**数据结构课程设计报告**

**设计题目：**  哈夫曼编/译码器

**班 级：**

**学 号：**

**姓 名：**

**数据结构课程设计报告内容**

1. **课程设计题目**

哈夫曼编/译码器：

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) 读入一篇不少于5000字符的英文短文(source.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，其中求最小权值要求用堆实现；

(4）利用已建好的哈夫曼树，将文本文件进行编码，将编码结果保存到文件中(code.dat)。

(5）根据相应哈夫曼编码，对编码后的文件进行译码，将code.dat文件译码为txt文件，与原文件(source.txt)进行比较。

1. **算法设计思想**

**哈夫曼编码贪心算法:**

**贪心算法：**

在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的仅是在某种意义上的局部最优解。贪心算法没有固定的算法框架，算法设计的关键是贪心策略的选择。必须注意的是，贪心算法不是对于所有问题都能得到整体最优解，选择的贪心策略必须具有无后效性，即某个状态以后的过程不会影响以前的状态，只与当前的状态有关。所以对所采取的贪心策略一定要仔细分析是否满足无后效性。

**基本步骤：**

1. 建立数学模型来描述问题。
2. 把求解的问题分成若干个子问题。
3. 对每一子问题求解，得到子问题的布局最优解。
4. 把子问题的解局部最优解合成原来解问题的一个解。

**哈夫曼编码：**

根据每个元素出现的频率由小到大进行排序，总是选择权值最小的两个节点相加合成一个新的节点，直至最后剩一个节点，此节点就为哈夫曼树。对于哈夫曼树的左节点链接’0’右节点链接’1’，从树根到树叶依次记录所有字母编码。

本实验主要采用哈夫曼树来构建编码。哈夫曼树是一种带权路径最短的二叉树，成为最优二叉树。

哈夫曼编码：

假如有数据A，权值为2。数据B，权值为5。数据C，权值为7，数据D，权值为15。

在构建哈夫曼树时，首先需要定义结点的属性。一个结点作为子节点有对应的数据以及该数据的权值，即出现的频数。作为根节点有左子树，右子树两个子节点。

在构建哈夫曼树时，要选择两个权值最小的树（数据）作为左右子树，然后其新树（根节点）的权值等于这两个子树的权值之和，并且保持左子树的权值小于等于右子树。在新树生成后，就删除这两颗左右子树，把新树作为一个对象添加到总数据中。重复这几部操作直至只剩一颗树，就得到了哈夫曼树。



比如在ABCD中，A与B的权值最小，则选择A作为左子树，B作为右子树。因为A的权值小于B，所以选择B作为右子树，A作为左子树。合并A和B后产生权值为7的新树（假设他为E）。然后在剩下的数据中最小的两个数据为C和E，重复比较、合并得到新树F（权值为14）最后将F与D合并得到哈夫曼树。

如果定义跟到每一个叶子节点都是一条路径，而通往左子树的路径为0，通往右子树的路径为1，那么可以得到每个叶子节点的字符编码。而构建哈夫曼树的过程就是对于这些数据的编码过程。（顺序为由上到下。）

如图可以直接得出A的编码为110，B的编码111，C的编码为10，D的编码为0。

编码后需要对该哈夫曼树解码来得到每个数据对应的编码。

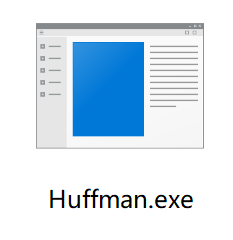
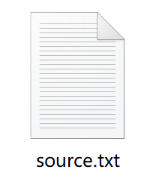
**3．程序结构**



主程序首先调用ReadFile函数，读取source.txt的内容，对于ASCII编码内的每个字符进行计数，返回文件包含的字符类别数。随后调用线性表SqList及相关函数，存储哈夫曼编码的节点权值和字符。然后进行哈夫曼编码，生成对应的哈夫曼树及哈夫曼编码，在生成哈夫曼树时利用最小堆获得权值最小的两个点。然后调用showHC（）输出生成的哈夫曼编码。调用WriteCode（）输出包含哈夫曼编码信息的文件huffmancode.txt。调用ReadandCompressFile2()读取哈夫曼编码文件huffmancode.txt，利用该文件对原文件source.txt进行压缩输出文件code.dat，输出压缩率和文件长度。调用GetMessage（），获得huffmancode.txt的哈夫曼编码结构，然后通过ReadFile1（）获得source.txt的文件长度，通过DeCompressFile（）对code.dat进行解码解压缩为自定名字文件，然后返回文件长度，最终得到完整率。

