数据结构

2022

实 验 报 告

实验项目名称： 实现二叉树各种遍历算法及构造生成哈夫曼树

班级： 7

学号： 2021302181140

姓名： 应晓宇

指导教师： 沈志东

实验时间： 2022.5.5

实验二： 单链表实现多项式乘法

一、实验要求

（1）独立完成实验

（2）撰写实验报告

二、实验环境

硬 件： CPU： AMD RYZEM 5900HX GPU： RTX 3070

操作系统：windows 10

软件：visual studio2022

三、实验步骤及思路

（1）题目分析：要求实现各种遍历算法，那么就设计各种遍历算法，先是递归算法，较简单；非递归算法需要用到栈，层次遍历则需要用到队列。设计完各个函数后在主函数中调用。

```c
#include<stdio.h>
#include<stdlib.h>
#define maxsize 20
typedef struct tree
{
    char data;
    struct tree *lchild,*rchild;
}tree,*bittree;
typedef struct
{
    bittree ch[maxsize];
    int front,rare;
}queue,*queuepoint;
bittree create()
{
    char data;
    bittree t;
    scanf("%c", &data);
    getchar();
    if(data == '0')
    return NULL;
    else
    t=(tree *)malloc(sizeof(struct tree));
    t->data = data;
    printf("请输入%c的左子树: ",data);
    t->lchild = create();
    printf("请输入%c的右子树: ",data);
    t->rchild = create();
```

```
    return t;
}
void preg(bittree t)
{
    if(t==NULL) return;
    printf("%c", t->data);
    preg(t->lchild);
    preg(t->rchild);
}
void midg(bittree t)
{
    if(t==NULL) return;
    midg(t->lchild);
    printf("%c", t->data);
    midg(t->rchild);
}
void lastg(bittree t)
{
    if(t==NULL) return;
    lastg(t->lchild);
    lastg(t->rchild);
    printf("%c",t->data);
}
void pred(bittree t)
{
    if(t==NULL) return;
    bittree stack[maxsize]={0};
    int size=0;
```

```
    int size=0;
    stack[size++]=t;
    while(size!=0)
    {
        t=stack[--size];
        printf("%c",t->data);
        if(t->rchild)
        stack[size++]=t->rchild;
        if(t->lchild)
        stack[size++]=t->lchild;
    }
}
void midd(bittree t)
{
    bittree stack[maxsize]={0};
    int size=0;
    while(size!=0||t!=NULL)
    {
        if(t!=NULL)
        {
            stack[size++]=t;
            t=t->lchild;
        }
        else
        {
            printf("%c",stack[--size]->data);
            t=stack[size]->rchild;
        }
    }
```

