北京郵電大學

实验报告

题目：实验三：树-哈夫曼编/译码器

班　　级：2022211320

学　　号：2022211683

姓　　名：张晨阳

学　　院：计算机学院（国家示范性软件学院）

2023 年 11 月 30 日

一、实验目的

1. 熟悉哈夫曼树的构造实现及应用；
2. 培养根据实际问题合理选择数据结构的能力；
3. 学习自己查找相关资料以解决实际问题的能力。

二、实验环境

1. Windows 11
2. Visual Studio Code 1.83.1
3. x86-64 gcc 10.3

三、实验内容（简单行编辑程序）

问题描述：

哈夫曼编码是一种基于最优二叉树的无损编码方案。需要根据字符集和频度的实际统计构建哈夫曼树，然后进行编码和译码。

基本要求：

1. 初始化，从终端读入字符集大小n，以及n个字符和n个权值，建立哈夫曼树，并将它存于文件hfmTree中。
2. 编码：利用已经建好的哈夫曼树，对文件ToBeTran中的报文进行编码，然后将结果存入文件CodeFile中。（为简化处理，可以在CodeFile中用一个字节来存储码字中的一个0/1比特位）
3. 译码：利用已建好的哈夫曼树，对CodeFile中的代码进行译码，结果存入TextFile中。

选做内容

对一个 512 * 512 的 lena.bmp 灰度图片进行哈夫曼编码。BMP 文件由：BMP 文件头+像素数据组成，灰度图1个像素占用1个字节。lena.bmp 文件大小是263222字节，包括1078字节的头部+ 512 * 512 个像素值。lena.bmp 文件见实验作业附件

四、实验步骤

0. 准备工作

用来实验初步测试的编码字符集为 PPT 中的例子，即：

```
对time tries truth哈夫曼编码
字符集      D={t, i, m, e, r, s, u, h}
出现频次 W={4, 2, 1, 2, 2, 1, 1, 1}
```

1. 初始化

确定建立哈夫曼树的方法

1. **创建叶子节点数组：** 首先创建一个包含 n 个叶子节点的数组 `leaves`。每个叶子节点都包含一个字符值（从输入数组 `characters` 中获取）、权值（从输入数组 `weights` 中获取），以及左右子节点指针，初始时左右子节点指针为 `NULL`。
2. **构建哈夫曼树：** 在每一次循环中，找到权值最小的两个节点，合并成为一个新的节点，然后将新节点插入到数组中。
    1. **找到两个权值最小的节点：** 在每次循环中，先找到数组中权值最小的两个节点，记录它们的索引 `min1` 和 `min2`。具体方法：
    首先比较第一个和第二个节点，将较小的节点索引记录为 `min1`，较大的节点索引记录为 `min2`，然后从数组的第三个节点开始遍历，依次更新 `min1` 和 `min2`。
    2. **创建新节点并合并：** 使用找到的两个最小权值的节点，创建一个新的节点，该新节点的权值为这两个节点的权值之和。新节点的左右子节点分别指向这两个最小权值的节点（默认权值小的为左节点）。
    3. **更新数组：** 将新节点加入到数组中，并删除已经合并的两个最小权值的节点。这里使用 `leaves[min1]` 存储新节点，而 `leaves[min2]` 存储数组中最后一个节点，同时减小数组的大小 `n--`。
3. **重复步骤 2：** 回到循环开始，直到数组中只剩下一个节点，这个节点就是哈夫曼树的根节点。
4. **返回根节点：** 最终返回哈夫曼树的根节点，即数组中唯一剩下的节点 `leaves[0]`。

