2022212153 陈祥烨 第七章作业 计科22-2 班

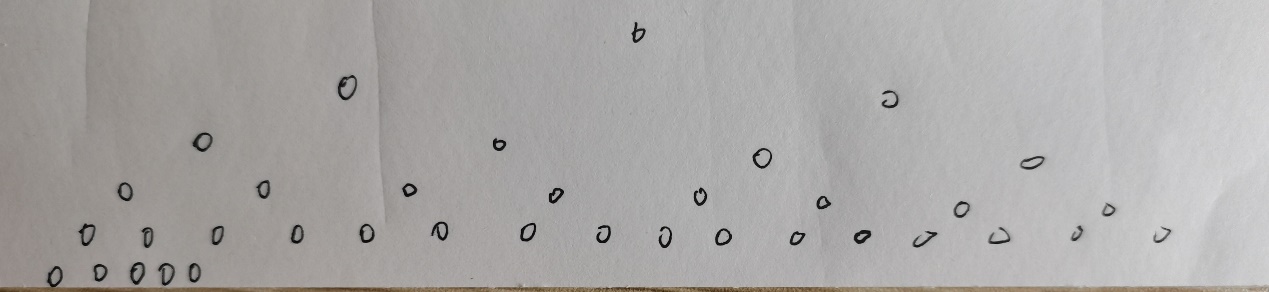
**第1题**

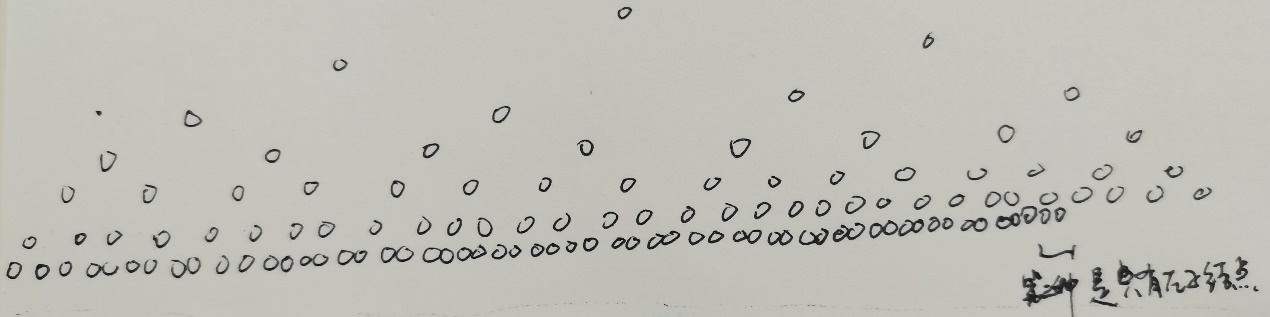
**题目：**

如果已知完全二叉树的第6层有5个叶子，试画出所有满足这一条件的完全二叉树，并指出结点数目最多的那棵完全二叉树的叶子结点数目。

**答案：**

**总共三种**

****



第三种是之后左子结点。

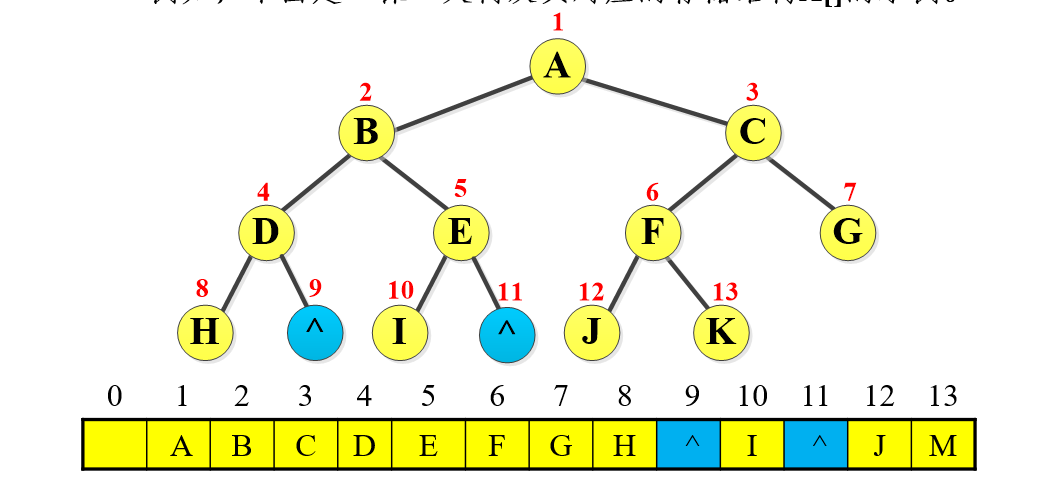
叶子结点数目：60个

**第2题**

题目：设计算法将二叉链表存储的二叉树转换为顺序存储形式。

存储在 A[]中，并将对应空结点的元素的值设置为“^”。

例如，下面是一棵二叉树及其对应的存储结构A[]的示例。

****

**思路：**

利用二叉树结点与其左右子结点之间的编号关系，得到所有结点在顺序表中的位置，并赋值

具体实现需要做两件事：

1. 得到最大序列 tree.Get\_MaxIndex(maxindex);

2. 将二叉树转换为顺序表 tree.TurnToArray(Array);

都采取左右子结点递归的方式进行探索

**代码：**

**Test.cpp**

#include"Tree.h"

void test()

{

Tree tree("A(B(D(H,),E(I,)),C(F(J,K),G)#");

cout << "先序遍历：";

tree.Preorder();

cout << endl;

char Array[100];

for (int i = 0; i < 100; i++)

Array[i] = '^';

int maxindex = 1;

tree.Get\_MaxIndex(maxindex);

tree.TurnToArray(Array);

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

{

cout << Array[i] << " ";

}

cout << endl;

}

int main()

{

test();

return 0;

}

**Tree.h**

#pragma once

#ifndef TREE\_H

#define TREE\_H

#include<iostream>

#include"Stack.hpp"

#include"Linked List.h"

using namespace std;

//---------------------------Tree类----------------------------

struct TreeNode

{

char data;

TreeNode\* lchild, \* rchild;

};

class Tree

{

public:

Tree(); //默认构造函数

Tree(const char\* str); //广义表构造函数接口

Tree(Tree& tree); //拷贝构造函数

~Tree(); //析构函数

TreeNode\* Get\_root() { return root; } //获得根结点指针

int Get\_count()const { return count; } //获得结点数

void Preorder() const; //先序遍历输出接口

void Get\_MaxIndex(int& max); //获得二叉树转顺序表最大序号接口

void TurnToArray(char array[]); //二叉树转顺序表接口

private:

void CreateTree(TreeNode\*& node, const char\* str); //广义表构造函数实现

