一、问题分析

- 处理的数据对象：字符串
- 实现的功能： 利用树结构来存储两个字符串然后进行字符串比较，观察一棵树是否是另一颗树的子树（也就是判断一个字符串是否是另一个字符串的子串)
- 处理后结果的显示： 是子树就输出“yes”，不是则输出“no”
- 样例举例求解结果

```
输入数据是前序遍历
样例输入1：
     AB##C##（父树）
     AB##C##（子树）
进行手动模拟，根据题目给的条件“#”代表空指针NULL，来建立树，然后进行比较
       A
    B     C    发现两棵树相等，显然是子树,故输出“yes”
 # #    # #
样例输入2：
     ABD##E##CGH###F##
     CGH###F##
同样进行手动模拟建立树，然后发现两棵树可以匹配，是第二棵树是子树，所以输出“yes”
```

二、数据结构和算法设计

抽象数据类型的设计

```
/*
 * 该数据结构是节点类
 * 用来存储数据元素
 */
template<typename E>class BinNode
{
    public:

        virtual ~BinNode(){}   //析构函数

        virtual E& getValue() = 0;

        virtual void setValue(const &E) = 0;

        virtual BinNode* left() const = 0;

        virtual void setLeft(BinNode *) = 0;

        virtual BinNode* right() const = 0;

        virtual void setRight(BinNode*) = 0;

        virtual bool isLeaf() = 0;
};
```

物理数据对象设计

```cpp
/*
 *BinNode节点类的具体实现，其可以存储两个指针和一个数据元素
 */
template<typename E>class binNode{
    private:
        binNode* _lc;
        binNode* _rc;
        E _elem;
    public:
        //默认构造函数
        binNode(E temp , binNode<E>* lc = NULL , binNode<E>* rc = NULL){
            _elem = temp;
            _lc = lc;
            _rc = rc;
        }

        binNode(binNode<E>* lc  = NULL, binNode<E>* rc = NULL){
            _lc = lc;
            _rc = rc;
        }
        E getValue()const{return _elem;}
        binNode<E>* getleft()const{return _lc;}
        binNode<E>* getRight()const{return _rc;}
        void setLeft(binNode<E>* left){
            _lc = left;
        }
        void setRight(binNode<E>* right){
            _rc = right;
        }
        void setValue(const E elem)
        {
            _elem = elem;
        }
        //判断是否是叶节点
        bool isLeaf()
        {
            return (_rc==nullptr&&_lc == nullptr);
        }
};
```

```cpp
/*
 *二叉树的类的实现
 *二叉树的很多应该有的功能我附在了后面，解决本题目不需要多么复杂的功能，只需要查询插入
 *二叉树完整功能的代码附在实验源代码中，这里为了篇幅就不复制粘贴了
 *就可以
 */
template<typename E>class BinTree{
    private:
    //根节点
        binNode<E>* root;
    //析构函数的帮助
        void clear(binNode<E>*r)
        {
            if(r == nullptr)return;
            clear(r->left());
            clear(r->right());
```

```
            delete r;
        }
    public:
        BinTree()
        {
            root = new binNode<E>();
        }
        ~BinTree()
        {
            clear();
        }
    //得到根节点
        binNode<E>* getRoot()
        {
            return root;
        }
    //设置根节点
        void setRoot(binNode<E>* r)
        {
            root = r;
        }
};
```

算法思想的设计

- 此题题目要处理字符串，我们直接用C++提供的string来存储数据进而来建立两棵树，最后经过一个判断两棵树是否是父子树的关系的函数来输出结果。
- 关键在于建立树的操作和判断是否是父子树的操作。

```
    为了利用string类，我们可以令BinTree的每个节点都存储string类型的数据（也可以存储char类型的，但是操作相较于复杂一点），然后将数据输入string中，最后给树节点一一赋值完成建立树的操作。

    判断两棵树是否是父子树的关系，我们可以利用递归，实现逐一比较
```

