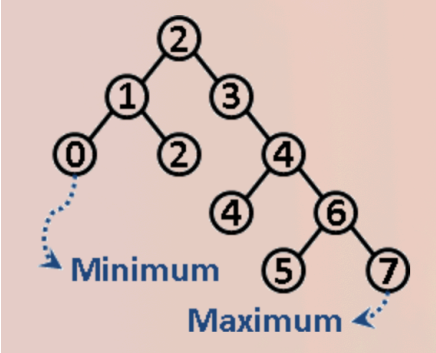
Binary Search Tree 原理

1. Binary Search Tree是Binary Tree排列的資料結構。
2. 稱「二元搜尋樹」，置放大量數字並且進行排序的資料結構。
3. 應用範圍很廣，可以利用在搜索、排序和提供資料集合基本結構，發展其他資料結構，所以也是重要的資料結構之一。
4. 原理是 Divide and Conquer ，樹根中，左子樹較小或相等，右子樹較大，然後遞迴分割下去。
5. 性質:

    若任意節點的左子樹不空，則左子樹上所有節點的值均小於它的根節點的值。
    若任意節點的右子樹不空，則右子樹上所有節點的值均大於它的根節點的值。
    任意節點的左、右子樹也分別為二元搜尋樹。

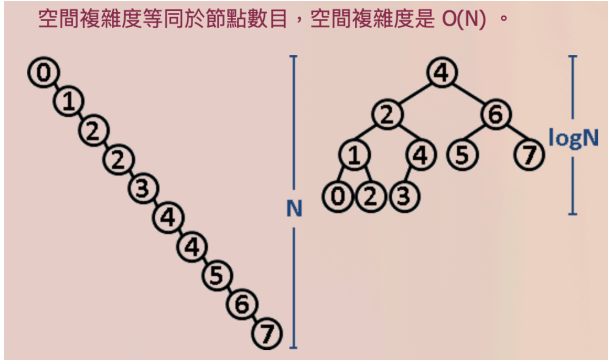
- 二元搜尋樹相比於其他資料結構的優勢在於尋找、插入的時間複雜度較低，為O(log n)。二元搜尋樹是基礎性資料結構，用於構建更為抽象的資料結構，如集合、多重集、關聯陣列等。

演算法	平均	最差
空間	$O(n)$	$O(n)$
搜尋	$O(\log n)$	$O(n)$
插入	$O(\log n)$	$O(n)$
刪除	$O(\log n)$	$O(n)$

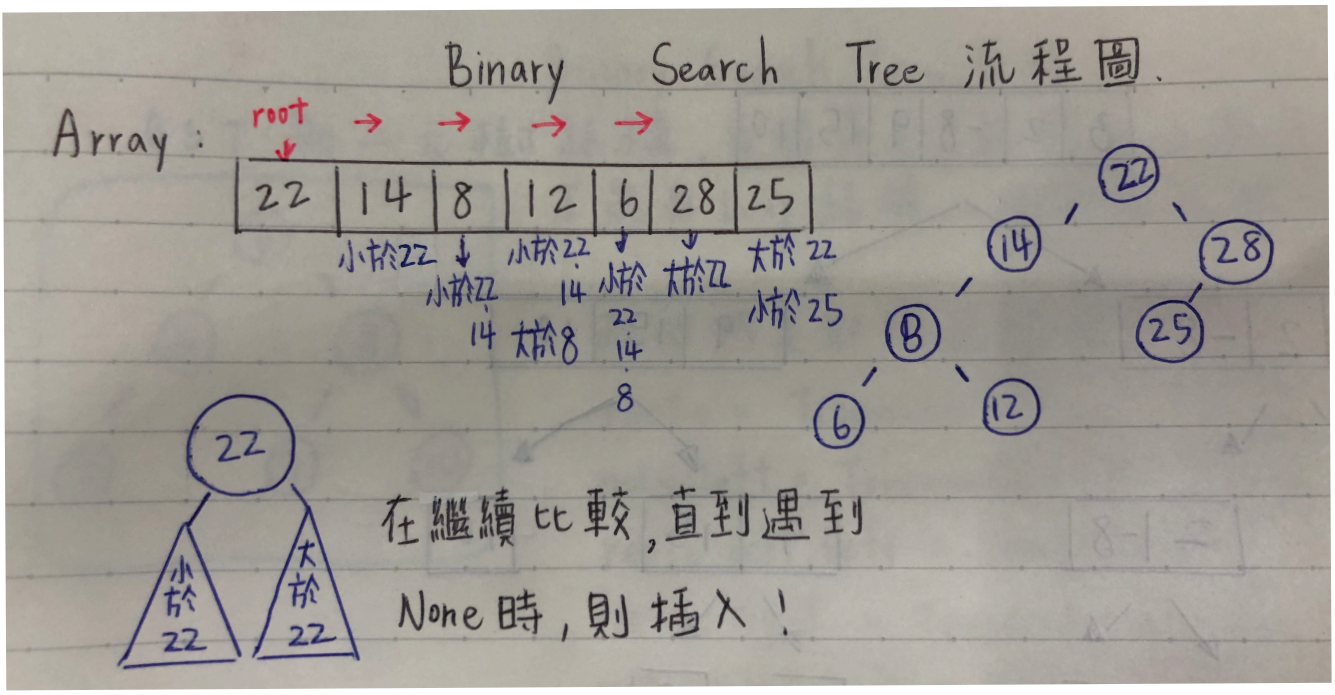


- 最小節點：從樹根開始往左小孩走到底，最大節點：從樹根開始往右小孩走到底。
- 時間複雜度等同於二元搜尋樹的高度。

- 當樹不平衡時，會需要產生非常巨大的陣列，很容易就出現記憶體不足的現象，又加上刪除節點時需要重新調整陣列的元素位置很多，實作上不容易效能也較差，所以一般使用連結串列的方式實作較好。



