实验 4　二叉树的应用

计科 2004 202008010404 赖业智

一.问题分析

1）问题与功能描述：

1. 需要处理的数据是两行字符串。
2. 实现的功能有：

•读入一行字符串按照前序遍历建树 A

•读入一行字符串按照前序遍历建树 B

•对输入的两棵二叉树 A 和 B，判断 B 是不是 A 的子树

•若是输出“yes”，若不是输出“no”

2）样例分析：

求解方法：对于输入的字符串可以简单通过判断第二个字符串是否是第一个字符串的子串来判断二叉树 B 是否是 A 的子树。故建树后先进行遍历，寻找元素相同的结点，找到之后通过递归判断它们的左孩子和右孩子是否对应相同，若递归到最后都相同，那么 B 是 A 的子树。

具体样例：

【样例输入 1】

ABD##E##CGH###F##

CGH###F##

【样例输出 1】

Yes

【样例说明 1】

二叉树 B 是二叉树 A 的一棵子树

【样例输入 2】

ABD##E##CGH###F##

CG##F##

【样例输出 2】

no

【样例说明 2】

二叉树 B 不是二叉树 A 的一棵子树

数据结构分析：

1）数据对象 本题处理的数据为字符串(char)数组。

数据关系 本题处理的数据为字符型有限数据，题目要求使用树形结构，并且要求前序遍历构建二叉树。故数据之间关系为链式关系，根元素对应 0-2 个子元素。

2）基本操作

- **【功能描述】**寻找结点

  【名字】 find

  【输入】 元素 e，代表结点中元素的值

  【输出】 返回该结点

- **【功能描述】** 判断两结点是否包含

  【名字】 isPart

  【输入】 BinNode<E>* tmp1, BinNode<E>* tmp2

  【输出】 返回 1 或 0

- **【功能描述】** 判断 B 是否是 A 的子树

  【名字】 isPartTree

  【输入】 BinTree<E>* t1, BinTree<E>* t2

  【输出】 返回 1 或 0

【物理实现】

```
BinNode<E>*   find(BinNode<E>* tmp, E   e)
{
    if (tmp == NULL) return NULL;
    if (tmp->getValue() == e) return tmp;
    if (find(tmp->left(), e) != NULL)//左右子树递归查找
        return find(tmp->left(),e);
    else if (find(tmp->right(), e)!= NULL)
        return find(tmp->right(), e);
    else
        return NULL;
      }

    bool isPart(BinNode<E>* tmp1, BinNode<E>* tmp2) {
    if (tmp1 == NULL&&tmp2==NULL) return true;//分析多种情况，只有相等才能返回 true
    if (tmp1 == NULL&&tmp2!=NULL) return false;
    if (tmp1 != NULL && tmp2 == NULL) return false;
    return (isPart(tmp1->left(), tmp2->left()) && isPart(tmp1->right(),
    tmp2->right()))&&tmp1->getValue() == tmp2->getValue();//判断左右子树值是否相等
    }

    bool isPartTree(BinTree<E>* t1, BinTree<E>* t2) {
    E temp = t2->getRoot()->getValue();//获得 B 树的根结点元素
    BinNode<E>* b=t1->find(temp);//在 A 中查找
    return isPart(b, t2->getRoot());
        }
```

二. 算法分析

1）算法思想：先建树，然后在 A 寻找和 B 根结点元素相等的结点，若 A 中无则返回 0，若 A 有则以该结点为根结点，获得一棵子树 C，比较 B 和 C 是否完全相同。通过递归可以实现。

2）关键步骤：

```
template<typename  E>//建树函数
BinNode<E>* creatBinaryTree(string  s, int& x, int  n)
{
    if (s[x] == '#')//输入#为空结点
        return  NULL;
    else
    {
        BinNode<char>* node = new  BinNode<char>;
        x = x + 1;
        if (x < n)
        node->setLeft(creatBinaryTree<E>(s, x, n));
        x = x + 1;
        if (x < n)
        node->setRight(creatBinaryTree<E>(s, x, n));
        return  node;
    }
}
```

```
    bool isPart(BinNode<E>* tmp1, BinNode<E>* tmp2) {//判断是否包含的函数
    if (tmp1 == NULL&&tmp2==NULL) return true;
    if (tmp1 == NULL&&tmp2!=NULL) return false;
    if (tmp1 != NULL && tmp2 == NULL) return false;
    return (isPart(tmp1->left(), tmp2->left()) && isPart(tmp1->right(),
    tmp2->right()))&&tmp1->getValue() == tmp2->getValue();
    //找值相同的根结点（遍历解决）
    //判断两结点是否包含（递归：值、左孩子、右孩子分别相同）
}


Int main (){ BinTree<char>* BT1 = new BinTree<char>;//main 函数
 getline(cin, s);
 BT1->setRoot(creatBinaryTree<char>(s, a, s.size()));//建树
 BinTree<char>* BT2 = new BinTree<char>;
 getline(cin, s);
 BT2->setRoot(creatBinaryTree<char>(s, b, s.size()));
 if (BT1->isPartTree(BT1, BT2))//比较是否包含
       cout << "yes";
 else
            cout << "no";}
```

3）性能分析：

【时间复杂度】由于在 A 中找与 B 根结点元素相同的结点，需要 find 函数遍历查找，然后用 isPart 函数递归判断是否包含，相当于遍历子树，故复杂度为 O(n)。

【空间复杂度】由于采用递归的方式来建树，所以空间复杂度为 O(log(n))