

# 数据结构课程设计报告内容

## 一. 课程设计题目

哈夫曼编/译码器

### 【基本要求】

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) 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)；
- (5) 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；
- (6) 根据相应哈夫曼编码，对编码后的进行译码，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

给出哈夫曼树的定义如下：

假设有  $n$  个权值  $\{w_1, w_2, \dots, w_n\}$ ，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 最小的二叉树称作为最优二叉树或哈夫曼树。

观察后易得，由于哈夫曼树没有度为 1 的结点（严格二叉树），则一颗有  $n$  个叶子结点的哈夫曼树总共有  $2n-1$  个结点（度为 1： 0； 度为 0：  $n$ ； 度为 2：  $n-1$ ），可以存储在一个大小为  $2n-1$  的一维数组中，对于每个结点而言，既需要知道双亲的信息又需要知道其左右孩子的信

息。

所以我们构造如下结构体用于存储哈夫曼树：

```
typedef struct{
    unsigned int weight;
    unsigned int parent, lchild, rchild;
}HTNode, * HuffmanCode;
//定义无符号整型变量以对应下文的初值 '0'
```

同时，编码表存储在如下动态数组中：

```
typedef char **HuffmanCode;
```

下文中的 HT 代表哈夫曼树，HC 代表编码表，n 表示字符种类数，w 表示某字符对应的权值。

在构建 HT 过程中，我们需要知道每个字符的权值（这在读入文本 Text.txt 时就已经处理好），开始为 HT 分配  $2n$  个存储空间（我们不使用 0 号单元，以方便使用，HC 也一样），后在 HT 的  $1\sim n$  的空间里赋值  $\{w, 0, 0, 0\}$  作为  $n$  个初始结点，在  $(n+1)\sim (2n-1)$  的空间里赋值  $\{0, 0, 0, 0\}$ 。然后设置一个循环体对  $(n+1)\sim (2n-1)$  结点进行操作， $i$  作为循环是否结束的标志变量 ( $i\leq 2n-1$ )，选取  $1\sim (i-1)$  中 parent 为 0 且 weight 最小的两个结点（Select 函数实现），让他们分别做为  $i$  号结点左孩子与右孩子，两结点的双亲为  $i$ ， $i$  号结点的 weight 为其左右孩子 weight 之和。

在编码过程中，采用由叶子结点到根逆向求编码的方法。首先给 HC 分配  $n+1$  个空间（0 号单元不使用），再对 cd 分配  $n$  个空间作为分配求编码的工作空间（ $n$  个字符的赫夫曼编码长度一定小于  $n$ ），后逐个字符求赫夫曼编码存在 cd 中，最后复制到 HC 中，释放 cd 的空间。

相似的在译码过程中，可以从根出发走一条根到叶子的路径。由于本程序在构造 HT 的同时，也构造了 CharArray 存放出现的字符，构造 HC 时，HC[1]~HC[n]中的字符顺序与 CharArray 中字符顺序相同，故在读入编码时，可边读边由根到叶子结点走，用动态数组 st 暂时存储编码，走到叶子结点后停止，对比 st 与 HC，从而确定对应 CharArray 中的字符，输出到文件 Decode.txt 中，最后释放 st 的空间。

此外，本程序还会额外生成 Encoded.txt 文件，作为 01 码存储。因为在译码过程中，我们要将编码后的 01 码每七位转化为十进制 0~127 的 ASCII 码，将其按照对应的字符存储在 EncodedText.huf 中；译码亦是如此，先转化为 01 码再对照密码表翻译为源文件。

除了主体哈夫曼算法，本程序还有操作数队列，文件操作，显示界面算法。下面仅做简单介绍。

对于操作数，本程序允许逐个输入或者连续输入，操作数存储采用队列，符合“先进先出原则”。结构体定义如下：

```
//用队列存储操作数
typedef char ElemType;
typedef struct QNode{
    ElemType    data;
    struct QNode *next;
}QNode,*QueuePtr;
typedef struct{
    QueuePtr front;           //头指针
    QueuePtr rear;           //尾指针
    unsigned short int length;//队列长度
    unsigned short int flag;  //初始化后 flag=1, 否则 flag=0
}LinkQueue;
```

对于文件操作，在读入文件的时候，本程序会做好基本信息处理，关于基本信息的存储结构如下：

```
//ASCII 码表 256 个字符取值
typedef struct{
    char filename[50];        //记录文件名
    char *character;          //复制 txt 文本内容
    unsigned int  *letter_w;//各种字符的权值，标号为 ASCII
    int  ch_quantity;         //出现字符种类个数
    int  length;              //文本长度
    short int flag;           //初始化后 flag=1, 否则 flag=0
    short int mark;           //未编译 mark=0, 编译后 mark=1
}Text;
```

### 三. 程序结构

所有函数定义存储在 operand.h 文件里，操作数队列结构体定义在 operand.h，哈夫曼树结构体定义在 huffmantree.h，文本信息存储结构体定义在 file.h 中。

MAIN.cpp 中为 main 函数与 END 函数；FILE.cpp 存放着与文件操作有关的函数（InitText, DestroyText, PrintText, FileName, Read\_File）；HFT.cpp 中为与哈夫曼构造，编码译码有关的函数（HFT\_Code, HuffmanCoding, Select, PrintCode, HFT\_Decode）；OPERAND QUEUE.cpp 中为与队列操作有关的函数（InitQueue, EnLQueue, DeQueue, DestroyQueue, GetHead, Empty, PrintQueue, LenQueue）；OPERAND.cpp 中为与界面操作有关的函数（Interface\_A, Inerface\_B, Initialization, EmptyData, Operate）。

全局变量有：

```
typedef char **HuffmanCode;    //顺序存放对应编码
```

```

unsigned short int *CharArray; //顺序存放字符数组
unsigned int *HFW;             //存放连续的有效权值
short int X;                   //01 码能否被 7 整除的标志

