数据结构上机报告

题目：平衡二叉树操作的演示

班级：04班 姓名： 学号： 完成日期：2020.12.18

一、需求分析

1.问题描述

利用平衡二叉树实现一个动态查找表

2.基本要求

实现动态查找表的三种基本功能：查找、插入和删除。

3.预设需求

（1）实现图形界面化操作，并能在系统中直接建立二叉树并进行查找、插入、删除操作。

（2）平衡二叉树通过图形界面画出，并实现每部操作的即使存储和可重复性。

二、概要设计:

程序用结构体数组类型存储平衡二叉树，先输出操作界面及其基本功能，并根据用户的相应选择实现相应操作。

1. 平衡二叉树的定义类型

typedef struct BSTNode {

int data;

int bf;

struct BSTNode \*lchild,\*rchild;

} BSTNode,\*BSTree;

2、基本操作函数：

void R\_Rotate (BSTree &p) { //对以\*p为根的二叉排序树作右旋处理

BSTree lc;

lc=p->lchild;

p->lchild=lc->rchild;

lc->rchild=p;

p=lc;

}

void L\_Rotate(BSTree &p) { //对以\*p为根的二叉排序树作左旋处理

BSTree rc;

rc=p->rchild;

p->rchild=rc->lchild;

rc->lchild=p;

p=rc;

}

3、主程序流程

a.首先输出基本操作方式以及图形界面，并根据用户输入的数据新建平衡二叉树。

b.根据用户输入的操作进行相应处理，并在操作完成后存储当前的二叉树信息。

c.将当前的二叉树打印，并询问是否进行下一步操作。

三、详细设计

各操作实现函数

void LeftBalance(BSTree &T) { //对以指针Ｔ所指结点为根的二叉树作左平衡旋转处理

BSTree lc,rd;

lc=T->lchild;

switch(lc->bf) {

case LH:

T->bf=lc->bf=EH;

R\_Rotate(T);

break;

case RH:

rd=lc->rchild;

switch(rd->bf) {

case LH:

T->bf=RH;

lc->bf=EH;

break;

case EH:

T->bf=lc->bf=EH;

break;

case RH:

T->bf=EH;

lc->bf=LH;

break;

}

rd->bf=EH;

L\_Rotate(T->lchild);

R\_Rotate(T);

}

}

void RightBalance(BSTree &T) { //对以指针Ｔ所指结点为根的二叉树作右平衡旋转处理

BSTree rc,ld;

rc=T->rchild;

switch(rc->bf) {

case RH:

T->bf=rc->bf=EH;

L\_Rotate(T);

break;

case LH:

ld=rc->lchild;

switch(ld->bf) {

case RH:

T->bf=LH;

rc->bf=EH;

break;

case EH:

T->bf=rc->bf=EH;

break;

case LH:

T->bf=EH;

rc->bf=RH;

break;

}

ld->bf=EH;

R\_Rotate(T->rchild);

L\_Rotate(T);

}

}

bool InsertAVL(BSTree &T,int e,bool &taller) { //插入结点e

if(!T) {

T=(BSTree)malloc(sizeof(BSTNode));

T->data=e;

T->lchild=T->rchild=NULL;

T->bf=EH;

taller=true;

} else {

if(e==T->data) {

taller=false;

printf("已存在相同关键字的结点!\n");

return 0;

}

if(e<T->data) {

if(!InsertAVL(T->lchild,e,taller))

return 0;

if(taller)

switch(T->bf) {

case LH:

LeftBalance(T);

taller=false;

break;

case EH:

T->bf=LH;

taller=true;

break;

case RH:

T->bf=EH;

taller=false;

break;

}

} else {

if(!InsertAVL(T->rchild,e,taller))

return 0;

if(taller)

switch(T->bf) {

case LH:

T->bf=EH;

taller=false;

break;

case EH:

T->bf=RH;

taller=true;

break;

case RH:

RightBalance(T);

taller=false;

break;

