**姓名：岳宇轩**

**学号：19020011038**

**科目：数据结构与算法**

**指导老师：纪筱鹏**

**1实验题目**

Huffman树以及Huffman编码的算法实现

**2实验目的**

1.了解该树的应用实例，熟悉掌握Huffman树的构造方法及 Huffman编码的应用，

2.了解Huffman树在通信、编码领域的应用过程**。**

**3实验要求**

1.输入一段100—200字的英文短文，存入一文件a中。2.写函数统计短文出现的字母个数n及每个字母的出现次数

3.写函数以字母出现次数作权值，建Huffman树(n个叶子)，给出每个字母的Huffman编码。

4.用每个字母编码对原短文进行编码，码文存入文件b中。

5.用Huffman树对b中码文进行译码，结果存入文件c中，比较a、c是否一致，以检验编码、译码的正确性。

**4实验内容和实验步骤**

**4.1需求分析**

陈述程序设计的任务，强调程序要做什么，明确规定:

1. 输入的形式和输入值的范围;

输入的形式：输入字符

输入值范围：0-128的字符

1. 输出的形式;

输出b和c是否一致

1. 程序所能实现的功能;

构建Huffman树以及Huffman编码，对输入进行编码和解码

**4.2概要设计**

**4.2.1数据结构定义**

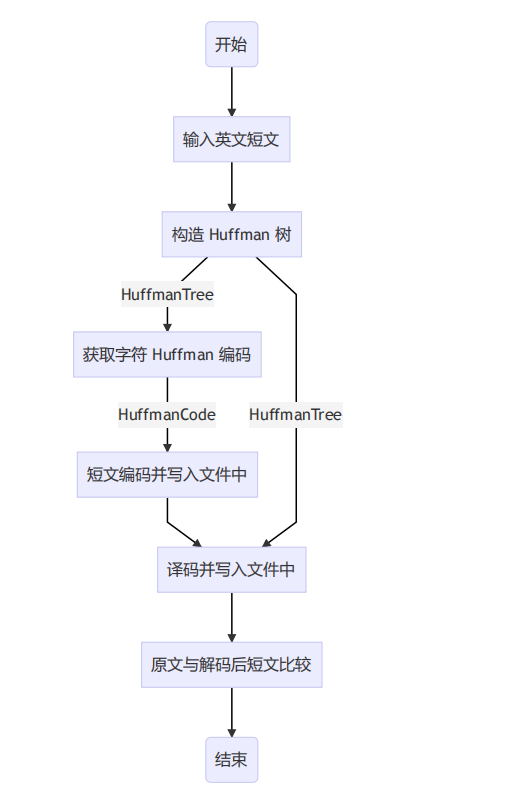
Huffman树定义

typedef struct {  
 char letter;  
 unsigned int weight;  
 unsigned int parent, left, right;  
} Node, \*Tree;

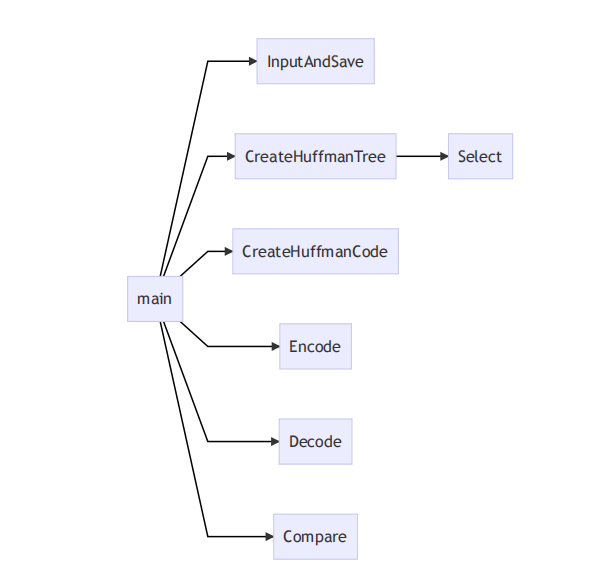
Huffman编码定义

typedef char \*Code;

4.2.2主程序流程



4.2.3各程序模块之间的调用关系



4.3详细设计

p.s.可能是编码的问题，加中文注释编译不通过，一直没有解决这个问题，所以把相关函数的分析写在下面了）

4.3.1主程序入口

int main() {  
 Code hc[200] = {nullptr};  
 Tree ht;  
 int n;  
 InputAndSave("a.txt");  
 CreateHuffmanTree("a.txt", ht, n);  
 CreateHuffmanCode(hc, ht, n);  
 Encode("a.txt", "b.txt", hc, ht, n);  
 Decode("b.txt", "c.txt", ht, n);  
 printf("a.txt is %s to c.txt", Compare("a.txt", "c.txt") ? "equal" : "not equal");  
 free(ht);  
 return 0;  
}

