# 哈夫曼编码实现压缩与解压缩实验报告

目录

[一：需求分析 2](#_Toc86412534)

[二: 概要设计 2](#_Toc86412535)

[三：调试分析报告 5](#_Toc86412542)

[四：设计实验的回顾讨论 6](#_Toc86412543)

[五：用户使用说明 6](#_Toc86412544)

[六：测试结果 7](#_Toc86412545)

[七：实验结果 7](#_Toc86412546)

题目：哈夫曼编码实现压缩与解压缩

班级：2020211306 姓名：马天成 学号：2020211376 分工：撰写代码+实验报告

班级：2020211306 姓名：黄洪建 学号：2020211371 分工：撰写代码

班级：2020211306 姓名：马紫薇 学号：2020211392 分工：撰写实验报告

# 一：需求分析

## 题目的目的：

将任意一个指定的文件进行哈夫曼编码，并以真正的二进制位生成一个二进制文件（压缩文件）；反过来，可将一个压缩文件解码还原为原来的文件。

**输入形式：**

将文件以二进制的方式读入，一次读入一定的长度。

**输出形式**

输出形式有两种：

1. 压缩：得到文件中每个字符的权值后，构造对应哈夫曼的哈夫曼树，从而得到对应哈夫曼编码。编码后以二进制文件的格式（.hfm文件）输出。
2. 解压：得到每个字符的编码后，解码二进制文件并输出解压后的文件。

**二: 概要设计**

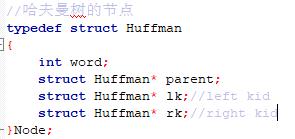
**问题解决思路：**

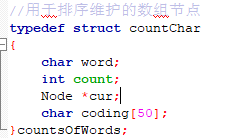
压缩：以二进制的形式读入文件，以8位即每个字节转换为相应的ascii码，随后存入数组中，统计这些ascii码出现的次数，扫描完整个文件后，

快速组成一个数组，然后建立一个哈夫曼树，每建立一个节点，就对数组进行一次维护，然后重新找两个最小的值再建立新结点，构造完哈夫曼树后，使用先序遍历遍历每个叶子节点，并且在遍历过程中记录路径，从而制作出每个字符对应的编码，并把这些编码输出到压缩文件中。最后再扫描一次文件，使用对应的编码替代对应的ascii码，自此完成压缩。

解压：先读入所有字符对应的编码，逐个字符读入，将其转化为二进制存放在数组中，然后对其解码，输出到解压文件中。

**存储结构定义**





**主函数伪代码**

while (模式不等于3)

{

if (模式等于1)

进行编码（进入编码函数）

else if(模式等于2)

进行解码（进入解码函数）

}

**子函数伪代码**

char selectMode(void)

{

while (模式不等于1，2，3)

{

提示用户输入

扫描输入

}

}

void encode(void)

{

打开文件

统计字符频率

建哈夫曼树

进行编码

输出二进制文件

}

void decode(void)

{

char\* openFile(void)

{

while (文件无法打开)

{

提示用户输入文件路径

}

}

countsOfWords\* countFrequency(FILE\* doc,int\* numsOfWord)

{

while (文件没有读完)

{

字符出现数加一

}

}

Node\* createTree(countsOfWords\* words,int count)

{

while (没有跑到数组尽头)

{

为i和i+1的节点创建父节点

i+1的节点指向父节点

i=i+1

重新排序

}

}

char\*\* encoding(countsOfWords\* words,Node\* root);

{

申请数组空间并初始化

深搜哈夫曼树，往左走为0，往右走为1

}

void outputFile(FILE\* in,FILE\* out,char\*\* words, char\* filePath);

{

更改输出文件名

while (文件没读完)

{

把字符对应的编码存入缓冲区

if (缓冲区字数大于等于8)

转换为ascii码输出

}

}

void getCode(FILE\* doc,int words[])

{

while (i<256)

{

读取编码

}

}

void outputDecodeFile(FILE\* doc,int words[],char\* fileName);

{

while (文件没读完)