```

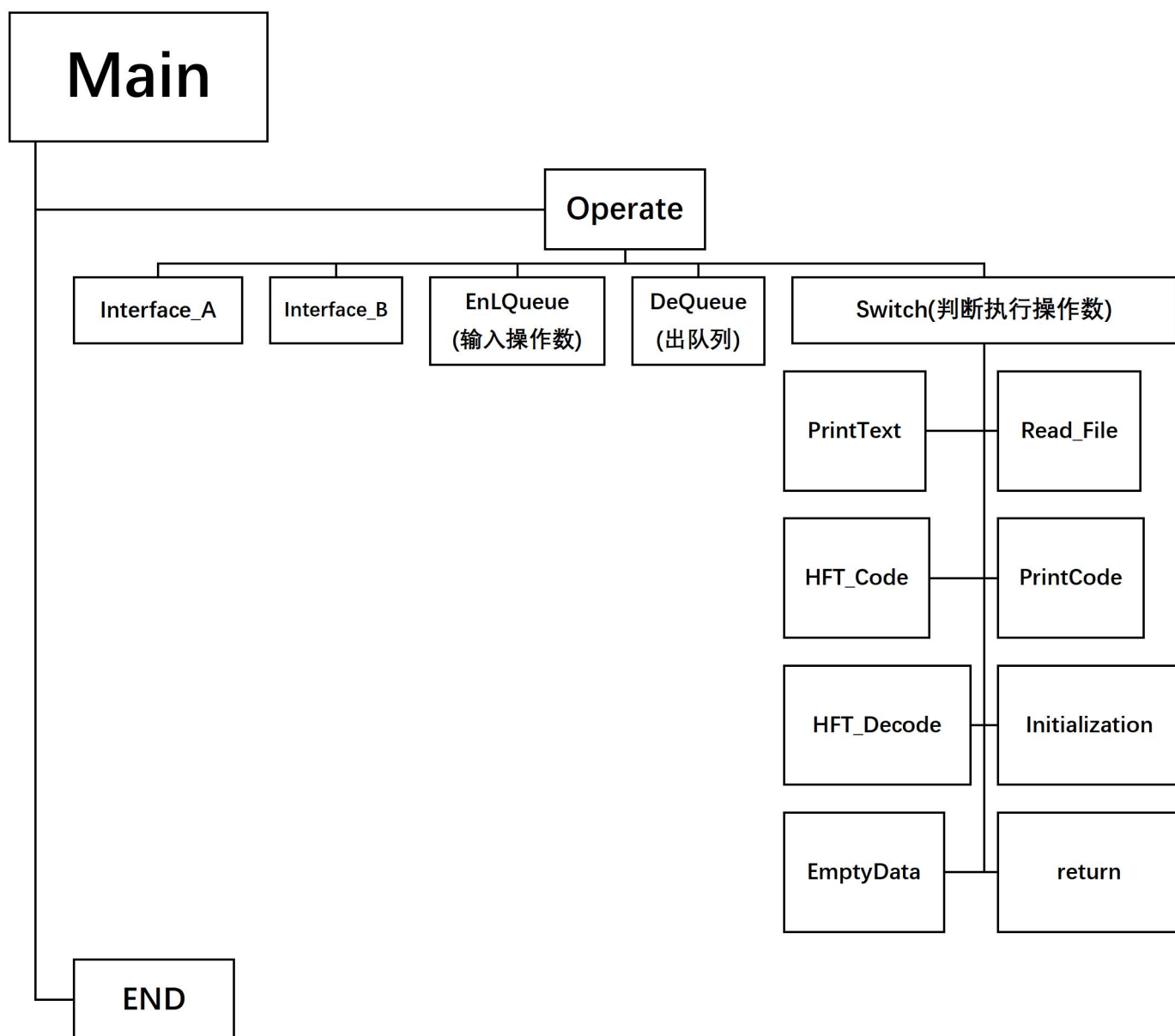
宏定义有：

```

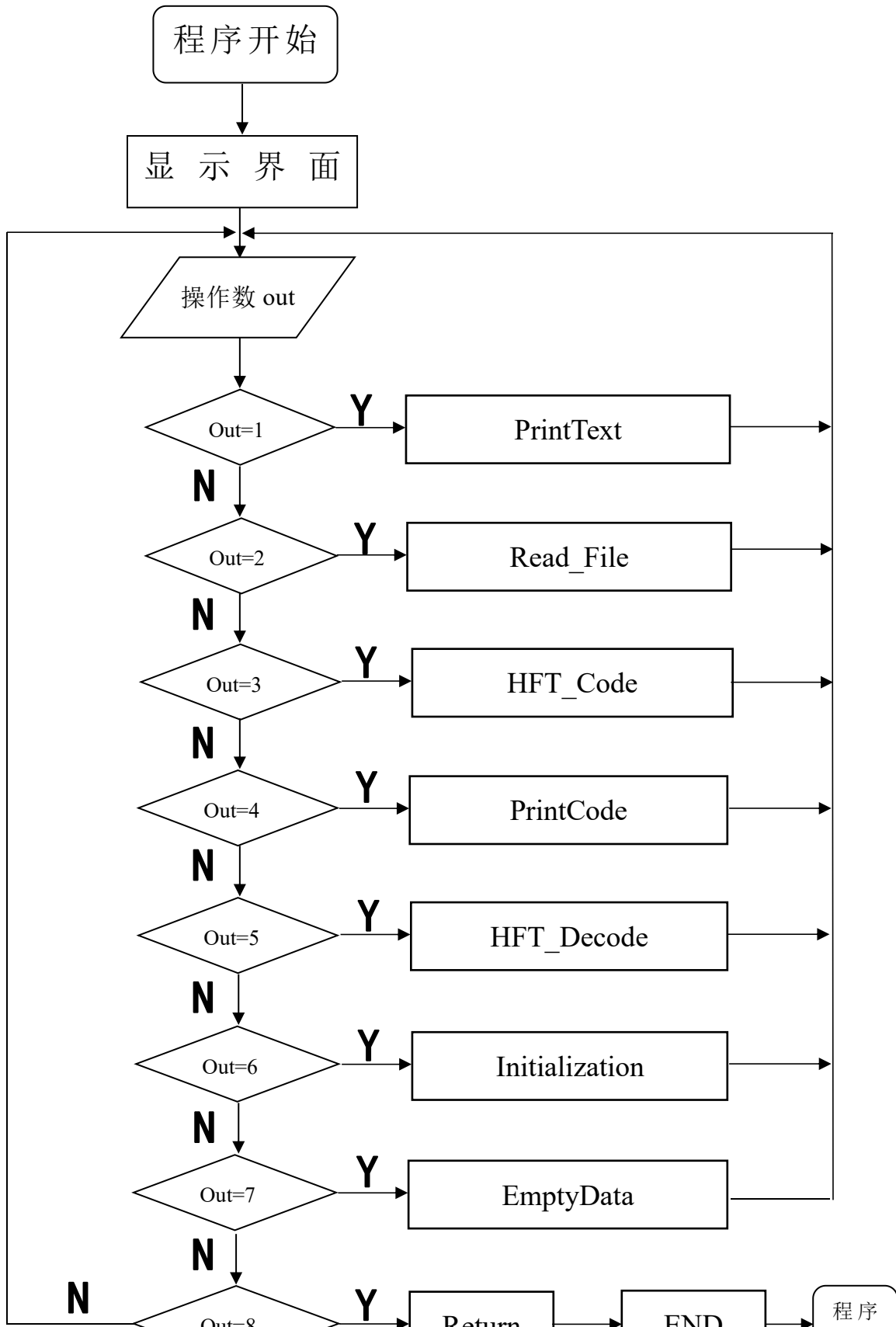
#define ASCII 256
#define NEWLINE printf("\n")      /*换行*/
#define LOOP while(1)             /*死循环*/
#define SPACE printf("          ") /*十个空格*/
#define MAX_N 50000               //最大字符数
#define second 990                //作为 Sleep 函数参数

```

主要函数调用关系如下图：



主要程序流程图如下图：



## 四. 实验结果与分析

### 1. 用户使用说明

备注：

1. 若直接打开 HuffmanTree Encoder and Decoder\Debug 目录下的 HuffmanTree Encoder and Decoder.exe 应用程序进行编码和译码，请将要操作的 txt 存储在 debug 目录下；若在 vc 中打开 HuffmanTree Encoder and Decoder 目录下的 dsw 文件，则请将要操作的 txt 存储在 HuffmanTree Encoder and Decoder 目录下。

2. 在操作键位中，“6. 程序初始化”会将操作数队列初始化，请不要连续输入时在 6 后面另加操作数。如：“1263”实际只会执行“126”而不会执行接下来的“3”。“7. 清空数据”则不会清空操作数队列。

3. 程序会生成四个文件。Codetxt：存放密码表；Encoded.txt：存放 01 码；EncodedText.huf：目标文件的压缩文件；Decoded.txt：译码后文件。

第一步：打开 HuffmanTree Encoder and Decoder\Debug 目录下的 HuffmanTree Encoder and Decoder.exe 应用程序。

第二步：阅读使用说明后，在开始界面输入 y/Y，从而进入操作界面。

```
赫夫曼 编/译 码器
作者: 19218101
Huffman encoder / decoder
Designer: 19218101

-----

使用说明
本程序可对一个文本文件中的字符进行赫夫曼编码并且生成编码文件。也可将编码文件译码还原为文本文件。
Direction for use
Huffman encoding a file(.txt) to generate a file(Encoded txt.huf). And also decode it and restore it to a file(Decoded.txt).

-----

Huffman encoder && decoder by 19218101
Compiled on Nov 8 2019 at 19:03:34

-----
完成阅读, 是否进入下一步操作? (Y/N): y
```

第三步：在操作界面阅读按键说明后，输入操作数序列。操作数序列可以连续输入也可以单步输入。例如输入：121345\n 和

```
Huffman encoder / decoder

-----

键位说明
1. 打印文本存储信息
2. 打开并读取文件
3. 赫夫曼编码并生成压缩文件
4. 打印码表并存储在code.txt中
5. 对编码后生成的文件译码并生成txt文件
6. 程序初始化
7. 清空数据
8. 退出程序

-----

操作数可以连续输入, 输入完成后按 回车 结束输入
例如: 输入 [72314568回车] 或者 [8回车]
输入操作数序列(1 - 8): 121345
你输入的操作数序列: 1 2 1 3 4 5
```

1\n2\n1\n3\n4\n5\n 是等价的。

```
C:\Users\bzdel\\Desktop\19218101\HuffmanTree Encoder and Decoder - 副本\Debug\

Huffman encoder / decoder

键位说明
1. 打印文本存储信息
2. 打开并读取文件
3. 赫夫曼编码并生成压缩文件
4. 打印码表并存储在code.txt中
5. 对编码后生成的文件译码并生成txt文件
6. 程序初始化
7. 清空数据
8. 退出程序

操作数可以连续输入，输入完成后按 回车 结束输入
例如：输入 [72314568回车] 或者 [8回车]
输入操作数序列(1 - 8): 1
你输入的操作数序列: 1
```

