数据结构课程设计文档

平衡二叉树

小组人员：刘子欣、史文翰、舒星源

1. **问题描述**

**输入一组数据，构造平衡二叉树，最后输出此平衡二叉树。**

**（为简化问题，结点加入左子树高度域lh，右子树高度域rh，父指针域fa，为简化输出，只输出最后的树形，并输出每一次平衡旋转时所做出的操作）**

1. **算法思路**

**建立平衡二叉树，采用搜索-插入的算法思路。如果该元素检索后不存在，那么将它插入到适合的位置，插入后可能会引起平衡树的失衡，那么需要通过旋转操作重新实之平衡。如果该元素已经存在，即检索成功，那么不再进行插入操作。**

1. **核心算法描述**

**bool insertBST(BSTree &T,int x)**

**{**

**if(searchBST(T,x,p))**

**{//T是该平衡树的树根，x为待查元素，\*p为待查元素的父结点**

**//该函数返回一个bool值，如果搜索成功则返回1**

**如果搜索成功，不做插入；**

**}**

**else**

**{**

**建立新结点；**

**if(!p)**

**{**

**如果p为空，那么建立平衡树的树根；**

**}**

**else**

**{**

**if(\*p的data>x)**

**{**

**该结点插在\*p的左侧；**

**}**

**else**

**{**

**该结点插在\*p的右侧；**

**}**

**}**

**while(插入节点的父指针非空)**

**{//更新平衡树结点的左右树高**

**if(如果该结点的父结点的左孩子)**

**{**

**父结点的lh+1；**

**}**

**else**

**{**

**父结点的rh+1；**

**}**

**标记指针向上（树根）方向位移；**

**}**

**引入另一个标记指针，该指针从插入结点开始向上攀爬，直到指空或者遇到该结点的左右子树高度之差等于2；引入一个数组来存储沿途结点的左右子树高度差；**

**while(该标记指针非空)**

**{**

**switch(左右子树高度差)**

**{**

**case -2:**

**switch(路径数组中的上一个元素)**

**{**

**case -1:**

**以该结点为树根，左旋处理；**

**更新平衡树各节点的lh和rh；**

**break;**

**case 1:**

**以该结点的右子树为树根，右旋处理；**

**以该结点为树根，左旋处理；**

**更新平衡树各节点的lh和rh；**

**break;**

**default:**

**出现错误；**

**break;**

**}**

**goto jump;//跳出**

**case 2:**

**switch(路径数组中的上一个元素)**

**{**

**case -1:**

**以该结点的左子树为树根，左旋处理；**

**以该节点为树根，右旋处理；**

**更新平衡树各节点的lh和rh；**

**break;**

**case 1:**

**以该节点为树根，右旋处理；**

**更新平衡树各节点的lh和rh；**

**break;**

**default:**

**出现错误；**

**break;**

**}**

**goto jump;//跳出**

**default:**

**标记指针向树根攀爬；**

**路径数组更新；**

**break;**

**}**

**}**

**jump:**

**return 1;**

**}**

**}**

1. **源程序**

**#include<iostream>**

**using namespace std;**

**int const MAX\_NODES=10;**

**typedef struct BSTNode{**

**int lh,rh;**

**int data;**

**BSTNode \*fa,\*lc,\*rc;**

**}BSTNode,\* BSTree;**

**bool searchBST(BSTree &T,int x,BSTNode \*&p)**

**{ // if x exists,return 1**

**// if T is empty, return p=NULL**

**BSTNode \*t=T;**

**p=NULL;**

**while(t)**

**{**

**if(t->data==x)**

**return 1;**

**else**

**{**

**p=t;**

**if(t->data<x) t=t->rc;**

**else t=t->lc;**

**}**

**}**

**return 0;**

**}**

**bool L\_Rotate(BSTree &T)**

**{**

**BSTNode \*temp;**

**temp=T->rc;**

**T->rc=temp->lc;**

**if(T->rc) T->rc->fa=temp;**

**temp->lc=T;**

**T=temp;**

**T->lc->fa=T;**

**return 1;**

**}**