{

将读入的字符存入缓冲区2

缓冲区2贴去缓冲区1

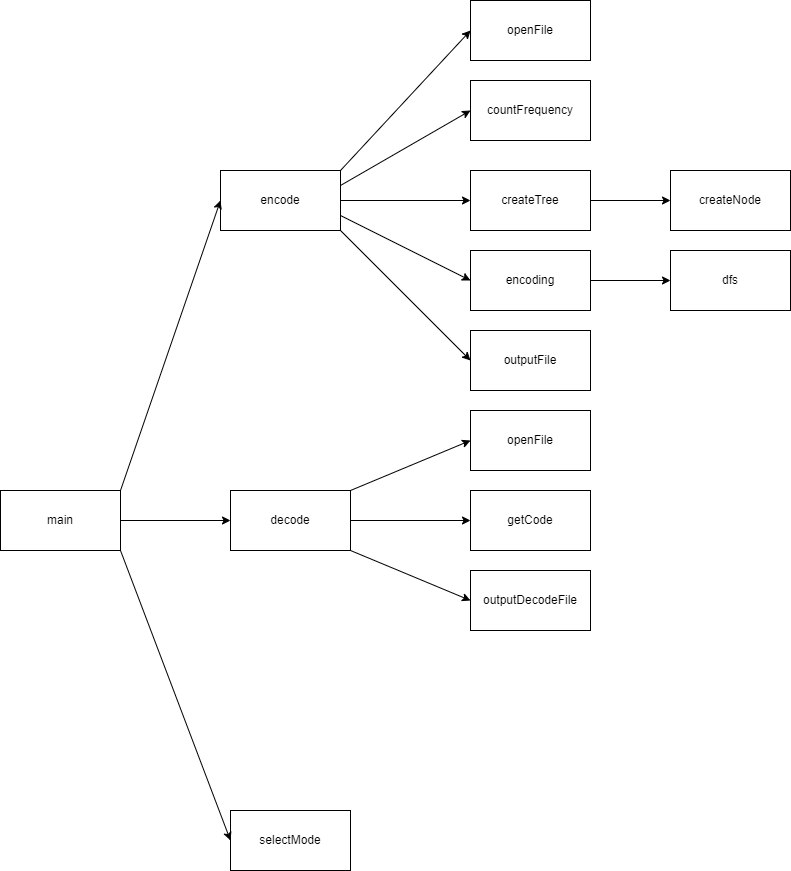
if (缓冲区1中符合编码规则)

输出原字符

}

}

**函数调用关系**



**三：调试分析报告**

**遇到的问题及解决方法**

1. 不会用qsort函数对结构体进行快速排序

解决方法：通过与队友交流和不断尝试找到了正确的方法

1. 没有考虑到有负数的ascii码，导致统计部分ascii出现次数的时候会出现bug

解决方法：将ascii码加上128再放入数组

1. 解压文件命名格式没有考虑到二次压缩和解压的情况，因此在二次解压时文件名会缺少一部分

解决方法：压缩时无论文件后缀名是什么，都加上.hfm

1. 当输入不是数字的模式时会陷入死循环

解决方法：把模式的格式由整型改为字符型

**时空复杂度分析**

总体时间复杂度：

编码：O(nlogn)，编码程序中时间复杂度上限是qsort函数，其时间复杂度为O(nlogn)。

解码：O(n)，严格来说是O(8n)，解码程序为一个一个字符读入，然后遍历每个字符二进制的1和0，再解码。

总体空间复杂度：

编码：O(1)，只需要创建256个节点来存储字符出现次数和构建哈夫曼树。

解码：O(1)，只需大小固定的缓冲区和数字来临时存储读入的数据即可。

子函数时间复杂度：

char selectMode(void);——O(1)

void encode(void); ——O(nlogn)

void decode(void); ——O(n)

char\* openFile(void); ——O(1)

countsOfWords\* countFrequency(FILE\* doc,int\* numsOfWord); ——O(n)

Node\* createTree(countsOfWords\* words,int count); ——O(n)

int cmpfunc (const void \* a, const void \* b); ——O(1)

