**程序设计与算法综合训练**

**实验报告**

|  |  |
| --- | --- |
| **实验名称**： | 哈夫曼编/译码系统 |
| **专业**： | 机器人工程 |
| **班级**： | 机器人工程三班 |
| **学号**： | WA2224013 |
| **姓名**： | 郭义月 |

**目录**

[一、 实验内容及要求 4](#_Toc183000920)

[1.1 实验目的 4](#_Toc183000921)

[1.2 实验内容 4](#_Toc183000922)

[1.3 实验要求 4](#_Toc183000923)

[1.4 实验任务 5](#_Toc183000924)

[二、 哈夫曼编/译码相关知识 5](#_Toc183000925)

[2.1 树与二叉树 5](#_Toc183000926)

[2.2 哈夫曼编码 6](#_Toc183000927)

[2.2.1 创建哈夫曼树的步骤 6](#_Toc183000928)

[2.2.2 哈夫曼解码的步骤 7](#_Toc183000929)

[三、 任务1：给定字母的哈夫曼编码 8](#_Toc183000930)

[四、 任务2：统计字符串文件中的字符频率 8](#_Toc183000931)

[4.1 存储结构 9](#_Toc183000932)

[4.1.1 Huffman树存储结构：顺序表 9](#_Toc183000933)

[4.1.2 Huffman编码存储结构：顺序表 9](#_Toc183000934)

[4.2 统计字符出现的频率 9](#_Toc183000935)

[4.2.1 问题分析 9](#_Toc183000936)

[4.2.2 结果截图 10](#_Toc183000937)

[五、 任务3实现哈夫曼编码译码系统 11](#_Toc183000938)

[5.1 从文件DataFile.data中读取字符和对应权值，建立哈夫曼树 11](#_Toc183000939)

[5.1.1 逻辑分析 11](#_Toc183000940)

[5.1.2 结果展示 12](#_Toc183000941)

[5.2 对文件ToBeTran中的文本进行编码形成报文，写入文件Code.txt 12](#_Toc183000942)

[5.2.1 任务分析与关键代码展示 12](#_Toc183000943)

[5.2.2 结果展示 13](#_Toc183000944)

[5.3 对文件CodeFile中的代码进行解码形成原文，存入文件Textfile.txt 13](#_Toc183000945)

[5.3.1 逻辑分析与关键代码 13](#_Toc183000946)

[5.3.2 结果展示 14](#_Toc183000947)

[六、 实验总结与常见问题汇总 14](#_Toc183000948)

[6.1 实验总结 14](#_Toc183000949)

[6.1.1 加深了对哈夫曼编码的理解 14](#_Toc183000950)

[6.1.2 掌握了哈夫曼树的构造方法 15](#_Toc183000951)

[6.1.3 熟练进行文件的读取与保存 15](#_Toc183000952)

[6.1.4 加深了对自定义哈夫曼结构体的理解 15](#_Toc183000953)

[6.1.5 体会到哈夫曼编码的优势 16](#_Toc183000954)

[6.1.6 提高了调试能力 16](#_Toc183000955)

[6.1.7 提高了算法实现的综合能力 16](#_Toc183000956)

[6.2 常见问题汇总 16](#_Toc183000957)

# 实验内容及要求

## 实验目的

加深对树和二叉树数据结构的理解，强化学生的逻辑思维能力和动手能力，巩固良好的编程习惯，掌握工程软件设计的基本方法，为后续课程的学习打下坚实基础。

## 实验内容

问题描述：

利用哈夫曼编码进行通信可以大大提高信道利用率，缩短信息传输时间，降低传输成本。但是，这要求在发送端通过一个编码系统对待传数据预先编码，在接收端将传来的数据进行译码（解码）。对于双工信道（即可以双向传输信息的信道），每端都需要一个完整的编/译码系统。试为这样的信息收发站设计一个哈夫曼编译码系统。