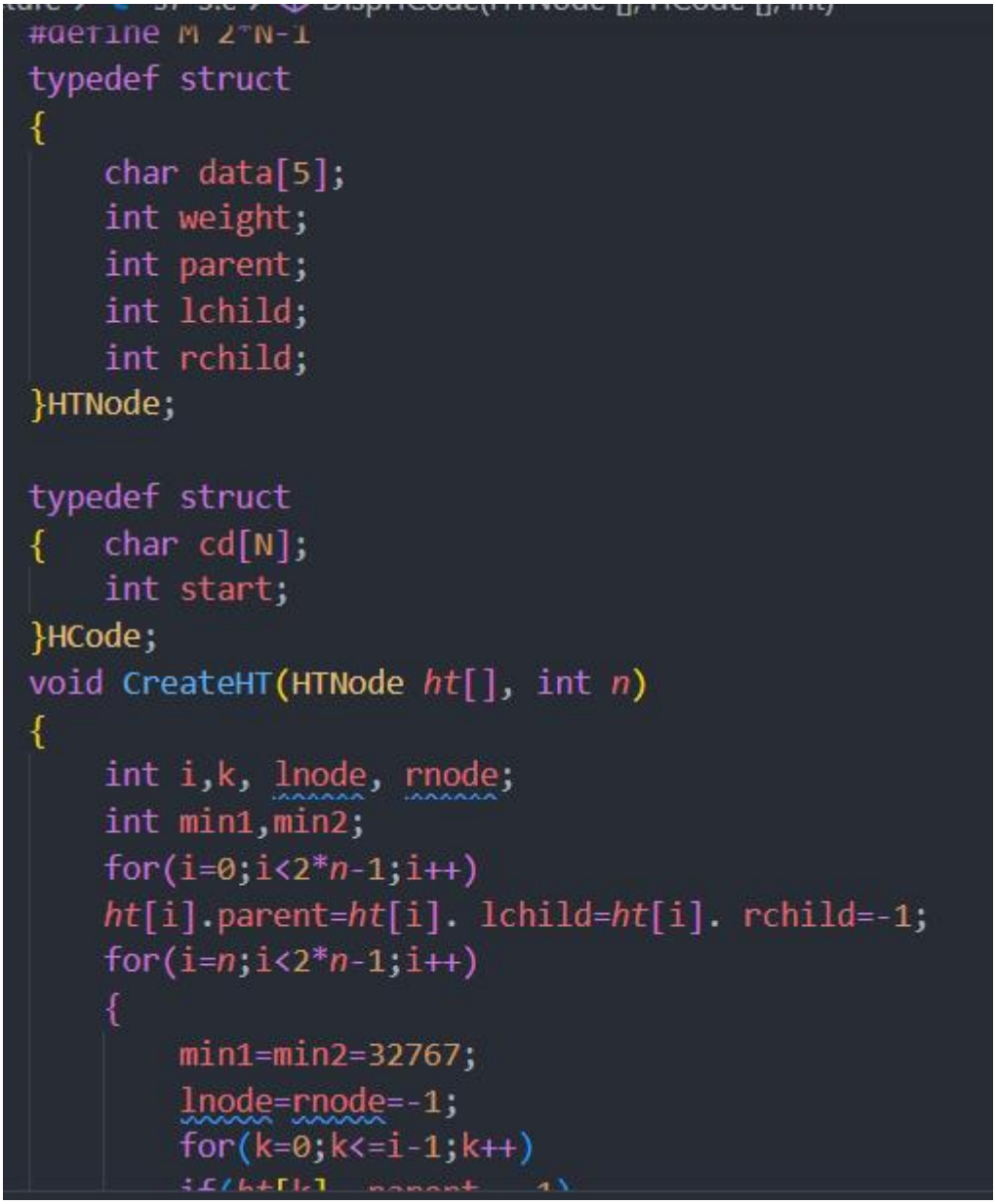
```c
void lastd(bittree t)
{
    if(t==NULL) return;
    bittree stack1[maxsize]={0};
    bittree stack2[maxsize]={0};
    int size1=0;
    int size2=0;
    stack1[size1++]=t;
    while(size1!=0)
    {
        t=stack1[--size1];
        stack2[size2++]=t;
        if(t->lchild!=NULL)
        stack1[size1++]=t->lchild;
        if(t->rchild!=NULL)
        stack1[size1++]=t->rchild;
    }
    while(size2--!=0)
    printf("%c",stack2[size2]->data);
}
queuepoint iniststack()
{
    queuepoint l=(queuepoint)malloc(sizeof(queue));
    l->front=0;
    l->rare=0;
    return l;
}
void enqueue(queuepoint l,bittree t)
```

```
void enqueue(queuepoint l,bittree t)
{
    if((l->rare+1)%maxsize==l->front)
    {
        printf("队列已满\n");
        exit(-1);
    }
    else
    {
        l->ch[l->rare]=t;
        l->rare=(l->rare+1)%maxsize;
    }
}
int isempty(queuepoint l)
{
    if(l->front==l->rare)
    return 1;
    else
    return 0;
}
bittree outstack(queuepoint l)
{
    bittree fuzhu;
    fuzhu=l->ch[l->front];
    l->front=(l->front+1)%maxsize;
    return fuzhu;
}
void cengci(bittree t)
```

```
bittree outstack(queuepoint l)
{
    bittree fuzhu;
    fuzhu=l->ch[l->front];
    l->front=(l->front+1)%maxsize;
    return fuzhu;
}
void cengci(bittree t)
{
    queuepoint s;
    s=iniststack();
    enqueue(s,t);
    while(!isempty(s))
    {
        bittree fuzhu=outstack(s);
        printf("%c",fuzhu->data);
        if(fuzhu->lchild)
        enqueue(s,fuzhu->lchild);
        if (fuzhu->rchild)
        enqueue(s,fuzhu->rchild);

    }
}
```