void Preorder(TreeNode\* node) const; //先序遍历输出实现

void DestroyTree(TreeNode\* node); //删除结点实现

TreeNode\* Copy(TreeNode\* node); //复制结点实现

void Get\_MaxIndex(TreeNode\* node, int& max, int cur); //获得二叉树转顺序表最大序号实现

void TurnToArray(TreeNode\* node, char array[],int index); //二叉树转顺序表实现

private:

TreeNode\* root; //根节点包含指针域

int count;

};

#endif // !TREE\_H

**Tree.cpp**

#include"Tree.h"

//---------------------------Tree类实现----------------------------

Tree::Tree()

{

TreeNode\* node = new TreeNode;

node->lchild = NULL;

node->rchild = NULL;

root = node;

count = 0;

}

Tree::Tree(const char\* str)

{

CreateTree(root, str);

}

enum Child { LEFT, RIGHT };

void Tree::CreateTree(TreeNode\*& node, const char\* str)

{

// ( 构造左子结点

// ）结束该结点构造

// ，构造右子结点

// # 结束构造

// 其他则更新数据

Stack<TreeNode\*> stack;

node = NULL;

TreeNode\* cur = NULL; //当前结点

TreeNode\* top = NULL; //栈顶结点

Child child = LEFT; //记录是左子结点还是右子结点

while (\*str != '#')

{

if (\*str == '(')

{

stack.push(cur);

child = LEFT;

}

else if (\*str == ')')

{

stack.pop();

}

else if (\*str == ',')

{

child = RIGHT;

}

else

{

cur = new TreeNode;

cur->data = \*str;

cur->lchild = NULL;

cur->rchild = NULL;

count++;

{

if (node == NULL)

node = cur;

else if (child == LEFT)

{

stack.get\_top(top);

top->lchild = cur;

}

else if (child == RIGHT)

{

stack.get\_top(top);

top->rchild = cur;

}

}

}

str++;

}

}

void Tree::Preorder() const

{

if (count == 0)return;

Preorder(root);

}

void Tree::Preorder(TreeNode\* node) const

{

if (node == NULL) { return; }

cout << node->data;

Preorder(node->lchild);

Preorder(node->rchild);

}

Tree::Tree(Tree& tree)

{

count = tree.Get\_count();