## 实验要求

基本要求：

（1）编码（EnCoding）。用已建好的哈夫曼树，对文件ToBeTran.data中的文本进行编码形成报文，将报文写在文件Code.txt中；

（2）译码（Decoding）。利用已建好的哈夫曼树，对文件CodeFile.data中的代码进行解码形成原文，结果存入文件Textfile.txt中；

（3）输出（Output）。输出DataFile.data中出现的字符以及各字符出现的频度（或概率）；输出ToBeTran.data及其报文Code.txt；输出CodeFile.data及其原文Textfile.txt。

## 实验任务

**任务一:**利用给出的字符和字符频率对字符串进行哈夫曼编码。

**任务二:**尝试统计包含大小写英文的字符串文件中的字符频率

**任务三：**

从文件DataFile.data中读取字符和对应权值，建立哈夫曼树

对文件ToBeTran.data中的文本进行编码形成报文，写入文件Code.txt

对文件CodeFile.data中的代码进行解码形成原文，存入文件Textfile.txt

输出DataFile.data中出现的字符和字符出现的频度；

输出ToBeTran.data及其报文Code.txt；

输出CodeFile.data及其原文Textfile.txt。

# 哈夫曼编/译码相关知识

## 树与二叉树

**树：**树是一种非线性的数据结构，由若干个节点和它们之间的边组成。树的一个节点可以有零个或多个子节点，除了根节点外，每个节点都有且仅有一个父节点。

**二叉树：**特殊的树结构，每个节点最多有两个子节点，称为左子节点和右子节点。每个节点都包含一个数据元素和指向左右子节点的指针。

**森林：**是多个不相交的树集合，每个集合称为一个森林。森林中的每个节点都没有父节点，可以有零个或多个子节点。可以把森林看作是若干棵树的集合，其中没有任何两棵树有交集

**树与路径相关知识**

**路径**：由一结点到另一结点间的分支所构成

**路径长度**：路径上的分支数目 root->a = 2

**树的路径长度**：从树根到每一结点的路径长度之和

**带权路径长度**：结点到根的路径长度与结点上权的乘积

**树的带权路径长度**：树中所有叶子结点的带权路径长度之和

**哈夫曼树**：带权路径长度最小的树

## 哈夫曼编码

戴维·霍夫曼发明了著名的的哈夫曼编码。这是一种应用于计算机领域的无损数据压缩算法. 哈夫曼提出了一种新的思路：按照字符出现的频率来构建一棵哈夫曼树，通过遍历哈夫曼树来得到每个字符的编码哈夫曼编码是一种可变长度编码（Variable Length Code），是基于字符出现频率设计的一种编码方式。它的主要思想是，将出现频率高的字符用较短的编码，而出现频率低的字符用较长的编码，以达到尽可能减少编码长度的目的。这种编码已广泛应用于网络通信中。

### 创建哈夫曼树的步骤

创建哈夫曼树的步骤如下：

1. 统计频率：

计算每个字符在文本中的出现频率，将每个字符及其频率作为一个节点（叶子节点）存储起来。每个节点包含字符和其对应的频率信息。

2. 构建初始节点列表：

将所有字符频率节点按频率大小进行排序，形成一个优先队列（通常是最小堆），频率最小的节点排在最前面。

3. 构建哈夫曼树：

通过以下步骤，依次从优先队列中选取两个频率最小的节点来构建树，直到最终生成根节点：

1. 取出频率最小的两个节点，作为树的左子节点和右子节点，构造一个新节点作为它们的父节点，父节点的频率等于两个子节点频率之和。
2. 将新节点加入优先队列，并重新排序（或调整堆）以保持最小频率节点排在前面。
3. 重复以上步骤，直到优先队列中只剩下一个节点时，这个节点即为哈夫曼树的根节点。

