数据结构课程设计报告

设计题目：Huffman 编/译码器的设计与实现

班　　级：网络工程 161 班

学　　号：19316117

姓　　名：陈扬

南京农业大学计算机系

目录

Huffman 编/译码器的设计与实现

一、课程设计题目

设计一个哈夫曼编码、译码系统。对一个文本文件中的字符进行哈夫曼编码，生成编码文件；反过来，可将编码文件译码还原为一个文本文件。

(1) 读入一篇英文短文(文件扩展名为 txt)；

(2) 统计并输出不同字符在文章中出现的频率(空格、换行、标点等也按字符处理)；

(3）根据字符频率构建哈夫曼树，并给出每个字符的哈夫曼编码；

(4）输出哈夫曼树、哈夫曼编码；

(5）利用已建好的哈夫曼树，将文本文件进行编码，生成压缩文件(编码文件后缀名为.huf)；

(6）用哈夫曼编码存储的文件和输入文本文件大小进行比较，计算文件压缩率；

(7）进行译码，将 huf 文件译码为 txt 文件，与原 txt 文件进行比较。

测试数据：文本文件自行选择，至少含 3000 个字符。

二、算法设计思想

（一）Huffman 树

1. Huffman 树简介

（1）基本概念

路径：树中一个结点到另一个结点之间的分支构成这两个结点之间的路径。

路径长度：路径上的分支数目称作路径长度。

树的路径长度：从树根到每一个结点的路径长度之和。

结点的带权路径长度：在一棵树中，如果其结点上附带有一个权值，通常把该结点的路径长度与该结点上的权值的乘机称为节点的带权路径长度。

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，通常记作 $WPL=\sum_{k=1}^{n} w_k l_k$

（2）Huffman 树的概念

假设有 n 个权值$\{w_1, w_2, \ldots, w_n\}$，试构造一颗有 n 个叶子结点的二叉树，每个叶子结点带权为 w_i，则其中带权路径长度 WPL 最小的二叉树称做最优二叉树或 Huffman 树。

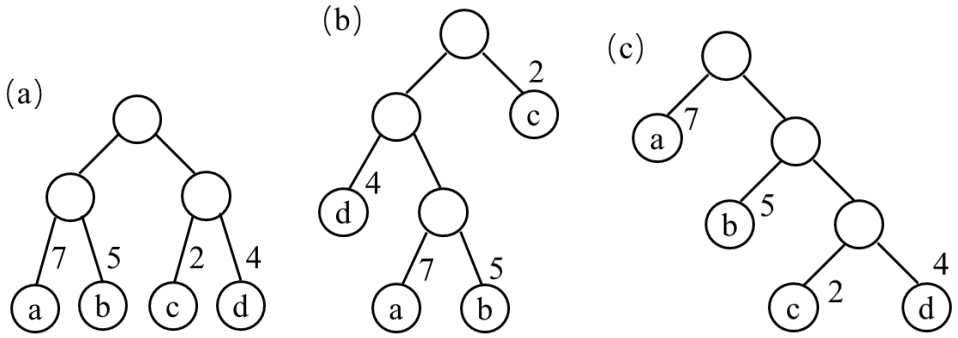
Huffman 树是一类带权路径最短的树，又称最优二叉树。

例如：（a)、(b)、(c）为 3 棵二叉树，都有 4 个叶子结点 a、b、c、d，分别带权 7、5、2、4，他们的带权路径长度分别为：

（a）WPL=7×2+5×2+2×2+4×2=36

（b）WPL=7×3+5×3+2×1+4×2=46

（c）WPL=7×1+5×2+2×3+4×3=35



2. Huffman 树的构造

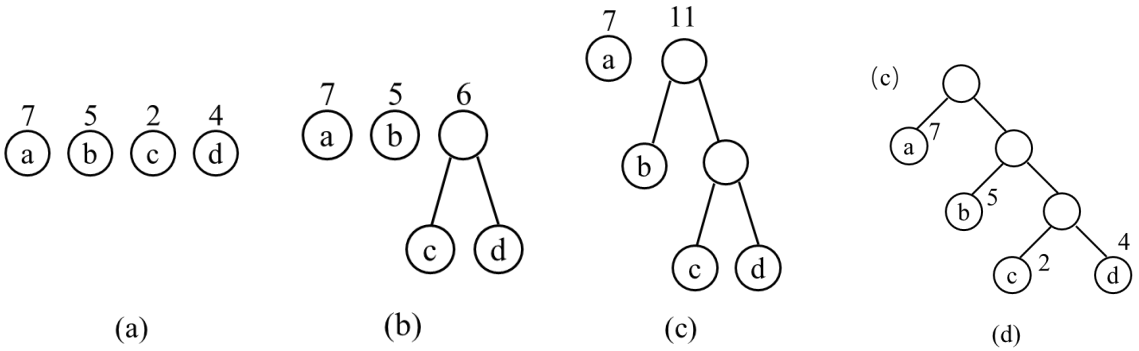
（1）根据给定的 n 个权值{w_1，w_2，…，w_n}构成 n 棵二叉树的集合 F={T_1，T_2，…，T_n}，其中每棵二叉树 T_i 中只有一个带权为 w_i 的根节点，其左右子树均空。

（2）在 F 中选取两棵根节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节点的权值为其左、右子树上根节点的权值之和。

（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 Huffman 树。

例如：图（a1）、（b1）、（c1）、（d1）展示了 Huffman 树（c）的构造过程。其中，根节点上标注的数字是所赋的权。



（二）Huffman 编码

1. Huffman 编码简介

目前，进行快速远距离通信的主要手段是电报，即将需传送的文字转换成由二进制的字符组成的字符串。在设计编码时需要遵守两个原则：

（1）发送方传输的二进制编码，到接收方解码后必须具有唯一性，即解码结果与发送方发送的电文完全一样；

（2）发送的二进制编码尽可能地短。下面介绍两种编码的方式：

a.等长编码

这种编码方式的特点是每个字符的编码长度相同（编码长度就是每个编码所含的二进制位数）。假设字符集只含有 4 个字符 A，B，C，D，用二进制两位表示的编码分别为 00，01，10，11。若现在有一段电文为：ABACCDA，则应发送二进制序列：00010010101100，总长度为 14 位。当接收方接收到这段电文后，将按两位一段进行译码。这种编码的特点是译码简单且具有唯一性，但编码长度并不是最短的。

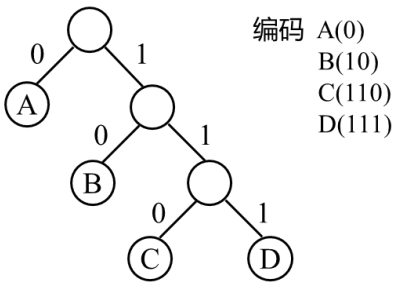
b.不等长编码

在传送电文时，为了使其二进制位数尽可能地少，可以将每个字符的编码设计为不等长的，使用频度较高的字符分配一个相对比较短的编码，使用频度较低的字符分配一个比较长的编码。例如，可以为 A，B，C，D 四个字符分别分配 0，00，1，01，并可将上述电文用二进制序列：000011010 发送，其长度只有 9 个二进制位，但随之带来了一个问题，接收方接到这段电文后无法进行译码，因为无法断定前面 4 个 0 是 4 个 A，1 个 B、2 个 A，还是 2 个 B，即译码不唯一，因此这种编码方法不可使用。

因此，若要设计长短不等的编码，则必须是任一个字符的编码都不是另一个字符的编码的前缀，这种编码称做**前缀编码。**

2. Huffman 编码的实现

可以利用二叉树来设计二进制的前缀编码。假设有一棵如图（2a）所示的二叉树，其 4 个叶子结点分别表示 A、B、C、D 这 4 个字符，且约定左分支表示字符'0'，右分支表示字符'1'，则可以从根节点到叶子结点的路径上分支字符组成的字符串作为该叶子结点字符的编码。可以证明，如此得到的必为二进制前缀编码。如图（2a）所得 A、B、C、D 的二进制前缀编码分别为 0、10、110 和 111。



（2a）

假设每种字符在电文中出现的次数为 w_i，其编码长度为 l_i，电文中只有 n 种字符，则电文总长为$\sum_{i=1}^{n} w_i l_i$。对应到二叉树上，若置 w_i 为叶子结点的权，l_i 恰为从根到叶子的路径长度。则$\sum_{i=1}^{n} w_i l_i$恰为二叉树上带权路径长度。由此可见，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，设计一棵 Huffman 树的问题，由此得到的二进制前缀编码便称为 Huffman 编码。由于 Huffman 树中没有度为 1 的结点，则一棵有 n 个叶子结点的 Huffman 树共有 $2n-1$ 个结点，可以存储在一个大小为 $2n-1$ 的一维数组中。由于在构成 Huffman 树之后，为求编码需从叶子结点出发走一条从叶子到根的路径；而为译码需从根出发走一条从根到叶子的路径。则对每个结点而言，既需知双亲的信息，又需知孩子结点的信息。由此，设定下述存储结构：

```
typedef struct {
    unsigned int weight;
    unsigned int parent, lchild, rchild;
}HTNode,*HuffmanTree;   //动态分配数组存储Huffman树
typedef char **HuffmanCode;   //动态分配数组存储 Huffman 编码表
```

注：具体实现方法与程序代码见（三、程序结构）和（六、源代码）和（附件 huffmancode.cpp）

（三）Huffman 译码

1. Huffman 译码简介

哈夫曼树译码是指由给定的代码求出代码所表示的结点值，它是哈夫曼树编码的逆过程。

从文本中，依次读入原先编码的 Huffman 编码，再利用已经存储的 Huffman 树结构，根据编码信息往下进行搜索，则可得该 Huffman 编码所对应的字符。

2. Huffman 译码的实现

Huffman 译码的基本思想是：从根结点出发，逐个读入电文中的二进制代码；若代码为 0 则走向左孩子，否则走向右孩子；一旦到达叶子结点，便可译出代码所对应的字符。然后又重新从根结点开始继续译码，直到二进制电文结束。

操作步骤如下：

（1）读入一个字符，判断是'1'还是'0'

（2）从根节点出发，出发走到该字符所对应的分支

（3）若该分支节点没有左孩子和右孩子，则输出该节点所对应的字符；若有，则继续执行步骤（2）直至找到

（4）若已经译出一个字符，则将数组清空，重新开始步骤（1）

```
while (!feof(encodefile)) {
  c = fgetc(encodefile);
  if (word == text_length)
     break;
   if (c == '0')
     q = HT[q].lchid;
  else if (c == '1')
        q = HT[q].rchild;
     if (HT[q].lchid == 0 && HT[q].rchild == 0) {
      fputc(HT[q].ASCII_CODE, decodefile);
      q = 2 * char_number - 1;
       word++;
       }
 }
```

（四）huf 文件编码算法

1. 对二进制文件的写入

在 Windows 操作系统中，二进制以 8 位（1 字节）为单位存储，8 位二进制无符号数的表示范围为 2^0-1~2^7-1，即 0~255。而 0~255 能够恰好对应 ASCII 码对照表中的所有 256 个字符，因此，在压缩存储 Huffman 编码时，每一次可以取 8 位 0/1 编码，将其对应到一个字节的每一个 bit 位上，再将这个字节转换成 ASCII 码值，在文件中写入其对应的字符。

2. Huffman 编码的实现

在进行 Huffman 编码压缩时，每一次取 8 位字符'0'或'1'， 将其对应到一个字节的每一个 bit 位上，再将这个字节转换成 ASCII 码值，在文件中写入其对应的字符。

例如：

假设之前已由 Huffman 编码得到 A=0，B=10，C=110，D=111。则字符串

“ABCD……”所对应的 Huffman 编码为 010110111……。第一次取前 8 位 01011011，将其对应写入 1 字节的 8 个 bit 位上，即 n=(01011011)2=(91)10，ASCII 码 91 所对应的字符为’[’，则将字符’[’以二进制形式写入文件。以此类推，可将全部的 Huffman 编码全部写入。生成的 huf 文件大小将变为原来 Huffman 编码文件的 1/8。

在进行二进制转换时，需要考虑 Huffman 编码总个数不是 8 的整数倍的情况。所以，在第一次写入前，需要提前计算 Huffman 编码的总个数，判断其是否为 8 的整数倍。若是，则不用修改；若不是，则需要在 Huffman 编码的最后补 1（或补 0，本程序采用补 1）使其凑满 8 的整数倍。假设 n 为 Huffman 编码总长度，m 为需要补 1 的个数，则有 m=8-n%8。

为了保证 huf 文件能够准确译出，所以在 huf 文件的二进制编码中，利用其第一个字节（前 8 位）来存储补 1 的个数。

在进行转换时，需要用到&、|、~、^、<<、>>运算，实现对字节的位操作。

```
char bit_number_transform(int n[]) {
    char num ;
    int i;
    for (i = 0; i < 8; i++) {
        if (n[i] == 1)
            num |= (1 << (7 - i));
        else
            num &= ~(1 << (7 - i));
    }
    return num;
}
```

利用以上代码，可以实现字符’0’’1’的转化。

（五）huf 文件译码算法

1. 对二进制文件的读取

在 C 语言中，对文件的读取操作以 8 位（1 字节）为最小单位。对应“huf 文件编码算法”中的写入操作，每一位字符的 ASCII 值一定在 2^0-1~2^7-1，即 0~255 中。所以，每一个字符能够翻译成为 8 位’0’’1’字符。

2. 对 Huffman 编码的解压缩

在进行 Huffman 编码解压缩时，每一次取一个字节的字符，先将其转换成对应的 ASCII 码，再将 ASCII 码的 8 位二进制位’0’/’1’依次输出即可得到原来的 Huffman 编码。

例如：

假设取得一个字符’A’，则程序先将其转化成对应的 ASCII 码 65，$(65)_{10}=(0100\ 0001)_2$，则程序将 $(0100\ 0001)_2$ 依次写入文件中，得到译码文件。

在进行 huf 文件译码时，还需考虑到翻译 huf 文件的第一位字符，得出该文件的补 1 个数。设 huf 文件的大小为 m 字节，翻译的到的补 1 个数为 p，则有原 Huffman 编码长度$n = 8 \times (m - 1) - p$

为了保证 huf 文件译码的准确性，需要通过控制打印字符个数来控制 Huffman 编码的长度。

在进行转换时，需要用到&、|、~、^、<<、>>运算，实现对字节的位操作。

```
while (!feof(huf)) {
```

```
    c = fgetc(huf);
        for (t = 0; t < 8; t++) {
       if (((c >> (7 - t)) & 1) == 1) {
              count++;
            fputc(49, unzip);
            if (count == (8 * (huflength - 1) - code_mod))
            break;
       }
       else if (((c >> (7 - t)) & 1) == 0) {
              count++;
              fputc(48, unzip);
              if (count == (8 * (huflength - 1) - code_mod))
               break;
        }
       }
       if (count == (8 * (huflength - 1) - code_mod))
        break;
}
```

三、程序结构

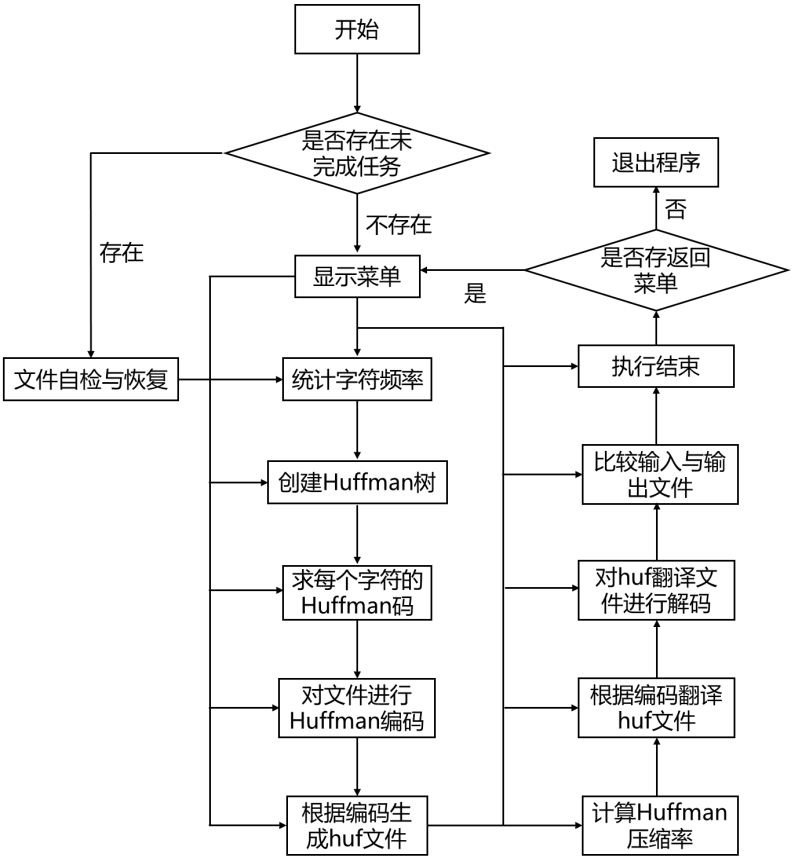
（一）程序执行结构

1. 程序实现功能

①显示菜单；②统计字符频率；③创建 Huffman 树；④求每个字符的 Huffman 码；⑤对文件进行 Huffman 编码；⑥根据编码生成 huf 文件；⑦计算 Huffman 压缩率；⑧根据编码翻译 huf 文件；⑨对 huf 翻译文件进行解码；⑩比较输入文件与输出文件；⑾文件自检与恢复；⑿读入未完成文件；