Node\* createNode(void); ——O(1)

char\*\* encoding(countsOfWords\* words,Node\* root); ——O(n)

void dfs(char\*\* words,Node\* cur,int i,char code[]);——O(n)

void outputFile(FILE\* in,FILE\* out,char\*\* words, char\* filePath);——O(n)

void getCode(FILE\* doc,int words[]);——O(1)

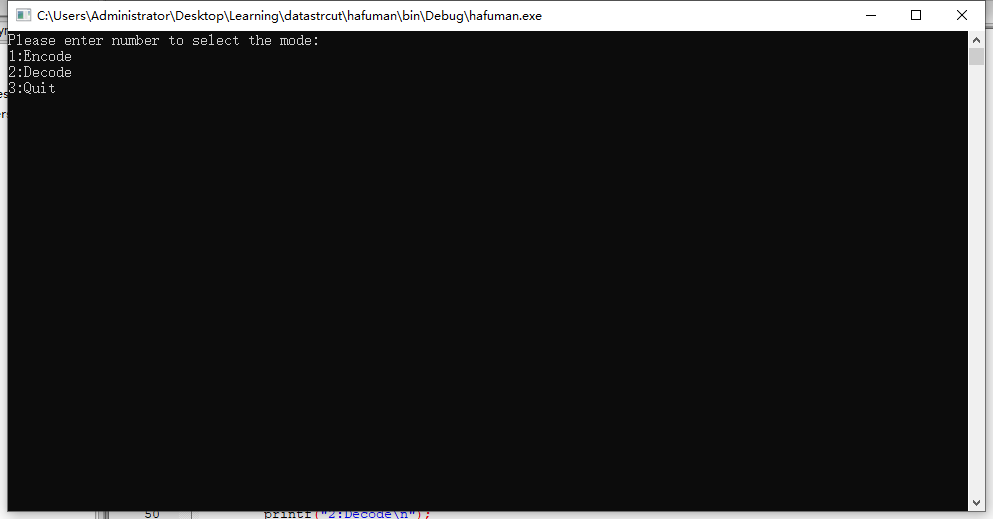
void outputDecodeFile(FILE\* doc,int words[],char\* fileName);——O(n)

**四：设计实验的回顾讨论**

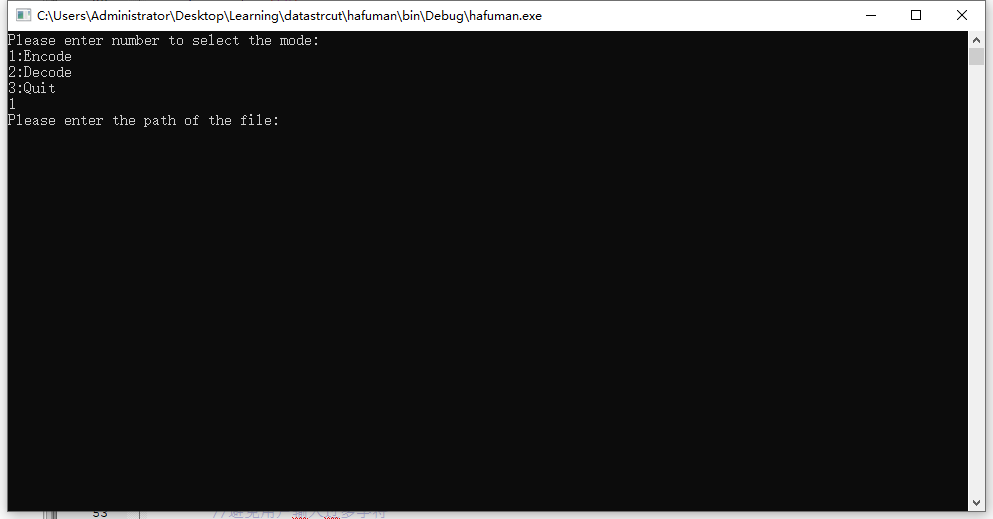
1. 在输入过程中需要记录文件里的单字节字符数量总数，然后需要输入编码的文件中用于后续编码时判断现在时候到达最后一个字符的编码处。
2. 在统计哈夫曼树的叶子节点数量时，需要统计字符出现的次数。
3. 在获得最小的两个字符的权值后进行循环建树，此过程其实就是建立静态分配空间以后的哈夫曼树的过程。
4. 在编码完毕后，重新按照文件的字符出现的顺序，找到它相对应的编码，以每8位完成一个编码并将其写入文件中，到达最后一个字符时，当它不足8位时，要补充编码。（在输出时要注意数据类型,有时不能够直接输出）。
5. 通过测试，我们发现本程序压缩效率较低，只有文件在重复度较高的时候能有较好的压缩效果，其他时候甚至有可能产生反效果。我们初步推测是因为我们编码时为了避免文件中只出现了一个字符的情况而在每个编码前面都加了一个‘1’，这导致了整个程序的压缩效果变差。
6. 我们将读入用户输入的scanf(“%s”)全部改成了scanf(“%c”)的循环，这样可以避免溢出和缓存区攻击。

