



北京航空航天大学
B E I H A N G U N I V E R S I T Y

MATLAB 基础及其应用

Huffman 编码与解码

实验报告

北京航空航天大学

计算机学院

陈麒先 金泽晖 李贞子 王润泽

二〇一八年十二月

郑重声明

关于诚实守信公约：

本实验报告中参考了文献或互联网资料的部分均有引用标注，另有与老师或同学研究后的成果引用将在特别致谢中说明。另，实验报告撰写仓促，如有错误，在所难免，欢迎批评指正！特此声明。

陈麒先 金泽晖 李贞子 王润泽

原创性声明

实验报告未注释部分内容由作者原创。抄袭行为在任何情况下都是不能容忍的 (COPY is strictly prohibited under any circumstances)！转载或引用须征得作者本人同意，并注明出处！**勿谓言之不预！**

陈麒先 金泽晖 李贞子 王润泽



目录

一、问题描述	1
二、问题分析	2
1、哈夫曼编码概述	2
2、构造哈夫曼树	2
3、利用哈夫曼树进行编码	2
三、利用 MATLAB 解决哈夫曼编解码	3
1、预处理输入	3
2、构造哈夫曼树	4
3、利用哈夫曼树进行编码	7
4、利用哈夫曼编码生成压缩后文本	9
5、哈夫曼编码程序最终运行结果	10
6、利用哈夫曼编码进行解码	11
四、组内同学承担任务说明	14

一、问题描述

哈夫曼编码（英语：Huffman Coding），又译为霍夫曼编码、赫夫曼编码，是一种用于无损数据压缩的算法。由美国计算机科学家大卫·霍夫曼（David Albert Huffman）在 1952 年发明。

在计算机数据处理中，哈夫曼编码使用变长编码表对源符号（如文件中的一个字母）进行编码。在这个变长编码表中，出现机率高的字母使用较短的编码，反之出现机率低的则使用较长的编码，这便使编码之后的字符串的平均长度、期望值降低，从而达到无损压缩数据的目的。

例如，在英文中，e 的出现机率最高，而 z 的出现概率则最低。当利用哈夫曼编码对一篇英文进行压缩时，e 极有可能用一个 bit 来表示，而 z 则可能花去 25 个 bit。用普通的表示方法时，每个英文字母均占用 1 个 byte，即 8 个 bit。二者相比，e 使用了一般编码的 $1/8$ 的长度，z 则使用了 3 倍多。倘若我们能实现对于英文中各个字母出现概率的较准确的估算，就可以大幅度提高无损压缩的比例。

二、问题分析

1、哈夫曼编码概述

要利用哈夫曼编码对一个文件进行无损压缩，算法一般由以下两个步骤构成：

- 1) 根据输入的文本构造哈夫曼树（哈夫曼树构造算法）
- 2) 遍历哈夫曼树，为文本中每个字符分配对应的哈夫曼编码

2、构造哈夫曼树

1) 哈夫曼树定义

哈夫曼树又称最优二叉树，是一种带权路径长度最短的二叉树。所谓树的带权路径长度，就是树中所有的叶结点的权值乘上其到根结点的路径长度。树的路径长度是从树根到每一结点的路径长度之和，记为 $WPL = (W_1 * L_1 + W_2 * L_2 + W_3 * L_3 + \dots + W_n * L_n)$ ， N 个权值 W_i ($i=1,2,\dots,n$) 构成一棵有 N 个叶结点的二叉树，相应的叶结点的路径长度为 L_i ($i=1,2,\dots,n$)。可以证明哈夫曼树的 WPL 是最小的。

2) 哈夫曼树构造算法如下

假设有 n 个权值，则构造出的哈夫曼树有 n 个叶子结点。 n 个权值分别设为 w_1 、 w_2 、...、 w_n ，则哈夫曼树的构造规则为：

- (1) 将 w_1 、 w_2 、...、 w_n 看成是有 n 棵树的森林(每棵树仅有一个结点)；
- (2) 在森林中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，且新树的根结点权值为其左、右子树根结点权值之和；
- (3) 从森林中删除选取的两棵树，并将新树加入森林；
- (4) 重复(2)、(3)步，直到森林中只剩一棵树为止，该树即为所求得的哈夫曼树。

