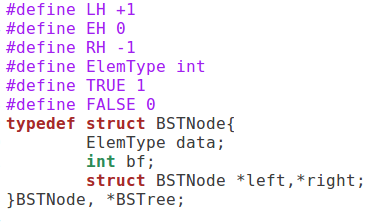
实验报告

实验6.4 平衡二叉树操作的展示

1. 需求分析：
2. 本实验利用平衡二叉树实现一个动态查找表，可以实现查找，插入，删除，合并，分裂五种操作。
3. 结点元素默认为整形，但可以定义为任意类型，只需更改宏定义与比较函数即可。
4. 本实验在linux的终端进行输入与输出，输出采用凹入表的形式。
5. 概要设计：
6. 抽象数据类型定义：



基本操作：

Build（）建立一棵平衡二叉树

Insert（）实现结点的插入

Delete（）实现结点的删除

LeftBalace（）对特定结点进行左平衡处理

RightBalance（）右平衡处理

L\_Rotate()左旋转

R\_Rotate()右旋转

1. 本程序包含两个模块
2. 主程序模块

Int main(){

初始化

Do{

接受命令；

处理命令；

}while（继续命令）

}

}

1. 各个具体操作的模块

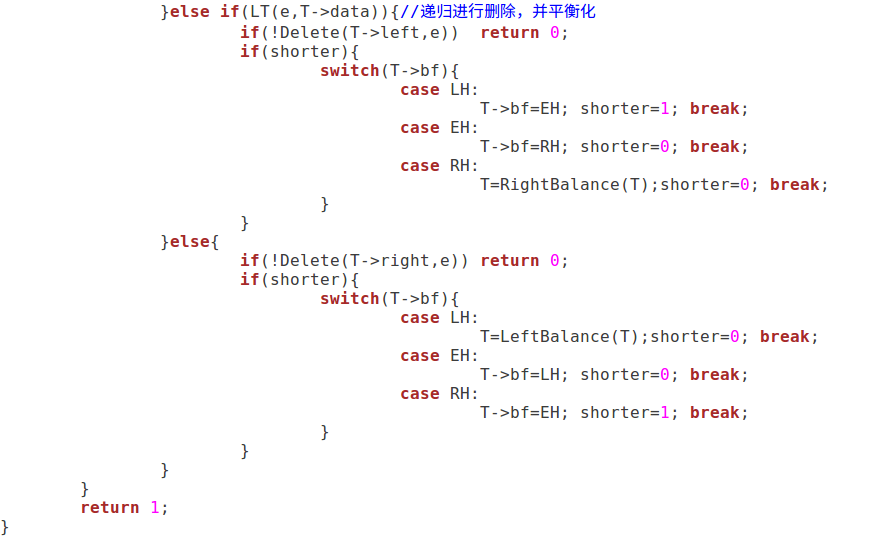
各模块之间的调用关系为主模块调用各个子模块。

1. 详细设计
2. 采用基本的二叉树链表存储结构
3. 插入函数：

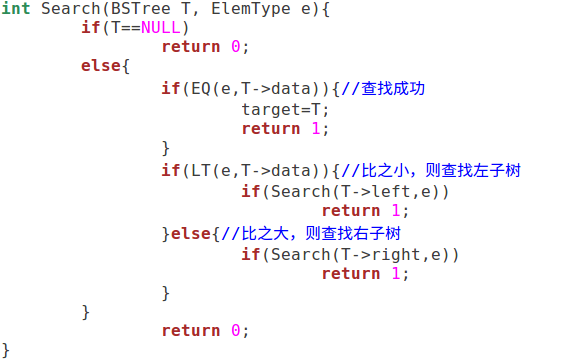


1. 删除函数：





查找：



1. 调试分析
2. 对随机输入的数据构造平衡二叉树，程序执行速度很快，基本感受不到延迟。
3. 由于查找、插入、平衡处理在书上均有详细代码，所以实现较为轻松。但删除需要自己思考，仔细思考插入的代码，举一反三，将思路应用到删除上即可。其实插入与删除均可视为两部分，第一部分是递归查找，确定位置，第二部分是进行平衡处理，此时插入和删除恰好互逆。左子树长高正对应于右子树变矮。
4. 合并与分裂操作均可基于插入函数上实现。
5. 用户手册

