计算机与信息工程学院实验报告

••••••••••••••••••••••••••••••••• 密 ••••••••••••••••••••••••••••••••• 封 ••••••••••••••••••••••••••••••••• 线 •••••••••••••••••••••••••••••••••

姓名：\_\_于晨彤\_\_\_\_\_\_ 学号：\_2012080038\_\_\_\_\_\_\_ 专业：\_\_信息安全\_\_\_\_\_\_ 年级：\_\_2020\_\_\_\_\_\_

课程： 数据结构 主讲教师：\_\_\_\_刘成\_\_\_\_ 辅导教师：\_\_\_\_\_刘成\_\_

实验时间：\_2022\_\_年 \_5\_月 \_\_3\_日 \_下\_午\_17\_时至\_19\_时，实验地点\_\_计算机与信息工程学院608\_\_\_\_\_\_

实验题目： 二叉树的应用 赫夫曼树的构建

实验目的：

通过该实验，使学生理解二叉树的链式存储，掌握二叉树的几种遍历算法，并通过该实验使学生理解递归的含义，掌握C语言编写递归函数的方法和注意事项。

通过该实验，使学生理解赫夫曼树的概念，掌握赫夫曼树及赫夫曼编码的构造过程，体会网络发送端和接收端编码和译码过程及其工作原理。

实验环境（硬件和软件） 普通PC机 CodeBlocks

实验内容：

实现教材中算法6.4描述的二叉树创建算法，在此基础上实现二叉树的先序、后序递归遍历算法、两种非递归中序遍历、层序遍历、求二叉树的深度。

实验步骤：

在CodeBlocks中输入如下代码：

1. 二叉树：

#include<iostream>

#define TRUE 1

#define FASLE 0

#define OK 1

#define ERROR 0

#define OVERFLOW -2

#define MAXSIZE 1000

using namespace std;

typedef char TElemtype;

typedef int Status;

typedef struct BiTNode{

TElemtype data;

struct BiTNode \*lchild,\*rchild;

}BiTNode,\*BiTree;

typedef BiTree SElemType;

typedef struct{

SElemType \*base;

SElemType \*top;

int satcksize;

}SqStack;

typedef BiTree QElemType;

typedef struct QNode{

QElemType data;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct{

QueuePtr fron;

QueuePtr rear;

}LinkQueue;

//stack

//初始化栈

Status InitStack(SqStack &S){

S.base = new SElemType[MAXSIZE];

if(!S.base)

exit(OVERFLOW);

S.top = S.base;

S.satcksize=MAXSIZE;

return OK;

}

//入栈

Status Push(SqStack &S,SElemType e){

if(S.top-S.base==S.satcksize)

return ERROR;

\*S.top++=e;

return OK;

}

//出栈

Status Pop(SqStack &S,SElemType &e){

if (S.top==S.base){

return ERROR;

}

e=\*(--S.top);

return OK;

}

bool StackEmpty(SqStack S){

if(S.top==S.base)

return true;

else

return false;

}

//队列操作

//构造空队列

Status InitQueue(LinkQueue &Q){

Q.fron = Q.rear = new QNode;

Q.fron->next = NULL;

return OK;

}

//入队操作

Status EnQueue(LinkQueue &Q,QElemType e){

QueuePtr p = new QNode;

p->data = e;

p->next = NULL;

Q.rear->next = p;

Q.rear = p;

return OK;

}

//出队操作

Status DeQueue(LinkQueue &Q,QElemType &e){

if(Q.fron==Q.rear){

return ERROR;

}

QueuePtr p = Q.fron->next;

e = p->data;

Q.fron->next = p->next;

if(Q.rear==p){

Q.rear=Q.fron;

delete p;

return OK;

}

}

//判断链队列是否为空

bool QueueEmpty(LinkQueue Q){

return(Q.fron==Q.rear);

}

//创建二叉树

void CreateBiTree(BiTree &T){

char ch;

cin>>ch;

if (ch=='$'){

T = NULL;

}

else{

T = new BiTNode;

T->data = ch;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

}

//先序遍历二叉树

void PreOrderTraverse(BiTree T){

if(T){

cout<<T->data<<" ";