3、利用哈夫曼树进行编码

得到哈夫曼树后，利用中序遍历或前序遍历扫描整棵树，在根节点时编码字符串为空，之后每进入一个节点的左子树时给编码字符串补'1'，进入右子树时给编码字符串补'0'，直到扫描到叶节点，此时这个叶节点对应的编码就是编码字符串。返回父节点时去掉补上得字符即可。通过这样一个递归的过程就得到了每一个字符的哈夫曼编码。再将原文本的每一个字符都转化成哈夫曼编码，从而生成一个压缩后的只有 0 和 1 的编码文件。

三、利用 MATLAB 解决哈夫曼编解码

为方便叙述，以待压缩文本 acbaeabddbcbaac 为例阐述利用 MATLAB 实现哈夫曼编码解码的过程。

1、预处理输入

(1) 读取输入文件

```
%按照输入类型读取文本，0为读取指定文件test.txt，1为自定义文件路径
prompt = 'Please select the file you want to process: 0: process test.txt 1: custom file';
type = input(prompt);
if(type == 0)
    [text] = textread('test.txt','%[^\n]'); %读文件,text为一个cell型变量
else
    path = input('Please input a file path:','s');%输入文件路径
    [text] = textread(path,'%[^\n]');
end
```

(2) 将输入文件转化为一行字符串

```
%预处理文本
str = char(text);%通过char函数将text转化成一个字符串变量str
s = '';
ls = size(str); %获取字符串的行数
linenum = ls(1); %总行数
for i = 1 : linenum % 文件字符串格式处理
    s = strcat(s , str(i,:));%将多行文本拼接成一行
end
```

(3) 记录节点的信息，调用 Huffman 函数

```
b=unique(s);%计算有多少个不重复的字符串
for i=1:length(b)
    num(i) = length(strfind(s,b(i))); %统计字符的数目
end

b;%显示字符（调试用）
num;%显示个数（调试用）
for i = 1 : length(b) % 建立一个结构数组
    map(i).ch = b(i); % 每个结构体记录字符值和出现次数
    map(i).num = num(i);
end
map;

[num_sorted , id] = sort([map.num]); %按照出现次数将结构数组排序，排序后的下标按升序存入id中
huffman_result = Huffman(num);
```

2、构造哈夫曼树

根据我们在课堂上所学到的知识，MATLAB 中的指针运算不是其强项，矩阵运算以及相关操作是应用 MATLAB 的精华所在。为了充分利用 MATLAB 在处理矩阵过程中的便捷性，以及考虑到具体问题具体分析原则，最终我们小组决定利用矩阵来模拟哈夫曼树的生成过程。

具体算法如下：

初始情况

字符	a	b	c	d	e
频率	5	4	3	2	1

(1) 将这五个初始频率从小到大排序后存入矩阵的第一行，代表哈夫曼树的叶节点

1	2	3	4	5

矩阵



树

相应代码：Huffman 为计算哈夫曼树的函数，函数参数 P 为排序后的出现次数序列，返回一个记录每个字符对应编码的矩阵 a

```

function [a]=Huffman(P)
P=sort(P); % 对传入序列进行排序（升序）
A=P; % A 为初始序列
B=[];
i=1; % i为Huffman数的层数（自底向上构造）
LL=length(P); %LL为序列总长度（即不同字母个数）
L=LL; % L为当前序列长度
B(1,:)= P; % B 也为初始序列，之后B矩阵存放Huffman树