Copy(tree.Get\_root());

}

TreeNode\* Tree::Copy(TreeNode\* node)

{

if (node == NULL)

{

return NULL;

}

else

{

TreeNode\* temp = new TreeNode;

temp->data = node->data;

temp->lchild = Copy(node->lchild);

temp->rchild = Copy(node->rchild);

return temp;

}

}

Tree::~Tree()

{

DestroyTree(root);

root = NULL;

count = 0;

}

void Tree::DestroyTree(TreeNode\* node)

{

if (node == NULL)return;

DestroyTree(node->lchild);

DestroyTree(node->rchild);

delete node;

}

void Tree::Get\_MaxIndex(int& max)

{

max = 1;

Get\_MaxIndex(root, max,1);

}

void Tree::TurnToArray(char array[])

{

if (root == NULL) { return; }

TurnToArray(root,array,1);

}

void Tree::Get\_MaxIndex(TreeNode\* node, int& max, int cur)

{

if (node == NULL) { return; }

if (cur > max) { max = cur; }

Get\_MaxIndex(node->lchild, max, cur \* 2);

Get\_MaxIndex(node->rchild, max, cur \* 2 + 1);

}

void Tree::TurnToArray(TreeNode\* node, char array[], int index)

{

if (node == NULL) { return; }

array[index] = node->data;

TurnToArray(node->lchild, array, 2 \* index);

TurnToArray(node->rchild, array, 2 \* index + 1);

}

**Tree.Stack**

#pragma once

#include<iostream>

using namespace std;

template<class T>

struct node

{

T data;

node<T>\* next;

};

template<class T>

class Stack

{

public:

Stack(); //默认构造函数

~Stack(); //析构函数

bool empty(); //判断栈空

bool get\_top(T& x); //取栈顶元素

bool push(const T x); //入栈

bool pop(); //出栈

private:

int count; //总数

node<T>\* top; //栈顶指针

};

template<class T>

Stack<T>::Stack()

{

count = 0;

top = NULL;

}

template<class T>

bool Stack<T>::empty()

{

return count == 0; //return top == NULL

}

template<class T>

bool Stack<T>::get\_top(T& x)

{

if (empty())

return false;

x = top->data;

return true;

}

template<class T>

bool Stack<T>::push(const T x)

{

node<T>\* s = new node<T>;

s->data = x;

s->next = top;

top = s;

count++;

return true;

}

template<class T>

bool Stack<T>::pop()

{

if (empty())

return false;

node<T>\* temp = top;

top = top->next;

delete temp;

count--;

return true;

}

template<class T>

Stack<T>::~Stack()

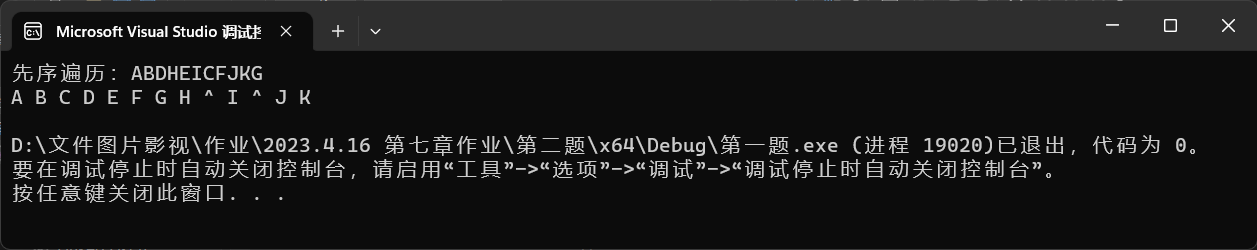
{

while (!empty())

pop();