注：其他功能及函数请见下一项介绍

2. 程序执行流程



3. 程序执行逻辑

（1）程序启动，首先检测是否有上一次未完成的任务
如果有，则询问是否继续，跳转至第（4）步开始生成 Huffman 树；如果没有，则跳转至第（2）步显示开始菜单。

（2）显示菜单，询问要执行哪一步操作。当输入相应数字后，程序直接跳转。

（3）统计字符频率。结束后询问是否继续下一步
如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（4）创建 Huffman 树。结束后询问是否继续下一步
如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（5）求每个字符的 Huffman 码。结束后询问是否继续下一步
如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（6）对文件进行 Huffman 编码。结束后询问是否继续下一步
如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（7）根据编码生成 huf 文件。结束后询问是否继续下一步
如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（8）计算 Huffman 压缩率。结束后询问是否继续下一步
如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（9）根据编码翻译 huf 文件。结束后询问是否继续下一步
如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（10）对 huf 翻译文件进行解码。结束后询问是否继续下一步
如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（11）比较输入文件与输出文件。结束后询问是否继续下一步

如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（12）文件自检与恢复。结束后询问是否继续下一步

如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（13）读入未完成文件。结束后询问是否继续下一步

如果选择是，则继续下一步；如果不是，则询问是否返回菜单

（14）程序执行结束，询问是否返回菜单。

如果选择是，则返回菜单；如果不是，则退出程序

（二）函数功能

1. typedef struct {
    int char_ASCII;
    int value;
    char *Huffmancode;
    int Huffmancode_bit = 0;
}ElemType;

函数说明：链表中的元素结构体，char_ASCII 为字符所对应的 ASCII 值；value 为字符的出现频率或权值；Huffmancode 为 Huffman 编码所在数组的指针；Huffmancode bit 为 Huffmancode 数组的长度。

功能：定义链表中的元素结构体

使用方法：预定义，无需手动调用

2. typedef struct node {
    ElemType elem;
    struct node *next;
}LinkNode, *LinkList;

函数说明：链表结构体。elem 为元素，*next 为下一链表的地址。

功能：定义链表结构体。

使用方法：预定义，无需手动调用

3. Status List_Init(LinkList &L)

函数说明：链表的初始化操作

功能：创建一个可用的链表并将其初始化

使用方法：输入 LinkList 类型的链表 L

4. Status List_Insert(LinkList &L, int i, ElemType e)

函数说明：链表的插入操作

功能：将一个元素插入到链表的指定位置

使用方法：输入 LinkList 类型的链表 L，插入位置 i，要插入的 ElemType 类型的元素 e

5. Status List_Destroy(LinkList &L)

函数说明：删除链表，将整个链表空间摧毁

功能：删除给定的链表 L

使用方法：输入一个 LinkList 类型的 L 变量，摧毁 L

6. typedef struct {
   unsigned int value, ASCII_CODE;

```
    unsigned int parent, lchid, rchild;
}HTNode, *HuffmanTree;
```

函数说明：Huffman 树结构体定义，value 为叶子结点的权值；ASCII_CODE 为叶子结点所对应的字符的 ASCII 码；parent、lchild、rchild 为双亲、左孩子、右孩子

功能：定义 Huffman 树的结构体

使用方法：由其他函数调用，无需手动调用

7. Status File_sourceload()

功能：读取源文件信息，统计字符出现频率

8. Status File_read_char_num(LinkList &L, int &text_length, int &char_number)

功能：读取 huffman_temp\\char_frequency.txt 中的信息

9. Status HuffmanTree_Select(HuffmanTree HT, int n, int &s1, int &s2)

功能：选择权值最小的两个结点

10. Status HuffmanTree_Create(HuffmanTree &HT, int n, LinkList L,int if_print)

功能：创建 Huffman 树

11. Status HuffmanTree_Code(HuffmanTree HT, int n, LinkList &L)

功能：从叶子到根逆向求每个字符的 Huffman 编码，储存在指针 L.Huffmancode 中

12. Status HuffmanCode_Write(HuffmanTree HT, int n, LinkList &L)

功能：存储 Huffman 编码为字典

13. char * HuffmanCode_CharPoint(LinkList L, char c)

功能：返回某个字符的 Huffman 编码所在的指针

14. Status HuffmanCode_Encode(HuffmanTree HT, LinkList L,int if_print)

功能：对文件进行 Huffman 编码，存储在 huffman_encode.txt 中

15. char bit_number_transform(int n[])

功能：每 8 位转化为对应的 ASCII 码

16. Status text_huffmancode_create()

功能：生成 huf 文件

17. Status text_huffmancode_unzip()

功能：解压 huf 文件

18. Status HuffmanCode_FileDecode(HuffmanTree HT, int char_number, int text_length)

功能：对文件进行 Huffman 解码，存储在 text_decode.txt 中

19. Status File_Error_Percentage()

功能：对输入文件和输出文件进行比较，求正确率

20. Status File_Zip_Percentage()
功能：计算压缩率

21. void menu()
功能：显示菜单

22. int if_go_next()
功能：询问是否进行下一步操作

23. void if_backto_menu()
功能：询问是否返回菜单

24. void menu_function1()
功能：菜单功能 1，按流程执行所有操作

25. void menu_function2()
功能：菜单功能 2，统计字符频率

26. void menu_function3()
功能：菜单功能 3，创建 Huffman 树

27. void menu_function4()
功能：菜单功能 4，求每个字符的 Huffman 编码

28. void menu_function5()
功能：菜单功能 5，对文件进行 Huffman 编码

29. void menu_function6()
功能：菜单功能 6，根据编码生成.huf 文件

30. void menu_function7()
功能：菜单功能 7，计算 Huffman 压缩率

31. void menu_function8()
功能：菜单功能 8，根据编码翻译.huf 文件

32. void menu_function9()
功能：菜单功能 9，对翻译得到的文件进行解码

33. void menu_function10()
功能：菜单功能 10，比较输入文件与输出文件

34. void menu_function11()
功能：菜单功能 11，退出程序

35. void Project_Recovery()
功能：任务恢复。重新读入未完成的 char_frequency.txt 文档，继续执行任务。

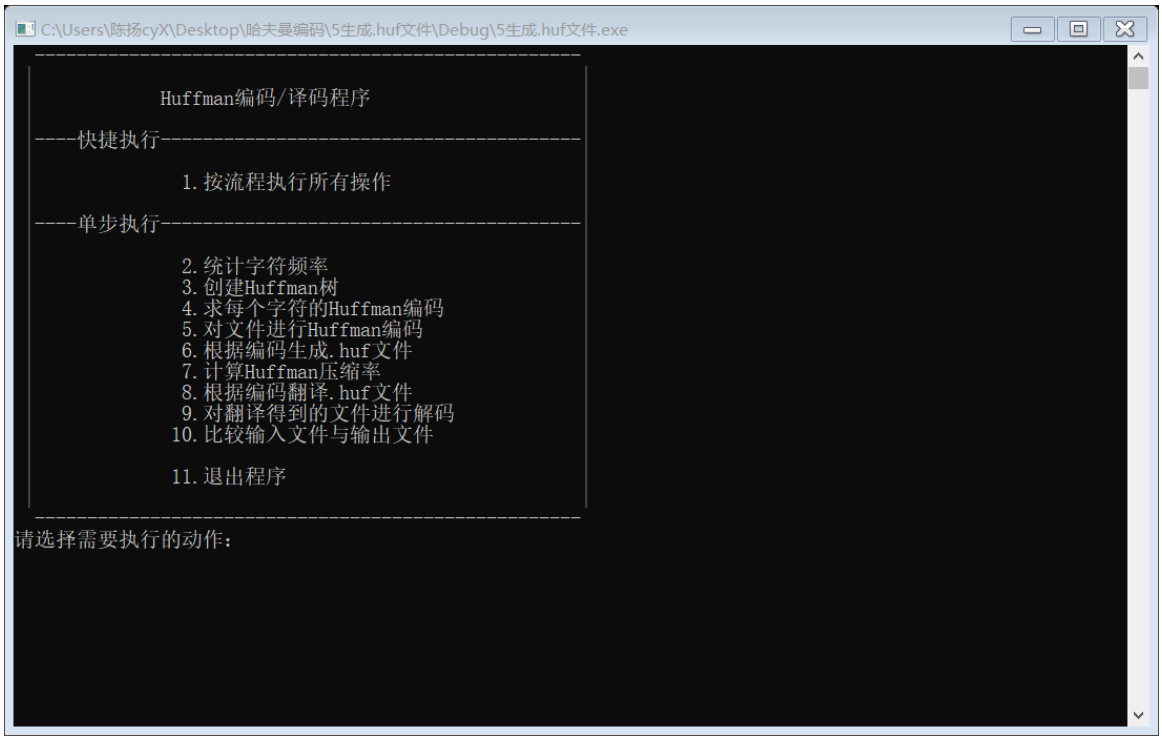
36. void main()
函数说明：main 函数
功能：判断程序目录下是否存在未完成的工程，若有则继续执行，若没有则跳转至菜单。
使用方法：程序入口，无需手动执行。

四、实验结果与分析

（一）程序执行结果

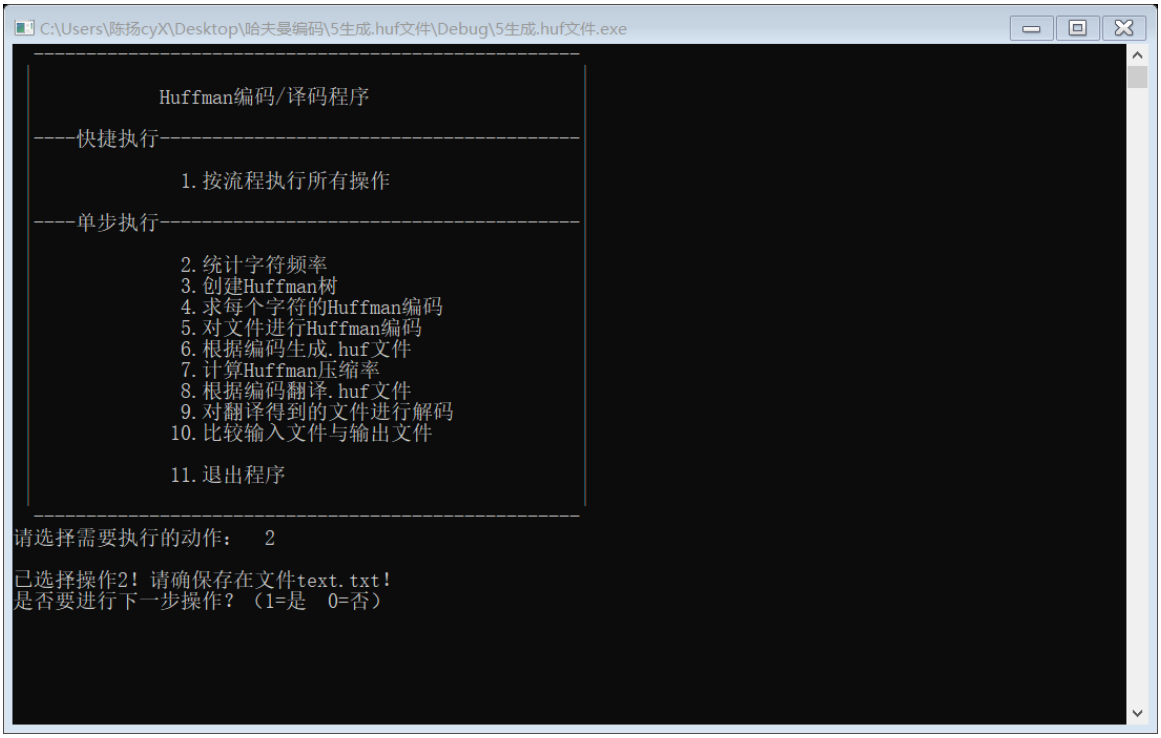
1. 进入菜单界面

进入程序，会自动打开程序的菜单界面。
程序的菜单界面总共提供 11 个功能：1.按流程执行所有操作 2.统计字符频率 3.创建 Huffman 树 4.求每个字符的 Huffman 编码 5.对文件进行 Huffman 编码 6.根据编码生成.huf 文件 7.计算 Huffman 压缩率 8.根据编码翻译.huf 文件 9.对翻译得到的文件进行解码 10.比较输入文件与输出文件 11.退出程序
程序会要求选择需要执行的动作，只需要输入相对应的数字执行即可。若输入的整数 n＜1 或 n＞11，则程序会要求重新输入，直至输入正确可读的数字为止。



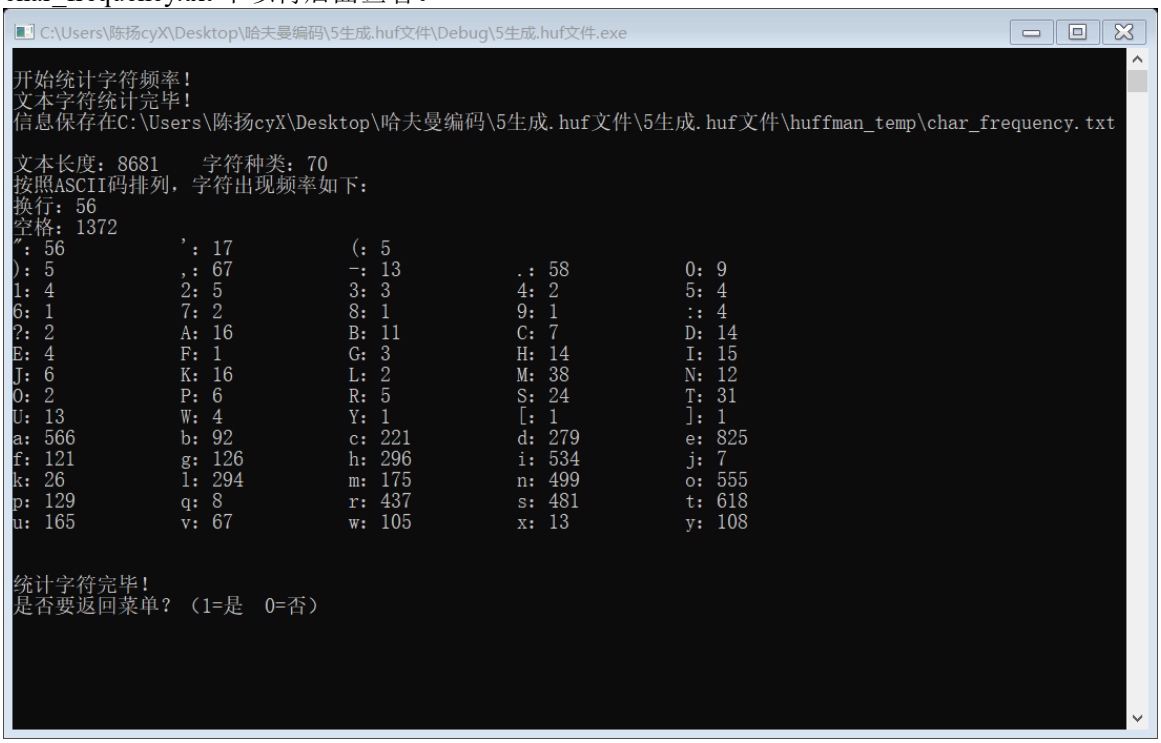
2. 统计字符频率

选择第 2 项“统计字符频率”，程序会自动弹出提示“已选择操作 2！请确保存在文件 text.txt！”，并要求确认是否进行下一步操作。若输入 1，则进行下一步，若输入 0，则程序将会询问是否返回菜单。