**bool R\_Rotate(BSTree &T)**

**{**

**BSTNode \*temp;**

**temp=T->lc;**

**T->lc=temp->rc;**

**if(T->lc) T->lc->fa=temp;**

**temp->rc=T;**

**T=temp;**

**T->rc->fa=T;**

**return 1;**

**}**

**bool R\_Rotate\_F(BSTree &T,BSTree &checkP)**

**{ //Because of the pointer "fa"**

**//Change the "fa" based on R\_Rotate**

**int tag=0;**

**BSTNode \*pfa;**

**pfa=checkP->fa;**

**if(pfa)**

**{**

**if(pfa->lc==checkP)**

**tag=-1;**

**else**

**tag=1;**

**}**

**else//Directly change the root "T"**

**{**

**if(T->lc==checkP)**

**tag=-1;**

**else**

**tag=1;**

**}**

**R\_Rotate(checkP);**

**checkP->fa=pfa;**

**if(pfa)**

**{**

**if(tag==-1)**

**pfa->lc=checkP;**

**else**

**pfa->rc=checkP;**

**}**

**else T=checkP;**

**return 1;**

**}**

**bool L\_Rotate\_F(BSTree &T,BSTree &checkP)**

**{ // the same with R\_Rotate\_F**

**int tag=0;**

**BSTNode \*pfa;**

**pfa=checkP->fa;**

**if(pfa)**

**{**

**if(pfa->lc==checkP)**

**tag=-1;**

**else**

**tag=1;**

**}**

**else**

**{**

**if(T->lc==checkP)**

**tag=-1;**

**else**

**tag=1;**

**}**

**L\_Rotate(checkP);**

**checkP->fa=pfa;**

**if(pfa)**

**{**

**if(tag==-1)**

**pfa->lc=checkP;**

**else**

**pfa->rc=checkP;**

**}**

**else T=checkP;**

**return 1;**

**}**

**int getLH(BSTree& );**

**int getRH(BSTree& );**

**int getLH(BSTree &T)**

**{**

**if(!T) return -1;**

**else**

**{ //Choose the higher one**

**int clh,crh;**

**clh=getLH(T->lc)+1;**

**crh=getRH(T->lc)+1;**

**return clh>crh? clh : crh;**

**}**

**}**

**int getRH(BSTree &T)**

**{**

**if(!T) return -1;**

**else**

**{ //The same with getLH**

**int clh,crh;**

**clh=getLH(T->rc)+1;**

**crh=getRH(T->rc)+1;**

**return clh>crh? clh : crh;**

**}**

**}**

**bool resetH(BSTree &T)**

**{ //Reset the "bf" with funcs getLH & getRH**

**if(T)**

**{**

**T->lh=getLH(T);**

**T->rh=getRH(T);**

**if(T->lc) resetH(T->lc);**

**if(T->rc) resetH(T->rc);**

**}**

**}**

**bool insertBST(BSTree &T,int x)**

**{**

**BSTNode \*p,\*temp;**

**int bal[255]={0};**

**int count=0;**

**int balance;**

**if(searchBST(T,x,p))**

**{ //x exists**

**cout<<"Existance element:"<<x<<endl;**

**return 0;**

**}**

**else**

**{**

**temp=new BSTNode;**

**temp->data=x;**

**temp->lh=temp->rh=0;**

**temp->fa=p;**

**temp->lc=NULL;**