递归方式实现哈夫曼树的编码并写入文件

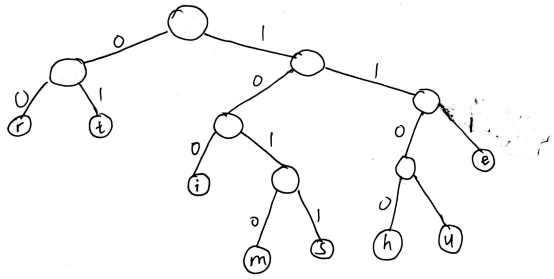
1. **确定函数参数：**
    - `root`: 当前节点。
    - `code[]`: 存储当前路径的数组，用于构建字符的编码。
    - `top`: 当前路径的长度，即当前编码的位置。
    - `outFile`: 输出文件指针，用于将字符和对应的编码写入文件。
2. **递归遍历哈夫曼树：**
对于每个节点，如果它有左子节点，将 `'0'` 存储在 `code` 数组的当前位置，然后递归调用 `encodeHuffmanTree` 处理左子节点。同样，如果有右子节点，将 `'1'` 存储在 `code` 数组的当前位置，然后递归调用 `encodeHuffmanTree` 处理右子节点。
3. **处理叶子节点并输出到文件：**
当递归到叶子节点（即没有左子节点和右子节点的节点）时，使用 `fprintf` 函数将该字符以及当前路径（即 `code` 编码）写入到文件中。

输入初始化数据

```
PS D:\VSCODE\C_C++\Lab03> .\code
Step 1:Initialize
Enter the number of characters: 8
Enter the characters and their weights:
t 4
i 2
m 1
e 2
r 2
s 1
u 1
h 1
```

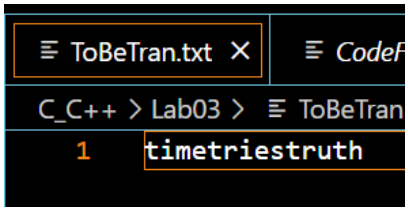
```
PS D:\VSCODE\C_C++\Lab03> cat hfmTree

r:00
t:01
i:100
m:1010
s:1011
h:1100
u:1101
e:111
PS D:\VSCODE\C_C++\Lab03>
```

左侧为输入的字符集，右侧为存入文件的已建立好的哈夫曼树。
下图即为建立的哈夫曼树：



2. 编码

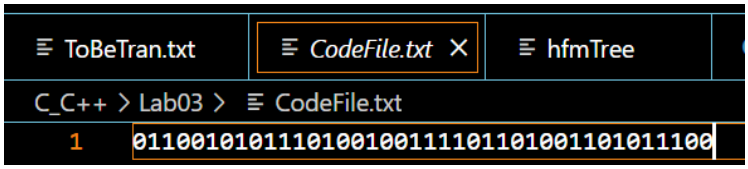
文件 ToBeTran 内容：



1. **读取字符：**
   - 使用 `fgetc` 函数逐字符读取输入文件。`fgetc` 函数每次读取一个字符，并且在读取到文件结尾时返回 `EOF`。
   - `while` 循环持续进行，直到读取到文件结尾 `EOF`。
   - 在循环中，每次读取一个字符 `ch`。
2. **编码：**
   - 调用函数 `getHuffmanCode` 来获取字符 `ch` 相对应的哈夫曼编码，并用指针 `huffmanCode` 指向编码结果。
3. **获取哈夫曼编码函数 `getHuffmanCode` 的实现：**
   - 输入参数：
     - `root`：哈夫曼树的根节点。
     - `ch`：目标字符，要获取其哈夫曼编码。
   - 调用 `getHuffmanCodeHelper` 辅助函数。
     - `getHuffmanCodeHelper` 辅助函数实现：
     - 输入参数：

        - `root`：哈夫曼树的根节点。
        - `ch`：目标字符，要获取其哈夫曼编码。
        - `path`：存储当前路径的字符数组。
        - `pathLen`：当前路径的长度。
        - `result`：存储最终哈夫曼编码的字符数组。
    - 该函数与 `encodeHuffmanTree` 相似，通过递归方式遍历哈夫曼树，从根节点开始，根据左右子树的关系向下寻找目标字符的叶子节点。
    - 在路径上添加字符 '0' 表示向左子树移动，添加字符 '1' 表示向右子树移动。
    - 当到达叶子节点时，检查是否为目标字符，如果是，则将当前路径拷贝到 `result` 中。
4. **将当前字符的编码写入文件**
    - 使用 `fputs(huffmanCode, outputFile)` 将其写入到输出文件中。
5. **结束处理：**
    - 循环结束后，所有字符都已经被读取并相应地进行了编码，并将编码结果写入了输出文件。使用 `fclose` 关闭文件流并输出成功编码的信息。

