山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201900302030 | 姓名：邵嘉明 | | 班级： 数字媒体技术 |
| 实验题目：实验五 二叉树操作 | | | |
| 实验学时：2 | | 实验日期： 2020.11.16 | |
| 实验目的：  掌握二叉树的基本概念，二叉树的存储结构使用链表。 | | | |
| 硬件环境：PC | | | |
| 软件环境：Visual Studio 2019 | | | |
| 实验步骤与内容：   1. 输入一个完全二叉树的层次遍历字符串，创建这个二叉树，输出这个二叉树的前序遍历字符串、中序遍历字符串、后序遍历字符串、结点数目、二叉树高度(上述每一个结果独立一行显示)。 2. 输入二叉树前序序列和中序序列(各元素各不相同)，创建这个二叉树，输出该二叉树的后序序列、层次遍历。 | | | |
| 结论分析与体会：  掌握了二叉树的基本概念，熟悉了链表的操作 | | | |

**本次实验主要代码：**

#include <iostream>

#include <cmath>

#include <string>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

struct treeNode {

char treeNode\_value;

treeNode\* leftChild;

treeNode\* rightChild;

treeNode(){

leftChild = rightChild = NULL;

}

};

class binaryTree {

private:

treeNode\* rootNode;

int treeHeight;

int treeLayer;

int index\_checked;

int calc\_layer(int treeHeight){

return ceil(log2(treeHeight + 1));

}

string ceBreakLeft(const string& ce, char rootNode\_value){

string tempStr;

int i = 0;

while (ce.at(i) != rootNode\_value) { i++; }

tempStr.append(ce, 0, i);

return tempStr;

}

string ceBreakRight(const string& ce, char rootNode\_value){

string tempStr;

int i = 0;

while (ce.at(i) != rootNode\_value) { i++; }

tempStr.append(ce, i + 1, ce.size() - 1 - i);

return tempStr;

}

string befBreak\_leftRight(const string& bef, const string& ce\_leftlRight){

string tempStr;

vector<int> index\_Array;

for (int i = 0; i < ce\_leftlRight.size(); i++) {

for (int j = 0; j < bef.size(); j++) {

if (bef.at(j) == ce\_leftlRight.at(i)) { index\_Array.push\_back(j); }

}

}

sort(index\_Array.begin(), index\_Array.end());

tempStr.append(bef, index\_Array[0], ce\_leftlRight.size());

return tempStr;

}

void bef\_trave(treeNode\* biTreeNode){

if (biTreeNode != NULL){

cout << biTreeNode->treeNode\_value << ((this->index\_checked < this->treeHeight - 1) ? "," : "");

this->index\_checked++;

if (this->index\_checked == this->treeHeight) { cout << endl; }

bef\_trave(biTreeNode->leftChild);

bef\_trave(biTreeNode->rightChild);

}

}

void me\_trave(treeNode\* biTreeNode){

if (biTreeNode != NULL){

me\_trave(biTreeNode->leftChild);

cout << biTreeNode->treeNode\_value << ((this->index\_checked < this->treeHeight - 1) ? "," : "");

this->index\_checked++;

if (this->index\_checked == this->treeHeight) { cout << endl; }

me\_trave(biTreeNode->rightChild);

}

}

void af\_trave(treeNode\* biTreeNode){

if (biTreeNode != NULL){

af\_trave(biTreeNode->leftChild);

af\_trave(biTreeNode->rightChild);

cout << biTreeNode->treeNode\_value << ((this->index\_checked < this->treeHeight - 1) ? "," : "");

this->index\_checked++;

if (this->index\_checked == this->treeHeight) { cout << endl; }

}

}

void tree\_generate(treeNode\* biTreeNode, const string& bef, const string& ce, int base\_Index){

if (base\_Index == 0 && bef.size() == 2){

this->rootNode[0].treeNode\_value = bef.at(0);

this->rootNode[1].treeNode\_value = bef.at(1);

if (ce.at(0) == bef.at(0)) { this->rootNode[0].rightChild = rootNode + 1; }

else { this->rootNode[0].leftChild = rootNode + 1; }

return;

