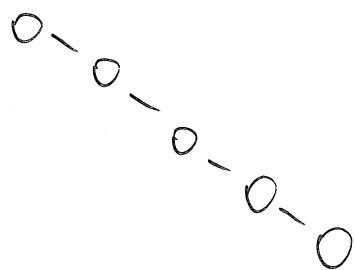


AVL Tree: 高度 = 元平衡樹。

如果單純用 Binary tree 找資料, 有可能樹的形狀不是最好的。ex.



最糟的情況是要找的資料在最下面。

Time Complexity -

$O(\text{樹高})$

我們希望樹的形狀是



假設節點數為 n , 那找資料的

Time complexity = $O(\log n)$

AVL Tree 平衡的定義為每個節點的左子樹和右子樹高度差不大於 1。

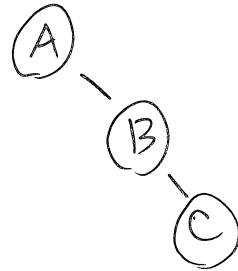
檢查節點的權重 = $|\text{左子樹高} - \text{右子樹高}| \leq 1$

(從 leaf 的父節點開始檢查, 因為 leaf 權重都為 0)

如果那個節點權重大於1，那就不符合AVL Tree的平衡，需要旋轉、調整到AVL Tree。

有四種狀況：（如果那點不符合AVL Tree，需要旋轉，是不會動到那點的父節點。）

1. Right Right Case (RR) :



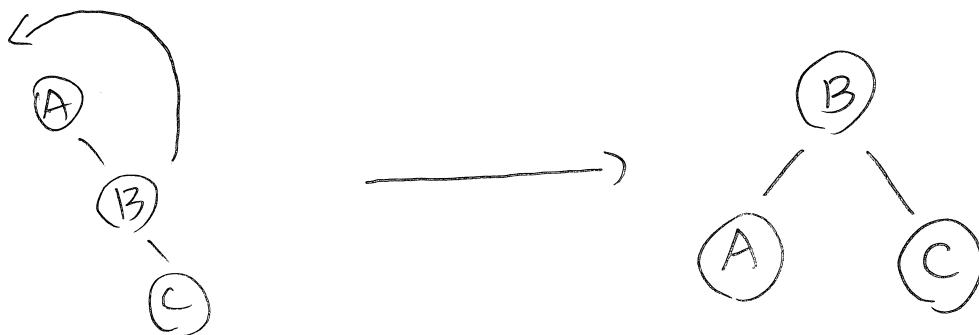
從下面開始檢查。

$$\textcircled{B} \text{ 權重} = |0 - 1| = 1$$

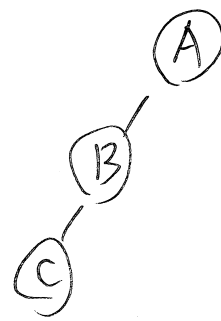
$$\textcircled{A} \text{ 權重} = |0 - 2| = 2$$

The node "A" is not sufficient condition for AVL Tree.

⇒ 由右子樹向左旋轉。



2. Left Left Case (LL) :

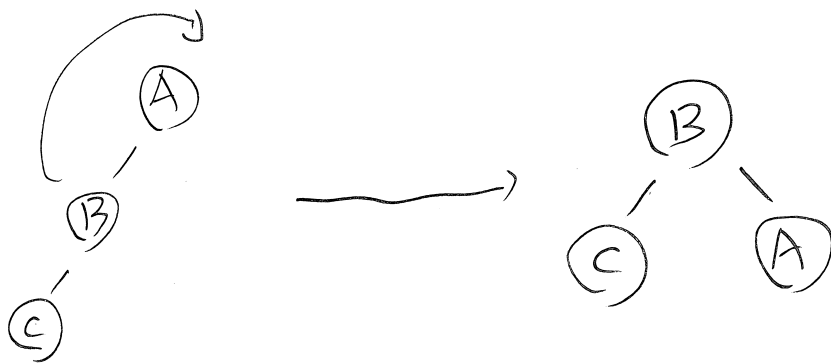


(B) 權重: $|1 - 0| = 1$

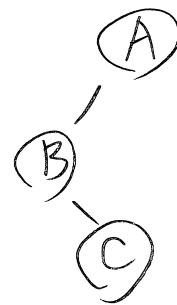
(A) 權重: $|2 - 0| = 2$

The node "A" is not sufficient condition for AVL Tree.

