**实习报告 6.4**

**平衡二叉树操作的演示**

陈彦帆 2018K8009918002

李国峰 2018K8009922027

一、需求分析

本程序演示平衡二叉搜索树（BBST）的基本操作，包括创建平衡二叉树，查找、插入结点，删除结点，打印BBST图形，合并两棵树，把一棵树分裂为两棵树。

为了支持演示多棵树的插入，本程序通过编号（0-999）确定待操作的平衡二叉树。编号的范围可以很容易地从程序调整。

每次插入、删除操作会自动打印出操作后的树形图。每次操作有成功和失败提示。

本程序用纯C语言完成。

二、概要设计

**抽象数据类型**

虽然本题要求的是平衡二叉树，但下列关于树和树结点的抽象定义也可适用于所有二叉树。

typedef struct BINNODE{

int height; //树的高度

int succ; //中序遍历下的下一个节点的编号

ElemType data;

struct BINNODE \*parent, \*lchild, \*rchild;

} BinNode;//二叉树结点

typedef struct{

int size;

BinNode \* root;

} BinTree;//二叉树

BinNode \* hot;//用于指示查找结点的父节点

**基本操作：**

1.对二叉查找树（BST）适用的基本操作，同样适用于BBST：

BinNode \* BST\_Search(BinTree \* T, ElemType e)

在T中查找e，返回找到的结点，并更新hot。

int UpdateHeight(BinNode \* v)

更新结点v的高度

void UpdateHeightAbove(BinNode \* v)

更新结点v及其祖先的高度

Status BST\_Insert(BinTree \* T, ElemType e)

在BST T中插入e，若成功返回True，若已存在则返回False

#define islchild(v) ((v)->parent&&(v)->parent->lchild == (v))

判断v是否为左孩子

BinNode \* BST\_succ(BinNode \* v)

返回v在中序遍历的下一个节点。

Status BST\_Remove(BinTree \* T, ElemType e)

在T中删除结点e。成功返回True，否则返回False。

BinTree \* BST\_Init()

创建新排序树并返回，失败返回NULL。

Status BST\_Destroy(BinTree \* T)

删除一棵树并释放空间。

void BST\_Print(BinTree \* T)

按类似树形图的方法打印二叉树T

2，对BBST适用的基本操作

void BBST\_RotateAt(BinTree \* T,BinNode \* v)

将v结点作为孙子算起的祖孙3结点旋转到合适形状。

Status BBST\_Insert(BinTree \* T, ElemType e)

在BBST中插入e，成功返回True。

Status BBST\_Remove(BinTree \* T, ElemType e)

在BBST中删除e，成功返回True。

Status BBST\_Merge(BinTree \* T1, BinTree \* T2)

将T2归并到T1，并摧毁T2，成功返回True

Status BBST\_Separate(BinTree \* T1, BinTree \* T2, BinTree \* T3,ElemType e)

将T1小于等于e的结点拆分到T2，其他节点拆分到T3，并摧毁T1.成功返回True。

**抽象数据类型**

在层次遍历时需要用到**队列**，故定义队列为双向链表，数据结构如下。

typedef BinNode LinkType;

typedef struct LNODE{

LinkType data;

struct LNODE \* next, \* front;

}LNode;

typedef struct{

LNode \* head, \*tail;

}Quene;

**基本操作：**

Quene \* Quene\_Init()

初始化队列并返回

Status Quene\_In(Quene \* Q, LinkType e)

将元素e入队到Q，成功返回True。

Status Quene\_isEmpty(Quene \* Q)

判断队列是否为空，若是返回True。

Status Quene\_Out(Quene \* Q, LinkType \* e)

出队，并将数值通过指针传给\*e。成功返回True。

Status Queue\_Destroy(Quene \* Q)

摧毁队列。成功返回True。

**本程序的结构为：**

基本操作的函数实现；

main函数：与用户进行交互，并调用以上基本操作的函数实现演示。

**三、详细设计**

重点说明书上没有的删除算法。

//计算平衡因子

#define BalanceFac(v) (GetHeight((v)->lchild) - GetHeight((v)->rchild))

//判断是否平衡

#define isBalance(v) ((-2<BalanceFac(v))&&(BalanceFac(v))<2)

BinNode \* BBST\_Connect34(BinNode \*a,BinNode \*b,BinNode \*c,

BinNode \*T0,BinNode \*T1,BinNode \*T2,BinNode \*T3)