进入程序后按照提示进行具体操作即可。

1代表建立平衡二叉树，可一次性输入多个值

2代表插入，依次输入结点

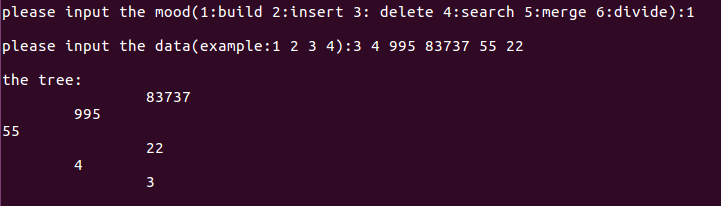
3代表删除，依次输入想要删除的值

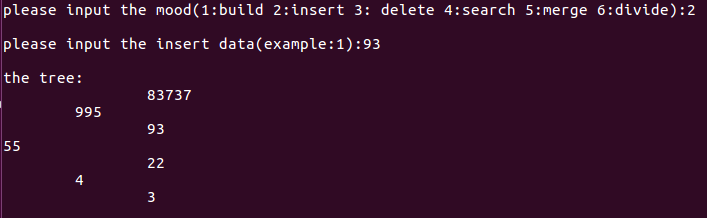
4 代表查找，返回查找成功/失败

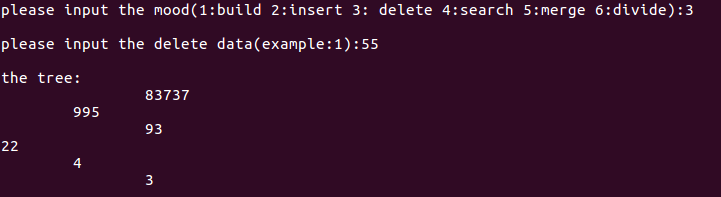
5代表合并，合并两棵平衡二叉树

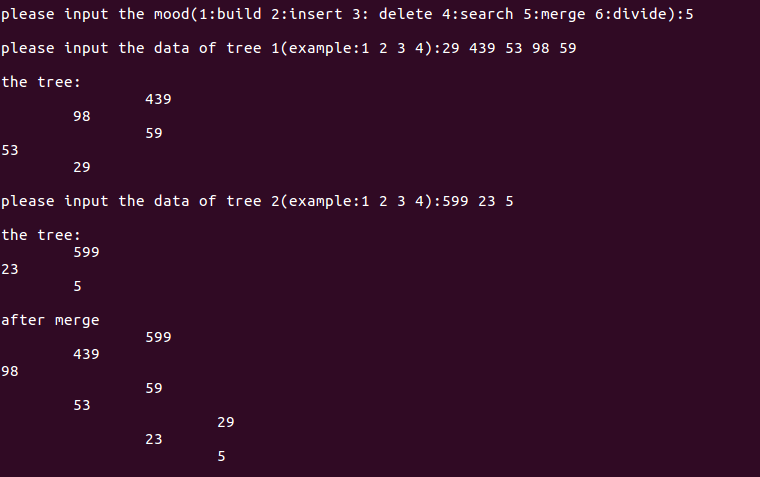
6 代表分裂，将一棵二叉树分裂成两棵。

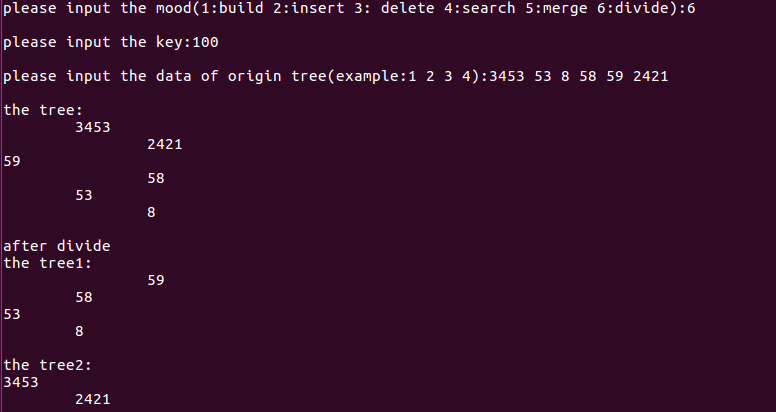
1. 测试结果：











1. 附录

源程序代码：

#include<stdio.h>

#include<stdlib.h>

#define LH +1

#define EH 0

#define RH -1

#define ElemType int

#define TRUE 1

#define FALSE 0

typedef struct BSTNode{

ElemType data;

int bf;

struct BSTNode \*left,\*right;

}BSTNode, \*BSTree;

BSTree target=NULL;

int taller=0;

int shorter=0;

int mode=0;

BSTree root,root1,root2;

BSTree R\_Rotate(BSTree p){

BSTree lc=p->left;

p->left=lc->right;

lc->right=p;

return lc;

}

BSTree L\_Rotate(BSTree p){

BSTree rc=p->right;

p->right=rc->left;

rc->left=p;

return rc;

}

BSTree LeftBalance(BSTree T){

BSTree lc=T->left;

BSTree rd=lc->right;

switch(lc->bf){

case LH:

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

T=R\_Rotate(T);

break;

case RH:

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;

T->left=L\_Rotate(T->left);

T=R\_Rotate(T);

}

return T;

}

BSTree RightBalance(BSTree T){

BSTree rc=T->right;

BSTree ld=rc->left;

switch(rc->bf){

case RH:

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

T=L\_Rotate(T);

break;

case LH:

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;

