Red BlackTree

定義：

又稱紅黑樹，簡單來說就是，他是 BST 和 AVL 的中間值

為什麼會這樣說，是因為 BST 可能會有最壞的情況發生，變成斜曲的二元樹

而 AVL 是為了避免這種情況的發生，嚴格執行平衡的動做，但相對付出的時間也就很多，而紅黑數則是不那麼要求平衡，你可以想成說他犧牲一點平衡去換來時間跟效率，他的尋找、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紅黑樹，之後如果需要資料時，即可透透過此紅黑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：

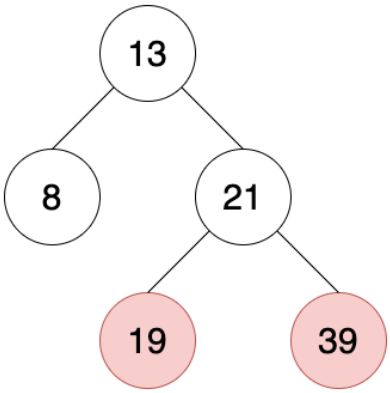
搜尋(一)

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)

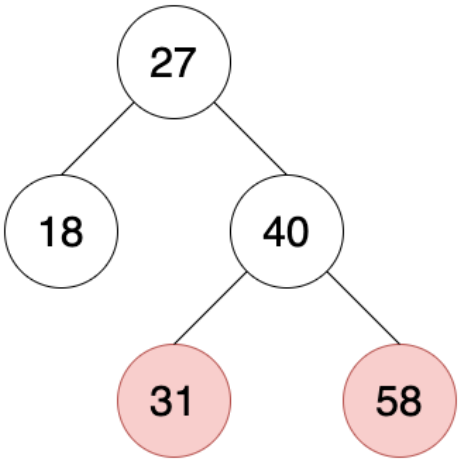
你可能會好奇假如找不到怎麼辦，請看下圖，假設要找 69

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，而新增的節點先標紅色，且在插入的時候，若發現某個 Node 兩個子點是紅色 Node 的話，則做 color change，再判斷是否要『旋轉』，旋轉是為了要達到符合『紅黑數的五大條件』，分別為

一. Node 必為黑色或紅色

二. root(跟節點)必為黑色

三. null(空節點)必為黑色

四. 不會有連續兩個紅色節點

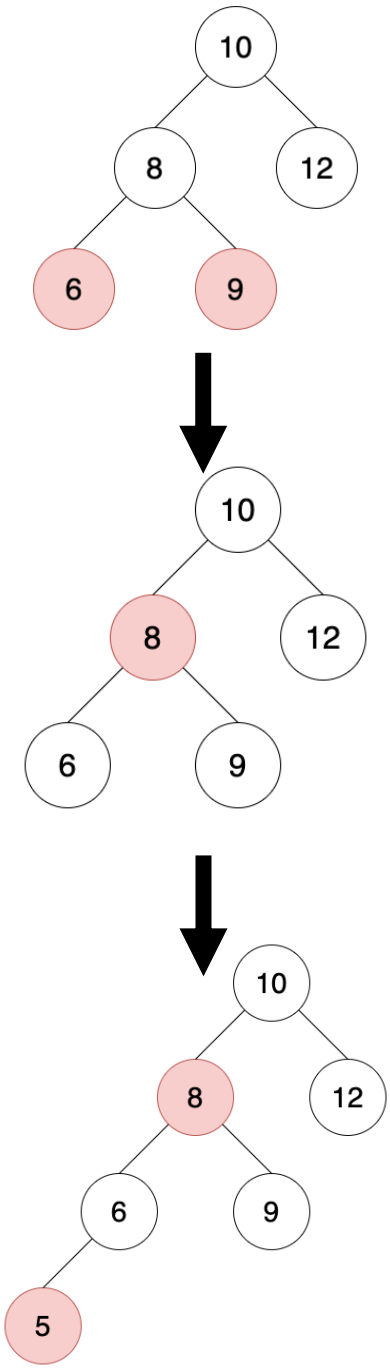
五. 到每個 leaf 上的黑色節點數是一樣的

旋轉又分為『LL 旋轉』,『RR 旋轉』,『LR 旋轉』,和『RL 旋轉』，和 AVL 的旋轉差不多，只是加上顏色的變化，而這些旋轉都圍繞著一個原則，『中間值向上提標黑，大的放左小的放右標紅』