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

}

}

//递归中序遍历二叉树

void InOrderTraverse1(BiTree T){

if(T){

InOrderTraverse1(T->lchild);

cout<<T->data<<" " ;

InOrderTraverse1(T->rchild);

}

}

//利用栈实现中序遍历

void InOrderTraverse2(BiTree T){

SqStack S;

InitStack(S);

BiTree p = T;

BiTree q;

while(p||!StackEmpty(S)){

if(p){

Push(S,p);

p = p->lchild;

}

else{

Pop(S,q);

cout<<q->data<<" ";

p = q->rchild;

}

}

}

//后序遍历

void PostOrderTraverse(BiTree T){

if(T){

PostOrderTraverse(T->lchild);

PostOrderTraverse(T->rchild);

cout<<T->data<<" ";

}

}

//层序遍历

void LayerOrderTraverse(BiTree T){

BiTree tr = T;

LinkQueue q;

InitQueue(q);

EnQueue(q,tr);

while(!QueueEmpty(q)){

DeQueue(q,tr);

cout<<tr->data<<" ";

if(tr->lchild)

EnQueue(q,tr->lchild);

if(tr->rchild)

EnQueue(q,tr->rchild);

}

}

//深度

int DepthTree(BiTree T){

if(!T){

return 0;

}

int m = DepthTree(T->lchild);

int n = DepthTree(T->rchild);

if(m>=n){

return m+1;

}

else{

return n+1;

}

}

//判断完全二叉树

bool isWholeT(BiTree T){

if(T==NULL){

return true;

}

BiTree tr = T;

bool leaf=false;

LinkQueue q;

InitQueue(q);

EnQueue(q,tr);

while(!QueueEmpty(q)){

DeQueue(q,tr);

if((leaf&&(tr->lchild!=NULL||tr->rchild!=NULL))||(tr->lchild==NULL&&tr->rchild!=NULL))

return false;

if(tr->lchild)

EnQueue(q,tr->lchild);

if(tr->rchild)

EnQueue(q,tr->rchild);

}

return true;

}

int main(){

BiTree T;

while(1){

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\* 1.创建二叉树 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\* 2.先序遍历二叉树 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\* 3.中序遍历二叉树1 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\* 4.中序遍历二叉树2 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\* 5.后序遍历二叉树 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\* 6.层序遍历二叉树 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\* 7.求二叉树的深度 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\* 8.退出 \*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

int n;

cin>>n;

if (n==1){

CreateBiTree(T);

cout<<"创建成功"<<endl;

}

if(n==2){

PreOrderTraverse(T);

}

if(n==3){

InOrderTraverse1(T);

}

if(n==4){

InOrderTraverse2(T);

}

if(n==5){

PostOrderTraverse(T);

}

if(n==6){

LayerOrderTraverse(T);

}

if(n==7){

cout<<"二叉树的深度为:"<<DepthTree(T)<<endl;

}

if(n==8){

break;

}

cout<<endl;

}

return 0;

}

1. 哈夫曼树：

#include<iostream>

#include<cstring>

#include<string.h>

#define OK 1

#define ERROR 0

#define OVERFLOW -2

#define MAXSIZE 100

using namespace std;

typedef int Status;

typedef struct{

int weight;

int parent,lchild,rchild;

}HTNode,\*HuffmanTree;

typedef char \*\*HuffmanCode;

void FindMin(HuffmanTree &HT,int i,int &min1,int &min2){

for(int j = 1;j<i;j++){

if(HT[j].parent==-1){

if(HT[j].weight<HT[min1].weight){

min1=j;

}

}

}

HT[min1].parent = 0;

for(int j = 1;j<i;j++){

if(HT[j].parent==-1){

if(HT[j].weight<HT[min2].weight){

min2=j;

}

}

}

}

//0:a--25:z

void InputHaffmanTree(int \*frequency){

cout<<"请输入a-z和空格和出现频率:"<<endl;

for(int i=0;i<27;i++){

cin>>frequency[i];

}

}

Status InitHuffmanTree(HuffmanTree &HT,int n,int \* frequency){

int m = 2\*n-1;

HT = new HTNode[m+1];

HT[0].weight = 10000;

HT[0].lchild = -1;

HT[0].rchild = -1;