第四步：程序开始时，我们可以先输入 1，查看当前文本存储情况，之后输入 2，输入你要编码的文件名（例如 Text.txt），再输入 1 查看是否读取文本成功。

以下图片显示的是 Text.txt 文档中对应的编码，即测试二。

```
请继续输入操作序列(1 - 8): 121345
你输入的操作数序列: 1 2 1 3 4 5

文本域如下:
Filename: 0
Types_num: 0
CharWeight([CHAR-ASCII]WEIGHT):
NULL
Length: 0
Initialization: 1
Encode: 0

Input your filename(xxx.txt): Text.txt

The program is reading the FILE: (Text.txt).
FILE: (Text.txt) read completely.
```



```

文本域如下:
Filename:          Text.txt
Types_num:         58
CharWeight([CHAR-ASCII]WEIGHT):
[ \n- 10] 42      [  - 32] 1583      [ !- 33] 6
[ " - 34] 18      [ , - 39] 10       [ , - 44] 77
[ - 45] 9         [ . - 46] 73       [ : - 58] 5
[ ; - 59] 1       [ ? - 63] 2        [ A - 65] 27
[ B - 66] 4       [ C - 67] 6        [ D - 68] 2
[ E - 69] 1       [ F - 70] 4        [ G - 71] 9
[ H - 72] 3       [ I - 73] 24       [ J - 74] 1
[ K - 75] 1       [ L - 76] 16       [ M - 77] 8
[ N - 78] 24      [ O - 79] 3        [ P - 80] 4
[ R - 82] 2       [ S - 83] 4        [ T - 84] 10
[ W - 87] 15      [ Y - 89] 3        [ a - 97] 527
[ b - 98] 105     [ c - 99] 172     [ d - 100] 260
[ e - 101] 870    [ f - 102] 213    [ g - 103] 159
[ h - 104] 377    [ i - 105] 528    [ j - 106] 20
[ k - 107] 51     [ l - 108] 318    [ m - 109] 179
[ n - 110] 449    [ o - 111] 587    [ p - 112] 90
[ q - 113] 7      [ r - 114] 401    [ s - 115] 407
[ t - 116] 649    [ u - 117] 176    [ v - 118] 81
[ w - 119] 147    [ x - 120] 5      [ y - 121] 122
[ z - 122] 6
Length:           8903
Initialization:   1
Encode:           0

```

第五步：文件读取成功后，我们可以进行编码和译码了。首先输入 3，程序会对读取的文件进行哈夫曼编码，并且生成名为“EncodedText.huf”的压缩文件存在 debug 目录下；再输入 4，程序会在界面里打印文本所有出现的字符以及它对应的 ASCII 码、权值和哈夫曼编码，同时也会在 debug 目录下生成一个名为“Code.txt”的文本文件，里面存储的是各个字符对应的哈夫曼编码；最后输入 5，程序会对之前生成的 EncodedText.huf 压缩文件译码，并生成“Decode.txt”文件在 debug 目录下。其中额外生成的 Encoed.txt 为存储 01 码的文件，作为编码译码时的中介文件。

```

Text.txt has been encoded.
Encoded filename: Encode Text.huf.

The Huffman Coding Schedule is as follows:
字符(ASCII码) 权值  编码
\n( 10)      42    11001101
( 32)        1583   111
( 33)        6     0010000011

```

```

Text.txt has been encoded.
Encoded filename_A: Encode.txt.
Encoded filename_B: Encode Text.huf.

```

```
y(121)      122      011001
z(122)       6        0010001101

Encoded filename: Text.txt
The Code filename: Code.txt
Completely the encode filename: Decoded.txt
```

第六步：译码完成后可以输入 78 或者 8 来达到清除数据并退出系统或者直接退出系统的目的。

```
-----
                        感谢使用
                    Huffman encoder && decoder
                    -----
程序将于 3s 后退出    2_
```

## 2. 测试结果

(文件夹里 Text.txt 中存放的文本为测试二的内容)

测试一：

Text.txt 中的文本内容为：AAABBBCCCCDEEEEEEEEEFF

结果如下：

读取文件后，文本存储信息如下：

```
文本域如下：
Filename:      Text.txt
Types_num:     6
CharWeight([CHAR-ASCII]WEIGHT):
[ A- 65]   3      [ B- 66]   3      [ C- 67]   4
[ D- 68]   1      [ E- 69]   9      [ F- 70]   2
Length:       22
Initialization: 1
Encode:       1
```

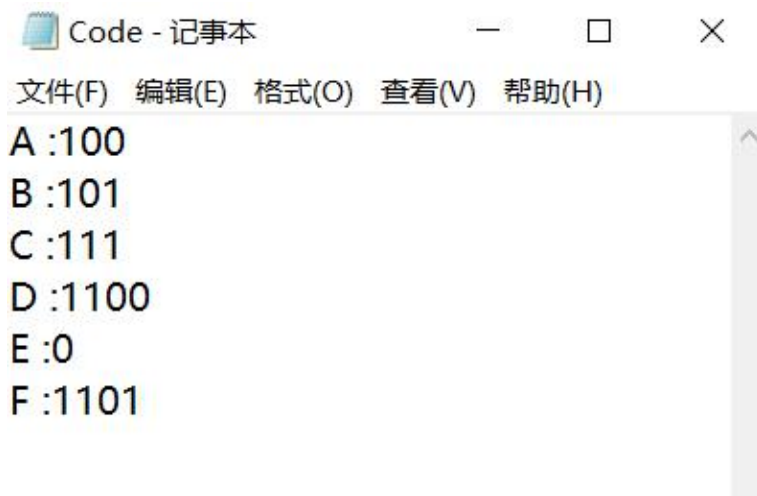
哈夫曼编码如下：

The Huffman Coding Schedule is as follows:

字符(ASCII码)	权值	编码
A( 65)	3	100
B( 66)	3	101
C( 67)	4	111
D( 68)	1	1100
E( 69)	9	0
F( 70)	2	1101

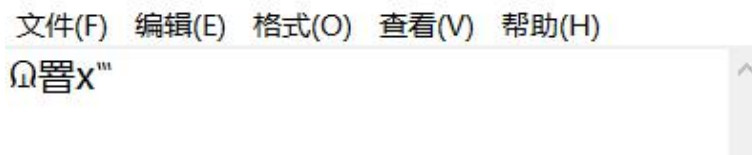
生成的四个文件内容如下:

Code. txt



```
Code - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
A :100
B :101
C :111
D :1100
E :0
F :1101
```

EncodeText. huf (用记事本打开)



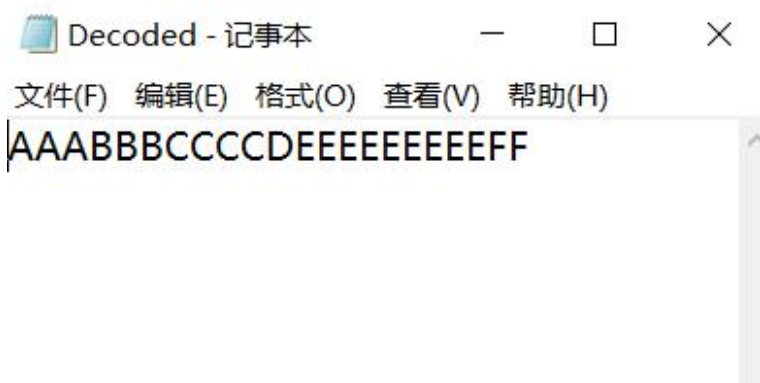
```
EncodeText.huf - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Ω罌x™
```

Encoded. txt 文件

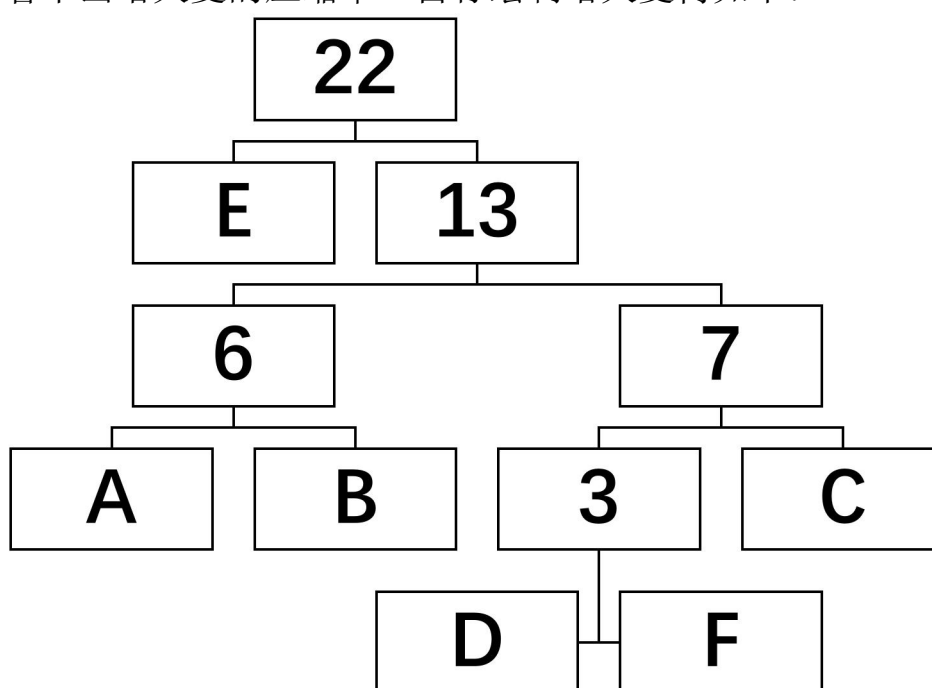


```
Encoded Text.huf - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
100100100101101101111111111111000000000011011101
```

