Binary Search Tree

定義：

又稱二元搜尋樹，簡單來說就是，任一個節點的左子節點都比父節點小 ，右子節點都比父節點大。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：

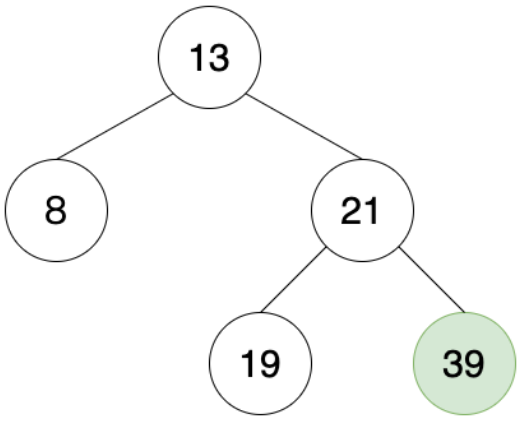
搜尋(一)

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)

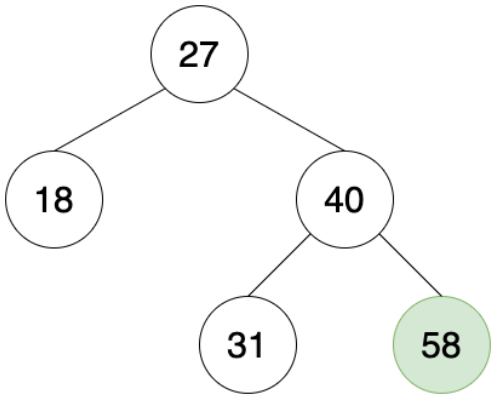
你可能會好奇假如找不到怎麼辦，請看下圖，假設要找 69

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

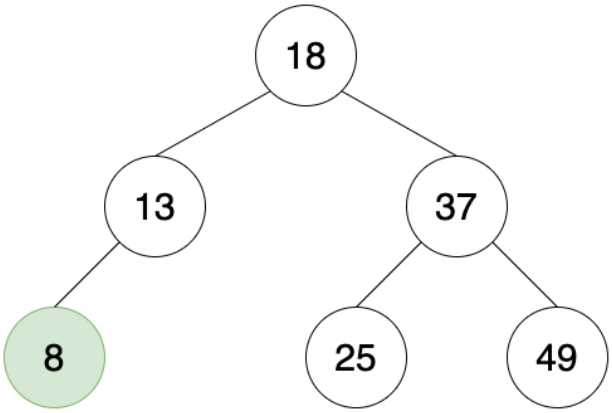


插入

基於搜尋的規則，先找到適合插入的位置，之後再以大的放右小的放左的規則執行，請看下方圖示插入 8 為何

第一步：8 比 18 小，往 18 的右子樹走

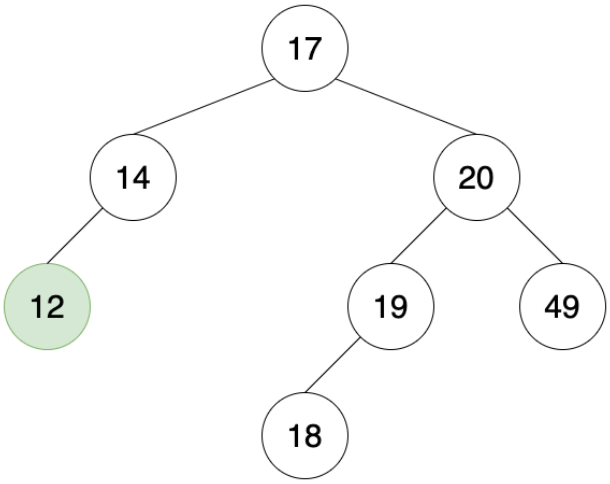
第二步：8 比 13 小，往 13 的右子樹放



移除

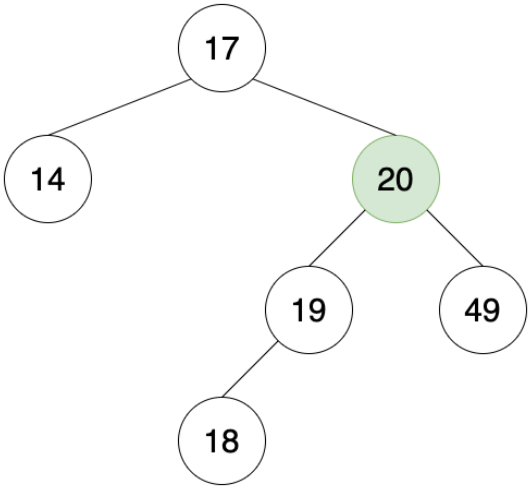
原則為若為 leaf 則直接移除，若有子節點則以左子樹最大或右子樹最小代替原本位置，其餘依序排成二元搜尋樹，請看下方圖示移除 12 和 20 為何

