第4次作業-作業-HW4

> 學號：112111219
> 姓名：陳恩偉
> 作業撰寫時間：60 (mins，包含程式撰寫時間)
> 最後撰寫文件日期：2024/12/31

本份文件包含以下主題：(至少需下面兩項，若是有多者可以自行新增)

- ☑ 說明內容
- ☑ 個人認為完成作業須具備觀念

說明程式與內容

1. 請回答下面問題。

Ans:

```
class BinarySearchTree:
    def __init__(self, size):
        self.tree = [None] * size

    def insert(self, value, index=1):
        if index >= len(self.tree):
            return
        if self.tree[index] is None:
            self.tree[index] = value
        elif value < self.tree[index]:
            self.insert(value, 2 * index)  # 左
        else:
            self.insert(value, 2 * index + 1)  # 右

    def display(self):
        print("Binary Search Tree as Array:")
        print(self.tree)

# 測試
bst = BinarySearchTree(15)
data = [28, 23, 33, 41, 22, 27]
for num in data:
    bst.insert(num)
bst.display()
```

2. 請回答下面問題。

Ans:

```python
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None


class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, value):
        self.root = self._insert(self.root, value)

    def _insert(self, node, value):
        if not node:
            return Node(value)
        if value < node.value:
            node.left = self._insert(node.left, value)
        else:
            node.right = self._insert(node.right, value)
        return node

    def find_and_add_child(self, parent_value, child_value):
        def find(node):
            if not node or node.value == parent_value:
                return node
            return find(node.left if parent_value < node.value else node.right)

        parent = find(self.root)
        if parent:
            if child_value < parent.value:
                parent.left = Node(child_value)
            else:
                parent.right = Node(child_value)
        else:
            print("Parent node not found.")

    def display(self):
        def inorder(node):
            return inorder(node.left) + [node.value] + inorder(node.right) if node
else []
        print("In-order Traversal:", inorder(self.root))


# 測試
bst = BinarySearchTree()
for num in [28, 23, 33, 41, 22, 27]:
    bst.insert(num)

bst.display()  # 初始
bst.find_and_add_child(33, 35)  # 覆蓋右節點
bst.display()  # 結果
```

3. 請回答下面問題：

Ans: 最好的情況： 如果樹是平衡的，即每個節點的左右子樹高度差不超過 1，那麼樹的高度為 O(log n)。因此，每次插入操作的時間複雜度是 O(log n)，最終建樹的時間複雜度是 O(n log n)，因為要插入 n 個節點，每個節點的插入操作都需要 O(log n) 的時間。當樹完全不平衡時，插入操作的每次時間複雜度是 O(n)。

最壞的情況： 如果樹完全不平衡（例如每次插入的節點都是按照升序排列的），那麼樹變成一條直線，樹的高度 h 會是 O(n)。每次插入操作的時間複雜度為 O(n)，而插入所有 n 個節點的總時間複雜度就是 O(n * n)，即 O(n^2)。當樹保持平衡時，插入操作的每次時間複雜度是 O(log n)。

4. 請回答下面問題：

Ans: 樹狀結構常用於儲存層級資料，如檔案系統、資料庫索引、網頁 DOM 結構等，能高效地處理資料查詢、插入、修改和刪除。

操作過程： 1.新增節點： 從根節點開始，依值比較並插入到左或右子樹空位置。

2.修改節點內容： 查找指定節點，找到後直接修改其值。

3.刪除節點： 若節點無子節點，直接刪除。 若有一個子節點，刪除並將子節點繼承位置。 若有兩個子節點，將右子樹的最小節點移至該節點位置，然後刪除最小節點。

個人認為完成作業須具備觀念

開始寫說明，需要說明本次練習需學會那些觀念 (需寫成文章，需最少50字，並且文內不得有你、我、他三種文字)且必須提供完整與練習相關過程的notion筆記連結

了解樹狀結構的基本概念及其應用，尤其是二元搜尋樹的插入、查找、修改和刪除操作。這些操作能夠有效地處理資料的層級關係，並能在查詢過程中提高效率。熟悉遞迴的運用，因為許多樹狀結構的操作都依賴遞迴方法來遍歷或修改節點。理解時間複雜度分析，能夠根據樹的結構判斷操作的最壞情況和最佳情況，進而選擇合適的資料結構。