**4． 实验结果与分析**

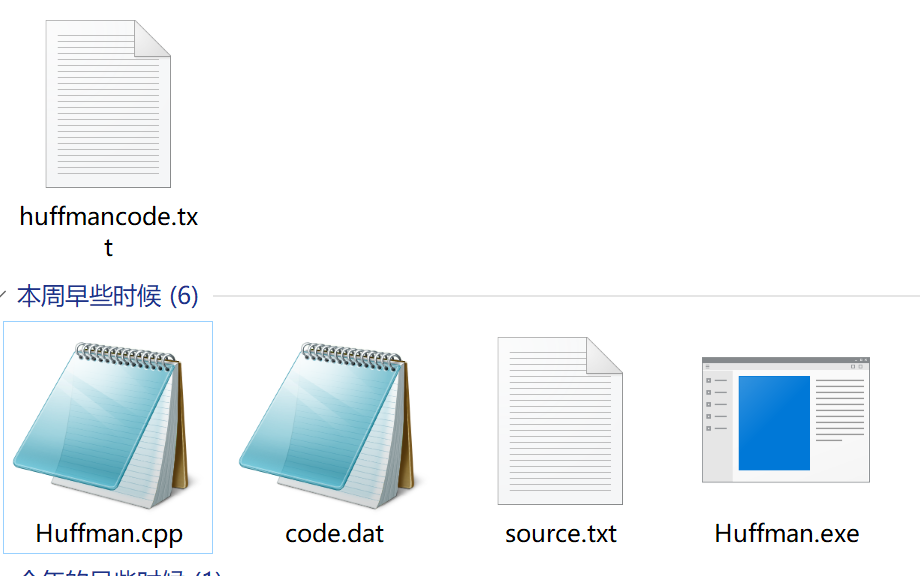
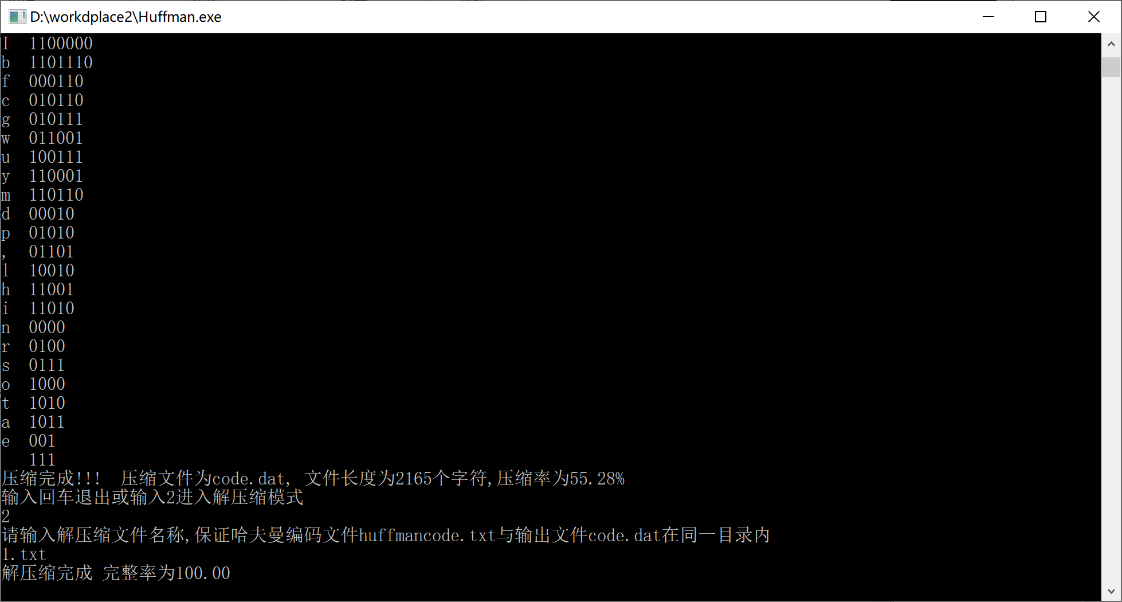
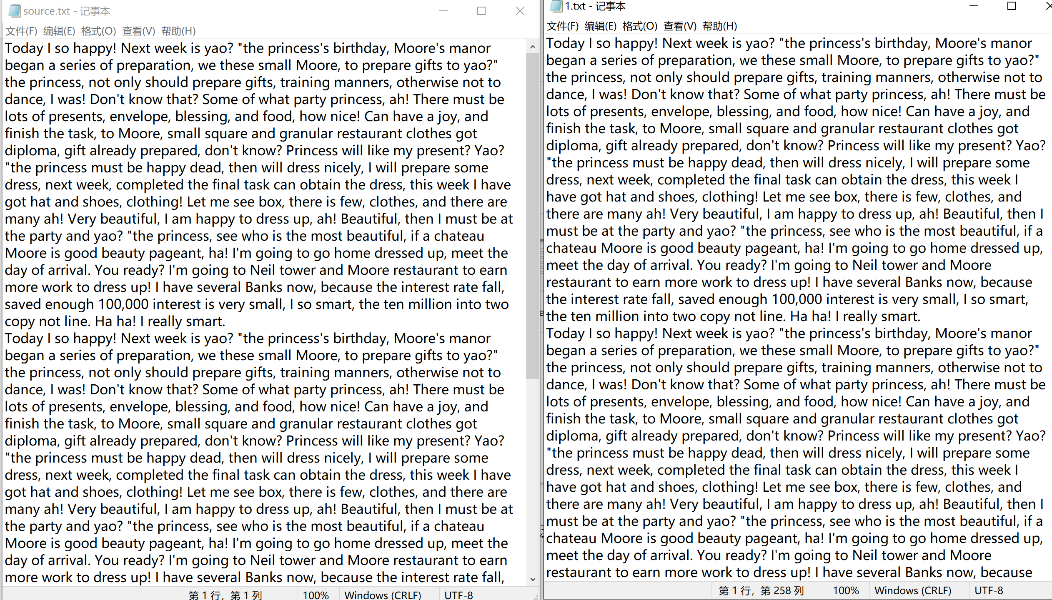
**4.1 用户使用说明**

1. 双击打开文件Huffman.exe 
2. 确定文件同目录下含有输入文件source.txt 并且该文件内不包含ASCII码以外的字符。
3. 

在控制台窗口内输入1并输入回车。

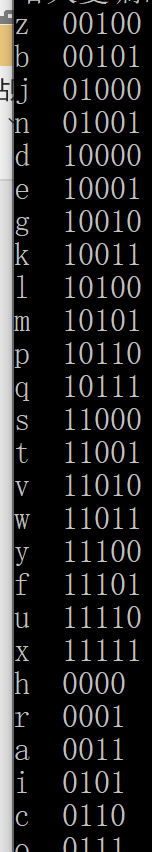
4．

确认符合信息后回车。

1. 得到对应的字符哈夫曼编码，压缩文件code.dat，压缩文件长度和压缩率。此时输入回车退出或者输入2进入解压缩模式。这边可以选择输入2进入解压缩模式。此时文件目录内包含cpp,exe执行文件，source.txt原文件，code.dat压缩文件，huffmancode.txt哈夫曼编码文件。
2. 输入2进入解压缩模式，输入需要的输出文件名，这里设为1.txt。
3. 文件解压缩完成，可以打开目录进行对比，发现内容保持一致。
   1. **测试结果**
4. 纯英文字母测试

测试文件：qwertyuioplkjhgfdsazxcvbnmpoiiuytrewqasdfghjklkmjnbvcxzqawsedrftgyhujikolpmbvcxxreasoinouihcaronaohfcauohifrceqcqawwcxzsdcdcxfvrtgbyuyvhtgmkiolppomli