编码结果：

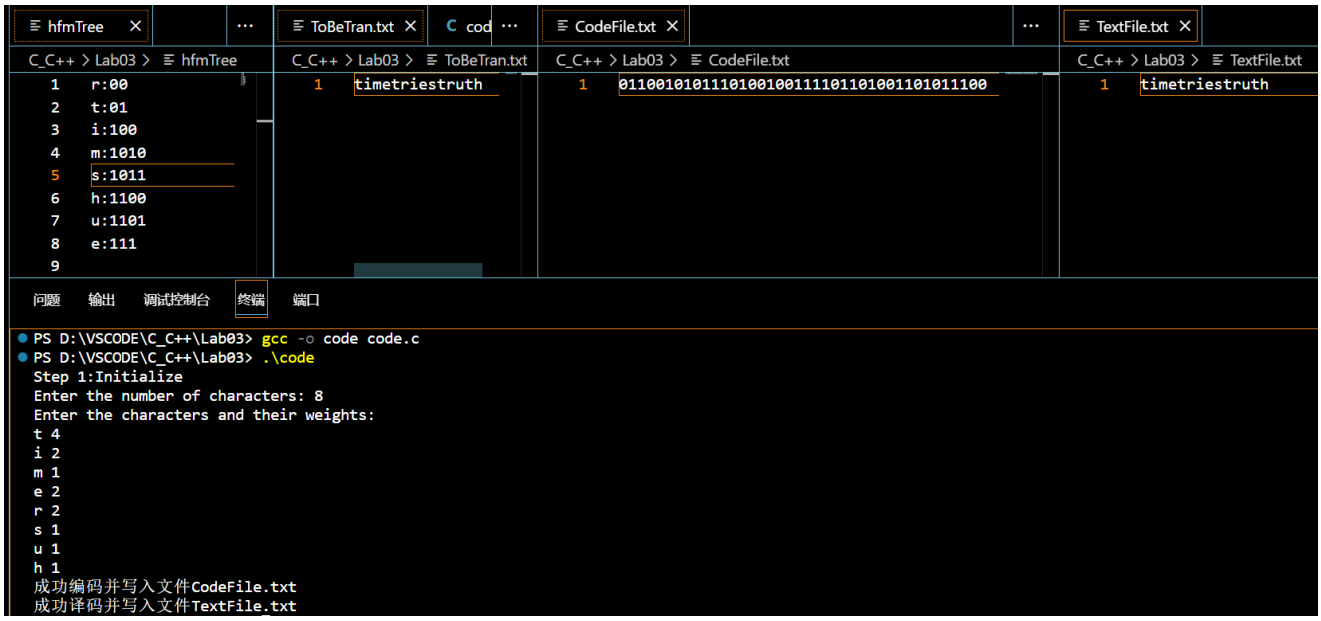


3. 译码

译码并写入文件

1. **初始化：**
   使用一个指针 `current` 来跟踪当前在哈夫曼树中的位置，初始位置为根节点 `root`。`ch` 用于存储读取的每个二进制字符。
2. **遍历输入文件：**
   使用 `fgetc` 逐字符读取输入文件，直到文件结束（`EOF`）为止。
3. **根据码字移动并检查叶子节点：**
   根据读取的二进制字符 `ch` 决定向左或向右孩子移动；如果当前节点是叶子节点，表示找到了一个字符，将该字符写入输出文件，并将当前节点重置为根节点，以便继续译码下一个字符。
4. **关闭文件流并输出译码成功信息**

译码结果展示：



4. 界面完善、测试案例补充及程序检验

界面完善

添加了选项界面，以及每次操作成功/失败后的信息提示：

```
PS D:\VSCODE\C_C++\Lab03> .\HuffmanTree
Step 1:Initialize
Enter the number of characters: 8
Enter the characters and their weights:
t 4
i 2
m 1
e 2
r 2
s 1
u 1
h 1
Successfully build Huffman tree and store in file!

1. Build Huffman tree and store in file(Already done. Don't choose!)
2. Encode file content
3. Decode message
4. Exit
Please enter the operation you want to perform: 2
Successfully encoded and written to the file CodeFile.txt

1. Build Huffman tree and store in file(Already done. Don't choose!)
2. Encode file content
3. Decode message
4. Exit
Please enter the operation you want to perform: 3
Successfully decoded and written to the file TextFile.txt!

1. Build Huffman tree and store in file(Already done. Don't choose!)
2. Encode file content
3. Decode message
4. Exit
Please enter the operation you want to perform: 4
Exit
```

