Adelson-Velsky and Landis Tree

定義：

又稱自平衡二元搜尋樹(AVL)，簡單來說就是，任一個節點的左子樹都比父節點小 ，右子樹都比父節點大。所以當我們要查找資料的時候，就可以從根節點開始，比根節點小的就從左子樹開始找，比較大的就從右子樹開始找。

相對於其他資料結構而言，尋找、插入的時間複雜度較低，為O(logN)。

使用：

我們可以先將資料建成二元搜尋樹，之後如果需要資料時，即可透透過此二元搜尋樹快速找到我們想要的資料，以降低我們查詢資料的時間，另外可以對他進行增加和刪除的動作。

操作：

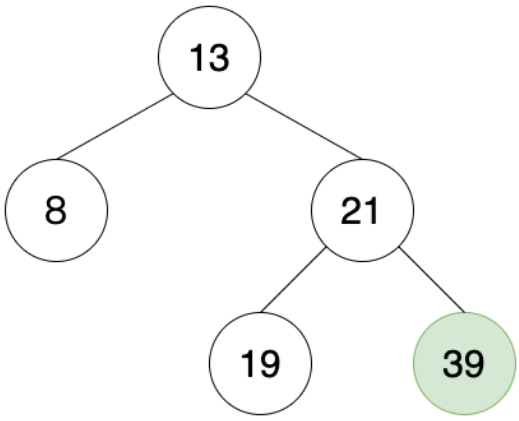
搜尋(一)

原則為依序比較比節點大或小，以下圖搜尋 39 為例

第一步：39 比 13 大，往 13 的右子樹走

第二步：39 比 21 大，往 21 的右子樹走

第三步：找到 39



搜尋(二)

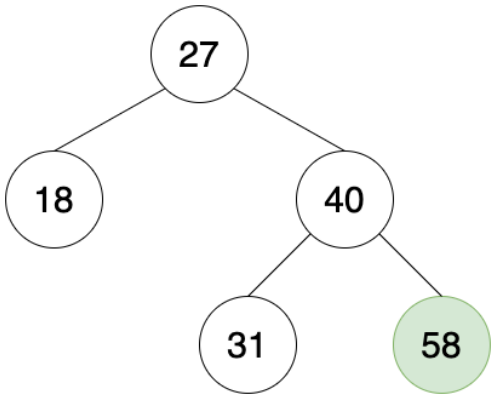
你可能會好奇假如找不到怎麼辦，請看下圖，假設要找 69

第一步：69 比 27 大，往 27 的右子樹走

第二步：69 比 40 大，往 40 的右子樹走

第三步：69 比 58 大，往 58 的右子樹走

第四步：58 沒有右子數，表示找不到，則回傳 null

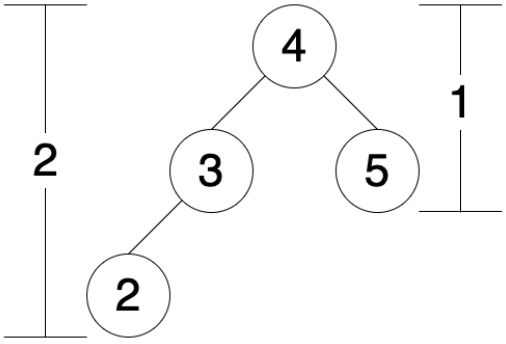


插入(規則)

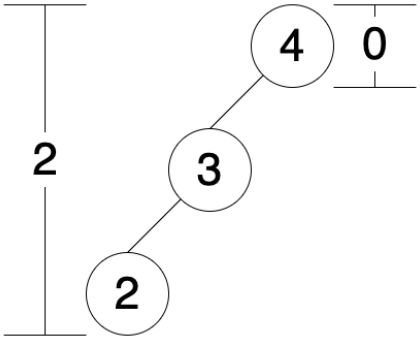
基於搜尋的規則，先找到適合插入的位置，之後再判斷是否要『旋轉』，旋轉是為了要達到平衡化，平衡的意思是左子樹的高度和右子樹的高度只能差 1，

請看下圖所示

圖中的左子樹高度為 2，右子樹高度為 1，兩著相差為 1，故為 AVL



圖中的左子樹高度為 2，右子樹高度為 0， 兩著相差為 2(大於 1)，故不為 AVL



旋轉又分為『LL 旋轉』,『RR 旋轉』,『LR 旋轉』,和『RL 旋轉』

而這些旋轉都圍繞著一個原則,『中間值向上提，大的放左小的放右』

插入(LL 旋轉)

