实验报告

主题：Huffman编解码

09组

组员：陈麒先 金泽晖 李贞子 王润泽

1. ***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。

1. ***问题分析***

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**

1. **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=（W1\*L1+W2\*L2+W3\*L3+...+Wn\*Ln），N个权值Wi（i=1,2,...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Li（i=1,2,...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 w1、w2、…、wn，则哈夫曼树的构造规则为：

(1) 将w1、w2、…，wn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。

1. **利用哈夫曼树进行编码**

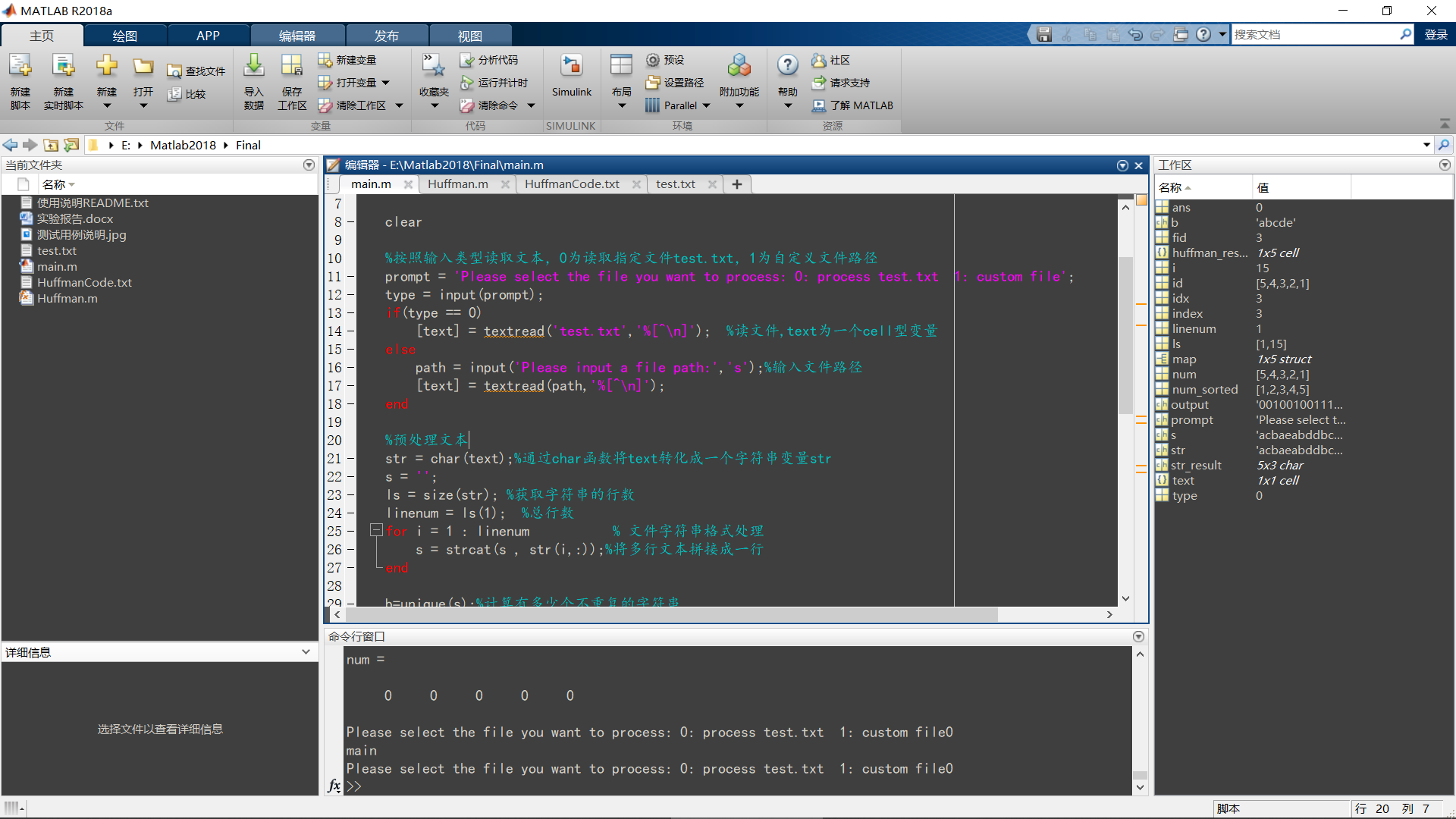
得到哈夫曼树后，利用中序遍历或前序遍历扫描整棵树，在根节点时编码字符串为空，之后每进入一个节点的左子树时给编码字符串补’1’，进入右子树时给编码字符串补’0’, 直到扫描到叶节点，此时这个叶节点对应的编码就是编码字符串。返回父节点时去掉补上得字符即可。通过这样一个递归的过程就得到了每一个字符的哈夫曼编码。再将原文本的每一个字符都转化成哈夫曼编码，从而生成一个压缩后的只有0和1的编码文件。