Decoded. txt



对比 Decoded.txt 与 Text.txt 文件，内容完全一致，由于测试数据过少，看不出哈夫曼的压缩率。自行绘制哈夫曼树如下：



符合哈夫曼树定义，所以编码译码均无错误。

测试二：

下面尝试长文本。

Text.txt 中存储的文本内容为马丁路德金演讲稿《I have a dream》原文。其中字符总数=字符数(计空格)+段落数(结尾无换行符)-1=8862+42-1=8903

字符数(不计空格)	7,278
字符数(计空格)	8,862
段落数	42

该图为 words 文档字数统计

测试结果如下：

读取文件后，文本存储信息如下：

```
文本域如下：
Filename:          Text.txt
Types_num:        58
CharWeight([CHAR-ASCII]WEIGHT):
[ \n- 10] 42      [  - 32]1583      [  !- 33] 6
[  "- 34] 18      [  ' - 39] 10      [  , - 44] 77
[  - 45] 9        [  . - 46] 73      [  :- 58] 5
[  ; - 59] 1      [  ? - 63] 2        [  A- 65] 27
[  B- 66] 4        [  C- 67] 6        [  D- 68] 2
[  E- 69] 1        [  F- 70] 4        [  G- 71] 9
[  H- 72] 3        [  I- 73] 24       [  J- 74] 1
[  K- 75] 1        [  L- 76] 16       [  M- 77] 8
[  N- 78] 24       [  O- 79] 3        [  P- 80] 4
[  R- 82] 2        [  S- 83] 4        [  T- 84] 10
[  W- 87] 15       [  Y- 89] 3        [  a- 97] 527
[  b- 98] 105      [  c- 99] 172      [  d-100] 260
[  e-101] 870      [  f-102] 213      [  g-103] 159
[  h-104] 377      [  i-105] 528      [  j-106] 20
[  k-107] 51       [  l-108] 318      [  m-109] 179
[  n-110] 449      [  o-111] 587      [  p-112] 90
[  q-113] 7        [  r-114] 401      [  s-115] 407
[  t-116] 649      [  u-117] 176      [  v-118] 81
[  w-119] 147      [  x-120] 5        [  y-121] 122
[  z-122] 6
Length:           8903
Initialization:   1
Encode:           0
```

哈夫曼编码如下：

The Huffman Coding Schedule is as follows:					
字符(ASCII码)	权值	编码	B( 66)	4	10100000010
\n( 10)	42	11001101	C( 67)	6	00100011100
( 32)	1583	111	D( 68)	2	1010000011101
!( 33)	6	0010000011	E( 69)	1	1010000011111
"( 34)	18	101000010	F( 70)	4	10100000011
'( 39)	10	1100110010	G( 71)	9	1010000111
,( 44)	77	1010100	H( 72)	3	00100000101
-( 45)	9	1010000110	I( 73)	24	00100001
.( 46)	73	1010001	J( 74)	1	001000001000
:( 58)	5	0010000000	K( 75)	1	001000001001
;( 59)	1	10100000111110	L( 76)	16	011000111
?( 63)	2	101000001100	M( 77)	8	1010000000
A( 65)	27	01100010	N( 78)	24	00100010
B( 66)	4	10100000010	O( 79)	3	00100011100
			P( 80)	4	10100000100

R( 82)	2	101000001110	k(107)	51	0110000
S( 83)	4	10100000101	l(108)	318	11000
T( 84)	10	1100110011	m(109)	179	110101
W( 87)	15	011000110	n(110)	449	0011
Y( 89)	3	00100011101	o(111)	587	1001
a( 97)	527	0111	p(112)	90	1100111
b( 98)	105	001001	q(113)	7	0010001111
c( 99)	172	110010	r(114)	401	0000
d(100)	260	01101	s(115)	407	0001
e(101)	870	010	t(116)	649	1011
f(102)	213	00101	u(117)	176	110100
g(103)	159	101011	v(118)	81	1010101
h(104)	377	11011	w(119)	147	101001
i(105)	528	1000	x(120)	5	0010000001
j(106)	20	110011000	y(121)	122	011001
			z(122)	6	0010001101

生成的四个文件如下：

Code. txt

```

Code - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
\n:11001101
:111
!:0010000011
":101000010
':1100110010
,:1010100
-:1010000110
.:1010001
::0010000000
;:1010000011110
?:101000001100
A:01100010
B:10100000010
C:0010001100
D:101000001101
E:1010000011111
F:10100000011
G:1010000111
H:00100000101
I:00100001
J:001000001000
K:001000001001
L:011000111
M:1010000000
N:00100010
O:00100011100
P:10100000100
R:101000001110
S:10100000101
T:1100110011
W:011000110
Y:00100011101
a:0111
b:001001
c:110010
d:01101
e:010
f:00101
g:101011
h:11011
i:1000
j:110011000
k:0110000
l:11000
m:110101
n:0011
o:1001
p:1100111
q:0010001111
r:0000
s:0001
t:1011
u:110100
v:1010101
w:101001
x:0010000001
y:011001
z:0010001101

```

EncodedText. huf (用记事本打开)



EncodedText.huf - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```

py ^U;} OWI3ta81z/4 L^4CoWv|| ^
fM{m=}.<QN{DOhKj{DNGspV:Qn
8>wZrp}(>%z1=P u|qhF%#d|OL
{_|>{(d|{E,-M6@Z/dkwPm 7Y' ~4
O=W00
c^<N7e *#C/^C4 ^ex=P*=v_<K\
c^8]\|A^||<K=Wjs>,|KbUA+=b'n6)'
>%vTdsqN(+3n6y06Gw
"X\|^0^@|iafm1
^z+*We+v@^rkY{O4Uw |];>T.L>
z+*We+n>:Mw 2d|x[qwlc_?eZda#
'Cr0}{ctVwS\->T:O&:0o^^GNic
||xhh. !_@|lUDt\Zv>{={*2?mcm"_

```

Encoded. txt

Encoded - 记事本

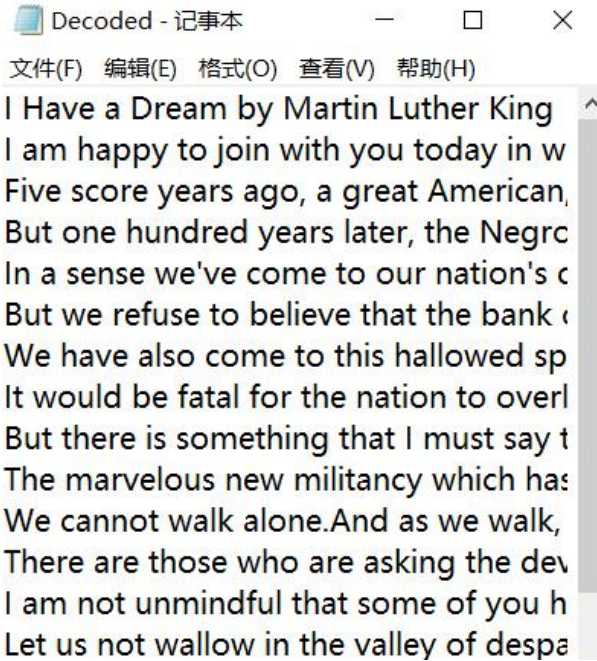
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```

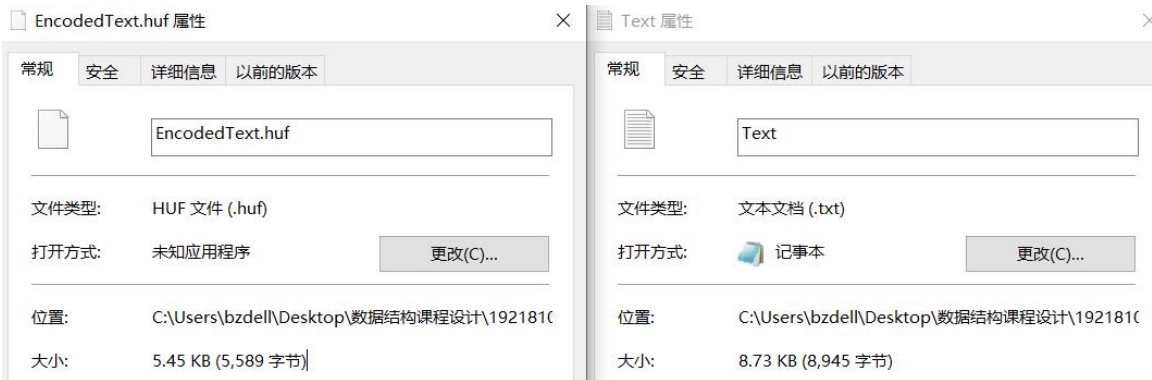
0010000111100100000101011110101 ^
1110001000110010111000111011011
0100100000100111011000011110111
0011101000111100100011100001101
0011000110000101111000001111111
1100110110111110111101101011110
0010111100100101111101000010011
0111111011110110101110010001001
0111001111100111100100001011110
1100100110011111001001011110010
0000101010110111101110001001001
0111100100101011100101011110110
1001110100101111101111110010010
0110000100111110000001111101011

