**实验报告：Huffman算法**

**实验题目：**

Huffman算法

**实验要求：**

1. 输入英文短文，统计字符，计算权值
2. 实现Huffman编码，写一个输出文件

**编程思路：**

1. 用一个整型数组记录各个小写字母的权值
2. 以每个小写字母建立一个节点，将节点的指针储存在Nodeline数组中
3. 以权值建立Huffman二叉树，建立二叉树的同时进行编码

**核心代码：**

1. 统计英文字符

**void alpha\_number(char \*Text,int \*alphanum){//统计字母，计算权值**

for(int i=0;Text[i]!='\0';i++){

if(!islower(Text[i])) continue;//判断是否为小写字母

alphanum[Text[i]-'a']++;

}

}

i**nt rank(BTNode \*Node){//递归算法计算节点权值**

if(Node==NULL) return 0;

return (Node->alphanum+rank(Node->LeftChild)+rank(Node->RightChild));

}

1. Huffman算法

//其中的Nodeline为一个储存各个字母节点的数组

**int Conbine(BTNode \*\*Nodeline){//合并节点，返回非空节点数**

int nodenum=0;//非空节点数

int min1=-1,min2=-1;

//寻找权值最小的两个非空节点

for(int i=0;i<ALPHANUM;i++){//寻找非空节点

if(Nodeline[i]==NULL) continue;

nodenum++;

if(min1==-1){//初始化

min1=i;

continue;

}

if(min2==-1){

if(min1<min2){

min2=i;

continue;

}

else{

min2=min1;

min1=i;

continue;

}

}

if(rank(Nodeline[i])<rank(Nodeline[min1])){

min1=i;

continue;

}

if(rank(Nodeline[i])<rank(Nodeline[min2])){

min2=i;

continue;

}

}

//合并、删除节点,给节点编码

addcode(Nodeline[min1],0); **//给节点编码**

addcode(Nodeline[min2],1);

Nodeline[min1]=initnode('\0',0,Nodeline[min1],Nodeline[min2]);

Nodeline[min2]=NULL;

nodenum--;

return nodenum;

}

**TREE Haffman(BTNode \*\*Nodeline){//哈夫曼算法，不断合并直至只有一个节点**

while(Conbine(Nodeline)>1);

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

if(Nodeline[i]!=NULL) return Nodeline[i];

}

return NULL;

}

**void addcode(BTNode \*Node,int n){//给节点编码，n=0表示左子树，n=1表示右子树**

if(Node==NULL) return;

if(n==0){

strcat(Node->Code,"0");

}

if(n==1){

strcat(Node->Code,"1");

}

addcode(Node->LeftChild,n);

addcode(Node->RightChild,n);

return;

}

**实验结果：**

**Text.txt文件内容：**

abbcccddddee## eeeffffffggggggghhhhhhhhiiiiiiiiiijjjjjjjjjjjkkkkkkkkkkkk

**Output.txt文件内容：**

a的数量为：1,哈夫曼编码为：1011001

b的数量为：2,哈夫曼编码为：101101

c的数量为：3,哈夫曼编码为：10111

d的数量为：4,哈夫曼编码为：1111

e的数量为：5,哈夫曼编码为：1110

f的数量为：6,哈夫曼编码为：1010

g的数量为：7,哈夫曼编码为：010

h的数量为：8,哈夫曼编码为：011

i的数量为：10,哈夫曼编码为：110

j的数量为：11,哈夫曼编码为：100

k的数量为：12,哈夫曼编码为：00

l的数量为：0,哈夫曼编码为：101100000000000000000

m的数量为：0,哈夫曼编码为：101100000000000000001

n的数量为：0,哈夫曼编码为：10110000000000000001

o的数量为：0,哈夫曼编码为：1011000000000000001

p的数量为：0,哈夫曼编码为：101100000000000001

q的数量为：0,哈夫曼编码为：10110000000000001

r的数量为：0,哈夫曼编码为：1011000000000001

s的数量为：0,哈夫曼编码为：101100000000001

t的数量为：0,哈夫曼编码为：10110000000001

u的数量为：0,哈夫曼编码为：1011000000001

v的数量为：0,哈夫曼编码为：101100000001

w的数量为：0,哈夫曼编码为：10110000001

x的数量为：0,哈夫曼编码为：1011000001

y的数量为：0,哈夫曼编码为：101100001

z的数量为：0,哈夫曼编码为：10110001

**源码全文：**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define MAX\_TEXT\_SIZE 2000

#define ALPHANUM 26

typedef struct BTNode{//二叉树节点

char alpha;//字母

int alphanum;//字母数量

char Code[ALPHANUM];//Huffman编码

BTNode \*LeftChild;//左子树

BTNode \*RightChild;//右子树

}BTNode;

typedef BTNode\* TREE;