插入(color change)

請看下方兩張圖

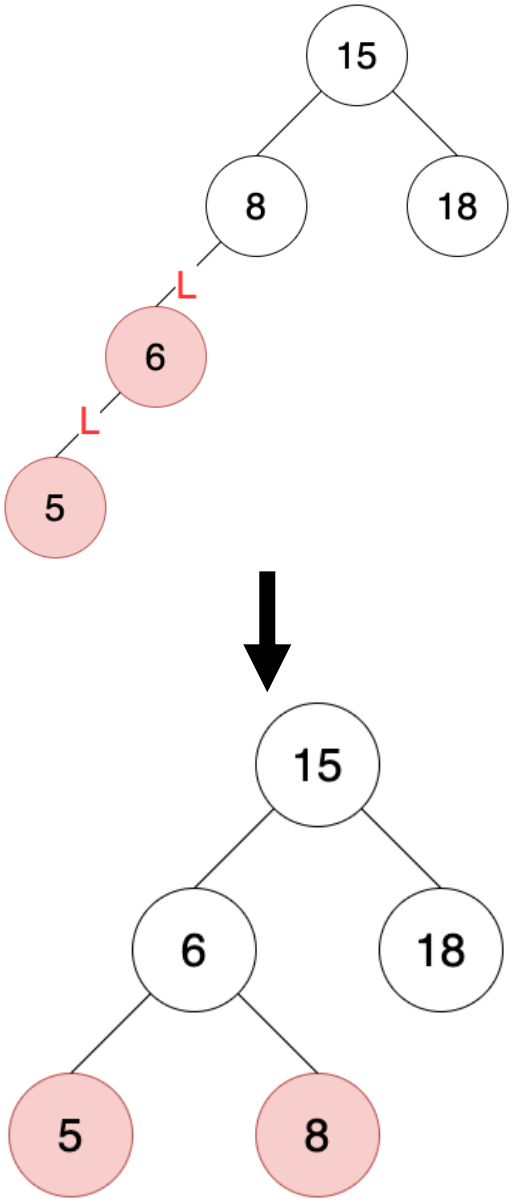
因為新增 Node 5 的時候，必須先搜尋要插入的位置，搜尋的過程中發現，有 Node 8 的兩個子點為紅色，因此必須做 color change，而 color change 的作法為『該 Node 改為紅色，子點改為黑色』，並且檢查有無連續的紅節點，若無才可進祥下一步，因此我們可以插入 Node 5 了，插入完還要檢查是否有違反規則



插入(LL 旋轉)