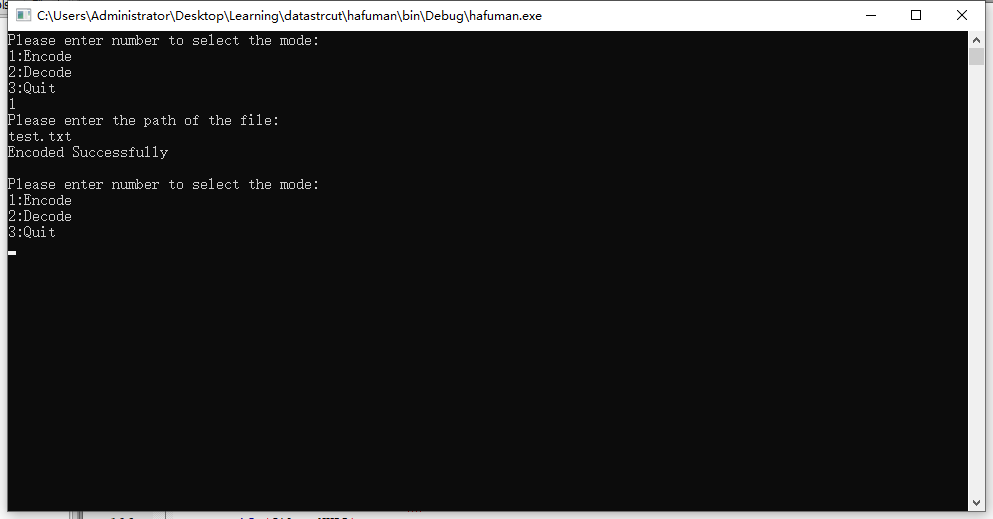
# 五：用户使用说明

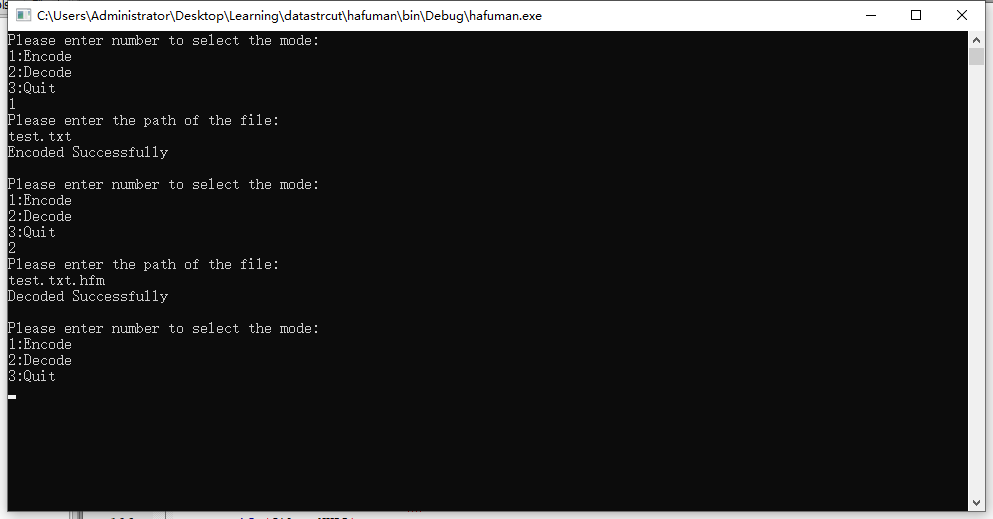
1. 打开程序，首先会出现如下界面：

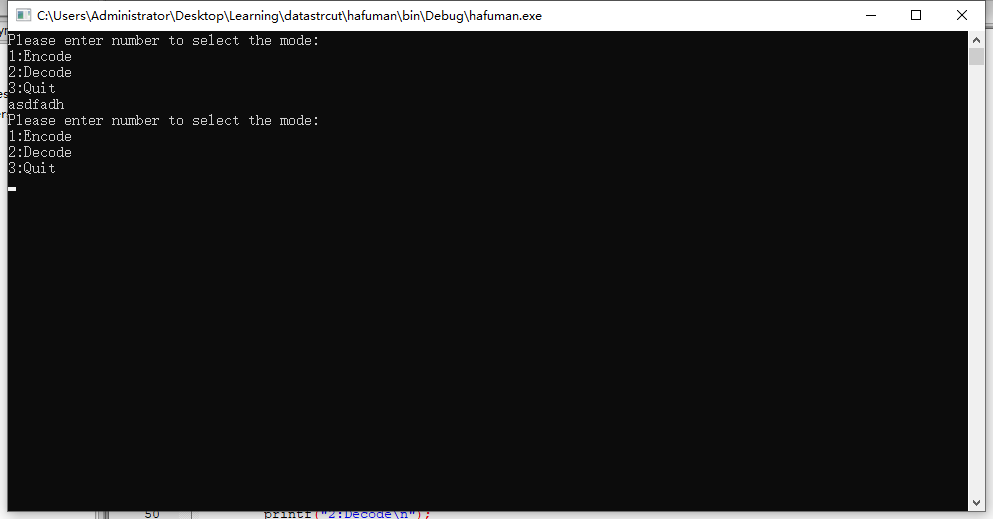


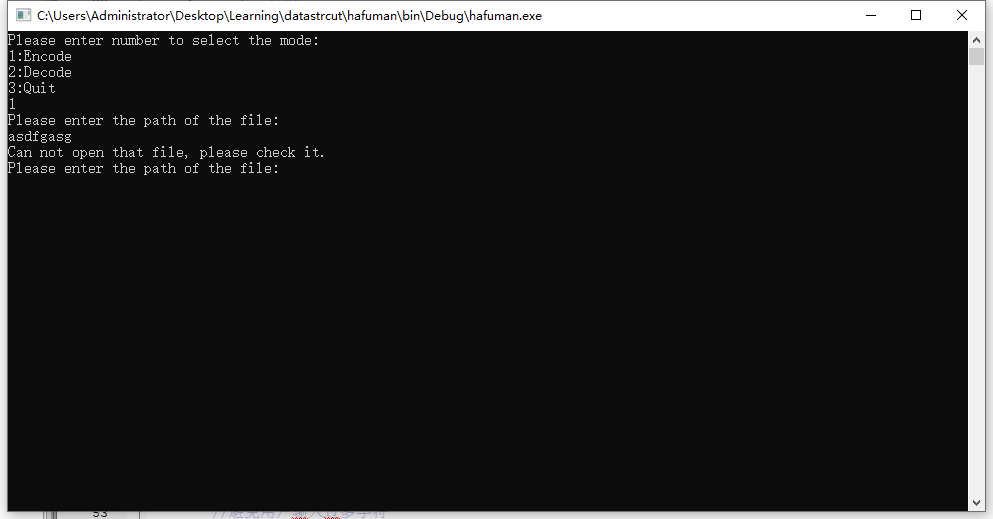