HT[0].parent = -1;

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

HT[i].weight = frequency[i-1];

HT[i].parent = -1;

HT[i].lchild = -1;

HT[i].rchild = -1;

}

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

HT[i].weight = 0;

HT[i].parent = -1;

HT[i].lchild = -1;

HT[i].rchild = -1;

}

return OK;

}

Status CreateHuffmanTree(HuffmanTree &HT,int n){

int m = 2\*n-1;

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

int min1 =0,min2=0;

FindMin(HT,i,min1,min2);

HT[min1].parent = i;

HT[min2].parent = i;

HT[i].weight = HT[min1].weight+HT[min2].weight;

HT[i].lchild = min1;

HT[i].rchild = min2;

}

return OK;

}

Status CreateHuffmanCode(HuffmanTree HT,HuffmanCode &HC,int n){

HC = new char\*[n+1];

char \*cd = new char[n];

cd[n-1]= '\0';

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

int start = n-1;

int c=i;

int f =HT[i].parent;

while(f!=-1){

--start;

if(HT[f].lchild==c){

cd[start]='0';

}else{

cd[start]='1';

}

c=f;

f=HT[f].parent;

}

HC[i] = new char[n-start];

strcpy(HC[i],&cd[start]);

}

delete cd;

return OK;

}

void OutPutHuffmanCode(HuffmanCode HC,int n){

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

cout<<(char)(i-1+'a')<<":"<<HC[i]<<endl;

}

cout<<" :"<<HC[n]<<endl;

}

string HuffmanCodeToChar(HuffmanCode HC,string code,int n){

string str="";

string curcode = "";

for(int i=0;i<code.length();i++){

if(code[i]=='%'){

return str;

}

curcode += code[i];

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

if(curcode==HC[i]){

if(i==n){

str+=" ";

}

else{

str+=(char)(i-1+'a');

}

curcode="";

}

}

}

return "输入的哈夫曼编码有误";

}

string CharToHuffmanCode(string str,HuffmanCode HC,int n){

int len = str.length();

string code="";

for(int i=0;i<len-1;i++){

if(str[i]==' '){

code+=HC[n];

}else{

code+=HC[str[i]-'a'+1];

}

}

return code;

}

int main(){

HuffmanTree HT ;

int frequency[27];

int n=27;

while(1){

HuffmanCode HC;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\* 1.输入HuffmanTree的参数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\* 2.初始化HuffmanTree参数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\* 3.创建HuffmanTree和编码表 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\* 4.输出编码表 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\* 5.输入编码，并翻译为字符 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\* 6.输入字符，并实现转码 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\* 7.退出 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "请输入你的选择：" ;

int a;

cin>>a;

switch(a){

case 1:

InputHaffmanTree(frequency);

break;

case 2:

if(InitHuffmanTree(HT,n,frequency)){

cout<<"初始化成功"<<endl;

}

else{

cout<<"初始化失败"<<endl;

}

break;

case 3:

if(CreateHuffmanTree(HT,n)){

cout<<"Huffman树创建成功"<<endl;

}

else{

cout<<"Huffman树创建失败"<<endl;

}

if(CreateHuffmanCode(HT,HC,n)){

cout<<"Huffman编码创建成功"<<endl;

}else{

cout<<"Huffman编码创建失败"<<endl;

}

break;

case 4:

OutPutHuffmanCode(HC,n);

break;

case 5:

{

string code;

cout<<"请输入哈夫曼编码:";

cin>>code;

cout<<HuffmanCodeToChar(HC,code,n)<<endl;

break;

}

case 6:{

string str;

cout<<"请输入字符串:";

char ch = cin.get();

getline(cin,str);

cout<<str<<endl;

for(int i=0;i<str.size();i++){

str[i] = tolower(str[i]);

}

cout<<CharToHuffmanCode(str,HC,n)<<endl;

break;

}

case 7:

break;

default:

cout<<"输入有误，请重新输入"<<endl;

