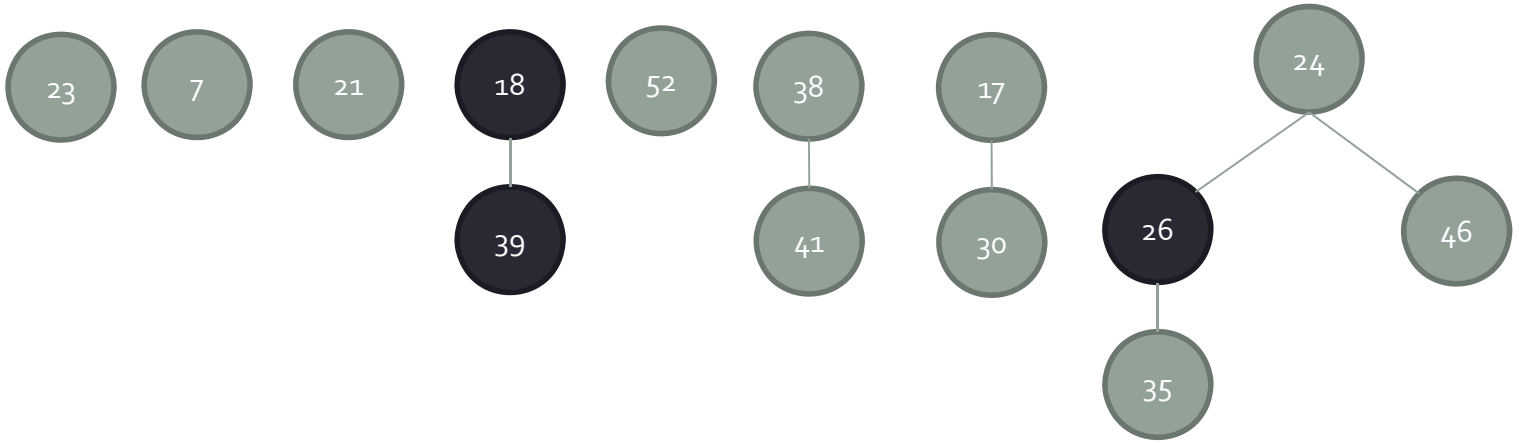
delete min例子

0	1	2	3
	17		



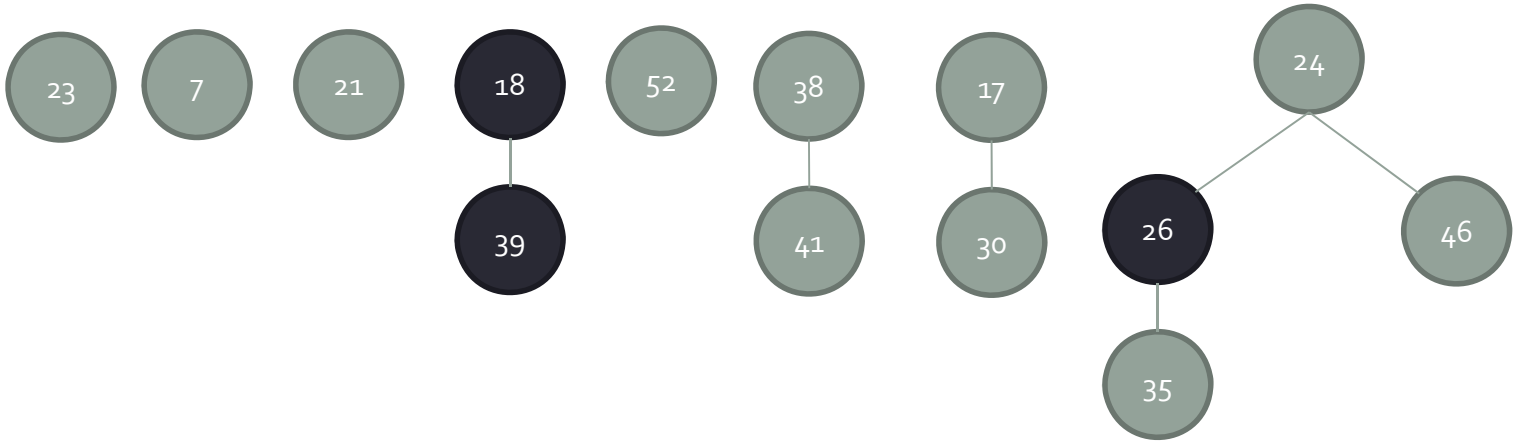
delete min例子

0	1	2	3
	17	24	