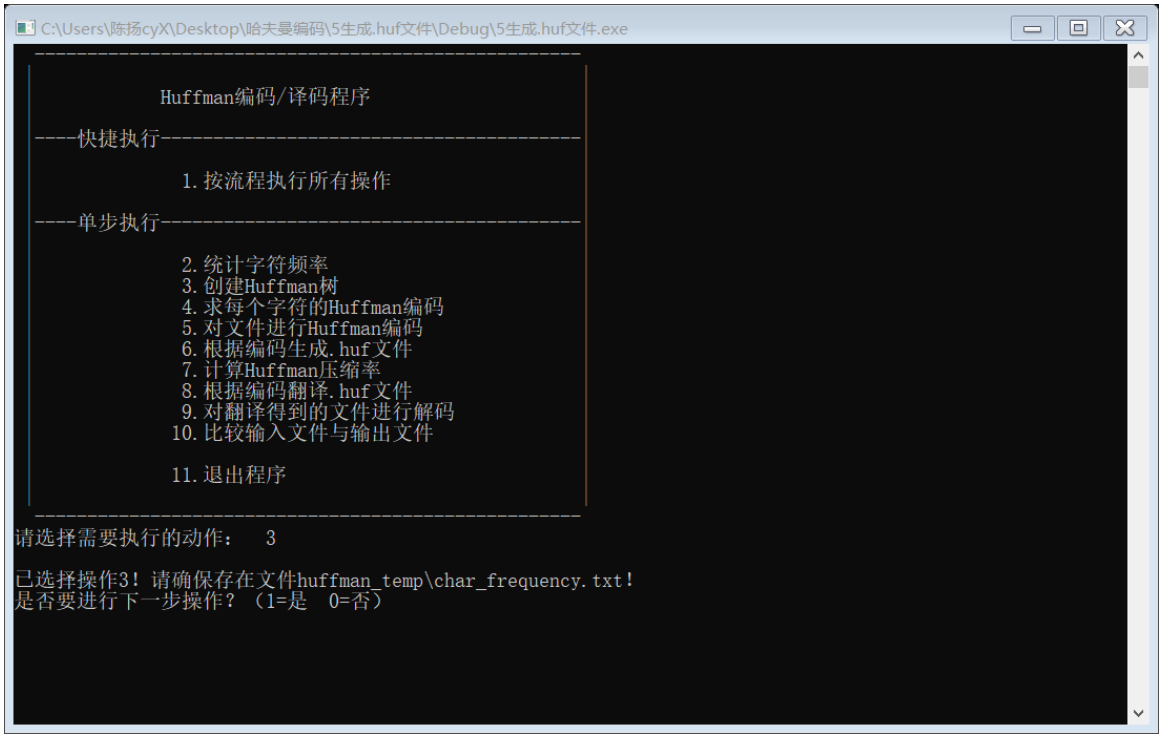
输入 1，确定进行下一步操作，则程序将会自动统计位于目录下的 text.txt 中的字符串长度、字符种类和每个字符的出现频率。

统计完成后，程序将会弹出相应提示，并将所有信息写入到文件目录下的 char_frequency.txt 中以待后面查看。

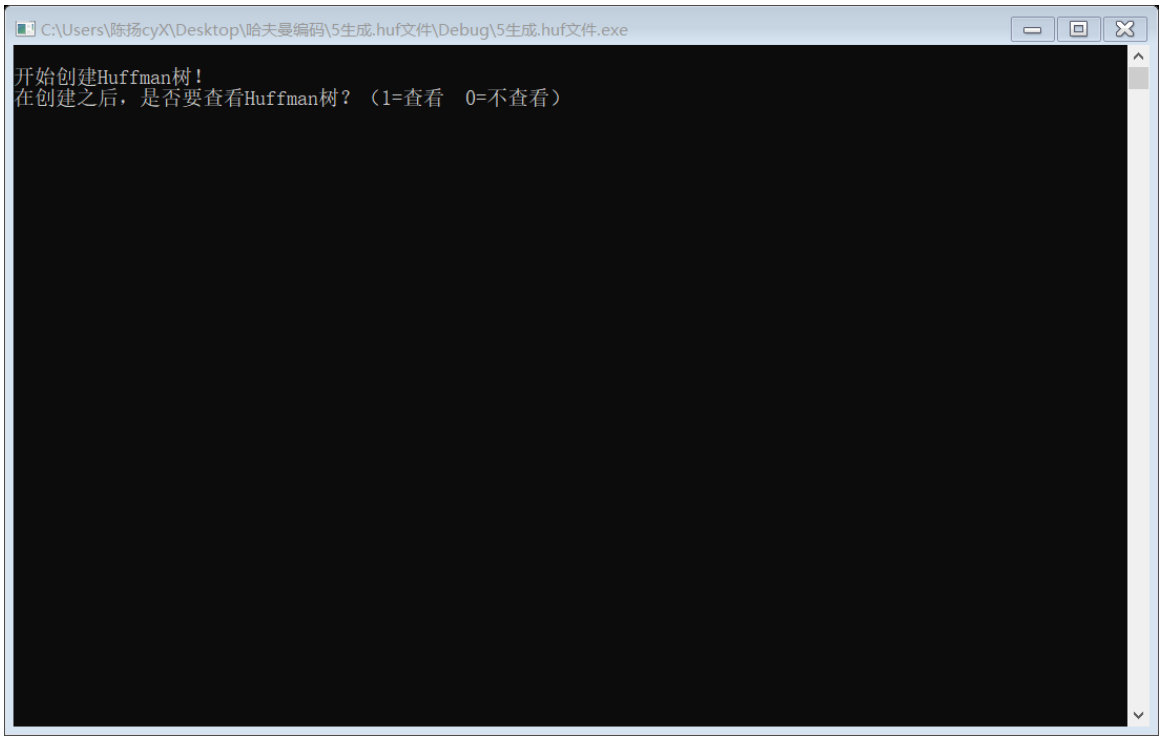


3. 创建 Huffman 树

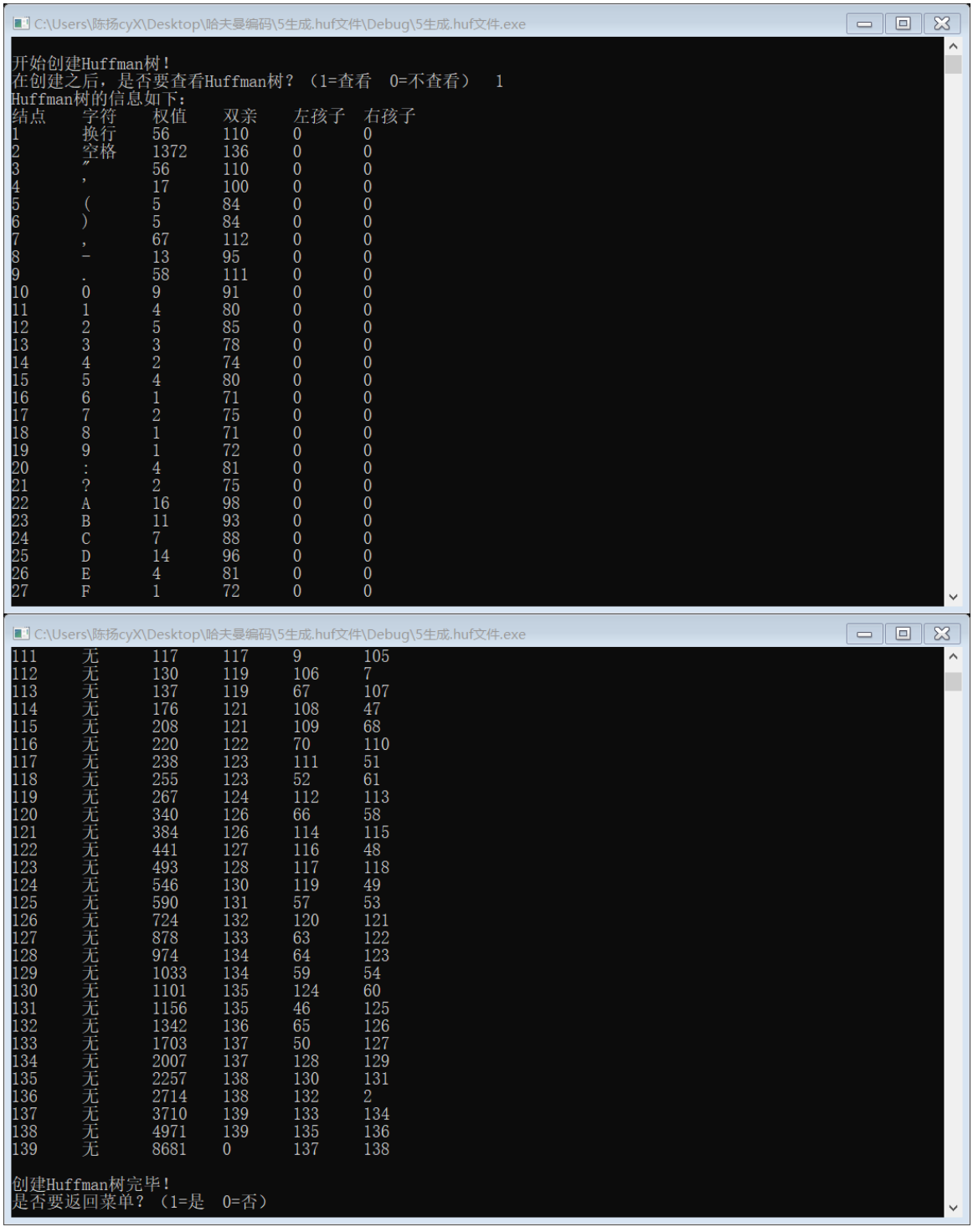
选择 3 项“创建 Huffman 树”，程序会自动弹出“已选择操作 3！请确保存在文件 huffman_temp\char_frequency.txt！”，并要求确认是否进行下一步操作。若输入 1，则进行下一步，若输入 0，则程序将会询问是否返回菜单。



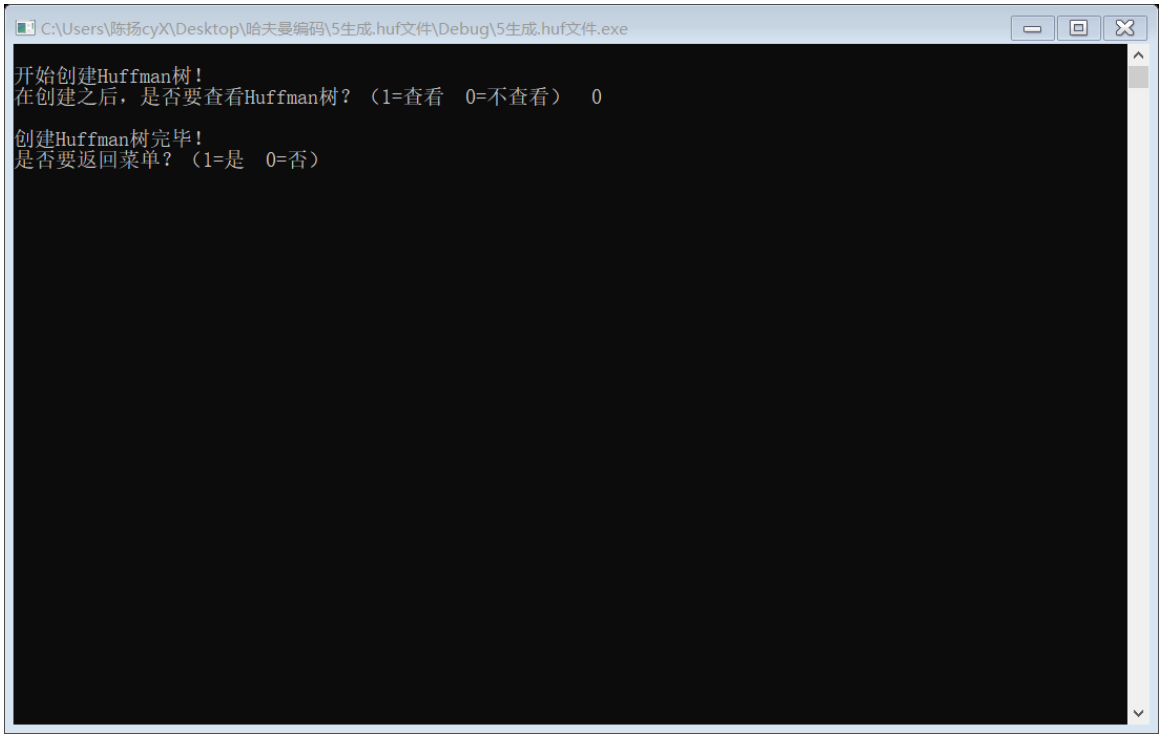
输入 1，确定进行下一步操作。
程序将会弹出提示，“开始创建 Huffman 树！”，并询问“在创建之后，是否要查看 Huffman 树？（1=查看 0=不查看）”。



输入 1，确认在生成后要查看 Huffman 树的内容。

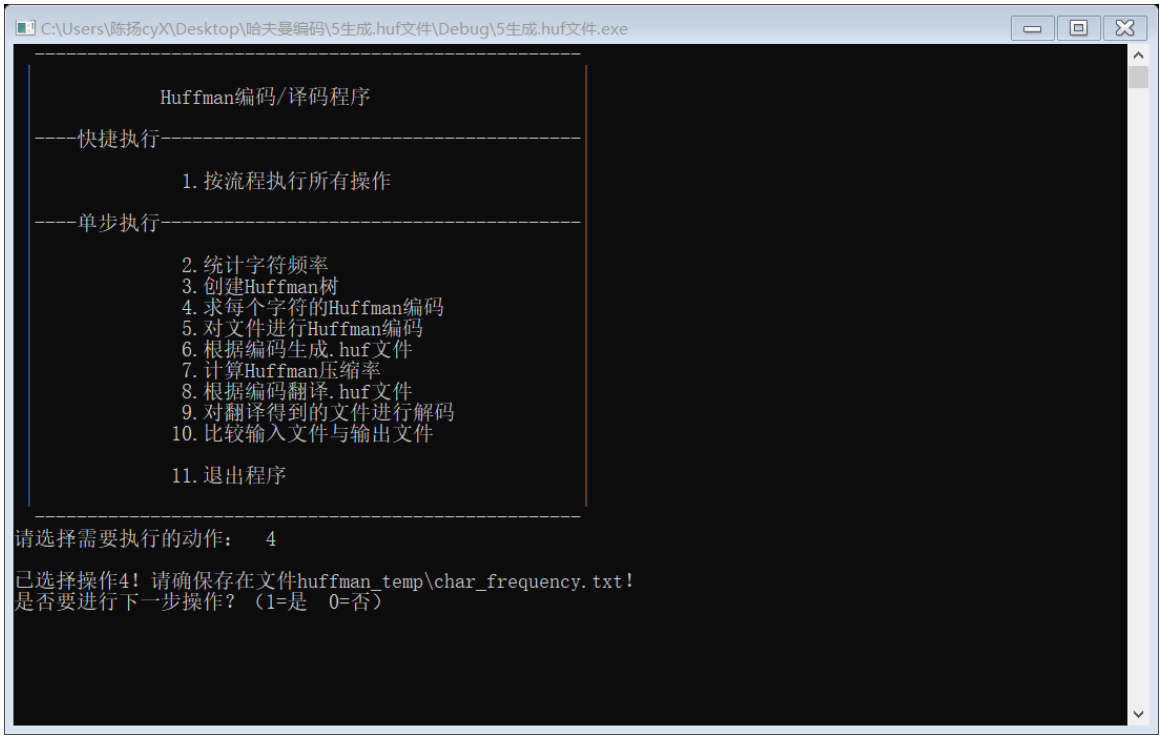


输入 0，确认在生成后不查看 Huffman 树的内容。
则程序默认在后台创建完成 Huffman 树，并不会将结果展示出来。



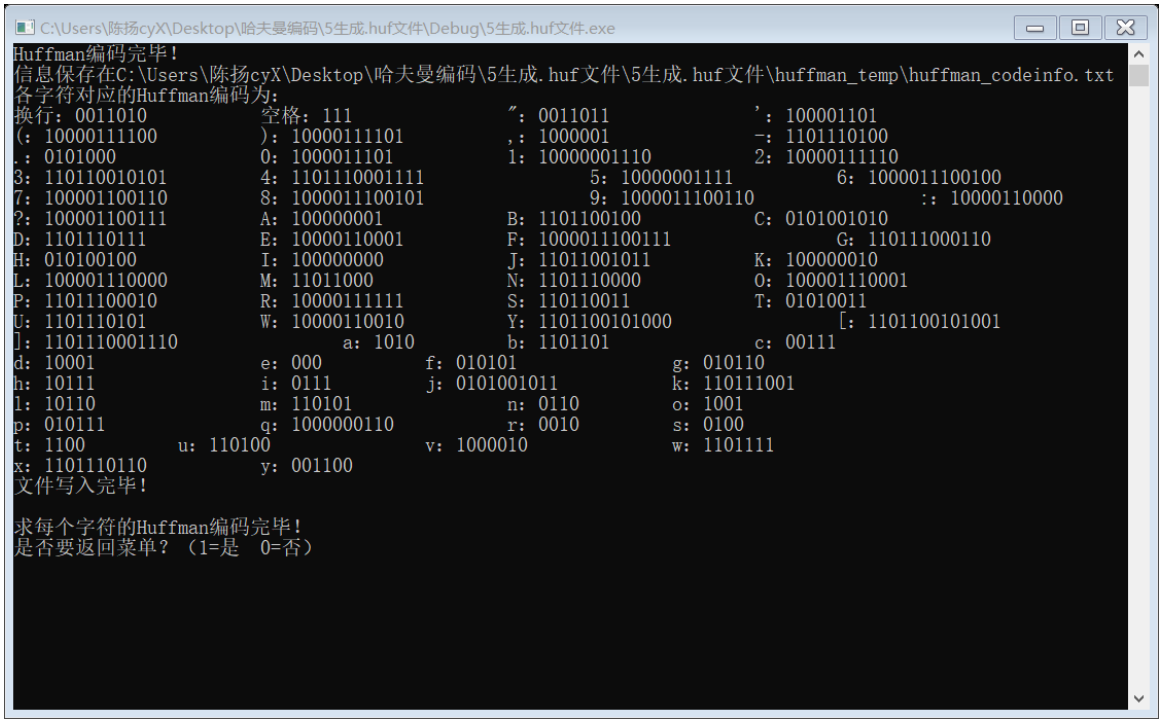
4. 求每个字符的 Huffman 编码

选择 4 项“求每个字符的 Huffman 编码”，程序会自动弹出“已选择操作 4！请确保存在文件 huffman_temp\char_frequency.txt！”，并要求确认是否进行下一步操作。若输入 1，则进行下一步，若输入 0，则程序将会询问是否返回菜单。