1. **利用哈夫曼树进行解码**
2. ***利用MATLAB解决哈夫曼编解码***

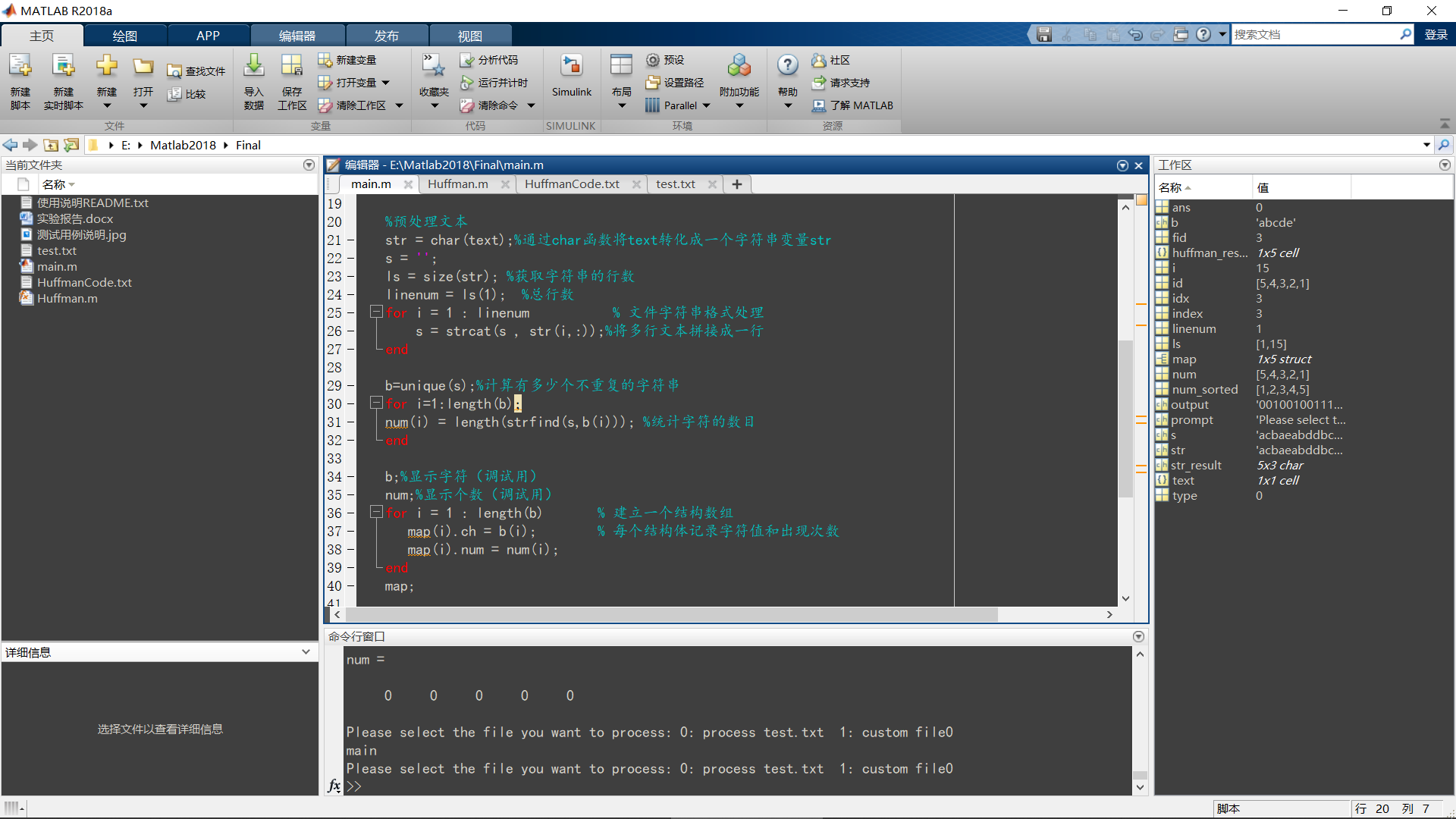
为方便叙述，以待压缩文本acbaeabddbcbaac为例阐述利用MATLAB实现哈夫曼编码解码的过程