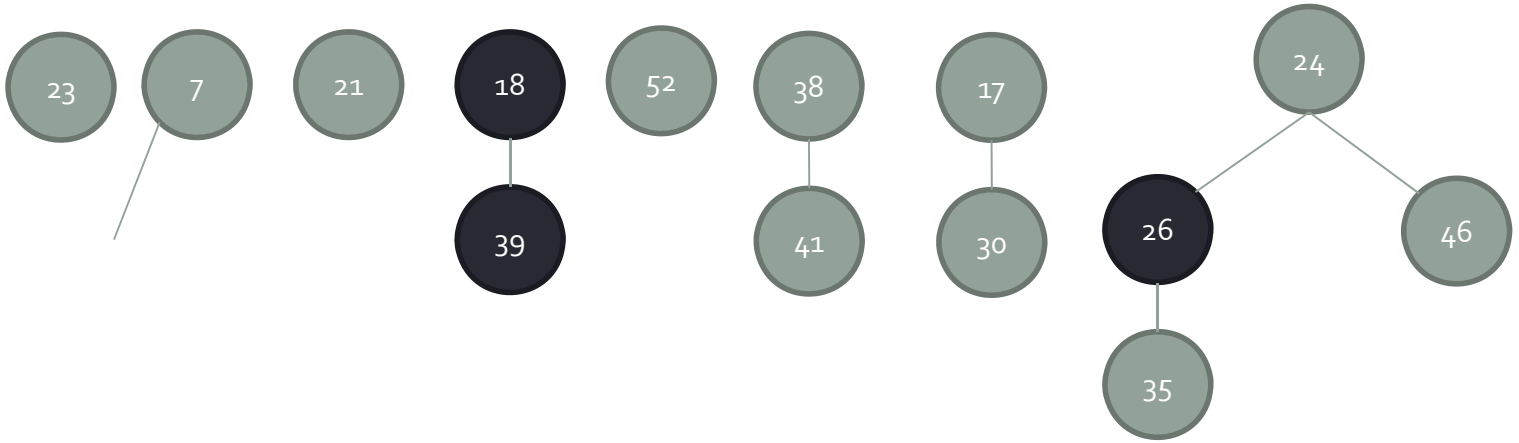
delete min例子

0	1	2	3
23	17	24	



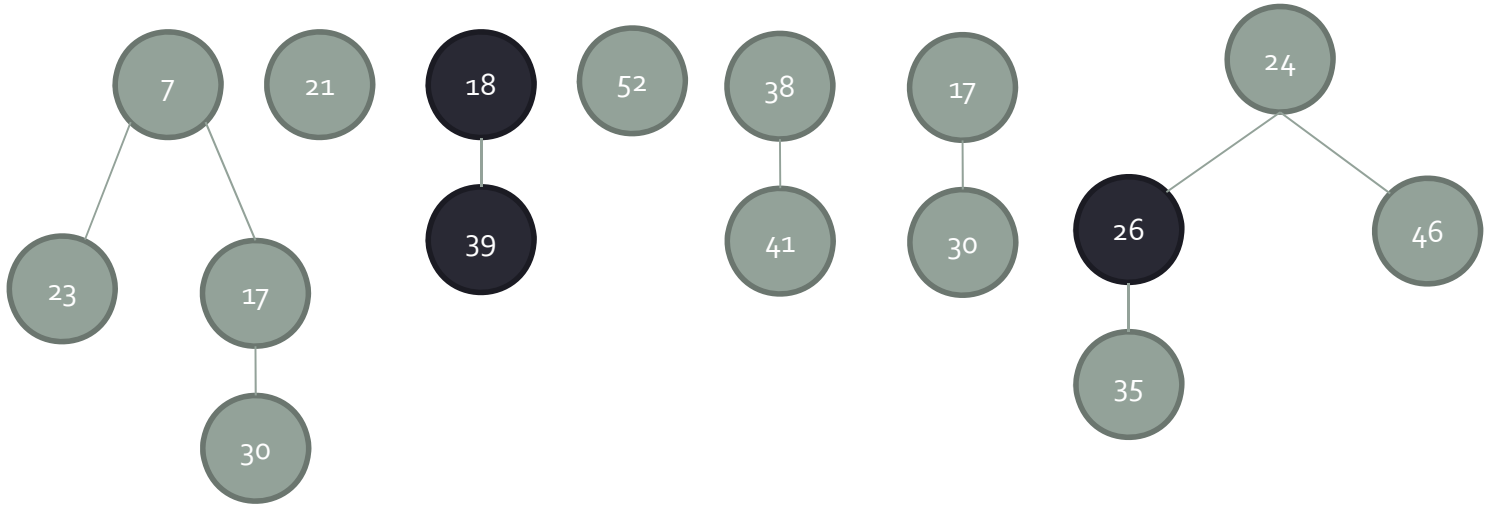
delete min例子

0	1	2	3
23	17	24	



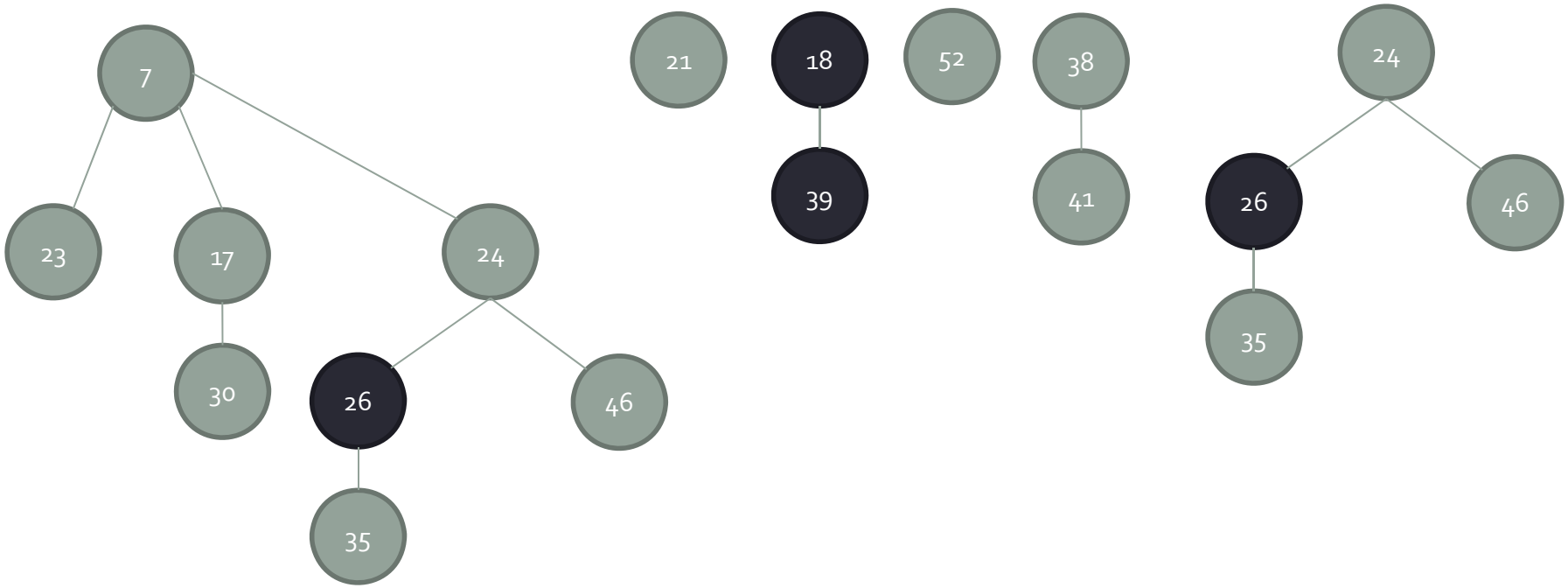