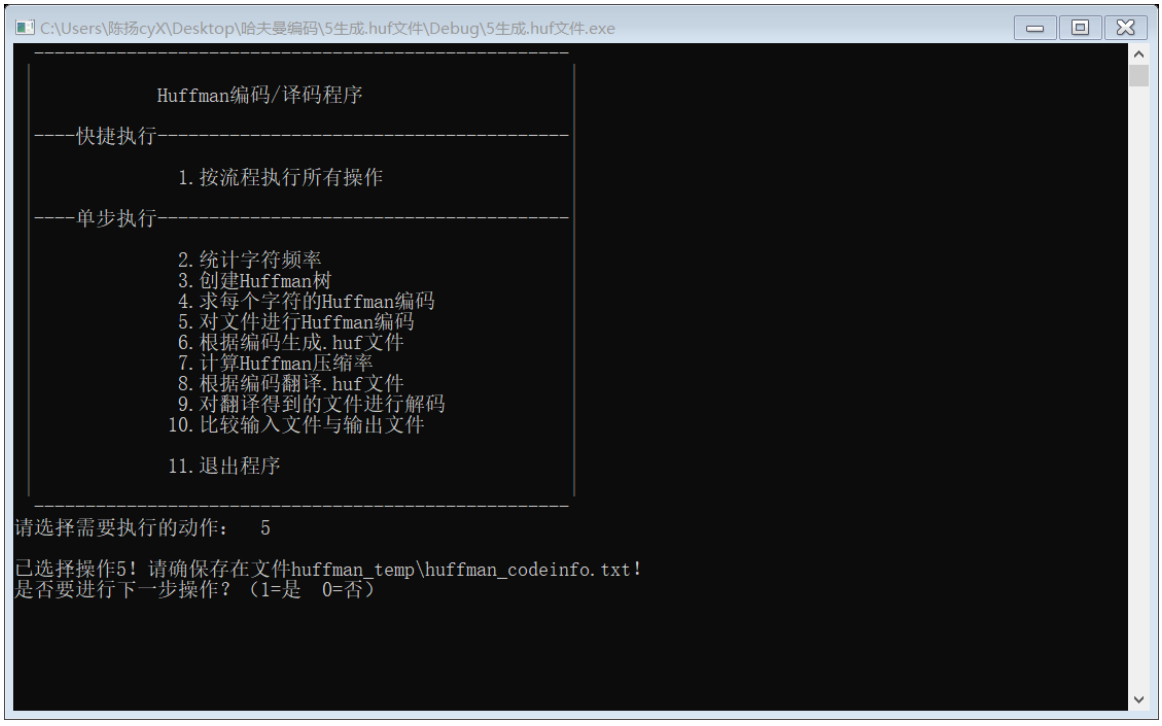
选择 1，确认进行下一步，则程序将会统计出 text.txt 文件中每个字符的 Huffman 编码，并在屏幕上显示。

同时，程序还会将该信息写入到文件 huffman_temp\huffman_codeinfo.txt 中，方便日后查看。

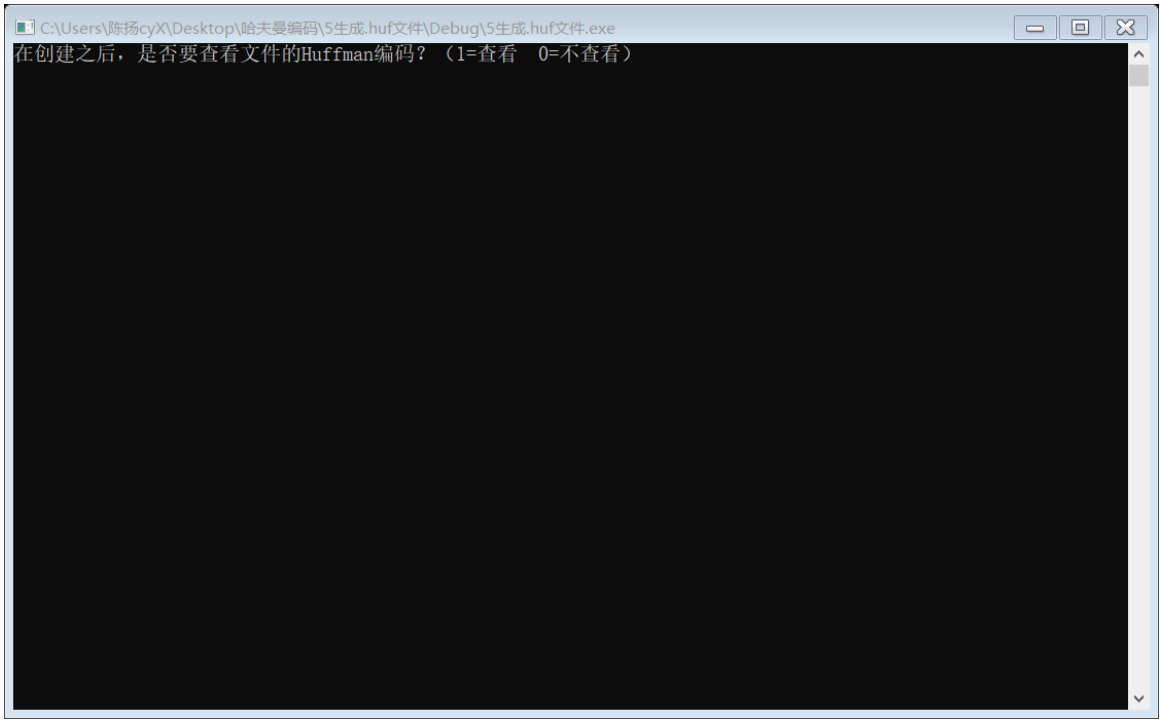


5. 对文件进行 Huffman 编码

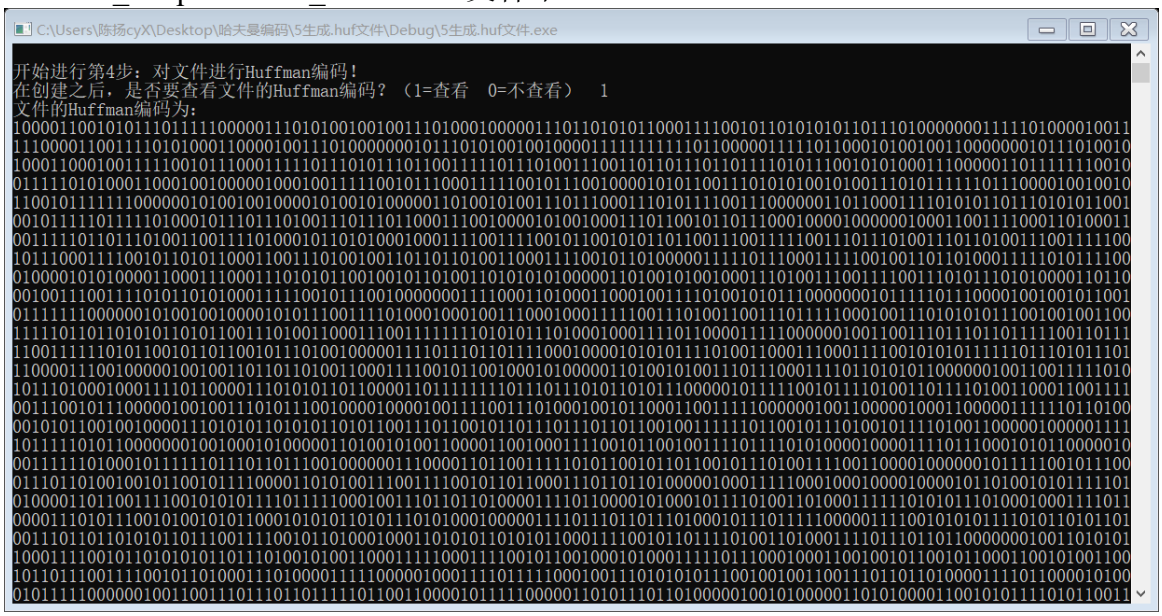
选择 5 项“对文件进行 Huffman 编码”，程序会自动弹出“已选择操作 5！请确保存在文件 huffman_temp\huffman_codeinfo.txt！”，并要求确认是否进行下一步操作。若输入 1，则进行下一步，若输入 0，则程序将会询问是否返回菜单。

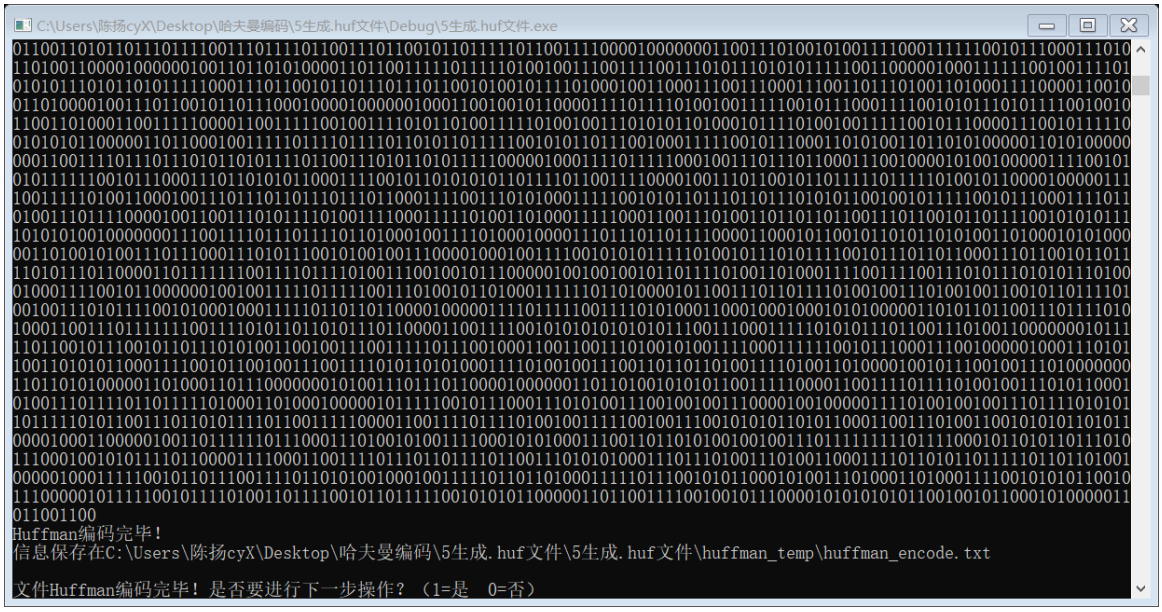


输入 1，确认进行下一步。程序还会询问“在创建之后，是否要查看文件的 Huffman 编码？（1=查看 0=不查看）”

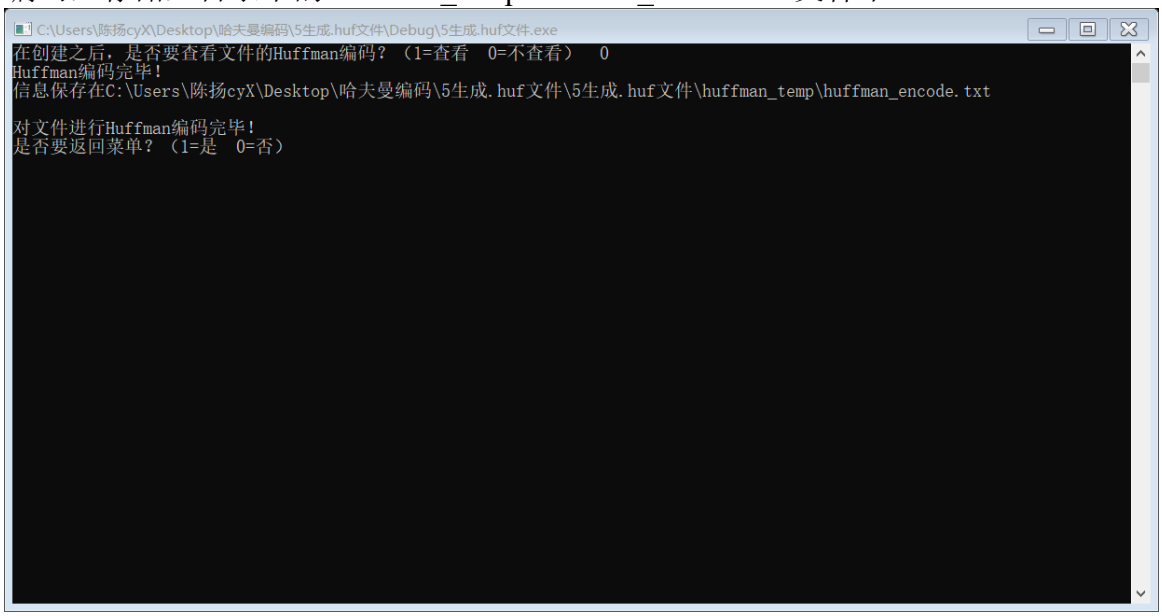


输入 1，确认要查看文件的 Huffman 编码，则程序将会自动将编码的全部 Huffman 编码显示出来。
同时，程序还会将该编码信息存储至目录下的 huffman_temp\huffman_encode.txt 文件中。



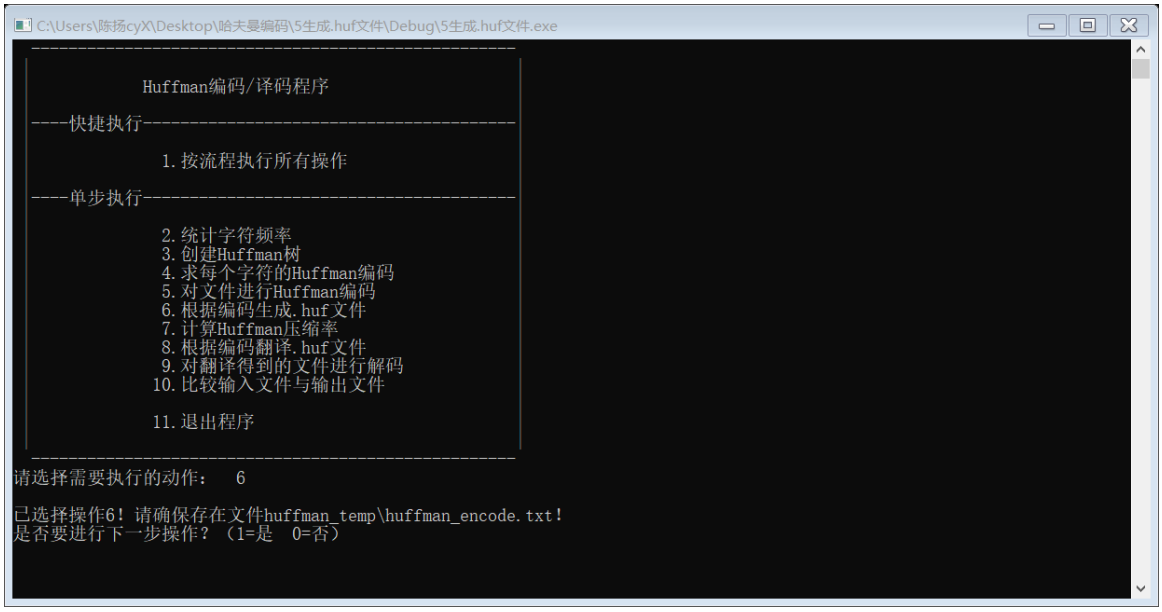


输入 0，确认在编码后不查看 Huffman 编码，则程序会在后台自动对文件进行编码，存储至目录下的 huffman_temp\huffman_encode.txt 文件中。

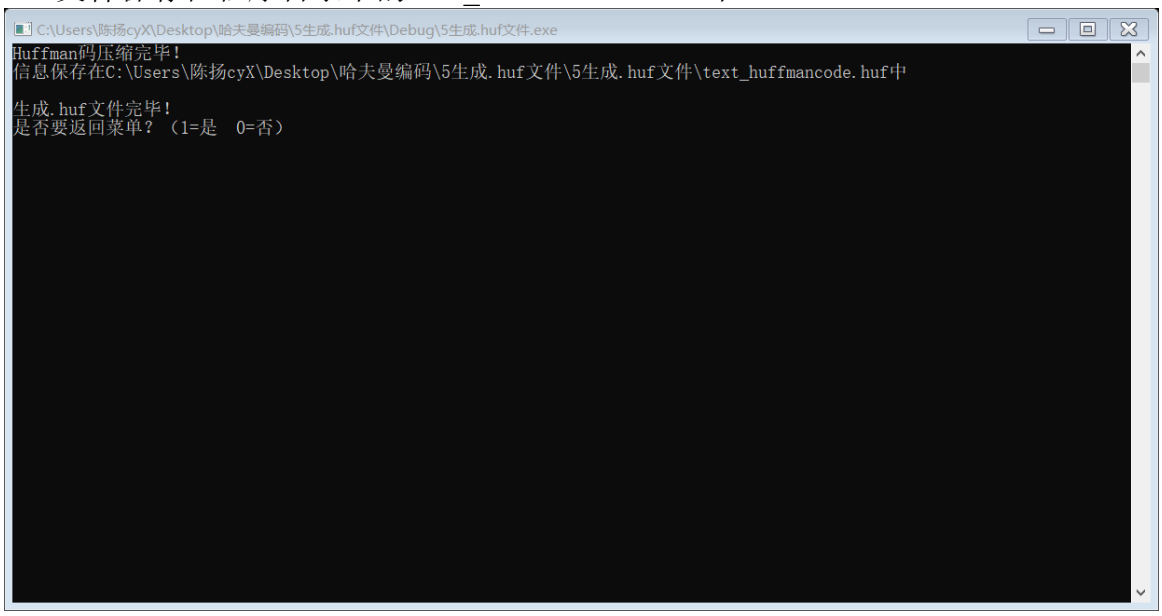


6. 根据编码生成.huf 文件

选择 6 项“根据编码生成.huf 文件”，程序会自动弹出“已选择操作 6！请确保存在文件 huffman_temp\huffman_encode.txt！”，并要求确认是否进行下一步操作。若输入 1，则进行下一步，若输入 0，则程序将会询问是否返回菜单。