⇒ A 點左子樹向右旋轉,



3. Left Right Case (LR) :



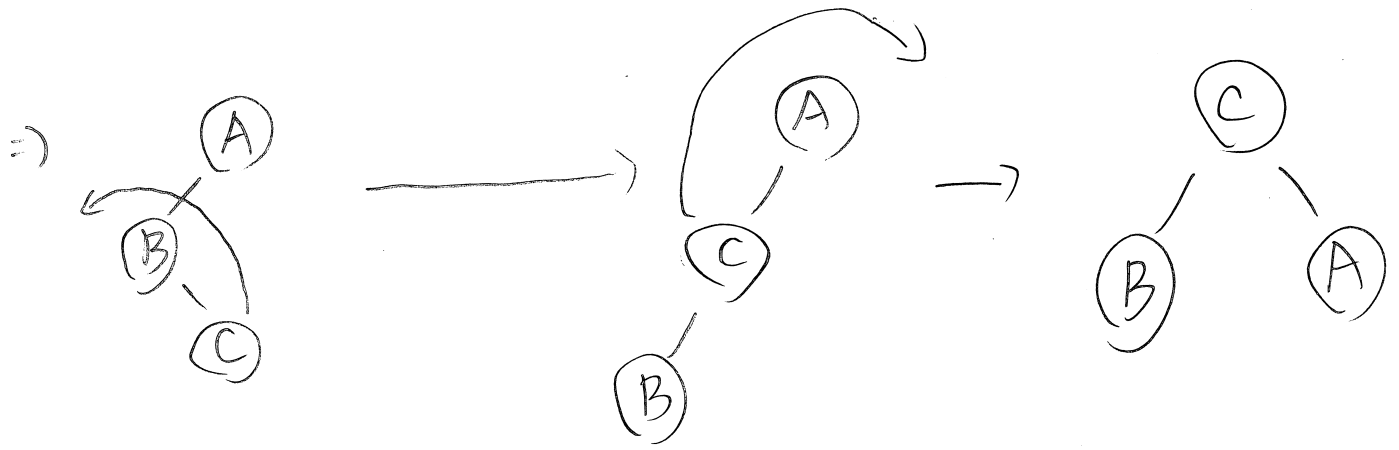
(B) 權重: $|0 - 1| = 1$

(A) 權重: $|2 - 0| = 2$

The node "A" is not sufficient for AVL Tree.

⇒ A 點不動, A 點左子樹向左旋轉, 旋轉調整好後, (變成 LL 問題), 現在 A 可動, 再由 A 的新左子樹

向右旋轉：



4. Right Left Case (RL) :

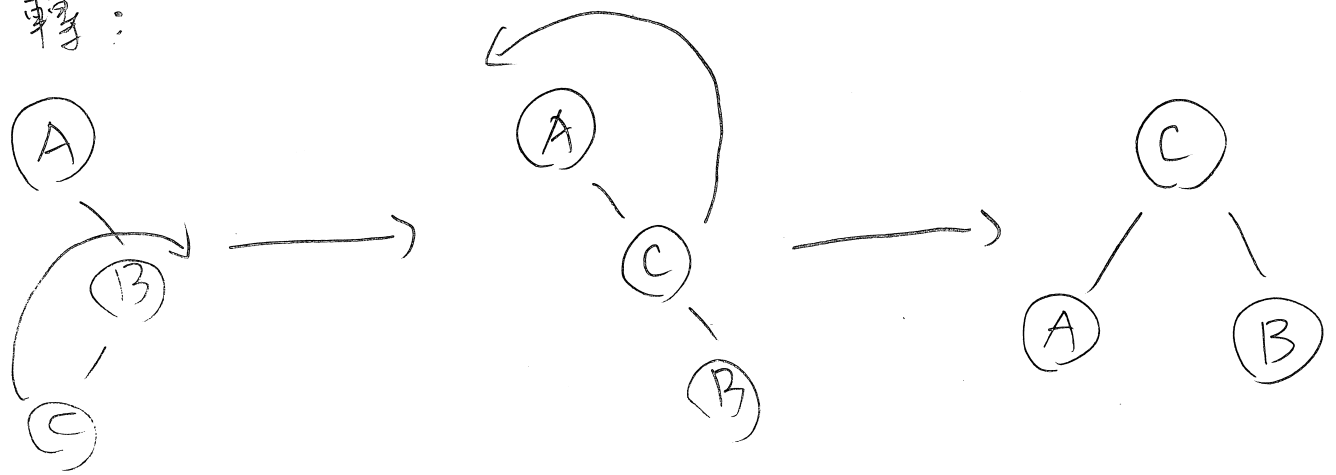
③ 權重: $|1 - 0| = 1$

① 權重: $|0 - 2| = 2$




A點不符合 AVL Tree 條件。

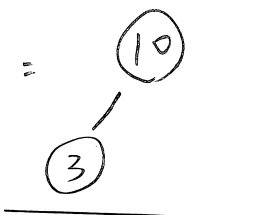
⇒ A點不動，將 A 的右子樹向右旋轉（問題變成 RR），此時 A 點可動，A 的新右子樹向左旋轉：



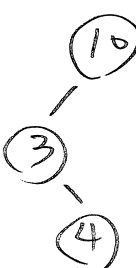
ex. 創建 AVL Tree:

< 10, 3, 4, 50, 30, 2, 18, 20, 25, 15 >

(1) Insert 10 = 

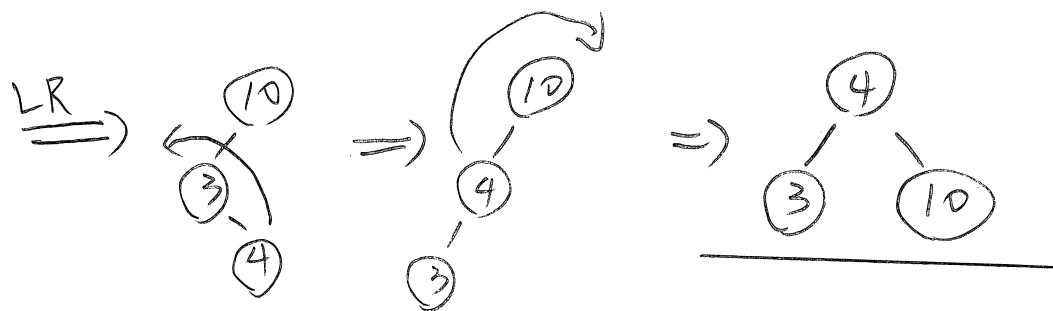
(2) Insert 3 = 

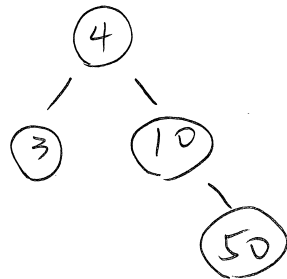
(10) 權: $|1-0|=1$ 符合

(3) Insert 4 = 

(3) 權: $|0-1|=1$

(10) 權: $|2-0|=2 \Rightarrow$ 不符



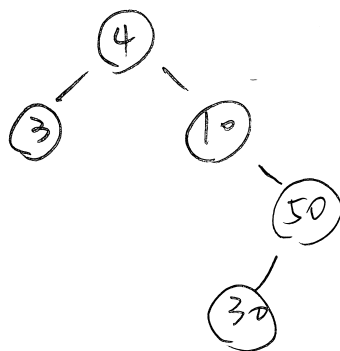
(4) Insert 50 = 

(10) = $|0-1|=1$

(4) = $|1-2|=1$

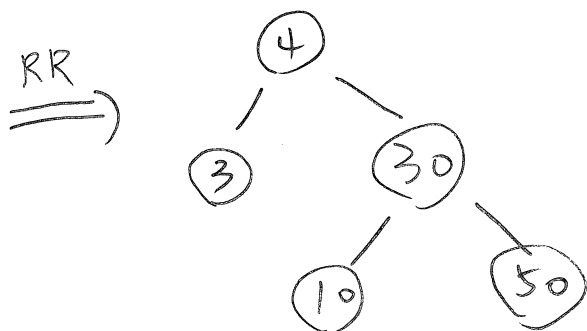
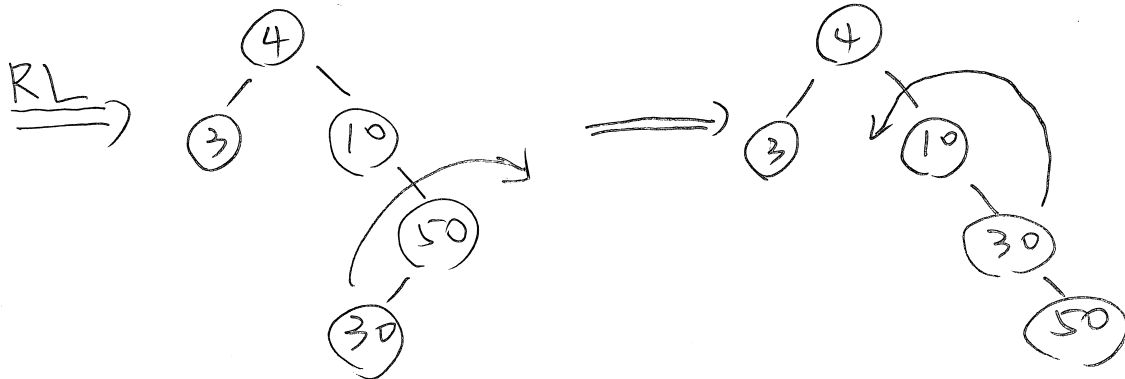
(5)

Insert 30 =



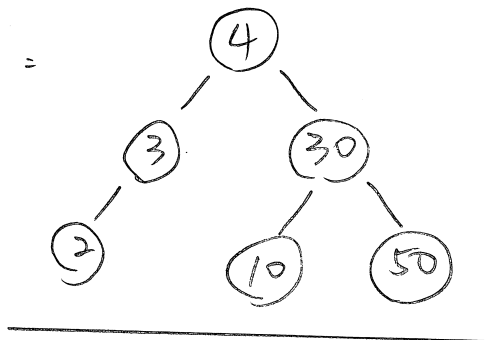
$$\textcircled{50} = |1 - 0| = 1$$

$$\textcircled{10} = |0 - 2| = 2 \Rightarrow \text{不符}$$



(6)