delete min例子

0	1	2	3
	17	24	



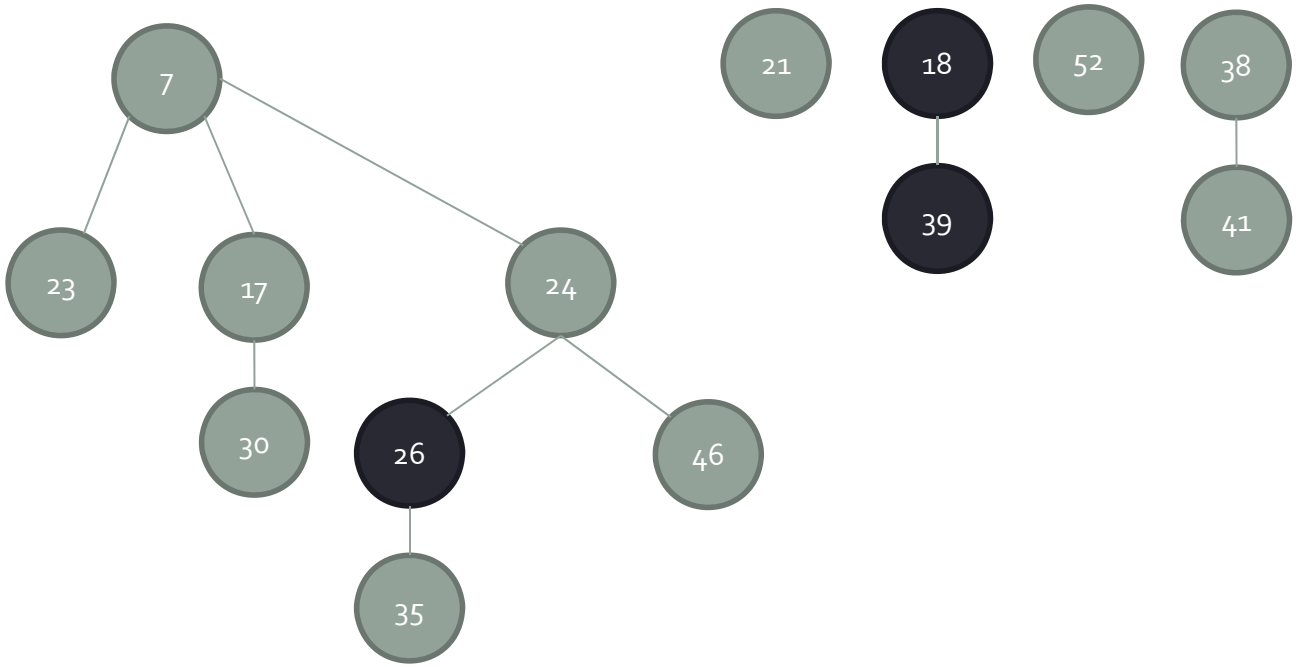
delete min例子

0	1	2	3
		24	



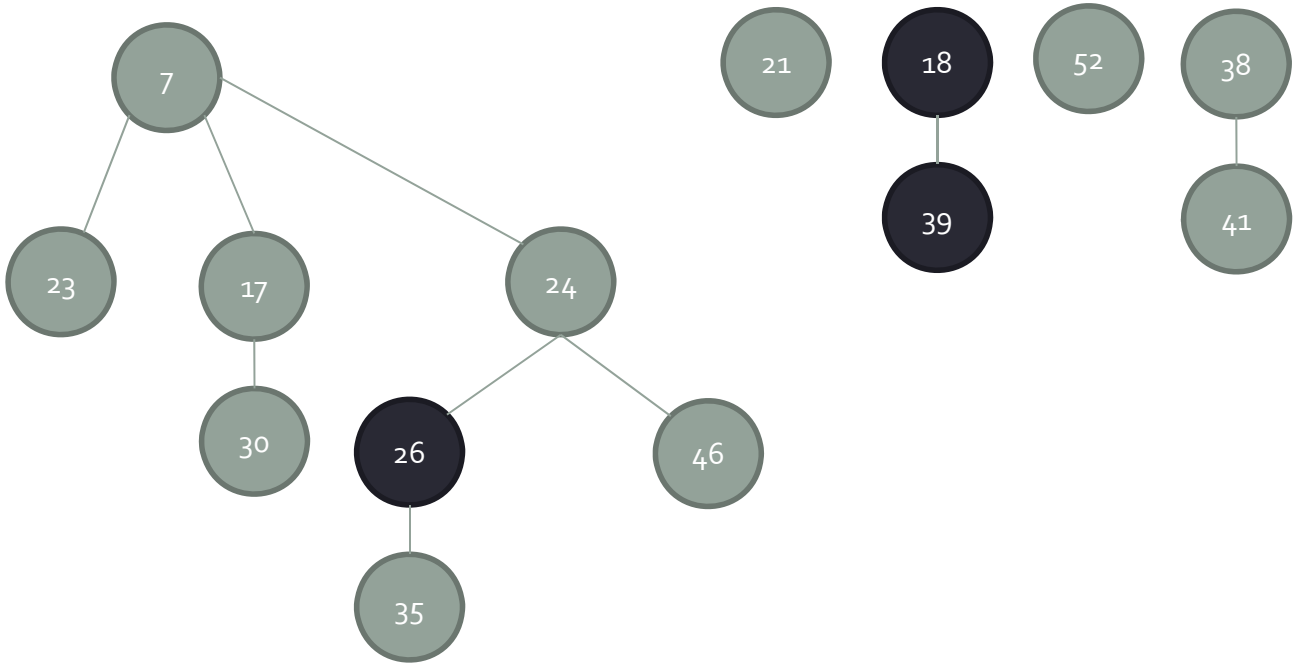