4. 生成哈夫曼编码：

在构建好的哈夫曼树中，从根节点开始，对每个左边路径标记为“0”，右边路径标记为“1”。根据路径给每个叶子节点分配一个唯一的二进制编码，这就是该字符的哈夫曼编码。

5. 应用哈夫曼编码：

用生成的哈夫曼编码对文本进行压缩，将每个字符替换成相应的编码，从而得到压缩后的二进制码流。

通过以上步骤，哈夫曼树实现了高频字符用短编码、低频字符用长编码的方式，从而达到压缩数据的效果。

### 哈夫曼解码的步骤

哈夫曼解码的过程如下：

1. 构建哈夫曼树：

解码之前，必须有哈夫曼树。哈夫曼树可以通过已知的编码表重建（即字符和对应的哈夫曼编码），或者直接用编码过程中生成的哈夫曼树来进行解码。

2. 准备编码数据：

获取已编码的二进制数据流，这是编码过程中将文本转换成哈夫曼编码后的二进制数据。

3. 从根节点开始遍历哈夫曼树：

对于二进制数据流中的每一位，从哈夫曼树的根节点开始进行遍历：

1. 如果当前位是“0”，则沿树的左分支移动到左子节点；
2. 如果当前位是“1”，则沿树的右分支移动到右子节点。

4. 到达叶子节点：

每次移动到某个叶子节点时，就找到了一个对应字符（因为叶子节点存储了字符信息）。将这个字符添加到解码结果中。

5. 返回根节点，继续解码：

找到一个字符后，返回哈夫曼树的根节点，继续处理剩余的二进制数据流，重复步骤 3 和 4，直到二进制数据流全部解码完成。

6. 输出解码结果：

将所有找到的字符组合起来，即为解码后的原始文本。

# 任务1：给定字母的哈夫曼编码

任务一：先给出8个字母 {a, b, c, d, e, f, g, h} ，它们出现的概率分别为{ 0.07, 0.19, 0.02, 0.06, 0.32, 0.03, 0.21, 0.10}，组成字符串为:abcdfhgedecf,请设计哈夫曼编码并输出编码后二进制码。

按照哈夫曼树的创建过程，左子树分配 0，右子树分配 1，得到以下编码（具体编码取决于树的结构）：

a -> 1101

b -> 00

c -> 11100

d -> 11101

e -> 10

f -> 11110

g -> 01

h -> 1100

合并所有编码结果，得到编码后的序列：11010011110011101111011000101110110111110

# 任务2：统计字符串文件中的字符频率

试着统计一下:包含大小写英文的字符串文件中的字符频率。要求：要将每个字符的大小写频率都要统计到一起

## 存储结构

### Huffman树存储结构：顺序表

typedef struct

{

int weight;

int parent, lchild, rchild;

}HTNode, \*HuffmanTree;

1. HTNode：哈夫曼树节点的结构体。每个节点包含权值（weight）和三个指针：父节点（parent）、左子节点（lchild）和右子节点（rchild）。
2. HuffmanTree：指向哈夫曼树的指针类型。实际上，HuffmanTree类型就是HTNode类型的指针，因为哈夫曼树的每个节点都是HTNode类型。

### Huffman编码存储结构：顺序表

typedef struct

{

char ch;

char code[NUM];

}HuffmanCode;

HuffmanCode：存储每个字符及其哈夫曼编码，在编码阶段建立，用于哈夫曼解码。

## 统计字符出现的频率

### 问题分析

英文文章字符的ASC II码限于0~150。读取文档的各个字符，每个字符出现频率为权值，再将数组压缩：没有出现的字符从数组中删去。

算法如下**：**

统计字符的频率，即确定权值;

定义两个数组，分别存放英文字母和英文字母出现的频率。

a. 将两个数组初始化都为0.

b. 将字母转换为数字，每一个字符对应一个数字，同一个字符出现一次频率就加1，直到全部统计。

c. 将转换的数字再转换为相应的字符，以便为下面的建树方便调用。

最后将字符与频率匹配制作成DataFile.data文件

DataFile.data文件中第一列代表字符串，第二列代表对应字符对应的权值，代码如下：

int letterCount[52] = { 0 };

char c;

int spacecount = 0;

while (inFile.get(c))

