组号: 5

上海大学计算机工程与科学学院

实 验 报 告

(数据结构 1)

学　　期: 2024-2025 年冬季

组　　长: 冯俊佳

学　　号: 23122721

指导教师: 朱能军

成绩评定: _____(教师填写)

二〇二四年十一月二十九日

小组信息				
登记序号	姓名	学号	贡献比	签名
1	冯俊佳	23122721	1/3	冯俊佳
2	钱傲栋	23122803	1/3	钱傲栋
3	邱添	23122720	1/3	邱添

实验概述	
实验零	（熟悉上机环境、进度安排、评分制度；确定小组成员）
实验一	抽取简历
实验二	车厢调度
实验三	文学研究助手
实验四	二叉树拓展及标记二叉树

实验四

1 实验题目

1.1 左右子树交换

在二叉链表类模板中增加函数成员 Revolute()，实现二叉树中所有结点的左右子树交换。

1.2 标记二叉树

1.2.1 问题描述

一颗二叉树，根结点标记为（1，1），规定：如果一个结点标记为（a，b），则它的左孩子（如果存在）标记为（a+b，b），它的右孩子（如果存在）标记为（a，a+b）。现在已知某个结点的标记为（a，b），求从根结点开始需要经过多少次左分支和多少次右分支才能达到结点（a，b）。

1.2.2 输入文件

输入文件第一行只有一个整数 n，表示测试的数据组数。

接下来 n 行（第 2～n+1）行，每行包括两个整数 a 和 b。

1.2.3 输出文件

输出文件有 n 行，每行包括两个整数，分别表示从根结点开始到（a，b）需要经过的左分支数和右分支数。

1.2.4 输入样例

```
2
42      1
3       4
```

1.2.5 输出样例

```
41      0
2       1
```

2 实验内容

本次实验是实现左右子树交换和二叉树的标记，主要涉及二叉树的算法设计

和实现。具体包括:二叉树的先序、中序、后序遍历的递归和非递归实现，二叉树的层次遍历，二叉树的复制、高度计算、结点个数计算、叶子结点个数计算和二叉树的翻转。

3 解决方案

3.1 算法设计

3.1.1 数据结构

本次实验主要数据结构是二叉树，它由节点构成，每个节点包括数据、左子树引用和右子树引用。这种数据结构自然地反映了树形结构的特点。

3.1.2 算法思想

左右子树的交换：

- 首先处理空指针的情况，如果输入的根节点‘r’为空，则直接返回空指针。
- 接着创建一个新的二叉树节点’newRoot’，其数据与原始节点’r’的数据相同。
- 之后采用调用递归函数的方式，对原始节点’r’的右子树进行左右子树交换，并把结果赋给新节点’newRoot’的左孩子。
- 对于’r’的左子树进行同样的操作。
- 最后返回新创建的节点’newRoot’。

3.1.3 主要操作

标记二叉树：

先根据题意梳理一遍标记过程，如果一个结点标记为（a，b），则它的左孩子（如果存在）标记为（a+b，b），它的右孩子（如果存在）标记为（a，a+b）。那么从（1，1）根节点开始的树每一个标记的点的数值都是固定的，如图 1、图 2 所示。

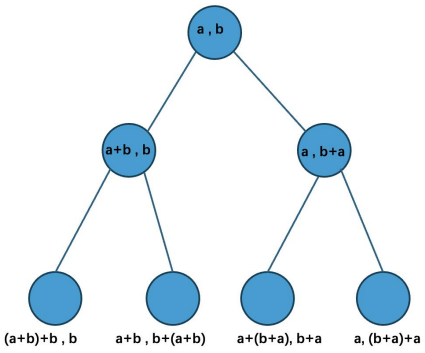

图 1

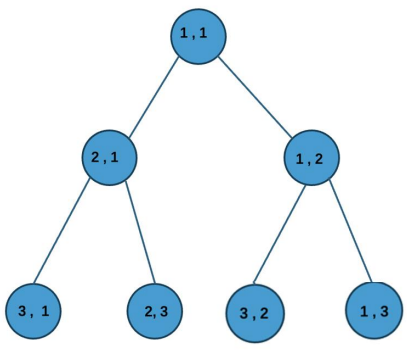

图 2

在实现每一步标记的时候，都是将一边加到另一边，所以我们可以非常容易的根据只一个节点的标记数字，来判断它上一步实现的操作，近而可以还原至上一节点状态，以此类推，一直到根节点。代码思路梳理如图 3 所示(以 2,3 为例)。