1. 用户输入合法时将会继续显示：







1. 当输入不合法时，会出现：



用户可根据提示，重新输入相应字符并回车即可。

**六：测试结果**

原文件：





压缩文件：



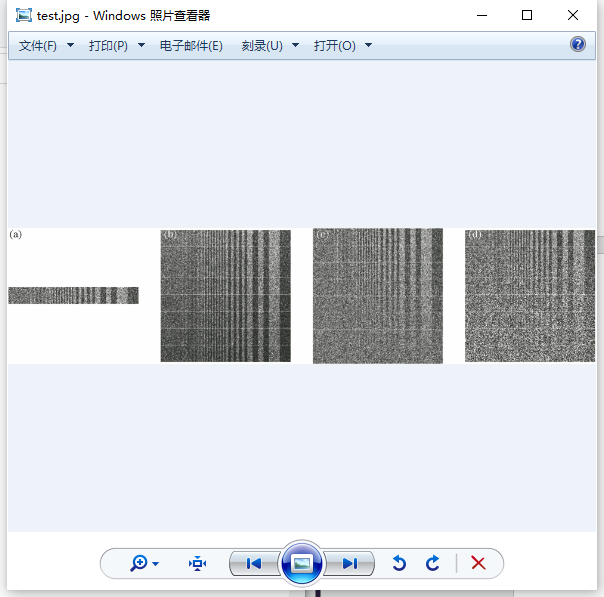


解压文件：