测试案例补充及程序检验

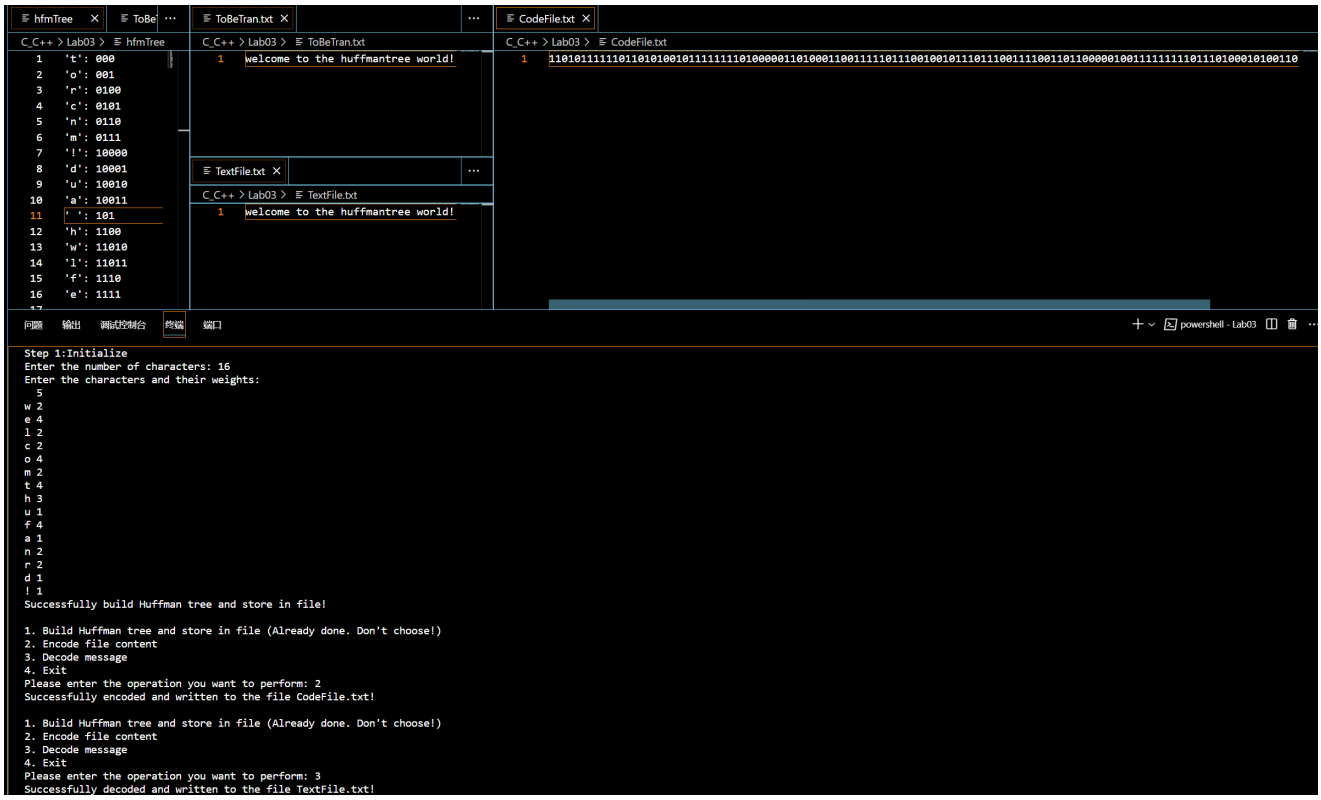
测试案例增加空格字符 `' '` 以及标点符号 `'!'`

报文为：`welcome to the huffmantree world!`

字符权值：

字符	空格	w	e	l	c	o	m	t	h	u	f	a	n	r	d	!
权值	5	2	4	2	2	4	2	4	3	1	4	1	2	2	1	1

测试结果如下：



成功。

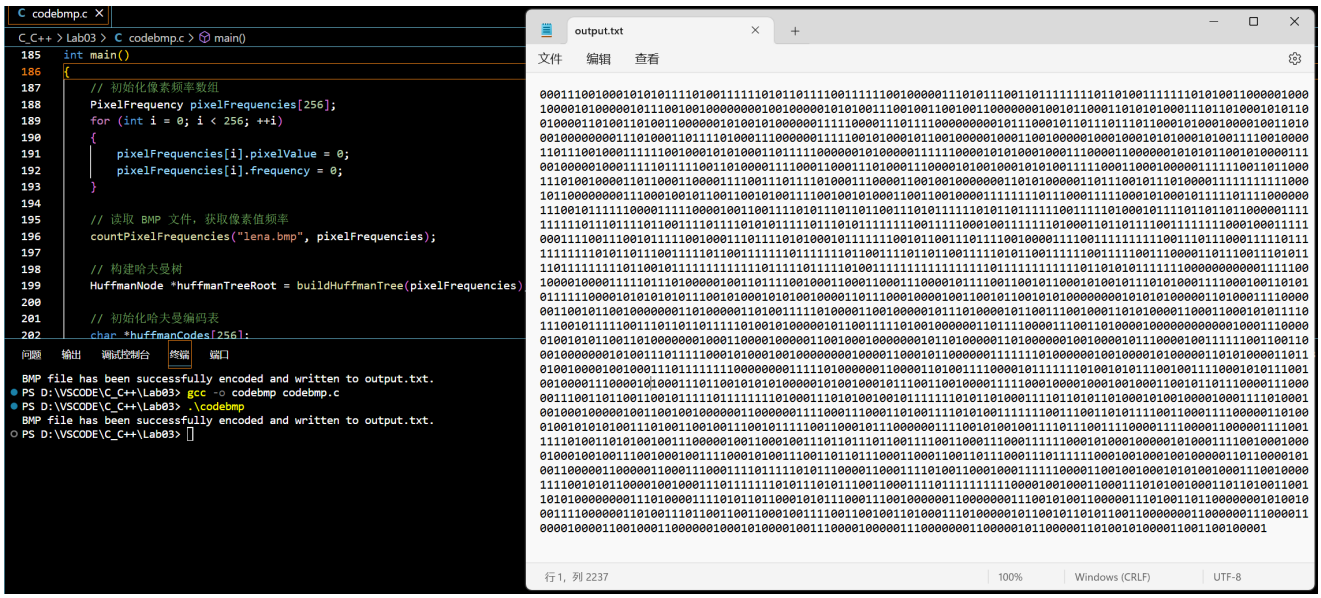
5. 选做内容

1. **读取 BMP 文件，获取像素值频率**
   - **文件打开：** 注意使用二进制读取模式（rb）打开 bmp 文件。
   - **移动文件指针：** 使用 `fseek` 将文件指针移动到像素数据的开头，由题目所给可知像素数据从 BMP 文件的第 1078 个字节开始，故为 `fseek(inputFile, 1078, SEEK_SET);`
   - **读取像素数据：** 将像素数据读入数组 `pixelData`。图像大小 512 x 512。
   - **统计像素频率：** 遍历 `pixelData` 中的每个像素，并检查像素值是否已经在 `pixelFrequencies` 中。如果找到，则增加该像素值的频率。如果未找到，则在 `pixelFrequencies` 中寻找一个空槽（其中 `pixelValue` 为 0），并将新的像素值添加到该位置，并设置频率为 1。
2. **构建哈夫曼树，生成哈夫曼编码表**
   这部分类似基本要求的代码，原理相同，使用递归实现编码表，不多赘述。
3. **编码并写入文件**
   - 前部分打开文件不多赘述。
   - **读取 BMP 文件并进行编码：**

- 在一个循环中，使用 `fread` 函数逐个读取 BMP 文件中的像素值（每次读取一个字节）。
- 对于每个读取的像素值，通过 `huffmanCodes` 数组找到其对应的 Huffman 编码，并使用 `fprintf` 函数将该编码写入到输出文件中。

- 编码结束后，关闭文件流并输出成功信息。

操作结果如下：



五、实验分析和总结

所遇问题及解决方法

- **输入字符和权值时未处理缓冲区导致的乱码问题**
  直接在循环中使用 `scanf("%c %d", &characters[i], &weights[i]);` 接受输入，导致换行符号接受到 `%c` 中，最终导致建立的哈夫曼树和写入文件的内容乱码。
  解决方法： 在每行输入后，接一个 `getchar();` 即可。如图：

```
for (int i = 0; i < n; i++)
{
    scanf("%c %d", &characters[i], &weights[i]);
    getchar();
}
```