```

（2）执行以下步骤

每次将矩阵最后一行的前两个数相加，结果放入新的一行的行首

将上一行的其它元素照搬下来

将新生成的一行从小到大排序

若新的一行的元素个数大于 2，返回第一步

直到最后一行只剩两个数，这两个数就是哈夫曼树的根节点的左右子节点，两个数的和就是哈夫曼树的根节点

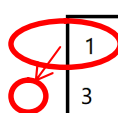
相应代码：

```

while (L>2) % 该循环为Huffman树的计算过程
    i=i+1;
    B(i,1)=A(1)+A(2); %选排序最小的两个节点相加
    for j=2:(L-1)
        B(i,j)=A(j+1); % 利用循环将剩下的节点值顺序移动
    end
    L=L-1; % 更新剩余节点总长度
    B(i,1:L)=sort(B(i,1:L)); % 对当前层进行排序
    A=B(i,1:L); % 更新A数组用于下次循环计算
end

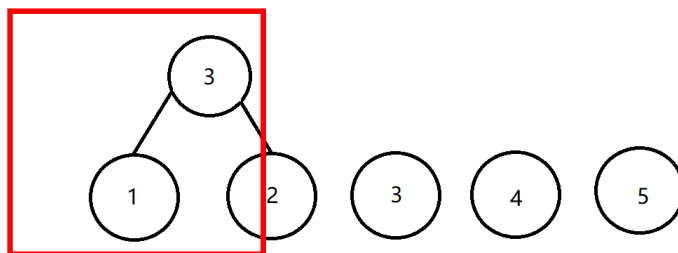
```