請看下方兩張圖

因為 Node 5 的新增導致 Node 10 的不平衡，因此從 Node 10 開始標記兩個 L

因 Node 5 在 Node 10 的左邊，所以在 Node 10 到 Node 8 之間標記 L

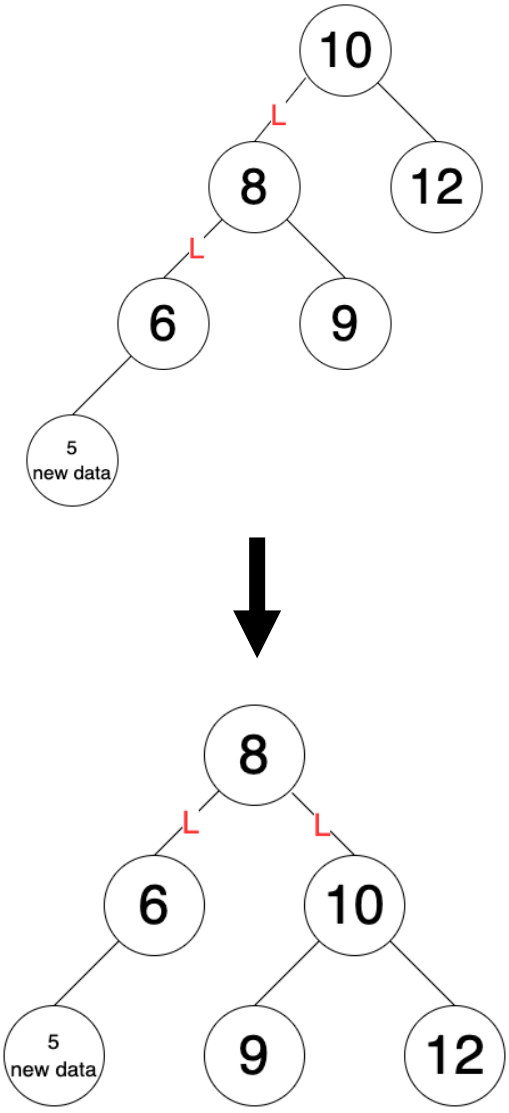
因 Node 5 在 Node 8 的左邊，所以在 Node 8 到 Node 6 之間標記 L

標記好之後就可以開始旋轉，目前被選取到的分別是 Node 10, Node 8, Node 6

而中間值為 Node 8，所以他向上提，左節點則為 Node 6，右節點則為 Node 10

你可能會想說啊其他節點像 Node 12 怎麼辦，別忘了 AVL 也是 BST 的一種

所以請依照 BST 的規則排好即可，做到這裡你已經完成 LL 旋轉了!



插入(RR 旋轉)

請看下方兩張圖

因為 Node 15 的新增導致 Node 8 的不平衡，因此從 Node 8 開始標記兩個 L

因 Node 15 在 Node 8 的右邊，所以在 Node 8 到 Node 10 之間標記 R

因 Node 15 在 Node 10 的左邊，所以在 Node 10 到 Node 12 之間標記 R

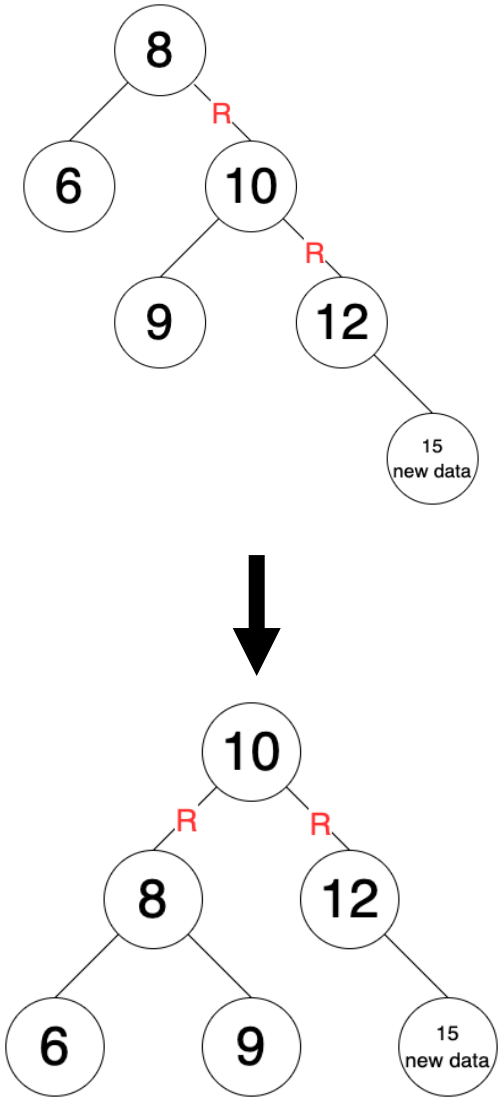
標記好之後就可以開始旋轉，目前被選取到的分別是 Node 8, Node 10, Node 12

