第4次作業-作業-HW4

學號：112111207
姓名：陳品霖
作業撰寫時間：120 (mins，包含程式撰寫時間)
最後撰寫文件日期：2024/12/26

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

1.在二元搜尋樹(binary search tree)中，於投影片Topic 6中有介紹如何使用陣列方法如何建樹，請打開1.py檔案，完成以陣列(array)方式進行建樹。

Ans:

```
tree = ['A', 'B', 'C', 'D', 'E', 'F', None]

print(f"Tree：{tree}")
```

2.呈第一題，因為陣列(array)可以快速的使用索引(index)，到達該節點(node)的位置，進而對該節點進行操作，如新增子節點(child node)。若當使用鏈結串列(linked list)的方式進行撰寫時，二元搜尋樹要如何快速找到該節點，並且於該節點下新增子節點，請打開2.py檔，以完成該程式

```
class CompactBinaryTree:
    def __init__(self, elements):
        """初始化二元樹，接收一個陣列作為基礎樹結構。"""
        self.tree = elements

    def find_node_index(self, value):
        """找到節點索引，若不存在返回 None"""
        return self.tree.index(value) if value in self.tree else None

    def add_child(self, parent_value, child_value, position='left'):
        """新增子節點到父節點的左或右位置"""
        parent_index = self.find_node_index(parent_value)
```

```
        if parent_index is None:
            print(f"父節點 {parent_value} 不存在。")
            return False

        # 計算子節點索引 (左子節點: 2i+2, 右子節點: 2i+1)
        child_index = 2 * parent_index + (2 if position == 'left' else 1)

        # 確保陣列長度足夠
        if len(self.tree) <= child_index:
            self.tree += [None] * (child_index - len(self.tree) + 1)

        # 確保該位置為空後新增子節點
        if self.tree[child_index] is None:
            self.tree[child_index] = child_value
            print(f"成功在 {parent_value} 的 {position} 新增子節點
{child_value}。")
        else:
            print(f"{parent_value} 的 {position} 已有子節點
{self.tree[child_index]}。")

    def display(self):
        """顯示樹的結構 (過濾掉 None)"""
        print([node for node in self.tree if node is not None])


class TreeNode:
    def __init__(self, value, index):
        self.value = value
        self.index = index
        self.left = None
        self.right = None


class BinaryTree:
    def __init__(self, values):
        self.root = self.build_tree(values)

    def build_tree(self, values):
        if not values:
            return None
        nodes = [TreeNode(value, i) if value is not None else None for i, value in
enumerate(values)]
        for i in range(len(nodes)):
            if nodes[i] is not None:
                left_index = 2 * i + 1
                right_index = 2 * i + 2
                if left_index < len(nodes):
                    nodes[i].left = nodes[left_index]
                if right_index < len(nodes):
                    nodes[i].right = nodes[right_index]
        return nodes[0]

    def find(self, value):
        return self._find_recursively(self.root, value)
```

```
    def _find_recursively(self, current, value):
        if current is None:
            return None
        if current.value == value:
            return current
        left_result = self._find_recursively(current.left, value)
        if left_result:
            return left_result
        return self._find_recursively(current.right, value)

    def add_child(self, parent_value, child_value, is_left=True):
        parent_node = self.find(parent_value)
        if parent_node:
            if is_left:
                if parent_node.left is None:
                    parent_node.left = TreeNode(child_value, -1)  # -1 表示索引未知
                    print(f"新增成功：節點 {child_value} 作為 {parent_value} 的左子節點。")
                else:
                    print("錯誤：左子節點已存在。")
            else:
                if parent_node.right is None:
                    parent_node.right = TreeNode(child_value, -1)  # -1 表示索引未知
                    print(f"新增成功：節點 {child_value} 作為 {parent_value} 的右子節點。")
                else:
                    print("錯誤：右子節點已存在。")
        else:
            print(f"錯誤：找不到節點 {parent_value}。")


def main():
    print("選擇測試模式：")
    print("1. 純陣列寫法")
    print("2. 鏈結串列 (Linked List) 寫法")
    mode = input("請輸入模式編號（1 或 2）：")

    if mode == "1":
        initial_tree = ['A', 'B', 'C', 'D', 'E', 'F']
        tree = CompactBinaryTree(initial_tree)
        print("初始二元樹已建立（純陣列寫法）：")

        while True:
            print("\n二元樹操作選單：")
            print(f"二元樹：{[node for node in tree.tree if node is not None]}")
            print("1. 搜尋節點")
            print("2. 新增子節點")
            print("3. 離開")
            choice = input("請輸入選項（1-3）：")

            if choice == "1":
                value = input("請輸入要搜尋的節點值：")
```

```
                index = tree.find_node_index(value)
                if index is not None:
                    print(f"找到節點：值={value}, 索引={index+1}")
                else:
                    print("節點不存在。")
            elif choice == "2":
                parent = input("請輸入父節點的值：")
                child = input("請輸入要新增的子節點值：")
                position = input("請輸入新增子節點的位置 ('left' 或
'right')：").lower()
                if position not in ['left', 'right']:
                    print("輸入的位置無效，請重新輸入 'left' 或 'right'。")
                    continue
                tree.add_child(parent, child, position=position)
            elif choice == "3":
                print("感謝使用，程式結束！")
                break
            else:
                print("無效選項，請重新輸入。")

    elif mode == "2":
        initial_tree = ['A', 'B', 'C', 'D', 'E', 'F']
        tree = BinaryTree(initial_tree)
        print("初始二元樹已建立（鏈結串列寫法）：")

        while True:
            print("\n二元樹操作選單：")
            print(f"二元樹：{initial_tree}")
            print("1. 搜尋節點")
            print("2. 新增子節點")
            print("3. 離開")
            choice = input("請輸入選項（1-3）：")

            if choice == "1":
                value = input("請輸入要搜尋的節點值：")
                node = tree.find(value)
                if node:
                    print(f"找到節點：值={node.value}, 索引={node.index+1}")
                else:
                    print("節點不存在。")
            elif choice == "2":
                parent_value = input("請輸入父節點的值：")
                child_value = input("請輸入要新增的子節點值：")
                position = input("請輸入子節點位置（左子節點輸入 'L'，右子節點輸入
'R'）：").upper()
                is_left = position == "L"
                tree.add_child(parent_value, child_value, is_left)
            elif choice == "3":
                print("感謝使用，程式結束！")
                break
            else:
                print("無效選項，請重新輸入。")

    else:
```

```
        print("無效模式，程式結束。")


if __name__ == "__main__":
    main()
```