Insert 2 =



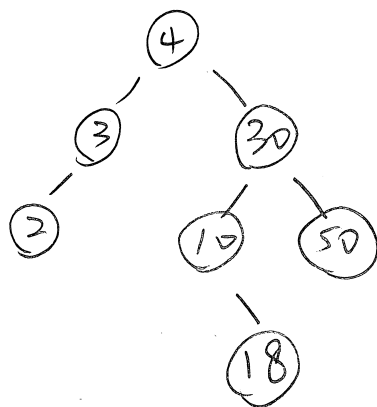
$$\textcircled{3} = |1 - 0| = 1$$

$$\textcircled{30} = |1 - 1| = 0$$

$$\textcircled{4} = |2 - 2| = 0$$

(7)

Insert 18 :



$$10 : |0 - 1| = 1$$

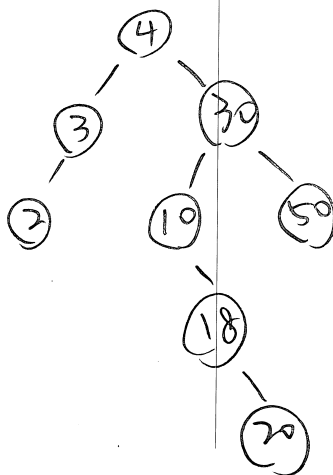
$$30 : |2 - 1| = 1$$

$$3 : |1 - 0| = 1$$

$$4 : |2 - 3| = 1$$

(8)

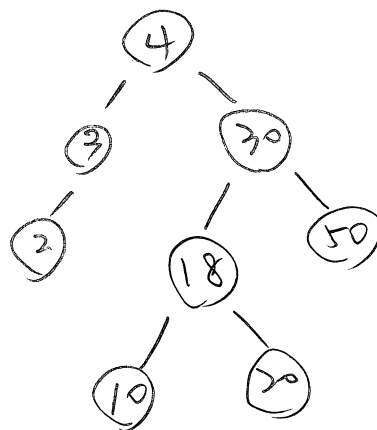
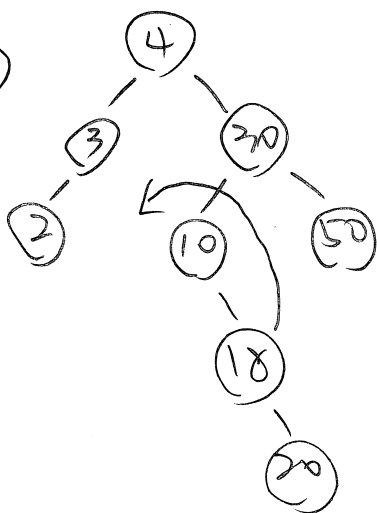
Insert 20 :



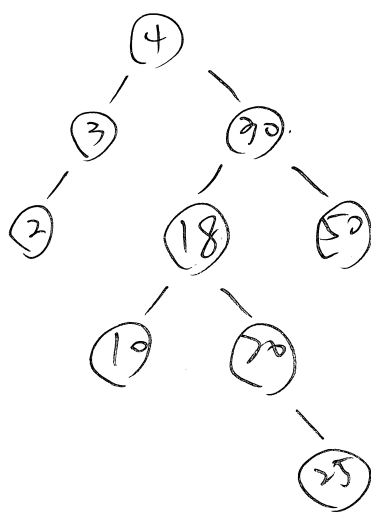
$$18 : |0 - 1| = 1$$

$$10 : |0 - 2| = 2 = \text{不符}$$

RR \Rightarrow



(9) Insert 25 :