{

    if (c >= 'a' && c <= 'z')

    {

        letterCount[c - 'a']++; // 如果是小写字母，则对应字母计数器加1

    }

    if(c >= 'A' && c <= 'Z')

    {

        letterCount[c - 'A' + 26]++; // 如果是大写字母，则对应字母计数器加1

    }

    if (c == ' '){

        spacecount++;

    }

}

ofstream outFile("./data/DataFile.data");

### 结果截图

统计./data/Words.txt中字符出现的频率，放到./data/DataFile.data文件中。

|  |  |
| --- | --- |
|  |  |

# 任务3实现哈夫曼编码译码系统

## 从文件DataFile.data中读取字符和对应权值，建立哈夫曼树

### 逻辑分析

创建哈夫曼树的逻辑为：

首先，打开一个数据文件DataFile.data，在while循环中逐行读取字符及其对应的权重信息，将这些信息分别存储到 characters 和 weights 两个向量中。如果字符是 "workspace"，会用空格代替。随后，将这些字符组合成一个字符串并确定其长度，用于初始化哈夫曼树。

接着，创建一个哈夫曼树节点数组 HTree，大小为 2 \* len + 1，并初始化每个节点的 parent、lchild 和 rchild 为 0，同时将权重值赋值给前 len 个叶子节点。

然后，通过逐步合并权值最小的两个节点，构建哈夫曼树的父节点。合并的过程使用一个 Select 函数，选出当前范围内权值最小的两个节点，将它们设为新节点的左右子节点，并计算新节点的权重值。

完成哈夫曼树的构建后，程序将结果保存到文件HuffmanTree.data中，同时在控制台打印整个哈夫曼树的结构，包括节点编号、字符或权重、父节点和左右子节点的信息。最后，通过设置标志变量 TreeNum 表明哈夫曼树已成功构建。

### 结果展示

|  |  |
| --- | --- |
|  |  |

## 对文件ToBeTran中的文本进行编码形成报文，写入文件Code.txt

### 任务分析与关键代码展示

在while循环中，从当前节点开始，沿着父节点逐层向上追溯，直到根节点（父节点编号为0）。在每次迭代中，首先将编码字符串的位置前移一位，然后判断当前节点是其父节点的左孩子还是右孩子：如果是左孩子，就在编码字符串的当前位置写入字符 '0'；如果是右孩子，就写入字符 '1'。接着，将当前节点更新为父节点，继续下一次循环。通过这种方式，逐步构建从叶子节点到根节点的路径编码，最终得到对应字符的哈夫曼编码。

while (father != 0)

    {

        --start;

        // 每一次追溯到父节点时，将当前节点的编码存入cd中

        if (HTree[father].lchild == now)

            codenunm[start] = '0';

        else

            codenunm[start] = '1';