演示图例：



1	2	3	4	5
3	3	4	5	

矩阵

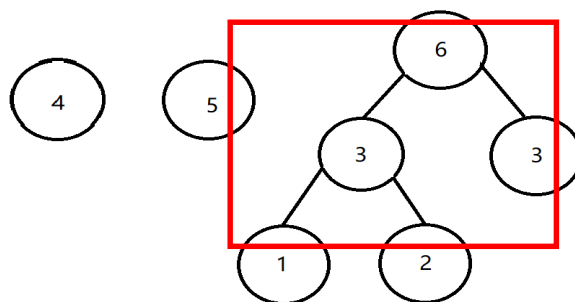


树

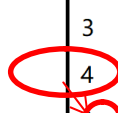



1	2	3	4	5
3	3	4	5	
4	5	6		

矩阵

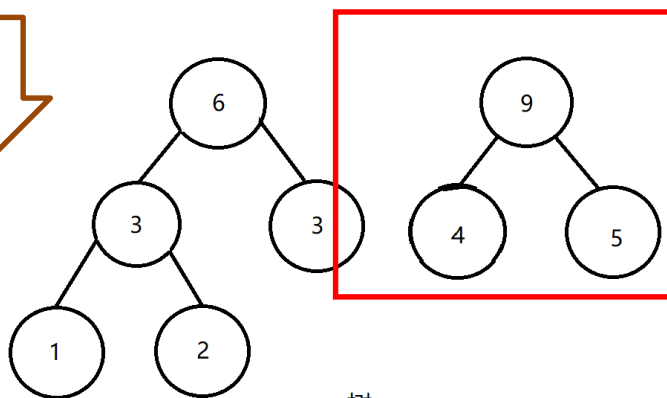


树

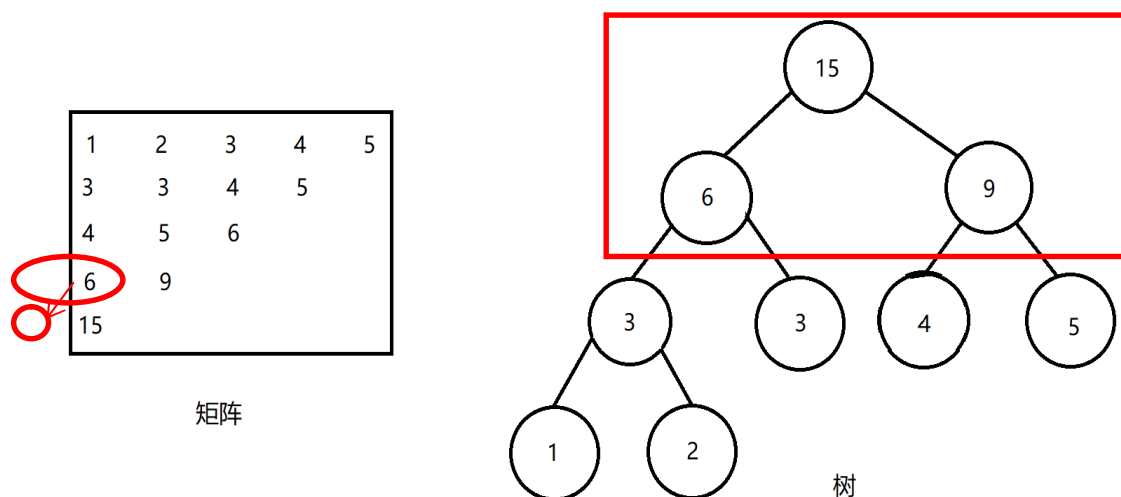
1	2	3	4	5
3	3	4	5	
4	5	6		
9				

矩阵



树





至此，哈夫曼树就已经构造完成了，并且其所有信息都已经存储在矩阵中了。

可以看到，在整个模拟过程中：

每次选取最后一行的前两个字符相加相当于为当前森林中最小的两个根节点生成一个父节点并且将两棵树连接起来，这个新树的根节点的值就是原来两个根节点值的和。

排序是为了使得下一次要取的根节点（当前森林中最小的两个根节点）就位于最后一行的前两个，省去了遍历下一行选出两个最小节点的过程。

将最后一行第三个及之后的元素照搬到下一行代表对当前森林中不是最小两个根节点的其他根节点不做处理。

矩阵并没有包括最后的根节点（最后一行有两个树），因为到最后一步时没有必要计算出根节点的值并加进矩阵，只要比较最后一行两个数的大小就可以明白哪一个是左子树哪一个是右子树。

所以可以认为生成这个矩阵 B 的过程就是生成哈夫曼树的过程。

3、利用哈夫曼树进行编码

完成哈夫曼树的构建以后，下一步就是利用哈夫曼树生成每一个字符的哈夫曼编码。刚才生成的哈夫曼矩阵记为 B

(1) 初始化编码矩阵 W ，置根节点左子节点为 1， W 是一个元胞矩阵，与 B 同规格。其中 $W(x,y)$ 为 $B(x,y)$ 对应的节点的哈夫曼编码，是一个元胞类型，元胞中是编码的字符串形式。

```

for ll=1:i % 初始化huffman编码矩阵
    for n=1:LL
        W(ll,n)={'0'}; % 采用cell结构里面含有字符串，为每一个节点的初始编码置0
    end
end
W(i,1)={'1'}; %根节点左子节点为1

```

第 i 行为根的左右子节点

(2) 遍历哈夫曼树生成编码

```

34 - for m=(i-1):-1:1 % 从上往下遍历（递归下降）
35 -     BB=B(m,1)+B(m,2); % 计算该层前两个节点（数值最小两个）的和到BB
36 -     BBB=find(B(m+1,:)==BB); % 在上一层中找到该节点
37 -     BBB=BBB(1); % 获取这两个节点父节点的列坐标
38 -
39 -     W(m,1:2)=W(m+1, BBB); % 该层前两个节点复制其父节点的信息
40 -     W(m,1)=strcat(W(m,1), '1'); % 左子节点连 '1'（较小节点）
41 -     W(m,2)=strcat(W(m,2), '0'); % 右子节点连 '0'
42 -     uu=zeros(1,LL);
43 -     uu(1)=BBB;
44 -     y=1;
45 -
46 -     for n=3:(LL+1-m) % 处理除了前两个节点剩下的节点
47 -         fd3=find(B(m,n)==B(m+1,:)); % 在上一层中找到与当前处理节点值相等的节点索引记录到fd3
48 -         % 处理上一层有冲突的情况（一个值对应多个节点，即fd3中有多个元素的取值）
49 -         for pp=1:length(fd3)
50 -             kk=isempty(find(uu==fd3(pp)));
51 -             if(kk==1)
52 -                 y=y+1;
53 -                 fd3=fd3(pp);
54 -                 uu(y)=fd3;
55 -                 break;
56 -             end
57 -         end
58 -         % bug fix
59 -         W(m,n)=W(m+1, fd3); % 将上一层中找到与该节点值相等的huffman码值复制下来
60 -     end
61 - end
62 - a=W(1,:);
63 -
64 - end