而中間值為 Node 10，所以他向上提，左節點則為 Node 8，右節點則為 Node 12

你可能會想說啊其他節點像 Node 6 或 Node 9 怎麼辦，

別忘了 AVL 也是 BST 的一種

所以請依照 BST 的規則排好即可，做到這裡你已經完成 RR 旋轉了！



插入(LR 旋轉)

請看下方兩張圖

因為 Node 13 的新增導致 Node 15 的不平衡

因此從 Node 15 開始標記一個 L, 一個 R

因 Node 13 在 Node 15 的左邊，所以在 Node 8 到 Node 15 之間標記 L

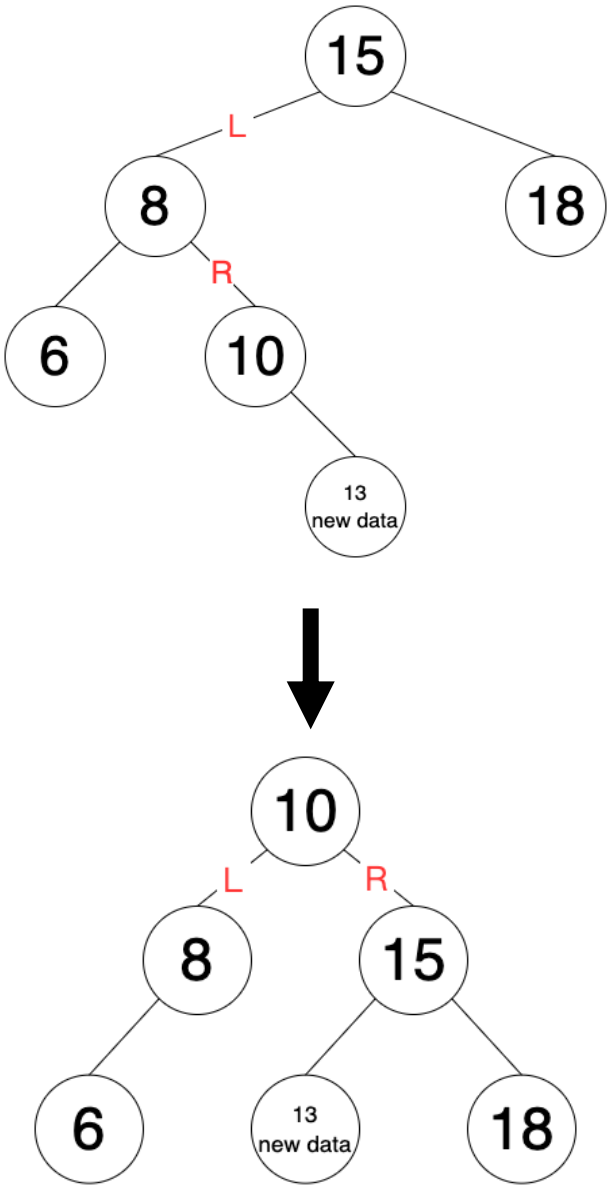
因 Node 13 在 Node 8 的左邊，所以在 Node 8 到 Node 10 之間標記 R

標記好之後就可以開始旋轉，目前被選取到的分別是 Node 8, Node 10, Node 15

而中間值為 Node 10，所以他向上提，左節點則為 Node 8，右節點則為 Node 15

你又可能會想說啊其他節點呢別忘了 AVL 也是 BST 的一種

所以請依照 BST 的規則排好即可，做到這裡你已經完成 LR 旋轉了！



插入(RL 旋轉)

