**二叉树的应用实验报告**

姓名：陆明奇 学号：2050283 日期：2021年11月14日

1. **涉及数据结构和相关背景**

给定N个权值作为N个叶子结点，构造一棵二叉树，若该树的带权路径长度达到最小，称这样的二叉树为最优二叉树，也称为哈夫曼树(Huffman Tree)。哈夫曼树是带权路径长度最短的树，权值较大的结点离根较近。在计算机数据处理中，哈夫曼编码使用变长编码表对源符号（如文件中的一个字母）进行编码，其中变长编码表是通过一种评估来源符号出现机率的方法得到的，出现机率高的字母使用较短的编码，反之出现机率低的则使用较长的编码，这便使编码之后的字符串的平均长度、期望值降低，从而达到无损压缩数据的目的。

**2. 实验内容**

**2.1 问题描述**

实现对ASCII字符文本进行Huffman压缩，并且能够进行解压。

**2.2 数据结构设计**

此题考察的是用栈模拟n的阶乘的递归调用过程。因此采用栈这一数据结构。

**2.4功能说明（函数、类）**

int input(char a)

{

char\_weight[a]++;

if (char\_weight[a] == 1) {//判断a是不是第一次出现，是则计数

return 1;

}

else {

return 0;

}

}

功能：输入需要进行压缩的字符

Para a: 输入的需要压缩的字符

返回值：输出的字符是不是第一次出现，即是否为一个新字符。

复杂度：O（1）

void Select(HuffmanTree a, int b, int& c, int& d)

{

int k = 1;

HuffmanTree e = a;

int num1 = 0;

while (num1 != 2){

if (e->parent == 0){

if (num1 == 0){

c = e - a;

}

else{

d = e - a;

}

num1++;

}

e++;

}

while ((e - a) <= b){

if (e->parent == 0){

if (e->weight < (a + c)->weight || e->weight < (a + d)->weight) {

if ((a + c)->weight <= (a + d)->weight){

d = e - a;

}

else{

c = d;

d = e - a;

}

}

}

e++;

}

}

功能：选择霍夫曼树左右孩子结点

Para a: 指向霍夫曼树的指针

Para b: 可选择的最大结点的编号

Para c: 选择出的其中一个weight最小的结点编号

Para d: 选择出的另外一个weight最小的结点编号

返回值：无返回值

复杂度：O(n)

void HuffmanCoding(HuffmanTree& HT, HuffmanCode& HC, node\* w, int n)

{

HuffmanTree p;

char\* cd;

int i, s1, s2, start;

unsigned int c, f;

if (n <= 1) {

return;

} // n为字符数目，m为结点数目

int m = 2 \* n - 1;

HT = (HuffmanTree)malloc((m + 1) \* sizeof(HTNode));//0号单元未用

p = HT;

p->weight = -1;

p->parent = -1;

p->lchild = -1;

p->rchild = -1;

for (p = HT + 1, i = 1; i <= n; ++i, ++p, ++w){

p->weight = w->weight;

p->itself = w->itself;

p->parent = 0;

p->lchild = 0;

p->rchild = 0;

}

for (; i <= m; ++i, ++p){

p->weight = 0;

p->parent = 0;

p->lchild = 0;

p->rchild = 0;

}

for (i = n + 1; i <= m; ++i){ // 建赫夫曼树

//在HT[1,i-1]中选择parent为0且weight最小的两个结点，其序号为s1，s2

Select(HT, i - 1, s1, s2); //已被选择过的节点不能再选

HT[s1].parent = i;

HT[s2].parent = i;

HT[i].lchild = s1;

HT[i].rchild = s2;

HT[i].weight = HT[s1].weight + HT[s2].weight;

}

HC = (HuffmanCode)malloc((n + 1) \* sizeof(char\*));// 0号单元未用

cd = (char\*)malloc(n \* sizeof(char));

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

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

start = n - 1;

for (c = i, f = HT[c].parent; f != 0; c = f, f = HT[f].parent) {

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

cd[--start] = '0';

}

else {

cd[--start] = '1';

}

}

HC[i] = (char\*)malloc((n - start) \* sizeof(char));

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

}

free(cd);

}

功能：建立哈夫曼树

Para HT: 指向哈夫曼树的指针

Para HC: 指向存储哈夫曼编码表的数组的指针

Para w: 指向存储每个字符的值和权重的数组的指针

Para n: 不同字符的数量

返回值：无返回值

复杂度：O(n^2)

void Decode(HuffmanTree& a，char\* code\_1)

{

cout << "解码结果如下" << endl;

//解码

count2 = 0;

HuffmanTree temp;

HuffmanTree temp2;

temp = &a[2 \* num\_l - 1];

temp2 = temp;

while (code\_1[count2] != 0) {//是否读到了尾0

if (temp2->lchild == 0 && temp2->rchild == 0) {

cout << temp2->itself;

temp2 = temp;

}

else {

if (code\_1[count2] == '0') {

temp2 = (a + temp2->lchild);

count2++;

}

else if (code\_1[count2] == '1') {

temp2 = (a + temp2->rchild);

count2++;

}

}

}

cout << temp2->itself;

}

功能：解哈夫曼编码

Para a: 指向哈夫曼树的指针

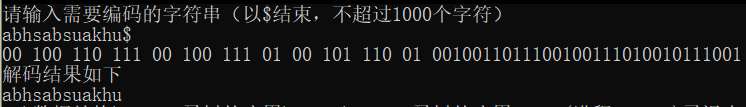
Para code\_1: 指向存储哈夫曼编码的数组的指针

返回值：无返回值

复杂度：O(n^2)

**2.5 调试分析（遇到的问题和解决方法）**

在进行哈夫曼编码时，需要选择parent为0且weight最小的两个结点。但我最初编写的select函数选择出来的结点一直是相同的两个结点。导致错误的原因是我没有对每个结点标记是否已经被选过。之后我在遍历所有节点时加入了一个判断条件：该结点的parent是否为0来区别该结点是否被选择过。随后问题得到了解决。在生成哈夫曼编码的时候，需要一开始在存储编码的数组的末尾添加一个尾0。我最初没有添加尾0导致额外输出了很多奇怪的东西。



运行结果

**3.实验总结**

设计电文总长最短的二进制前缀编码即为以n种字符出现的频率作权，设计一颗哈夫曼树的问题。由此得到的二进制前缀编码称为哈夫曼编码。建立哈夫曼树的过程为：（1）根据给定的n个权值{w1，w2，……，wn}构成n颗二叉树的集合F={T1,T2,……，Tn}，其中每棵二叉树的Ti中只有一个带权为wi的根结点，其左右子树为空。（2）：在F中选取两棵根结点的权值最小的树作为左右子树构造一棵新的二叉树，且置新的二叉树的根结点的权值为其左、右子树上根结点的权值之和。（3）：在F中删除这两棵树，同时将新得到的二叉树加入F中。（4）：重复（2）和（3），直到F只含一棵树为止。这棵树便是构造出来的哈夫曼树。求编码的过程为从叶子结点出发走一条从叶子到根的路径；而译码过程为从根出发走一条从根到叶子的路径，即按字符是0或1确定找作左孩子还是右孩子，直到叶子结点。