**实习报告**

题目：实现哈夫曼编/译器

班级：电信1809 姓名：朱燚 学号：U201813488 完成日期：2019/4/22

1. 需要分析

（1）本演示程序中，输入值都为int型。

（2）演示程序以用户和计算机的对话方式执行，即在计算机终端上显示“提示信息”之后，由用户在键盘上输入演示程序中规定的运算命令。

（3）程序执行的命令包括：

1）需要输入报文的编码。

1. 概要设计

为了实现上述程序，应使用二叉树。

抽象数据类型二叉树的定义：

ADT BinaryTree{

数据对象D: D是具有相同特性的数据类型元素的集合

数据关系R：

若D=，则R=，称BinaryTree为空二叉树；

若DR={H}，H是如下二元关系：

1. 若D中存在唯一的称为根的数据元素root，它在关系H下无前驱；
2. 若D-{root}，则存在D-root={}，且=；
3. 若，中存在唯一的元素，<root,>H, 且存在上的关系；，则中存在唯一的元素H,且存在上的关系;H={<root,>,,};
4. ()是一棵符合定义的二叉树，称为根的左子树，()是一棵符合本定义的二叉树，称为根的右子树。

基本操作P：

InitBiTree（&T）

操作结果：构造空二叉树T。

DestroyBiTree（&T）；

初始条件：二叉树T存在。

操作结果：销毁二叉树。

CreateBiTree（&T，definition）;

初始条件：definition给出二叉树T的定义。

操作结果：按difinition构造二叉树。

本程序包含两个模块：

1. 主函数模块
2. 树模块

主函数模块调用树模块

3.详细设计

1）二叉树类型

typedef struct HtNode

{

int weight;

int parent, lchild, rchild;

int flag;

}HtNode;

typedef struct HtTree

{

HtNode ht[3000];

int root;

}phtree;

2)主程序和其他伪码算法

int main()

{

//主函数

int i, n, cnt = 1;

char c;

int w[101];

char s[101];

int pcode[100][100];

for(i = 0;i < 100;i++)

{

for(int j = 0;j < 100;j++)

{

pcode[i][j] = -1;

}

}

scanf("%d\n",&n);

while(1)

{

scanf("%c\n",&c);

if(cnt == n+1)

{

break;

}

else if(c != ' ')

{

s[cnt++] = c;

}

}

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

{

scanf("%d\n",&w[i]);

}

phtree\* pht;

pht = huffman(n,w);

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

{

traversehuffman(pht,n,pcode);

}

output(pcode,n,s);

return 0;

}

void select(phtree\* pht,int pos,int \*x1,int \*x2)

{

int m1, m2;

m1 = m2 = 10000;

for(int i = 1;i < pos;i++)

{

if(pht->ht[i].weight < m1 && pht->ht[i].parent == 0)

{

m2 = m1;

\*x2 = \*x1;

m1 = pht->ht[i].weight;

\*x1 = i;

}

else if(pht->ht[i].weight < m2 && pht->ht[i].parent == 0 && pht->ht[i].weight > m1)

{

m2 = pht->ht[i].weight;

\*x2 = i;

}

}

}

phtree \*huffman(int n,int \*w)

{

phtree\* pht;

int i, x1, x2;

pht = (phtree\* )malloc(sizeof(phtree));

pht->ht[0].flag = 2;

pht->ht[0].lchild = 0;

pht->ht[0].rchild = 0;

pht->ht[0].parent = 0;

pht->ht[0].weight = 0;

for(i = 2\*n;i < 3000;i++)

{

pht->ht[i].flag = 2;

pht->ht[i].lchild = 0;

pht->ht[0].rchild = 0;

pht->ht[0].parent = 0;

pht->ht[0].weight = 0;

}

for(i = 1;i <= 2\*n-1;i++)

{

pht->ht[i].lchild = 0;

pht->ht[i].rchild = 0;

pht->ht[i].parent = 0;

pht->ht[i].flag = 2;

if(i <= n)

{

pht->ht[i].weight = w[i];

}

else

{

pht->ht[i].weight = 0;

}

}

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

{

select(pht,n+i,&x1,&x2);

pht->ht[x1].parent = n+i;

pht->ht[x1].flag = 0;

pht->ht[x2].parent = n+i;

pht->ht[x2].flag = 1;

pht->ht[n+i].weight = pht->ht[x1].weight + pht->ht[x2].weight;

pht->ht[n+i].lchild = x1;

pht->ht[n+i].rchild = x2;

pht->root = n+i;

}

return pht;

}

void traversehuffman(phtree \*t,int n,int pcode[][100])

{

int i = 1;

HtNode node;

node = t->ht[pc];

while(node.parent != 0)

{

pcode[pc][i] = node.flag;

node = t->ht[node.parent];

i++;

}

pc++;

}

void output(int pcode[][100],int n,char \*x)

{

int i, j, k, cnt;

int code[100][100];

for(i =0;i < 100;i++)

{

for(j = 0;j < 100;j++)

{

code[i][j] = -1;

}

}

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

{

for(cnt = 1;pcode[i][cnt+1] == 1 || pcode[i][cnt+1] == 0;cnt++);

k = cnt;

for(j = 1;j<= k;j++)

{

code[i][j] = pcode[i][cnt--];

}

}

char s[100], c;

scanf("%s\n",s);

int f = strlen(s);

for(i = 0;i <= f;i++)

{

for(j = 1;j <= n;j++)

{

if(s[i] == x[j])

{

for(k = 1;code[j][k] == 0 || code[j][k] == 1;k++)

{

printf("%d",code[j][k]);

}

}

}

}

printf("\n%s\n",s);

}

1. 函数的调用关系

主函数调用phtree \*huffman，traversehuffman，output函数，phtree \*huffman调用select函数。

4.调试分析

1. 本次作业很难调试，用c语言的编译器难以实现某些算法，scanf把回车当做字符输入导致错误
2. 算法的时空分析：
3. 该算法各操作时间复杂度比较合理。
4. 通过分析，该算法的复杂度为

5.用户手册：（略）

6.测试结果：（略）

7.附录：

源程序文件名清单（略）