}

**测试：**



**第3题**

**题目：**

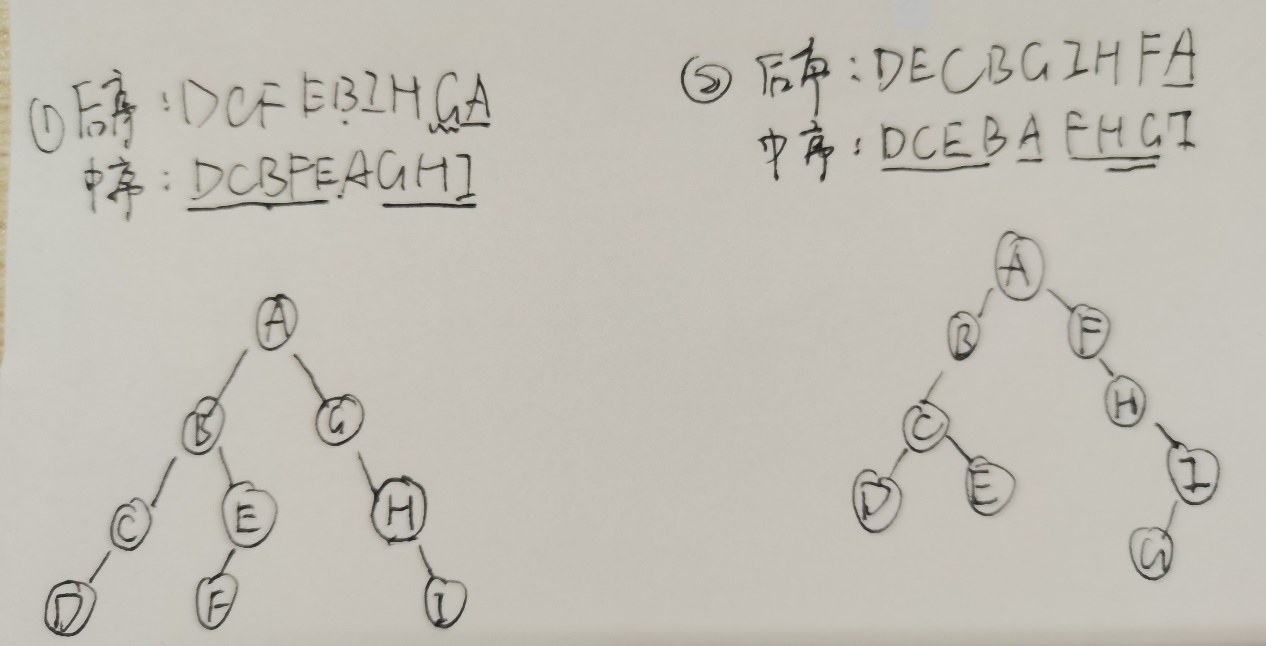
9. 并分别由下面的两个序列构造出相应的二叉树：

①后序：DCFEBIHGA ②后序：DECBGIHFA

中序：DCBFEAGHI 中序：DCEBAFHGI

**答案：**

中序可以分清左子树右子树，后序从后往前看可以群定子树的根



**第4题**

**题目：**

5. 设计算法求树/森林中所有的父子对。

**思路：**

运用递归的思想，主要的难点是森林的二叉链表是左右子树“地位不平等”的，所以处理起来左右子树会不一样。

**代码：**

void ParentAndChild(csNode\* t)

{

csNode\* p;

if (t)

{

if (t->firstChild)

{

p = t->firstChild;

printf("(%c,%c)", t->data, p->data);

ParentAndChild(p);

p = p->nextSibling;

while (p)

{

printf("(%c,%c)", t->data, p->data);

p = p->nextSibling;

}

}

if (t->nextSibling)

{

ParentAndChild(t->nextSibling);

}

}

}

**第5题**

**题目：**

3. 设计一个小系统，通过扫描一个文件获得文件中相关字符的权重，以此实现对文件的压缩，并同时构建出相应的解压缩功能，以还原所压缩文件，以便判断所实现功能的正确性。（5\*）

**代码：**

#include<iostream>

#include <fstream>

#include<map>

#include<queue>

#include <string>

using namespace std;

struct HuffmanNode {

HuffmanNode\* lChild = nullptr;

HuffmanNode\* rChild = nullptr;

char Data = '#';

int Weight;

HuffmanNode(char Data, int Weight) {

this->Data = Data;

this->Weight = Weight;

}

HuffmanNode() {};

};

struct HNodeCompare

{

bool operator() (const HuffmanNode\* a, const HuffmanNode\* b)

{

return a->Weight > b->Weight;

}

};

class HuffmanTree

{

private:

HuffmanNode\* root;

map <char, string> HuffmanMap;

map <char, int> OriginMap;

void GiveWeight(char str);

void CreateHuffmanTree();

void ReadFile();

void DisplayHuffmanTree(HuffmanNode\* Temp, string s);

void Pre(HuffmanNode\* p);

public:

HuffmanTree()

{

root = new HuffmanNode();

}

void ZIP();

void UNZIP();

};

void HuffmanTree::CreateHuffmanTree()

{

priority\_queue<HuffmanNode\*, vector<HuffmanNode\*>, HNodeCompare> HuffmanQueue;

map<char, int>::iterator it = OriginMap.begin();

while (it != OriginMap.end()) {

char ch = it->first;

int frequency = it->second;