請看下方兩張圖

因為 Node 14 的新增導致 Node 10 的不平衡

因此從 Node 10 開始標記一個 R，一個 L

因 Node 14 在 Node 10 的左邊，所以在 Node 10 到 Node 15 之間標記 R

因 Node 14 在 Node 15 的左邊，所以在 Node 15 到 Node 13 之間標記 L

標記好之後就可以開始旋轉，目前被選取到的分別是 Node 10, Node 13, Node 15

而中間值為 Node 13，所以他向上提

左節點則為 Node 10，右節點則為 Node 15

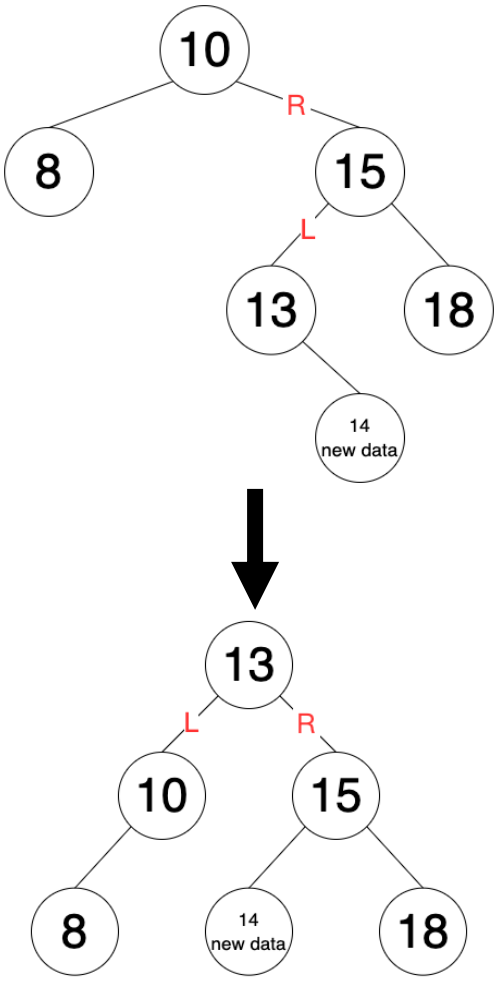
你可能會想說啊其他節點像 Node 8 或 Node 18 怎麼辦

Node 8 原本就是 Node 10 的左子點，不需要改動

Node 18 也是同樣的道理，他原本就是 Node 15 的因此不需要變

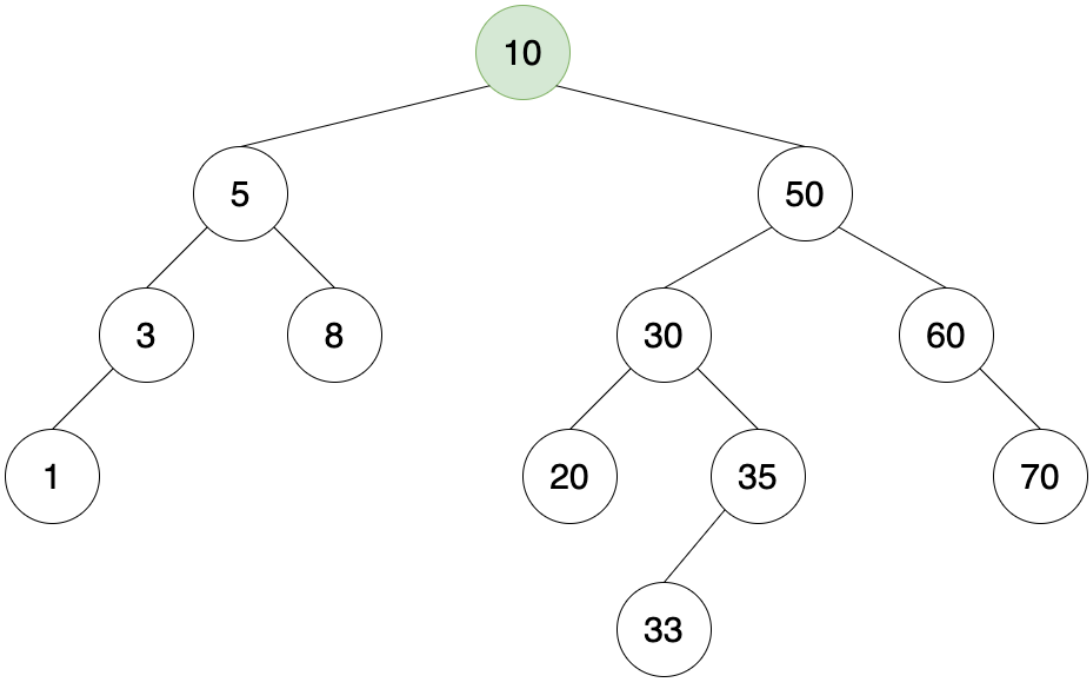
你又可能會想說啊其他節點呢別忘了 AVL 也是 BST 的一種

所以請依照 BST 的規則排好即可，做到這裡你已經完成 RL 旋轉了！

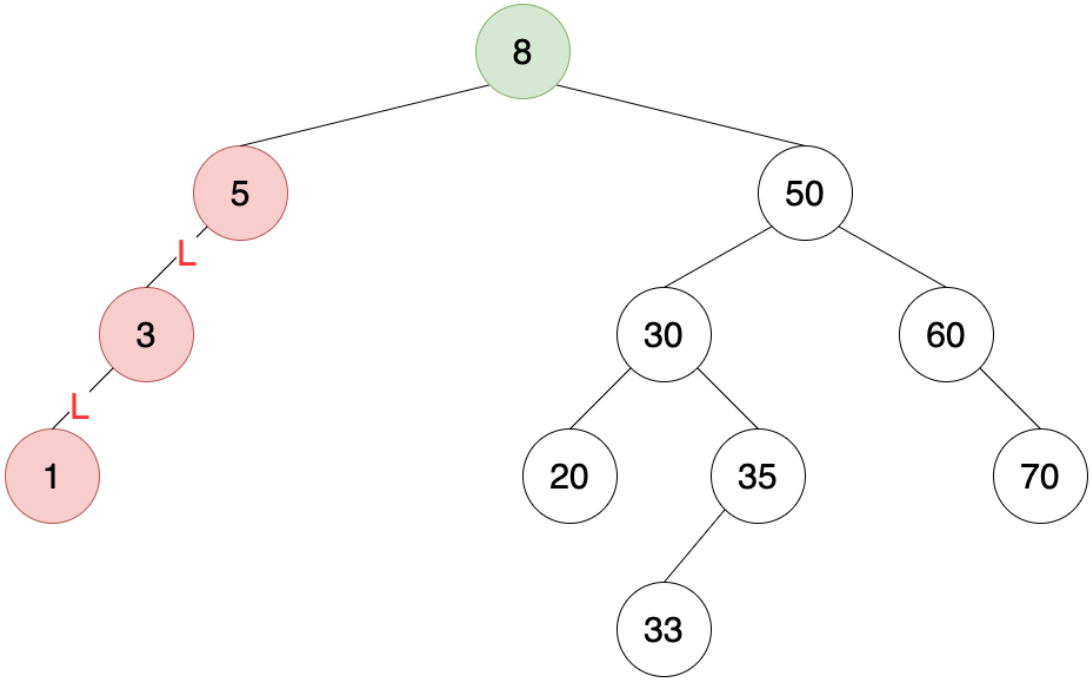


移除(規則)