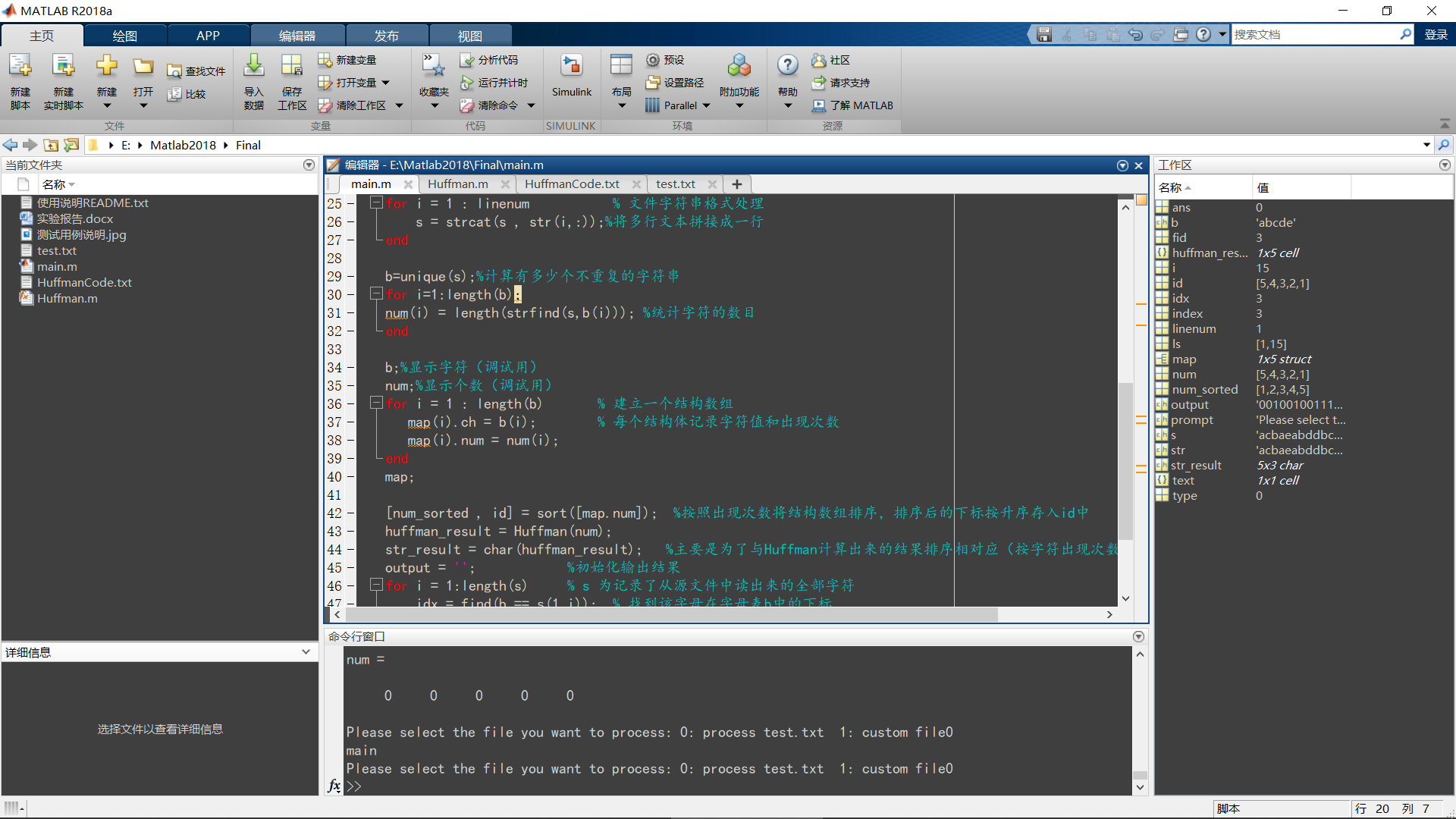
1. **预处理输入**

（1）读取输入文件



（2）将输入文件转化为一行字符串