（2）题目分析：实验要求构造哈夫曼树和哈夫曼编码，那就设计函数实现需求

（3）



```
#define M 2*N-1
typedef struct
{
    char data[5];
    int weight;
    int parent;
    int lchild;
    int rchild;
}HTNode;

typedef struct
{   char cd[N];
    int start;
}HCode;
void CreateHT(HTNode ht[], int n)
{
    int i,k, lnode, rnode;
    int min1,min2;
    for(i=0;i<2*n-1;i++)
    ht[i].parent=ht[i]. lchild=ht[i]. rchild=-1;
    for(i=n;i<2*n-1;i++)
    {
        min1=min2=32767;
        lnode=rnode=-1;
        for(k=0;k<=i-1;k++)
```

```
        if(ht[k]. parent==-1)
        {
            if(ht[k].weight<min1)
            {
                min2=min1;rnode=lnode;
                min1=ht[k].weight;lnode=k;
            }
            else if(ht[k].weight<min2){
            min2=ht[k].weight;rnode=k;}
        }
    ht[lnode].parent=i;ht[rnode].parent=i;
    ht[i].weight=ht[lnode].weight+ht[rnode].weight;
    ht[i].lchild=lnode;ht[i].rchild=rnode;
    }
}
void CreatHCode(HTNode ht[],HCode hcd[],int n)
{
    int i,f,c;
    HCode hc;
    for(i=0;i<n;i++)
    {
        hc.start=n;
        c=i;
        f=ht[i].parent;
        while(f!=-1)
```

```
        f=ht[i].parent;
        while(f!=-1)
        {
            if(ht[f].lchild==c)
            hc.cd[hc.start--]='0';
            else
            hc.cd[hc.start--]='1';
            c=f;f=ht[f].parent;
        }
        hc.start++;
        hcd[i]=hc;
    }
}
void DispHCode(HTNode ht[],HCode hcd[],int n)
{
    int i,k;
    int sum=0,m=0,j;
    printf("输出哈夫曼编码:\n");
    for(i=0;i<n;i++)
    {
        j=0;
        printf("    %s:",ht[i].data);
        for(k=hcd[i].start;k<=n;k++)
        {
            printf("%c",hcd[i].cd[k]);
```

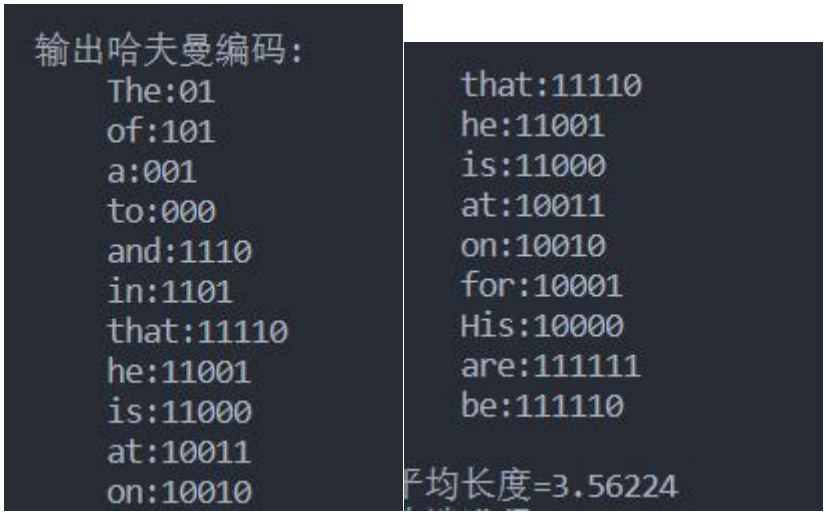
```
            j++;
        }
        m+=ht[i].weight;
        sum+=ht[i].weight*j;
        printf("\n");
    }

    printf("\n平均长度=%g\n",1.0*sum/m);
}
int main()
{
    int n=15,i;
    char *str[]={"The","of","a","to","and","in","that","he","is","at","on","for","His","are","be"};
    int fnum[]={1192,677,541,518,462,452,242,195,190,181,174,157,138,124,123};
    HTNode ht[M];
    HCode hcd[N];
    for(i=0;i<n;i++)
    {
        strcpy(ht[i].data,str[i]);
        ht[i].weight=fnum[i];
    }
    CreateHT(ht,n);
    CreatHCode(ht,hcd,n);
    DispHCode(ht,hcd,n);
    return 1;
```

四、实验结果及分析

```
请输入根结点数据:
a
请输入a的左子树: b
请输入b的左子树: d
请输入d的左子树: 0
请输入d的右子树: 0
请输入b的右子树: e
请输入e的左子树: h
请输入h的左子树: j
请输入j的左子树: 0
```

```
请输入g的左子树: 0
请输入g的右子树: i
请输入i的左子树: 0
请输入i的右子树: 0
递归先序遍历结果:
abdehjklmncfgi
非递归先序遍历结果:
abdehjklmncfgi
递归中序遍历结果:
dbjhlkmneafcgi
非递归中序遍历结果:
dbjhlkmneafcgi
```

```
递归中序遍历结果:
dbjhlkmneafcgi
非递归中序遍历结果:
dbjhlkmneafcgi
递归后序遍历结果:
djlnmkhebfigca
非递归后序遍历结果:
djlnmkhebfigca
层次遍历结果:
abcdefghijklmn
```



五、总结

实验完成，结果正确。