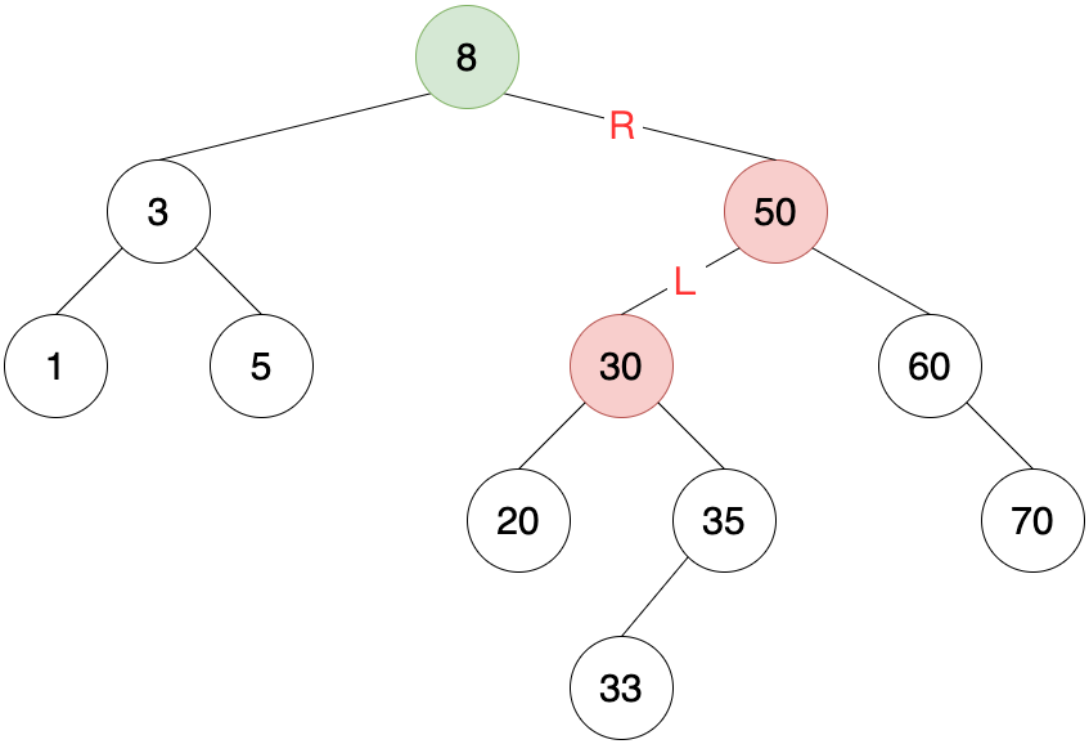
動作和 BST 的移除類似，只是多了要檢查是否移除會造成不平衡，若會造成不平衡記得旋轉，且可能會多次旋轉，請看下圖移除 10，且以左子樹最大取代



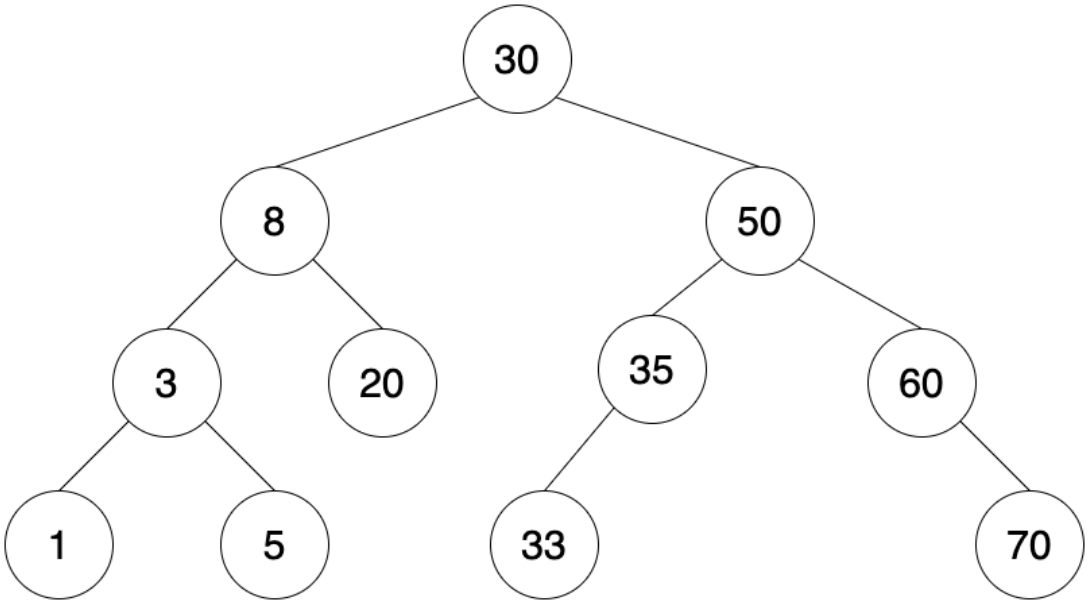
首先直接移除 10，且以左子樹最大 8 取代原本位置，但會造成 8 的左子樹不平衡，分別為 5, 3 和 1，因此需做 LL 旋轉



中間值為 3，因此 3 往上提 5 和 1 放兩邊，但這時 8 的右子數不平衡，因此需做 RL 旋轉



中間值為 30，因此 30 往上提，8 和 50 放兩旁，其餘依大的放右小的放左排好，這樣一來他就是一顆完整的 AVL 了



建立一顆自平衡二元搜尋樹

假設有筆資料為[16,21,35,54,68]，建成一顆二元搜尋樹該怎麼建呢?