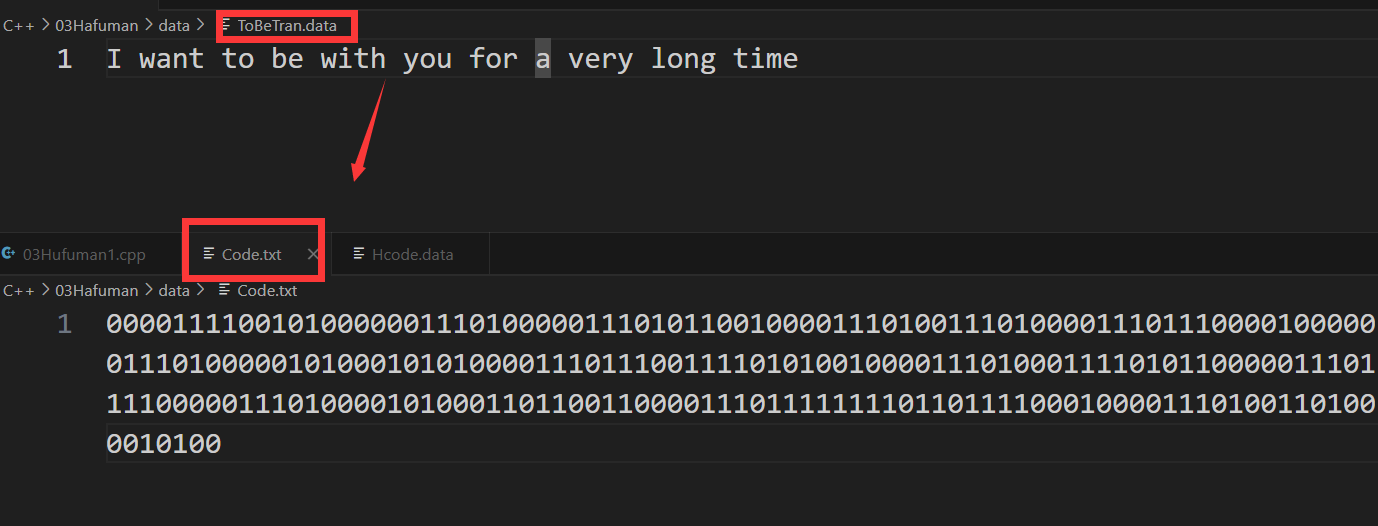
        now = father;

        father = HTree[father].parent;

        // 将当前节点的父节点赋给当前节点，向上追溯

    }

### 结果展示



## 对文件CodeFile中的代码进行解码形成原文，存入文件Textfile.txt

### 逻辑分析与关键代码

哈夫曼解码的逻辑是基于已构建好的哈夫曼树，将编码转化回原始字符。对于文件CodeFile中的代码，解码时从根节点开始，逐位读取编码数据。每读到一位编码，如果是 `'0'`，则沿着哈夫曼树向左子节点移动；如果是 `'1'`，则向右子节点移动。当到达一个叶子节点时，该节点代表一个解码出的字符，将其存储或输出，然后从根节点重新开始处理下一段编码。重复这一过程直到所有编码都被处理完毕，依次存入文件Textfile.txt，即完成整个解码过程。核心是利用编码中的每一位引导树的遍历方向，最终定位到对应的字符。

for (int i = 0; i < str.length(); ++i)

{

    // 判断索引是否超出范围

    if (i >= str.length())

    {

        break;

    }

    if (str[i] == '0') {

        q = nodes[q-1].lchild;

    }  // 如果该字符为'0'，进入左子树

    if (str[i] == '1') {

        q = nodes[q-1].rchild;  // 如果该字符为'1'，进入右子树

    }

    if (nodes[q-1].lchild == 0 && nodes[q-1].rchild == 0)

    {//当该节点为叶子节点时

        if(codes[q - 1].ch == "spacework"){

            textfile << " ";   // 将该叶子节点对应的字符写入到Textfile.txt中

            cout << " ";  // 输出该叶子节点对应的字符

        }

        else{

            textfile << codes[q - 1].ch;   // 将该叶子节点对应的字符写入到Textfile.txt中

            cout << codes[q - 1].ch;  // 输出该叶子节点对应的字符

        }

        q = len;   // 将q重置为根节点的下标

        // q = 2\*len-1;   // 将q重置为根节点的下标

    }

}

### 结果展示

|  |  |
| --- | --- |
|  |  |

# 实验总结与常见问题汇总

## 实验总结

### 加深了对哈夫曼编码的理解

通过本次实验，我对哈夫曼编码的核心思想有了更加深刻的认识。哈夫曼编码是一种基于贪心策略的最优前缀编码方法，它通过统计字符出现的频率来构建哈夫曼树，并生成长度可变的二进制编码。在实验中，通过对权值最小的两个节点进行合并，不断构建新的父节点，直至形成完整的树结构，我理解到这是如何在编码长度上实现最优解的。特别是在解码过程中，通过路径引导的方式重现字符信息，让我体会到这种编码方式的高效性和可靠性。