}

}

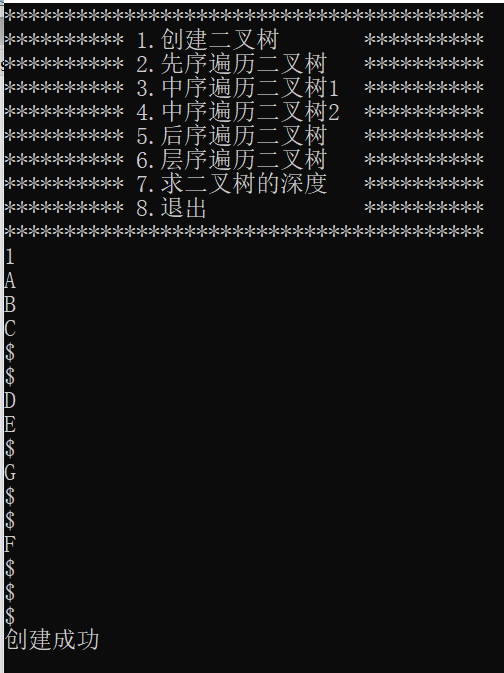
return 0;

}

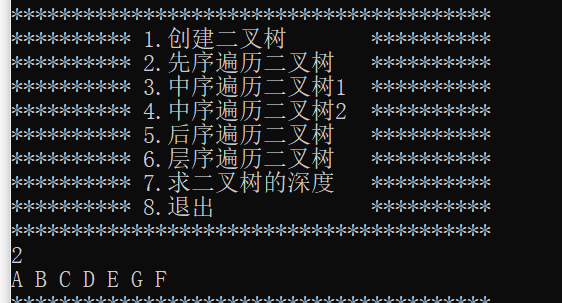
实验数据记录：

1. 二叉树

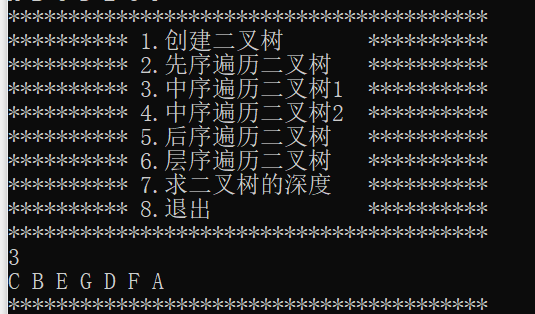
创建



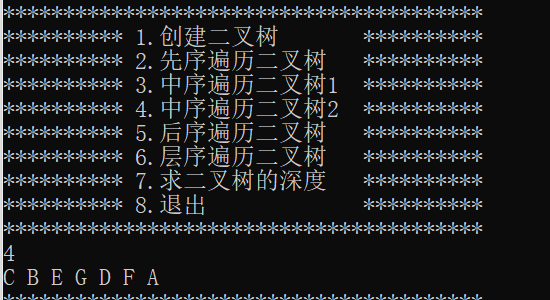
先序遍历