关键功能的算法步骤

```
//建立树
template<typename E>
binNode<E>*  creatBinaryTree(string  s[],  int  &x,int  n)
{
//如果遇见空结点就返回空
        if  (s[x]  == "#" )
                return  NULL;
        else
        {
                binNode<E>*node  =  new  binNode<E>(s[x]); //新建结点
                x  =  x  +  1;
                if  (x  <  n)
                node->setLeft(creatBinaryTree<E>(s,  x,n)); //对结点左孩子
                                                            //赋值
                x  =  x  +  1;
                if  (x  <  n)
                node->setRight(creatBinaryTree<E>(s,  x,n));//对结点右孩子
                                                            //进行赋值
```

```
                return  node;                                   //递归返回
        }
}
void  creatBinaryTree(BinTree<string>*BT)
{

        string tree;    //输入的数据暂时存储
        cin>>tree;
        int n = tree.length();
         string*s  =  new  string[n];  //保留每个节点的信息
         for  (int  i  =  0;  i  <  n;  i++)
         {
                 s[i] = tree[i];  //赋值
         }
         int  now  =  0;
         BT->setRoot(creatBinaryTree<string>(s,  now,  n));  //建立树
}
```

```
/*
 *最核心的部分------判断是否是子树
 *首先判断两个结点及其子节点是否是子树关系（设为root1 和 root2）
 *       1. 如果root2为空 肯定是子树
 *       2. 如果root1为空，肯定不是子树关系
 *       3. 如果root1.value ! = root2.value 那么肯定也不是子树关系
 *       4. 诸侯root1.value一定等于root2.value，所以就比较它们的左孩子和右孩子是
 *                                       否也满足这个关系 （递归下去就可以判断）
 *
 *判断两棵树是否为子树关系
 * 首先从两棵树根节点开始 root1 可以不是从根节点开始，以为它是父树，root2一定得从根节点就匹配，因为它是子树。
 * 如果两个根节点相同就判断他们的孩子是否也相同，即调用isPart()函数
 * 如果result为真即是子树就直接返回，如果不是就让root1的左孩子和右孩子和root2进行比较
 */

template<typename E>
bool isPart(binNode<E>* root1 , binNode<E>* root2)
{
    if(root2 == NULL)return true;
    if(root1 == NULL)return false;
    if(root1->getValue() != root2 ->getValue())return false;
    return isPart(root1->getleft(),root2->getleft()) &&
        isPart(root1->getRight() , root2 -> getRight());
}

template<typename E>
bool isPartTree(binNode<E>* rootA , binNode<E>* rootB)
{
    bool result = false;
    if(rootA!=NULL&&rootB!=NULL)
    {
        if(rootA->getValue() == rootB->getValue())
        result = isPart(rootA,rootB);
        if(!result) result = isPartTree(rootA->getleft(),rootB);
        if(!result) result = isPartTree(rootA->getRight(), rootB);
    }
    return result;
```

```
38 }
```

算法性能分析

本算法主要消耗在建树和判断是否为子树上，两个操作是平行操作。

1. 建立树消耗 遍历了一遍 O（m+n）的复杂度 （n个父树结点，m个子树结点）
2. 判断是否是子树操作 O（mn）
   所以时间复杂度为**O（n^2）**

日志

```
1 　　这道题仔细分析一下不难发现，就是一个字符串匹配的问题，所以取巧的方法就是写个KMP算法交上去，我尝试了，然后直接全过，当然，这是没分的，所以之后我又进行了二叉树数据结构的设计。
2 　　然后差不多在一个小时写好了代码，但一直出现类“ambiguous”的错误，改了一个小时发现是BinNode类的构造函数冲突了，有歧义。
3 　　总结上述就是，写代码还是要细心，特别是这种多文件复杂类的编程，细节尤其重要，可以大大节省时间。
```