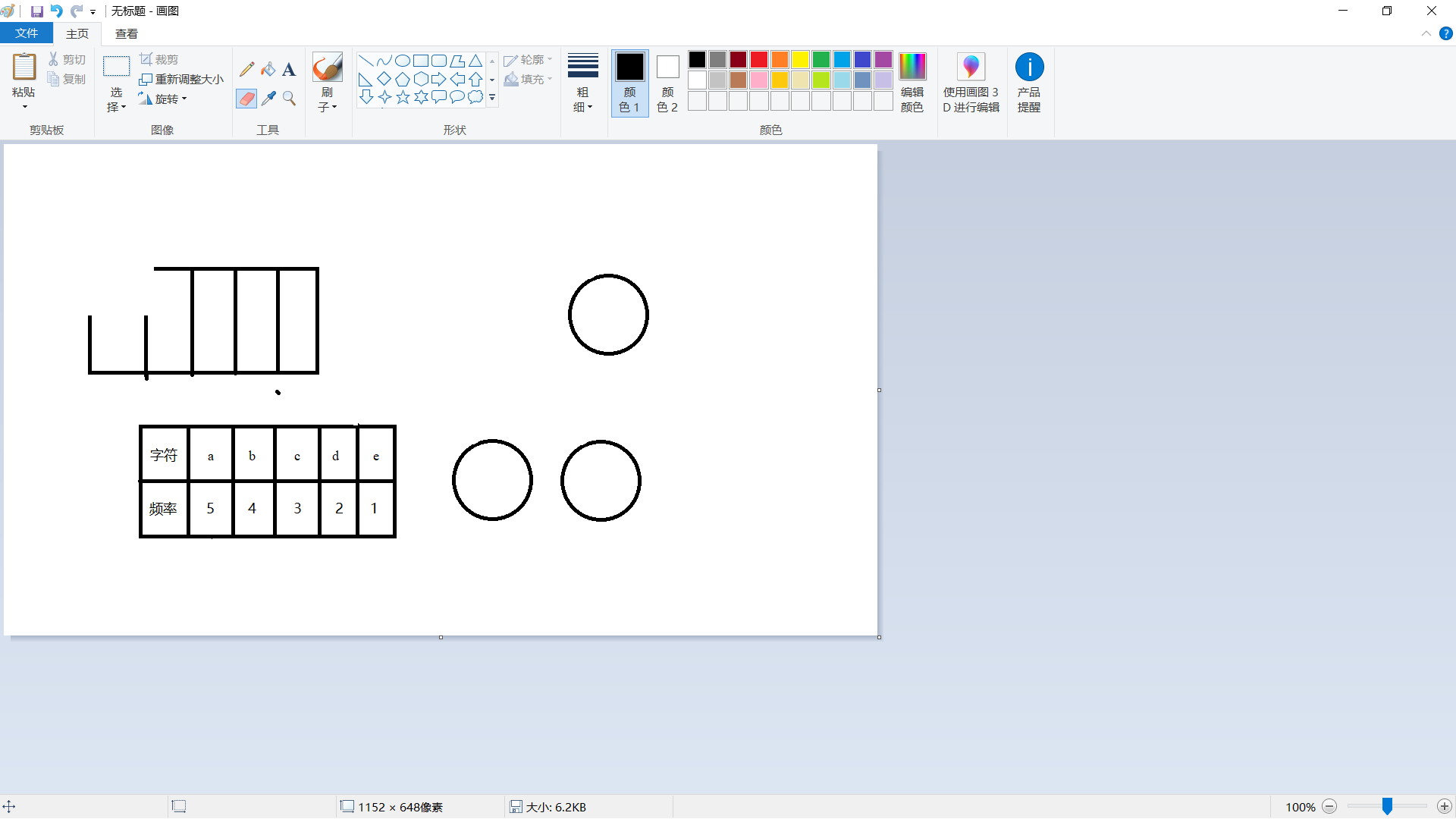


（3）记录节点的信息，调用Huffman函数

1. **构造哈夫曼树**

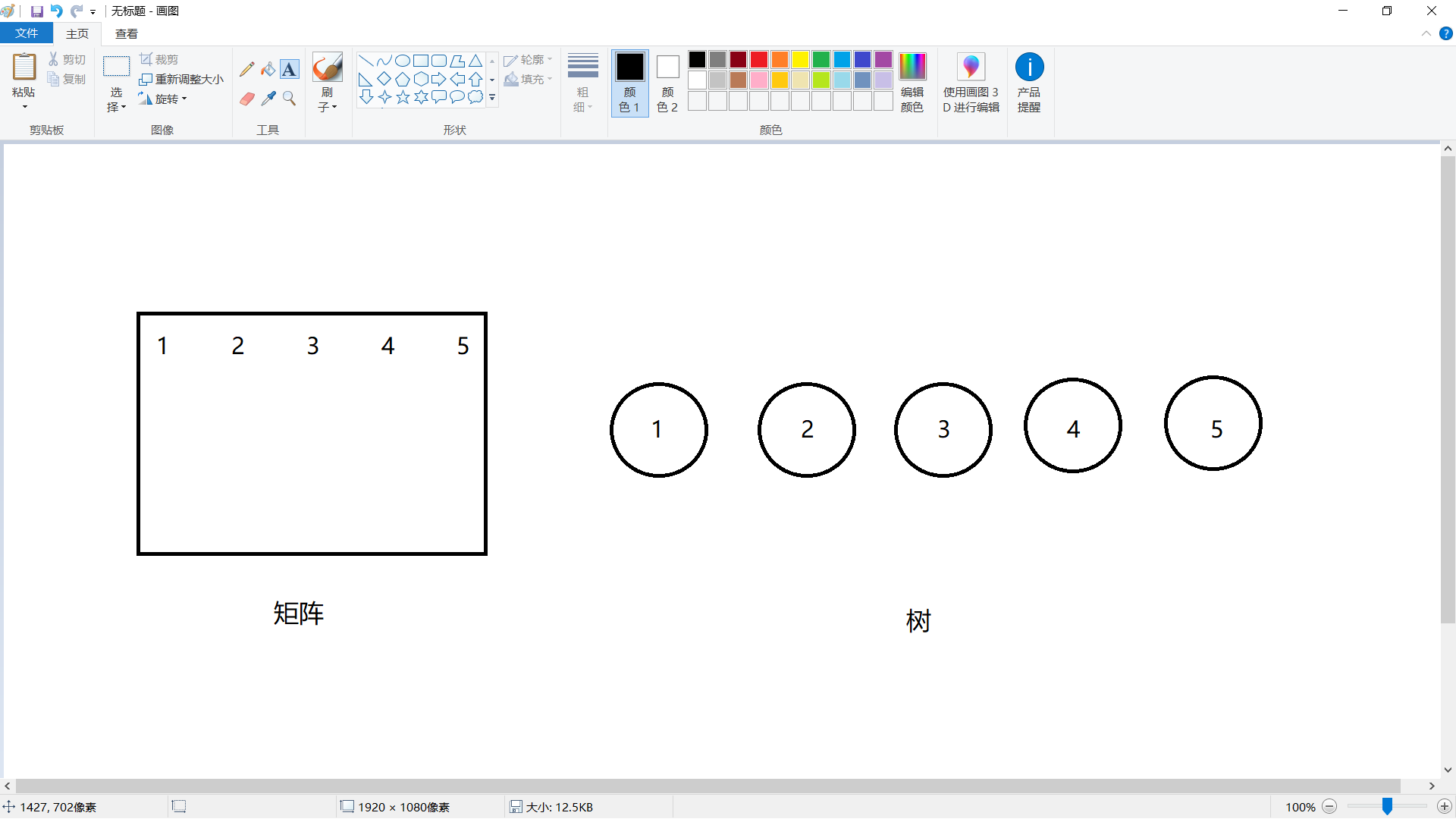
MATLAB中并没有类似C语言中指针的结构，进而无法构造树来实现算法。这一点带来了不小的困难。最终我们小组决定利用**矩阵**来模拟哈夫曼树的生成过程