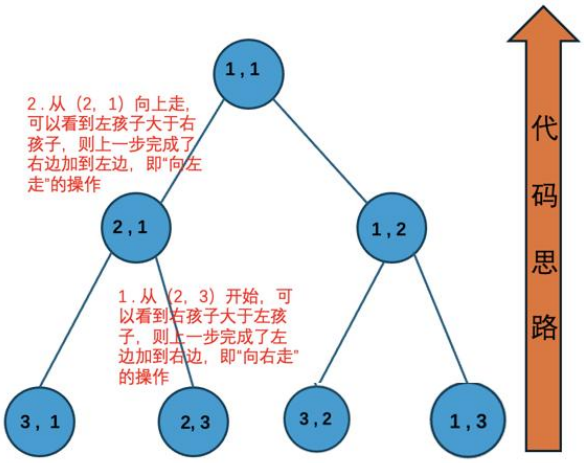

图 3

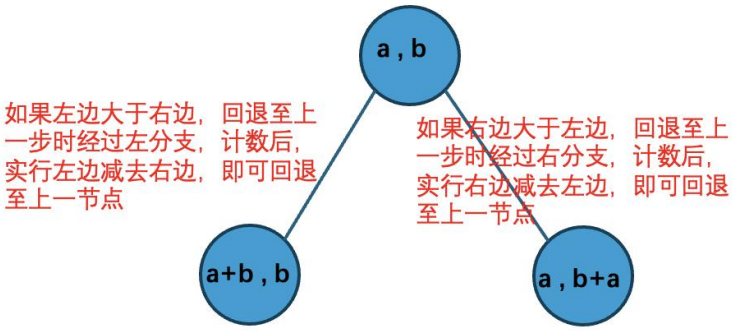

图 4

进而可以用循环来实现该操作，每一次的循环思路如图 4 所示。

3.2 源程序代码

以下简介几个该程序的核心代码，其中包括：左右子树的交换，标记二叉树。

3.2.1 左右子树的交换——Revolute 函数

交换左右子树，其实就是对于每一个节点做 leftchild 和 rightchild 的交换。为了读取每一个节点，其必要条件就是进行树的遍历。而树的遍历有四种方式：先序遍历，中序遍历，后序遍历和层次遍历。四种遍历方式理论上均可成立，在本题中，为了让代码更可观，我们小组采取后序遍历，通过递归的方式达到该目的。代码如下所示：

```
void BinaryTree::Revolute(BinTreeNode *p)
{
    if(p!=NULL)
    {
        Revolute(p->leftchild);
        Revolute(p->rightchild);
        BinTreeNode *tmp;              //左右孩子交换位置
        tmp=p->leftchild;
        p->leftchild=p->rightchild;
        p->rightchild=tmp;
    }
}
```

3.2.2 在未建立的树中寻找左右分支数——BranchCount 函数

1.处理方式：在 a! =b 循环中：比较 a 与 b 大小确定该节点是左孩子还是右孩子，后分别进行 a-=b 或 b-=a 操作回到父节点，再给 leftcount 或 rightcount 自增计数；直至 a=b=1 回到根节点，从而 leftcount 和 rightcount 是经过的左分支和右分支次数。重复 n 次得到 n 组数据结果。

2.特判条件：当 n>max 时，超出最大组数。当 a<1 或 b<1 或 a=b 且 a!=1 时，标记数据有误。

代码如下所示：

```
void BranchCount()
{
    system("cls");
    cout << "**********left and right branch counts*********" << endl;
    int n,a,b;
    cout << "Group Number:";
    cin >> n;       //数据组数
    if(n<0||n>MAX)
    {
        cout << "Max size!" << endl;
        Sleep(SleepTime);
        return;
    }
    for(int i=0;i<n;i++)
    {
        int leftcount=0,rightcount=0;
        cout << "a and b:";
        cin >> a >> b;
        if(a<1||b<1||(a==b&&a!=1))
        {
            cout << "Wrong number!" << endl;
            Sleep(SleepTime);
            return;
        }
        while(a!=b)
        {
            if(a>b)
            {
                a=a-b;
                ++leftcount;
            }
            else if(a<b)
            {
                b=b-a;
                ++rightcount;
            }
        }
        cout << "leftcount:" << leftcount << " rightcount:" << rightcount << endl;
    }
    Sleep(SleepTime);
```

3.2.3 在已建立的树中寻找左右分支数——TreeSearch 函数、Search 函数

1.处理方式：根据输入的 a，b 寻找结点：判断标记数是否符合搜索条件，若相同，输出其标记数据与经过的左右分支数，否则递归继续搜索。重复 n 次得到 n 组数据结果。

代码如下所示：

```
void TreeSearch(BinaryTree &BT)
{
    system("cls");
    cout << "**********Search a particular node*********" << endl;
    int n,a,b;
    cout << "Group Number:";
    cin >> n;       //数据组数
    if(n>=MAX) cout << "Max Size!" << endl;
    for(int i=0;i<n;i++)
    {
        cout << "a and b:";
        cin >> a >> b;
        if(!BT.Search(a,b)) i--;
    }
    Sleep(SleepTime);
}
```

```
void BinaryTree::Search(BinTreeNode *p,int ln,int rn,bool &found)          //寻找leftnum=ln，rightnum=rn的节点
{
    if(p!=NULL)
    {
        if(ln==p->leftnum && rn==p->rightnum)
        {
            cout << "The data of the node:" << p->data << endl;
            cout << "Left count:" << p->leftcount << endl;
            cout << "Right count:" << p->rightcount << endl;
            found=true;
            return;
        }
        Search(p->leftchild,ln,rn,found);
        Search(p->rightchild,ln,rn,found);
    }
}
```