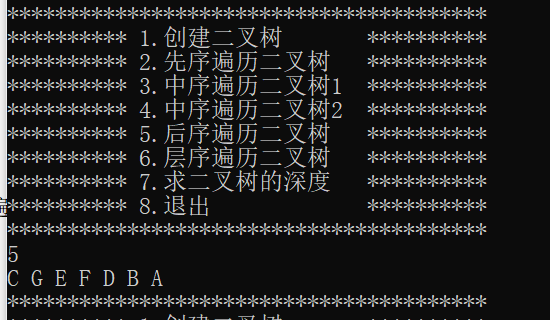
中序遍历1



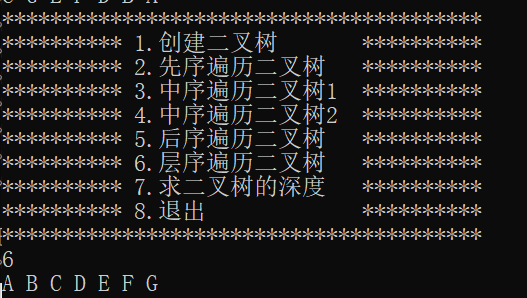
中序遍历2



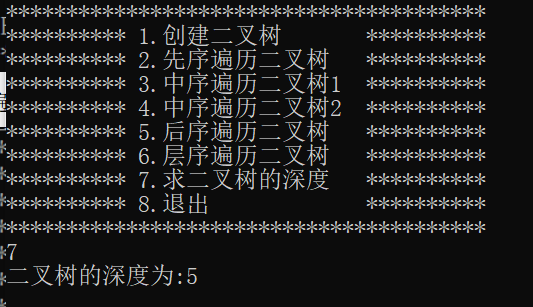
后续遍历



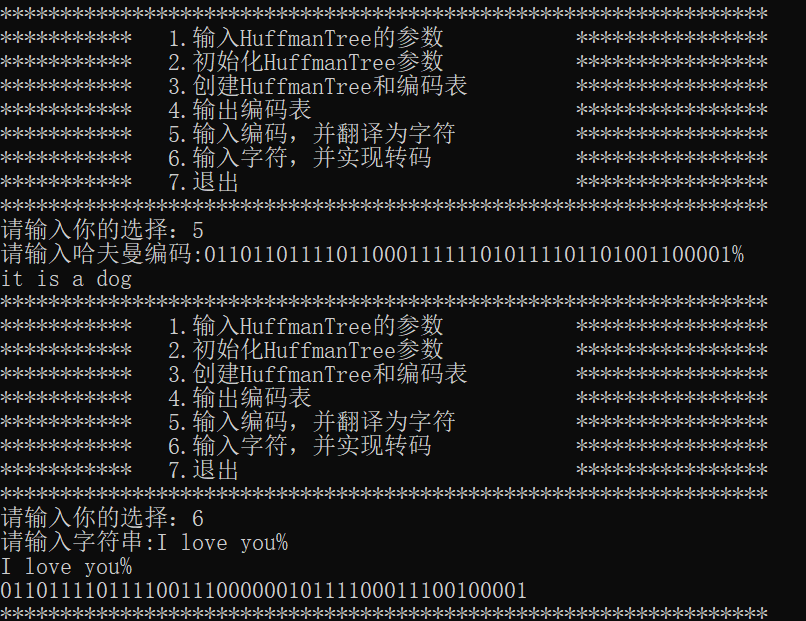
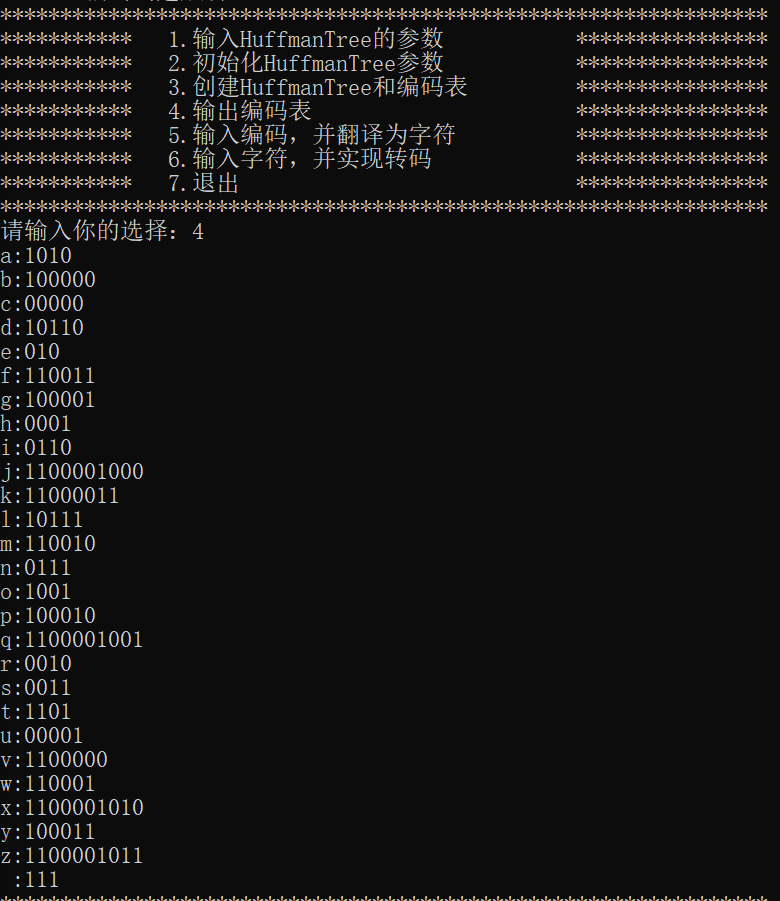
