**2020秋季学期**

**数据结构实验报告**

**实验三 树的遍历**

**班级：\_\_计实验19 \_**

**学号：\_19101130120\_**

**姓名：\_\_\_黄周杰\_\_\_**

|  |
| --- |
| **评语：**  **日期：** |

**1、**树的遍历

**需求分析：**（包括对问题的理解，解决问题的策略、方法描述）

在数据结构上，就是一个普通的二叉树。两个串都用字符串类型保存。我们需要干的就是利用先序遍历的结果，以及后续遍历的结果构造出树的原本结构，再通过后续遍历，输出结果。

**系统设计：**（包括数据结构定义、抽象出基本操作描述、主程序模块处理过程描述）

struct Node

**{**

char c**;**

Node **\***lChild**,\***rChild**;**

**};**

二叉树

**基本操作：**

Node**\*** build**(**string post**,**string mid**)**

操作结果：返回先序遍历post以及后续遍历mid对应的树的顶上的地址。

void traverse**(**Node**\*** p**)**

操作结果：返回这个二叉树的后续遍历结果。

**过程描述：**

先是输入先序遍历和中序遍历的字串，然后将其输入到build函数，得到对应的二叉树，再用traverse函数遍历二叉树，得到对应的后续遍历。

**基本操作的实现：**（对各基本操作实现的描述）

void traverse**(**Node**\*** p**)**//后续遍历

**{**

**if(**p **==** **NULL)** **return;**

traverse**(**p**->**lChild**);**

traverse**(**p**->**rChild**);**

cout**<<**p**->**c**;**

**}**

traverse

描述：p为二叉树的头节点

build

Node**\*** build**(**string post**,**string mid**)**//构造树

**{** //假如字串为空，说明没有这个子树，返回

**if(**post**.**length**()** **==** 0**)** **return** **NULL;**

//不为空时，就创建一个Node，这个Node的值附上c

char c **=** post**[**0**];**

Node**\*** p **=** **(**Node**\*)**malloc**(sizeof(**Node**));**

p**->**c **=** c**;**

//找到这个字母在中序串中的位置

int i**;**

i **=** mid**.**find**(**c**);**

//递归，去建立左右子串的树

p**->**lChild **=** build**(**post**.**substr**(**1**,**i**),**mid**.**substr**(**0**,**i**));**

p**->**rChild **=** build**(**post**.**substr**(**i**+**1**),**mid**.**substr**(**i**+**1**));**

**return** p**;**//返回这个节点的地址

**}**

描述：post为先序遍历结果，mid为后续遍历结果

**测试结果：**（输入的测试数据及运行结果、正确性、在线测试情况）

ABDEC

DBEAC

输入

DEBCA

结果

正确性：完全正确

**调试分析：**（包括调试过程中对原设计的修改，以及遇到的问题和解决的方法）

无

2、哈夫曼树

**需求分析：**（包括对问题的理解，解决问题的策略、方法描述）

先输入各个点的频率，我们的储存方式既要保证他原本的数据顺序信息，还要能建立二叉树。于是我们用一个二叉树节点对应的数字，以及信息，再用一个链表储存对应的节点的位置。存储之后，我们遍历链表，找到概率最小的两个点，然后merge出一个节点，放在链表最后，在前面的小节点为左孩子，在后面的为右节点。循环N-1次，然后遍历一遍二叉树，二叉树的头节点为链表的最后一个元素，给齐附上对应的编码。最后根据链表遍历前N个输出对应的编码。

**系统设计：**（包括数据结构定义、抽象出基本操作描述、主程序模块处理过程描述）

struct Lnode**{**

int num**;**

int used**;**

char node**[**100**];**

Lnode**\*** lchild**;**

Lnode**\*** rchild**;**

**};**

Lnode

struct Linklist**{**

Lnode**\*** p**;**

Linklist**\*** next**;**

**};**

Linklist

**基本操作：**

void input**(**Linklist**\*** **&**L**,**int **&**n**)**

操作结果：将各个概率，储存在链表L对应的p中

int getmin**(**Linklist**\*** **&**L**,**Lnode**\*** **&**p1**)**

操作结果：p1指向概率最小的节点，返回p1在链表中的位置

void merge**(**Linklist**\*** **&**L**,**Lnode**\*** **&**p1**,**Lnode**\*** **&**p2**)**

操作结果：在L的尾部添加一个节点，这个节点的左孩子为p1,右孩子为p2，值为p1+p2

void traverse\_count**(**Lnode**\*** **&**p**,**char s**[**100**],**int tail**)**

操作结果：每个节点附上对应的编码。

void traverse\_print**(**Linklist**\*** L**,**int n**)**

操作结果：输出前n个对应的编码。

void getnode**(**Linklist**\*** **&**L**,**int n**)**

操作结果：调用上两个函数，达到编码和输出的作用。

**过程描述**

先调用input函数，储存信息，然后调用n次merge函数，构造哈夫曼树。再调用getnode函数，这个函数中调用了traverse\_count和traverse\_print函数分别实现了编码以及输出的功能。

**基本操作的实现：**（对各基本操作实现的描述）

void input**(**Linklist**\*** **&**L**,**int **&**n**)**//给一个链表头指针，这个链表储存了各个数值的指针