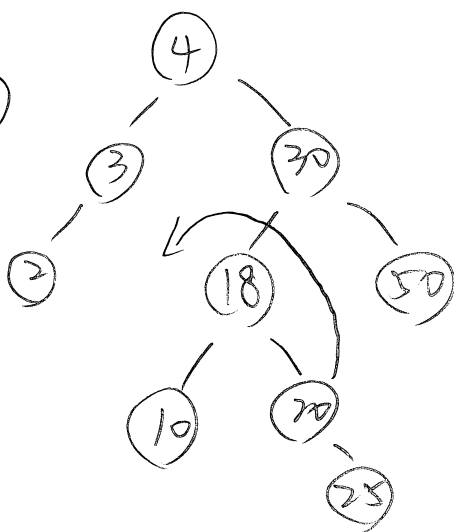


$$(20) : |0-1| = 1$$

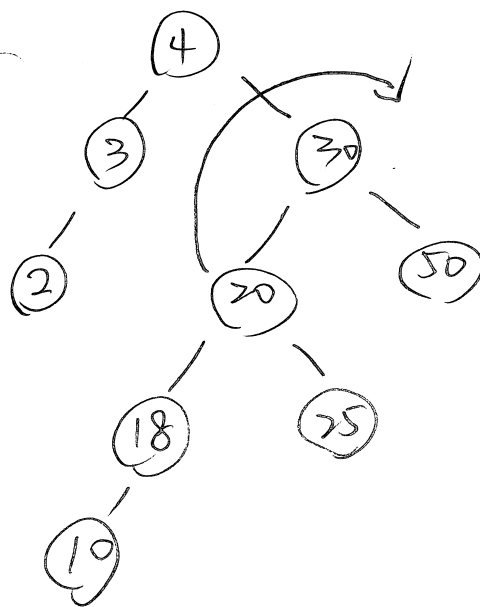
$$(18) : |1-2| = 1$$

$$(30) : |3-1| = 2 \Rightarrow \text{不平衡}$$

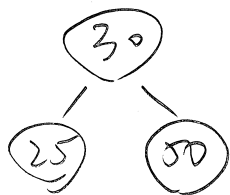
LR \Rightarrow



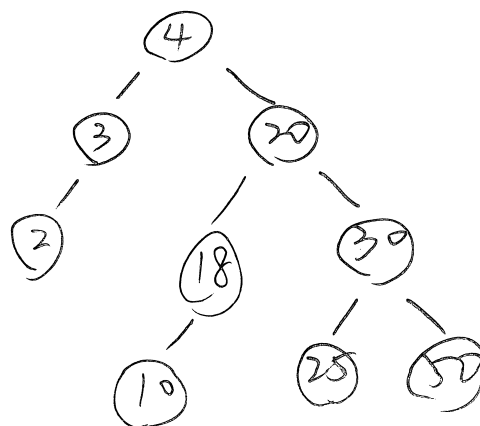
\Rightarrow



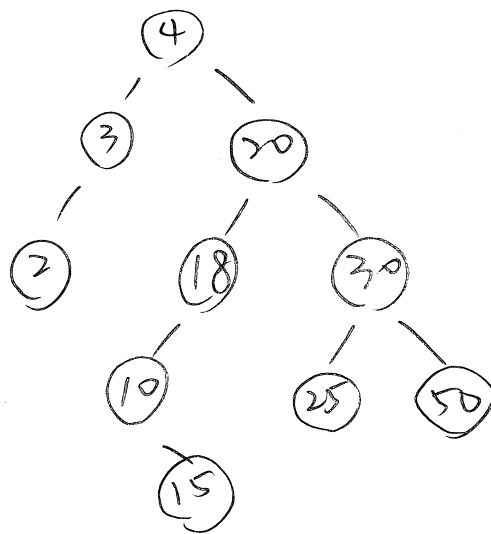
\Rightarrow 因為 30 要成為 20 的新右節點，所以原本的 25 會在 30 做 Insert 成為



整體 \Rightarrow



(10) Insert 15 :