}

}

}

}

bool SearchBST(BSTree &T,int key) { //查找元素key是否在树Ｔ中

if(!T)

return false;

else if(key==T->data)

return true;

else if(key<T->data)

return SearchBST(T->lchild,key);

else

return SearchBST(T->rchild,key);

}

void LeftBalance\_div(BSTree &p,int &shorter) { //删除结点时左平衡旋转处理

BSTree p1,p2;

if(p->bf==1) {

p->bf=0;

shorter=1;

} else if(p->bf==0) {

p->bf=-1;

shorter=0;

} else {

p1=p->rchild;

if(p1->bf==0) {

L\_Rotate(p);

p1->bf=1;

p->bf=-1;

shorter=0;

} else if(p1->bf==-1) {

L\_Rotate(p);

p1->bf=p->bf=0;

shorter=1;

} else {

p2=p1->lchild;

p1->lchild=p2->rchild;

p2->rchild=p1;

p->rchild=p2->lchild;

p2->lchild=p;

if(p2->bf==0) {

p->bf=0;

p1->bf=0;

} else if(p2->bf==-1) {

p->bf=1;

p1->bf=0;

} else {

p->bf=0;

p1->bf=-1;

}

p2->bf=0;

p=p2;

shorter=1;

}

}

}

void RightBalance\_div(BSTree &p,int &shorter) { //删除结点时右平衡旋转处理

BSTree p1,p2;

if(p->bf==-1) {

p->bf=0;

shorter=1;

} else if(p->bf==0) {

p->bf=1;

shorter=0;

} else {

p1=p->lchild;

if(p1->bf==0) {

R\_Rotate(p);

p1->bf=-1;

p->bf=1;

shorter=0;

} else if(p1->bf==1) {

R\_Rotate(p);

p1->bf=p->bf=0;

shorter=1;

} else {

p2=p1->rchild;

p1->rchild=p2->lchild;

p2->lchild=p1;

p->lchild=p2->rchild;

p2->rchild=p;

if(p2->bf==0) {

p->bf=0;

p1->bf=0;

} else if(p2->bf==1) {

p->bf=-1;

p1->bf=0;

} else {

p->bf=0;

p1->bf=1;

}

p2->bf=0;

p=p2;

shorter=1;

}

}

}

void Delete(BSTree q,BSTree &r,int &shorter) { //删除结点

if(r->rchild==NULL) {

q->data=r->data;

q=r;

r=r->lchild;

free(q);

shorter=1;

} else {

Delete(q,r->rchild,shorter);

if(shorter==1)

RightBalance\_div(r,shorter);

}

}

int DeleteAVL(BSTree &p,int x,int &shorter) { //平衡二叉树的删除操作

int k;

BSTree q;

if(p==NULL) {

printf("不存在要删除的关键字!\n");

return 0;

} else if(x<p->data) {

k=DeleteAVL(p->lchild,x,shorter);

if(shorter==1)

LeftBalance\_div(p,shorter);

return k;

} else if(x>p->data) {

k=DeleteAVL(p->rchild,x,shorter);

if(shorter==1)

RightBalance\_div(p,shorter);

return k;

} else {

q=p;

if(p->rchild==NULL) {

p=p->lchild;

free(q);

shorter=1;

} else if(p->lchild==NULL) {

p=p->rchild;

free(q);

shorter=1;

} else {

Delete(q,q->lchild,shorter);

if(shorter==1)

LeftBalance\_div(p,shorter);

p=q;

}

return 1;

}

}

void PrintBST(BSTree T,int depth) {

int i;

if(T->rchild)

PrintBST(T->rchild,depth+1);

for(i=1; i<=depth; i++)

printf(" ");

printf("%d\n",T->data);

if(T->lchild)

PrintBST(T->lchild,depth+1);