T->right=R\_Rotate(T->right);

T=L\_Rotate(T);

}

return T;

}

int EQ(ElemType a, ElemType b){

if(a==b)

return 1;

else

return 0;

}

int LT(ElemType a, ElemType b){

if(a<b)

return 1;

else

return 0;

}

int Delete(BSTree &T,ElemType e){

if(T==NULL)

return 0;

else{

BSTree Q=T;

if(EQ(e,T->data)){

if(T->left==NULL){//如果左右子树为空，则直接替换

T=T->right;

free(Q);

shorter=1;

}else if(T->right==NULL){

T=T->left;

free(Q);

shorter=1;

}else{//左右子树均不为空

Q=Q->left;

while(Q->right){//找到左子树的最大值

Q=Q->right;

}

T->data=Q->data;

Delete(T->left,Q->data);//删除最大值对应的节点

if(shorter){//进行平衡操作

switch(T->bf){

case LH:

T->bf=EH; shorter=1; break;

case EH:

T->bf=RH; shorter=0; break;

case RH:

T=RightBalance(T);shorter=0; break;

}

}

}

}else if(LT(e,T->data)){//递归进行删除，并平衡化

if(!Delete(T->left,e)) return 0;

if(shorter){

switch(T->bf){

case LH:

T->bf=EH; shorter=1; break;

case EH:

T->bf=RH; shorter=0; break;

case RH:

T=RightBalance(T);shorter=0; break;

}

}

}else{

if(!Delete(T->right,e)) return 0;

if(shorter){

switch(T->bf){

case LH:

T=LeftBalance(T);shorter=0; break;

case EH:

T->bf=LH; shorter=0; break;

case RH:

T->bf=EH; shorter=1; break;

}

}

}

}

return 1;

}

int insert(BSTree &T,ElemType e){

if(T==NULL){

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

T->right=T->left=NULL;

T->bf=EH;

T->data=e;

taller=1;

}else{

if(EQ(e,T->data)){

taller=FALSE;

return 0;