**temp->rc=NULL;**

**if(!p)**

**{ //if the tree is empty**

**T=temp;**

**}**

**else//insert the node with x**

**{**

**if(p->data>x)**

**{**

**p->lc=temp;**

**}**

**else**

**{**

**p->rc=temp;**

**}**

**}**

**BSTNode \*checkP=temp;**

**while(temp->fa)//reset the lh & rh related to x**

**{**

**if(temp->fa->lc==temp)**

**{**

**temp->fa->lh++;**

**}**

**else**

**{**

**temp->fa->rh++;**

**}**

**temp=temp->fa;**

**}**

**while(checkP)**

**{ //use balance to calculate "bf"**

**balance=checkP->lh-checkP->rh;**

**switch(balance)**

**{**

**case -2:**

**switch(bal[--count])**

**{**

**case -1:**

**cout<<"Balance rotate:RR"<<endl;**

**L\_Rotate\_F(T,checkP);**

**resetH(T);**

**break;**

**case 1:**

**cout<<"Balance rotate:RL"<<endl;**

**R\_Rotate\_F(T,checkP->rc);**

**L\_Rotate\_F(T,checkP);**

**resetH(T);**

**break;**

**default:**

**cout<<"error"<<endl;**

**break;**

**}**

**goto jump;**

**case 2:**

**switch(bal[--count])**

**{**

**case -1:**

**cout<<"Balance rotate:LR"<<endl;**

**L\_Rotate\_F(T,checkP->lc);**

**R\_Rotate\_F(T,checkP);**

**resetH(T);**

**break;**

**case 1:**

**cout<<"Balance rotate:LL"<<endl;**

**R\_Rotate\_F(T,checkP);**

**resetH(T);**

**break;**

**default:**

**cout<<"error"<<endl;**

**break;**

**}**

**goto jump;**

**default:**

**bal[count++]=balance;**

**checkP=checkP->fa;**

**break;**

**}**

**}**

**jump:**

**return 1;**

**}**

**}**

**bool printTree(BSTree &T,int depth)**

**{ //It is equivalent with RDL**

**if(T)**

**{**

**printTree(T->rc,depth+1);**

**for(int i=0;i<=depth-1;i++)**

**{**

**cout<<" ";**

**}**

**//Node: lh\*data\*rh**

**cout<<T->lh<<"\*"<<T->data<<"\*"<<T->rh<<endl;**

**printTree(T->lc,depth+1);**

**}**

**}**

**int main(void)**

**{**

**cout<<"Getting the input from the file 'bst\_input.txt' "<<endl;**

**cout<<"10 elements by default."<<endl;**

**freopen("bst\_input.txt","r",stdin);**

**int x;**

**BSTree T=NULL;**

**for(int i=0;i<=MAX\_NODES-1;i++)**

**{**

**cin>>x;**

**insertBST(T,x);**

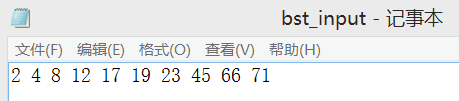
**}**

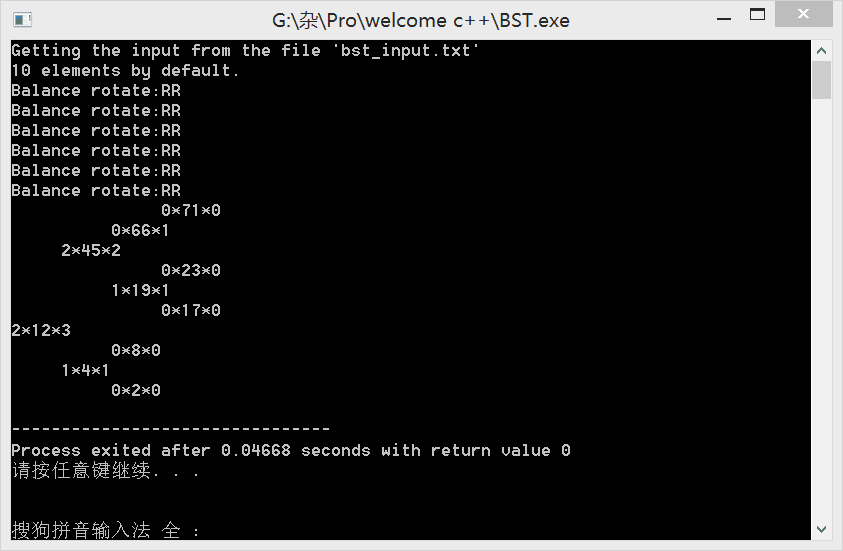
**printTree(T,0);**

**return 0;**

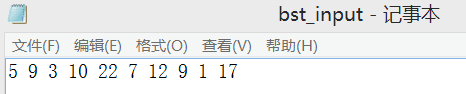
**}**

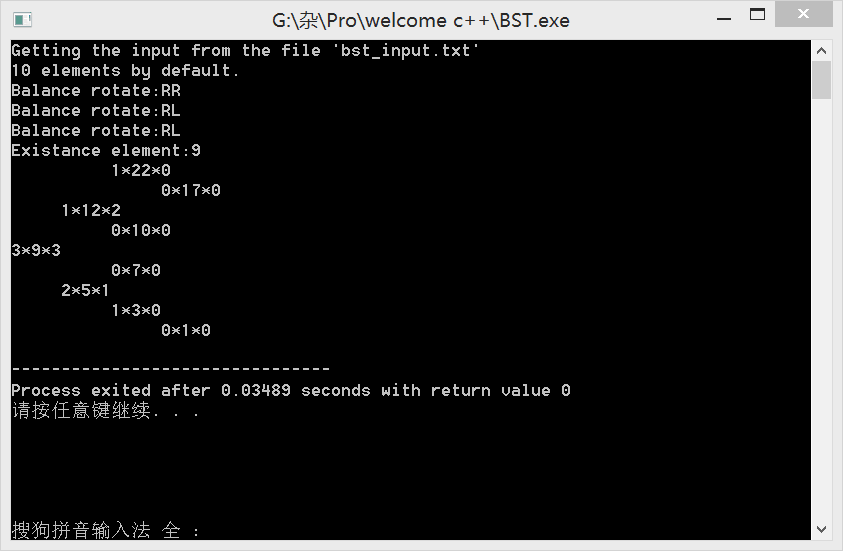
1. **测试数据及说明**
2. **RR倾向性测试（如无特殊说明，均以每组10个数据为默认数据输入大小）**