分析：主函数一开始声明了三个变量：hc，ht,n，分别代表Huffman编码，Huffman树，输入文章中不同的字符数。接着一次调用有如下功能的函数：

1输入英文文章并保存到a.txt中

2构造Huffman树

3获取Huffman编码

4短文编码

5短文解码

6调用函数比较原文与解码后的结果

4.3.2文章读入

void InputAndSave(const char \*filename) {  
 FILE \*fp = fopen(filename, "w");  
 printf("Please input an essay, end with an enter:");  
 while (true) {  
 char ch = getchar();  
 if (ch == '\n')  
 break;  
 fputc(ch, fp);  
 }  
 fclose(fp);  
}

分析：先打开（没有则新建）一个文件，每次循环得到一个输入字符并存入文件中，当输入为换行符时退出循环。

4.3.3构造Huffman树

void CreateHuffmanTree(const char \*filename, Tree &ht, int &n) {  
 FILE \*fp = fopen(filename, "r");  
 int count[128] = {0};  
 n = 0;  
 while (true) {  
 char ch = fgetc(fp);  
 if (ch == EOF)  
 break;  
 if (count[ch] == 0)  
 n += 1;  
 count[ch] += 1;  
 }  
 fclose(fp);  
  
 ht = (Tree) malloc(2 \* n \* sizeof(Node));  
  
 Tree p = ht + 1;  
 for (int i = 0; i < 128; i++) {  
 if (count[i] != 0) {  
 p->letter = i;  
 p->weight = count[i];  
 p->parent = 0;  
 p->left = 0;  
 p->right = 0;  
 p = p + 1;  
 }  
 }  
  
 for (int i = n + 1; i < 2 \* n; i++) {  
 int s1 = 0, s2 = 0;  
 Select(ht, i - 1, s1, s2);  
 ht[s1].parent = i;  
 ht[s2].parent = i;  
 ht[i].weight = ht[s1].weight + ht[s2].weight;  
 ht[i].parent = 0;  
 ht[i].left = s1;  
 ht[i].right = s2;  
 ht[i].letter = '\0';  
 }  
}

分析：

1. 首先用一个count[128]的数组用来存储字符在文章中出现的次数，如果是首次出现，则表示文章中字符类型增加1，所以n+=1。
2. 不使用0号单元，从0-127遍历count数组，如果不为0，表示文章中有出现这个字符，则需要将它存入Huffman树，初始化节点的letter为其字符本身，weight为出现次数，左右子树和父亲默认为0.
3. 回顾Huffman树的构建过程：从所有没有父节点的节点中挑出权值最小的两个，作为左右子树合并到一个新节点上。所以我们要先在ht[1....i-1]中挑出weight最小的两个。这里使用的Select函数（后面会具体分析该函数的实现）。
4. 设置这两个节点的父节点为新节点，设置新节点的weight为这两个的weight之和，左右孩子分别为这两个节点，父节点默认为0，letter值为’\0’（这个其实无所谓）.

Select函数：

void Select(Tree ht, int n, int &s1, int &s2) {  
 int count = 0;  
 for (int i = 1; i <= n; i++) {  
 if (ht[i].parent != 0)  
 continue;  
 else {  
 if (count == 0) {  
 s1 = i;  
 count++;  
 } else if (count == 1) {  
 if (ht[i].weight < ht[s1].weight) {  
 s2 = s1;  
 s1 = i;  
 } else {  
 s2 = i;  
 }  
 count++;  
 } else {  
 if (ht[i].weight < ht[s1].weight) {  
 s2 = s1;  
 s1 = i;  
 } else if (ht[i].weight < ht[s2].weight && ht[i].weight > ht[s1].weight) {  
 s2 = i;  
 } else {  
 continue;  
 }  
 }  
 }  
 }  
}

分析：

1用一个变量count来记录已经挑选出的节点数量，自然设置初始值为0。用s1记录最小的结点，用s2记录次最小结点。

2对ht[1..n]进行循环遍历（这里的n在构造Huffman树的函数中，随着每次调用Select函数，它的值都会+1.也就是说,ht[0]是不使用的空间，ht[1..n]存储叶子节点，ht[n+1..2n-1]存储Huffman树构建过程中新加入的节点）

3对于其中没有父节点的结点，有以下三种情况：count==0，count==1，count>1

4对于count==0的情况，直接把当前节点作为s1

5对于count==1的情况，如果当前节点weight小于s1，则把s1赋值给s2，把当前节点赋值给s1；否则，把当前结点赋值给s2

6对于count>1的情况，当前结点weight小于s1，则把s1赋值给s2，把当前节点赋值给s1；若当前结点weight大于s1且小于s2，则把当前结点赋值给s2；初次之外不进行其它操作

4.3.4获取Huffman编码

void CreateHuffmanCode(Code hc[], Tree ht, int n) {  
 int position;  
 for (int i = 1; i <= n; i++) {  
 char \*cd = (char \*) malloc(n \* sizeof(char));  
 cd[n - 1] = '\0';  
 position = n - 2;  
 unsigned parent = ht[i].parent;  
 unsigned current = i;  
 while (parent != 0) {  
 if (current == ht[parent].left)  
 cd[position] = '0';  
 else  
 cd[position] = '1';  
 current = parent;  
 parent = ht[parent].parent;  
 position--;  
 }  
 position++;  
 cd = cd + position;  
 hc[i] = cd;  
 }  
}

