第7次作业

## 第5章习题

5.22分别画出下图所示的森林的双亲表示形式、孩子链表表示形式和二叉链表表示形式。

图5-3 题5.22图

**A**

**B**

**C**

**D**

**E**

**G**

**H**

**F**

**I**

**J**

**K**

**L**

**N**

**M**

**O**

**P**

5.23 将图5-3中的森林转换为对应的二叉树。

5.27将下图中的二叉树转换为对应的森林。

图5-4 题5.27图

**A**

**B**

**D**

**E**

**H**

**C**

**F**

**I**

**G**

**L**

**N**

**O**

**M**

**J**

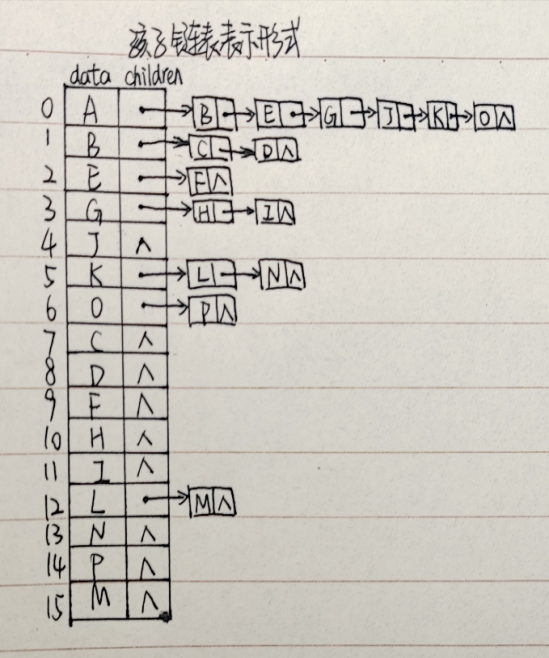
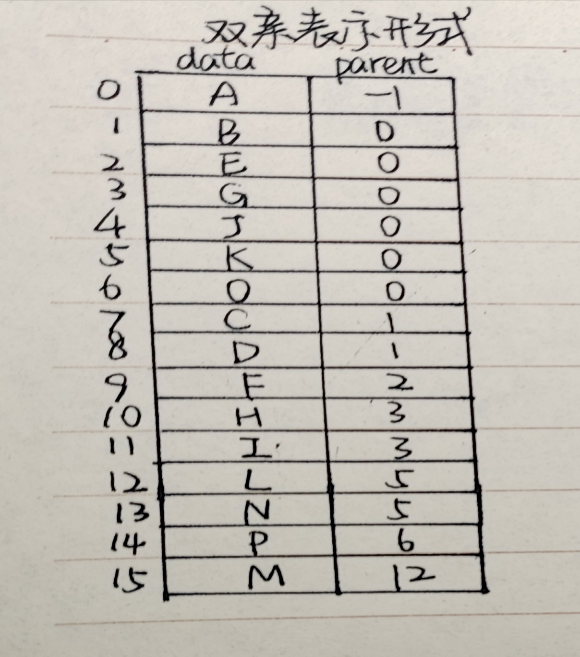
**K**