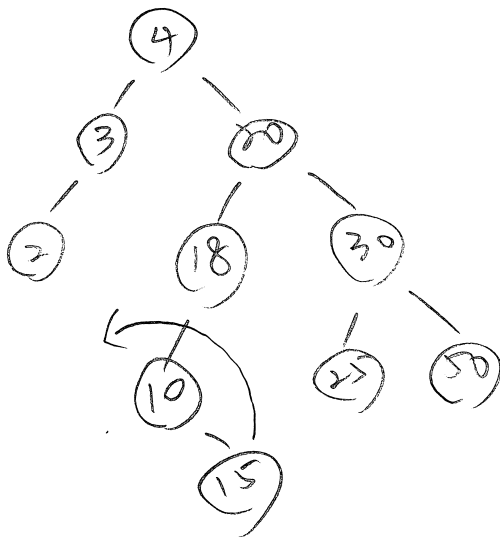


$$(10) : |0 - 1| = 1$$

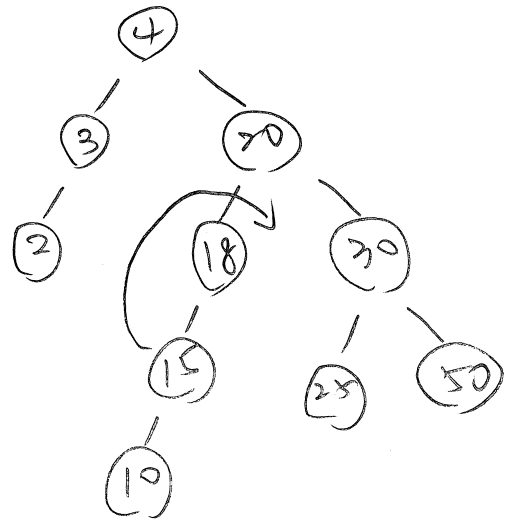
$$(18) : |2 - 0| = 2$$

\Rightarrow 不平衡

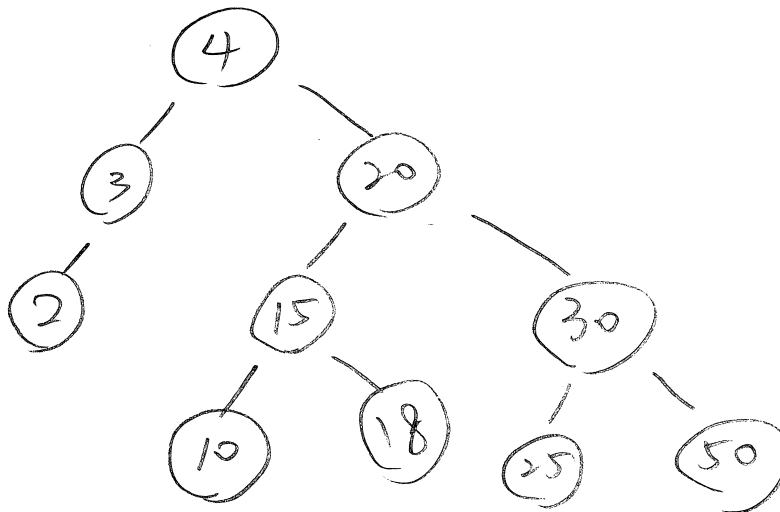
LR \Rightarrow



\Rightarrow



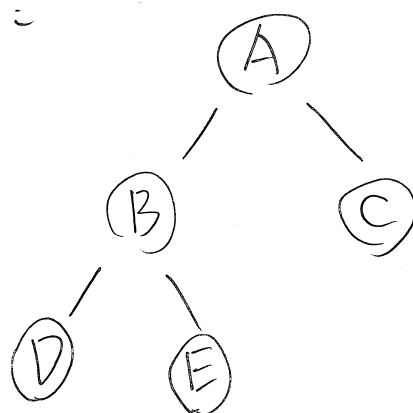
\Rightarrow



///

AVL Tree 刪除節點要注意的地方。

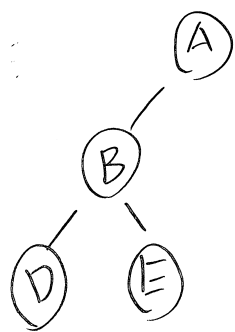
假設有棵 AVL Tree:



Inorder traversal:

DBEAC

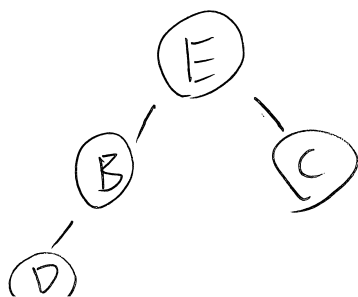
刪除 C 點，會造成不平衡。



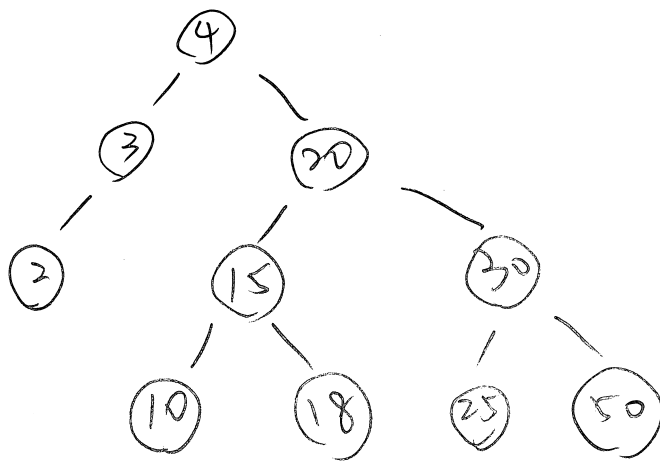
優先權: $RR \text{ or } LL > LR \text{ or } RL$

那如果是刪除 A 點，繼承 A 點的優先權為左子樹 > 右子樹。

刪除的方法跟 Binary Tree 一樣，用 Inorder successor 方式選擇，再做 AVL Tree 的平衡。

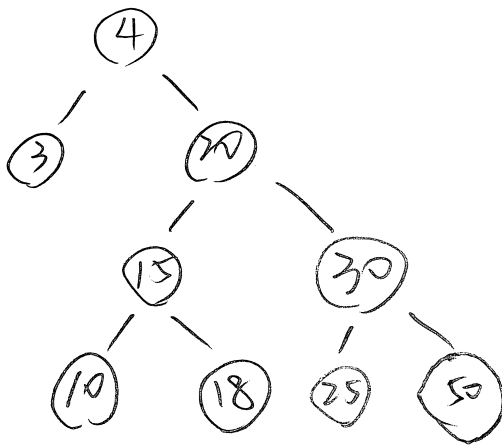


ex.