請看下方兩張圖

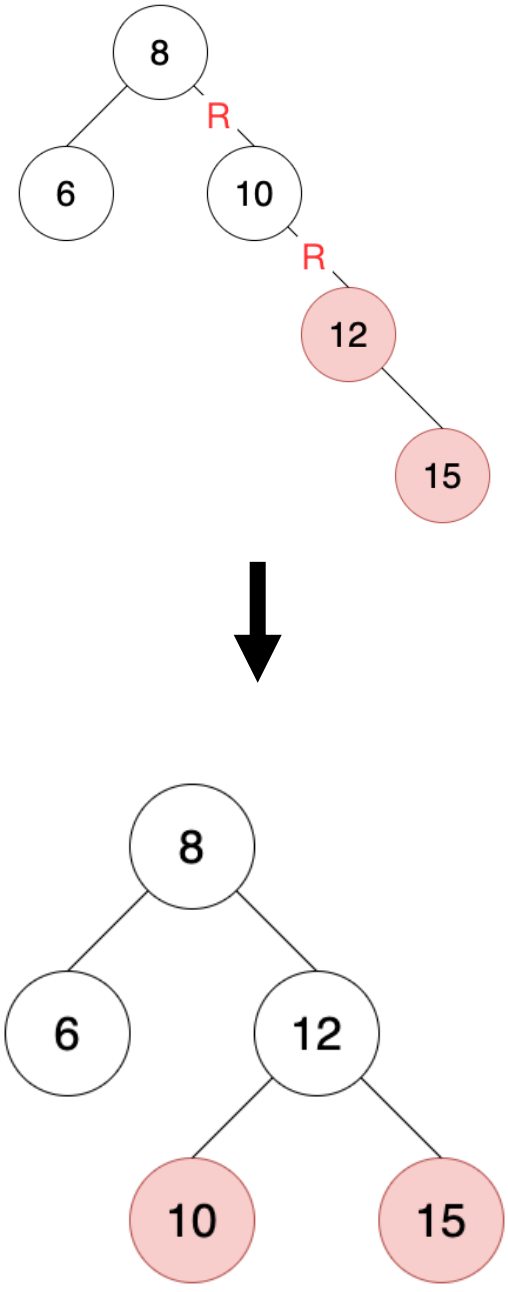
因為新增 Node 5 的時候，必須先搜尋要插入的位置，搜尋的過程中沒發現有 Node 的兩個子點為紅色，可進行下一步 Node 5 的插入，但插入完後發現有連續的紅節點，從違反規則的地方，由後往前算分別為 Node 5,Node 6，Node 8，因此需做 LL 旋轉，中間值為 Node 6 往上提標黑，左節點則為 Node 5 標紅，右節點則為 Node 8 標紅，做完之後需檢查有無違反規則，若皆無違反的話，恭喜你完成 LL 旋轉了!



插入(RR 旋轉)

請看下方兩張圖

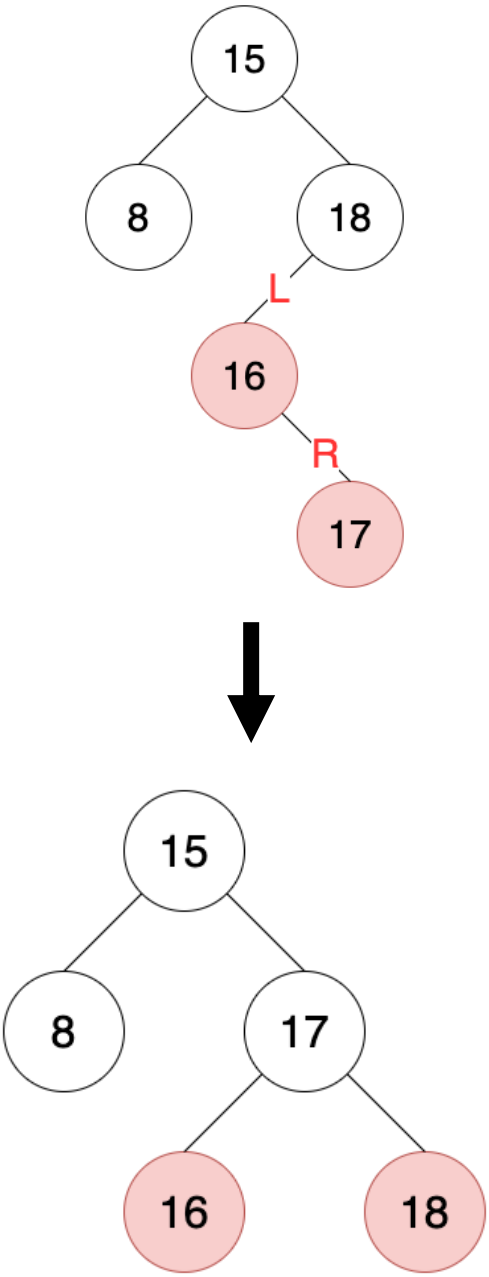
因為新增 Node 15 的時候，必須先搜尋要插入的位置，搜尋的過程中沒發現有 Node 的兩個子點為紅色，可進行下一步 Node 15 的插入，但插入完後發現有連續的紅節點，從違反規則的地方，由後往前算分別為 Node 15,Node 12，Node 10，因此需做 RR 旋轉，中間值為 Node 12 往上提標黑，左節點則為 Node 10 標紅，右節點則為 Node 15 標紅，做完之後需檢查有無違反規則，若皆無違反的話，恭喜你完成 RR 旋轉了！