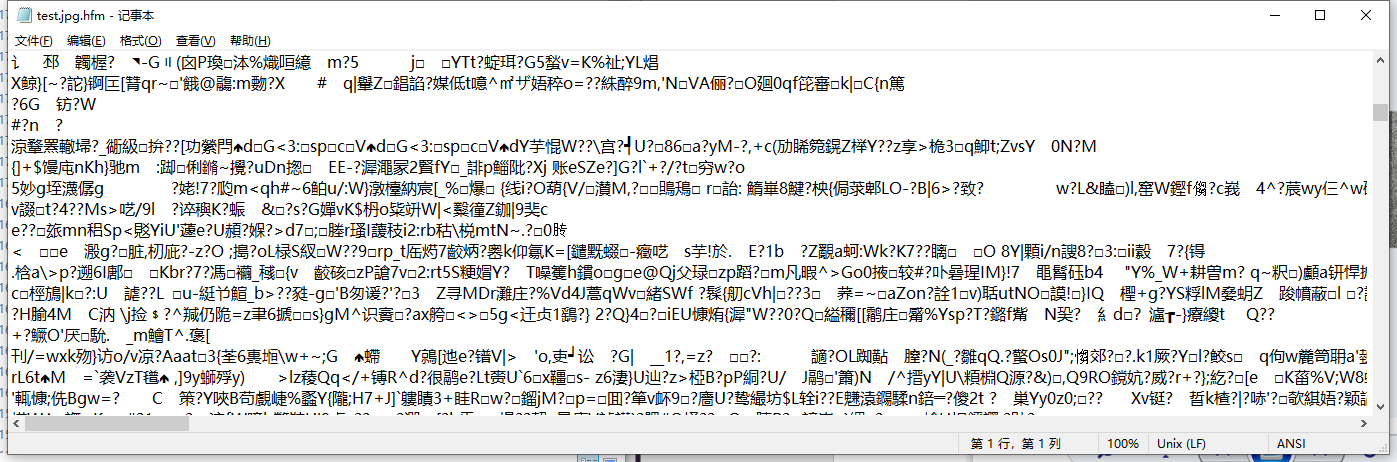


原文件：



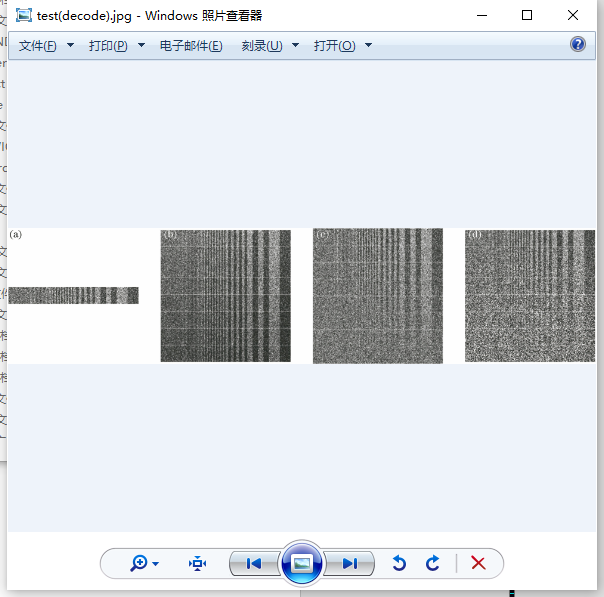


压缩文件：



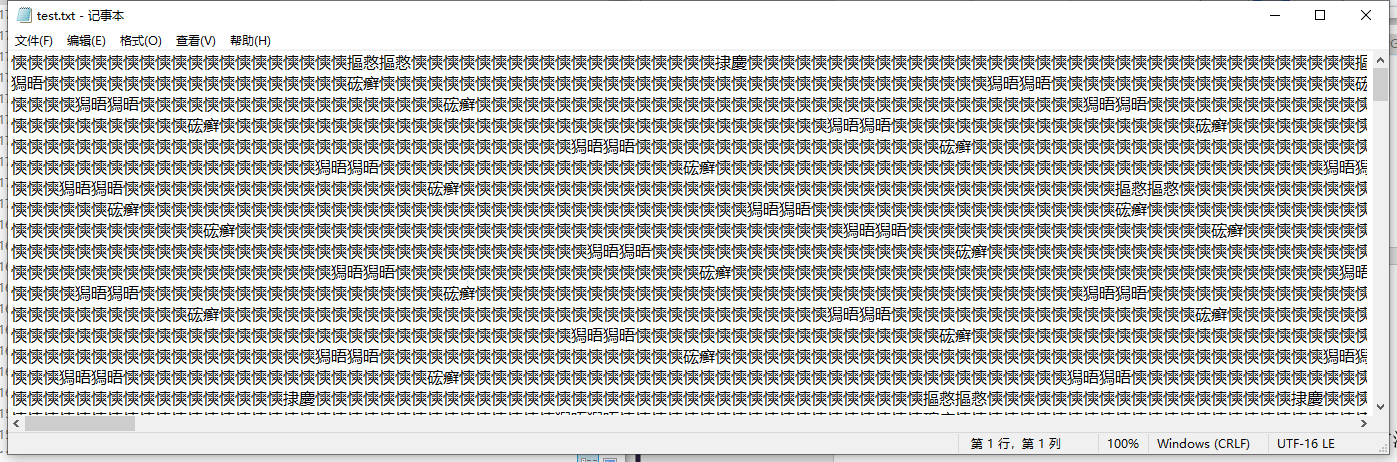


解压文件：



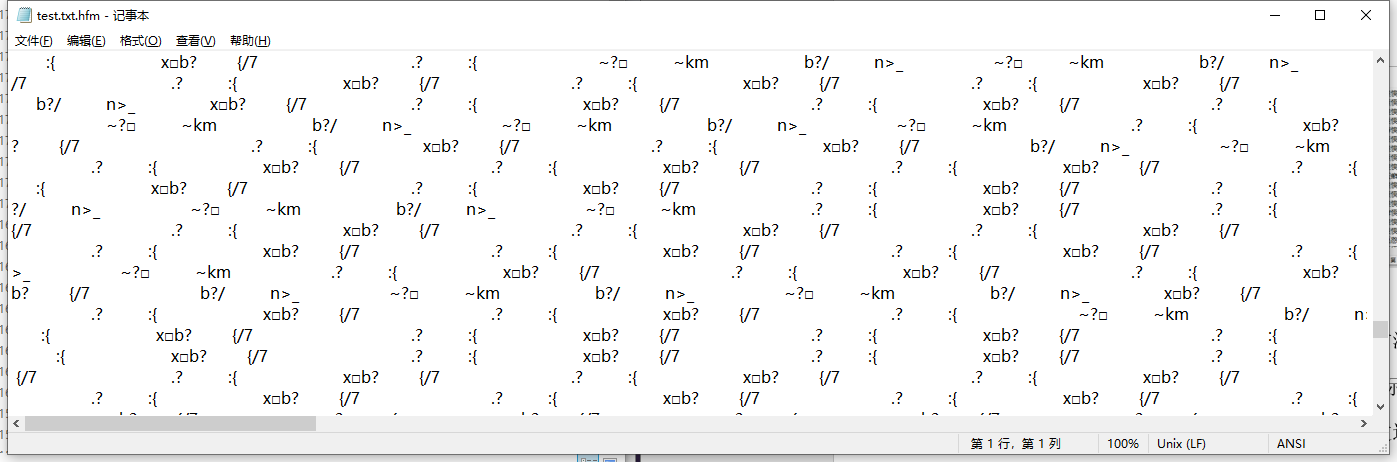


原文件：