因為 12 為 leaf，所以直接移除

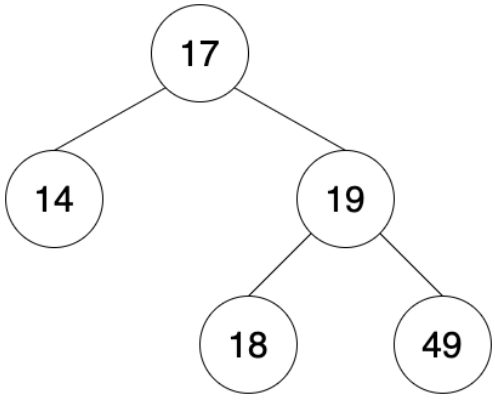


但 20 有子節點，所以必須選擇要以左子樹最大或右子樹最小替換原本位置

這裡以左子樹最大舉例



所以 19 替換到原本位址，其餘皆按照大的放右小的放左的原則排列



建立一顆二元搜尋樹

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

記住一個原則大的放右小的放左即可，我們以第一個當作根節點。

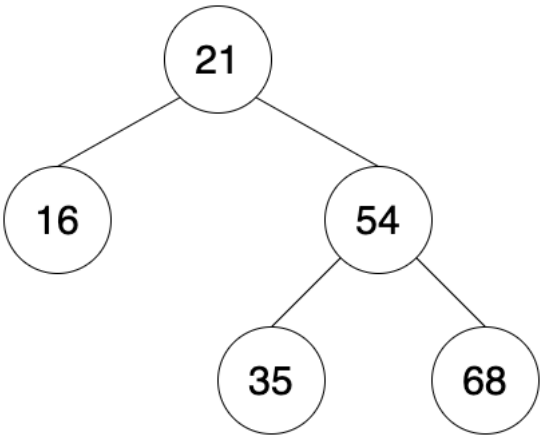
第一步：21 當樹根

第二步：54 比 21 大，往 21 的右子樹放

第三步：16 比 21 小，往 12 的左子樹放

第四步：35 比 21 大，35 比 54 小，往 54 的左子樹放

第五步：68 比 21 大，69 比 54 大，往 54 的右子樹放



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

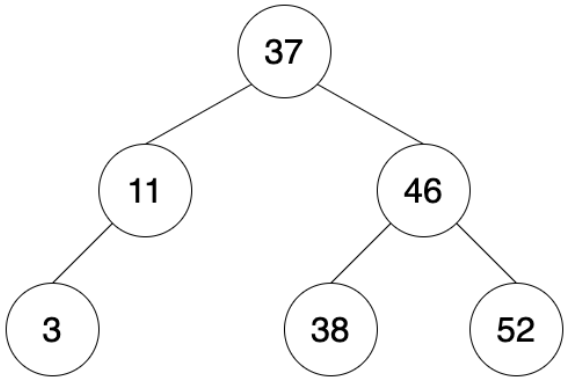
第十步：印出 46，46 有右子樹，往 46 的右子樹走

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

因此這顆二元搜尋樹的中序走訪為『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，

二元搜尋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

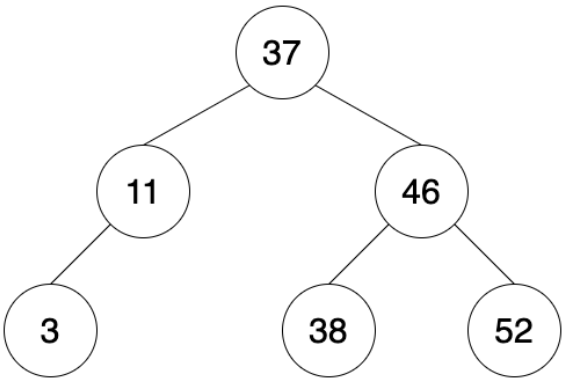
第六步：印出 46, 46 有左子樹，往 46 的左子樹走

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

因此這顆二元搜尋樹的前序走訪為『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

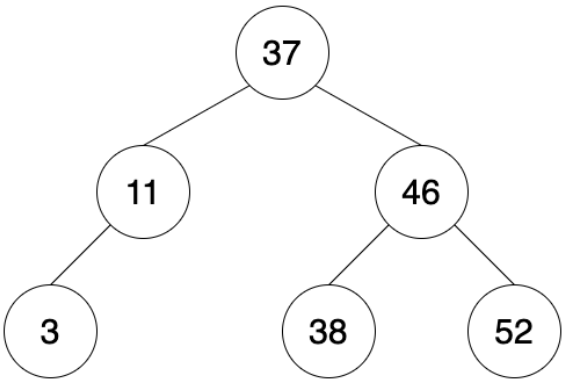
第八步：46 有右子樹，往 46 的右子樹走

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

因此這顆二元搜尋樹的後序走訪為『3,11,38,52,46,37』



總結，二元搜尋樹的中序走訪及為由小到大的順序，而前序的話則為先印再往左再往友，而後序則為先往左再往右最後再印，多練習很快你就能跟我一樣，很快就能寫出答案了。

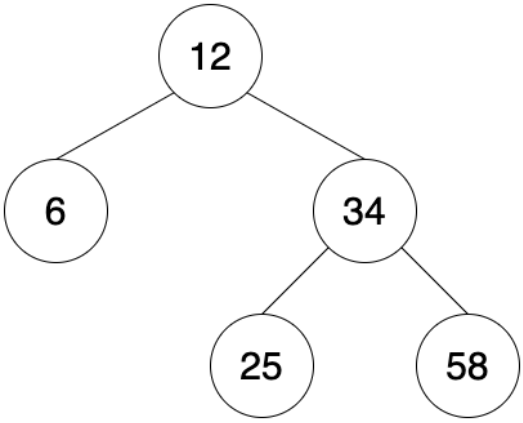
實際應用：

假設有筆資料為[12,34,6,25,58]，我們先將它建成二元搜尋樹，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]，再依序建成二元搜尋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

如果要找到 58 的話，需要 5 步才能找到，這種樹我們稱為『斜曲二元樹』，他根本沒有優化我們搜尋的過程。