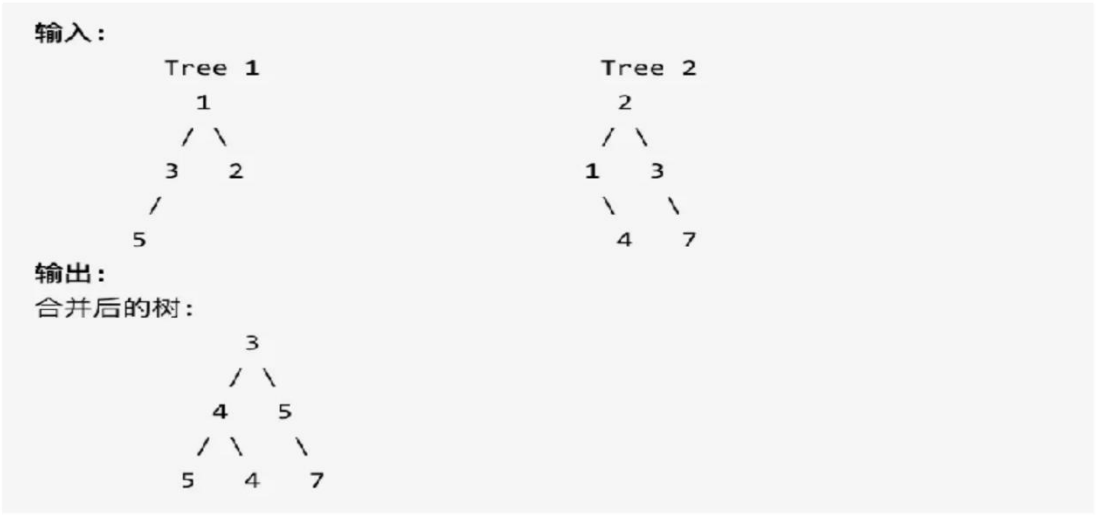
二叉树的应用实验报告

一、问题分析

- 处理对象：两颗二叉树，将它们对应节点元素值合并
- 实现功能：按输入的数据构造两颗二叉树，合并二者，将它们中的一个覆盖到另一个上，形成一颗新的二叉树，按先序遍历输出新的二叉树
- 结果显示：按先序遍历输出的合并后新二叉树
- 题目输入样例： 135###2##

  21#4##3#7##

示例：



- 求解过程：对应的节点都不为 NULL，则二者元素值相加，如 Tree1 根结点与 Tree2 根结点相加得 3，是合并后新树的根结点，不为 NULL 的节点直接作为新树的对应节点，如 Tree1 的 3 的右节点为 NULL，Tree2 对应 1 的节点的右节点为 4，则 4 作为新树的对应节点。

二、数据结构和算法设计

1. 抽象数据类型设计：

即 BinTree_ADT，其中封装了二叉树的多种函数接口，如二叉树的构造、析构、判断空树、获取根结点、先序遍历和合并，具体设计如下：

```
template<typename  E>
class  BinTree_ADT //二叉树类 ADT
{
private:
    BinTree_ADT
(const  BinTree_ADT&)  {}                    //  Protect
copy  constructor
public:
    BinTree_ADT() {}//默认构造函数

    virtual ~BinTree_ADT() {}//析构函数

    virtual bool  BinTreeEmpty() = 0;//判断二叉树是否为空

    virtual BinNode<E>*getRoot() = 0;//获得根节点

    virtual void  setRoot(BinNode<E>*r) = 0;//设置根节点

    virtual void  clear(BinNode<E>*r) = 0;   //清空二叉树

    virtual
void  preOrder(BinNode<E>*tmp,void(*visit)(BinNode<E>*nod
e)) = 0;
    //先序遍历，传入相对应的访问函数即可对该当前结点实现不同功能的访
问（本程序为输出）

    virtual void merge (BinNode<E>*, BinNode<E>*) =
0;      //合并二叉树
```

2. 物理数据对象设计：

首先是二叉树节点模版类 BinNode，用于实例化二叉树中的节点，每个节点具有私有成员数据 elem(元素值)，以及节点

指针 lc(指向左孩子)、rc(指向右孩子), 数据类型为 BinNode* 通过其公有成员函数设置和访问。

其次是二叉树模版类 BinTree, 其公有继承了 BinTree_ADT, 具体实现其中接口函数，实现对二叉树的基本操作，以及题目要求的二叉树的合并操作，且设置私有成员节点指针变量 root 储存根结点，数据类型为 BinNode* 。

3.算法思想设计

1. 根据输入的两行字符串, 构建两颗储存 char 类型的数据的二叉树 BT1，BT2，注意构建过程中遇到‘#’字符代表空指针，设置对应节点为空。

2. 同时先序遍历两颗二叉树，每遍历到一对节点，按题目的合并规则，将 BT2 节点合并到 BT1 对应的节点上，形成新的 BT1。

3. 输出为合并之后的二叉树 BT1 的前序遍历顺序表示法。

4.关键功能的算法步骤

1. 采用先序遍历的算法思想，两颗树同时进行先序遍历，保证两颗树的节点一一对应。

2. 根据合并规则，有三种情况：

- BT1 的节点和 BT2 对应的节点都不为空，则二者元素值相加，将 BT1 该节点的值重新设置为二者相加的结果。
- BT1 的节点不为空，BT2 的对应节点为空，则 BT1 该节点不变，不做处理。

- BT1 的节点为空，BT2 的对应节点不为空，则 BT2 的节点覆盖掉 BT1 的空节点，退出一层递归，回到空节点的父节点，重新设置空节点为 BT2 的对应节点。

三、算法性能分析

1. 采用的是先序遍历的算法思想，设 BT1 和 BT2 的节点数分别为 a，b，因为对两颗二叉树都要遍历，因此递归次数为 N = max(a,b),所以算法时间复杂度为 Θ(N)。

2. 空间复杂度：空间复杂度取决于递归调用的次数，因此为 O(max(a,b))。

3. 算法优点是易理解，时间复杂度是线性一阶的，缺点在于需将两颗二叉树都要遍历完。