```

line 34: 从矩阵 $i-1$ 行开始向上遍历（ B 为上一步生成的矩阵，一共有 i 行），相当于从树的根节点向下遍历

line 35-37: 取出矩阵第 $i-1$ 行的前两个节点，求和生成 BB ，在下一行中找 BB 的位置， BBB 存储第 $i-1$ 行前两个节点的在下一行的父节点的列坐标。父节点坐标因此为 $(m+1, BBB)$

line 39-44: 对于哈夫曼编码而言，左子节点（较小节点）的编码为父节点编码加 1，右子节点的编码为父节点编码加 0，所以找到父节点后取出父节点的编码 $W(m+1, BBB)$ (因为是从矩阵底到矩阵顶遍历，所以 $m+1$ 行的值要么之前算过了要么就是初始化的值)，按规则生成 $W(m, 1)$ 为左节点编码， $W(m, 2)$ 为右节点编码

line 46-47: 由规律可知，矩阵的第 m 行有 $LL+1-m$ 个非空元素，现在需要本行把第三个元素到最后一个元素的编码值从下一行搬上来（因为生成矩阵的时候就是把这些元素照搬到矩阵的下一行的），第 47 行要在矩阵的 $m+1$ 行找到 m 行照搬元素的索引，记

在 fd3 中。

line 49-57: 处理冲突情况。对于 fd3 中每一个索引（对应在下一行每一个跟该元素相等的值），都判断一下这个索引在不在 uu（uu 中放着下一行一些元素的索引，这些元素具有这样的特点：已经和当前行某一个元素匹配上了。这个向量初始时存着 BBB）中，如果不在（kk==1），就认为这个索引对应的下一行的那个元素就是当前行元素照搬过去的，fd3 置为这个索引，并把这个索引加入 uu。最后第 61 行，把匹配到的下一行元素 (m+1,fd3)处的编码赋值给这一行

line 62: 由 B 矩阵的构造可知，其第一行放着每一个元素都对应文本中的一个字符，按照字符出现的频率由小到大排序，因此 W 矩阵中第一行就是每一个字符的哈夫曼编码，编码的顺序与字符的顺序一样，所以 W(1,x)就是 B(1,x)对应的哈夫曼编码，也就是出现频率第 x 低的字符的编码。把 W(1,:)这一行赋给数组 a，也就是 Huffman 函数的返回值

验证结果：

```
B =
     1     2     3     4     5
     3     3     4     5     0
     4     5     6     0     0
     6     9     0     0     0

W =
4×5 cell 数组
    {'111'}    {'110'}    {'10'}    {'01'}    {'00'}
    {'11'}    {'10'}    {'01'}    {'00'}    {'0'}
    {'01'}    {'00'}    {'1'}    {'0'}    {'0'}
    {'1'}    {'0'}    {'0'}    {'0'}    {'0'}

a =
1×5 cell 数组
    {'111'}    {'110'}    {'10'}    {'01'}    {'00'}
```

结果符合预期，说明生成哈夫曼编码函数编写正确

4、利用哈夫曼编码生成压缩后文本

调用完 Huffman 函数得到每个字符的编码之后，让我们回到 main 函数中

```

43-   huffman_result = Huffman(num);
44-   str_result = char(huffman_result); %主要是为了与Huffman计算出来的结果排序相对应（按字符出现次数
45-   output = ''; %初始化输出结果
46-   for i = 1:length(s) % s 为记录了从源文件中读出来的全部字符
47-       idx = find(b == s(1,i)); % 找到该字母在字母表b中的下标
48-       index = find(id == idx); % 用下标idx找到其在结构体排序索引表id中的下标（即按字母出现次数排列
49-       output = strcat(output, str_result(index, :)); % 从huffman编码结果中拼接字符串
50-   end
51-
52-   fid = fopen('HuffmanCode.txt','w');
53-   fprintf(fid, '%s\n', output);
54-   fclose(fid);