具体算法如下

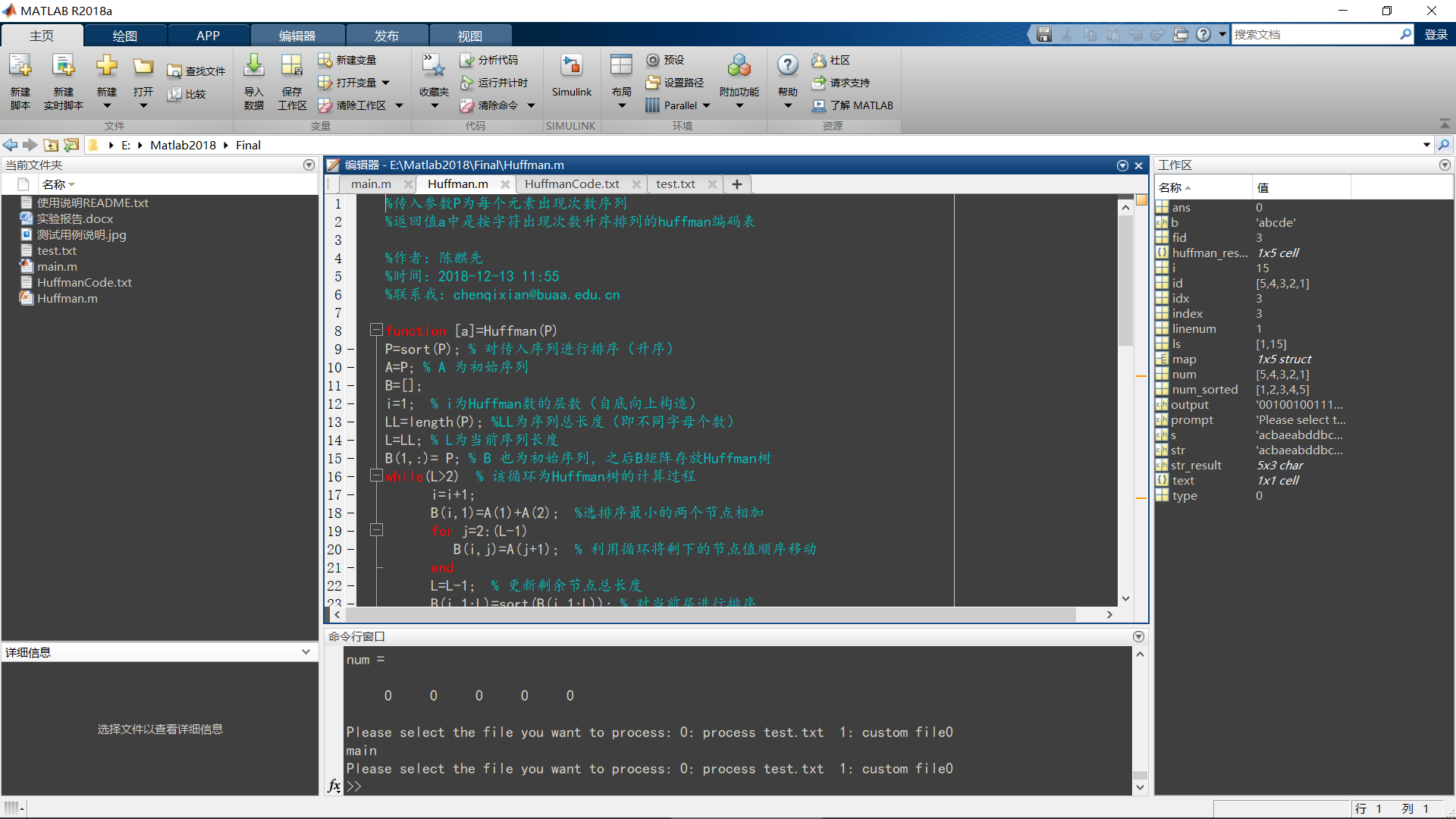


初始情况

（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