插入(LR 旋轉)

請看下方兩張圖

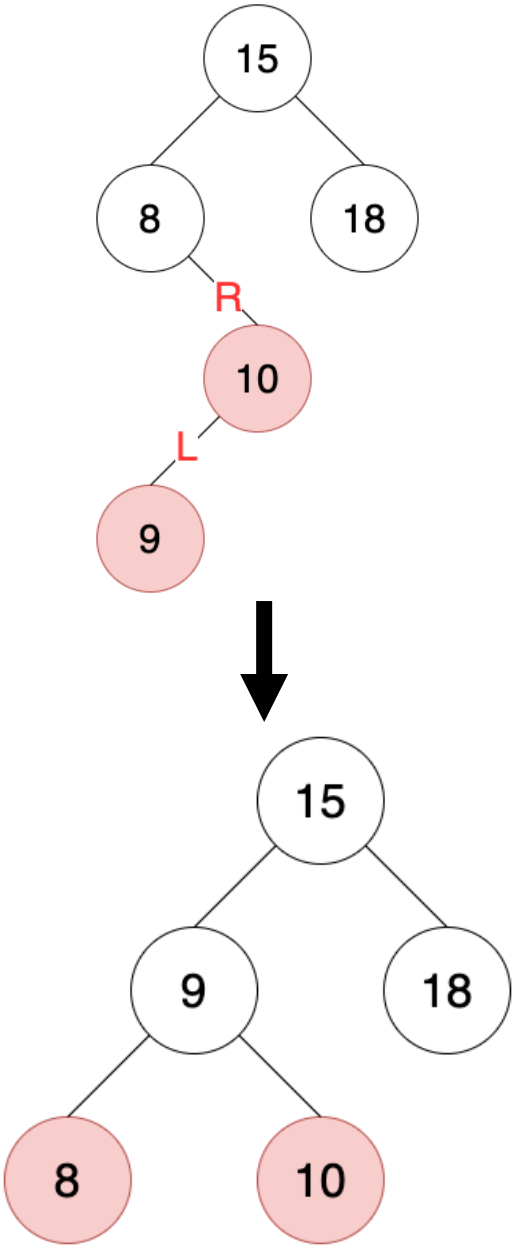
因為新增 Node 17 的時候，必須先搜尋要插入的位置，搜尋的過程中沒發現有 Node 的兩個子點為紅色，可進行下一步 Node 17 的插入，但插入完後發現有連續的紅節點，從違反規則的地方，由後往前算分別為 Node 17,Node 16，Node 18，因此需做 LR 旋轉，中間值為 Node 17 往上提標黑，左節點則為 Node 16 標紅，右節點則為 Node 18 標紅，做完之後需檢查有無違反規則，若皆無違反的話，恭喜你完成 LR 旋轉了！



插入(RL 旋轉)