```

line 43: huffman_result 保存了 Huffman 函数的返回值

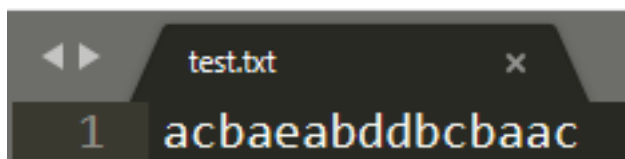
line 44: 由一个元胞向量变成一个 char 型矩阵, 按行从上到下为出现次数从少到多的字符的哈夫曼编码

line 45-50: s 为一行文本 'acbaeabddbcbaac', b 为 'abcde', 在 b 中找到 s 中每一个字符的位置记为 index, str_result 的 index 行即为该字符对应的 huffman 编码, 把这个编码粘贴到已有的 output 末尾。

line 52-54: 输出到指定文件

5、哈夫曼编码程序最终运行结果

测试用例 1:



test.txt

1 acbaeabddbcbaac

程序输出:



HuffmanCode.txt

1 001001001110001110110011001000010

测试用例 2:


```
test2.txt x
1 The report, a draft of which was obtained by The
Washington Post, is the first to analyze the
millions of posts provided by major technology
firms to the Senate Intelligence Committee.
```

程序输出：

```
HuffmanCode.txt x
1 010100110111110010100111101010100010100111000001100010000
000101000101001000001000111000100011000100101010001011011
101101110110010101000000001001001000101100011000000011110
10111010001001011000010000001010100110111100100001010000
001001101101111010011010110000110100010000100000011001110
00001100010111100100111010111100110001011101001100111000
111000010010000010100000001011010000011001111100111010111
110011000001110101101011011100011010100100100011000100101
010100011001110100100101010101001000101100100011101000111
101000100101100001000000110000000000110010100010100100111
011101101110111010000101011000101101001000000110001011101
001100001001001110000100111010111110010000101111110100000
011011100100001001101011011101011010110111011010111101001
10111110010000111000011000010000011111011011111100001111
```

结果符合预期，说明程序正确

6、利用哈夫曼编码进行解码

得到哈夫曼编码之后进行译码

(1) 首先在主函数中通过 for 循环得到 sort_chars 数组，用于存储按照出现次数升序排序的字符；调用 decode 函数进行译码，参数 huffman_result 为按照字符出现次数升序排序的对应哈夫曼编码数组，output 为得到的源文件的哈夫曼编码序列

(2) decode 函数：返回的 content 为译码结果数组

```

1 % input: huffman code and the matrix produced by Huffman.m
2 %       original_chars is an array of sorted original characters
3 % output: the original content of huffman code
4
5 % 2018/12/17 Zehui Jin
6
7 function [content] = decode(huffman_result, huffman_code, original_chars)
8
9     content = '';
10    token = '';
11    Length = length(huffman_code);
12    Count = length(huffman_result);
13    for i=1:Length
14        char = huffman_code(i); % read next number
15        token = strcat(token, char); % strcat
16        for j=1:Count
17            result = strcmp(token, huffman_result(j));
18            if(result==1)
19                content = strcat(content, original_chars(j));
20                token = ''; % clear token
21            end
22        end
23    end
24
25 end
26

```

content: 记录哈夫曼译码结果

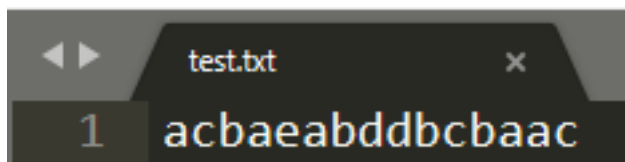
token: 临时数组，记录哈夫曼编码组合

外层 for 循环: 遍历哈夫曼编码数组，将读到的编码拼接到 token

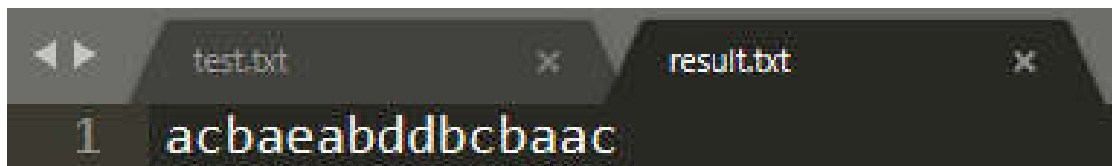
内层循环: 遍历哈夫曼编码元胞数组，对比每一个编码组合与当前 token 是否相同，若相同则得到对应的字符并拼接到 content 中，否则读取下一个编码