}

四、调试分析

1、调试问题

（1）对平衡二叉树的删除的算法设计程序存在很大问题。删除节点后需要对新的排序树平衡化，改变节点的信息，使之形成一棵新的平衡二叉树。

（2）主函数中的实参和子函数中的实参相等，造成调用该子函数时，虽然没有错误，但其功能不能正确的实现。改变该变量后程序成功实现各种功能。

（3）一些逻辑逻辑运算符书写不正确，造成实现的功能不正确或程序死循环。

1. 算法改进设想

目前程序存在图形界面设计较差、时间复杂度较高等问题，后期可通过完善图形化界面，增强程序健壮性，优化基本操作算法提高程序的可用性。

1. 经验和体会

熟悉了平衡二叉树的建立与基本操作，加深了对指针运算的理解。

五、用户使用说明

用户需要先输入节点创建二叉树，然后根据图形界面的提示输入相应操作。

六、测试结果

输入样例

请选择操作的编号：1

请输入关键字(以-114514结束建立平衡二叉树):23

请输入关键字(以-114514结束建立平衡二叉树):1456

请输入关键字(以-114514结束建立平衡二叉树):155

请输入关键字(以-114514结束建立平衡二叉树):-55

请输入关键字(以-114514结束建立平衡二叉树):5-

请输入关键字(以-114514结束建立平衡二叉树):51

请输入关键字(以-114514结束建立平衡二叉树):-256

请输入关键字(以-114514结束建立平衡二叉树):-214

请输入关键字(以-114514结束建立平衡二叉树):364

请输入关键字(以-114514结束建立平衡二叉树):22

请输入关键字(以-114514结束建立平衡二叉树):-535

请输入关键字(以-114514结束建立平衡二叉树):-200

请输入关键字(以-114514结束建立平衡二叉树):500

请输入关键字(以-114514结束建立平衡二叉树):144

请输入关键字(以-114514结束建立平衡二叉树):64

请输入关键字(以-114514结束建立平衡二叉树):-244

请输入关键字(以-114514结束建立平衡二叉树):2645

请输入关键字(以-114514结束建立平衡二叉树):-114514

继续吗? y/n: y

请选择操作的编号：3

请输入您要插入的关键字：55

继续吗? y/n: y

请选择操作的编号：2

继续吗? y/n: y

请选择操作的编号：4

请输入你要删除的关键字: 55

输出样例：

您创建的二叉树为

2645

1456

500

364

155

144

64

51

23

22

5

-55

-200

-214

-244

-256

-535

2645

1456

500

364

155

144

64

55

51

23

22

5

-55

-200

-214

-244

-256

-535

请输入您要查找的关键字：55

关键字55存在，查找成功!

请输入你要删除的关键字: 55

2645

1456

500

364

155

144

64

51

23

22

5

-55

-200

-214

-244

-256

-535

1. 附录

#include<stdio.h>

#include<stdlib.h>

#define LH +1

#define EH 0

#define RH -1

#define NULL 0

typedef struct BSTNode {

int data;

int bf;

struct BSTNode \*lchild,\*rchild;

} BSTNode,\*BSTree;

void CreatBST(BSTree &T);

void R\_Rotate (BSTree &p);

void L\_Rotate(BSTree &p);

void LeftBalance(BSTree &T);

void RightBalance(BSTree &T);

bool InsertAVL(BSTree &T,int e,bool &taller);

bool SearchBST(BSTree &T,int key);

void LeftBalance\_div(BSTree &p,int &shorter);

void RightBalance\_div(BSTree &p,int &shorter);

void Delete(BSTree q,BSTree &r,int &shorter);

int DeleteAVL(BSTree &p,int x,int &shorter);

void PrintBST(BSTree T,int depth);

int main() {

BSTree T;

int sear,cmd,depth;

char ch;

int shorter=0;

bool taller=false;

T=(BSTree)malloc(sizeof(BSTNode));

T=NULL;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*平衡二叉树的操作菜单\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 1--创建\n");

printf(" 2--查找\n");

printf(" 3--插入\n");

printf(" 4--删除\n");

printf(" 5--退出\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

do {

printf("\n请选择操作的编号：");

scanf("%d",&cmd);

getchar();

switch(cmd) {

case 1:

CreatBST(T);

break;

case 2:

printf("请输入您要查找的关键字：");

scanf("%d",&sear);

getchar();

if(SearchBST(T,sear)) printf("关键字%d存在，查找成功!\n",sear);

else printf("查找失败!\n");

break;

case 3:

printf("请输入您要插入的关键字：");

scanf("%d",&sear);

getchar;

InsertAVL(T,sear,taller);

depth=0;

PrintBST(T,depth);

break;

case 4:

depth=0;

printf("请输入你要删除的关键字: ");

scanf("%d",&sear);

getchar();

DeleteAVL(T,sear,shorter);

PrintBST(T,depth);

break;

case 5:

printf("结束!\n");

break;

default:

printf("输入错误!\n");

}

if(cmd==5)

break;

printf("\n继续吗? y/n: ");

scanf("%s",&ch);

getchar();

printf("\n");

} while(ch=='y');

printf("\n");

return 0;

}

void CreatBST(BSTree &T) {

int depth;

int e;

bool taller=false;

T = NULL;

printf("\n请输入关键字(以-114514结束建立平衡二叉树):");

scanf("%d",&e);

getchar();

while(e != -114514) {

InsertAVL(T,e,taller);

printf("\n请输入关键字(以-114514结束建立平衡二叉树):");

scanf("%d",&e);

getchar();

taller=false;

}

depth=0;

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 您创建的二叉树为\n");

if(T)

PrintBST(T,depth);

else

printf("这是一棵空树!\n");

}

void R\_Rotate (BSTree &p) { //对以\*p为根的二叉排序树作右旋处理

BSTree lc;

lc=p->lchild;

p->lchild=lc->rchild;

lc->rchild=p;

p=lc;

}

void L\_Rotate(BSTree &p) { //对以\*p为根的二叉排序树作左旋处理

BSTree rc;

rc=p->rchild;

p->rchild=rc->lchild;

rc->lchild=p;

p=rc;

}

void LeftBalance(BSTree &T) { //对以指针Ｔ所指结点为根的二叉树作左平衡旋转处理

BSTree lc,rd;

lc=T->lchild;

switch(lc->bf) {

case LH:

T->bf=lc->bf=EH;

R\_Rotate(T);

break;

case RH:

rd=lc->rchild;

switch(rd->bf) {

case LH:

T->bf=RH;

lc->bf=EH;

break;

case EH:

T->bf=lc->bf=EH;

break;

case RH:

T->bf=EH;

lc->bf=LH;

break;

}

rd->bf=EH;

L\_Rotate(T->lchild);

R\_Rotate(T);

}

}

void RightBalance(BSTree &T) { //对以指针Ｔ所指结点为根的二叉树作右平衡旋转处理

BSTree rc,ld;

rc=T->rchild;

switch(rc->bf) {

case RH:

T->bf=rc->bf=EH;

L\_Rotate(T);

break;

case LH:

ld=rc->lchild;

switch(ld->bf) {

case RH:

T->bf=LH;

rc->bf=EH;

break;

case EH:

T->bf=rc->bf=EH;

break;

case LH:

T->bf=EH;

rc->bf=RH;

break;

}

ld->bf=EH;

R\_Rotate(T->rchild);

L\_Rotate(T);

}

}

bool InsertAVL(BSTree &T,int e,bool &taller) { //插入结点e

if(!T) {

T=(BSTree)malloc(sizeof(BSTNode));

T->data=e;

T->lchild=T->rchild=NULL;

T->bf=EH;

taller=true;

} else {

if(e==T->data) {

taller=false;

printf("已存在相同关键字的结点!\n");

return 0;

}

if(e<T->data) {

if(!InsertAVL(T->lchild,e,taller))

return 0;

if(taller)

switch(T->bf) {

case LH:

LeftBalance(T);

taller=false;

break;

case EH:

T->bf=LH;

taller=true;

break;

case RH:

T->bf=EH;

taller=false;

break;

}

} else {

if(!InsertAVL(T->rchild,e,taller))

return 0;

if(taller)

switch(T->bf) {

case LH:

T->bf=EH;

taller=false;

break;

case EH:

T->bf=RH;

taller=true;

break;

case RH:

RightBalance(T);

taller=false;

break;

}

}

}

}

bool SearchBST(BSTree &T,int key) { //查找元素key是否在树Ｔ中

if(!T)

return false;

else if(key==T->data)

return true;

else if(key<T->data)

return SearchBST(T->lchild,key);

else

return SearchBST(T->rchild,key);

}

void LeftBalance\_div(BSTree &p,int &shorter) { //删除结点时左平衡旋转处理

BSTree p1,p2;

if(p->bf==1) {

p->bf=0;

shorter=1;

} else if(p->bf==0) {

p->bf=-1;

shorter=0;

} else {

p1=p->rchild;

if(p1->bf==0) {

L\_Rotate(p);

p1->bf=1;

p->bf=-1;

shorter=0;

} else if(p1->bf==-1) {

L\_Rotate(p);

p1->bf=p->bf=0;

shorter=1;

} else {

p2=p1->lchild;

p1->lchild=p2->rchild;

p2->rchild=p1;

p->rchild=p2->lchild;

p2->lchild=p;

if(p2->bf==0) {

p->bf=0;

p1->bf=0;

} else if(p2->bf==-1) {

p->bf=1;

p1->bf=0;

} else {

p->bf=0;

p1->bf=-1;

}

p2->bf=0;

p=p2;

shorter=1;

}

}

}

void RightBalance\_div(BSTree &p,int &shorter) { //删除结点时右平衡旋转处理

BSTree p1,p2;

if(p->bf==-1) {

p->bf=0;

shorter=1;

} else if(p->bf==0) {

p->bf=1;

shorter=0;

} else {

p1=p->lchild;

if(p1->bf==0) {

R\_Rotate(p);

p1->bf=-1;

p->bf=1;

shorter=0;

} else if(p1->bf==1) {

R\_Rotate(p);

p1->bf=p->bf=0;

shorter=1;

} else {

p2=p1->rchild;

p1->rchild=p2->lchild;

p2->lchild=p1;

p->lchild=p2->rchild;

p2->rchild=p;

if(p2->bf==0) {

p->bf=0;

p1->bf=0;

} else if(p2->bf==1) {

p->bf=-1;

p1->bf=0;

} else {

p->bf=0;

p1->bf=1;

}

p2->bf=0;

p=p2;

shorter=1;

}

}

}

void Delete(BSTree q,BSTree &r,int &shorter) { //删除结点

if(r->rchild==NULL) {

q->data=r->data;

q=r;

r=r->lchild;

free(q);

shorter=1;

} else {

Delete(q,r->rchild,shorter);

if(shorter==1)

RightBalance\_div(r,shorter);

}

}

int DeleteAVL(BSTree &p,int x,int &shorter) { //平衡二叉树的删除操作

int k;

BSTree q;

if(p==NULL) {

printf("不存在要删除的关键字!\n");

return 0;

} else if(x<p->data) {

k=DeleteAVL(p->lchild,x,shorter);

if(shorter==1)

LeftBalance\_div(p,shorter);

return k;

} else if(x>p->data) {

k=DeleteAVL(p->rchild,x,shorter);

if(shorter==1)

RightBalance\_div(p,shorter);

return k;

} else {

q=p;

if(p->rchild==NULL) {

p=p->lchild;

free(q);

shorter=1;

} else if(p->lchild==NULL) {

p=p->rchild;

free(q);

shorter=1;

} else {

Delete(q,q->lchild,shorter);

if(shorter==1)

LeftBalance\_div(p,shorter);

p=q;

}

return 1;

}

}

void PrintBST(BSTree T,int depth) {

int i;

if(T->rchild)

PrintBST(T->rchild,depth+1);

for(i=1; i<=depth; i++)

printf(" ");

printf("%d\n",T->data);

if(T->lchild)

PrintBST(T->lchild,depth+1);

}