哈夫曼编码：

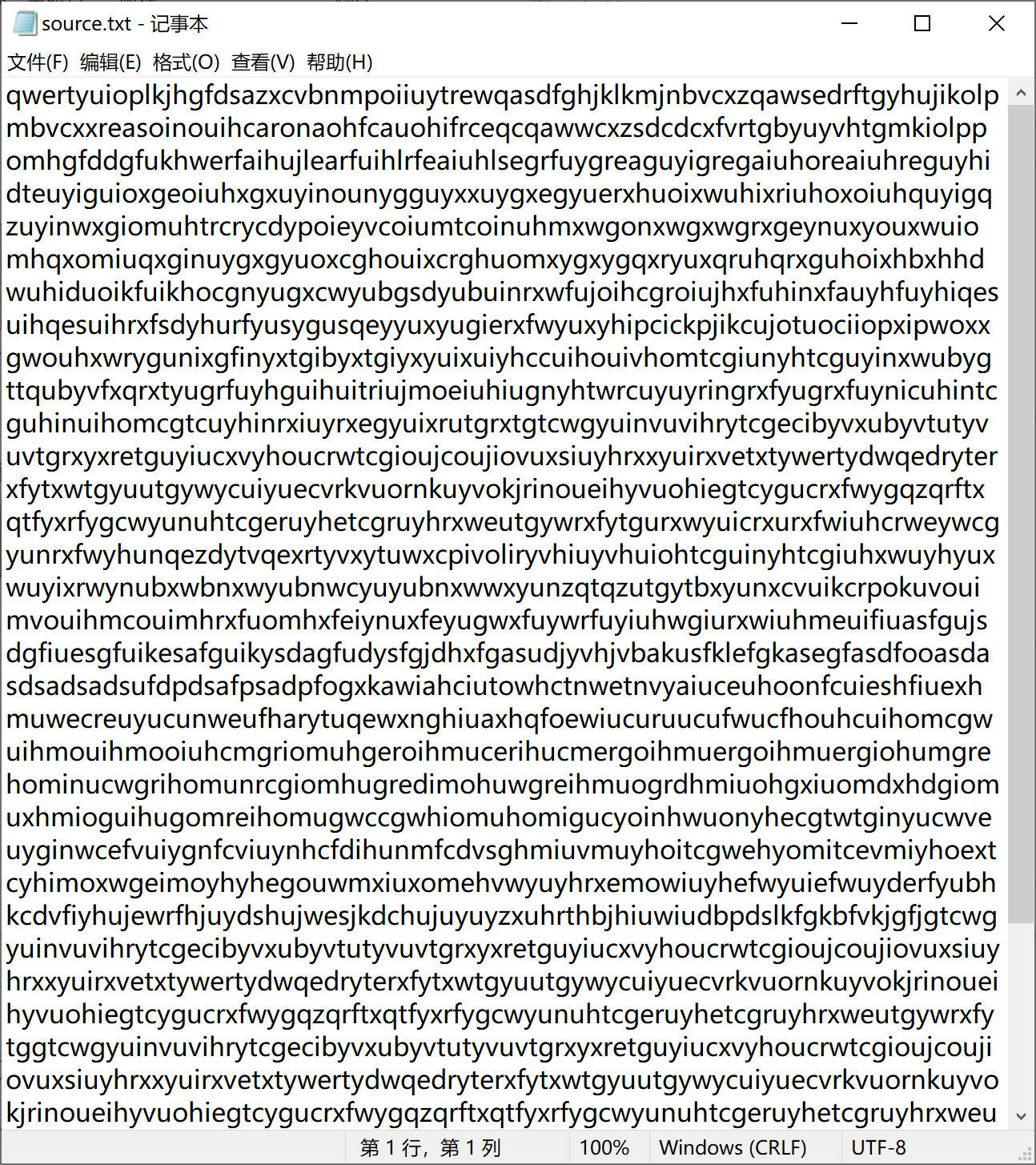


输出文件：qwertyuioplkjhgfdsazxcvbnmpoiiuytrewqasdfghjklkmjnbvcxzqawsedrftgyhujikolpmbvcxxreasoinouihcaronaohfcauohifrceqcqawwcxzsdcdcxfvrtgbyuyvhtgmkiolppom