```

Decoded. txt



随机选取某句话作对比，译码后与原文本一致，表示编码译码正确。再对比 EncodedText.huf 与 Text.txt 文件大小，不难观察出确实节省了不少空间。压缩率大约为  $5.45/8.73=62.42\%$ 。



### 3. 调试分析

在调试过程中，遇到过多次重复定义问题，查阅工具书与百度，最终发现是重复引用头文件中的全局变量 HFW 与 CharArray，解决方法是将他们在 MAIN.cpp 中定义，在其他 cpp 文件中 extern 引用。

其实最复杂的便是编码与译码过程了。起初我是直接输出 01 码的，后来发现这样导致 huf 文件反而大很多，因为 0 与 1 当做字符处理。考虑到 ASCII 中 0-127 可以显示出来，于是我将 01 码每 7 位作为一个二进制数，程序中将其转换为十进制数存储在 huf 中。由于我是采用二进制打开 huf 以方便 windows 识别中文字符。但是，如果 01 码位数



不被 7 整除呢？这样会导致压缩文件损失部分数据，于是我设置标志符 X，在编码时将剩下的数后面补零（如 011 按照 0110000 对应十进制数存储），在译码时读到 EOF 时，hint 记录前一个字符，将其对应十进制数转换为二进制数，将最后一个 ‘1’ 后面的 ‘0’ 去掉，再输出到 Encoded.txt 中以便进行后续将 01 码翻译为原文档的操作。这样就可以保证本程序的通用性。

我在写译码部分的时候，考虑过除了从根出发到叶子结点之外的方法，即在读入编码的同时暂时存储在 st 数组中，每读入一个字符就与 HC 编码表做一次比较，直到 st == HC[i]，后输出 CharArray[i] 对应的字符。事实证明这太过繁琐，有很多不必要的判断，所以最后舍去了。该部分代码如下：

```
FILE *ifp = fopen("Encoded Text.huf", "r");
FILE *ofp = fopen("Decoded.txt", "wb");
char ch;
char *st;
st = (char *)malloc(t.ch_quantity * sizeof(char));
int x, y;
unsigned short int Flag;
while((ch = getc(ifp)) != EOF)
{
    Flag = 0;
    for(x = 0; x < t.ch_quantity; x++)
    {
        st[x] = ch;
        st[x+1] = '\0';
        for(y = 0; y < t.ch_quantity; y++)
        {
            if(strcmp(st, HC[y+1]) == 0)
            {
                fputc(CharArray[y], ofp);
                Flag = 1; break;
            }
            else
                continue;
        }
        if(Flag)
            break;
        ch = getc(ifp);
        if(ch == EOF)
            break;
    }
}
free(st);
```

由于在写代码前做好一定的算法设计，所以逻辑上遇到的问题并不多，遇到也会很快解决。以下是我对哈夫曼算法的思考。

在本顺序存储哈夫曼算法中，其时间复杂度主要取决于在构造哈夫曼树的时候使用的何种查找/排序方法（Select 函数），本程序中为顺序查找最小节点。

查找的时候循环约：

$$2(n + n+1 + n+2 + \cdots + 2n-2) = 3n^2 - 5n + 2$$

构建中要循环查找  $n$  次，总共操作不会超过：

$$3n^3 - 5n^2 + 2n$$

所以时间复杂度为  $O(n^3)$ ，在处理字符数多的情况下，效率较低。

我的算法改进设想有两种，都是用链队列作为存储空间。

第一种是先将  $n$  个终端节点存储在优先队列中；如果队内节点大于 1，则：

1. 移除权值最小的节点
2. 构造移除结点的双亲结点
3. 双亲结点入队列

最后队列中留下的为根结点。

假设移除结点时用的算法时间复杂度为  $O'$ ，所以这种算法时间复杂度为  $O(O' \log n)$ 。

第二种是使用两个队列 fir 与 sec，fir 存储  $n$  个终端节点，sec 存储两两权重的和节点，这样可以保证 sec 永远是从小到大排序。

具体步骤如下：在 fir 存储中，先对节点快速排序  $O(n \log n)$  从小到大入队列；如果 fir 队列内的节点数  $> 1$ ，则：

1. 从 fir 队列前端移除两个最低权重的节点
2. 将移除的两个节点权重相加合成一个新节点
3. 加入 sec 队列

最后在 fir 队列的节点为根节点。

这样最后构建哈夫曼树时间复杂度下降至  $O(n)$ ，虽然加上之前的排序总时间复杂度为  $O(n \log n)$ ，但对比前两种，明显较优。

## 五. 总结（收获与体会）

个人感觉，在本次课程设计中，我锻炼到了写代码时候的模块设计，算法分析技巧，遇到了不少从来没见过的 errors，并且尝试去解决。另外我也锻炼了 c 语言中有关文件使用的函数，暑假中阅读的《C 语言程序设计现代方法》中的内容也得到了实用。我认为写程序就是不断解决问题的过程，解决项目的最终问题，解决写程序中出现的問題。所以我一定会全面考虑，写之前大致列个提纲，估计要处理的问题以及其解决方案（算法），各个函数的封装等，最后再开始写程序，当然完成后的调试修改，数据处理也是必不可少的环节。

在程序设计中，除了高效精简，我认为有一个友好的界面是很重要的，所以我在这方面废了些功夫。不过，之前数据结构课在写“停车场管理系统”的时候我就积累一部分经验，对界面显示，程序控制，函数封装上有一定的练习，所以本次课程设计这方面进行的比较顺利。

既然在大一下学期学习了数据结构这门课，学到了顺序表、链表、栈、队列、串、堆、树、图等结构，在解决问题的时候就要选择最优的结构加以使用；又例如哈夫曼（贪心）算法、各种查找排序算法、迪杰斯卡算法、克鲁斯卡尔算法、普利姆算法等等，要因题适用，从而提高自己对它们的实践能力。

本次课程设计中，我通过查阅资料，学到了不少关于 c 语言语法、编程技巧以及数据结构算法的新知识，在“调试分析”模块中我已经列举了，这里就不做重复说明。我会在今后学习中，拓展课外学习，阅读有关算法分析的书目，提高个人学习能力。以上就是我对本次课程设计的总结。

## 六. 源程序

## 头文件

### file.h

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define ASCII 256
#define NEWLINE printf("\n")    /*换行*/
#define LOOP while(1)          /*死循环*/
#define SPACE printf(" ")      /*十个空格*/
//ASCII 码表 256 个字符取值
typedef struct{
    char filename[50];          //记录文件名
    char *character;            //复制 txt 文本内容
    unsigned int *letter_w;     //各种字符的权值，标号为 ASCII
    int ch_quantity;           //出现字符种类个数
    int length;                 //文本长度
    short int flag;             //初始化后 flag=1，否则 flag=0
    short int mark;             //未编译 mark=0，编译后 mark=1
}Text;
```

### huffmantree.h

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

//最大字符数
#define MAX_N 50000

//动态分配数组存储赫夫曼树
typedef struct{
    unsigned int weight;
    unsigned int parent,lchild,rchild;
}HTNode,*HuffmanTree;

//动态分配存储赫夫曼编码表
typedef char **HuffmanCode;    //顺序存放对应编码
```