請看下方兩張圖

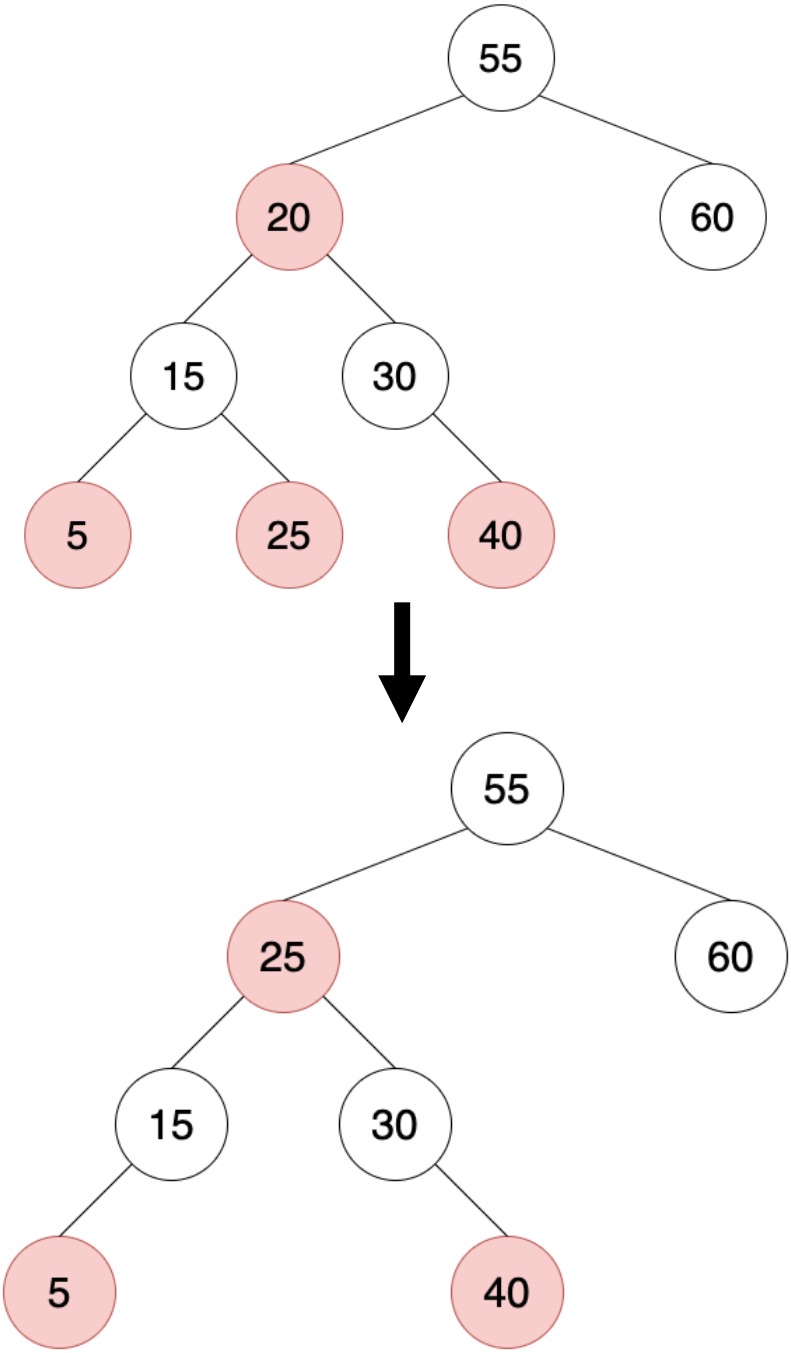
因為新增 Node 9 的時候，必須先搜尋要插入的位置，搜尋的過程中沒發現有 Node 的兩個子點為紅色，可進行下一步 Node 9 的插入，但插入完後發現有連續的紅節點，從違反規則的地方，由後往前算分別為 Node 9,Node 10，Node 8，因此需做 RL 旋轉，中間值為 Node 9 往上提標黑，左節點則為 Node 8 標紅，右節點則為 Node 10 標紅，做完之後需檢查有無違反規則，若皆無違反的話，恭喜你完成 RL 旋轉了！



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違反規則，若會造成違反規則記得旋轉，請看下圖移除 20，且以左子樹最大取代

首先直接移除 20，並且以 25 取代原本位置，沒有違反任何規則



建立一顆紅黑數

假設有筆資料為[40,60,55,15,20,5,25,30]，建成一顆紅黑數該怎麼建呢?