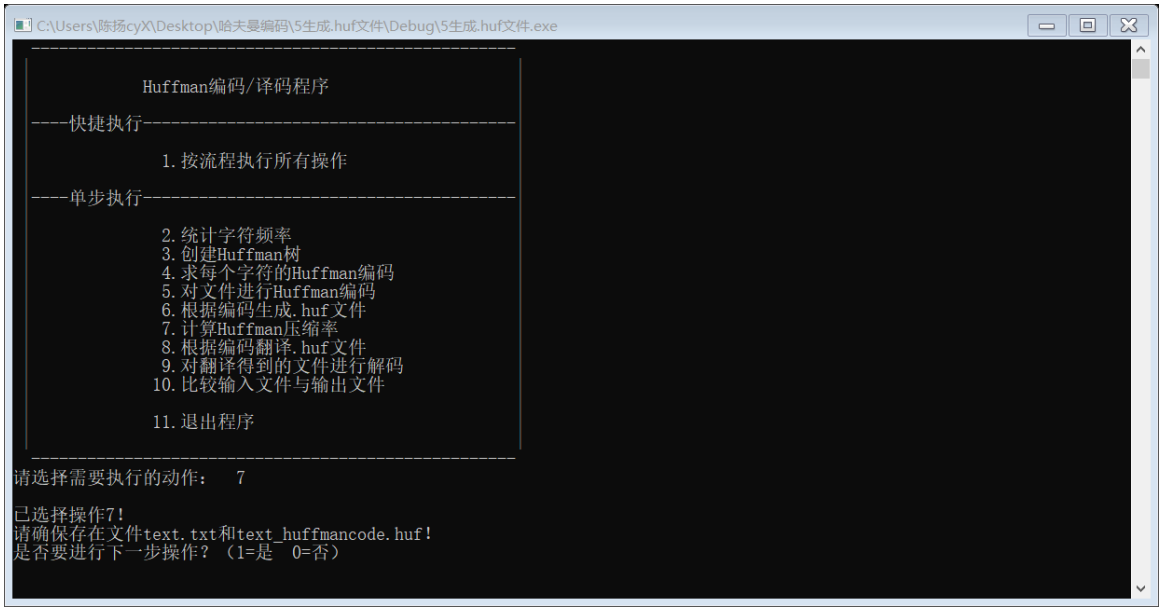


输入 1，确定进行下一步操作，则程序会自动根据 huffman_encode.txt 文件中的编码信息，生成 huf 文件。
huf 文件保存在程序目录下的 text_huffmancode.huf 中。

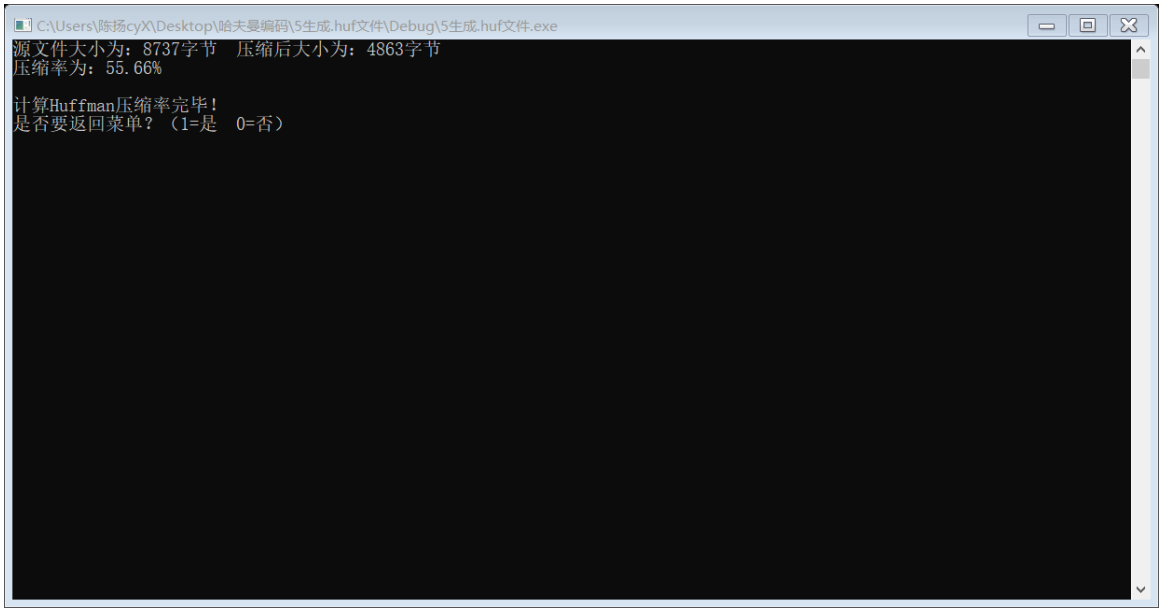


7. 计算 Huffman 压缩率

选择 7 项“计算 Huffman 压缩率”，程序会自动弹出“已选择操作 7！请确保存在文件 text.txt 和 text_huffmancode.huf”，并要求确认是否进行下一步操作。若输入 1，则进行下一步，若输入 0，则程序将会询问是否返回菜单。

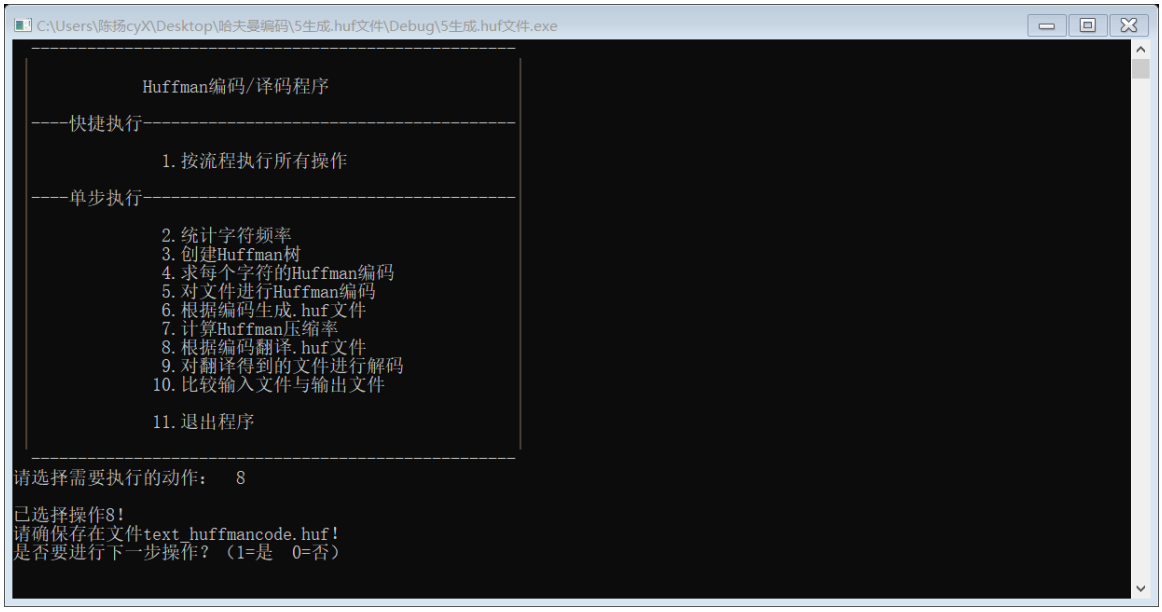


输入 1，确定进行下一步，则程序会自动计算 text.txt 和 text_huffmancode.huf 的文件大小，运算后得出压缩率。

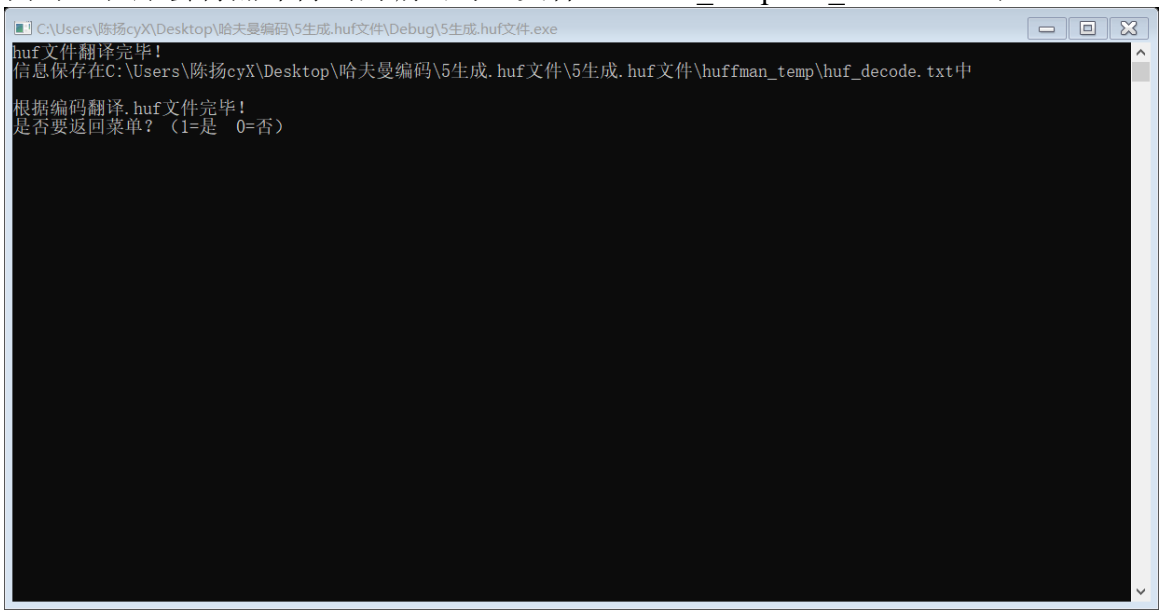


8. 根据编码翻译.huf 文件

选择 8 项“根据编码翻译.huf 文件”，程序会自动弹出“已选择操作 8！请确保存在文件 text_huffmancode.huf！”，并要求确认是否进行下一步操作。若输入 1，则进行下一步，若输入 0，则程序将会询问是否返回菜单。

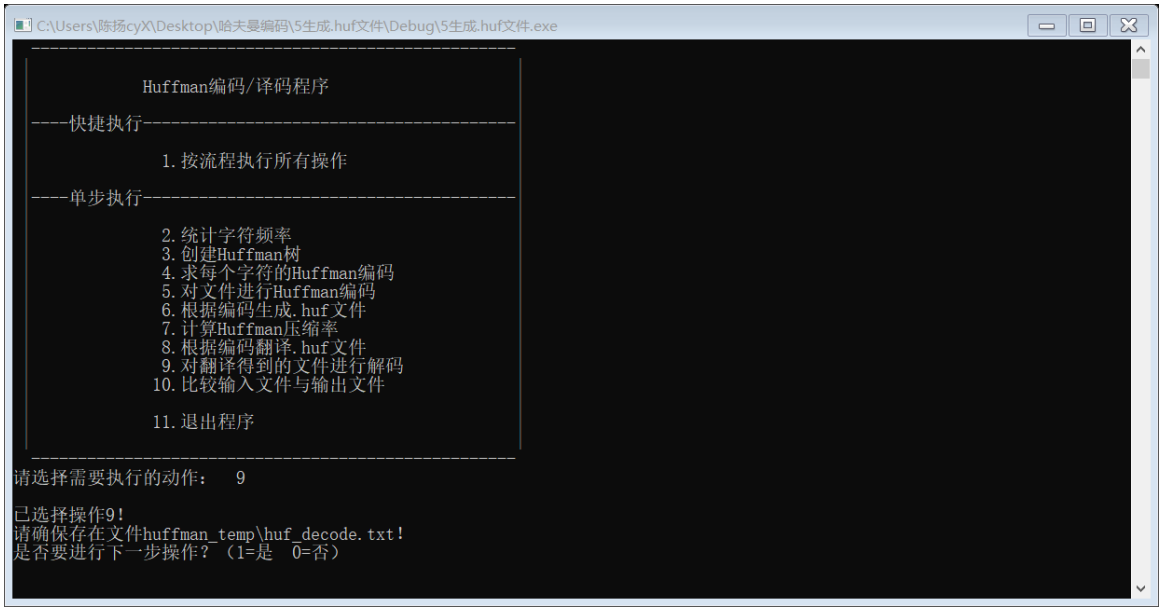


输入 1，确定进行下一步操作。则程序将会在后台自动将 text_huffmancode.huf 文件进行翻译成为只有 0/1 的 Huffman 编码文件，留待下一步操作。
同时，程序会将翻译得到的编码写入文件 huffman_temp\huf_decode.txt 中。

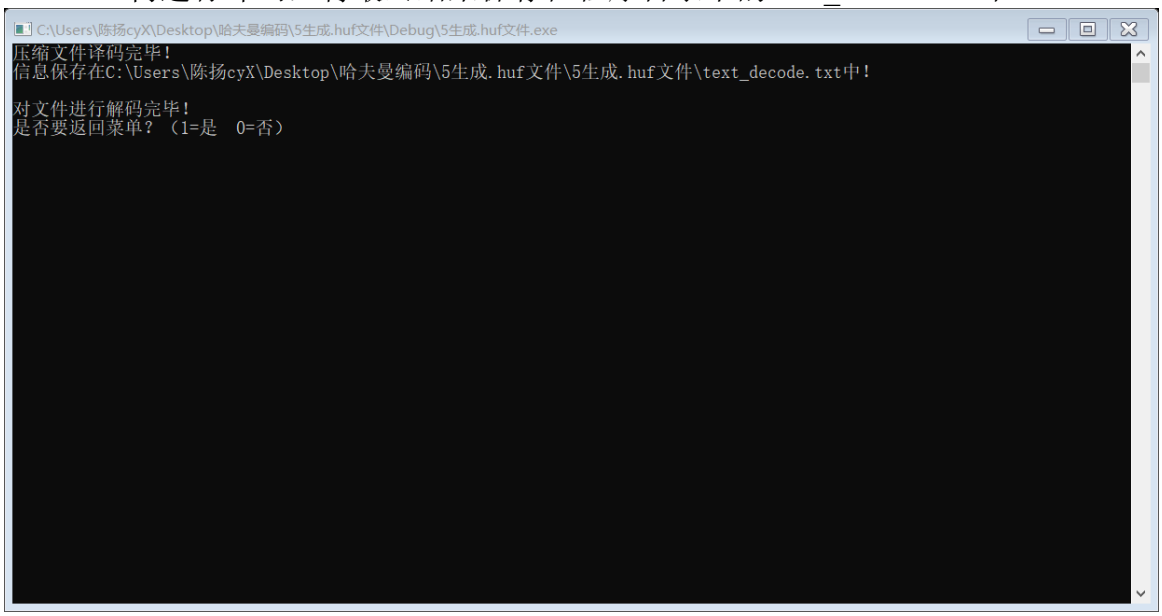


9. 对翻译得到的文件进行解码

选择 9 项“对翻译得到的文件进行解码”，程序会自动弹出“已选择操作 9！请确保存在文件 huffman_temp\huffman_decode.txt！”，并要求确认是否进行下一步操作。若输入 1，则进行下一步，若输入 0，则程序将会询问是否返回菜单。