### operand.h

```
#include"huffmantree.h"
#include"file.h"

#include<stdio.h>
#include<stdlib.h>
```

```

#include<string.h>
#include<windows.h>
#include<math.h>

//作为 Sleep 函数参数
#define second 990

//用队列存储操作数
typedef char ElemType;
typedef struct QNode{
    ElemType    data;
    struct QNode *next;
}QNode,*QueuePtr;
typedef struct{
    QueuePtr front;           //头指针
    QueuePtr rear;           //尾指针
    unsigned short int length; //队列长度
    unsigned short int flag;   //初始化后 flag=1, 否则 flag=0
}LinkQueue;

/*****  队列相关函数  *****/
int InitQueue(LinkQueue &Q);           //初始化队列
int EnLQueue(LinkQueue &Q,ElemType e); //入队列
int DeQueue(LinkQueue &Q,ElemType &e); //出队列
void DestroyQueue(LinkQueue &Q);       //销毁队列
int GetHead(LinkQueue Q,ElemType &e);  //取队头元素
int Empty(LinkQueue Q);                 //判断队列是否为空
int PrintQueue(LinkQueue Q);            //遍历队列
int LenQueue(LinkQueue Q);              //队列长度

/*****  操作及显示相关函数  *****/
int  Interface_A();                     //第一显示界面
void Interface_B();                     //第二显示界面
void Initialization(LinkQueue &Q, Text &T); //初始化
int  Operand_correct(LinkQueue Q, Text T); //操作数序列输入判断函数
void Operate();                         //主操作函数
void EmptyData(HuffmanTree &HT, HuffmanCode &HC, Text &T);
                                         //清空程序数据
void HFT_Code(HuffmanTree &HT, HuffmanCode &HC, Text &t);
                                         //赫夫曼树封装函数
void PrintCode(HuffmanCode HC, Text t);  //打印并生成赫夫曼码表
void HFT_Decode(HuffmanTree HT, HuffmanCode HC, Text &t); //译码

/*****  文件相关函数  *****/
void InitText(Text &T);                 //初始化文本
void DestroyText(Text &T);              //销毁文本
void ReadText(Text &T);                 //从 txt 文件逐个字符读入文本

```

```

void HUF();           //生成压缩文件
void GenerateText(Text T); //译码后生成目标文件
void PrintText(Text T);  //打印文本标志域信息
void FileName(Text &T);  //输入文件名
void Read_File(Text &T); //打开并读取文件

/***** 赫夫曼构造函数 *****/
void HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, unsigned int *w, unsigned n);
//w 存放 n 个字符的权值(w>0)，构造赫夫曼树 HT，并求出 n 个字符的赫夫曼编码 HC
void Select(HuffmanTree HT, unsigned y, unsigned &s1, unsigned &s2);
//在 HT[1...n]选择 parent 为 0 且 weight 最小的两个结点，其序号为 s1, s2 (s1<=s2)

```

## Cpp 文件

### FILE.cpp

```

#include"operand.h"
extern unsigned short int *CharArray;
extern unsigned int *HFW;

void InitText(Text &T)
//初始化文本
{
    T.character = (char *)malloc(MAX_N * sizeof(char));
    T.length = 0;
    T.ch_quantity = 0;
    T.mark = 0;
    T.flag = 1;
    T.filename[0] = '\0';    T.filename[1] = '\0';
    T.letter_w = (unsigned int *)malloc((ASCII+1) * sizeof(unsigned int));
    for(int i = 0; i <= ASCII; i++)
        T.letter_w[i] = 0;
}

void DestroyText(Text &T)
//销毁文本
{
    free(T.character);
    free(T.letter_w);
    free(CharArray);
    free(HFW);
    T.filename[0] = '\0';
    T.flag = 0;
    T.length = T.ch_quantity = T.mark = 0;
}

void PrintText(Text T)

```

//打印文本存储信息

```
{
    NEWLINE;
    printf("        文本域如下: ");    NEWLINE;
    SPACE; printf("Filename:        "); printf("%s", T.filename);    NEWLINE;
    SPACE; printf("Types_num:        "); printf("%d", T.ch_quantity);NEWLINE;
    SPACE; printf("CharWeight([CHAR-ASCII]WEIGHT):");    NEWLINE;
    for(int i = 0, j = 0; i < ASCII; i++)
    {
        if(T.letter_w[i] > 0)
        {
            SPACE; j++;
            if(i == (int)'\n')
                printf("[\\n-%3d]%4d    ", i, T.letter_w[i]);
            else if(i == (int)'\t')
                printf("[\\t-%3d]%4d    ", i, T.letter_w[i]);
            else
            {
                printf("[%2c-%3d]%4d    ", i, i, T.letter_w[i]);
            }
            if(j%3 == 0)
                NEWLINE;
        }
    }
    if(j == 0)
    {
        SPACE; printf("NULL");    NEWLINE;
    }
    if(j%3 != 0)
        NEWLINE;
    SPACE; printf("Length:        ");    printf("%d", T.length);    NEWLINE;
    SPACE; printf("Initialization: "); printf("%d", T.flag);    NEWLINE;
    SPACE; printf("Encode:        ");    printf("%d", T.mark);    NEWLINE;
    NEWLINE;
}
```

void FileName(Text &T)

//输入文件名

```
{
    NEWLINE;
    SPACE;
    printf("Input your filename(xxx.txt): ");
    char ch;    int i=0;
    ch = getchar();
    while(ch != '\n')
    {
        T.filename[i] = ch;
```

```

        i++;
        ch = getchar();
    }
    T.filename[i] = '\0';
    flushall();
    NEWLINE;
}

void Read_File(Text &T)
{
    char ch;

    FileName(T);

    SPACE; printf("The program is reading the FILE: (%s).", T.filename);
    NEWLINE;
    FILE *fp = fopen(T.filename, "r");
    if(fp == NULL)
    {
        SPACE;
        printf("Failed to open the FILE: (T.filename)."); NEWLINE;
        T.filename[0] = '0'; T.filename[1] = '\0';
        NEWLINE;
        return ;
    }
    while((ch =getc(fp)) != EOF)
    {
        T.character[T.length] = ch;
        T.length++;
        if(T.letter_w[(int)ch] == 0)
            T.ch_quantity++;
        T.letter_w[(int)ch]++;
    }
    fclose(fp);
    SPACE; printf("FILE: (%s) read completely. ", T.filename); NEWLINE;
    NEWLINE;
}

```

## HFT. cpp

```

#include"operand.h"
extern unsigned short int *CharArray;
extern unsigned int *HFW;
extern short int X;

//封装函数
void HFT_Code(HuffmanTree &HT, HuffmanCode &HC, Text &t)
{

```



```

if(t.filename[0] == '0')
{
    SPACE; printf("Error: NO FILE\n");
    return;
}

unsigned short int i, j;

HFW = (unsigned int *)malloc(t.ch_quantity * sizeof(unsigned int));
CharArray = (unsigned short int *)malloc((t.ch_quantity+1) * sizeof(unsigned short
int));
CharArray[t.ch_quantity]='\0';
for(i = 0, j = 0; j < ASCII; j++)
{
    if(t.letter_w[j] > 0)
    {
        HFW[i] = t.letter_w[j];
        CharArray[i] = (unsigned short int)j;
        i++;
    }
}
//将 t.letter_w 中的有效权值(大于 0)复制到 w 中, 使有效权值连续
//存储 t.letter_w 中表示的有效字符(权值大于 0)
HuffmanCoding(HT, HC, HFW, t.ch_quantity);
t.mark = 1;

//生成 01 码文件
FILE *in_fp = fopen(t.filename, "r");
FILE *out_fp = fopen("Encoded.txt", "wb");
char ch;
int x;
while((ch = getc(in_fp)) != EOF)
{
    for(x = 0; x < t.ch_quantity; x++)
    {
        if(ch == CharArray[x])
        {
            fputs(HC[x+1], out_fp);
            break;
        }
        else
            continue;
    }
}

fclose(in_fp);
fclose(out_fp);