delete min例子

0	1	2	3
			7



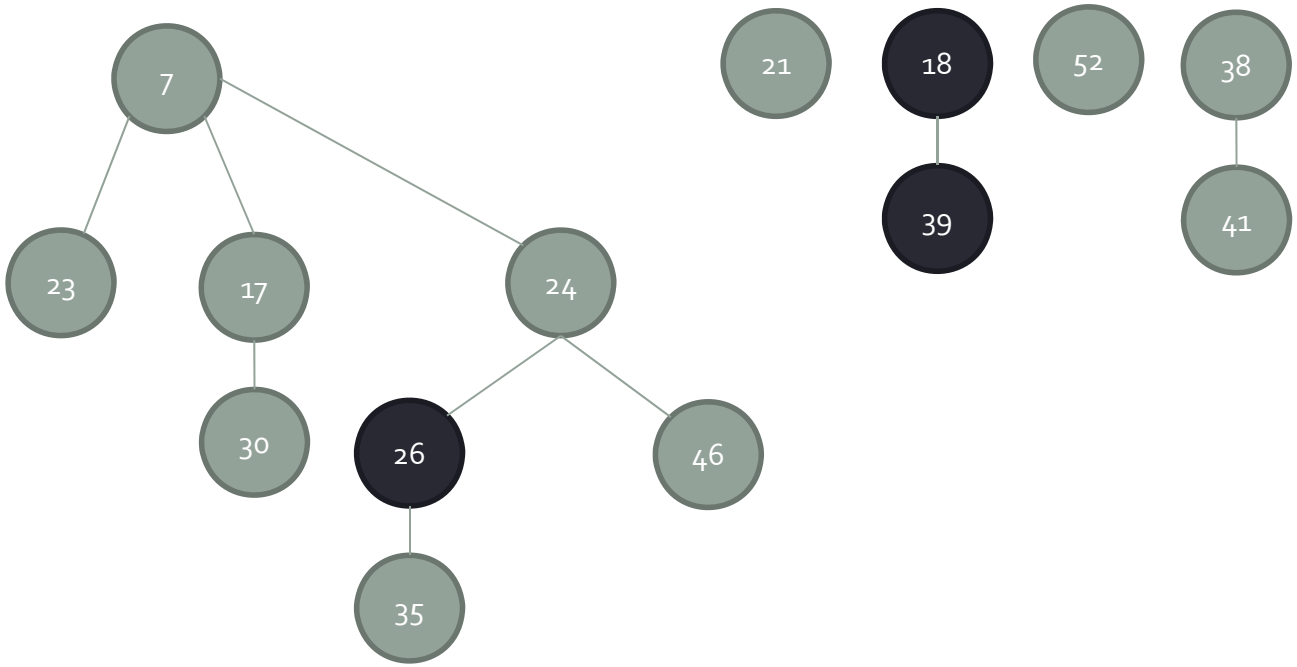
delete min例子

0	1	2	3
21			7



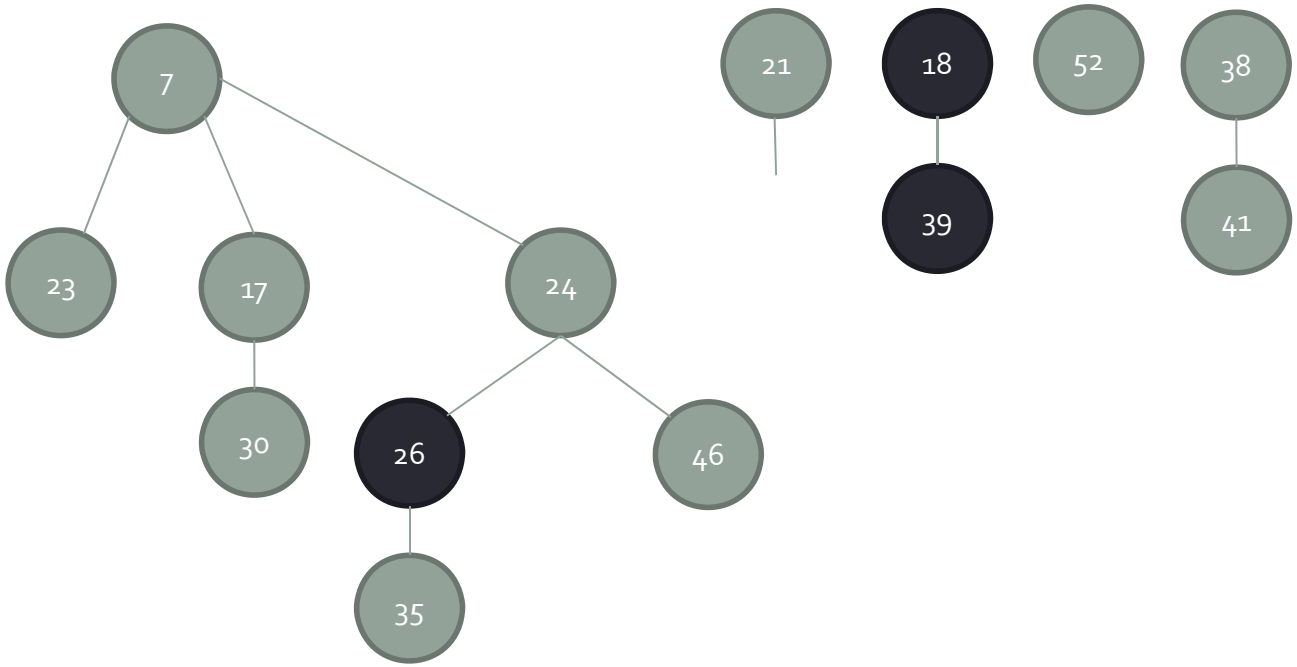
delete min例子