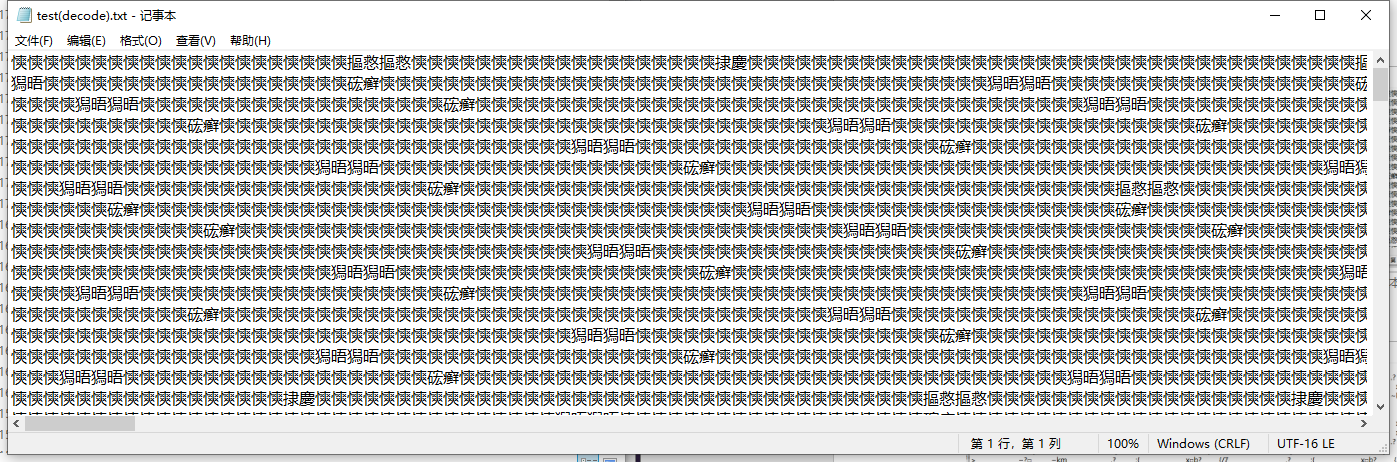


压缩文件：





解压文件：





**七：实验结果**

通过此次实验，加深了对创建哈夫曼树的方法和哈夫曼树的存储结构，明白了压缩文件的实现过程，解压文件实际上就是它的逆过程，这次试验使我们熟悉了哈夫曼编码的过程，通过这种方式可以实现使用有效的数据压缩技术来节省数据文件的存储空间和计算机网络的传送时间。更重要的是，在程序的编写过程中，一定要重视程序的模块化，要具体掌握各个模块之间的逻辑关系和整体程序的结构。在此次试验过程中，了解到了不断进行调试程序的重要性，只有这样，我们才能获得最后程序的成功运行结果。这次试验使我们成功将理论与现实问题相结合起来，为我们的日常生活提供了一种更加便利的方法，将所学习的知识运用到了生活中，做到了学以致用。