{//将待调整的3个结点、四颗子树拆开，按中间节点在最上的顺序重组即可。

a->lchild = T0; if(T0) T0->parent = a;

a->rchild = T1; if(T1) T1->parent = a; UpdateHeight(a);

c->lchild = T2; if(T2) T2->parent = c;

c->rchild = T3; if(T3) T3->parent = c; UpdateHeight(c);

b->lchild = a; a->parent = b;

b->rchild = c; c->parent = b; UpdateHeight(b);

return b;

}

void BBST\_RotateAt(BinTree \* T,BinNode \* v)

{ //将失衡结点旋转为平衡，其中失衡结点为孙子

BinNode \* p = v->parent;

BinNode \* g = p->parent;

BinNode \* r = g->parent;//需要考虑祖父结点为根节点的情况，可能要调整根节点指针

if(islchild(p)){ //zig //分四种情况考虑

if(islchild(v)){ //zig-zig

p->parent = r;

if(r&&islchild(g)) r->lchild = p;

else if(r) r->rchild = p;

else T->root = p;

BBST\_Connect34(v,p,g,v->lchild,v->rchild,p->rchild,g->rchild);

}

else{ //zig-zag

v->parent = r;

if(r&&islchild(g)) r->lchild = v;

else if(r) r->rchild = v;

else T->root = v;

BBST\_Connect34(p,v,g,p->lchild,v->lchild,v->rchild,g->rchild);

}

}

else{

if(islchild(v)){ //zag-zig

v->parent = r;

if(r&&islchild(g)) r->lchild = v;

else if(r) r->rchild = v;

else T->root = v;

BBST\_Connect34(g,v,p,g->lchild,v->lchild,v->rchild,p->rchild);

}

else{ //zag-zag

p->parent = r;

if(r&&islchild(g)) r->lchild = p;

else if(r) r->rchild = p;

else T->root = p;

BBST\_Connect34(g,p,v,g->lchild,p->lchild,v->lchild,v->rchild);

}

}

}

//计算高度较大的孩子

#define tallerchild(v) (GetHeight((v)->lchild)>GetHeight((v)->rchild)?(v)->lchild:(v)->rchild)

Status BBST\_Remove(BinTree \* T, ElemType e)

{//BBST的删除主函数

BinNode \* v = BST\_Search(T,e);

if(!v) return False;

BST\_RemoveAt(T,v);

T->size--;

for(BinNode \* g = hot;g;){

if(!isBalance(g)){

BinNode \* r = g->parent;

BBST\_RotateAt(T,tallerchild(tallerchild(g)));

g = r; //与插入不同，删除节点的重平衡需要一直向上追溯。

}

else{

UpdateHeight(g);

g=g->parent;

}

}

//BST\_Print(T);

return True;

}

四、调试分析

主要描述几个实现难点。

1，查找、插入算法等。

这些基础算法在课本上都有讲解，这里不再赘述。需要注意的是，BST的查找算法同样适用于BBST。而BBST的插入算法可先调用BST的插入算法，再进行调整旋转即可。

2，删除算法

课本上已有插入算法的设计。在此基础上，考虑到BBST在删除并调整完一个节点后，其祖先仍可能失去平衡，故只需一直向祖先检查是否失衡，若是，则按插入算法的方法调整重平衡。复杂度是O（logn）的。

3，重平衡算法的实现

在本程序中没有采用教材上的左右旋转的算法，而参考了清华邓俊晖老师的方法：将待调整的3个结点、四颗子树拆开重组即可。每次调整的复杂度为O（1）

4，合并两棵树

只需遍历其中一棵树，依次插入另一棵树即可。遍历采用先序遍历，可以减少插入时调整的次数。最后把被合并的树删除。

5，拆分一棵树。

只需遍历这棵树，看结点于元素e的关系选择插入第二棵树或者第三颗树即可。

6，按树形打印二叉树

我选择的是控制台打印而非图形界面。我的做法是，中序遍历可以确定每个节点距离左侧的空格数（按遍历顺序依次递增），再通过层次遍历即可逐层打印出树形。层次遍历需要用到队列，由于C语言没有现成的结构，于是手动实现了一个双向链表作为队列。