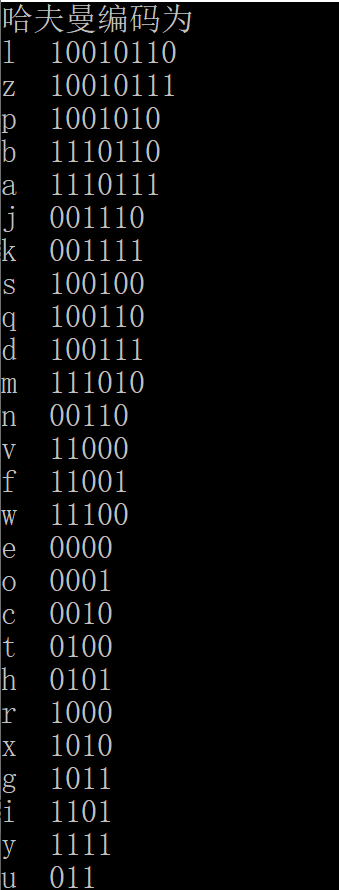
在文件末尾产生错误，原因为结尾哈夫曼10编码不满8个二进制位导致的错误。

1. 纯英文字母测试（字符5000+）

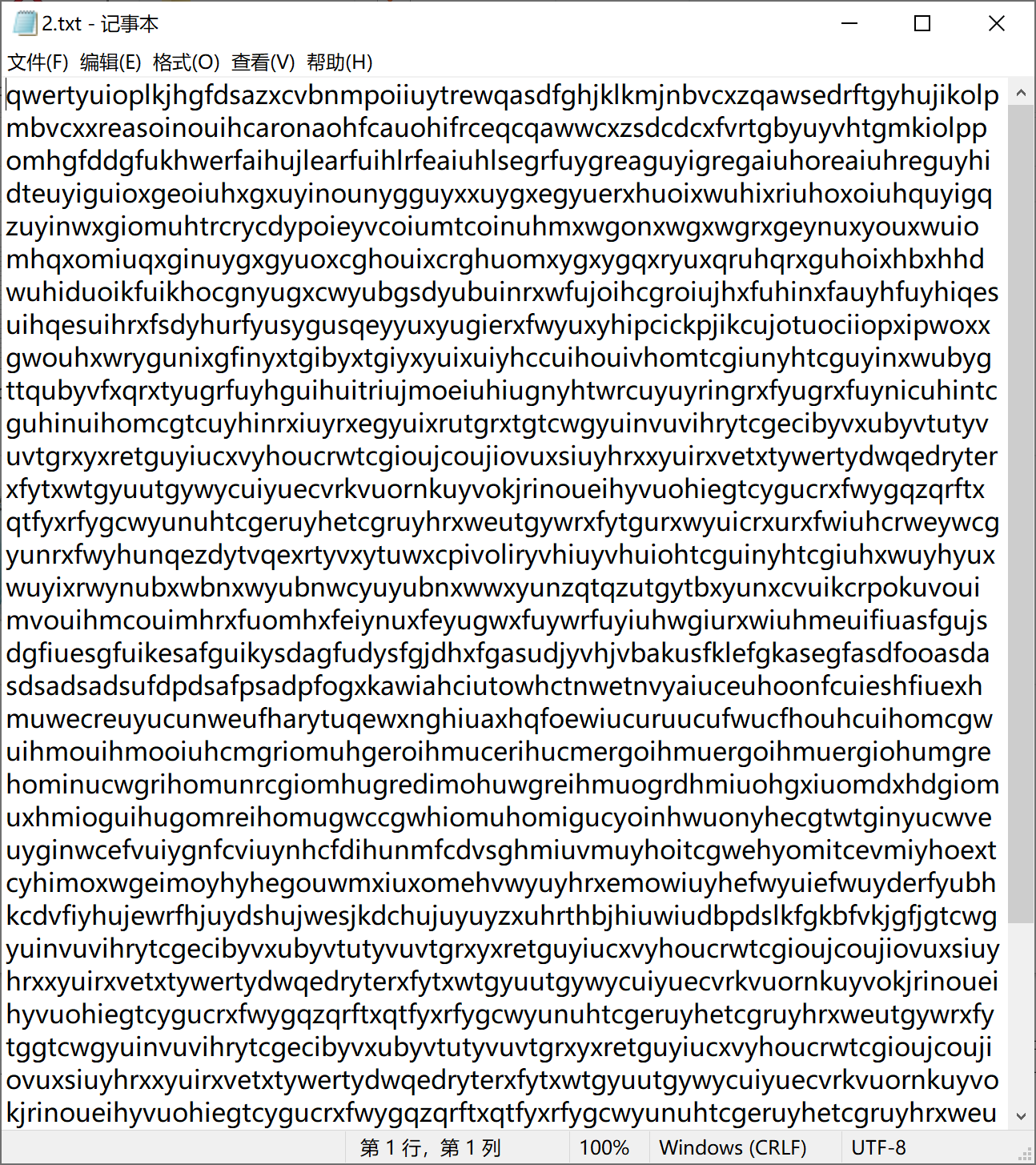
测试文件：



哈夫曼编码：



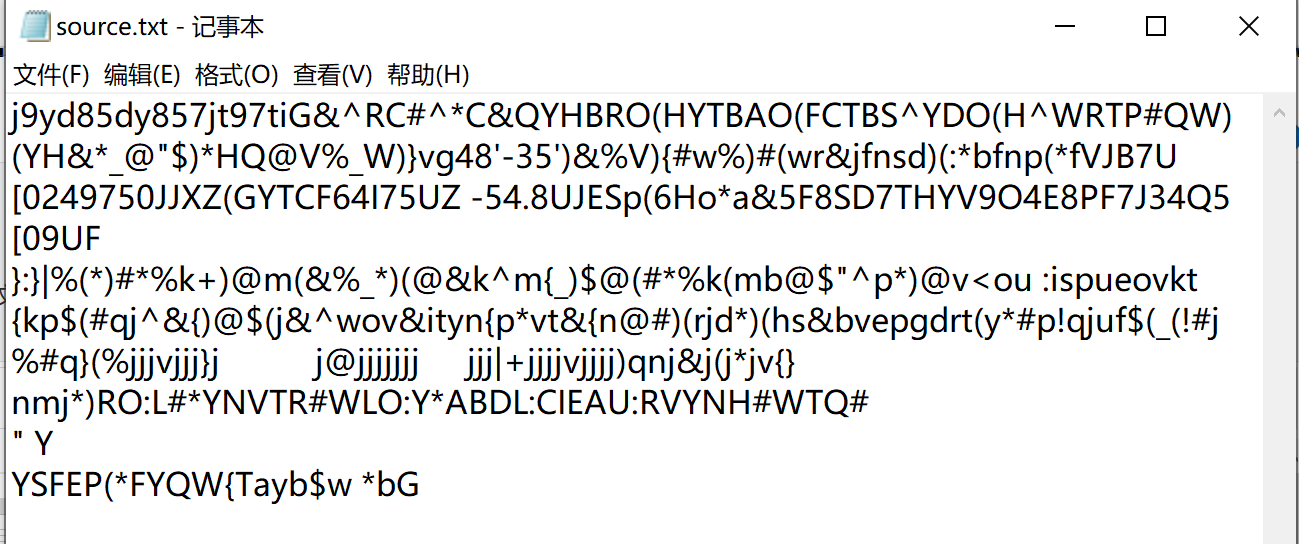
输出文件：



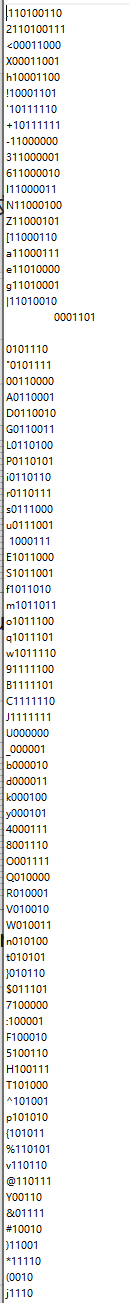
在文件末尾产生错误，原因为结尾哈夫曼10编码不满8个二进制位导致的错误。

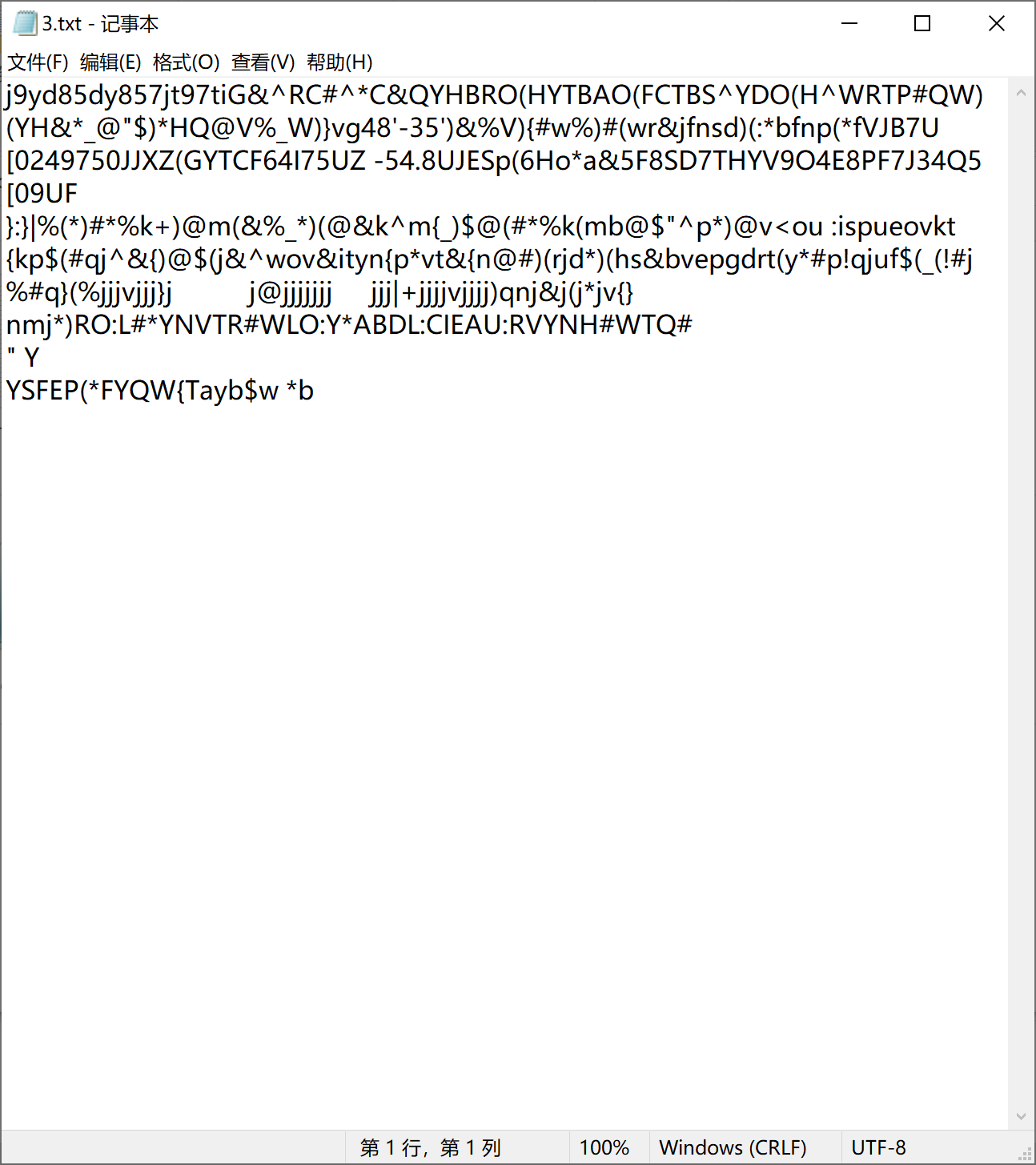
1. 多字符测试

测试文件为：



哈夫曼编码：



输出文件：

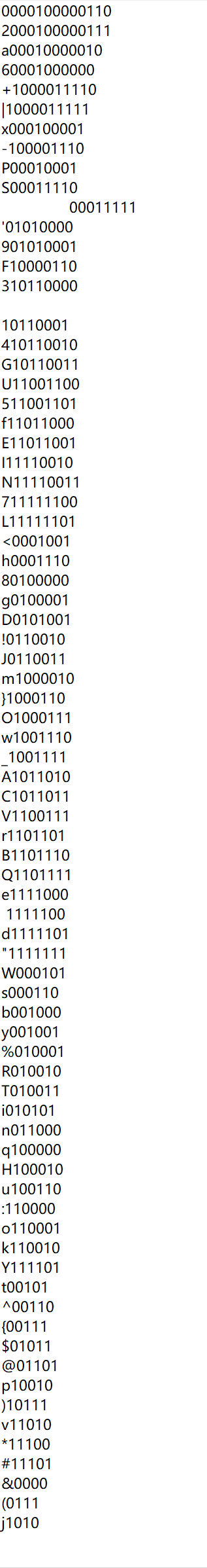
在文件末尾产生错误，原因为结尾哈夫曼10编码不满8个二进制位导致的错误。

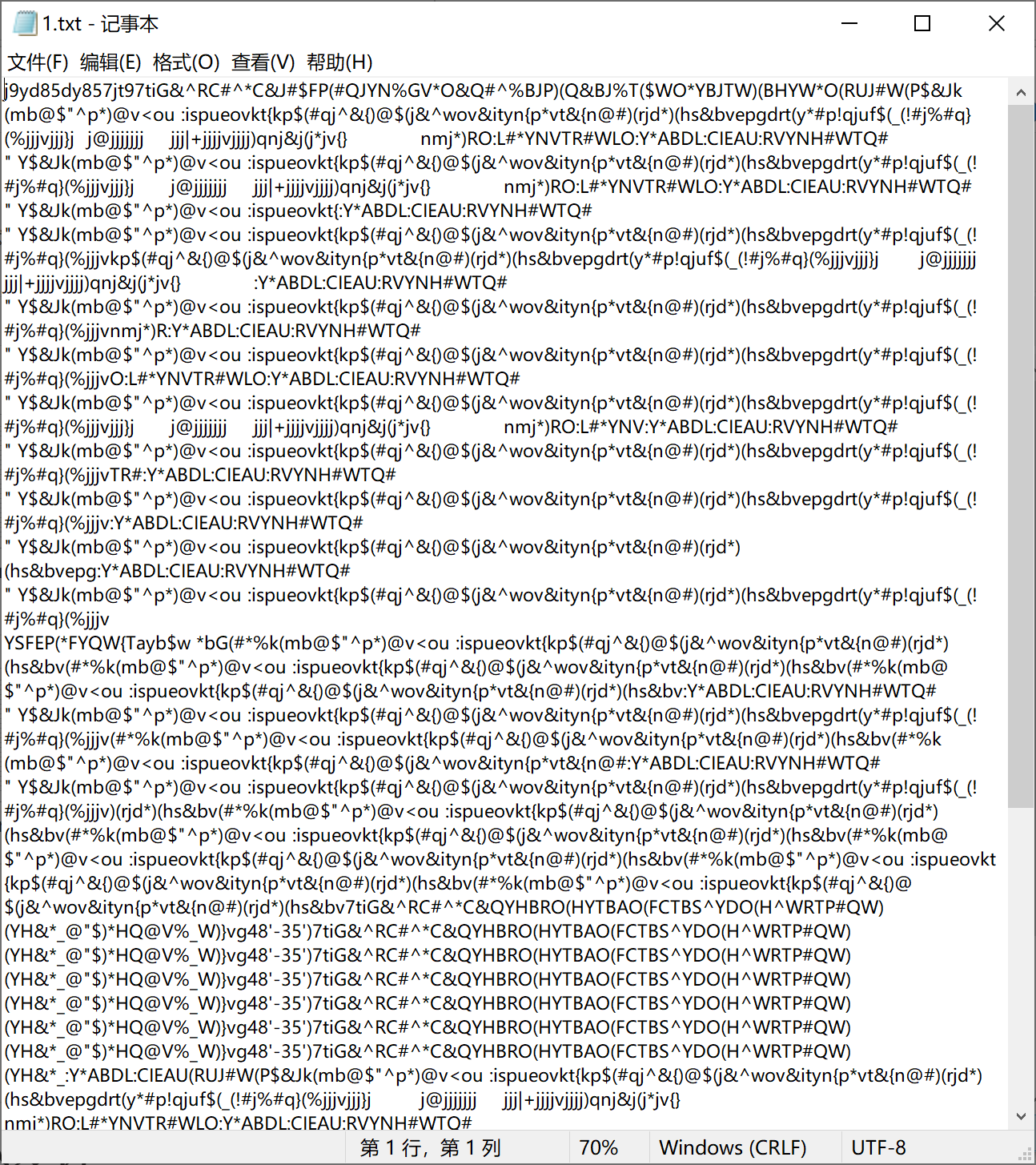
1. 多字符测试（字符5000+）

测试文件为：



哈夫曼编码为：

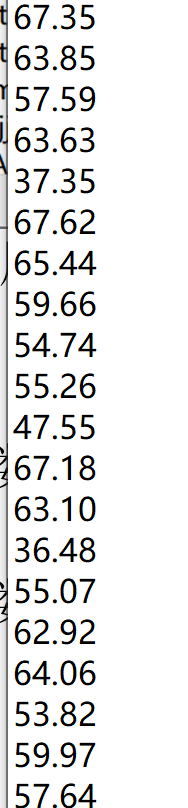


输出文件为：

在文件末尾产生错误，原因为结尾哈夫曼10编码不满8个二进制位导致的错误。

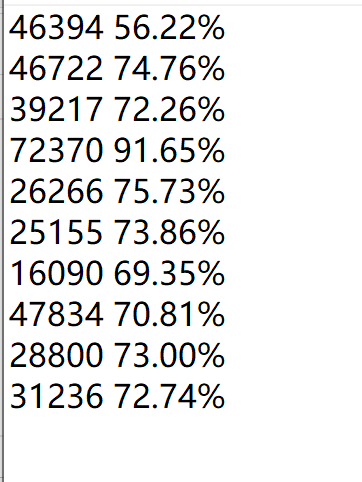
1. 压缩率测试。

测试20组字符数较少数据（0，1000），对于压缩率取平均值。

对于压缩率取平均值为55.63%。

对于压缩率较小的数据则因为是某一字符权值过高，导致其哈夫曼编码较短，从而节省大量存储空间。

测试10组字符数较多数据（5000，）

测试后平均值为70.04%，发现当字符数较多时，一般来说字符种类也会相应加多，尤其当字符总数到达72370时，压缩率高达91.65%。

**4.3 调试分析**

问题：

1. 首先在c语言学习中并没有了解过关于文件读写的代码，所以在实现时也遇到了很多困难。比如在文件读写时，对于非二进制读取（字符读取）是存在读取终止符EOF的，在我的实验中EOF的默认值为8位整型数int 26。但如果我要写入这个数的话，虽然在写文件的时候是正常写入的，但如果用字符读取来直接读取这个写入的文件，则会在遇到EOF时直接终止读取，所以在对于文件读写时需要使用二进制读写。
2. 在解码时需要把int型转化为8位二进制数，而在转换时发现数据不一致。最后发现在转换时需要把得到的8位二进制数的字符数组进行字符串的倒置，才能得到输入的正确结果。
3. 在一开始实验时，默认把哈夫曼编码，代码中的哈夫曼编码HC作为整段代码内置的一段数据，即一段文件进行压缩后立即利用哈夫曼编码HC进行解压缩，这样显然是不可取的。所以在代码中增加了哈夫曼代码HC的输出与读取。