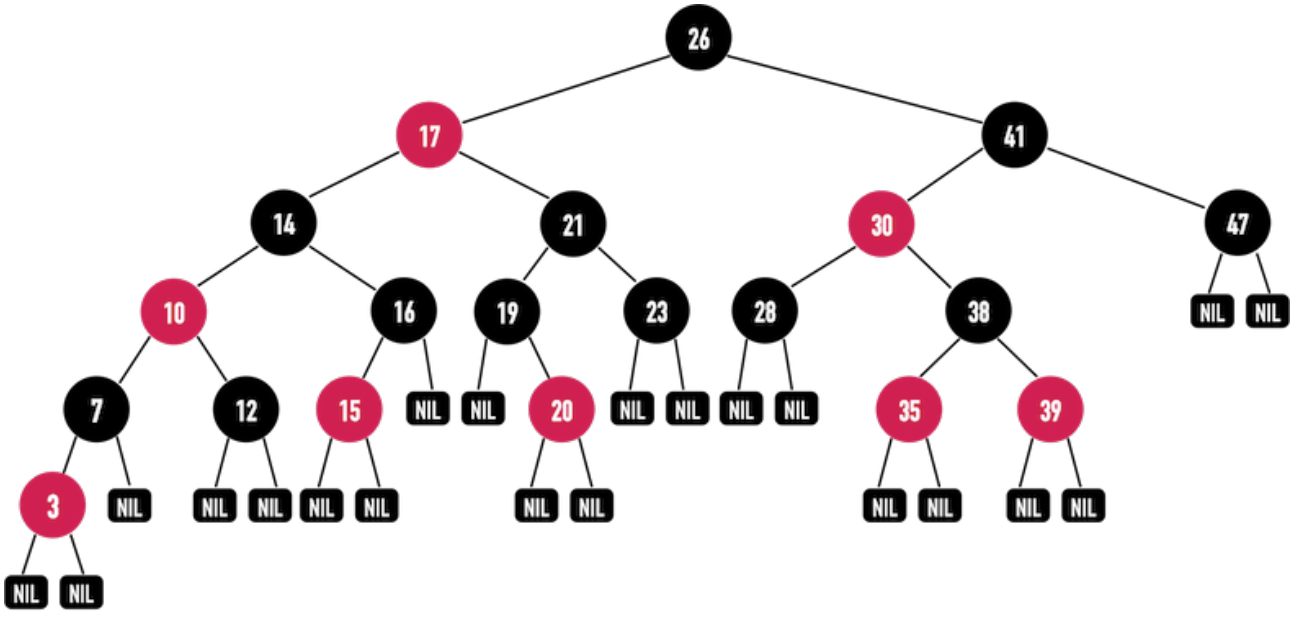
Binary Search Tree 流程圖



Red Black Tree 原理

Red Black Tree的特徵

Red Black Tree(RBT)是node塗了「顏色」的Binary Search Tree(BST)，藉由控制顏色，能夠保證在RBT中，**最長path(路徑)不會超過最短path的兩倍**(若最短的path是5，最長的path至多只能是10），如此，RBT便能夠近似地視為平衡，如圖。



圖四中，所有原本在BST中指向NULL的pointer，在RBT中，全部指向了NIL。什麼是NIL？NIL是永遠為黑色、並且實際占有記憶體的node，因為有配置記憶體，因此能夠以Node->color的方式取得某個node之顏色(若使用NULL則無法)，這項設計將在後續介紹如何於RBT中Insert(新增資料)與Delete(刪除資料)時派上用場。
接著來看RBT的五項特徵：

1. RBT中的每一個node不是黑色就是紅色。
2. root一定是黑色。
3. 每一個leaf node(也就是NIL)一定是黑色。
4. 如果某個node是紅色，那麼其兩個child必定是黑色，不能有兩個紅色node相連，如圖四中的node(17)、node(30)。
    - 若某個node為黑色，其child之顏色沒有限制，如圖四中的node(38)、node(26)、node(21)。
5. 站在任何一個node上，所有從該node走到其任意descendant leaf的path上之黑色node數必定相同。

根據上述特徵的第四點與第五點，RBT中path可能的長度最小值一定是全部node皆為黑色(如圖四最右path)，而path可能的長度最大值並定是紅色-黑色相間(如圖四最左path)，如此便確保RBT擁有**最長path(路徑)不會超過最短path的兩倍**的特性。