(3) 解码程序输出:

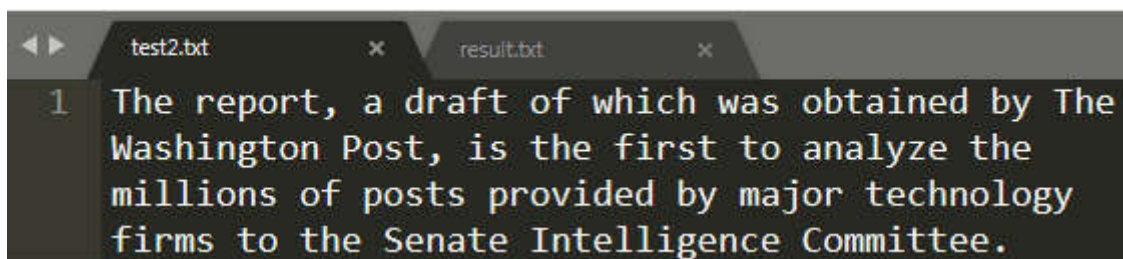
测试用例 1:



程序输出:

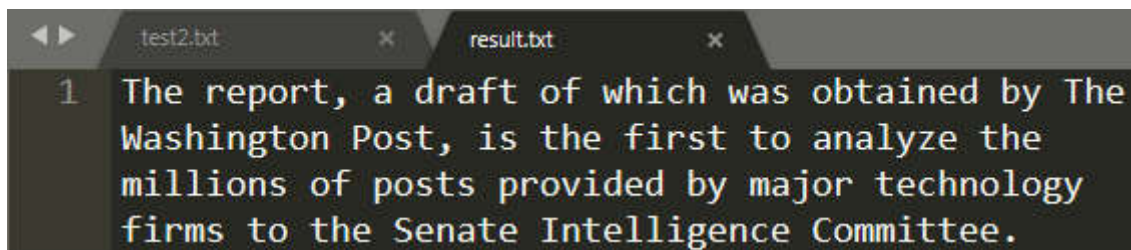


测试用例 2:



```
1 The report, a draft of which was obtained by The
Washington Post, is the first to analyze the
millions of posts provided by major technology
firms to the Senate Intelligence Committee.
```

程序输出：



```
1 The report, a draft of which was obtained by The
Washington Post, is the first to analyze the
millions of posts provided by major technology
firms to the Senate Intelligence Committee.
```

结果符合预期，说明程序运行结果正确。

综上，我们在本实验中利用 MATLAB 实现了对一个给定文本的编码、译码工作。

四、组内同学承担任务说明

	组内工作任务	贡献百分比
陈麒先	Huffman 编码过程的 MATLAB 源代码实现； 为编码过程部分编写详细注释； 为 MATLAB 源程序编写使用说明与测试说明文档； 完成了实验报告最终版的审查与修订； 完成了课堂汇报 PPT 最终版的审查与修订； 进行课堂汇报与展示。	25.00%
金泽晖	Huffman 译码过程的 MATLAB 源代码实现； 团队队长，统筹规划团队工作； 创建并维护 github 项目开展团队协作 在实验报告第 1 版的基础上补充与完善了译码部分； 进行课堂汇报与展示。	25.00%
王润泽	团队 QA，为工程构造测试集并开展系统性测试； 对 MATLAB 源代码进行调试并对其中错误进行修改； 完成实验报告的编写工作； 参与各个版本的实验报告的修订工作； 完成课堂展示 PPT 的制作工作； 参与最终提交成果的审核与修订工作。	25.00%
李贞子	团队 QC，为工程提出可行的优化方案； 依据优化方案对 MATLAB 源代码进行优化； 完成实验报告的编写工作； 参与各个版本的实验报告的修订工作； 完成课堂展示 PPT 的制作工作； 参与最终提交成果的审核与修订工作。	25.00%