****

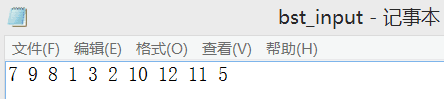
****

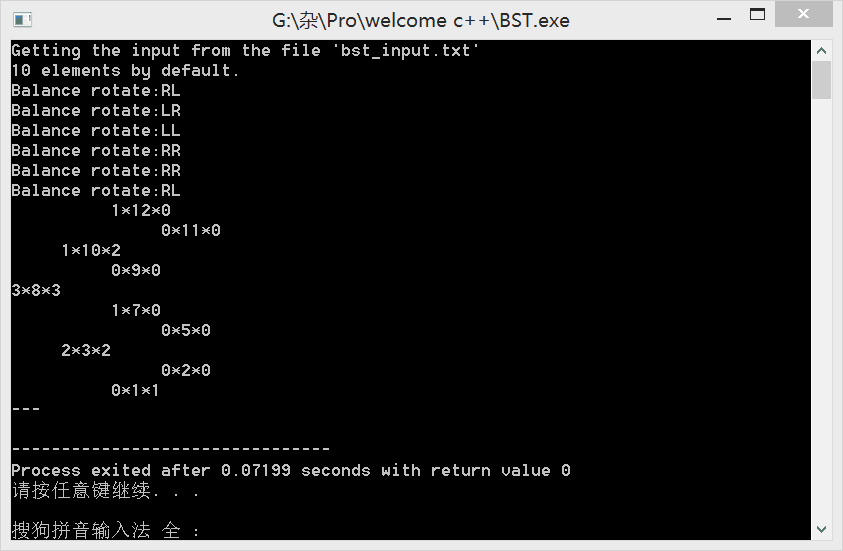
1. **RL倾向性测试**

****

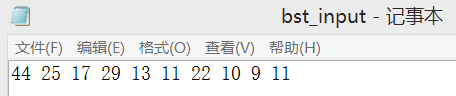
****

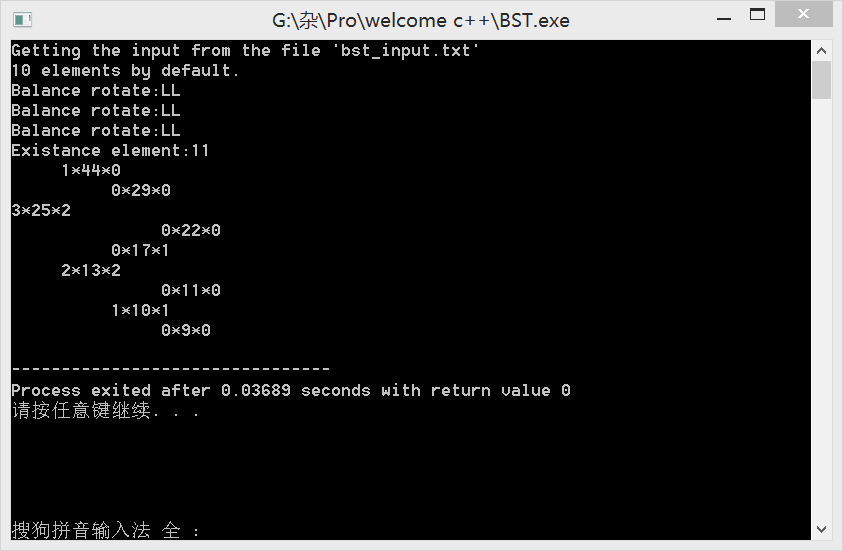
1. **综合倾向性测试**

****

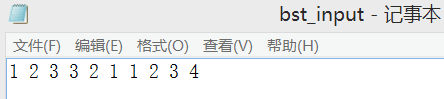
****

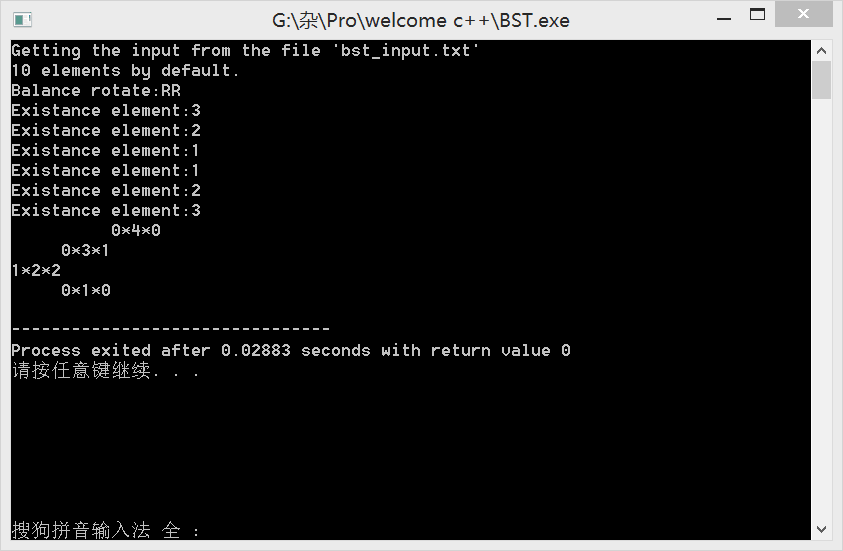
1. **带有重复数据的LL倾向性测试**

****

****

1. **重复数据输入测试**

****

****

1. **结果分析和结论**

**本次实验目的是建立平衡二叉树，首先我们要知道为什么要使搜索二叉树平衡，原因是这样可以使性能达到近似最优。由于要提高性能，一些技巧性的修改是必不可少的，这就是其中的左旋和右旋操作。其次，引入左右树高来代替平衡因子bf简化了运算和更新，引入了父结点简化了路径查找，但同时也加大了旋转操作时修改指针的难度，这里我们也看到了一个有趣的事实：不是所有的添枝加叶，都是完完全全会使事情简化的，有些简化，是要付出复杂化其他某些部分的代价的。**

1. **心得体会**