```

```

//将 X 位 01 码转换为一个 char 存储在压缩文件中
FILE *fp = fopen("Encoded.txt", "r");
FILE *hp = fopen("EncodedText.huf", "wb");

int Hflag = 64;
int num = 0;
char chput = 0;

ch = getc(fp);
while(!feof(fp))
{
    num++;
    chput += Hflag * (ch - '0');
    Hflag /= 2;
    X = 0;
    if(num % 7 == 0)
    {
        fputc(chput, hp);
        chput = 0;
        Hflag = 64;
        X = 1;
    }
    ch = getc(fp);
    if(X == 0 && ch == EOF)
    {
        fputc(chput, hp);
        break;
    }
}

fclose(fp);
fclose(hp);

NEWLINE;
SPACE; printf("%s has been encoded.", t.filename); NEWLINE;
SPACE; printf("Encoded filename_A: Encode.txt."); NEWLINE;
SPACE; printf("Encoded filename_B: Encode Text.huf."); NEWLINE;
NEWLINE;
}

//w 存放 n 个字符的权值 (w>0)，构造赫夫曼树 HT，并求出 n 个字符的赫夫曼编码 HC
void HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, unsigned int *w, unsigned n)
{
    unsigned i, m;
    unsigned s1, s2;
    HuffmanTree p;

```

```

m=2*n-1;
//分配结点空间
HT=(HuffmanTree)malloc((m+1)*sizeof(HTNode));
//初始化(不使用0结点以方便计算)
//0结点存放最大权值以方便之后取s1 s2
HT->weight=MAX_N;
HT->lchild=HT->rchild=HT->parent=0;
for(p=HT+1,i=1;i<=n;i++,p++,w++)
{
    p->weight=*w;
    p->lchild=p->rchild=p->parent=0;
}
for(;i<=m;i++,p++)
{
    p->lchild=p->rchild=p->parent=p->weight=0;
}
//建造哈夫曼树
for(i=n+1;i<=m;i++)
{
    Select(HT,i-1,s1,s2);
    HT[s1].parent=i; HT[s2].parent=i;
    HT[i].lchild=s1; HT[i].rchild=s2;
    HT[i].weight=HT[s1].weight+HT[s2].weight;
}

/*---从叶子结点到根逆向求每个字符的赫夫曼编码---*/
char *cd;
unsigned start,c,f;
HC=(HuffmanCode)malloc((n+1)*sizeof(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].lchild==c)cd[--start]='0';
        else cd[--start]='1';
    }
    HC[i]=(char *)malloc((n-start)*sizeof(char));
    strcpy(HC[i],&cd[start]);
}
free(cd);
}

```

//在 HT[1...n]选择 parent 为 0 且 weight 最小的两个结点，其序号为 s1, s2

```

(s1.weight<=s2.weight)
void Select(HuffmanTree HT, unsigned y, unsigned &s1, unsigned &s2)
{
    unsigned x;
    s1=s2=0;
    for(x=1;x<=y;x++)
    {
        if(HT[x].parent==0)
        {
            if(HT[x].weight<HT[s1].weight)
                s1=x;
            else continue;
        }
        else continue;
    }
    for(x=1;x<=y;x++)
    {
        if((HT[x].parent==0)&&(x!=s1))
        {
            if(HT[x].weight<HT[s2].weight)
                s2=x;
            else continue;
        }
        else continue;
    }
}

```

//打印并生成赫夫曼码表

```
void PrintCode(HuffmanCode HC, Text t)
```

```

{
    int i;

    if(HFW == NULL || CharArray == NULL)
    {
        SPACE;
        printf("Error: NO CODED");
        NEWLINE;    return;
    }
}

```

//打印并生成密码表

```
FILE *codep = fopen("Code.txt", "wb");
```

```

NEWLINE;
SPACE; printf("The Huffman Coding Schedule is as follows:");    NEWLINE;
SPACE; printf("字符(ASCII 码)    权值    编码");                NEWLINE;
for(i = 0; i < t.ch_quantity; i++)
{

```

```

SPACE;
if(CharArray[i] == (int)'\n')
    printf("\n(%3d)    %d %s", CharArray[i], HFW[i], HC[i+1]);
else if(CharArray[i] == (int)'\t')
    printf("\t(%3d)    %d %s", CharArray[i], HFW[i], HC[i+1]);
else
{
    printf("%2c(%3d)    %d %s", CharArray[i], CharArray[i], HFW[i],
HC[i+1]);
}
NEWLINE;    NEWLINE;

//写入 Code.txt
if(CharArray[i] == (int)'\n')
    fputs("\n:", codep);
else if(CharArray[i] == (int)'\t')
    fputs("\t:", codep);
else
{
    fputc(CharArray[i], codep); fputs(" :", codep);
}
fputs(HC[i+1], codep);
fputs("\n", codep);
}
}

```

//译码

```

void HFT_Decode(HuffmanTree HT, HuffmanCode HC, Text &t)
{
    if(t.flag == 0 || t.mark == 0)
    {
        SPACE; printf("Error: NO INIT/ENCODE");    NEWLINE;
        return ;
    }
    NEWLINE;
    SPACE; printf("Original filename: %s", t.filename);    NEWLINE;
    SPACE; printf("The Code filename: Code.txt");    NEWLINE;
    SPACE; printf("Encoded filename: EncodedText.huf");    NEWLINE;
    SPACE; printf("0-1code filename: Encoded.txt");    NEWLINE;

    FILE *ifp = fopen("EncodedText.huf", "rb");
    FILE *cfp = fopen("Encoded.txt", "wb");
    FILE *ofp = fopen("Decoded.txt", "wb");

    char ch;
    char *st;
    int x, y;

```

```

st = (char *)malloc(t.ch_quantity * sizeof(char));
unsigned int root, p;

//将 huf 压缩文件转换为 01 码存储在 cfp 指向的文件中 (默认为 Encoded.txt)
int hint;
int i, w;
char hcode[8];

ch = fgetc(ifp);
while(!feof(ifp))
{
    //hcode 数组置零
    for(i = 0; i < 7; i++)
        hcode[i] = '0';
    hcode[7] = '\0';
    //将 ch 转换为 01 码暂时存储在 hcode 中
    hint = (int)ch;
    ch = fgetc(ifp);
    if(feof(ifp) && !X)
    {
        w = 6;
        while(hint != 0)
        {
            hcode[w--] = (hint%2) + '0';
            hint = hint/2;
        }
        if(hcode[6] == '0')
        {
            for(int h = 6; h > 0; h--)
            {
                if(hcode[h-1] == '1')
                {
                    hcode[h] = '\0';
                    break;
                }
            }
        }
        fputs(hcode, cfp);
        break;
    }
    w = 6;
    while(hint != 0)
    {
        hcode[w--] = (hint%2) + '0';
        hint = hint/2;
    }
    fputs(hcode, cfp);
}

```

```

}
fclose(cfp);
fclose(ifp);

//译码
//找到根结点
for(x = 1; x <= (2*t.ch_quantity-1); x++)
{
    if(HT[x].parent == 0 && HT[x].weight != 0)
    {
        root = x;    break;
    }
    else
        continue;
}
if(x == 2*t.ch_quantity)
{
    NEWLINE;    SPACE;    printf("Error.");    return;
}

FILE *cfp2 = fopen("Encoded.txt", "r");
while((ch = getc(cfp2)) != EOF)
{
    x = 0;
    p = root;
    while(HT[p].lchild != 0 || HT[p].rchild != 0)
    {
        if(ch == '0')
        {
            st[x++] = ch;
            p = HT[p].lchild;
        }
        else
        {
            st[x++] = ch;
            p = HT[p].rchild;
        }
        if(HT[p].lchild != 0 || HT[p].rchild != 0)
            ch = getc(cfp2);
        if(ch == EOF)
            break;
    }
    st[x] = '\0';
    for(y = 0; y < t.ch_quantity; y++)
    {
        if(strcmp(st, HC[y+1]) == 0)

```

```

        {
            fputc(CharArray[y], ofp);
            break;
        }
    }
}
fclose(cfp2);
fclose(ofp);

SPACE; printf("Completely the decode filename: Decoded.txt"); NEWLINE;
NEWLINE;
}

```

## MAIN. cpp

```
#include"operand.h"
```

```

unsigned short int *CharArray;           //顺序存放字符数组
unsigned int *HFW;                      //存放连续的有效权值
short int X;                            //01 码能否被 7 整除的标志

```

```

void END()
{
    system("cls"); //清屏
    NEWLINE;
    printf("-----|"); NEWLINE;
    printf("-----|"); NEWLINE;
    printf("          感谢使用          |"); NEWLINE;
    printf("      Huffman encoder && decoder      |"); NEWLINE;
    printf("-----|"); NEWLINE;
    printf("      程序将于 3s 后退出      ");
    for(int i = 3; i >=1; i--)
    {
        printf("%d", i);
        Sleep(second);
        printf("\b");
    }
    system("cls"); //清屏
    exit(EXIT_SUCCESS);
}