7.对创建平衡二叉树，插入结点，删除结点，打印BBST图形，合并两棵树，把一棵树分裂为两棵树各种操作进行测试，程序运行良好，具体见测试结果部分。

五、用户手册

1，本程序可在编号0-999范围内插入1000棵BBST树并进行操作。考虑到内存的限制，1000棵的数量已经足够，不需要增加。

,2，本程序运行环境为Windows系统，采用控制台窗口。

3，进入程序后，显示Welcome to BBST demo, Please created a BBST first(0-999).此时需要用户输入以进行交互。每执行一个操作，按回车即可看到结果。

支持的输入操作示例如下：

**创建c**

示例：c 1 2 3

创建1、2、3号BBST树

**插入i：**

示例：i 4 5 6 7

向4号BBST插入元素5、6、7，并打印。若BBST未创建，将会提示插入失败。

**查找f**

示例：f 4 4 5

向4号BBST查找元素4、5，结果返回是否找到。

**删除r**

示例：r 4 5 6

向4号BBST删除元素5、6，并打印。若BBST未创建，将会提示删除失败。

**合并m**

示例：m 3 4

把4号BBST归并到3，并摧毁4号BBST，然后打印3号BBST。

**拆分s**

示例：s 4 5 6 7

把4号BBST拆分到5和6号，小于等于e的元素归到5号，其余到6号。若5和6号未创建，将会自动创建。然后摧毁4号BBST，

**打印p**

示例：p 4

打印4号二叉树。

**摧毁d**

示例：d 1 2 3

摧毁1、2、3号二叉树。

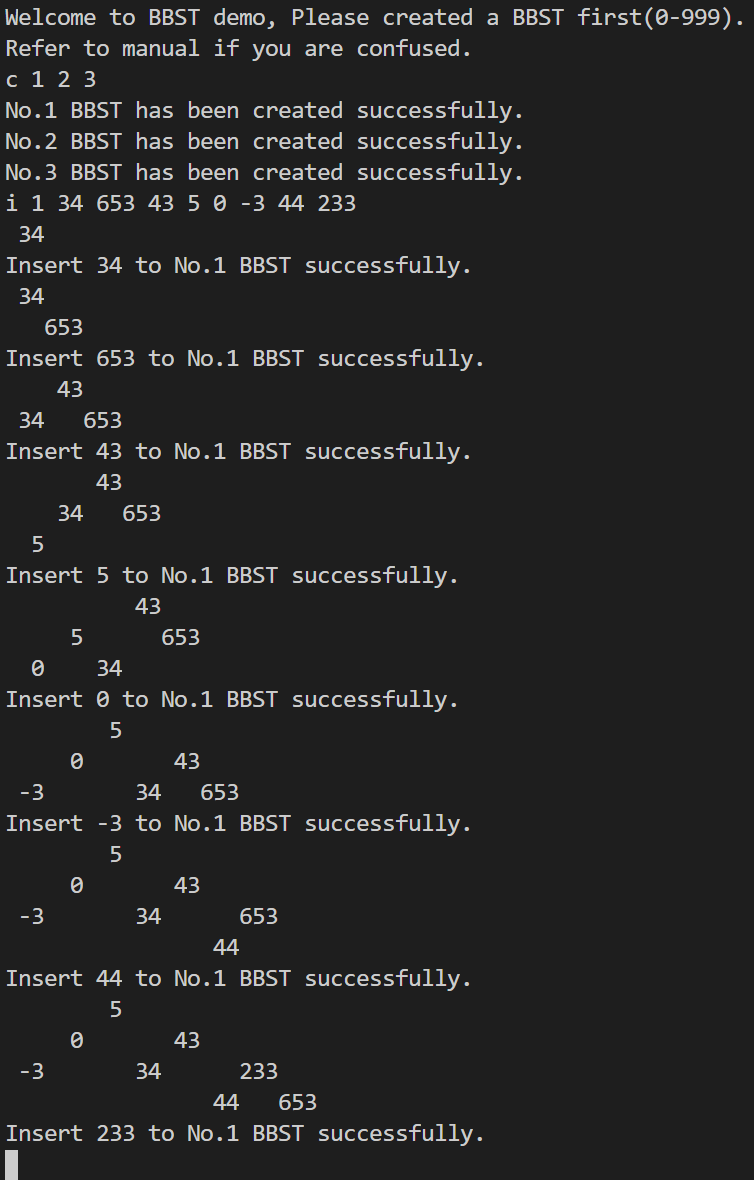
**退出q**

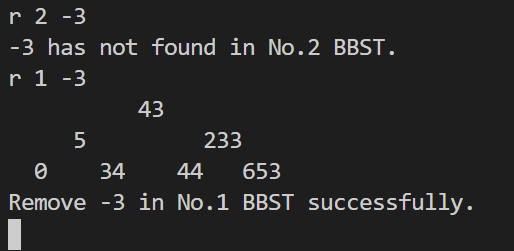
示例：q

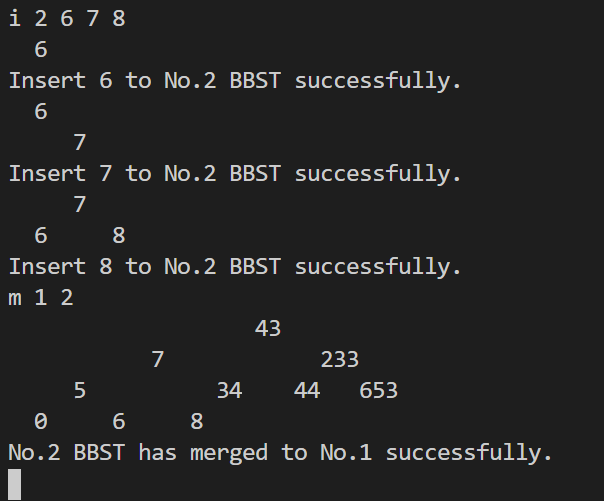
退出演示程序。

六、测试结果

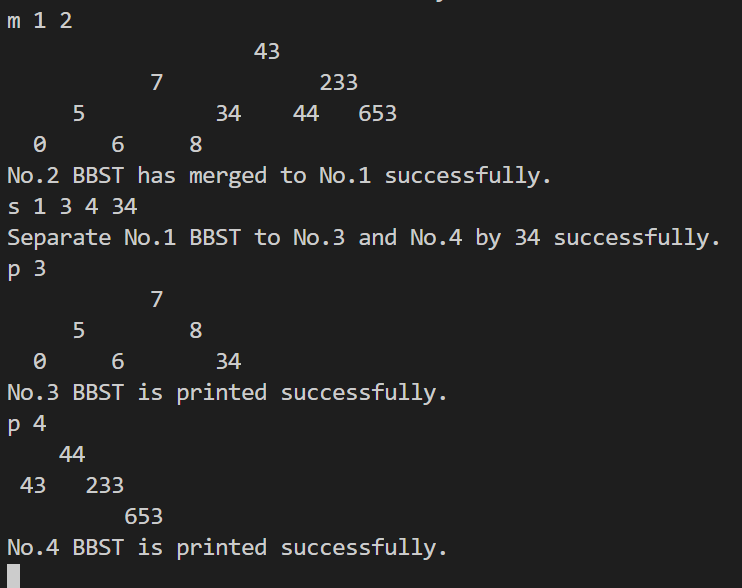
（见下页）

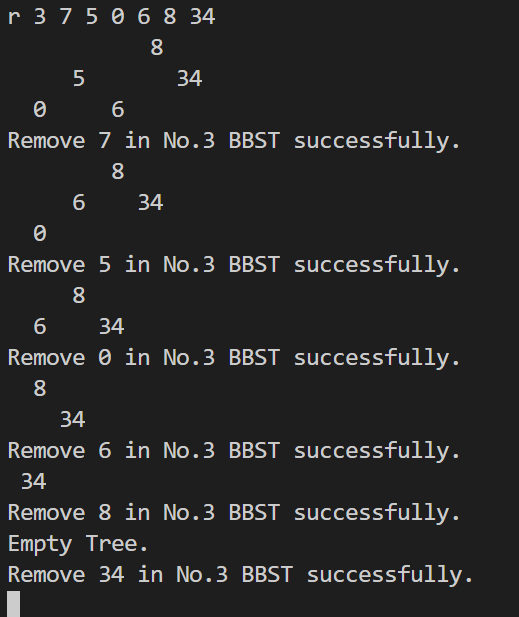
创建和插入：

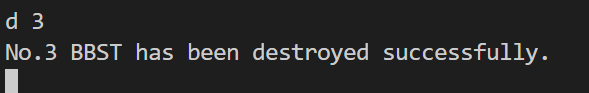
删除：

向树2插入并合并1、2：

拆分1到3、4，并打印：



依次删除3的结点：

摧毁：

退出：

以上测试结果均验证正确。

七、附录

源程序清单：

BBST.c

可执行程序清单：

BBST.exe

i 1 2 4 6 8 10

i 2 783 137 7 3 22 -98 0 99