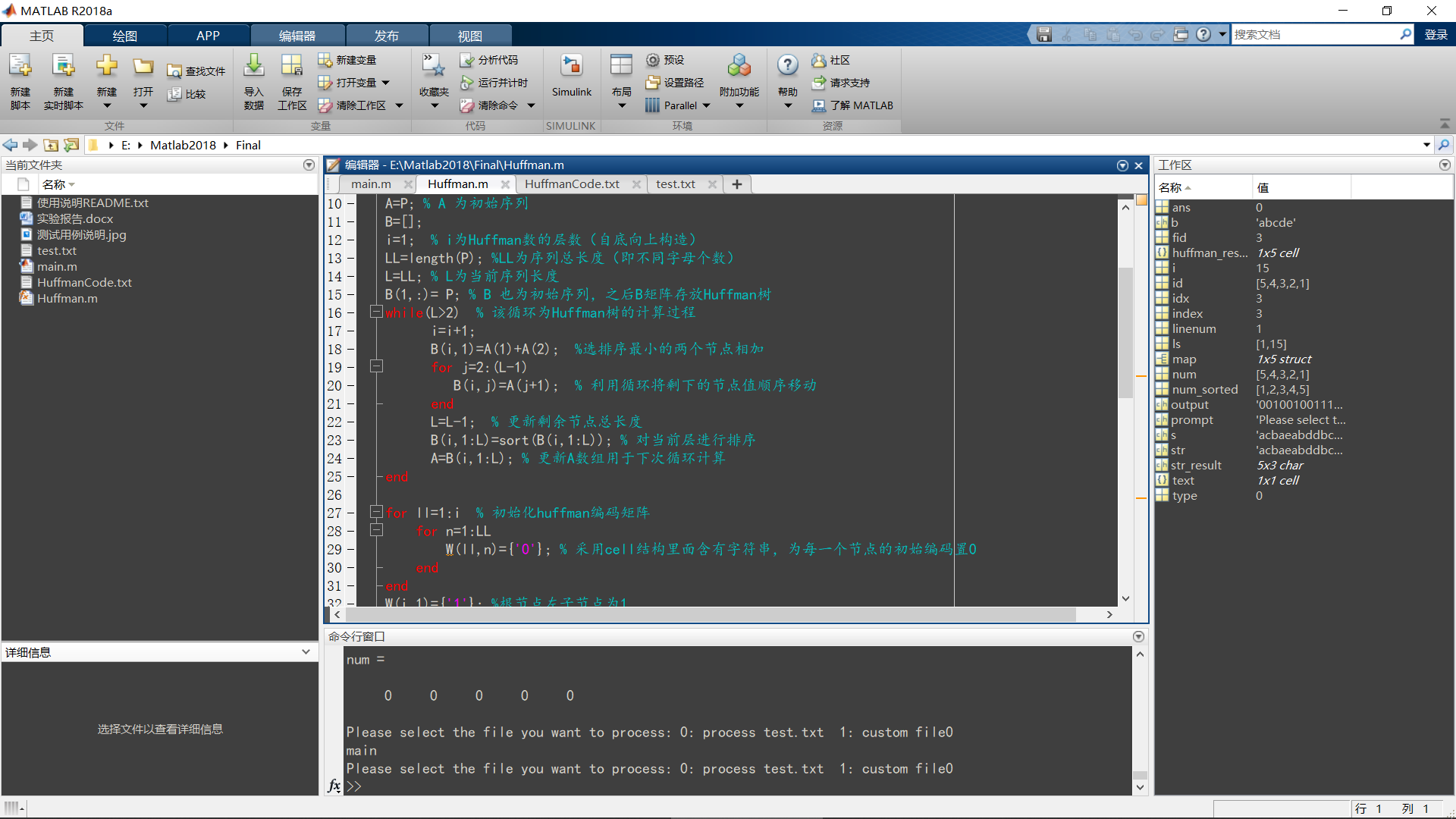
**相应代码**：Huffman为计算哈夫曼树的函数，函数参数P为排序后的出现次数序列，返回一个记录每个字符对应编码的矩阵a