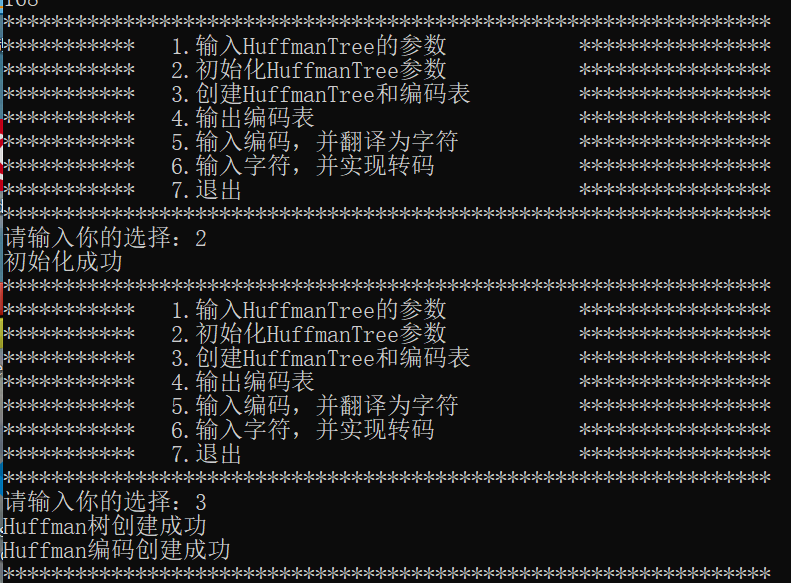
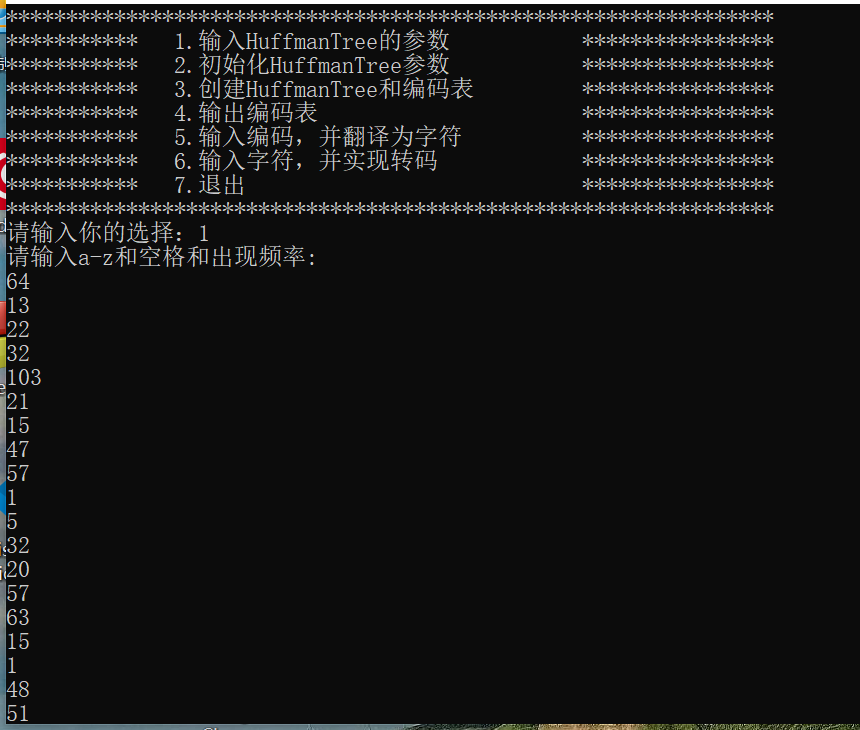
层序遍历



求二叉树深度



1. 哈夫曼树



问题讨论：

1、问题：在创构建哈夫曼树时一直出现莫名其妙的错误

解决：在初始化哈夫曼树时，忘记初始化，没有给它分配空间，导致后面出现错误

2、问题：在对字符串进行哈夫曼编码时，直接使用cin输入字符串，导致空格结束问题

解决：使用getline可以接受带空格的字符串