### 掌握了哈夫曼树的构造方法

在实验中，我从字符频率的统计出发，按照频率的大小依次构建哈夫曼树。这一过程涉及了权值初始化、动态节点分配、节点合并等多个步骤。通过不断挑选权值最小的两个节点并合并生成新的父节点，我掌握了如何用贪心策略构造哈夫曼树。特别是在实现的过程中，明确了如何使用数组来存储节点信息以及如何通过指针或索引实现节点之间的父子关联。这让我不仅对哈夫曼树的构造过程更加熟悉，也对其底层逻辑的实现有了更深的理解。

### 熟练进行文件的读取与保存

实验涉及多个文件操作，包括读取数据文件中的字符及其频率、将编码后的结果写入文件、以及从文件中读取编码结果并进行解码。这些操作让我熟悉了文件流的使用，例如 `ifstream` 和 `ofstream`，并掌握了文件路径管理和错误处理。在处理文件时，还需要特别注意文件是否存在、路径是否正确以及数据格式的匹配，这让我在编写代码时更加关注细节，提高了对文件操作的掌控能力。此外，能够成功将中间结果和最终结果存储到文件中，为调试和验证实验结果提供了便利。

### 加深了对自定义哈夫曼结构体的理解

实验过程中，我设计了一个哈夫曼树的节点结构体，其中包含了字符信息、权值以及父子节点的指针或索引。在构建哈夫曼树时，通过动态分配内存来管理这些节点，并通过父子关系实现了树的层级结构。这个实验让我认识到，在实际应用中如何灵活设计结构体以适应算法需求。例如，在编码过程中，如何高效存储和追溯节点路径；在解码过程中，如何通过结构体的层次信息定位字符。这种自定义数据结构的使用大大提高了代码的可读性和扩展性，也为以后实现类似的算法提供了经验。

### 体会到哈夫曼编码的优势

在实验中，我深刻感受到哈夫曼编码在数据压缩中的优越性。它根据字符出现的频率动态调整编码长度，频率越高的字符使用越短的编码，这种方法有效地减少了存储空间。相比于固定长度的编码方式，哈夫曼编码能够显著提升存储效率，尤其是当字符分布不均匀时，其优势更加明显。在实验中，通过观察编码后的结果文件和解码还原的原文，能够清晰地感受到这种压缩效果。这让我理解到，哈夫曼编码在实际应用中，如文件压缩和数据传输中的重要性。

### 提高了调试能力

在实现哈夫曼编码和解码的过程中，我遇到了许多问题，例如字符权值读取错误、文件路径设置不当、编码和解码逻辑不一致等。通过逐步调试代码，我学会了如何分解问题并逐一验证每个模块的正确性。在文件操作中，通过调试工具和日志输出，我成功定位到文件打开失败的原因并修复了路径问题。在编码解码中，通过逐步打印节点状态和中间结果，验证了树结构的正确性。这些调试经验让我在面对复杂问题时更加自信，也让我更加理解调试在开发中的重要性。

### 提高了算法实现的综合能力

实验中，哈夫曼编码的实现不仅涉及到算法设计，还结合了文件操作、数据结构管理、内存分配等多个方面。这让我在实际动手操作中，将所学的算法知识与编程技巧结合起来。例如，通过文件读取字符频率，并动态生成树结构，再将结果写入文件的完整流程，锻炼了我对程序整体架构的设计能力。此外，在实验中，我认识到算法效率在实际应用中的重要性，例如如何优化权值选择的过程以减少时间复杂度。这让我不仅提升了代码实现能力，也增强了对算法优化的关注。

## 常见问题汇总

本任务涉及到的文件名字比较多，为了方便自己理解与代码的编写，可以把每个文件的名字和关系自己先捋顺，这样有利于代码的编写