```

```

int main(void)
{
    Operate();
    END();
    return 0;
}

```



## OPERAND QUEUE.cpp

```
#include "operand.h"
```

```
int InitQueue(LinkQueue &Q)
```

```
//初始化队列
```

```
{
    Q.front = Q.rear = (QueuePtr)malloc(sizeof(QNode));
    if(!Q.front)
    {
        printf("存储分配失败\n");
        exit(EXIT_FAILURE);
    }
    Q.front->next = NULL;
    Q.lengh = 0;    Q.flag = 1;
    return 1;
}
```

```
int EnLQueue(LinkQueue &Q, ElemType e)
```

```
//元素 e 入队列
```

```
{
    QueuePtr p;
    p=(QueuePtr)malloc(sizeof(QNode));
    if(!p)
    {
        printf("存储分配失败\n");
        exit(EXIT_FAILURE);
    }
    p->data = e; p->next = NULL;
    Q.rear->next = p;
    Q.rear = p;
    Q.lengh++;
    return 1;
}
```

```
int DeQueue(LinkQueue &Q, ElemType &e)
```

```
//出队列，删除队头元素，并用 e 返回其值
```

```
{
    if(Q.front == Q.rear)
    {
        printf("Error\n");
        return 0;
    }//若队列空, 返回 Error
    QueuePtr p;
    p = Q.front->next;
    e = p->data;
    Q.front->next = p->next;
```

```

        if(Q.rear == p)
            Q.rear = Q.front; //对于链队列只有一个元素结点的情况要同时修改队尾指针
        free(p);
        Q.lengh--;
        return 1;
    }

```

```

void DestroyQueue(LinkQueue &Q)
//销毁队列
{
    if(Q.front->next == NULL)
        return;
    while(Q.front)
    {
        Q.rear = Q.front->next;
        free(Q.front);
        Q.front = Q.rear;
    }
    Q.lengh = 0;
    Q.flag = 0;
}

```

```

int GetHead(LinkQueue Q, ElemType &e)
//取队头元素，并储存在 e 中
{
    if(Empty(Q))
    {
        printf("队列为空\n");
        return 0;
    }
    QueuePtr p;
    p = Q.front->next;
    e = p->data;
    return 1;
}

```

```

int Empty(LinkQueue Q)
//判断队列是否为空
// 1 为空
// 0 为非空
{
    if(Q.front==Q.rear)
        return 1;
    return 0;
}

```

```

int PrintQueue(LinkQueue Q)

```

```

//遍历队列
{
    if (Empty(Q))
    {
        printf("该队列为空\n");
        return 0;
    }
    else
    {
        QueuePtr p;
        p = Q.front->next;
        while (p)
        {
            printf("%c ", p->data);
            p = p->next;
        }
        printf("\n");
    }
    return 1;
}

```

```

int LenQueue(LinkQueue Q)
//队列长度
{
    int i = 0;
    QueuePtr p;
    p = Q.front->next;
    if (Empty(Q))
        return 0;
    else
    {
        while (p)
        {
            i++;
            p = p->next;
        }
    }
    return i;
}

```

OPERAND.cpp

```

#include "operand.h"
extern unsigned short int *CharArray;
extern unsigned int *HFW;

```

```

int Interface_A()
{

```

```

char n;
NEWLINE;
printf("-----");
printf("赫夫曼 编/译 码器"); NEWLINE;
printf("作者: 19218101"); NEWLINE;
printf("Huffman encoder / decoder"); NEWLINE;
printf("Designer: 19218101"); NEWLINE;
printf("-----");
printf("使用说明"); NEWLINE;
printf("本程序可对一个文本文件中的字符进"); NEWLINE;
printf("行赫夫曼编码并且生成编码文件。也"); NEWLINE;
printf("可将编码文件译码还原为文本文件。"); NEWLINE;
printf("Direction for use"); NEWLINE;
printf("Huffman encoding a file(.txt) to"); NEWLINE;
printf("generate a file(Encoded txt.huf)."); NEWLINE;
printf("And also decode it and restore it"); NEWLINE;
printf("to a file(Decoded.txt)."); NEWLINE;
printf("-----");
printf("Huffman encoder && decoder by 19218101"); NEWLINE;
printf("Compiled on %s at %s |", __DATE__, __TIME__); NEWLINE;
printf("-----");
printf("完成阅读, 是否进入下一步操作? (Y/N): ");
LOOP
{
    n = getchar();
    if(n == 'Y' || n == 'y')
    {
        system("cls"); //清屏
        return 1;
    }
    else if(n == 'N' || n == 'n')
        return 0;
    else
    {
        printf("输入格式错误, 请重新输入"); NEWLINE;
        printf("完成阅读, 是否进入下一步操作? (Y/N): ");
        fflush(); //清空缓冲区
    }
}

```

```

    }
}

void Interface_B()
{
    NEWLINE;
    printf("-----");
    printf(" "); NEWLINE;
    printf(" "); NEWLINE;
    printf(" Huffman encoder / decoder "); NEWLINE;
    printf(" "); NEWLINE;
    printf("-----");
    printf(" "); NEWLINE;
    printf(" "); NEWLINE;
    printf(" 键位说明 "); NEWLINE;
    printf(" 1. 打印文本存储信息 "); NEWLINE;
    printf(" 2. 打开并读取文件 "); NEWLINE;
    printf(" 3. 赫夫曼编码并生成压缩文件 "); NEWLINE;
    printf(" 4. 打印码表并存储在 code.txt 中 ");
    NEWLINE;
    printf(" 5. 对编码后生成的文件译码并生成 txt 文件 ");
    NEWLINE;
    printf(" ");
    printf(" 6. 程序初始化 "); NEWLINE;
    printf(" 7. 清空数据 "); NEWLINE;
    printf(" 8. 退出程序 "); NEWLINE;
    printf(" "); NEWLINE;
    printf("-----");
    printf("操作数可以连续输入，输入完成后按 回车 结束输入"); NEWLINE;
    printf("例如：输入 [72314568 回车] 或者 [8 回车]"); NEWLINE;
    printf("输入操作数序列(1 - 8): ");
    fflush(); //清空缓冲区
}

```

```

void Initialization(LinkQueue &Q, Text &T)
{
    NEWLINE;
    SPACE; printf("Initializing"); NEWLINE;
    InitQueue(Q); //操作数队列初始化
    InitText(T); //文本结构体初始化
    SPACE; printf("Completely"); NEWLINE;
    NEWLINE;
}

```

```

void EmptyData(HuffmanTree &HT, HuffmanCode &HC, Text &T)
//清空文本数据
{
    NEWLINE;
    SPACE; printf("Destorying"); NEWLINE;
}

```

```

    DestroyText(T);
    SPACE; printf("Completely");    NEWLINE;
    NEWLINE;
}

void Operate()
{
    LinkQueue operand;           //定义操作数队列 operand
    ElemType in,out;             //记录进出队列的值，逐次更新
    Text t;                      //定义存储文本结构体
    HuffmanTree HFTree;          //定义赫夫曼树
    HuffmanCode HFCode;          //定义赫夫曼编码

    t.flag = 0; operand.flag = 0;
                                //标明未初始化

    /***第一显示界面***/
    if(!Interface_A())
        return;
    /***第二显示界面***/
    Interface_B();
    /***初始化***/
    InitQueue(operand);          //操作数队列初始化
    InitText(t);                 //文本结构体初始化

    /***死循环***/
    LOOP
    {
        /***进队列***/
        in = getchar();
        while(in != '\n')
        {
            EnLQueue(operand, in);
            in = getchar();
        }
        flushall();              //清空缓冲区
        printf("        你输入的操作数序列: ");
        PrintQueue(operand);

        /***出队列***/
        while(operand.lengh > 0)
        {
            DeQueue(operand, out);
            switch(out)
            {
                case '1': PrintText(t); break;
                case '2': Read_File(t); break;
                case '3': HFT_Code(HFTree, HFCode, t); break;
            }
        }
    }
}

```

```

        case '4': PrintCode(HFCode, t);          break;
        case '5': HFT_Decode(HFTree, HFCode, t);break;
        case '6': Initialization(operand, t);   break;
        case '7': EmptyData(HFTree, HFCode, t); break;
        case '8': return;
        default :
            {
                DestroyQueue(operand);
                InitQueue(operand);
                SPACE; printf("请输入 1 - 8 的数字序列");  NEWLINE;
            }
    }
}
flushall();
DestroyQueue(operand);  InitQueue(operand);
printf("          请继续输入操作序列(1 - 8): ");
}
}

```