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，且在每次的插入完後，一定要檢查是否符合規則。

第一步：40 當樹根標黑，因 root 必為黑

第二步；60 比 40 大，往 40 右子樹放標紅

第三步：55 比 40 大，55 比 60 小，往 60 的左子樹放，但違反『連續兩紅色節點』，因此需做 RL 旋轉

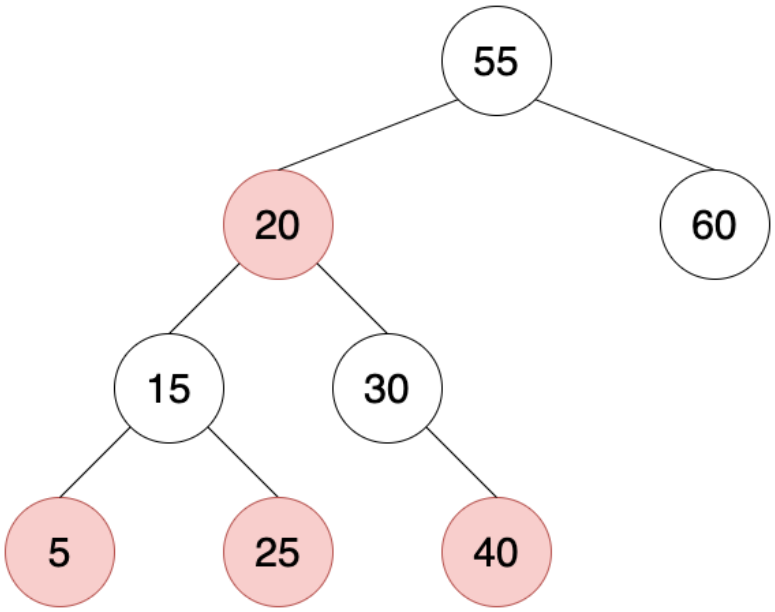
第四步：在 15 搜尋插入點時，發現 55 的兩節點為紅色，需做 color change，因此 40 和 60 改為黑色，但 55 為 root，因此標黑，再插入 15 標紅

第五步：20 比 55 小，20 比 40 小，20 比 15 大，往 15 的右子樹放，但違反『連續兩紅色節點』，因此需做 LR 旋轉

第六步：在 5 搜尋插入點時，發現 20 的兩節點為紅色，需做 color change，因此 15 和 40 改為黑色，20 標紅，再插入 5 標紅

第七步：25 比 55 小，25 比 20 大，25 比 40 小，往 40 的左子樹放

第八步：30 比 55 小，30 比 20 大，30 比 40 小，30 比 25 大，往 25 的右子樹放，但違反『連續兩紅色節點』，因此需做 LR 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