3.2.4 节点的加入——Insert 函数

1.处理方式：插入的节点值小于该结点数据域的左孩子，则递归直至寻找到合适的插入位置；右孩子同理，代码仅显示小于节点数据域的情况。

2.适应本题：将节点连入二叉树时，及时更新从根节点到新增节点的路径，其中包括 a 与 b 的值以及如何行走到这个新增节点的，统计左分支数和右分支数。

```
void BinaryTree::Insert(BinTreeNode *p,int d)
{
    //连入链
    if(root==NULL)
    {
        root=new BinTreeNode(d);
        root->leftnum=root->rightnum=1;
        root->leftcount=root->rightcount=0;
    }
    else
    {
        if(d<p->GetData())
        {
            if(p->leftchild==NULL)
            {
                BinTreeNode *q=new BinTreeNode(d);
                q->leftnum=p->leftnum+p->rightnum;       //更新新增节点a和b的值
                q->rightnum=p->rightnum;
                q->leftcount=p->leftcount+1;             //更新新增节点左右分支数量
                q->rightcount=p->rightcount;
                p->leftchild=q;                          //连入链
                return;
            }
            else Insert(p->leftchild,d);
        }
```

3.2.5 封装

为了实现函数的封装性，在 Search 函数、Print 函数等函数中均有所体现。将根节点作为传入函数的节点。保护了根节点的保护属性。

代码如下所示：

```
void BinaryTree::Insert(BinTreeNode *p,int d)

void BinaryTree::Insert(int d)
{
    Insert(root,d);
}

void BinaryTree::Revolute(BinTreeNode *p)

void BinaryTree::Revolute()
{
    Revolute(root);
}
```

3.3 运行结果

3.3.1 界面

```
题目要求的功能为选项4和6，其余是为了补充完整/审题不清而写的功能
**********Functions**********
1.Create or enlarge the tree
2.Print present tree
3.Search a particular node
4.Revolute the tree
5.Delete the tree
6.Branch count
****************************
Your choice:
```

3.3.2 选项 1 和 2：创立树并输出树（中序遍历）

```
**********Create or enlarge the tree**********
Tree Node Number:8
Tree Node Data:1 23 4 5 67 8 9 101
The tree has been created/enlarged!
```

```
**********Print Tree**********
The present tree:(InOrder)1 4 5 8 9 23 67 101
```

3.3.3 选项 1 和 2：插入节点并输出树（中序遍历）

```
**********Create or enlarge the tree**********
Tree Node Number:2
Tree Node Data:12 34
The tree has been created/enlarged!
```

```
**********Print Tree**********
The present tree:(InOrder)1 4 5 8 9 12 23 34 67 101
```

3.3.3 选项 4：转置树并输出树（中序遍历）

```
**********Revolute the tree*********
The tree has been revoluted!
```

```
**********Print Tree**********
The present tree:(InOrder)101 67 34 23 12 9 8 5 4 1
```

3.3.4 在建立的树中通过 a，b 查找左右分支和值

```
**********Search a particular node*********
Group Number:2
a and b:1 3
The data of the node:67
Left count:0
Right count:2
a and b:2 1
Node is not found in the tree!
a and b:3 2
The data of the node:4
Left count:1
Right count:1
```

3.3.5 删除树并输出树用以检查是否删除完毕

```
**********Delete the tree*********
The tree has been deleted!
```

```
**********Print Tree**********
The present tree:(InOrder)
```

3.3.6 在未建立的树中寻找左右分支数

```
**********left and right branch counts*********
Group Number:2
a and b:41 1
leftcount:40 rightcount:0
a and b:2 5
leftcount:1 rightcount:2
```

4 算法分析

4.1 算法时间复杂度分析

Revolute 函数：O(n)

BranchCount 函数：O(n * max(a, b))

Search 函数:O(n)

4.2 算法空间复杂度的分析

Revolute 函数：O(n)

BranchCount 函数：O(1)

Search 函数:O(n)

5 总结与心得

本次实验实现了二叉树左右子树交换与标记二叉树中的结点, 求从根结点开始需要经过多少次左分支和多少次右分支才能达到某一结点。我们小组采取中序遍历，交换每一结点的左右子树，通过递归交换整个二叉树的左右子树；通过比较 a 和 b 大小，将结点还原至其父结点以确定经过的是左分支还是右分支。

通过本次实验，我们熟悉了对二叉树这一数据结构的遍历等操作，加深了对二叉树类设计的理解。