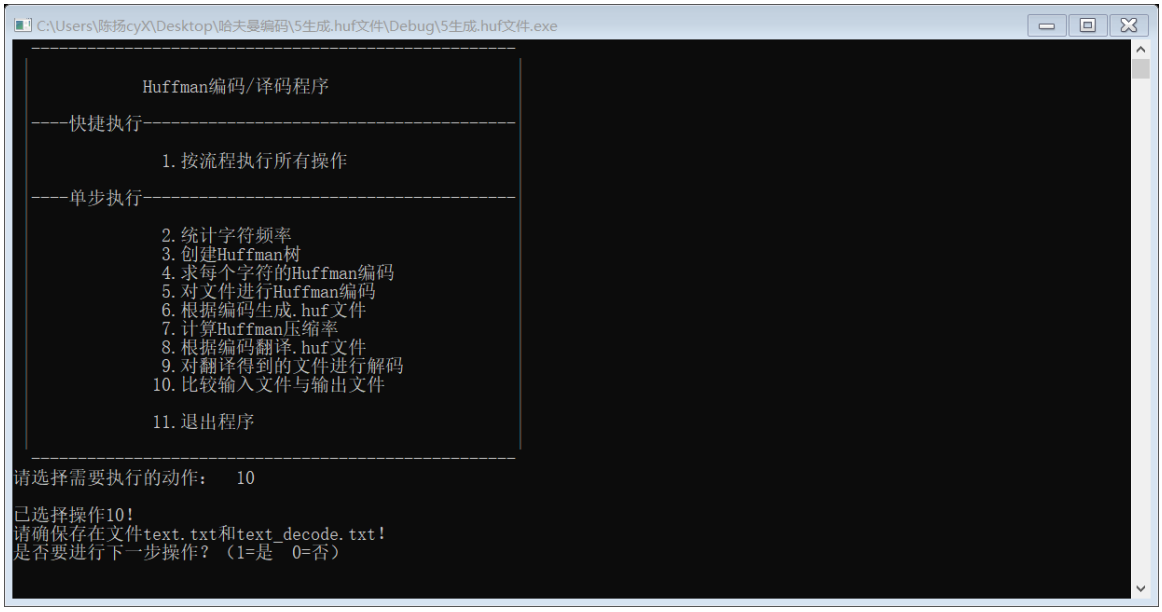


输入 1，确认进行下一步操作，则程序将会对 huf_decode.txt 按照生成的 Huffman 树进行译码，将最终结果保存在程序目录下的 text_decode.txt 中！

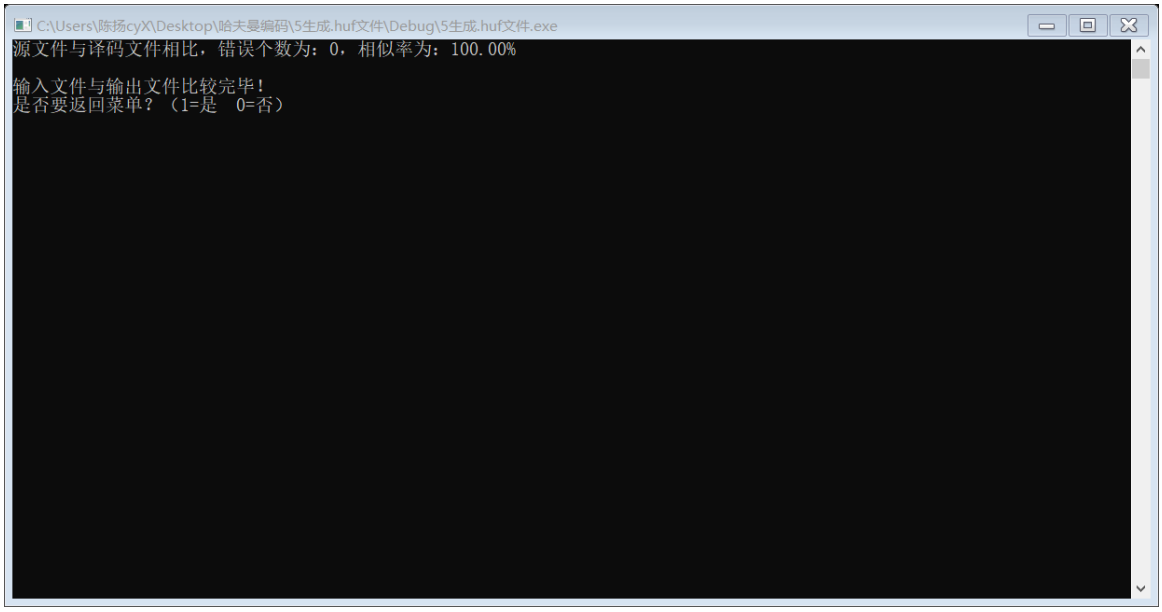


10. 比较输入文件与输出文件

选择 10 项“比较输入文件与输出文件”，程序会自动弹出“已选择操作 10！请确保存在文件 text.txt 和 text_decode.txt！”，并要求确认是否进行下一步操作。若输入 1，则进行下一步，若输入 0，则程序将会询问是否返回菜单。



输入 1，确定进行下一步操作，则程序将会遍历文件 text.txt 和 text_decode.txt，逐个字符进行比较，计算出错误个数，再将其与文本长度进行运算，得出正确率。



11. 按流程执行所有操作

选择该操作，即按照顺序执行菜单中的“2.统计字符频率”、“3.创建 Huffman 树”、“4.求每个字符的 Huffman 编码”、“5.对文件进行 Huffman 编码”、“6.根据编码生成.huf 文件”、“7.计算 Huffman 压缩率”、“8.根据编码翻译.huf 文件”、“9.对翻译得到的文件进行解码”、“10.比较输入文件与输出文件”

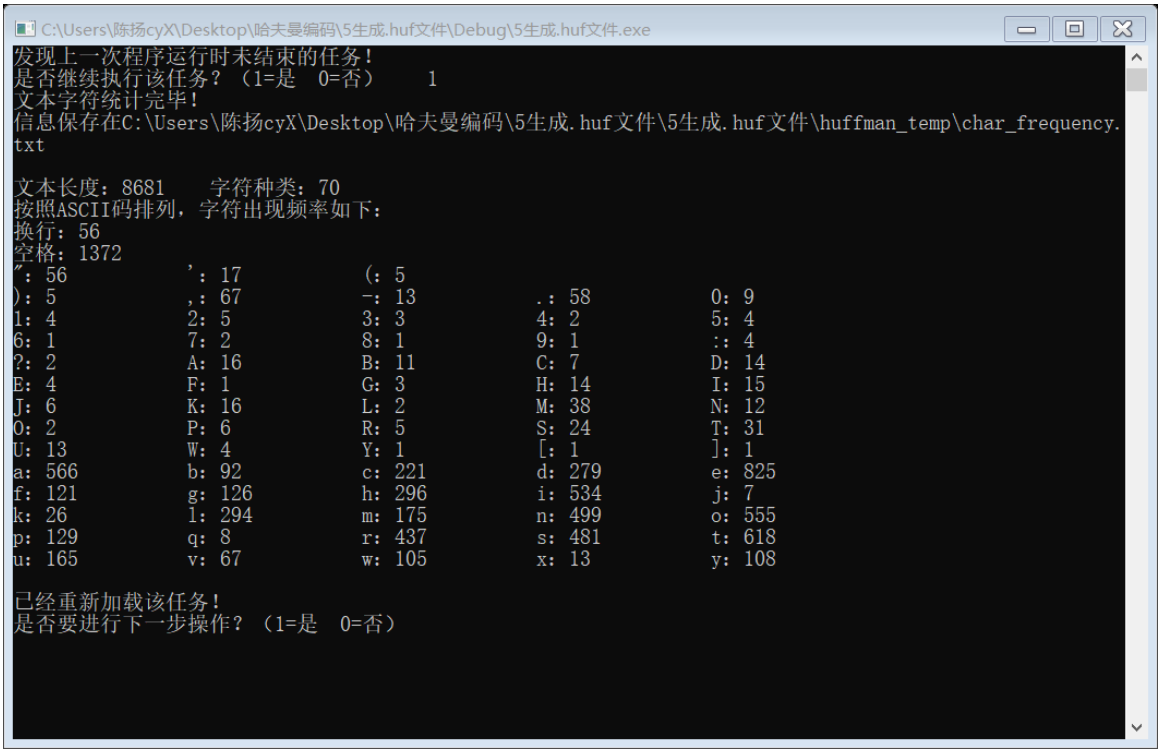
12. 退出程序

选择该操作，即执行 exit()函数，完全关闭并退出程序。

13. 对未完成任务的恢复处理

为确保程序文件的不丢失及恢复，引入了“对未完成任务的恢复处理”操作，该

操作在菜单中无选项，会在程序启动时自动执行。
通过判断程序目录下 huffman_temp\char_number.txt 文件是否存在，来判断该程序是否是第一次运行。若该文件存在，则说明程序已经运行过，且留有信息文档。此时，程序将会询问用户，是否继续上一次的任务。如果输入 1，确定继续执行，则程序将会跳过菜单步骤，直接读取 huffman_temp\char_number.txt 中的信息，按顺序执行**“3.创建 Huffman 树”、“4.求每个字符的 Huffman 编码”、“5.对文件进行 Huffman 编码”、“6.根据编码生成.huf 文件”、“7.计算 Huffman 压缩率”、“8.根据编码翻译.huf 文件”、“9.对翻译得到的文件进行解码”、“10.比较输入文件与输出文件”。**



如果输入 0，则程序将会显示菜单界面，重新开始运行。

（二）结果分析

1. 文件压缩率

使用 Huffman 编码对文本进行压缩的效果与多方面因素有关。文本长度、字符种类及字符的出现频率（权值）对文件的压缩率有着很大的影响。
笔者挑选了 10 篇不同的文章。对这 10 篇文章的 Huffman 编码/译码结果如下：

序号	文本长度	字符种类	源文件大小	huf 文件大小	压缩率	错误个数	准确率
1	4196	69	4228 字节	2380 字节	56.29%	0	100.00%
2	6299	64	6354 字节	3537 字节	55.67%	0	100.00%
3	8424	67	8496 字节	4770 字节	56.14%	0	100.00%
4	3989	67	4032 字节	2268 字节	56.25%	0	100.00%
5	9448	72	9517 字节	5361 字节	56.33%	0	100.00%
6	6432	80	6535 字节	3803 字节	58.37%	0	100.00%
7	380955	94	381465 字节	263758 字节	69.14%	0	100.00%
8	74351	93	76163 字节	45393 字节	59.60%	0	100.00%

9	161368	96	168965 字节	102226 字节	60.50%	0	100.00%
10	61574	88	62422 字节	39472 字节	63.23%	0	100.00%

再编写程序，随机生成指定个数的随机字符，进行 Huffman 编码/译码，结果如下：

序号	文本长度	字符种类	源文件大小	huf 文件大小	压缩率	错误个数	准确率
1	6000	95	6063 字节	4955 字节	81.73%	0	100%
2	12000	95	12113 字节	9934 字节	82.01%	0	100%
3	20000	95	20230 字节	16557 字节	81.84%	0	100%
4	25000	95	25236 字节	20726 字节	82.13%	0	100%
5	30000	95	30341 字节	24869 字节	81.96%	0	100%
6	35000	95	35355 字节	29028 字节	82.10%	0	100%
7	40000	95	40434 字节	33166 字节	82.03%	0	100%
8	100000	95	101025 字节	83016 字节	82.17%	0	100%
9	500000	95	505214 字节	415494 字节	82.24%	0	100%
10	1000000	95	1010493 字节	831125 字节	82.25%	0	100%
11	2000000	95	2021031 字节	1662545 字节	82.26%	0	100%
12	3000000	95	3031503 字节	2493853 字节	82.26%	0	100%
13	4000000	95	4042218 字节	3325383 字节	82.27%	0	100%
14	5000000	95	5052347 字节	4156946 字节	82.28%	0	100%
15	10000000	95	10105511 字节	8314397 字节	82.28%	0	100%

可见，Huffman 编码对规则文档（如新闻、论文等文章）编码时，压缩率能够保持在 50%~60%之间，有较好的压缩性能。然而，对于随机的无序字符，Huffman 的压缩性能并不是很出色，压缩率只有 82%左右。

2. 文件译码准确率

通过以上 25 篇不同文档的测试，发现本程序在 Huffman 编码/译码方面有着较好的性能，基本能够实现 100%的译码准确率。

但是，这并不代表本程序对于所有的文档译码都能够准确。在进行本程序测试时，未进行如下测试：

（1）对**中文文档**的编码/译码

（2）对 ASCII 码值不在 31~126 之间的字符的编码/译码

因此，本程序只能够基本保证对可显示字符（ASCII 码在 31~126 之间的字符）的编码/译码准确。

五、总结

（一）程序设计评价

经过测试，发现该程序能够实现 Huffman 编码/译码功能。程序能够实现的功能如下：

1.读入文档，该文档位于程序目录下的 text.txt。

2.统计并输出不同字符在文章中出现的频率。在处理空格、换行、标点等特殊字符时，为了让显示准确，程序会自动识别并作出标识，让用户能够更加清晰的了解。同时，程序还会将这些信息写入程序目录下的 huffman_temp\char_frequency.txt 中，方便以后的查看

3.根据字符频率构建 Huffman 树，并给出每个字符的 Huffman 编码。同时，程

序会将这些信息写入程序目录下的 huffman_temp\huffman_codeinfo.txt 中，方便以后的查看。
4.输出 Huffman 树，Huffman 编码。程序会根据用户的选择，选择是否展示 Huffman 树、Huffman 编码，若用户选择查看，则程序会自动显示每个字符对应的 Huffman 编码，及 text.txt 的 Huffman 编码。
5.利用已建好的 Huffman 编码，将文本文件进行编码，生成压缩文件 text_huffmancode.huf。
6.用 Huffman 编码存储的文件和输入文本文件进行比较，计算文件压缩率。
7.进行译码，将 huf 文件译码为 txt 文件，与原 txt 文件进行比较。
该程序完成了题目的全部要求。

然而，程序在设计时，仍有一些不足，如：
1.由于 Windows 系统特性，中文的编码方式与字符不同，无法使用 Huffman 算法进行编码，译码。
2.因时间与精力有限，没有对所有的 ASCII 字符进行测试。
3.因设备限制，没有将程序在不同的设备上运行，难以保证程序的兼容性、稳定性、准确性。
4.没有为程序编写 GUI 界面，非作者的用户使用起来仍需要一定时间熟悉程序的执行流程。
5.对于程序的执行逻辑、错误应对办法没有很好的优化，仍然存在 bug。

（二）Huffman 编码/译码技术评价

经过程序的编写与测试，笔者深入探索了 Huffman 编码的算法原理与执行过程，对 Huffman 编码/译码技术有如下的评价：
1.Huffman 编码/译码技术对于电文的传输、压缩有着一定的帮助作用，能够在一定程度上缩小电文的占用空间。
2.Huffman 编码技术对于字符规则的文档有较低的压缩效率，能够压缩至原文件大小的 50%~60%，是一项很不错的压缩技术。
3.Huffman 编码技术对于字符不规则的文档的压缩效果并不明显，压缩率为 82%左右，相比其他压缩技术，效果并不突出。
4.利用 Huffman 编码技术压缩文档时，需要占用大量的内存空间，可能引起程序的停止运行，仍需进一步优化，找出解决方案。

（三）参考文献

1. C 语言文件操作详解
https://www.cnblogs.com/likebeta/archive/2012/06/16/2551780.html
2. C 语言文件操作
http://blog.csdn.net/gneveek/article/details/6848473
3. C 语言获取文件大小
http://blog.csdn.net/yutianzuijin/article/details/27205121
4.《数据结构（C 语言版）》，严蔚敏，吴伟民，清华大学出版社
5. 哈夫曼树
http://blog.csdn.net/shuangde800/article/details/7341289
6. 百度百科-哈夫曼树
https://baike.baidu.com/item/%E5%93%88%E5%A4%AB%E6%9B%BC%E6%A0

%91/2305769?fr=aladdin
7. 【算法导论】哈夫曼树及编译码
http://blog.csdn.net/tengweitw/article/details/9838343

六、自我评价

经过了为期 2 个月左右的程序编写、报告撰写、PPT 制作等实践，现在自己相比于学期刚开始，更加深入了解了 Huffman 树、Huffman 编码/译码的程序设计思想与程序编写流程技巧。一方面，很好的完成了题目的要求；另一方面，很好的锻炼了自己的动手能力、资料收集能力、程序编写能力等。

这 2 个月的数据结构课程设计对我个人的锻炼的确很大，有几点令我印象深刻。

一是 C 语言对文件的读写操作。因为 C 语言老师并未教授这方面的知识，所以需要自学与理解。开始时，对于 FILE *fp=fopen(“”,”x”)中 x 的不同用法很是不解，因为 x 的用法很多，需要逐一甄别。比如 **r**：以只读方式打开文件，该文件必须存在；**w**：打开只写文件，若文件存在则文件长度清为 0，即该文件内容会消失，若文件不存在则建立该文件。**a**：以附加的方式打开只写文件，若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留；**wb**：只写打开或新建一个二进制文件，只允许写数据；**w+**：打开可读写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件等方式。现在可说，对于这些模式的运用和选择有了一定自己的理解。

二是对报告的撰写技巧的理解与完善。可能现在的报告中仍有不足之处，或存在某部分的叙述不清晰，不透彻，某部分的内容太啰嗦，太过冗长，日后仍需要进一步改进。

三是程序的编写中解决错误过程。在编写程序时，遇到过很多的大大小小的问题，很是令人头疼，有些是语法问题、有些是函数的调用问题、有些是程序本身的问题，都需要自己来进行调试来解决。尤其是在做&、|、^、<<、>>运算时，对自己的计算能力有很高的要求，尤其需要谨慎对待，

四是对 Huffman 树、Huffman 编码的理解加深。在学期初，其实自己对于 Huffman 树的理解仍是一知半解，只知道有这种编码方法，但并不知道其功能作用。经过这一次的课程设计实践，自己已经很大程度理解了 Huffman 树以及 Huffman 编码。

综上所述，经过这一次课程设计，自己很深刻的认识到，数据结构在程序编写、算法设计、算法分析等计算机领域中发挥着无比巨大的作用，学习数据结构是学习整个计算机理论的必经之路。

