資處科三年一班 12 號張皓承

程式設計進階
以物件導向實作二元搜尋樹

一、前言：

在這個學期當中我們在程式設計進階這堂課中我們除了利用 Java 試著寫寫看 APCS 實作題的題目中，老師也教導我們在解到樹狀圖 (tree)、圖形 (graph) 的題目時如果處理的問題較為複雜，例如二元搜尋樹的建構，實務是多利用一維陣列完成，但不利於節點的刪除。在課堂中，老師利用物件(Object)模擬指標(Pointer)來解決這個問題。這次的學習歷程檔案以二元樹舉例來物件化為主並延伸到其他的資料結構，來記錄這學期所學。

二、樹(Tree)的物件主要結構：

樹主要是用來儲存根節點與當節點須要往下延伸或增加資料的時候利用判別的方式來增加 (如圖 1)，亦或者要刪除樹裡的節點時透過尋找節點的方法來分析情況將樹刪除那個節點後重新連接(如圖 2)。

```
public static class Tree{//用來讓Node的根結點與輸出的方法設計
    Node root;
    public Tree(Node n) {//設定Tree的根
        this.root = n;
    }
    public void add(Node n){//把Node加進Tree裡，也就是往下延伸或是擴展
        Node temp = this.root;
        while(true) {//尋找可以放的位置
            if(temp.value > n.value) {
                if(temp.left == null) {temp.left = n;n.parent = temp;break;}
                else {temp = temp.left;}
            }
            else if(temp.value < n.value) {
                if(temp.right == null) {temp.right = n;n.parent = temp;break;}
                else {temp = temp.right;}
            }
public void remove(int n) {//刪除節點
    Node temp=this.findnode(n);
        if(temp.left!=null&&temp.right!=null) {//如果刪除的節點同時擁有left和right的Node就找出左子樹的最大值
```

圖 1

```
            temp.value = max.value;
            if(max.left!=null) {max.parent.right=max.left;}
            else {if(max.parent.left.equals(max)) {max.parent.left=null;}
                if(max.parent.right.equals(max)) {max.parent.right=null;}}
        }else if(temp.left!=null) {//如果只有左子樹有節點就直接讓被刪除的節點根指向被刪除節點的left
            if(temp.parent.left.equals(temp)) {temp.parent.left = temp.left;}
            if(temp.parent.right.equals(temp)) {temp.parent.right = temp.left;}
        }else if(temp.right!=null) {//如果只有右子樹有節點就直接讓被刪除的節點根指向被刪除節點的right
            if(temp.parent.left.equals(temp)) {temp.parent.left = temp.right;}
            if(temp.parent.right.equals(temp)) {temp.parent.right = temp.right;}
        }else {//如果被刪除的節點沒有left跟right的Node就直接刪除Node
            if(temp.parent.left.equals(temp)) {temp.parent.left=null;}
            if(temp.parent.right.equals(temp)) {temp.parent.right=null;}
        }
}
public Node findnode(int n) {//找到樹裡Node的value與n相同的的Node
    Node temp=this.root;
    while(true) {
        if(n>temp.value) {temp=temp.right;}
        if(n<temp.value) {temp=temp.left;}
        if(n==temp.value) {return temp;}
```

圖 2

三、節點(Node)的物件主要結構：

在節點中除了儲存節點的值以外也利用了指標的特性來讓節點的 left 和 right 的屬性指向這個節點的子節點的記憶體位置 (類似 Linked List) 來達到串接的效果 (也可以記錄自己的父節點記憶體位置，方便往上搜尋)，並且為了方便讓除錯更方便，我繼承了 ToString 的方法，來方便讓自己除錯 (如圖 3)。

```
public static class Node {
    int value;//值
    Node right=null;//右子節點
    Node left=null;//左子節點
    Node parent=null;//父節點
    public Node(int n) {
        this.value = n;
    }
    public String ToString() {//複寫toString的方法讓他return "{value,left,right,parent}"
        int a = this.value;
        String b="NULL",c="NULL",d="NULL";
        if(this.left!=null) {b = Integer.toString(this.left.value);}
        if(this.right!=null) {c = Integer.toString(this.right.value);}
        if(this.parent!=null) {d = Integer.toString(this.parent.value);}
        return "{" + a + "," + b +"," + c + "," + d + "}";
    }
}
```

圖 3

四、輸出與尋訪(Traversal)的使用：

完成了上述的物件設計後，就可以利用 Stack、Queue、遞迴...等等的方式來讓樹搜尋、輸出或是排序我們所需要的答案或資料，範例如下 (如圖 4)：

```
public static void Inorder(Node n) {//中序輸出(也就是排序 Sort)
    if(n.left!=null)Inorder(n.left);System.out.print(n.value);if(n.right!=null)Inorder(n.right);}
public static void Preorder(Node n) {//前序輸出
    System.out.print(n.value);if(n.left!=null)Preorder(n.left);if(n.right!=null)Preorder(n.right);}
public static void Postorder(Node n) {//後序輸出
    if(n.left!=null)Postorder(n.left);if(n.right!=null)Postorder(n.right);System.out.print(n.value);}
public static void DFS(Tree t,Node n) {//深度優先搜尋
    System.out.print(n.value);
    if(n.right!=null) {DFS(t,n.right);}
    if(n.left!=null) {DFS(t,n.left);}
}
public static void BFS(Tree t,Node n) {//廣度優先搜尋
    if(t.root.equals(n)) {System.out.print(n.value);}
    if(n.right!=null) System.out.print(n.right.value);
    if(n.left!=null) System.out.print(n.left.value);
    if(n.right!=null) {BFS(t,n.right);}
    if(n.left!=null) {BFS(t,n.left);}
}
public Node BS(int n) {//二分搜尋法
    Node temp=this.root;
    while(true) {
        if(temp.equals(null)) {return new Node(-1);}
        if(n>temp.value) {temp=temp.right;}
        if(n<temp.value) {temp=temp.left;}
        if(n==temp.value) {return temp;}
    }
}
```

圖 4

五、以物件導向實作其他資料結構補充：

在上述提到的 Node 與 Tree 的物件中更可以多加的延伸使其符合其他資料結構的要求，例如將多個 Node 位置儲存在 ArrayList 裡或著將要儲存的值設為 ArrayList 來達到連接多個結點或紀錄多個資料 (如圖 5)。

```
public static class Node {
    ArrayList<Integer> value = new ArrayList<Integer>();
    ArrayList<Node> child = new ArrayList<Node>();//指向一或多個Node(用於N-ray Tree或是Graph)
    ArrayList<Node> parent = new ArrayList<Node>();//被一或多個Node指向(用於Graph)
    public Node(int[] n) {//設定new Node的方法
        for(int i:n) {this.value.add(i);}
    }
}
```

六、心得：

在這次的課程中，我除了發現了類別(Class)的便利性，更發現在實作當中需要將版面與變數、方法的名稱更加的清楚、簡潔來讓之後的自己理解或是分組的同學們在合作時可以更快速的解決問題與修該程式。