}

int tempIndex = base\_Index;

string ce\_left, ce\_right, bef\_left, bef\_right;

if (bef.size() == 1 && bef == ce){

ce\_left = ce\_right = bef\_left = bef\_right = "";

}

else{

ce\_left = ceBreakLeft(ce, bef.at(0)),

ce\_right = ceBreakRight(ce, bef.at(0)),

bef\_left = befBreak\_leftRight(bef, ce\_left),

bef\_right = befBreak\_leftRight(bef, ce\_right);

}

biTreeNode->treeNode\_value = bef.at(0);

if (ce\_left.size() > 0) {biTreeNode->leftChild = ++tempIndex + this->rootNode;}

if (ce\_right.size() > 0) { biTreeNode->rightChild = ++tempIndex + this->rootNode; }

if (ce\_left.size() > 0) { tree\_generate(biTreeNode->leftChild, bef\_left, ce\_left, tempIndex); }

if (ce\_right.size() > 0) { tree\_generate(biTreeNode->rightChild, bef\_right, ce\_right, tempIndex); }

}

public:

binaryTree(const string& bef, const string& me){

this->treeHeight = bef.size();

this->rootNode = new treeNode[this->treeHeight];

tree\_generate(this->rootNode, bef, me, 0);

}

~binaryTree(){

delete[]this->rootNode;

}

binaryTree(const string& str){

int tempCount = 0;

this->treeHeight = str.size();

this->rootNode = new treeNode[this->treeHeight];

this->treeLayer = calc\_layer(this->treeHeight);

for (int i = 0; i < this->treeLayer; i++) {

for (int k = 0; k < pow(2, i) && tempCount < this->treeHeight; k++) {

rootNode[k + (int)pow(2, i) - 1].treeNode\_value = str.at(k + (int)pow(2, i) - 1);

if (2 \* (k + pow(2, i) - 1) + 1 < this->treeHeight) { rootNode[k + (int)pow(2, i) - 1].leftChild = rootNode + 2 \* (k + (int)pow(2, i) - 1) + 1; }

if (2 \* (k + pow(2, i) - 1) + 2 < this->treeHeight) { rootNode[k + (int)pow(2, i) - 1].rightChild = rootNode + 2 \* (k + (int)pow(2, i) - 1) + 2; }

tempCount++;

}

}

}

void bef\_trave(){

this->index\_checked = 0;

bef\_trave(this->rootNode);

}

void me\_trave(){

this->index\_checked = 0;

me\_trave(this->rootNode);

}

void af\_trave(){

this->index\_checked = 0;

af\_trave(this->rootNode);

}

void level\_trave(){

for (int i = 0; i < this->treeHeight; i++) cout << this->rootNode[i].treeNode\_value << ((i == this->treeHeight - 1) ? "" : ",");

cout << endl;

}

void print\_heightNum(){

cout << this->treeHeight << endl;

}

void print\_layerNum(){

cout << this->treeLayer << endl;

}

};

int main(int argc, char\* argv[]){

string inputStr, bef\_trave, me\_trave;

cout << "Input1" << endl;

cin >> inputStr;

binaryTree biTree\_01(inputStr);

cout << "Output1" << endl;

biTree\_01.bef\_trave();

biTree\_01.me\_trave();

biTree\_01.af\_trave();

biTree\_01.print\_heightNum();

biTree\_01.print\_layerNum();

cout << "Input2" << endl;

cin >> bef\_trave >> me\_trave;

binaryTree biTree\_02(bef\_trave, me\_trave);

cout << "Output2" << endl;

biTree\_02.af\_trave();

biTree\_02.level\_trave();

cout << "End";

return 0;

}