4. 在一段字符的最后几个字符可能会出现解压缩后错误或者缺失的情况，因为这几个字符的哈夫曼编码组合起来往往不是8的整数倍，对于这个情况还没有很好的解决。

时间复杂度：

时间复杂度最高的主要是堆的有序插入和删除为，堆的遍历，线性表的排序,其余函数基本都为。

改进思想：实验中的哈夫曼编码具有两种数据结构，哈夫曼节点独立于哈夫曼编码，并且在创建哈夫曼树时还是用了线性表一个结构，在优化时可以选择合并成单独的哈夫曼节点一种数据结构。同时在文件读写时含有一定冗余操作，可以减少定义的变量，优化查找排序的函数来达到优化整体的目的。

**5． 总结（收获与体会）**

在任务完成过程中，我丰富了对于指针定义字符数组、整型数组的理解，了解到地址与指针的使用与文件字符读写的代码使用,同时也加强了关于堆、哈夫曼树、线性表数据结构的使用。在学习过程中，实现一个目的的函数更像是一个工具，需要去学习了解。而写程序也是一个不断出错，不断试错的过程，需要你去了解一段代码代表的内容，了解寻找错误的方式，更多的错误源于对于代码的不熟悉和对实现逻辑的不通。遇到一个错误时，更多的要从程序的运行流程上逐步排查。而在运行程序时也需要考虑到所有可能的输入。所以调试能力也是代码输入中一个十分重要的部分。在实验中不光要摸透哈夫曼编码与解码，哈夫曼树的构造，还要去深入学习数据的输入与输出等相关操作。在霍夫曼编码方面，需要回顾数据结构已学知识，还要参考一些资料来完善自己的理解。而在算法的输入输出方面，并没有做到完善。首先在输入输出时，每八个位转换为一个byte，但如果当需要把这个数据写出时，如果最后的几位数据不满8位，则会导致输出错误。并且该算法对于文件的读取写入方式是单字节读取、写入，非常缓慢，只能读取一些文本量较小的文本，对于读取一些较大的文本和图片，或者视频时则会占用大量cpu并且处理很长时间，并且还不一定能成功。并且本实验也只是以代码的形式运行，没有创建界面，也不能自定义压缩或者解压缩的名字。而且该实验把压缩生成的霍夫曼编码文件和压缩文件分开存储，导致了不便。而且除了霍夫曼编码的算法，其他算法的读取写入和一些操作步骤都采用了蛮力法，耗时最久。通过该实验，我充分认识到自己的不足，还需要很多步骤来改进。

**6． 源程序**

列出所有程序文件名及相应源代码

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<limits.h>

#include<math.h>

#define LIST\_INIT\_SIZE 256 //线性表存储空间的初始分配量

static int count[LIST\_INIT\_SIZE]; //定义对于每个字符的计数数组

struct HNode { //哈夫曼堆的实现

int \*Data; // 存储int元素，即权值

int Size; // 当前堆内元素个数