0	1	2	3
21	18		7



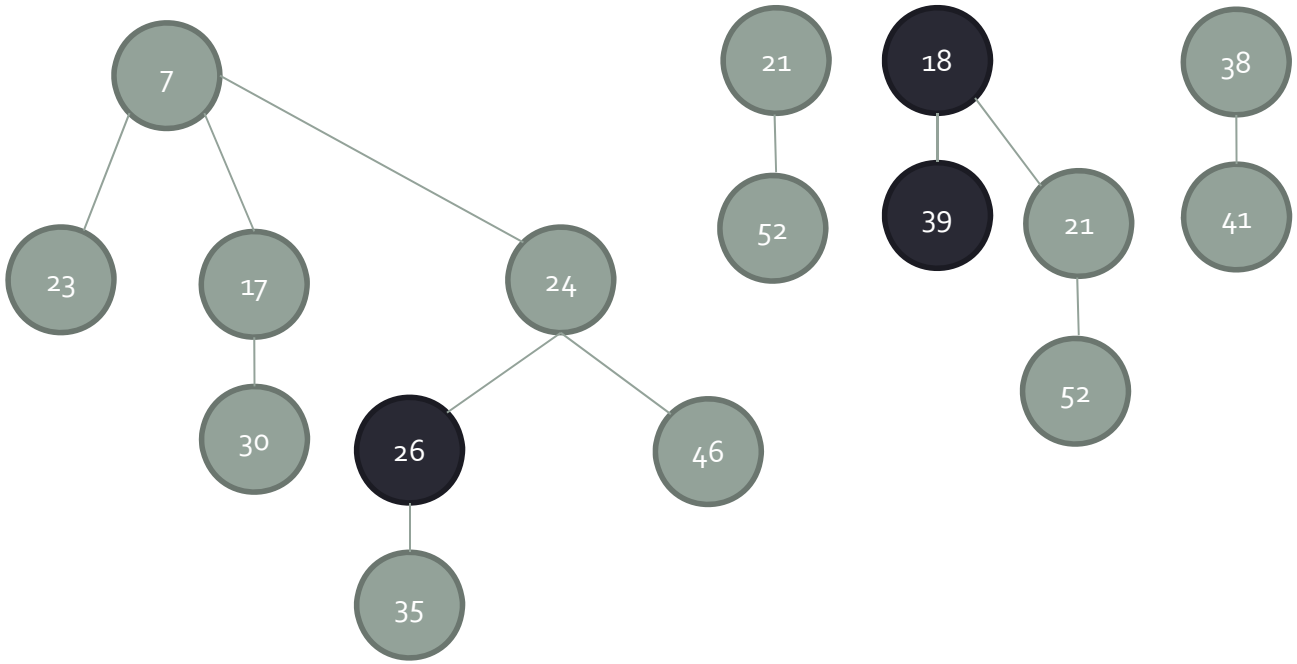
delete min例子

0	1	2	3
21	18		7



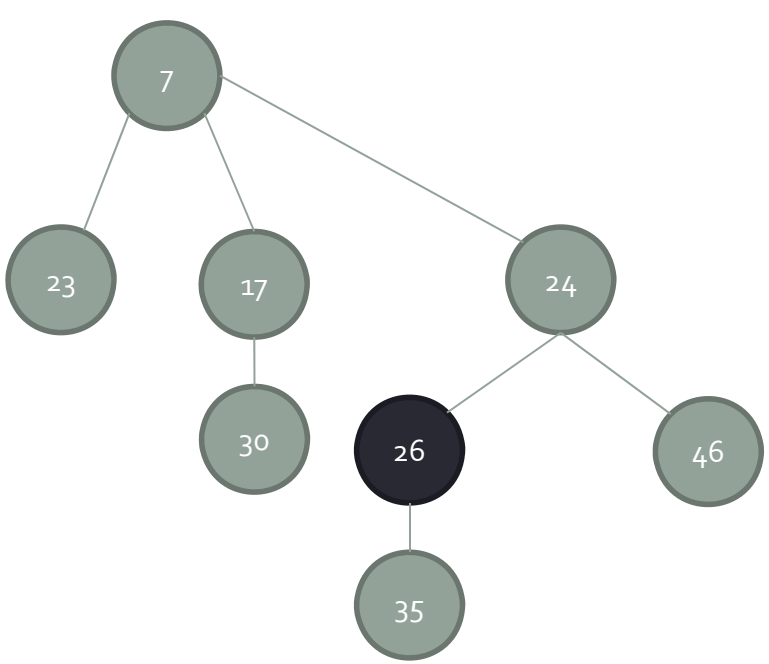
delete min例子