我們可以把建立看成多次的插入，記住一個原則大的放右小的放左，且在買次的插入完後，壹定要檢查是否平衡。

第一步：16 當樹根

第二步：21 比 16 大，往 21 的右子樹放

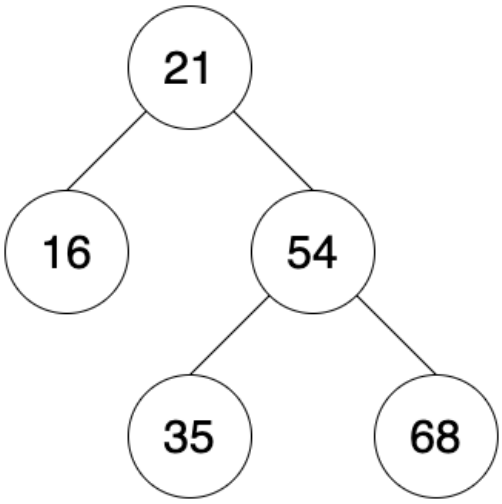
第三步：35 比 16 大，35 比 21 大，往 21 的左子樹放，但造成 16 的不平衡

要做 RR 旋轉

第四步：54 比 35 大，往 35 的右子樹放

第五步：68 比 35 大，68 比 54 大，往 54 的右子樹放，但造成 35 的不平衡

要做 RR 旋轉



中序

為一種走訪的順序，順序為拜訪左子樹(L)，印出該節點(D)，拜訪右子數(R)，每到一個新的節點，都會重複此動作，如果該節點沒有子樹，則走到下一步，起點皆為樹根，請看下圖

第一步：37 有左子樹，往 37 的左子樹走

第二步：11 有左子樹，往 11 的左子樹走

第三步：3 沒有左子樹，印出 3

第四步： 3 沒有右子樹，返回上一個，及為 11

第五步：印出 11，11 沒有右子樹，返回上一個，及為 37

第六步：印出 37，37 有右子樹，往 37 的右子樹走

第七步：46 有左子樹，往 46 的左子樹走

第八步：38 沒有左子樹，印出 38

第九步：38 沒有右子樹，返回上一個，及為 46

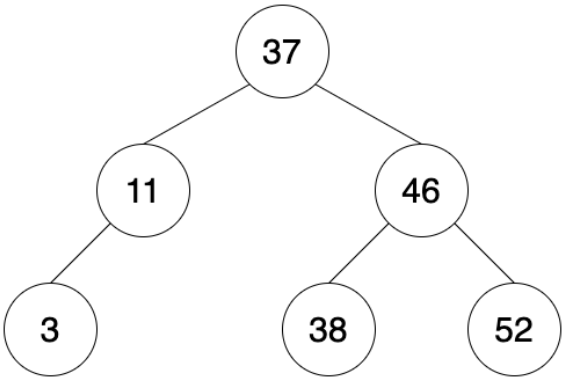
第十步：印出 46，46 有右子樹，往 46 的右子樹走

第十一步：52 沒有左子樹，印出 52

第十二步：52 沒有右子樹，結束中序走訪

因此這顆 AVL 的中序走訪為『3,11,37,38,46,52』，你可能會很訝異，剛好為由小到大的排序，這並不是剛好，而是二元搜尋樹的特性，

AVL 的中序走訪剛好為由小到大的順序



前序

是一種走訪的順序，順序為印出該節點(D)，拜訪左子樹(L)，拜訪右子數(R)，每到一個新的節點，都會重複此動作，如果該節點沒有子樹，則走到下一步，起點皆為樹根，請看下圖