- **更新节点数组时下标错误导致哈夫曼树建立出错**

```
Step 1:Initialize
Enter the number of characters: 8
Enter the characters and their weights:
 t 4
 i 2
 m 1
 e 2
 r 2
 s 1
 u 1
 h 1
PS D:\VSCODE\C_C++\Lab03> cat hfmtree
e00
r01
h100
m1010
s1011
u1101
i1111
```

上图为发生的错误：hfmTree 文件中只存储了七个字符且编码错误。

经排查发现：在建立哈夫曼树的更新数组这一步中，应该将数组中最后一个节点存储到 `leaves[min2]` 中，而其下标为 `n-1`，一开始直接写成了 `n--`，会导致访问出界，无法正确建立哈夫曼树。

```
// 删除已经合并的两个节点，并将新节点加入数组
leaves[min1] = newNode;
leaves[min2] = leaves[n--];
```

解决方法： 将代码修改如下图：

```
// 删除已经合并的两个节点，并将新节点加入数组
leaves[min1] = newNode;
leaves[min2] = leaves[n - 1];
n--;
```

成功建立哈夫曼树。

- **fputc 和 fputs 函数区别问题**

  程序报错：

```
PS D:\VSCODE\C_C++\Lab03> gcc -o code code.c
code.c: In function 'encodeFile':
code.c:163:15: warning: passing argument 1 of 'fputc' makes integer from pointer without a cast [-Wint-conversion]
        fputc(huffmanCode, outputFile);
              ^~~~~~~~~~~
In file included from code.c:1:
D:/x86_64-8.1.0-release-posix-sjlj-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/stdio.h:602:25: note: expected 'int' but argument is of type 'char *'
   int __cdecl fputc(int _Ch,FILE *_File);
                       ~~~~^~~
```

  经排查，发现 `fputc` 函数只能写入一个字符，而传入参数 huffmanCode 是字符串，故报错

  解决方法： 改为使用 `fputs` 函数即可。

```
// 获取字符 ch 对应的哈夫曼编码
char *huffmanCode = getHuffmanCode(root, ch);

// fputc(huffmanCode, outputFile);
fputs(huffmanCode, outputFile);
```

译码方法的选择

思考了两种实现哈夫曼译码的方法：

1. 依次增一位哈夫曼码，遍历哈夫曼树比较，时间复杂度 $O(n^2)$.
   - 将 first 指向**第一个字符**，代表从这个字符开始，**pos 赋值为 1**（代表选取一个字符），将选取得到的字符与哈夫曼树中的每个哈夫曼码进行对比；
   - 如果两字符**相等**，将此哈夫曼码相对应的字符直接输出，**first 指向 pos+first** 字符处，**pos 重置为 1**；若**不相等**，**pos+1**（选取两个字符比较），接着进行比较；
   - 重复前两步直到译码结束。
2. 依次取编码串的一位，根据码字查找哈夫曼树，时间复杂度 $O(n)$.
   - 从左到右**遍历编码串**，同时从根结点开始查找 Huffman 树；
   - 编码串出现**0（1）**，则进入 Huffman 树**左（右）孩子结点**，验证其是否为**叶子结点**：若否，则重复此步骤；若是，则输出该字符，并重置查找位置为根节点；
   - 重复前两步直到译码结束。

经比较，选择了第二种方法译码。

总结心得

这次实验加深了我对哈夫曼树以及编码译码的理解，特别是对于建立哈夫曼树、构造哈夫曼编码表、哈夫曼编码的过程清晰明了了很多，对于他们的操作也熟悉了不少。
最大的收获我认为是对译码方法的思考部分：除了题目要求里的提示方法（用一个字节来存储码字中的一个比特位），我还上网查阅了译码方法的资料，学习了不同的时间复杂度的译码方法，最终选择了时间复杂度为 $O(n)$ 的方法，这个过程大大的丰富了我对树的理解和使用。

六、程序源代码

基本要求部分源代码见源文件 HuffmanTree.c
附加要求部分源代码见源文件 codebmp.c