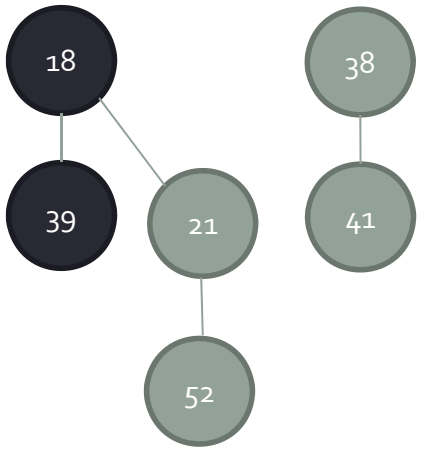
0	1	2	3
	18		7



delete min例子

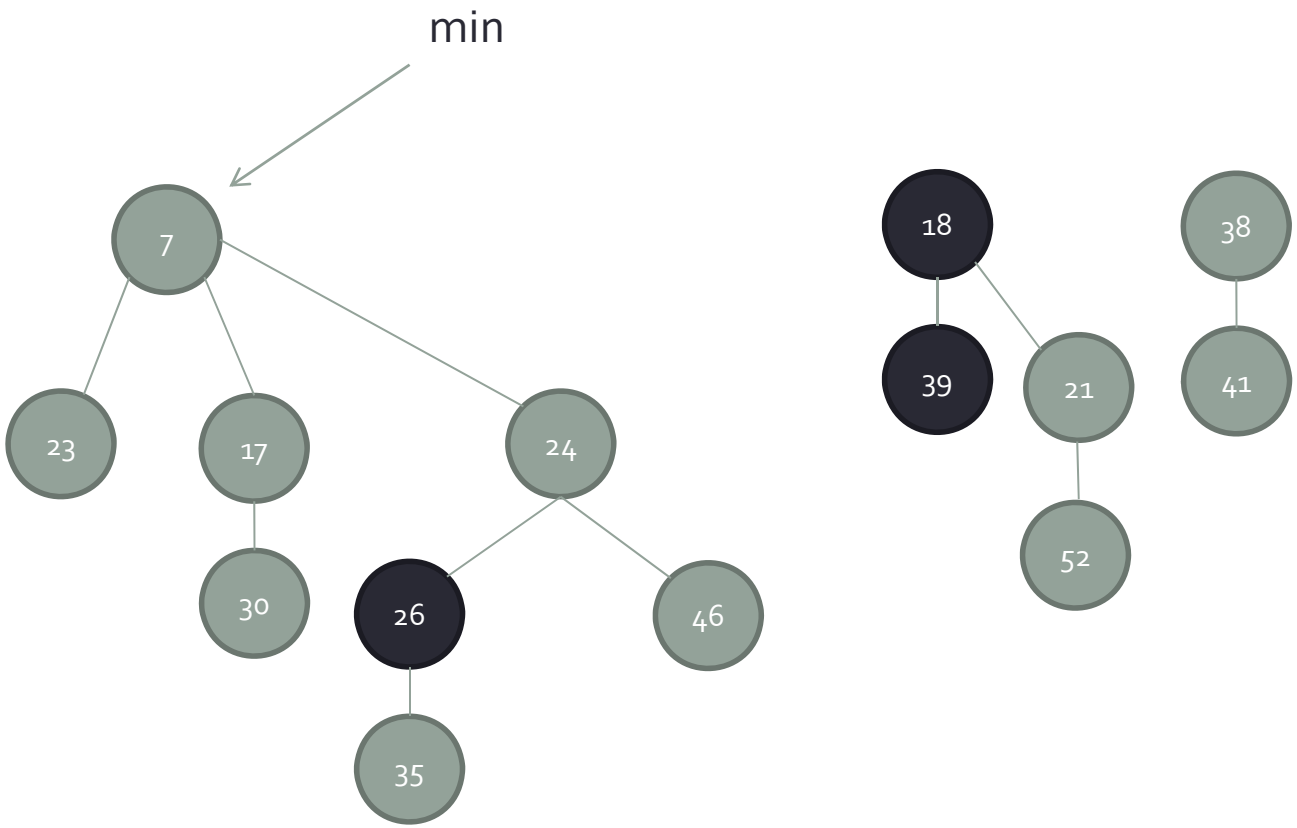
0	1	2	3
		18	7





delete min例子

0	1	2	3
	38	18	7