}

if(LT(e,T->data)){//递归插入，并进行平衡化

if(!insert(T->left,e))

return 0;

if(taller)

switch(T->bf){

case LH:

T=LeftBalance(T); taller=FALSE; break;

case EH:

T->bf=LH; taller=TRUE; break;

case RH:

T->bf=EH; taller=FALSE; break;

}

}else{

if(!insert(T->right,e))

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:

T=RightBalance(T); taller=FALSE; break;

}

}

}

return 1;

}

int Search(BSTree T, ElemType e){

if(T==NULL)

return 0;

else{

if(EQ(e,T->data)){//查找成功

target=T;

return 1;

}

if(LT(e,T->data)){//比之小，则查找左子树

if(Search(T->left,e))

return 1;

}else{//比之大，则查找右子树

if(Search(T->right,e))

return 1;

}

}

return 0;

}

void Print(BSTree T,int depth){//以凹入表形式输出

if(T){

if(T->right)

Print(T->right,depth+1);

for(int i=0;i<depth;i++)//展现凹入深度

printf("\t");

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

if(T->left)

Print(T->left,depth+1);

}

}

int main(){

root=NULL;

root1=NULL;

root2=NULL;

int temp,key;

char c;

char conti;

do{

printf("\nplease input the mood(1:build 2:insert 3: delete 4:search 5:merge 6:divide):");

scanf("%d",&mode);

switch(mode){

case 1://建立树

printf("\nplease input the data(example:1 2 3 4):");

do{

scanf("%d%c",&temp,&c);

insert(root,temp);

}while(c!='\n');

printf("\nthe tree:\n");

Print(root,0);

printf("\ncontinue? y/n:");

scanf("%c",&conti);

break;

case 2://插入

printf("\nplease input the insert data(example:1):");

scanf("%d%c",&temp,&c);

insert(root,temp);

printf("\nthe tree:\n");

Print(root,0);

printf("\ncontinue? y/n:");

scanf("%c",&conti);

break;

case 3://删除

printf("\nplease input the delete data(example:1):");

scanf("%d%c",&temp,&c);

Delete(root,temp);

printf("\nthe tree:\n");

Print(root,0);

printf("\ncontinue? y/n:");

scanf("%c",&conti);

break;

case 4://查找

printf("\nplease input the search data(example:1):");

scanf("%d%c",&temp,&c);

if(Search(root,temp)){

printf("\nexist!");

}else

printf("\nno exist!");

printf("\ncontinue? y/n:");

scanf("%c",&conti);

break;

case 5://合并

root1=root2=root=NULL;

printf("\nplease input the data of tree 1(example:1 2 3 4):");

do{

scanf("%d%c",&temp,&c);

insert(root1,temp);

}while(c!='\n');

printf("\nthe tree:\n");

Print(root1,0);

printf("\nplease input the data of tree 2(example:1 2 3 4):");

do{

scanf("%d%c",&temp,&c);

insert(root2,temp);

insert(root1,temp);

}while(c!='\n');

printf("\nthe tree:\n");

Print(root2,0);

printf("\nafter merge\n");

Print(root1,0);

root1=root2=root=NULL;

printf("\ncontinue? y/n:");

scanf("%c",&conti);

break;

case 6://分解

root1=root2=root=NULL;

printf("\nplease input the key:");

scanf("%d",&key);

printf("\nplease input the data of origin tree(example:1 2 3 4):");

do{

scanf("%d%c",&temp,&c);

insert(root,temp);

if(temp<=key)

insert(root1,temp);

else

insert(root2,temp);

}while(c!='\n');

printf("\nthe tree:\n");

Print(root,0);

printf("\nafter divide");

printf("\nthe tree1:\n");

Print(root1,0);

printf("\nthe tree2:\n");

Print(root2,0);

root1=root2=root=NULL;

printf("\ncontinue? y/n:");

scanf("%c",&conti);

break;

}

}while(conti=='y');

return 0;

}