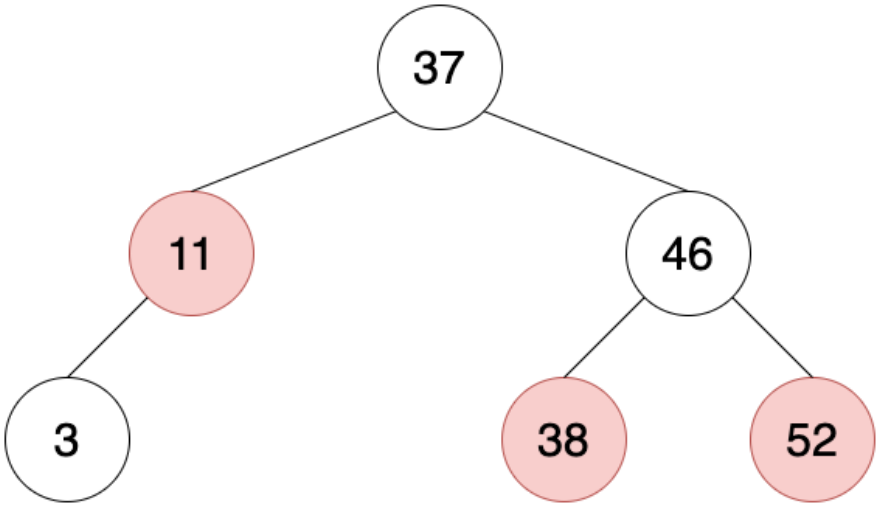
第十步：印出 46，46 有右子樹，往 46 的右子樹走

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

因此這顆紅黑數的中序走訪為『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，

紅黑樹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

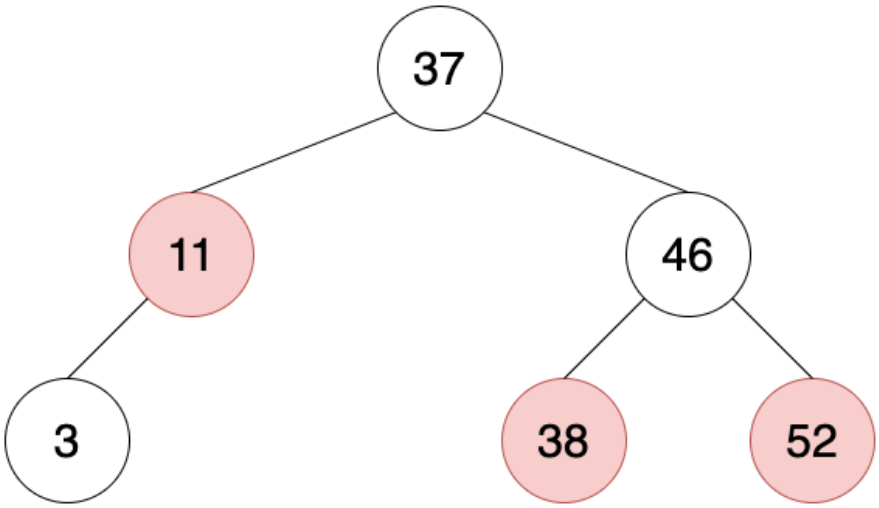
第六步：印出 46, 46 有左子樹，往 46 的左子樹走

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

因此這顆紅黑樹的前序走訪為『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

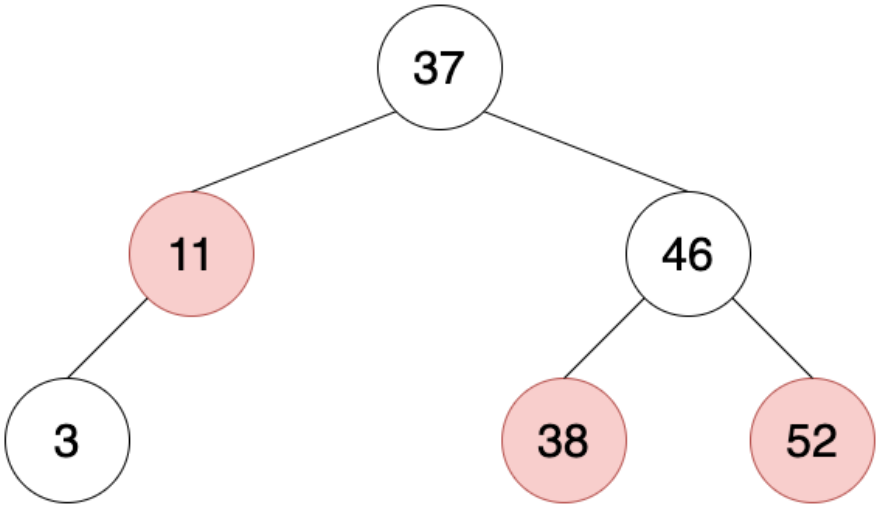
第八步：46 有右子樹，往 46 的右子樹走

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

因此這顆紅黑樹的後序走訪為『3,11,38,52,46,37』



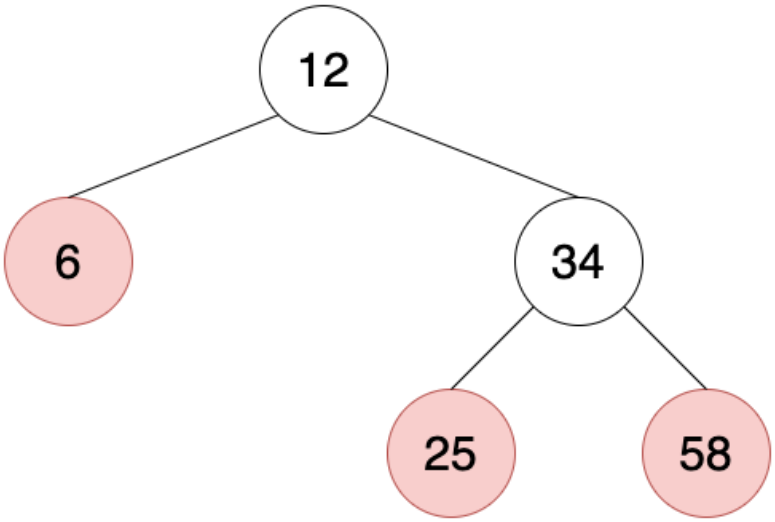
實際應用：

假設有筆資料為[12,34,6,25,58]，我們先將它建成紅黑樹，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]，再依序建成紅黑樹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，因為它會平衡