HuffmanNode\* temp = new HuffmanNode(ch, frequency);

HuffmanQueue.push(temp);

it++;

}

if (!HuffmanQueue.empty())

if (HuffmanQueue.size() == 1)

root->lChild = HuffmanQueue.top();

else if (HuffmanQueue.size() > 1)

while (HuffmanQueue.size() != 1) {

HuffmanNode\* hfNode\_01 = HuffmanQueue.top();

HuffmanQueue.pop();

HuffmanNode\* hfNode\_02 = HuffmanQueue.top();

HuffmanQueue.pop();

HuffmanNode\* SumNode = new HuffmanNode('#', hfNode\_01->Weight + hfNode\_02->Weight);

if (hfNode\_01->Weight < hfNode\_02->Weight)

{

SumNode->lChild = hfNode\_01;

SumNode->rChild = hfNode\_02;

}

else {

SumNode->lChild = hfNode\_02;

SumNode->rChild = hfNode\_01;

}

HuffmanQueue.push(SumNode);

root = SumNode;

}

cout << "哈夫曼处理后的新编码为： " << endl;

string s;

DisplayHuffmanTree(root, s);

}

void HuffmanTree::DisplayHuffmanTree(HuffmanNode\* Temp, string s) {

if (Temp != nullptr) {

if (Temp->Data != '#') {

HuffmanMap.insert(pair<char, string>(Temp->Data, s));

cout << Temp->Data << " " << s << " " << endl;

}

DisplayHuffmanTree(Temp->lChild, s.append("0"));

s.pop\_back();

DisplayHuffmanTree(Temp->rChild, s.append("1"));

}

}

void HuffmanTree::GiveWeight(char str)

{

if (OriginMap.count(str) == 0)

{

OriginMap.insert(pair<char, int>(str, 1));

}

else {

int frequency = OriginMap[str];

++frequency;

OriginMap.erase(str);

OriginMap.insert(pair<char, int>(str, frequency));

}

}

void HuffmanTree::ReadFile() {

ifstream ifs("C:\\Users\\DELL\\Desktop\\压缩文件\\待压缩的文件.txt");

char ch;

cin.unsetf(ios::skipws);

if (!ifs.is\_open())

cout << "文件打开失败" << endl;

while (ifs.get(ch))

GiveWeight(ch);

ifs.close();

}

void HuffmanTree::Pre(HuffmanNode\* p) {

if (p != NULL) {

cout << p->Weight << " ";

Pre(p->lChild);

Pre(p->rChild);

}

}

void HuffmanTree::ZIP() {

ReadFile();

CreateHuffmanTree();

ifstream ifs("C:\\Users\\DELL\\Desktop\\压缩文件\\待压缩的文件.txt");

ofstream ofs("C:\\Users\\DELL\\Desktop\\压缩文件\\压缩后的文件.txt");

if (!ifs.is\_open() || !ofs.is\_open()) {

cout << "文件打开失败";

return;

}

char ch;

cin.unsetf(ios::skipws);

while (ifs.get(ch))

ofs << HuffmanMap.find(ch)->second << endl;

ifs.close();

ofs.close();

ofstream ofsHuffmanCode("C:\\Users\\DELL\\Desktop\\压缩文件\\字符和生成的哈夫曼编码.txt");

map<char, string > ::iterator it = HuffmanMap.begin();

while (it != HuffmanMap.end()) {

ofsHuffmanCode << it->first << " " << it->second << endl;

it++;

}

}

void HuffmanTree::UNZIP() {

ifstream in("C:\\Users\\DELL\\Desktop\\压缩文件\\压缩后的文件.txt");

string line;

ofstream out("C:\\Users\\DELL\\Desktop\\压缩文件\\解压后的文件.txt");

if (in.is\_open())

{

while (getline(in, line))

{

for (map<char, string>::iterator it = HuffmanMap.begin(); it != HuffmanMap.end(); it++)

{

if (it->second == line)

out << it->first;

}

}

}

else

{

cout << "文件打开失败" << endl;

}

in.close();

out.close();

}

int main() {

HuffmanTree hfTree;

hfTree.ZIP();

hfTree.UNZIP();

}

**测试：**

**第6题**

**题目：**

6. 设计一个树和森林的小系统，包含以下功能，并可采用菜单方式来选择相应功能： （5\*）

（1）可以采用多种方式建树或森林（指定输入、读入文件等）；

（2）采用多种方式验证构建或求解结果的正确性；

（3）各遍历算法；

（4）与二叉树的相互转换与验证；

（5）其他所要求的的算法。

**思路：不是很会.**