附件1：实验报告中的诚信设计

数据结构与算法导论

实验报告

实验题目：\_\_\_\_\_\_\_\_\_\_\_使用二叉链表实现一个平衡二叉树\_\_\_\_\_\_\_\_\_\_\_\_\_

姓 名：\_\_\_\_\_\_\_\_\_\_耿翊中\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学 号：\_\_\_2021213382\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

日 期：\_\_\_2022.6.18\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

自我评分：\_\_\_\_\_\_\_\_\_【 A+ 】\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

自我评分说明：A+，A，B+，B，B-，C，D，分别对应分数95、90、85、80、75、70、60

诚信声明

本人郑重承诺：本实验程序和实验报告均是本人独立学习和工作所获得的成果。尽我所知，实验报告中除特别标注的地方外，不包含其他同学已经发表或撰写过的成果；实验程序中对代码工作的任何帮助者所作的贡献均做了明确的说明，并表达了谢意。

如有抄袭，本人原因承担因此而造成的任何后果。

特此声明。

签名：\_\_\_\_\_\_\_\_耿翊中\_\_\_\_\_\_\_\_

日期：\_\_2022.6.18\_\_\_\_\_\_\_\_\_\_

程序引用说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 引用项 | 来源 | 代码引用行数 |
| 1 | 删除函数 | 课本 | 15 |
| 2 | 树的类的声明 | 课本 | 18 |
| 3 | 结点的类的声明 | 课本 | 7 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| 小计 | | | 40 |

总代码行数\_\_\_\_\_\_\_\_\_\_\_\_40\_\_\_\_\_\_\_\_; 引用占比\_\_\_\_\_\_10%\_\_\_\_\_\_\_\_\_\_

1、实验简介

根据平衡二叉树的抽象数据类型的定义，使用二叉链表实现一个平衡二叉树。

二叉树的基本功能：

1、平衡二叉树的建立

2、平衡二叉树的查找

3、平衡二叉树的插入

4、平衡二叉树的删除

5、平衡二叉树的销毁

6、其他：自定义操作

编写测试main()函数测试平衡二叉树的正确性。

2、程序框架

BST(T r[], int n); //构造函数

//~BST(); //析构函数

void destroy(); //删除函数

void Release(BiNode<T>\*root);//析构辅助函数

bool Search(BiNode<T>\*R, T key); //查找关键字key

BiNode<T> \*insertBiNode(BiNode<T> \*&root, T val); //插入结点

void Delete(BiNode<T>\*&R); //删除结点

bool DeleteBST(BiNode<T>\*&R, T key); //删除关键字key

int height(BiNode<T>\* root); //判断树的高度

bool isBalanced(BiNode<T>\* root); //判断树是否为二叉平衡树

int diff(BiNode<T> \*root); //计算平衡因子

BiNode<T> \*balance(BiNode<T> \*root); //平衡操作

void inorder(BiNode<T>\* root, vector<int>&res);

vector<int> inorderTraversal(BiNode<T>\* root); //中序遍历函数

vector<vector<T>> levelOrder(BiNode<T>\* root); //层序遍历函数

//旋转函数集

BiNode<T> \*LL\_Rotation(BiNode<T> \*root);

BiNode<T> \*LR\_Rotation(BiNode<T> \*root);

BiNode<T> \*RL\_Rotation(BiNode<T> \*root);

BiNode<T> \*RR\_Rotation(BiNode<T> \*root);

3、关键代码实现

3.1 BiNode<T> \*BST<T>::balance(BiNode<T> \*root)

//平衡操作

template <class T>

BiNode<T> \*BST<T>::balance(BiNode<T> \*root)

{

int dis = diff(root);//计算结点的平衡因子

if (dis > 1) {//左

if (diff(root->lch) > 0)

return LL\_Rotation(root);

else

return LR\_Rotation(root);

}

else if (dis < -1) {//右

if (diff(root->rch) < 0)

return RR\_Rotation(root);

else

return RL\_Rotation(root);

}

return root;//无需转换时记得返回root

}

通过判断平衡因子，来对每一个插入结点后形成的二叉搜索树进行左旋或右旋的操作，使之成为平衡树

3.2 vector<vector<T>> BST<T>::levelOrder(BiNode<T>\* root)

//层序遍历

template<class T>

vector<vector<T>> BST<T>::levelOrder(BiNode<T>\* root)

{

vector <vector <T>> ret;

if (!root)

{

return ret;

}

queue<BiNode<T>\*>q;

q.push(root);

while (!q.empty())

{

int currentLevelSize = q.size();

ret.push\_back(vector <T>());

for (int i = 1; i <= currentLevelSize; ++i)

{

auto node = q.front(); q.pop();

ret.back().push\_back(node->val);

if (node->lch) q.push(node->lch);

if (node->rch) q.push(node->rch);

}

}

return ret;

}

通过STL中的vector实现层序遍历的功能，层序遍历的实现可以清晰展示树的结构

4、不足

不能直观描绘出树的结构，理想中可以通过中序遍历辅助层序遍历直接展示树的脉络，但没能实现

5、心得体会

二叉平衡树是在搜索上的升级，通过巧妙地改进插入算法简化了搜索过程用时，其思路相当精巧，在对树的不断了解中也逐渐会使用递归进行求解问题，曾经最不能理解的算法似乎可以变成我最仰仗的工具。