史文翰:在这次编码中我收获很大。二叉平衡搜索树的建立本身就是一个需要清晰逻辑思路的过程，徒手操作虽然简单，但是转化到程序上，需要一定的抽象能力。比如：单次左旋和右旋不难操作，但是如何把另外两种情况抽象成左旋右选的结合，就是一个逻辑上的强化（你当然可以重新写两个操作函数）。引入父指针，简化了路径的搜索，同时也加大了修改指针的难度，我们知道，当一个树涉及父指针以及变换操作的时候，本质上类似一个双向链表，因此，如果能引入头结点将会简化运算。我在考虑本此实验中的树是否也可以通过引入一个“树根头结点”来简化一些操作（比如涉及到根节点的旋转时需要判空），这次代码中没有具体实现，以后我会慢慢尝试。此外，在更新函数上，我们也有做的不足的地方，我们实际上相当于设置了三个递归函数来确保rh和lh的正确性，并在每一次旋转操作后从树根进行更新，有人提出有更简单的更新方法，而且不必每次从树根开始更新，但是会有降低代码可读性的代价，我们在这次编码中仅呈现可读性高，逻辑清晰的代码。

刘子欣：测试成功与否，在于是否能涵盖所有的情况。我们引入了倾向性测试的概念，对于四种旋转操作，我们都选择了最合适的数据来进行检验。其中包括同一种操作的不断进行（测试1），多种操作的复合（测试3），以及对已经存在的数据进行查找时的处理（测试4,5,）。我们在做LR和RL操作复合的时候，就发现了resetH函数在初次定义上有严重的不足，而这种不足，在测试1中是完完全全体现不出来，可见测试的广度对一个程序正确性的保证。

舒星源：我很欣赏我们小组的严谨态度，我们不仅仅做编码，我们还要做严密的测试和尽可能完美的文档，我们的程序也许会有瑕疵，但是我们在确保正确性、可读性的前提下，不断修缮，不断讨论，最终呈现的是高度集中的思考过程和对问题的清晰认识，这次写文档甚至觉得有点吃亏，因为相较于编码和测试，我学到的知识量还有待增加！

1. **分工与签名**

编码：史文翰

文档：舒星源

测试：刘子欣