七、源程序

文件：huffman.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <direct.h>
#include <string.h>
#include <math.h>
#define OK 1
```

```
#define ERROR 0

typedef int Status;

/*链表元素的定义*/
typedef struct {
    int char_ASCII;
    int value;
    char *Huffmancode;
    int Huffmancode_bit = 0;
}ElemType;

//链表的定义
typedef struct node {
    ElemType elem;
    struct node *next;
}LinkNode, *LinkList;

//链表初始化
Status List_Init(LinkList &L) {
    if (!(L = (LinkList)malloc(sizeof(LinkNode))))
        exit(ERROR);
    L->next = NULL;
    return OK;
}

//链表插入
Status List_Insert(LinkList &L, int i, ElemType e) {
    LinkList p, s;
    p = L;
    int j = 0;
    while (p&&j < i - 1) {
        p = p->next;
        j++;
    }
    if (j > i - 1 || !p) return ERROR;
    s = (LinkList)malloc(sizeof(LinkNode));
    s->elem = e;
    s->next = p->next;
    p->next = s;
    return OK;
}

//删除链表
Status List_Destroy(LinkList &L) {
    LinkList p;
    while (L) {
        p = L;
        L = p->next;
```

```
        free(p);
    }
    return OK;
}

/*Huffman 树的定义*/
typedef struct {
    unsigned int value, ASCII_CODE;
    unsigned int parent, lchid, rchild;
}HTNode, *HuffmanTree;

//读取源文件信息，统计字符出现频率
Status File_sourceload() {
    FILE *fp1 = fopen("text.txt", "r");
    int char_num[256] = { 0 };  //根据 ASCII 表，有 256 个字符
    int text_length = 0, char_number = 0;
    int i, l = 0;
    char c;

    //初始化文件，若不存在则退出
    if (fp1 == NULL) {
        printf("读取文件失败或文件不存在！请重试！\n");
        return (ERROR);
    }

    //统计字符频率及字符串长度
    while ((c = fgetc(fp1)) != EOF) {
        char_num[c]++;
        text_length++;
    }
    //统计字符串中有多少种不同字符
    for (i = 0; i < 256; i++) {
        if (char_num[i] != 0)
            char_number++;
    }
    fclose(fp1);

    //结果写入：字符串长度、出现的字符种类
    _mkdir("huffman_temp");
    FILE *fp2 = fopen("huffman_temp\\char_frequency.txt", "w+");
    fprintf(fp2, "%d\t%d", text_length, char_number);
    for (i = 0; i < 256; i++) {
        if (char_num[i] != 0)
            fprintf(fp2, "\n%d %d", i, char_num[i]);
    }
    //写入文本注释
    fprintf(fp2, "\n\n****注：该文件用来存储源文件中的字符数量、字符种类个数、每个字符对应的 ASCII 码及他们的出现频率！****\n");
```

```
    char fileadress[100];
    _getcwd(fileadress, 100);
    printf(" 文 本 字 符 统 计 完 毕 ！ \n 信 息 保 存
在%s\\huffman_temp\\char_frequency.txt\n\n", fileadress);
    fclose(fp2);

    //结果输出
    printf("文本长度:%d    字符种类:%d\n", text_length, char_number);
    printf("按照 ASCII 码排列，字符出现频率如下：\n");
    for (i = 0; i < 256; i++) {
        if (char_num[i] != 0) {
            if (i == 32)
                printf("空格：%d\n", char_num[i]);
            else if (i == 13)
                printf("回车：%d\n", char_num[i]);
            else if (i == 10)
                printf("换行：%d\n", char_num[i]);
            else
                printf("%c：%d\t\t", i, char_num[i]);
            l++;
            if (l % 5 == 0)
                printf("\n");
        }
    }
    printf("\n");
    return OK;
}

//读取 huffman_temp\char_frequenc.txt 中的信息
Status  File_read_char_num(LinkList  &L,  int  &text_length,  int
&char_number) {
    int i;
    ElemType e;
    List_Init(L);
    FILE *fp = fopen("huffman_temp\\char_frequency.txt", "r");
    fscanf(fp, "%d\t%d", &text_length, &char_number);
    for (i = 1; i <= char_number; i++) {
        fscanf(fp, "\n%d %d", &e.char_ASCII, &e.value);
        List_Insert(L, i, e);
    }
    fclose(fp);
    return OK;
}

//选择权值最小的两个节点
Status HuffmanTree_Select(HuffmanTree HT, int n, int &s1, int &s2)
{
    int i;
```

```
    for (i = 1; i <= n; i++) {
        if (HT[i].parent == 0) {
            s1 = i;
            break;
        }
    }
    for (i = 1; i <= n; i++) {
        if (HT[i].value < HT[s1].value&&HT[i].parent == 0)
            s1 = i;
    }
    for (i = 1; i <= n; i++) {
        if (HT[i].parent == 0 && i != s1) {
            s2 = i;
            break;
        }
    }
    for (i = 1; i <= n; i++) {
        if (HT[i].value < HT[s2].value&&HT[i].parent == 0 && i !=
s1)
            s2 = i;
    }
    return OK;
}

//创建 Huffman 树
Status HuffmanTree_Create(HuffmanTree &HT, int n, LinkList L, int
if_print) {
    /*HuffmanTree 的初始化*/
    if (n <= 1) return ERROR;
    int m = 2 * n - 1;
    int i;
    LinkList p = L->next;
    HT = (HuffmanTree)malloc((m + 1) * sizeof(HTNode));
    for (i = 1; i <= m; i++) {
        HT[i].parent = 0;
        HT[i].lchid = 0;
        HT[i].rchild = 0;
    }
    for (i = 1; i <= n; i++) {
        HT[i].value = p->elem.value;
        HT[i].ASCII_CODE = p->elem.char_ASCII;
        p = p->next;
    }
    /*HuffmanTree 的创建*/
    int s1, s2;
    for (i = n + 1; i <= m; i++) {
        HuffmanTree_Select(HT, i - 1, s1, s2);
        HT[s1].parent = i;
        HT[s2].parent = i;
```

```
        HT[i].lchid = s1;
        HT[i].rchild = s2;
        HT[i].value = HT[s1].value + HT[s2].value;
    }
    /*根据输入的 if_print 选择是否要输出树的信息*/
    if (if_print == 1) {
        printf("Huffman 树的信息如下：\n");
        printf("结点\t 字符\t 权值\t 双亲\t 左孩子\t 右孩子\t\n");
        for (i = 1; i <= m; i++) {
            if (HT[i].ASCII_CODE == 32)
                printf("%d\t  空  格  \t%d\t%d\t%d\t%d\t\n",   i,
HT[i].value, HT[i].parent, HT[i].lchid, HT[i].rchild);
            else if (HT[i].ASCII_CODE == 13)
                printf("%d\t  回  车  \t%d\t%d\t%d\t%d\t\n",   i,
HT[i].value, HT[i].parent, HT[i].lchid, HT[i].rchild);
            else if (HT[i].ASCII_CODE == 10)
                printf("%d\t  换  行  \t%d\t%d\t%d\t%d\t\n",   i,
HT[i].value, HT[i].parent, HT[i].lchid, HT[i].rchild);
            else if (HT[i].lchid != 0 || HT[i].rchild != 0)
                printf("%d\t    无   \t%d\t%d\t%d\t%d\t\n",    i,
HT[i].value, HT[i].parent, HT[i].lchid, HT[i].rchild);
            else
                printf("%d\t%c\t%d\t%d\t%d\t%d\t\n",          i,
HT[i].ASCII_CODE,   HT[i].value,   HT[i].parent,   HT[i].lchid,
HT[i].rchild);
        }
    }
    return OK;
}

//从叶子到根逆向求每个字符的 Huffman 编码，储存在指针 L.Huffmancode 中
Status HuffmanTree_Code(HuffmanTree HT, int n, LinkList &L) {
    int i, start, c, f;
    LinkList p = L->next;
    char *cd = (char *)malloc(n * sizeof(char));
    cd[n - 1] = '\0';
    for (i = 1; i <= n; i++) {
        start = n - 1;
        for (c = i, f = HT[i].parent; f != 0; c = f, f = HT[f].parent)
            if (HT[f].lchid == c) cd[--start] = '0';
            else cd[--start] = '1';
            p->elem.Huffmancode = (char *)malloc((n - start) *
sizeof(char));
            strcpy(p->elem.Huffmancode, &cd[start]);
            p->elem.Huffmancode_bit = n - start - 1;
            p = p->next;
    }
    free(cd);
```

```
    return OK;
}

//存储 Huffman 编码为字典
Status HuffmanCode_Write(HuffmanTree HT, int n, LinkList &L) {
    LinkList p = L->next;
    FILE *fp = fopen("huffman_temp\\huffman_codeinfo.txt", "w+");
    int i, line = 0;
    char fileadress[100];
    _getcwd(fileadress, 100);
    printf("Huffman 编码完毕！\n 信息保存
在%s\\huffman_temp\\huffman_codeinfo.txt\n", fileadress);
    printf("各字符对应的 Huffman 编码为：\n");
    fprintf(fp, "code number:%d\n", n);
    for (i = 1; i <= n; i++, p = p->next) {
        fprintf(fp,      "%d\t%d\t%d\t",      p->elem.char_ASCII,
p->elem.value, p->elem.Huffmancode_bit);
        if (p->elem.char_ASCII == 32)
            printf("空格：");
        else if (p->elem.char_ASCII == 13)
            printf("回车：");
        else if (p->elem.char_ASCII == 10)
            printf("换行：");
        else
            printf("%c：", p->elem.char_ASCII);
        p->elem.Huffmancode = &p->elem.Huffmancode[0];
        while (*p->elem.Huffmancode != '\0') {
            fprintf(fp, "%c", *p->elem.Huffmancode);
            printf("%c", *p->elem.Huffmancode);
            p->elem.Huffmancode++;
        }
        line++;  //line 用来控制输出的格式
        fprintf(fp, "\n");
        printf("\t\t");
        if (line % 4 == 0)
            printf("\n");
    }
    //写入文件注释
    fprintf(fp, "\n****注：该文件用来存储所有字符对应的 ASCII 码以及他
们的 Huffman 编码！****\n");
    printf("\n 文件写入完毕！\n");
    fclose(fp);
    List_Destroy(L);
    return OK;
}

//返回某个字符的 Huffman 编码所在的指针
char * HuffmanCode_CharPoint(LinkList L, char c) {
```

```
    LinkList p = L->next;
    while (p->next&&p->elem.char_ASCII != c)
        p = p->next;
    return p->elem.Huffmancode;
}

//对文件进行 Huffman 编码 huffman_encode.txt
Status HuffmanCode_Encode(HuffmanTree HT, LinkList L, int if_print)
{
    FILE *fp = fopen("huffman_temp\\huffman_codeinfo.txt", "r");
    int char_number, i;
    ElemType e;
    List_Init(L);
    fscanf(fp, "code number:%d\n", &char_number);
    for (i = 1; i <= char_number; i++) {
        fscanf(fp,   "%d\t%d\t%d\t",   &e.char_ASCII,   &e.value,
&e.Huffmancode_bit);
        e.Huffmancode = (char*)malloc(e.Huffmancode_bit * sizeof(char));
        fscanf(fp, "%s\n", e.Huffmancode);
        List_Insert(L, i, e);
    }
    char c, *ch;
    FILE *sourcefile = fopen("text.txt", "r");
    FILE                    *huffman_encode                    =
fopen("huffman_temp\\huffman_encode.txt", "w+");
    if (if_print == 1)
        printf("文件的 Huffman 编码为：\n");
    while (!feof(sourcefile)) {
        c = fgetc(sourcefile);
        ch = HuffmanCode_CharPoint(L, c);
        //写入 Huffman 0/1 编码
        while (*ch != '\0') {
            if (if_print == 1)
                printf("%c", *ch);
            fputc(*ch, huffman_encode);
            ch++;
        }
    }
    //根据输入的 if_print 判断是否输出文件的 Huffman 编码
    if (if_print == 1)
        printf("\n");
    char fileadress[100];
    _getcwd(fileadress, 100);
    printf("Huffman   编   码   完   毕   !   \n   信   息   保   存
在%s\\huffman_temp\\huffman_encode.txt\n", fileadress);
    fprintf(huffman_encode, "\n\n****注：该文件用来存储 text.txt 文件
的原始 Huffman 编码信息！****\n");
    fclose(fp);
```

```
    fclose(sourcefile);
    fclose(huffman_encode);
    List_Destroy(L);
    return OK;
}

//每 8 位转化为对应的 ASCII 码
char bit_number_transform(int n[]) {
    char num;
    int i;
    for (i = 0; i < 8; i++) {
        //对一个 byte 进行移位操作
        if (n[i] == 1)
            num |= (1 << (7 - i));
        else
            num &= ~(1 << (7 - i));
    }
    return num;
}

//生成.huf 文件
Status text_huffmancode_create() {
    FILE *fp = fopen("huffman_temp\\huffman_encode.txt", "r");
    FILE  *temp  =  fopen("huffman_temp\\code_readytowrite.txt",
"wb+");
    FILE *huf = fopen("text_huffmancode.huf", "wb+");
    int codeinfo[8] = { 0 }, infonum = -1;
    int code_length = 0, code_mod, i, t;
    char c;
    while (!feof(fp)) {
        if (c = fgetc(fp) == '\n')
            break;
        code_length++;
    }
    //计算需要补 1 的个数
    code_mod = code_length % 8;
    fseek(fp, 0L, SEEK_SET);
    for (i = 1; i <= code_length; i++) {
        c = fgetc(fp);
        fputc(c, temp);
    }
    //对文件进行补 1，使得总长度为 8 的整数倍
    if (code_mod != 0)
        for (i = 1; i <= 8 - code_mod; i++)
            fprintf(temp, "%d", 1);
    fseek(temp, 0L, SEEK_SET);
    fputc(8 - code_mod, huf);
    //对文件进行编码
    for (i = 1; i <= (code_mod == 0 ? code_length : code_length +
```

```
8 - code_mod); i++) {
        c = fgetc(temp);
        if (c == '1')
            codeinfo[++infonum] = 1;
        else
            codeinfo[++infonum] = 0;
        if (infonum == 7) {
            c = bit_number_transform(codeinfo);
            fputc(c, huf);
            for (t = 0; t < 8; t++)
                codeinfo[t] = 0;
            infonum = -1;
        }
    }
    char fileadress[100];
    _getcwd(fileadress, 100);
    printf("Huffman 码压缩完毕！\n 信息保存在%s\\text_huffmancode.huf
中\n", fileadress);
    fclose(fp);
    fclose(temp);
    fclose(huf);
    return OK;
}

//解压.huf 文件
Status text_huffmancode_unzip() {
    FILE *huf = fopen("text_huffmancode.huf", "rb");
    FILE *unzip = fopen("huffman_temp\\huf_decode.txt", "wb+");
    long huflength = 0;
    double code_mod = 0.0;
    char c;
    int i, t;
    long count = 0;
    //获取文件长度
    fseek(huf, 0L, SEEK_SET);
    fseek(huf, 0L, SEEK_END);
    huflength = ftell(huf);
    //重新定位文件指针，开始读取
    fseek(huf, 0L, SEEK_SET);
    c = fgetc(huf);
    //获取 huf 文件的第一个字符，计算得到补 1 的个数
    for (i = 0; i <= 4; i++) {
        code_mod += ((c >> i) & 1)*pow(2, i);
    }
    //对 huf 文件进行译码
    while (!feof(huf)) {
        c = fgetc(huf);
        for (t = 0; t < 8; t++) {
```

```
            if (((c >> (7 - t)) & 1) == 1) {
                count++;
                fputc(49, unzip);
                if (count == (8 * (huflength - 1) - code_mod))
                    break;
            }
            else if (((c >> (7 - t)) & 1) == 0) {
                count++;
                fputc(48, unzip);
                if (count == (8 * (huflength - 1) - code_mod))
                    break;
            }
        }
        if (count == (8 * (huflength - 1) - code_mod))
            break;
    }
    char fileadress[100];
    _getcwd(fileadress, 100);
    printf("huf 文 件 翻 译 完 毕 ! \n 信 息 保 存
在%s\\huffman_temp\\huf_decode.txt 中\n", fileadress);
    fclose(huf);
    fclose(unzip);
    return OK;
}

//对文件进行 Huffman 解码 text_decode.txt
Status HuffmanCode_FileDecode(HuffmanTree HT, int char_number, int
text_length) {
    FILE *encodefile = fopen("huffman_temp\\huf_decode.txt", "r");
    FILE *decodefile = fopen("text_decode.txt", "w+");
    int q = 2 * char_number - 1;
    int k = 0, word = 0;
    char c;
    //从根开始，依次寻找 Huffman 编码对应的字符
    while (!feof(encodefile)) {
        c = fgetc(encodefile);
        if (word == text_length)
            break;
        if (c == '0')
            q = HT[q].lchid;
        else if (c == '1')
            q = HT[q].rchild;
        if (HT[q].lchid == 0 && HT[q].rchild == 0) {
            fputc(HT[q].ASCII_CODE, decodefile);
            q = 2 * char_number - 1;
            word++;
        }
    }
    char fileadress[100];
```