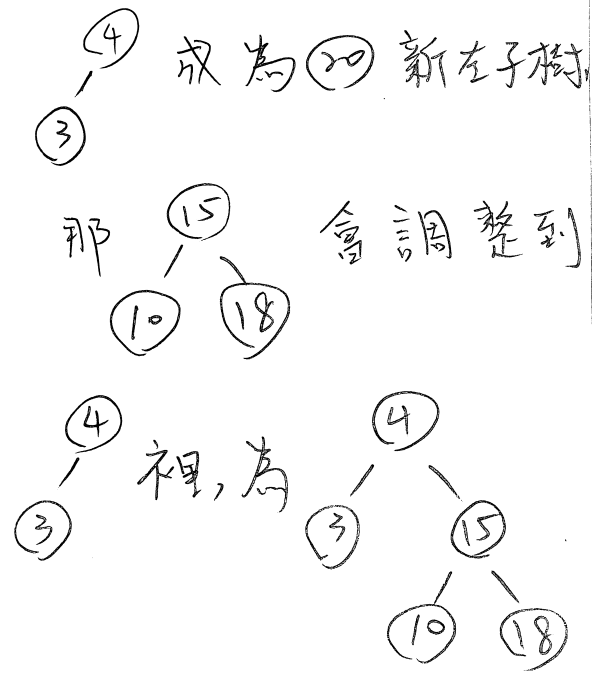
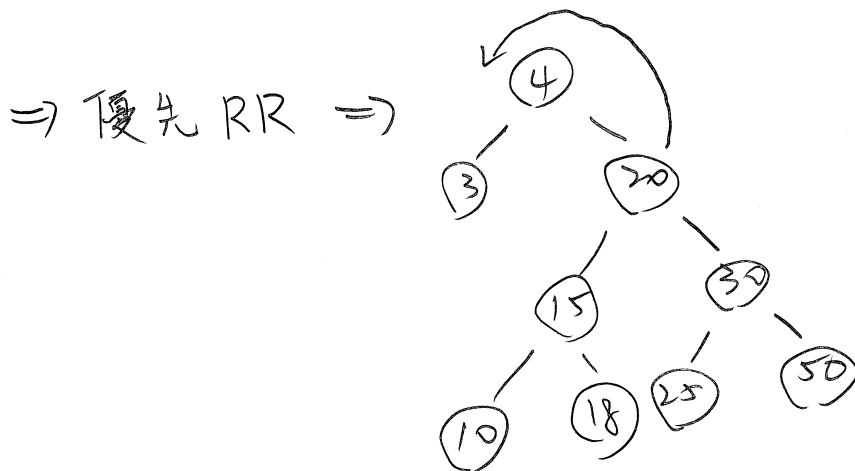


刪除 $< 2, 20, 25, 30 >$:

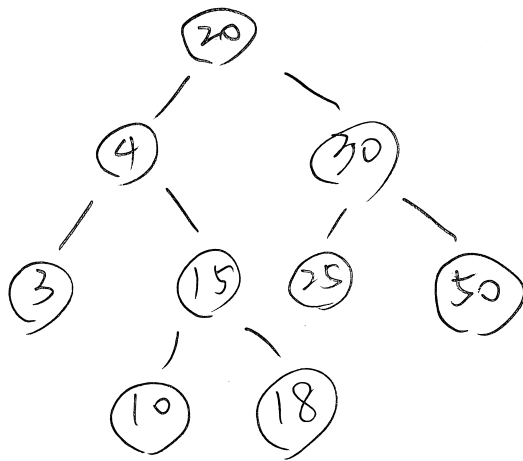
c1) Delete 2 :