int Capacity; // 堆的最大容量

};

typedef struct HNode \*Heap;

Heap CreateMinHeap(int MaxSize) //创建最小堆

{

Heap H = (Heap)malloc(sizeof(struct HNode)); //分配堆空间

H->Data = (int \*)malloc((MaxSize + 1) \* sizeof(int)); //分配堆的元素空间

H->Size = 0;

H->Capacity = MaxSize;

H->Data[0] = -1; //第一位为哨兵

return H;

}

bool IsFull(Heap H) //判断堆是否为满

{

return (H->Size == H->Capacity);

}

bool IsEmpty(Heap H) //判断堆是否为空

{

return (H->Size == 0);

}

bool Insert(Heap &H, int X) //排序插入堆

{

int i;

if (IsFull(H)) {

printf("最小堆已满");

return false;

}

i = ++H->Size; //元素加一

for (; H->Data[i / 2] > X; i /= 2) //选择位置插入

H->Data[i] = H->Data[i / 2];

H->Data[i] = X;

return true;

}

int ERROR=-1;

int Deletemin(Heap &H) //返回并去除最小值

{

int Parent, Child;

int MinItem, X;

if (IsEmpty(H)) {

printf("最小堆已为空");

return ERROR;

}

MinItem = H->Data[1]; //最小值为1号节点

X = H->Data[H->Size--];

for (Parent = 1; Parent \* 2 <= H->Size; Parent = Child) { //排序移动元素

Child = Parent \* 2;

if ((Child != H->Size) && (H->Data[Child]>H->Data[Child + 1]))

Child++;

if (X <= H->Data[Child]) break;

else

H->Data[Parent] = H->Data[Child];

}

H->Data[Parent] = X;

return MinItem;

}

void listprint(Heap H) //print堆

{

for(int i=1;i<=H->Size;i++)

printf("%d ",H->Data[i]);