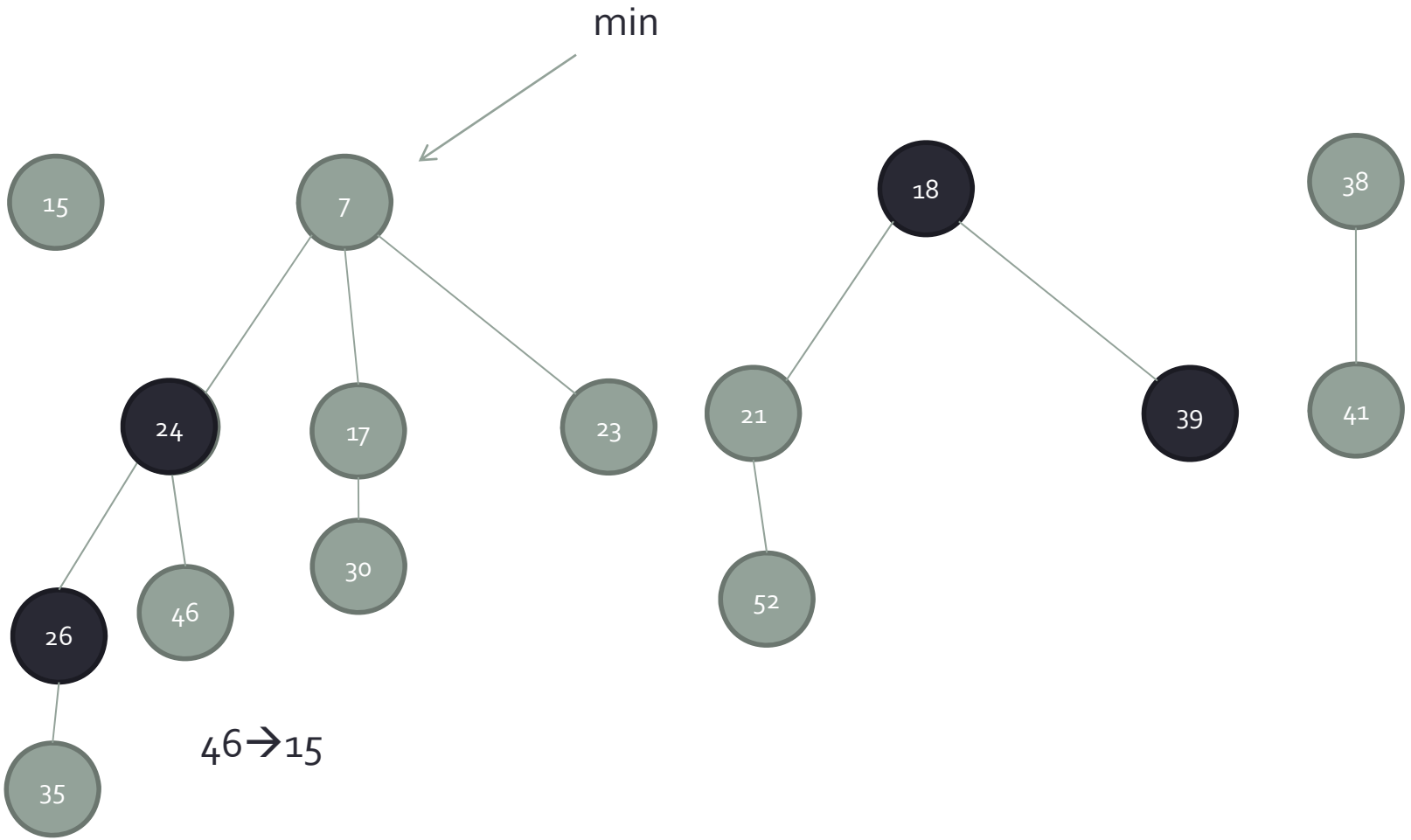


Decrease

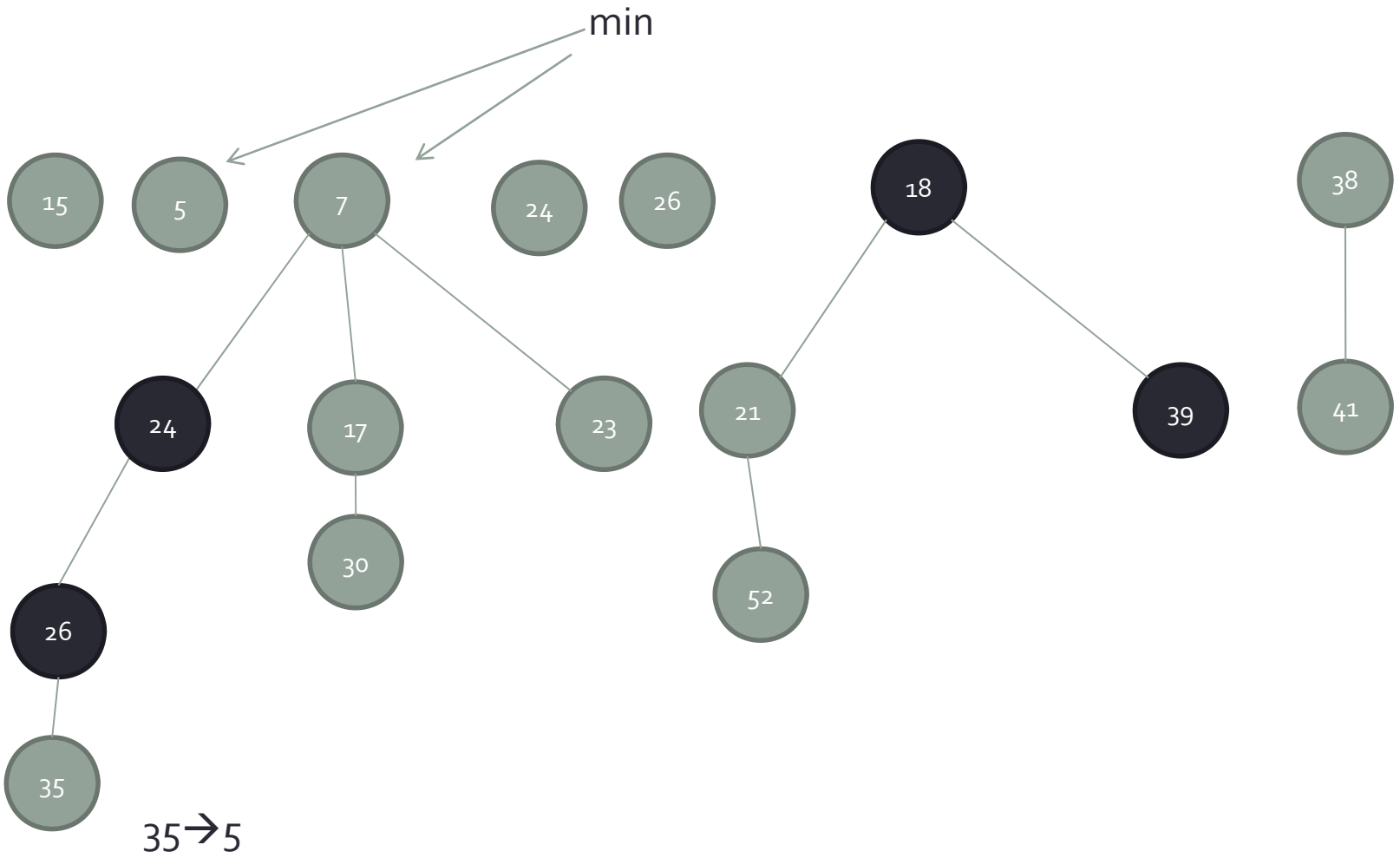
- 把某一個element(key)數字減少
- 減少以後, 那個element可能就比他的parent小了
- 必須做一些動作來修正