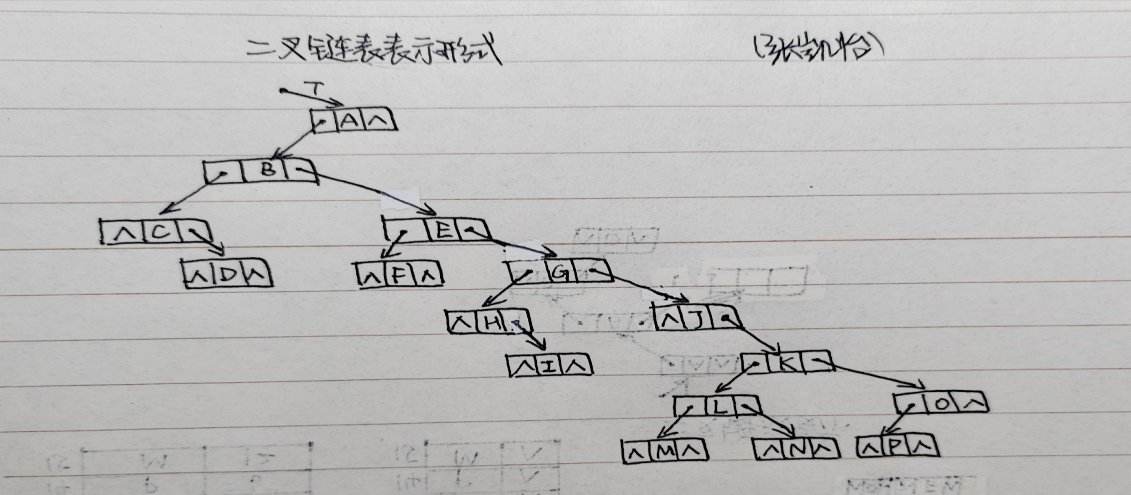
5.28设计算法按先序次序输出森林中每个结点的值及其对应的层次数。

5.29设计算法以求解森林的高度。

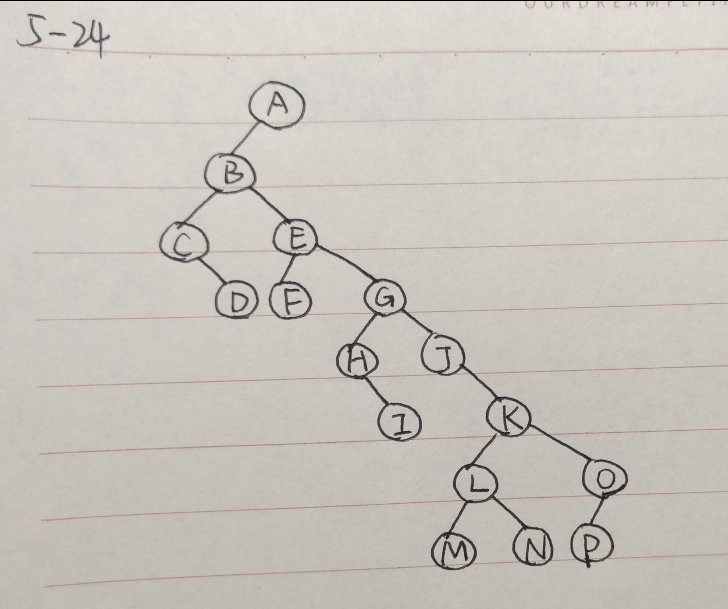
5.34已知一个文件中仅有10个不同的字符，各字符出现的个数分别为100,150,180,200,260,300,350,390,400,500。试对这些符号重新编码，以压缩文件的规模，并求出其压缩后的规模以及压缩比（压缩前后的规模比）

5.22

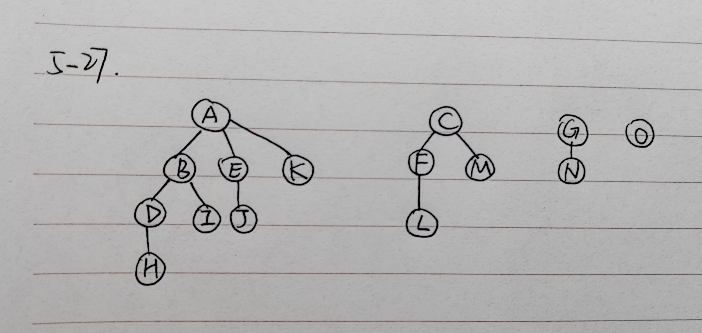




5.23



5.27



5.28

算法设计：

根据森林的先序遍历算法，只需设置一个表示层次的变量，在递归遍历t->firstson时，结点层次为原来t的层次加1；递归遍历t->nextbrother树时，结点层次与t相同。并且设置好输出形式：(结点值，层次)即可。