第一步：印出 37，37 有左子樹，往 37 的左子樹走

第二步：印出 11, 11 有左子樹，往 11 的左子樹走

第三步：印出 3，3 沒有左子樹，也沒右子樹，返回上一個，及為 11

第四步：11 沒有右子樹，返回上一個，及為 37

第五步：37 有右子樹，往 37 的右子樹走

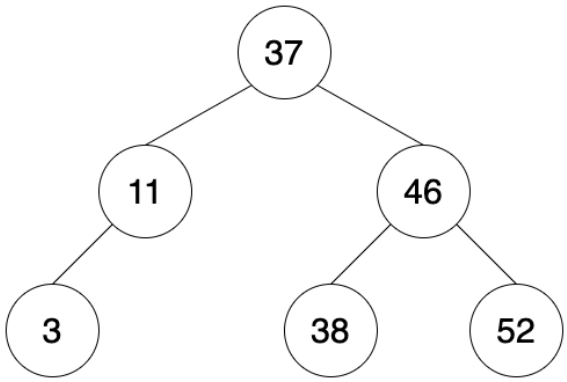
第六步：印出 46, 46 有左子樹，往 46 的左子樹走

第七步：印出 38, 38 沒有左子樹，也沒右子樹，返回上一個，及為 46

第八步：46 有右子樹，往 46 的右子樹走

第九步：印出 52，52 沒有左子樹，也沒右子樹，結束前序走訪

因此這顆 AVL 的前序走訪為『37,11,3,46,38,52』



後序

為一種走訪的順序，順序為拜訪左子樹(L) ，拜訪右子數(R)，印出該節點(D)，每到一個新的節點，都會重複此動作，如果該節點沒有子樹，則走到下一步，起點皆為樹根，請看下圖

第一步：37 有左子樹，往 37 的左子樹走

第二步：11 有左子樹，往 11 的左子樹走

第三步：3 沒有左子樹， 3 沒有右子樹，印出 3，返回上一個，及為 11

第四步：11 沒有右子樹，印出 11，返回上一個，及為 37

第五步：37 有右子樹，往 37 的右子樹走

第六步：46 有左子樹，往 46 的左子樹走

第七步：38 沒有左子樹，38 沒有右子樹，印出 38，返回上一個，及為 46

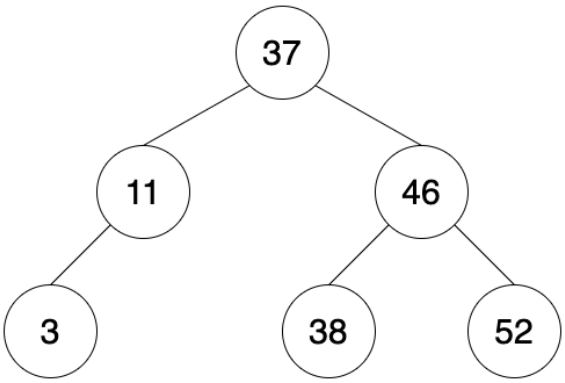
第八步：46 有右子樹，往 46 的右子樹走

第九步：52 沒有左子樹，52 沒有右子樹，印出 52，返回上一個，及為 46

第十步：印出 46，返回上一個，及為 37

第十一步：印出 37, 結束中序走訪

因此這顆 AVL 的後序走訪為『3,11,38,52,46,37』



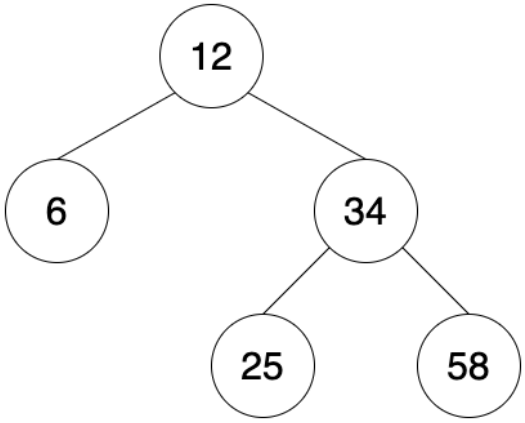
實際應用：

假設有筆資料為[12,34,6,25,58]，我們先將它建成 AVL，之後假設我們要找 58 這筆數據的話，我們只需要三部就可以找到了，分別是

第一步：58 比 12 大，往 12 的右子樹找

第二步：58 比 34 大，往 34 的右子樹找

第三步：此值剛好為 58，找到了



但假設我們的資料改為[6,12,25,34,58]，再依序建成 AVL 你覺得還會只需要三步就可找到 58 嗎？來看看下方的結果

答案是會的，因為它會自動平衡