}

int ReadFile(char filename[]) //读取文件

{

char buff[1024]; //设置读取缓冲区

FILE\* fp;

int len;

int typecount=0;

if ((fp = fopen(filename,"r")) == NULL)

{

perror("fail to read the file");

exit(1);

}

while(fgets(buff,1024,fp))

{

len=strlen(buff);

for(int i=0;i<len;i++)

{

if(buff[i]>256) printf("含有非法字符");

count[buff[i]]++;

}

}

for(int j=0;j<256;j++) //读取同时计数

if(count[j]!=0)

typecount++;

fclose(fp);

return typecount;

}

int ReadFile1(char filename[]) //读取文件

{

char buff[1024]; //设置读取缓冲区

FILE\* fp;

int len;

int count1=0;

if ((fp = fopen(filename,"r")) == NULL)

{

perror("fail to read the file");

exit(1);

}

while(fgets(buff,1024,fp))

{

len=strlen(buff);

count1+=len;

}

fclose(fp);

return count1;

}

typedef struct ElemType{ //哈夫曼节点 权值和字符名

int value;

char name;

};

typedef struct{ //线性表

ElemType \*elem;

int length;

int listsize;

}SqList;

int InitSqList(SqList &L)

{

L.elem=(ElemType \*)malloc(LIST\_INIT\_SIZE\*sizeof(ElemType)); //申请线性表空间

L.length=0; //当前长度

L.listsize=LIST\_INIT\_SIZE; //当前分配量

return 0;

}

int ListInsert(SqList &L,ElemType e) //插入元素默认为队尾

{

L.elem[L.length]=e;

L.length++;

return 0;

}

ElemType ListPeek(SqList L)

{

return L.elem[0];

}

void ListSort(SqList &L) //线性表排序

{

for (int i=0;i<L.length-1;i++)

for (int j=0;j<L.length-i-1;j++)

{

if (L.elem[j].value>L.elem[j+1].value) {

ElemType temp=L.elem[j];

L.elem[j]=L.elem[j+1];

L.elem[j+1]=temp;

}

}

}

void ListPrint(SqList L)

{

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

if(L.elem[i].name=='\n')

printf("name=换行,value=%d\n",L.elem[i].value);

else printf("name=%c,value=%d\n",L.elem[i].name,L.elem[i].value);

printf("------------------------------------------------------------------------------\n");

}

void JoinElemTypeNode(SqList &L) //线性表添加元素

{

for(int i=0;i<LIST\_INIT\_SIZE;i++)

if(count[i]!=0)

{

ElemType newone={count[i],(char)i};

ListInsert(L,newone);

}

ListSort(L);

ListPrint(L);

}

typedef struct //哈夫曼 本质为线性表

{

int value;

char name;

int parent, lchild, rchild;

}HTNode,\*HuffmanTree;

typedef char \* \*HuffmanCode;

int min(HuffmanTree ht,int n){ //旧获得权值最小元素，暂不使用

int i,ret;

int temp=INT\_MAX;

for(i=1;i<=n;i++)

if(ht[i].value<temp&&ht[i].parent==0){

temp=ht[i].value;

ret=i;

}

ht[ret].parent=1;

return ret;

}

int newSearch(HuffmanTree HT,int n,int e) //根据权值搜索相应的元素，并把parent设为1防止二次访问，等价。

{

for(int i=1;i<=n;i++)

if(HT[i].value==e&&HT[i].parent==0){

HT[i].parent=1;

return i;

}

}

void Select(HuffmanTree HT,int n,int &s1,int &s2){ //暂不使用

s1=min(HT,n);

s2=min(HT,n);

}

void HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, SqList L, int n) //哈夫曼编码，建立哈夫曼树并获得对应哈夫曼编码

{

int i,s1,s2,m,c,start,f;

HuffmanTree p;

char \*cd;

m=2\*n-1;

HT=(HuffmanTree)malloc((m+1)\*sizeof(HTNode)); //申请空间 第0个不使用

for(p=HT+1,i=1; i<=n; ++i,++p){ //前n个元素为初始节点

p->value=L.elem[i-1].value;

p->name=L.elem[i-1].name;

p->parent=0;

p->lchild=0;

p->rchild=0;

}

for(;i<=m; ++i,++p){ //初始化后n+1个元素

p->value=0;

p->name={};

p->parent=0;

p->lchild=0;

p->rchild=0;

} //对叶子之后的存储单元清零

Heap H=CreateMinHeap(m); //创建最小堆

for(int i=1;i<=n;i++)

Insert(H,L.elem[i-1].value); //添加n个元素

//listprint(H);

//printf("\n");

for(i=n+1;i<=m; ++i){ //建Huffman树

//Select(HT, i-1, s1, s2);

int e1=Deletemin(H);int e2=Deletemin(H); //获得堆的最小的2个元素

s1=newSearch(HT,i-1,e1);

s2=newSearch(HT,i-1,e2);

//在HT[1…i-1]选择parent为0且weight最小的两个结点，其序号分别为

//s1和s2

HT[s1].parent=i; HT[s2].parent=i;

HT[i].lchild=s1; HT[i].rchild=s2; //s1、s2分别作为i的左右孩子

HT[i].value=HT[s1].value+ HT[s2].value;

Insert(H,HT[i].value); //添加为权值之和的元素进入堆

}

HC=(HuffmanCode)malloc((n+1)\*sizeof(char\*));

//分配n个字符编码的头指针向量

cd=(char\*) malloc(n\*sizeof(char)); //分配求编码的工作空间

cd[n-1]='\0'; //编码结束符（从cd[0]~cd[n-1]为合法空间）

for(i=1;i<=n;++i){ //逐个字符求Huffman编码

start=n-1; //编码结束符位置

for(c=i,f=HT[i].parent; f!=0; c=f, f=HT[f].parent)

//从叶子到根逆向求编码

if(HT[f].lchild==c) cd[--start]='0';

else cd[--start]='1';

HC[i]=(char\*)malloc((n-start)\*sizeof(char));

//为第i个字符编码分配空间

strcpy(HC[i],&cd[start]); //从cd复制编码串到HC

}

free(cd); //释放工作空间

}

void showHC(HuffmanCode HC,HuffmanTree HT,int n)

{

for(int i=1;i<=n;i++)

{

printf("%c ",HT[i].name);

puts(HC[i]);

//puts(w);

}

}

void WriteCode(char filename[],HuffmanCode HC,HuffmanTree HT,int n)

{

FILE \*fp = NULL;

fp = fopen(filename, "w+"); //写哈夫曼编码文件

for(int i=1;i<=n;i++)

{

char w[30]={HT[i].name};

strcat(w,HC[i]);

strcat(w,"\n");

fputs(w, fp);

//puts(w);

}

fclose(fp);

}

void PrintHuffmanCode(HuffmanTree HT,int n)

{

for(int i=1;i<=2\*n-1;i++)

printf("%d \n",HT[i].value);

