数据结构

第五次上机作业实验文档

学号2017211123

班级 2017211301

序号 17

姓名 褚逸豪

## **问题描述**

输入一组关键字序列，并以此顺序建立一棵平衡二叉树（提示：为简化运算，可采用含有左、右子树高度和指向父母的指针的三叉链表表示），并在建树过程中用逆中序法输出每次插入新结点后的平衡二叉树形状。

## 算法思路

插入节点后向上回溯，如果平衡因子绝对值达到2则进行相应的旋转操作，使当前子树平衡，如此往复，直到回溯到根节点

## 算法描述

1. 查询/插入键值key
   1. 如果树为空，则新建节点，键值为key，将此节点设为树的根节点，退出插入节点过程
   2. 如果树不为空，向从根向下迭代，
      1. 如果键值大于当前节点，且当前节点不存在右儿子，则新建键值为key的节点作为当前节点的右儿子。若存在右儿子，则设当前节点为右儿子，再次从步骤b执行
      2. 如果键值小于当前节点，且当前节点不存在左儿子，则新建键值为key的节点作为当前节点的左儿子。若存在左儿子，则设当前节点为左儿子，再次从步骤b执行
      3. 如果键值等于当前节点，则找到键值，推出整个查询/插入过程
   3. 从当前节点向父亲迭代
      1. 判断当前节点的平衡因子，如果绝对值等于2，则进行avl旋转调整，调整后退出插入过程
      2. 令当前节点为其父结点，再次进行步骤c直到达到根节点

## 源程序及驱动程序

编译命令：g++ <文件名.cpp>

#include <cstdio>

#include <iostream>

#include <cstring>

using namespace std;

namespace avl { // 可以重构为class，以编在一个程序中使用多棵AVL树

const int MXN = 100007;

struct node { // 节点结构

node \*p, \*c[2];

int sz, mxdep; // 子树大小，最大深度

int key; // 键值

} buffer[MXN], \*root, \*cur, \*nil, \_nil; // root表示根节点，nil为一个空节点

bool isrson(node \*x) { // 右儿子判定函数，用于生成标识d与dd

return x->p->c[1] == x;

}

node \*newnode(node \*p, int key) { // 节点分配，p为父结点指针

cur->sz = cur->mxdep = 1;

cur->key = key;

cur->p = p;

cur->c[0] = cur->c[1] = nil;

return cur++;

}

void update(node \*x) { // 更新上传节点信息

x->mxdep = max(x->c[0]->mxdep, x->c[1]->mxdep) + 1;

x->sz = x->c[0]->sz + x->c[1]->sz + 1;

}

void init() { // 在使用前必须进行初始化

nil = &\_nil; // nil节点初始化

nil->c[0] = nil->c[1] = nil->p = nil;

nil->sz = nil->mxdep = nil->key = 0;

cur = buffer;

}

node\* access(int key) { // 查询[插入]键值为key的节点

if (root == NULL) { // 如果整棵树为空，则分配一个节点键值为key，作为树根

root = newnode(nil, key);

return root;

}

node \*p = root, \*rtn;

int dir = key > p->key;

while (p->key != key) { // 找到合适的位置

if (p->c[dir] == nil) {

p->c[dir] = newnode(p, key);

rtn = p->c[dir];

} else {

p = p->c[dir]; // 继续向下找

dir = key > p->key;

}

}

if (p->key == key) return p; // 如果已经有键值为key的元素，则返回该节点键值

node \*lp = p->c[dir], \*llp; // p当前节点，lp儿子，llp孙子

for (update(p); p != nil; llp = lp, lp = p, p = p->p, update(p)) { // 向上回溯

int alp = p->c[0]->mxdep - p->c[1]->mxdep;

if (alp <= -2 || alp >= 2) { // 如果平衡因子绝对值达到2

node \*pp = p->p;

int dd = isrson(p), d = isrson(lp);

if (!isrson(llp)) { // LL RR

p->c[d] = lp->c[d ^ 1];

if (p->c[d] != nil) p->c[d]->p = p;

lp->c[d ^ 1] = p;

p->p = lp;

if (pp == nil) root = lp;

else pp->c[dd] = lp;

} else { // LR RL

lp->c[d ^ 1] = llp->c[d];

p->c[d] = llp->c[d ^ 1];

if (lp->c[d ^ 1] != nil) lp->c[d ^ 1]->p = lp;

if (p->c[d] != nil) p->c[d]->p = p;

if (pp == nil) root = llp;

else pp->c[dd] = llp;

update(lp);

}

break; // 调整一次即可

}

}

return rtn; // 返回键值为key节点指针

}

void pdfs(node \*x) { // 逆中序遍历

if (x == nil) return;

putchar('(');

pdfs(x->c[1]);

printf("%d", x->key);

pdfs(x->c[0]);

putchar(')');

}

};

int main() {

avl::init();

int x;

while (~scanf("%d", &x)) { // 读一个树，查/插一个数

avl::access(x);

avl::pdfs(avl::root);

putchar('\n');

}

}

## 测试数据

### Input

78 79 123 8721 2198 389 8 92 120 321 2 02 -23 219 89743 12 -923

### Output

(78)

((79)78)

(((123)79)78)

((((8721)123)79)78)

((((8721(2198))123)79)78)

((((8721(2198(389)))123)79)78)

((((8721(2198(389)))123)79)78(8))

((((8721(2198(389)))123(92))79)78(8))

((((8721(2198(389)))123((120)92))79)78(8))

((((8721(2198(389(321))))123((120)92))79)78(8))

((((8721(2198(389(321))))123((120)92))79)78(8(2)))

((((8721(2198(389(321))))123((120)92))79)78(8(2)))

((((8721(2198(389(321))))123((120)92))79)78(8(2(-23))))

((((8721(2198(389(321(219)))))123((120)92))79)78(8(2(-23))))

(((((89743)8721(2198(389(321(219)))))123((120)92))79)78(8(2(-23))))

(((((89743)8721(2198(389(321(219)))))123((120)92))79)78((12)8(2(-23))))

(((((89743)8721(2198(389(321(219)))))123((120)92))79)78((12)8(2(-23(-923)))))

### Online Test

<https://ideone.com/3uBx8t>

## 结果分析

根据输出我们可以看到，树是高度平衡的，且符合二叉排序树的定义

## 结论

还是splay好写，AVL的旋转可以分类为两种，AVL是高度平衡的，复杂度在对数级别

## 心得体会

利用isrson函数可以降低编码量