void preorder(tNode\* t, int s) {

if (t) {

cout << "（" << t->data << "，" << s << "） "; //格式输出

preorder(t->firstson, s + 1); //先序遍历t的子树森林

preorder(t->nextbrother, s); //先序遍历t的兄弟森林

}

}

时间分析: 时间都消耗在递归遍历上，递归遍历次数为2n-1，所以O(n)。

5.29

#include<iostream>

using namespace std;

template<class T>

class TreeNode {

//添加友元类声明，便于森林类Tree对其进行一系列访问与修改

template<class T>

friend class Tree;

private:

T data;

TreeNode<T>\* child;

TreeNode<T>\* brother;

public:

//无参数的构造函数

TreeNode() :data(T()), child(NULL), brother(NULL) {}

//有参数的构造函数

TreeNode(T value) :data(value), child(NULL), brother(NULL) {}

//析构函数

virtual~TreeNode() {

child = NULL;

brother = NULL;

}

//判断当前结点是否为叶结点

bool isLeaf() {

if (child == NULL) {

return true;

}

else {

return false;

}

}

//返回结点的值

T getValue() {

return data;

}

//返回左子结点

TreeNode<T>\* getChild() {

return child;

}

//返回右兄弟结点

TreeNode<T>\* getBrother() {

return brother;

}

//设置当前结点的值

void setValue(const T& value) {

data = value;

}

//设置左子结点

void setChild(TreeNode<T>\* pointer) {

child = pointer;

}

//设置右兄弟结点

void setBrother(TreeNode<T>\* pointer) {

brother = pointer;

}

};

template<class T>

class Tree {

private:

TreeNode<T>\* root;

public:

//构造函数

Tree() {

root = NULL;

cout << "Now start to construct the tree! " << endl;

cout << "Please enter a value or '#' to stop:";

createTree(root);

}

//析构函数

virtual~Tree() {

root = NULL;

}

//先序构造森林

void createTree(TreeNode<T>\*& node) {

T value;

cin >> value;

if (value == '#') {

node = NULL;

}

else {

node = new TreeNode<T>(value);

createTree(node->child);

createTree(node->brother);

}

}

//取根结点

TreeNode<T>\* getRoot() {

return root;

}

//先根深度优先周游树

void rootFirst(TreeNode<T>\* node) {

while (node != NULL) {

cout << node->data << " ";

rootFirst(node->child);

node = node->brother;

}

}

//后根深度优先周游树

void rootLast(TreeNode<T>\* node) {

while (node != NULL) {

rootLast(node->child);

cout << node->data << " ";

node = node->brother;

}

}

//计算深度

int countDepth(TreeNode<T>\* node) {

if (node == NULL) {

return 0;

}

int depth = 0;

int childdepth = countDepth(node->child);

int brotherdepth = countDepth(node->brother);

depth = childdepth + 1 > brotherdepth ? childdepth + 1 : brotherdepth;

return depth;

}

};

int main() {

//AB#CE#F##D###

Tree<char> test;

cout << "\n先根深度优先周游结果是:";

test.rootFirst(test.getRoot());

cout << "\n后根深度优先周游是:";

test.rootLast(test.getRoot());

cout << "\n深度是:" << test.countDepth(test.getRoot());

}

5.34

A 1111111000 B 111111101 C 11111101 D 11111100

E 111110 F 11110 G 1110 H 110

I 10 J 0

对于原始文件中的每个字符，用其在哈夫曼树上对应的编码来代替，得到一个由 0 和 1 组成的比特串。将此串保存为压缩后的文件。

原始文件大小为 100+150+180+200+260+300+350+390+400+500=2930

压缩后的文件大小为14

所以，压缩比为 209.29:1，即压缩后的文件大小约为原始文件大小的0.478%