**{**

cin**>>**n**;**

int i **=** n**;**

Lnode **\***p1 **=(**Lnode**\*)**malloc**(sizeof(**Lnode**));**

Linklist**\*** p2 **=** L**;**

**while(**i**--)**

**{**

cin**>>**p1**->**num**;** //输入节点值，完成初始化

p1**->**used **=** 0**;**

p1**->**lchild **=** **NULL;**

p1**->**rchild **=** **NULL;**

p2**->**p **=** p1**;**//p2是当前的链表指针，p3则是下一个

Linklist**\*** p3 **=** **(**Linklist**\*)**malloc**(sizeof(**Linklist**));**

p2**->**next **=** p3**;**

p2 **=** p3**;**

p1 **=(**Lnode**\*)**malloc**(sizeof(**Lnode**));**

**}**

p2**->**next **=** **NULL;**//最后一个节点的特征为p2->next->next = NULL

**}**

描述：L为链表头指针，n为输入的个数。

描述：L为链表头指针，p1用于返回最小值对应的指针。

int getmin**(**Linklist**\*** **&**L**,**Lnode**\*** **&**p1**)**

**{**

p1 **=** **NULL;**

int n **=** 0**;**

Linklist **\***fp **=** L**;**

Lnode **\***p **=** **NULL;**

**while(**fp**->**next **!=** **NULL){**

p **=** fp**->**p**;**

**if(**p**->**used**){**//如果被用过 跳过

**}**

**else**

**{**

**if(!**p1**)**//没被用过的同时，p1还没初始化，则p1指向这节点

**{**

p1 **=** p**;**

**}**

**else**//初始化了之后，比较，更新出最小的

**{**

**if(**p1**->**num **>** p**->**num**)**

**{**

p1 **=** p**;**

**}**

**}**

**}**

fp **=** fp**->**next**;**

**}**

fp **=** L**;**

**while(**fp**->**next **!=** **NULL)** //找到最小值对应的位置

**{**

n**++;**

**if(**fp**->**p **==** p1**)**

**{**

**break;**

**}**

fp**=**fp**->**next**;**

**}**

p1**->**used **=** 1**;**

**return** n**;**

**}**

描述：L为链表头指针，p1为左孩子，p2为右孩子。

void traverse\_count**(**Lnode**\*** **&**p**,**char s**[**100**],**int tail**)**//s为这个节点的编码,tail为这个字符串的空位编号.

**{**

**if(**p **==** **NULL)** **return** **;**

strcpy**(**p**->**node**,** s**);**//p->node = s;

p**->**node**[**tail**]** **=** 0**;**

s**[**tail**]** **=** '0'**;**

traverse\_count**(**p**->**lchild**,**s**,**tail**+**1**);** //给左孩子编码

s**[**tail**]** **=** '1'**;**

traverse\_count**(**p**->**rchild**,**s**,**tail**+**1**);** //给右孩子编码

**}**

void merge**(**Linklist**\*** **&**L**,**Lnode**\*** **&**p1**,**Lnode**\*** **&**p2**)**//构造二叉树

**{**

Linklist **\***fp **=(**Linklist**\*)**malloc**(sizeof(**Linklist**));**//创建一个链表的节点，然后将其赋值

Lnode **\***p **=** **(**Lnode**\*)**malloc**(sizeof(**Lnode**));**

p**->**used **=** 0**;**

p**->**num **=** p1**->**num **+** p2**->**num**;**

p**->**lchild **=** p1**;**

p**->**rchild **=** p2**;**

fp**->**p **=** p**;**

Linklist **\***Lp**=** L**;** //将新创建的链表节点插入链表末尾.

**while(**Lp**->**next**->**next **!=** **NULL)**

**{**

Lp **=** Lp**->**next**;**

**}**

fp**->**next **=** Lp**->**next**;**

Lp**->**next **=** fp**;**

**}**

描述：p为一个节点，s为这个节点的编码，tail为这个节点字符串的尾下标。

描述：L为链表头指针，n为输出的个数。

void getnode**(**Linklist**\*** **&**L**,**int n**)**//给出对应的值，然后输出

**{**

Linklist **\***fp **=** L**;**

**while(**fp**->**next**->**next **!=** **NULL)**

**{**

fp **=** fp**->**next**;**

**}**

Lnode **\***hp **=** fp**->**p**;**

char a**[**100**]** **=** "0"**;**

char b**[**100**]** **=** "1"**;**

**if(**hp**->**lchild**)**

traverse\_count**(**hp**->**lchild**,** a**,** 1**);**//

**if(**hp**->**rchild**)**

traverse\_count**(**hp**->**rchild**,** b**,** 1**);**//

traverse\_print**(**L**,**n**);**

**}**

void traverse\_print**(**Linklist**\*** L**,**int n**)**

**{**

Linklist **\***p **=** L**;**

**while(**n**--)**

**{**

cout**<<**p**->**p**->**node**<<**endl**;**

p **=** p**->**next**;**

**}**

**}**

描述：L为链表头指针，n为个数。

**测试结果：**（输入的测试数据及运行结果、正确性、在线测试情况）

完全正确

**调试分析：**（包括调试过程中对原设计的修改，以及遇到的问题和解决的方法

用string储存节点编码的时候，用 =直接赋值会出错，之后尝试了strcpy()还是不行，最后使用char 字符串的形式储存编码，解决了这个问题。