程式碼說明：

1. 模式 1：純陣列寫法

- 使用一個陣列表示二元樹。節點的子節點索引透過公式計算：左子節點索引為 $2i+2$。右子節點2i+1。

2. 模式 2：鏈結串列 (Linked List) 寫法

- 每個節點是物件 (TreeNode)。 節點透過 left 和 right 屬性指向其子節點。

3.請試著說明第二題所寫程式於建樹時的時間複雜度

兩種方法的建樹時間複雜度對比：

方法	初始化複雜度	新增節點複雜度（單次）	總建樹複雜度（逐步新增）
純陣列寫法	O(1)	(O(n + k))	(O(m (n + k))
鏈結串列寫法	(O(n))	(O(n))$	(O(m*n))

結論：

- 若樹的結構較為穩定且節點數量固定，純陣列寫法會更高效，因為初始化是 $O(1)$。
- 若樹的結構頻繁變動且需要動態新增節點，鏈結串列寫法更靈活，因為它不需要處理陣列的動態擴展問題。

4.請說明何時需要使用到樹狀架構儲存資料的應用，並簡單說明該應用的例子於新增(insert)子節點、修改(modify)節點內容，以及刪除(delete)節點的過程。

樹狀架構儲存資料的應用場景

1. 檔案系統

- 應用場景： 檔案系統是典型的樹狀結構，根節點是磁碟分區或目錄，子節點是子目錄或檔案。
- 操作：
    - 新增：在某個目錄下新增檔案或資料夾。

- 修改：更新檔案名稱或內容。
- 刪除：刪除檔案或資料夾時，同時刪除其所有子項目。

個人認為完成作業須具備觀念

本次練習的重點在於學習樹狀結構的基本概念及其應用。需掌握樹的核心操作，包括新增節點、修改節點內容以及刪除節點等。特別是了解如何選擇適合的樹結構來處理分層資料，並將理論轉化為實際程式碼實作，進一步加深對樹狀資料操作的理解。