```
    _getcwd(fileadress, 100);
    printf("压缩文件译码完毕!\n 信息保存在%s\\text_decode.txt 中!\n",
fileadress);
    fclose(encodefile);
    fclose(decodefile);
    return OK;
}

//对输入文件和输出文件进行比较，求正确率
Status File_Error_Percentage() {
    int i;
    char ch1, ch2;
    float text_length = 3894.0, error_num = 0.0, percentage = 0.0;
    FILE *source = fopen("text.txt", "r");
    FILE *decode = fopen("text_decode.txt", "r");
    //逐个字符进行比较，不一样则计数，一样则跳过看下一个字符
    for (i = 0; i <= text_length; i++) {
        ch1 = fgetc(source);
        ch2 = fgetc(decode);
        if (ch1 != ch2)
            error_num++;
    }
    percentage = (text_length - error_num) / text_length;
    printf("源文件与译码文件相比，错误个数为：%d，相似率为：%.2f%%\n",
(int)error_num, percentage * 100);
    return OK;
}

//计算压缩率
Status File_Zip_Percentage() {
    long text_size, text_decode_size;
    float zippercentage;
    FILE *source = fopen("text.txt", "r");
    FILE *decode = fopen("text_huffmancode.huf", "r");
    //获取文件 text.txt 的大小
    fseek(source, 0L, SEEK_END);
    text_size = ftell(source);
    //获取文件 text_huffmancode.huf 的大小
    fseek(decode, 0L, SEEK_END);
    text_decode_size = ftell(decode);
    zippercentage = (float)text_decode_size / (float)text_size;
    printf("源文件大小为：%d 字节    压缩后大小为：%d 字节\n 压缩率
为：%.2f%%\n", text_size, text_decode_size, zippercentage*100.0);
    return OK;
}

//显示菜单
void menu() {
```

```
printf("  ------------------------------------------------\n");
printf(" |                                                |\n");
printf(" |           Huffman 编码/译码程序                 |\n");
printf(" |                                                |\n");
printf(" |----快捷执行-------------------------------------|\n");
printf(" |                                                |\n");
printf(" |             1.按流程执行所有操作                |\n");
printf(" |                                                |\n");
printf(" |----单步执行-------------------------------------|\n");
printf(" |                                                |\n");
printf(" |             2.统计字符频率                      |\n");
printf(" |             3.创建 Huffman 树                   |\n");
printf(" |             4.求每个字符的 Huffman 编码          |\n");
printf(" |             5.对文件进行 Huffman 编码            |\n");
printf(" |             6.根据编码生成.huf 文件              |\n");
printf(" |             7.计算 Huffman 压缩率                |\n");
printf(" |             8.根据编码翻译.huf 文件              |\n");
printf(" |             9.对翻译得到的文件进行解码           |\n");
printf(" |            10.比较输入文件与输出文件             |\n");
printf(" |                                                |\n");
printf(" |            11.退出程序                          |\n");
printf(" |                                                |\n");
printf("  ------------------------------------------------\n");
int choice = 0;
while (choice<1 || choice>11)
{
    printf("请选择需要执行的动作：  ");
    scanf("%d", &choice);
    if (choice<1 || choice>11)
        printf("\n 对不起，输入有误，请重新输入！\n\n");
}
//对后文中菜单函数的申明
void menu_function1();
void menu_function2();
void menu_function3();
void menu_function4();
void menu_function5();
void menu_function6();
void menu_function7();
void menu_function8();
void menu_function9();
void menu_function10();
void menu_function11();
switch (choice)
{
case 1:menu_function1(); break;
case 2:menu_function2(); break;
case 3:menu_function3(); break;
case 4:menu_function4(); break;
```

```
    case 5:menu_function5(); break;
    case 6:menu_function6(); break;
    case 7:menu_function7(); break;
    case 8:menu_function8(); break;
    case 9:menu_function9(); break;
    case 10:menu_function10(); break;
    case 11:menu_function11(); break;
    }
}

//询问是否进行下一步操作
int if_go_next() {
    int c = 2;
    while (c != 1 && c != 0) {
        printf("是否要进行下一步操作？（1=是  0=否）    ");
        scanf("%d", &c);
        getchar();
        if (c != 1 && c != 0)
            printf("\n 输入有误！请重新输入！\n");
    }
    if (c == 1) {
        system("cls");
        return 1;
    }
    else return 0;
}

//询问是否返回菜单
void if_backto_menu() {
    int c = 2;
    while (c != 1 && c != 0) {
        printf("是否要返回菜单？（1=是  0=否）    ");
        scanf("%d", &c);
        getchar();
        if (c != 1 && c != 0)
            printf("\n 输入有误！请重新输入！\n");
    }
    if (c == 1) {
        system("cls");
        menu();
    }
    else exit(OK);
}

//菜单功能 1
void menu_function1() {
    LinkList L;
    HuffmanTree HT;
```

```
    int text_length, char_number;
    int whether = 0;

    printf("\n 已选择操作 1! \n");
    if (if_go_next() == 0)
        if_backto_menu();

    printf("\n 开始进行第 1 步：统计字符频率! \n");
    FILE *fp1 = fopen("text.txt", "r");
    if (fp1 == NULL) {
        printf("读取文件失败或文件不存在! 请重试! \n");
        if_backto_menu();
    }
    File_sourceload();
    printf("\n 统计字符完毕! ");
    if (if_go_next() == 0)
        if_backto_menu();

    printf("\n 开始进行第 2 步：创建 Huffman 树! \n");
    printf("在创建之后，是否要查看 Huffman 树？（1=查看  0=不查看） ");
    scanf("%d", &whether);
    File_read_char_num(L, text_length, char_number);
    HuffmanTree_Create(HT, char_number, L, whether);
    printf("\n 创建 Huffman 树完毕! ");
    if (if_go_next() == 0)
        if_backto_menu();

    printf("\n 开始进行第 3 步：求每个字符的 Huffman 编码! \n");
    HuffmanTree_Code(HT, char_number, L);
    HuffmanCode_Write(HT, char_number, L);
    printf("\n 字符 Huffman 编码生成完毕! ");
    if (if_go_next() == 0)
        if_backto_menu();

    printf("\n 开始进行第 4 步：对文件进行 Huffman 编码! \n");
    printf("在创建之后，是否要查看文件的 Huffman 编码？（1=查看  0=不查
看） ");
    scanf("%d", &whether);
    HuffmanCode_Encode(HT, L, whether);
    printf("\n 文件 Huffman 编码完毕! ");
    if (if_go_next() == 0)
        if_backto_menu();

    printf("\n 开始进行第 5 步：根据编码生成.huf 文件! \n");
    text_huffmancode_create();
    printf("\n 生成.huf 文件完毕! ");
    if (if_go_next() == 0)
        if_backto_menu();
```

```
    printf("\n 开始进行第 6 步：计算 Huffman 压缩率！ \n");
    File_Zip_Percentage();
    printf("\n 计算 Huffman 压缩率完毕！ ");
    if (if_go_next() == 0)
        if_backto_menu();

    printf("\n 开始进行第 7 步：根据编码翻译.huf 文件！ \n");
    text_huffmancode_unzip();
    printf("\n 根据编码翻译.huf 文件完毕！ ");
    if (if_go_next() == 0)
        if_backto_menu();

    printf("\n 开始进行第 8 步：对翻译得到的文件进行解码！ \n");
    HuffmanCode_FileDecode(HT, char_number, text_length);
    printf("\n 对文件进行解码完毕！ ");
    if (if_go_next() == 0)
        if_backto_menu();

    printf("\n 开始进行第 9 步：比较输入文件与输出文件！ \n");
    File_Error_Percentage();

    printf("\n 所有操作执行完毕！ \n");
    if_backto_menu();
}

//菜单功能 2
void menu_function2() {
    printf("\n 已选择操作 2！请确保存在文件 text.txt！ \n");
    if (if_go_next() == 0)
        if_backto_menu();
    FILE *fp1 = fopen("text.txt", "r");
    if (fp1 == NULL) {
        printf("读取文件失败或文件不存在！请重试！ \n");
        if_backto_menu();
    }
    fclose(fp1);
    printf("\n 开始统计字符频率！ \n");
    File_sourceload();
    printf("\n 统计字符完毕！ \n");
    if_backto_menu();
}

//菜单功能 3
void menu_function3() {
    printf("\n 已选择操作 3！请确保存在文件
huffman_temp\\char_frequency.txt！ \n");
```

```
    if (if_go_next() == 0)
        if_backto_menu();
    FILE *fp = fopen("huffman_temp\\char_frequency.txt", "r");
    if (fp == NULL) {
        printf("读取文件失败或文件不存在！请重试！\n");
        if_backto_menu();
    }
    fclose(fp);
    printf("\n 开始创建 Huffman 树！\n");
    int whether = 0;
    printf("在创建之后，是否要查看 Huffman 树？（1=查看  0=不查看） ");
    scanf("%d", &whether);
    LinkList L;
    HuffmanTree HT;
    int text_length, char_number;
    File_read_char_num(L, text_length, char_number);
    HuffmanTree_Create(HT, char_number, L, whether);
    printf("\n 创建 Huffman 树完毕！\n");
    if_backto_menu();
}

//菜单功能 4
void menu_function4() {
    printf("\n 已 选 择 操 作 4 ！ 请 确 保 存 在 文 件
huffman_temp\\char_frequency.txt！\n");
    if (if_go_next() == 0)
        if_backto_menu();
    FILE *fp = fopen("huffman_temp\\char_frequency.txt", "r");
    if (fp == NULL) {
        printf("读取文件失败或文件不存在！请重试！\n");
        if_backto_menu();
    }
    fclose(fp);
    LinkList L;
    HuffmanTree HT;
    int text_length, char_number;
    File_read_char_num(L, text_length, char_number);
    HuffmanTree_Create(HT, char_number, L, 0);
    HuffmanTree_Code(HT, char_number, L);
    HuffmanCode_Write(HT, char_number, L);
    printf("\n 求每个字符的 Huffman 编码完毕！\n");
    if_backto_menu();
}

//菜单功能 5
void menu_function5() {
    printf("\n 已 选 择 操 作 5 ！ 请 确 保 存 在 文 件
huffman_temp\\huffman_codeinfo.txt！\n");
```

```
    if (if_go_next() == 0)
        if_backto_menu();
    FILE *fp = fopen("huffman_temp\\huffman_codeinfo.txt", "r");
    if (fp == NULL) {
        printf("读取文件失败或文件不存在！请重试！\n");
        if_backto_menu();
    }
    fclose(fp);
    int whether = 0;
    LinkList L;
    HuffmanTree HT;
    printf("在创建之后，是否要查看文件的 Huffman 编码？（1=查看  0=不查
看）  ");
    scanf("%d", &whether);
    int text_length, char_number;
    File_read_char_num(L, text_length, char_number);
    HuffmanTree_Create(HT, char_number, L, 0);
    HuffmanCode_Encode(HT, L, whether);
    printf("\n 对文件进行 Huffman 编码完毕！\n");
    if_backto_menu();
}

//菜单功能 6
void menu_function6() {
    printf("\n 已选择操作 6！请确保存在文件
huffman_temp\\huffman_encode.txt! \n");
    if (if_go_next() == 0)
        if_backto_menu();
    FILE *fp = fopen("huffman_temp\\huffman_encode.txt", "r");
    if (fp == NULL) {
        printf("读取文件失败或文件不存在！请重试！\n");
        if_backto_menu();
    }
    fclose(fp);
    text_huffmancode_create();
    printf("\n 生成.huf 文件完毕！\n");
    if_backto_menu();
}

//菜单功能 7
void menu_function7() {
    printf("\n 已选择操作 7！\n 请确保存在文件 text.txt 和
text_huffmancode.huf! \n");
    if (if_go_next() == 0)
        if_backto_menu();
    FILE *source = fopen("text.txt", "r");
    FILE *decode = fopen("text_huffmancode.huf", "r");
    if (source == NULL || decode == NULL) {
```

```
        printf("读取文件失败或文件不存在！请重试！\n");
        if_backto_menu();
    }
    fclose(source);
    fclose(decode);
    File_Zip_Percentage();
    printf("\n 计算 Huffman 压缩率完毕！\n");
    if_backto_menu();
}

//菜单功能 8
void menu_function8() {
    printf("\n 已选择操作 8！\n 请确保存在文件 text_huffmancode.huf!
\n");
    if (if_go_next() == 0)
        if_backto_menu();
    FILE *huf = fopen("text_huffmancode.huf", "rb");
    if (huf == NULL) {
        printf("读取文件失败或文件不存在！请重试！\n");
        if_backto_menu();
    }
    fclose(huf);
    text_huffmancode_unzip();
    printf("\n 根据编码翻译.huf 文件完毕！\n");
    if_backto_menu();
}

//菜单功能 9
void menu_function9() {
    printf("\n 已选择操作 9！\n 请确保存在文件
huffman_temp\\huf_decode.txt! \n");
    if (if_go_next() == 0)
        if_backto_menu();
    FILE *encodefile = fopen("huffman_temp\\huf_decode.txt", "r");
    if (encodefile == NULL) {
        printf("读取文件失败或文件不存在！请重试！\n");
        if_backto_menu();
    }
    fclose(encodefile);
    int text_length, char_number;
    HuffmanTree HT;
    LinkList L;
    File_read_char_num(L, text_length, char_number);
    HuffmanTree_Create(HT, char_number, L, 0);
    HuffmanCode_FileDecode(HT, char_number, text_length);
    printf("\n 对文件进行解码完毕！\n");
    if_backto_menu();
}
```

```
//菜单功能 10
void menu_function10() {
    printf("\n 已选择操作 10！\n 请确保存在文件 text.txt 和
text_decode.txt！\n");
    if (if_go_next() == 0)
        if_backto_menu();
    FILE *decode = fopen("text_decode.txt", "r");
    if (decode == NULL) {
        printf("读取文件失败或文件不存在！请重试！\n");
        if_backto_menu();
    }
    fclose(decode);
    File_Error_Percentage();
    printf("\n 输入文件与输出文件比较完毕！\n");
    if_backto_menu();
}

//菜单功能 11
void menu_function11() {
    exit(OK);
}

//任务恢复，如果选择是则直接跳过读取字符频率，继续执行
void Project_Recovery() {
    int whether = 0;
    LinkList L;
    HuffmanTree HT;
    int text_length, char_number;
    printf("是否继续执行该任务？（1=是  0=否）    ");
    scanf("%d", &whether);
    if (whether == 1) {
        File_sourceload();
        printf("已经重新加载该任务！\n");
        if (if_go_next() == 0)
            if_backto_menu();

        printf("\n 开始进行第 2 步：创建 Huffman 树！\n");
        printf("在创建之后，是否要查看 Huffman 树？（1=查看  0=不查看）
");
        scanf("%d", &whether);
        File_read_char_num(L, text_length, char_number);
        HuffmanTree_Create(HT, char_number, L, whether);
        printf("\n 创建 Huffman 树完毕！");
        if (if_go_next() == 0)
            if_backto_menu();

        printf("\n 开始进行第 3 步：求每个字符的 Huffman 编码！\n");
```

```
        HuffmanTree_Code(HT, char_number, L);
        HuffmanCode_Write(HT, char_number, L);
        printf("\n 字符 Huffman 编码生成完毕！ ");
        if (if_go_next() == 0)
            if_backto_menu();

        printf("\n 开始进行第 4 步：对文件进行 Huffman 编码！ \n");
        printf("在创建之后，是否要查看文件的 Huffman 编码？（1=查看  0=
不查看）  ");
        scanf("%d", &whether);
        HuffmanCode_Encode(HT, L, whether);
        printf("\n 文件 Huffman 编码完毕！ ");
        if (if_go_next() == 0)
            if_backto_menu();

        printf("\n 开始进行第 5 步：根据编码生成.huf 文件！ \n");
        text_huffmancode_create();
        printf("\n 生成.huf 文件完毕！ ");
        if (if_go_next() == 0)
            if_backto_menu();

        printf("\n 开始进行第 6 步：计算 Huffman 压缩率！ \n");
        File_Zip_Percentage();
        printf("\n 计算 Huffman 压缩率完毕！ ");
        if (if_go_next() == 0)
            if_backto_menu();

        printf("\n 开始进行第 7 步：根据编码翻译.huf 文件！ \n");
        text_huffmancode_unzip();
        printf("\n 根据编码翻译.huf 文件完毕！ ");
        if (if_go_next() == 0)
            if_backto_menu();

        printf("\n 开始进行第 8 步：对翻译得到的文件进行解码！ \n");
        HuffmanCode_FileDecode(HT, char_number, text_length);
        printf("\n 对文件进行解码完毕！ ");
        if (if_go_next() == 0)
            if_backto_menu();

        printf("\n 开始进行第 9 步：比较输入文件与输出文件！ \n");
        File_Error_Percentage();

        printf("\n 所有操作执行完毕！ \n");
        if_backto_menu();
    }
    else {
        system("cls");
        menu();
```

```
    }
}

void main() {
    FILE *fp = fopen("huffman_temp\\char_frequency.txt", "r");
    //判断是否存在未完成任务
    if (fp != NULL) {
        printf("发现上一次程序运行时未结束的任务! \n");
        Project_Recovery();
    }
    else
        menu();
    system("pause");
}
```