程式設計
Ch18. Binary Search Tree

Chuan-Chi Lai 賴傳淇

Department of Communications Engineering
National Chung Cheng University

Spring Semester, 2024

Outline

1. 二元搜尋樹簡介 (Introduction to Binary Search Tree)
2. 二元搜尋樹的搜尋 (Search of Binary Search Tree)
3. 二元搜尋樹的插入與刪除 (Insert and Delete of a Binary Search Tree)
4. 總結 (Summary)

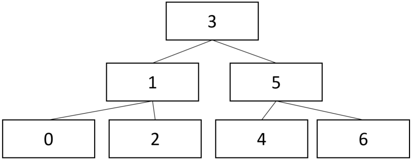
二元搜尋樹簡介
Introduction to Binary Search Tree

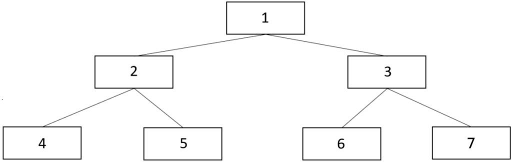
二元搜尋樹簡介

- 二元搜尋樹是一種二元樹，但有著特殊規則：
  - 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。
  - 沒有鍵值相等的節點。

二元搜尋樹簡介

- 在右邊兩個二元樹中，上面的為二元搜尋樹，下面的則為非





二元搜尋樹的搜尋
Search of Binary Search Tree

二元搜尋樹的搜尋

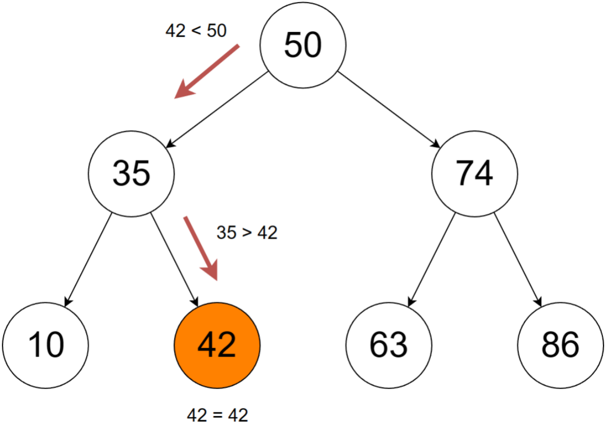
- 不同於二元樹的搜尋方式，由於二元搜尋樹的規則，我們會將較小的值放在左子樹，較大的值放在右子樹。
- 了解這個規則後，我們在搜索時可以通過當前節點的值，來確定繼續向左子樹搜尋或是向右子樹搜尋，就不需要兩邊子樹都做遍歷了。
- 時間複雜度為 $O(h)$, 其中 h 為樹高。

二元搜尋樹的搜尋

- 二元搜尋樹的搜尋步驟如下:
  1. 從根節點開始，將要查找的值與當前節點的值進行比較。
  2. 如果要查找的值等於當前節點的值，那麼找到了目標值，搜索結束。
  3. 如果要查找的值小於當前節點的值，根據二元搜尋樹的特性，目標值應該位於當前節點的左子樹中。因此繼續以當前節點的左子節點作為新的當前節點，重複步驟 1。
  4. 如果要查找的值大於當前節點的值，根據二元搜尋樹的特性，目標值應該位於當前節點的右子樹中。因此繼續以當前節點的右子節點作為新的當前節點，重複步驟 1。
  5. 如果在某個節點處找不到目標值，並且該節點沒有左子樹或右子樹，則說明目標值不存在於二元搜尋樹中。
  6. 可以選擇在到達葉子節點（沒有子節點）時停止搜索，或者繼續搜索到空節點為止。

二元搜尋樹的搜尋

- 示意圖如下，假設我們要找的數字為 42。



二元搜尋樹的搜尋

- 二元搜尋樹的搜尋程式碼範例如下：

```c
struct Node* BST_search(struct Node* root, int value) {
    if (root == NULL || root->data == value) {
        return root;
    }

    if (value < root->data) {
        return BST_search(root->left, value);
    }
    else {
        return BST_search(root->right, value);
    }
}
```

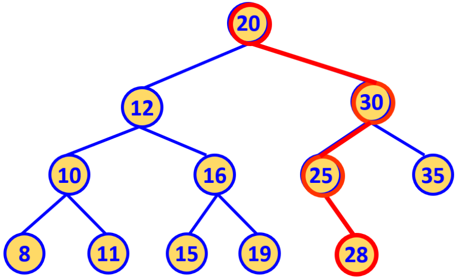
二元搜尋樹的插入與刪除
Insert and Delete of a Binary Search Tree

二元搜尋樹的插入與刪除

- 在二元搜尋樹中，因為我們要遵循特定的規則，不能隨便更改樹的結構，於是我們就需要用一些技術來達成二元搜尋樹的插入與刪除。

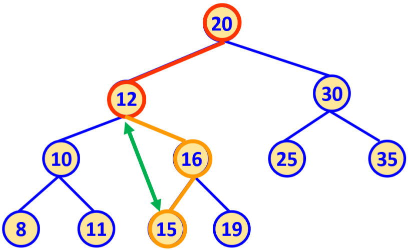
二元搜尋樹插入節點

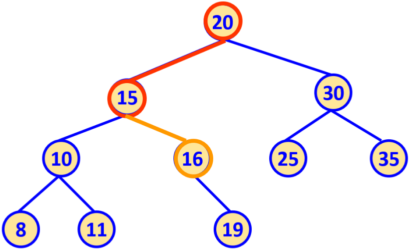
- 在插入的過程中，我們僅要注意插入元素的值一定比左邊大，也一定比右邊小，從根往下尋找到最下層並插入即可。
- 以下圖插入節點 28 為例：



二元搜尋樹刪除節點

- 而在刪除節點是就需要多一點步驟了，若想要刪除節點為葉節點，可直接刪除。
- 但若想要刪除節點為內部節點，則須先找到其右 (左) 子樹中最小 (大) 的值做交換位置，再進行刪除。
- 以下圖刪除節點 12 為例：





總結
Summary

總結 (1/2)

- 二元搜尋樹的高度決定了插入與刪除節點的時間複雜度。
- 然而，二元搜尋樹的高度會取決於資料節點的插入順序。
- 如果依序插入的節點是已排序過 (遞增或遞減) 的資料，二元搜尋樹會成為 (右或左) 歪斜樹。對於二元搜尋樹而言，此情況為 worst case，若此時插入新節點或刪除節點，時間複雜度為 $O(n)$。
- 如果插入的節點是隨機順序 (無排序過) 的資料，此情況下插入新節點或刪除節點，平均時間複雜度為 $O(\log n)$。
- Worst case 時間複雜度為 $O(\log n)$ 的搜尋樹，我們稱為平衡搜尋樹 (balanced search tree)，例如：
  - AVL trees、2-3 trees、Red-Black trees 和 B-trees

總結 (2/2)

- BST 的複雜度分析

	Average	Worst Case
Space 空間	$O(n)$	$O(n)$
Access 存取	$O(\log n)$	$O(n)$
Search 搜尋	$O(\log n)$	$O(n)$
Insertion 加入資料	$O(\log n)$	$O(n)$
Deletion 刪除資料	$O(\log n)$	$O(n)$

Q & A