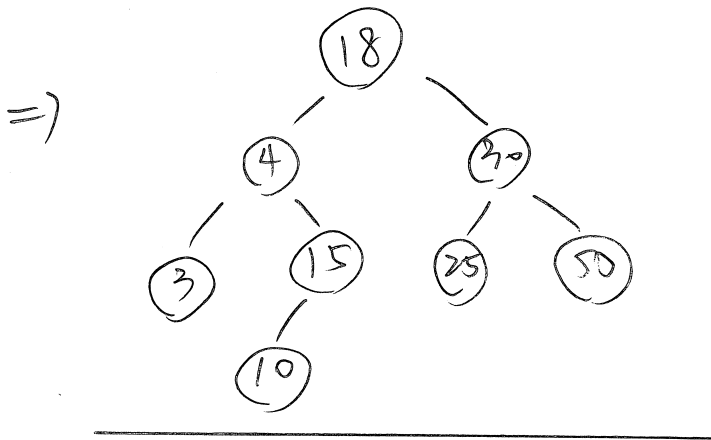
明顯 ④ 權重: $|1-3|=2$
不符 AVL Tree。



調整平衡
⇒



- (2) 用 1) (20), 由 Inorder successor 可知 (18) 接 (20),
(18) 成為新的繼承者。



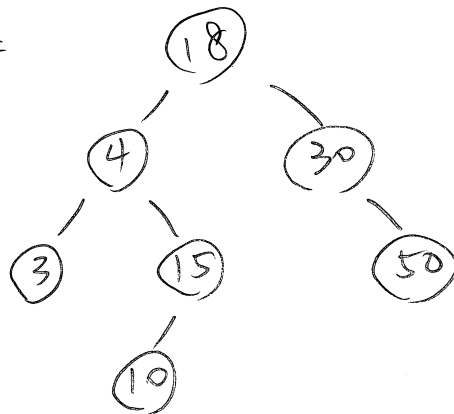
$$(15) = |1 - 0| = 1$$

$$(4) = |1 - 2| = 1$$

$$(30) = |1 - 1| = 0$$

$$(18) = |3 - 2| = 1$$

- (3) 用 1) (25) :



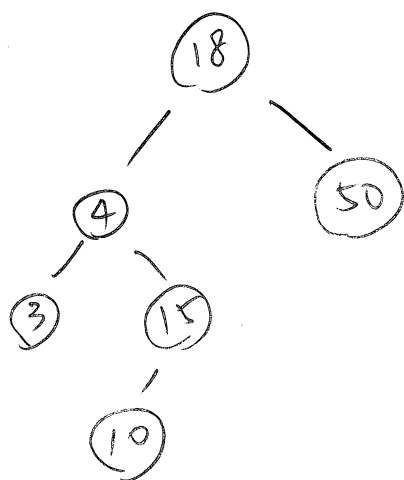
$$(15) = |1 - 0| = 1$$

$$(4) = |1 - 2| = 1$$

$$(30) = |0 - 1| = 1$$

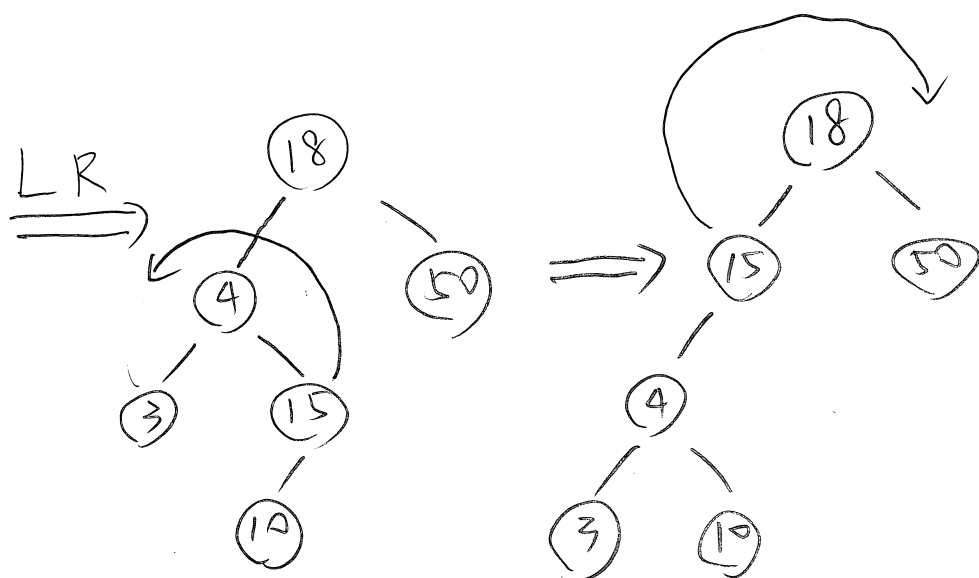
$$(18) = |3 - 2| = 1$$

(4) 剛 (30) =



繼承優先是從左子樹再右子樹，因 (30) 沒左子樹，由右子樹繼承。

明顯 (18) 權重: $|3 - 1| = 2$ ，不符合 AVL Tree。

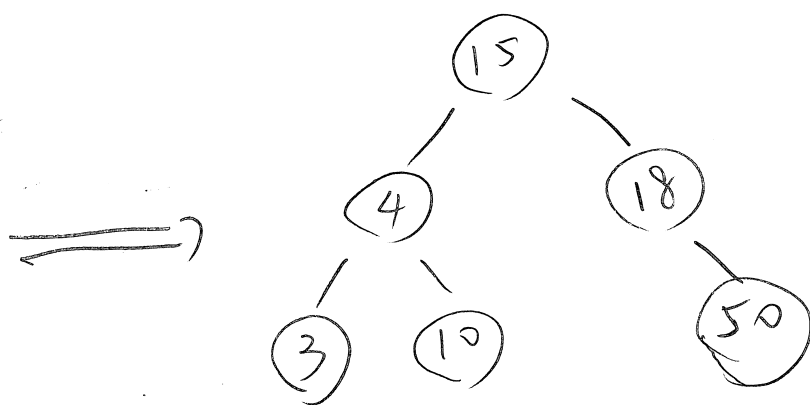


因 (3) 成為 (15)

的新左子樹，

(10) Insert (4)

裡，為 (4)



#