数据结构实验报告（二）

班级：计科1194 姓名：万一凡 学号： 201911111420

实验日期：2020年11月04日、2020年11月11日

实验名称：二叉树的操作及应用

实验内容：

(1) 建立一棵含有n个结点的二叉树，采用二叉链表存储；

(2) 输出前序、中序和后序遍历该二叉树的遍历结果。

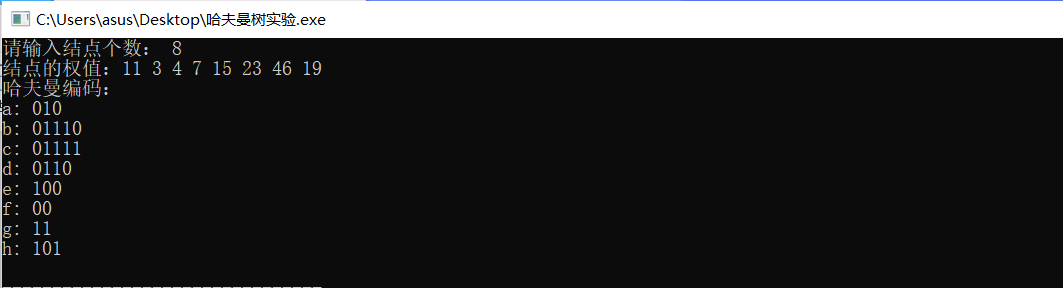
(3) 设计算法实现哈夫曼树的构建及编码

算法描述或算法思路：

1. 已知结点个数生成二叉树，并没有规定二叉树的形状，所以我生成完全二叉树，先根据结点求完全二叉树的高度。生成二叉树的时候，先生成根结点，结点的个数减一表示已经创建一个了。当结点的个数大于0时，递归处理左子树和右子树，在递归处理的时候高度依次减一，起到控制递归层次的作用，避免全在左子树上。生成的过程为前序遍历，所以需要输入树的前序结果，之后就可以生成了。
2. 前序中序和后序遍历，采用递归算法进行。
3. 哈夫曼树的建立采用树的结构体数组，数组长度至少是结点数量的2倍。数组前半部分存放结点。先寻找权值最小的两个点，两个点合并为一个点放在数组的后半部分，并修改双亲结点和左右子结点序号的值，实现哈夫曼树。
4. 编码的输出，由第一个结点开始，先递归处理双亲结点。如果是根节点，函数则返回结束，否则判断这个结点是他的双亲结点的左结点还是右结点，左结点输出“0”，右结点输出“1”。有点类似与后序遍历的递归，如此就可以输出得到哈夫曼编码了。

运行情况及分析：





源代码：

#include <stdio.h>

#include <stdlib.h>

struct tree{

char data;

tree \*right;

tree \*left;

};

int hei(int n){//由结点个数，求完全二叉树的高度

int count=1;

while(n>1){

count++;

n=n/2;

}

return count;

}

tree\* create(tree \*&p,int &n, int height){//p为根结点，n为结点的个数 ,height为高度

if(n>0){ //递归退出的条件

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

n--; //循环一次，少一代表创建了一个结点

char str;

scanf("%c",&str);

p->data=str;

p->left=NULL;

p->right=NULL;

if(height>1){ //此步通过高度判断，如果不是叶子结点就递归创建。

create(p->left,n,height-1); //高度控制递归的层级，不然一直在左子树生成。

create(p->right,n,height-1); //如果是叶子结点，则不执行，左右指向空。

} //此步是先序创建二叉树的递归过程 。

}

return p; //返回根结点

}

void qianxu(tree\*p){//前序遍历

if(p!=NULL){

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

qianxu(p->left);

qianxu(p->right);

}

}

void zhongxu(tree \*p){//中序遍历

if(p!=NULL){

zhongxu(p->left);

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

zhongxu(p->right);

}

}

void houxu(tree \*p){//后序遍历

if(p!=NULL){

houxu(p->left);

houxu(p->right);

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

}

}

int main(){

tree \*p,\*s;

s=(tree\*)malloc(sizeof(tree));

int n,h;

printf("要生成二叉树的个数：");

scanf("%d",&n);

h=hei(n);

printf("生成二叉树的前序序列:");

fflush(stdin); //清空缓存区，不然函数里的第一个字符会把回车读进去

s=create(p,n,h);

printf("\n");

printf("先序遍历结果：");

qianxu(s);

printf("\n");

printf("中序遍历结果：");

zhongxu(s);

printf("\n");

printf("后序遍历结果：");

houxu(s);

return 0;

}

**哈夫曼树：**

typedef struct

{ char data;

int weight;//结点的权值

int parent,left,right;//结点的双亲，左孩子，右孩子的下标

}hftree;

void min (hftree h[],int n,int &s1,int &s2){//返回权值最小的两个结点序号

int i,w1,w2; //w1,w2记录最小的两个权值

w1=w2=65536;

for (i=1;i<n;i++){

if(h[i].parent==0){

if(h[i].weight<w1){

w2=w1;s2=s1;

w1=h[i].weight;s1=i;

}else if(h[i].weight<w2){

w2=h[i].weight;

s2=i;

}

}

}

}

void create(hftree h[], int n) //构造哈夫曼树，n为带权值的叶子结点个数

{

/\*初始化\*/

if (n <= 1)

return;

int m = 2\*n - 1; //m为哈夫曼树中总结点的个数

for (int i = 0;i <= m;++i) //将1-m号单元的双亲，左右孩子的下标都初始化为0

{

h[i].parent = 0;

h[i].left = 0;

h[i].right = 0;

}

/\*下面开始创建哈夫曼树\*/

int s1,s2;

for (int i = n+1 ;i <= m;++i)

{ //通过n-1次的选择、删除、合并来创建哈夫曼树

min(h, i, s1, s2); //选择两个其双亲域为0且权值最小的结点

h[s1].parent = i;h[s2].parent = i; //得到新结点i,将s1\s2的双亲域由0改为i

h[i].left = s1;h[i].right= s2; //s1、s2分别作为i的左右孩子

h[i].weight = h[s1].weight + h[s2].weight; //i的权值为左右孩子的权值之和

}

}

void input(hftree h[],int n)//哈夫曼编码的输出

{

if(h[n].parent == 0)//递归终止条件

return ;

input(h,h[n].parent);//递归双亲结点

if(h[h[n].parent].left == n)//如果n的双亲的左孩子等于n，则输出0，否则右孩子输出1

printf("0");

else

printf("1");

}

int main(){

int n,i;

hftree h[100 ]; //定义实验中哈夫曼树数组的最大长度

printf("请输入结点个数： ");

scanf("%d",&n);

printf ("结点的权值：");

char str='a';

for(i=1;i<n+1;i++){

scanf("%d",&h[i].weight);

h[i].data=str; //实际data可以赋其他值，实验中默认abcd的英文字符的赋值

str++;

}

create(h,n);//构建哈夫曼树

printf("哈夫曼编码：\n");

for(i=1;i<=n;i++) //利用循环打印各个结点的哈夫曼编码

{

printf("%c: ",h[i].data);

input(h,i); //哈夫曼编码的输出

printf("\n");

}

return 0;

}

实验体会：由于二叉树的特殊性，关于二叉树树的算法应该优先考虑递归算法，使得代码看起来简洁，也便于人们的理解。哈夫曼编码是二叉树的实际应用，用顺序结构存储，便于生成。编码的输出，也可以使用递归的方法输出，使得工作量减小。