- 1. 如果減少以後, 比它的parent小, 就把它跟parent中間連結砍斷, 把以該element的subtree移到最上層
- 2. 如果這個parent已經被砍過一次小孩, 則把它和它的parent也砍斷, 移到最上層, 並繼續往上檢查是否有此情形...
- (此稱為cascading cut)

decrease 例子



decrease 例子



Delete

- 方法:
- 1. 把某一個element減少到負無限大 (此時它會被移動到最上層, 而且min會指著它)
- 2. delete min

Fibonacci Heap中node的degree

- 定理: $F_{k+2} \geq \phi^k, \mathrm{k} \geq 0, \phi = \frac{1+\sqrt{5}}{2}$
- 用歸納法.
- k=o and k=1時成立 ($F_2 = 1 = \phi^0, F_3 = 2 > 1.169 > \phi^1$)
- 假設$F_{i+2} \geq \phi^i$ for $i = 0,1, \ldots, k-1$
- $F_{k+2} = F_{k+1} + F_k \geq \phi^{k-1} + \phi^{k-2} = \phi^{k-2}(\phi + 1) = \phi^{k-2}\phi^2 = \phi^k$
- 得證

Fibonacci Heap中node的degree

- 定理: b為任何Fibonacci heap中的一個node, 則b的degree最多為$\log_\phi m$, $\phi = \frac{1+\sqrt{5}}{2}$, m是以b為root之subtree的node數目.
- 證明: 假設N_i為當b為degree i的subtree的最少node個數.
- 則$N_0 = 1, N_1 = 2$
- 假設 $c_1, c_2, \ldots, c_i$為b的children, 其中c_1比c_2早加入b, c_2比c_3早加入b等等依此類推
- 當c_i加入時, b已經有i-1個children, 所以c_i至少degree為i-1
- 此時, c_i最多可能少掉一個children (少兩個就會被移到最上層), 所以c_i至少degree為i-2
- $N_i = 1 + \mathrm{N}_0 + \sum_{k=0}^{i-2} N_k$

Fibonacci Heap中node的degree

- 我們現在要證明$N_k \geq F_{k+2}$, k為o或正整數, F_i為Fibonacci數列
- 用歸納法
- $N_i = 1 + \mathrm{N}_0 + \sum_{k=0}^{i-2} N_k$
- $N_0 = 1, N_1 = 2, N_2 = 3$
- $F_2 = 1, F_3 = 2, F_4 = 3$
- k=o,1,2時都成立
- 假設i=o,1,...,k-1時, $N_i \geq F_{k+2}$ 都成立

- 證明i=k時,是否成:
- $N_k = 1 + \mathrm{N}_0 + \sum_{i=0}^{k-2} N_i \geq 1 + \mathrm{N}_0 + \sum_{i=0}^{k-2} F_{i+2}$

Fibonacci Heap中node的degree

- $N_k = 1 + \mathrm{N}_0 + \sum_{i=0}^{k-2} N_i \geq 1 + \mathrm{N}_0 + \sum_{i=0}^{k-2} F_{i+2} = 1 + F_1 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i$

- Fibonacci number滿足
- $F_h = \sum_{k=0}^{h-2} F_k + 1, h > 1, F_0 = 0, F_1 = 1$
- 所以,
- $N_k \geq 1 + \sum_{i=0}^{k} F_i = F_{k+2}$
- 得證

Fibonacci Heap- time complexity

- 又, $F_{i+2} \geq \phi^i$
- 所以, $N_i \geq F_{i+2} \geq \phi^i$
- m為實際上以b為root之subtree的node數目
- $m \geq N_i \geq \phi^i$
- $i \leq \log_\phi m$. 找到bound了!

- Insert, merge, find min → O(1)

- 從Fibonacci tree degree的upper bound 我們可以得知:
- Delete min → O(degree(trees)+m) = O(log n + m), m=number of trees in the first level
- Decrease → O(c), c 為mark為黑色之node個數 (已經被刪掉一個小孩的個數)

下學期再見

- 教得不好請多多包涵
- 下學期敬請繼續支持
- 歡迎來聊天