void input\_text(char \*Text);//从文件读取文本，输入到字符串

void alpha\_number(char \*Text,int \*alphanum);//统计字母，计算权值

int rank(BTNode \*Node);//计算节点权值

BTNode \*initnode(char Alpha,int number,BTNode \*Left,BTNode \*Right);//初始化节点

void generate\_node(int \*alphanum,BTNode \*\*Nodeline);//建立节点数组

int Conbine(BTNode \*\*Nodeline);//合并节点，返回非空节点数

TREE Haffman(BTNode \*\*Nodeline);//哈夫曼算法，不断合并直至只有一个节点

void Output(BTNode \*\*Nodeline);//打印哈夫曼树，以文件形式输出

void addcode(BTNode \*Node,int n);//给节点编码

void Reversecode(BTNode \*Node);//编码倒序

BTNode \*Search(TREE Tree,char c);//寻找指定节点

void alphaline(TREE Tree,BTNode \*\*Nodeline);//将编码好的节点再次线性存储

BTNode \*initnode(char Alpha,int number,BTNode \*Left,BTNode \*Right);//初始化节点

int main(){

int alphanum[ALPHANUM]={0};//储存字母出现次数

BTNode \*nodeline[ALPHANUM]={NULL};//储存字母节点

char Text[MAX\_TEXT\_SIZE];//储存文本

input\_text(Text);//将文件输入至字符数组

alpha\_number(Text,alphanum);//统计字母出现次数

generate\_node(alphanum,nodeline);//生成字母节点

TREE Tree=Haffman(nodeline);//生成Huffman树

Reversecode(Tree);//编码倒序

alphaline(Tree,nodeline);//重新将编码好的字母节点排序

Output(nodeline);//输出

return 0;

}

void input\_text(char \*Text){//从文件读取文本，输入到字符串

FILE \*read;

read=fopen("Text.txt","r");

fgets(Text,MAX\_TEXT\_SIZE,read);

fclose(read);

}

void alpha\_number(char \*Text,int \*alphanum){//统计字母，计算权值

for(int i=0;Text[i]!='\0';i++){

if(!islower(Text[i])) continue;//判断是否为小写字母

alphanum[Text[i]-'a']++;

}

}

BTNode \*initnode(char Alpha,int number,BTNode \*Left,BTNode \*Right){//初始化节点

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

p->alpha=Alpha;

p->alphanum=number;

p->Code[0]='\0';

p->LeftChild=Left;

p->RightChild=Right;

return p;

}

void generate\_node(int \*alphanum,BTNode \*\*Nodeline){//建立节点数组

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

Nodeline[i]=initnode('a'+i,alphanum[i],NULL,NULL);

}

}

int Conbine(BTNode \*\*Nodeline){//合并节点，返回非空节点数

int nodenum=0;//非空节点数

int min1=-1,min2=-1;

//寻找权值最小的两个非空节点

for(int i=0;i<ALPHANUM;i++){//寻找非空节点

if(Nodeline[i]==NULL) continue;

nodenum++;

if(min1==-1){//初始化

min1=i;

continue;

}

if(min2==-1){

if(min1<min2){

min2=i;

continue;

}

else{

min2=min1;

min1=i;

continue;

}

}

if(rank(Nodeline[i])<rank(Nodeline[min1])){

min1=i;

continue;

}

if(rank(Nodeline[i])<rank(Nodeline[min2])){

min2=i;

continue;

}

}

//合并、删除节点,给节点编码

addcode(Nodeline[min1],0);

addcode(Nodeline[min2],1);

Nodeline[min1]=initnode('\0',0,Nodeline[min1],Nodeline[min2]);

Nodeline[min2]=NULL;

nodenum--;

return nodenum;

}

TREE Haffman(BTNode \*\*Nodeline){//哈夫曼算法，不断合并直至只有一个节点

while(Conbine(Nodeline)>1);

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

if(Nodeline[i]!=NULL) return Nodeline[i];

}

return NULL;

}

void addcode(BTNode \*Node,int n){//给节点编码

if(Node==NULL) return;

if(n==0){

strcat(Node->Code,"0");

}

if(n==1){

strcat(Node->Code,"1");

}

addcode(Node->LeftChild,n);

addcode(Node->RightChild,n);

return;

}

int rank(BTNode \*Node){//计算节点权值

if(Node==NULL) return 0;

return (Node->alphanum+rank(Node->LeftChild)+rank(Node->RightChild));

}

void Reversecode(BTNode \*Node){//编码倒序

if(Node==NULL) return;

Reversecode(Node->LeftChild);

Reversecode(Node->RightChild);

char string[ALPHANUM];

strcpy(string,Node->Code);

int i;

for(i=0;string[i]!='\0';i++);

i--;

int j;

for(j=0;i>=0;i--,j++){

Node->Code[j]=string[i];

}

Node->Code[j]='\0';

return;

}

void alphaline(TREE Tree,BTNode \*\*Nodeline){//将编码好的节点再次线性存储

for(int n=0;n<ALPHANUM;n++){

Nodeline[n]=Search(Tree,'a'+n);

}

}

BTNode \*Search(TREE Tree,char c){//寻找指定节点

if(Tree==NULL) return NULL;

BTNode \*p;

p=Search(Tree->LeftChild,c);

if(p!=NULL) return p;

p=Search(Tree->RightChild,c);

if(p!=NULL) return p;

if(Tree->alpha==c) return Tree;

return NULL;

}

void Output(BTNode \*\*Nodeline){//打印哈夫曼编码，以文件形式输出

FILE \*write;

write=fopen("Output.txt","w");

for(int n=0;n<ALPHANUM;n++){

fprintf(write,"%c的数量为：%d,哈夫曼编码为：%s\n",Nodeline[n]->alpha,Nodeline[n]->alphanum,Nodeline[n]->Code);

}

fclose(write);

}