分析：

1. 对于每个叶子结点，从叶子结点向上走到根节点，求取Huffman编码，用cd存储它的编码，最终存入hc中。
2. 用current指向当前节点，用parent指向当前节点的父节点，用position记录当前边的0/1应当存入cd的位置。
3. 循环遍历每个叶子结点，当其节点不为0时，若current是parent的左孩子，则在position处放入0，否则放入1
4. cd指针前移：由于不同叶子结点深度不同，所以其编码长度也是不同的。cd前移position个单位，可以指向最后一次存入编码（也就是编码首位）的位置。
5. 将cd存入hc中

4.3.5短文编码

void Encode(const char \*src, const char \*dst, Code hc[], Tree ht, int n) {  
 FILE \*fsrc = fopen(src, "r");  
 FILE \*fdst = fopen(dst, "w");  
  
 while (true) {  
 char ch = fgetc(fsrc);  
 if (ch == EOF)  
 break;  
 int i = 1;  
 for (; i <= n; i++) {  
 if (ht[i].letter == ch)  
 break;  
 }  
 int j = 0;  
 while (hc[i][j] != '\0') {  
 fputc(hc[i][j], fdst);  
 j++;  
 }  
 }  
 fclose(fsrc);  
 fclose(fdst);  
}

分析：

该函数的实现分为两部分

1. 首先在树中查找该字符对应的位置，再在Huffman编码中找到它的编码
2. 将编码依次写入目标文件

4.3.6短文解码

void Decode(const char \*src, const char \*dst, Tree ht, int n) {  
 FILE \*fsrc = fopen(src, "r");  
 FILE \*fdst = fopen(dst, "w");  
 unsigned position = 2 \* n - 1;  
 while(true){  
 if (ht[position].left == 0) {  
 fputc(ht[position].letter, fdst);  
 position = 2 \* n - 1;  
 } else {  
 char ch = fgetc(fsrc);  
 if(ch == EOF)  
 break;  
 if (ch == '0')  
 position = ht[position].left;  
 else  
 position = ht[position].right;  
 }  
 }  
 fclose(fsrc);  
 fclose(fdst);  
}

分析：

1. 解码的过程是根据编码从根节点开始走，走到叶子结点就输出对应字符，然后再从根节点开始走
2. 根据ht的结构，2n-1的位置是存储的根节点（这里我一开始想成n-1了，卡了好久）。用position表示当前结点
3. 如果当前结点是叶子结点（可以用左孩子是0来判断），则将当前结点的letter输出到dst文件，然后要更新position的位置为根节点2n-1(注意，如果是叶子节点的话，就不要再读取编码了)
4. 如果当前结点不是叶子结点，则读取一位编码，如果是0，向左孩子走；如果是1，向右孩子走。

4.3.7原文与解码后短文比较

int Compare(const char \*first, const char \*second) {  
 FILE \*f1 = fopen(first, "r");  
 FILE \*f2 = fopen(second, "r");  
  
 while (!feof(f1) && !feof(f2)) {  
 char c1 = fgetc(f1);  
 char c2 = fgetc(f2);  
 if (c1 != c2)  
 break;  
 }  
  
 int res = 1;  
 if (!feof(f1) || !feof(f2))  
 res = 0;  
  
 fclose(f1);  
 fclose(f2);  
 return res;  
}

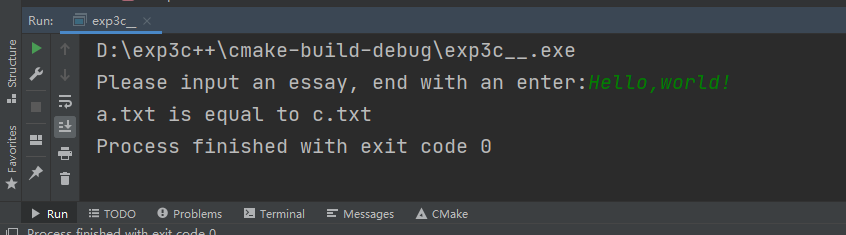
分析：逐个字符比较两个文件，如果字符不同则挑出循环，如果两个文件没有同事到达末尾，则不同。

4.4调试分析：输入总字符数为m，编码字符数n

|  |  |
| --- | --- |
| 时间复杂度 | 函数名 |
| O(n) | Select |
| O(mlogn) | Encode  Decode  Compare |
| O(n^2) | CreateHuffmanTree |
| O(m) | CreateHuffmanCode  InputAndSave |

**5实验用测试数据和相关结果分析**

**5.1实验结果**



**5.2实验总结**

心得：在求取两个最小数的时候，可以用一个变量来存储已挑选出的数量，这样能好想许多

问题：没想明白，如果Encode函数的参数不传ht的话该怎样实现。hc并不能体现字符和对应编码之间的关系，所以需要ht作为中间桥梁，先根据字符在ht中查下标，再根据下标在hc中查编码。