}

void ReadandCompressFile2(char filename[],char ALL[],char OUT[],int n,HuffmanCode HC,HuffmanTree HT,char outfilename[]) //读取并压缩文件

{

char buff[1024];

FILE\* fp;

if ((fp = fopen(filename,"r")) == NULL)

{

perror("fail to read the file");

exit(1);

}

while(fgets(buff,1024,fp))

{

strcat(ALL,buff); //获得输入文件

}

//puts(ALL); 输入文件

//printf("lengthjhshdhs=%d",strlen(ALL));

int length=strlen(ALL);

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

for(int j=1;j<=n;j++)

if(ALL[i]==HT[j].name) //变换输入文件为哈夫曼编码，10格式

strcat(OUT,HC[j]);

fp = fopen(outfilename, "wb+"); //二进制写入

//puts(OUT); 二进制10文件

//printf("\nlengthjhshdhs=%d\n",strlen(OUT));

int totalcount=0;

//compresss

for(int i=7;i<strlen(OUT);i+=8){

int temp=0;

for (int j=i;i-j<8;j--) //每八位转为0-255的int型

temp+=(OUT[j]-48)\*(pow(2,(i-j)));

// if(temp==0)printf("-----------------------------------------------------------------------------------may error");

//if(temp==26){temp=0;}

fputc(temp,fp);

// printf("%d ",temp);

totalcount++;

}

fclose(fp);

//printf("\ntotalcount%d\n",totalcount);

}

typedef struct HFCode{ //哈夫曼编码

char name;

char c[1024];

};

void del\_char(char a[],char c) //删除指定字符

{

int i,j;

for(i=0,j=0; \*(a+i)!='\0'; i++)

{

if(\*(a+i)==c)

continue;

else

{

\*(a+j)=\*(a+i);

j++;

}

}

\*(a+j)='\0';

}

void Getmessage(char filename[],HFCode IN[]) //从哈夫曼编码文件中获得哈夫曼编码 HFCOde

{

char buff[1024];

FILE\* fp;

if ((fp = fopen(filename,"r")) == NULL)

{

perror("fail to read the file");

exit(1);

}

int count1=1;

IN[0].name='a';

strcat(IN[0].c,"a");

while(fgets(buff,1024,fp))

{

if(buff[0]=='\n') //根据换行进行不同的读取

{

IN[count1].name='\n';

fgets(buff,1024,fp);

strtok(buff, "\n");

strcpy(IN[count1].c,buff);

count1++;

}

else

{

del\_char(buff,'\n');

IN[count1].name=buff[0];

strcpy(IN[count1].c,buff+1);

count1++;

}

}

fclose(fp);

}

void fun(char\*s) //字符串的倒置

{

int i,j=0,n;

char temp;

n=strlen(s);

for(i=0;i<n/2;i++){

temp = s[i];

s[i] = s[n-1-i];

s[n-1-i] = temp;

}

}

void substring(char a[],char b[],int start,int last) //字符串的截取函数

{

int i;

for(i=0;i<last-start;i++)

{

b[i]=a[start+i];

}

b[++i]='\0';

}

int DeCompressFile(char filename[],HFCode USE[],char outfile[]) //从哈夫曼编码文件与压缩文件解压为原文件

{

FILE\* fp;

if ((fp = fopen(filename,"rb")) == NULL) //二进制读取，规避Int 26终止符号

{

perror("fail to read the file");

exit(1);

}

char laststr[50000]={};

int e;

while((e=fgetc(fp))!=EOF)

{

//printf("%d ",e);输出读入int字符

char str[50]={};

int temp=0;

//if(e==0)

// temp=26;

// else

temp=e;

for(int j=0;j<8;j++) //获得每个int代表的8位二进制编码

{

int compare=abs(temp%2);

if(compare==1)

strcat(str,"1");

else

strcat(str,"0");

temp=temp/2;

}

fun(str); //倒置编码为正确。

strcat(laststr,str);

}

//puts("---------------------------------------------------------");

//puts(laststr);输出编码文件10格式

int uselen;

fp = fopen(outfile, "w+"); //写出解压缩文件

for(int i=0;USE[i].name!=NULL;i++)

uselen=i;

//printf("%d\n",uselen); 输出长度

//printf("%d",strlen(laststr)); 还原字符串长度

int supercountl=0;

for(int i=0;i<strlen(laststr);i++)

for(int j=i;j<strlen(laststr);j++){

char strin[50000]={};substring(laststr,strin,i,j);

for(int k=1;k<=uselen;k++)

if(strcmp(strin,USE[k].c)==0)

{

i=j;fputc(USE[k].name,fp);supercountl++;

break;

}

}

fclose(fp);

return supercountl;

}

int main()

{

printf("输入1 进行文件压缩\n输入2 进行文件解压缩\n");

char ch=getchar();

char filename2[]="huffmancode.txt";

char outfilename[]="code.dat";

char filename[]="source.txt";

if(ch=='1')

{

printf("请确认文件source.txt在同一目录内，如果无反应则文件内可能包含非ASCII字符，确认后输入回车\n");

getchar();

getchar();

int typecount=ReadFile(filename); //读取输入文件

printf("共%d种字符\n",typecount);

SqList L;

InitSqList(L);

JoinElemTypeNode(L);

HuffmanCode HC;

HuffmanTree HT;

HuffmanCoding(HT,HC,L,typecount); //哈夫曼编码

printf("哈夫曼编码为\n");

showHC(HC,HT,typecount);

WriteCode(filename2,HC,HT,typecount); //写哈夫曼文件

char OUT[500000]="\0";

char ALL[500000]="\0";

ReadandCompressFile2(filename,ALL,OUT,typecount,HC,HT,outfilename); //写压缩文件

//puts(OUT);

//puts(ALL);

printf("压缩完成!!! 压缩文件为code.dat, 文件长度为%d个字符,压缩率为%.2f%% \n",strlen(OUT)/8,100\*strlen(OUT)/8.0/strlen(ALL));

puts("输入回车退出或输入2进入解压缩模式");

ch=getchar();

}

if(ch=='2')

{

printf("请输入解压缩文件名称,保证哈夫曼编码文件huffmancode.txt与输出文件code.dat在同一目录内\n");

char orfile[30];

scanf("%s",orfile);

HFCode \*Co=(HFCode\*)malloc((256)\*sizeof(HFCode));

Getmessage(filename2,Co); //首先获得哈夫曼编码

//for(int i=1;i<=typecount;i++)

// printf("name=%c,len=%d,code=%s \n",Co[i].name,strlen(Co[i].c),(Co[i].c));

//for(int i=0;i<strlen(Co[0].c);i++)

//printf("%c",Co[0].c[0]);

int count1=ReadFile1(filename);

int count2=DeCompressFile(outfilename,Co,orfile); //还原原文件

printf("解压缩完成 完整率为%.2f\n",count2\*100.0/count1);

getchar();

getchar();

}

return 0;

}