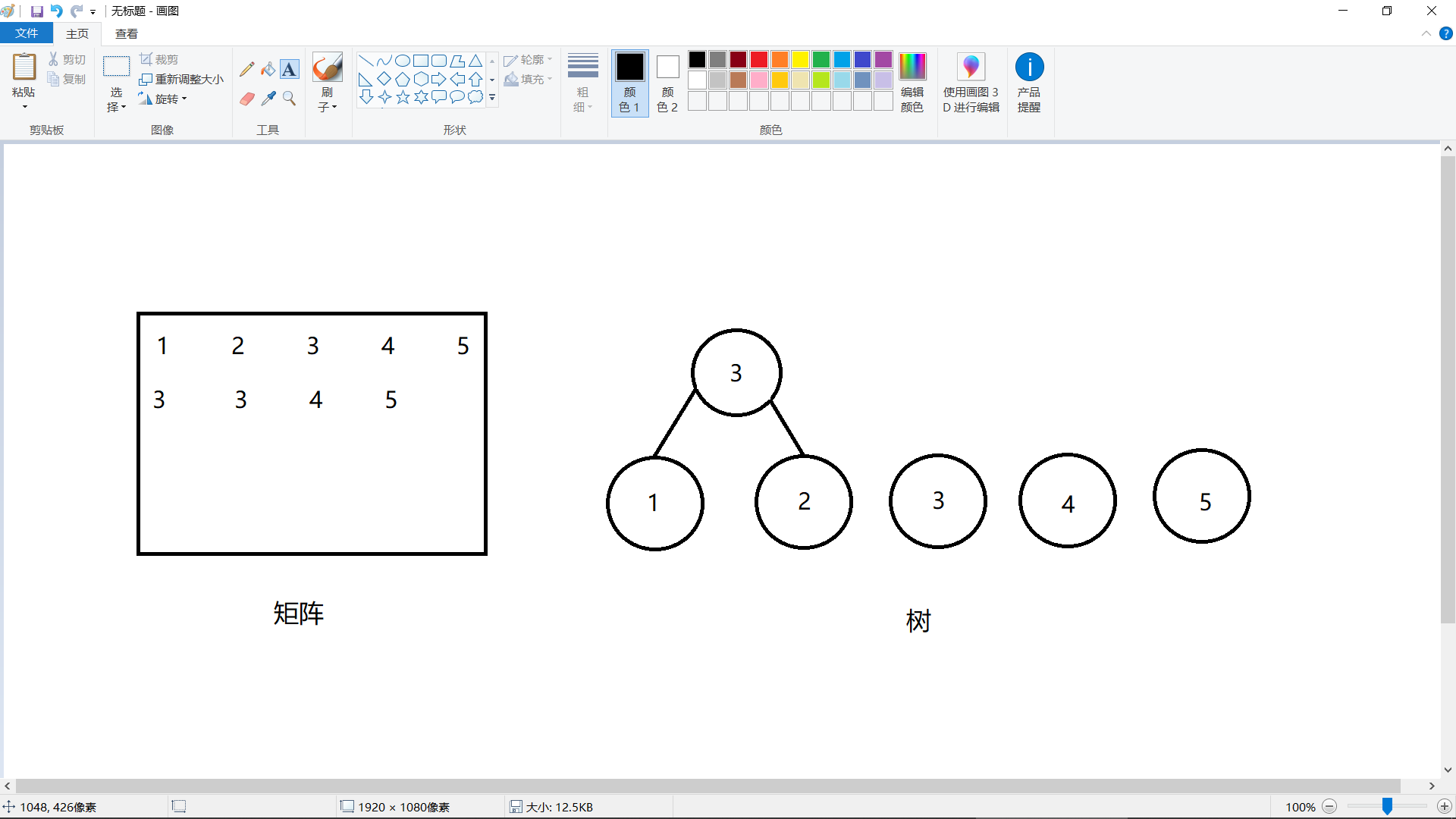
（2）执行以下步骤

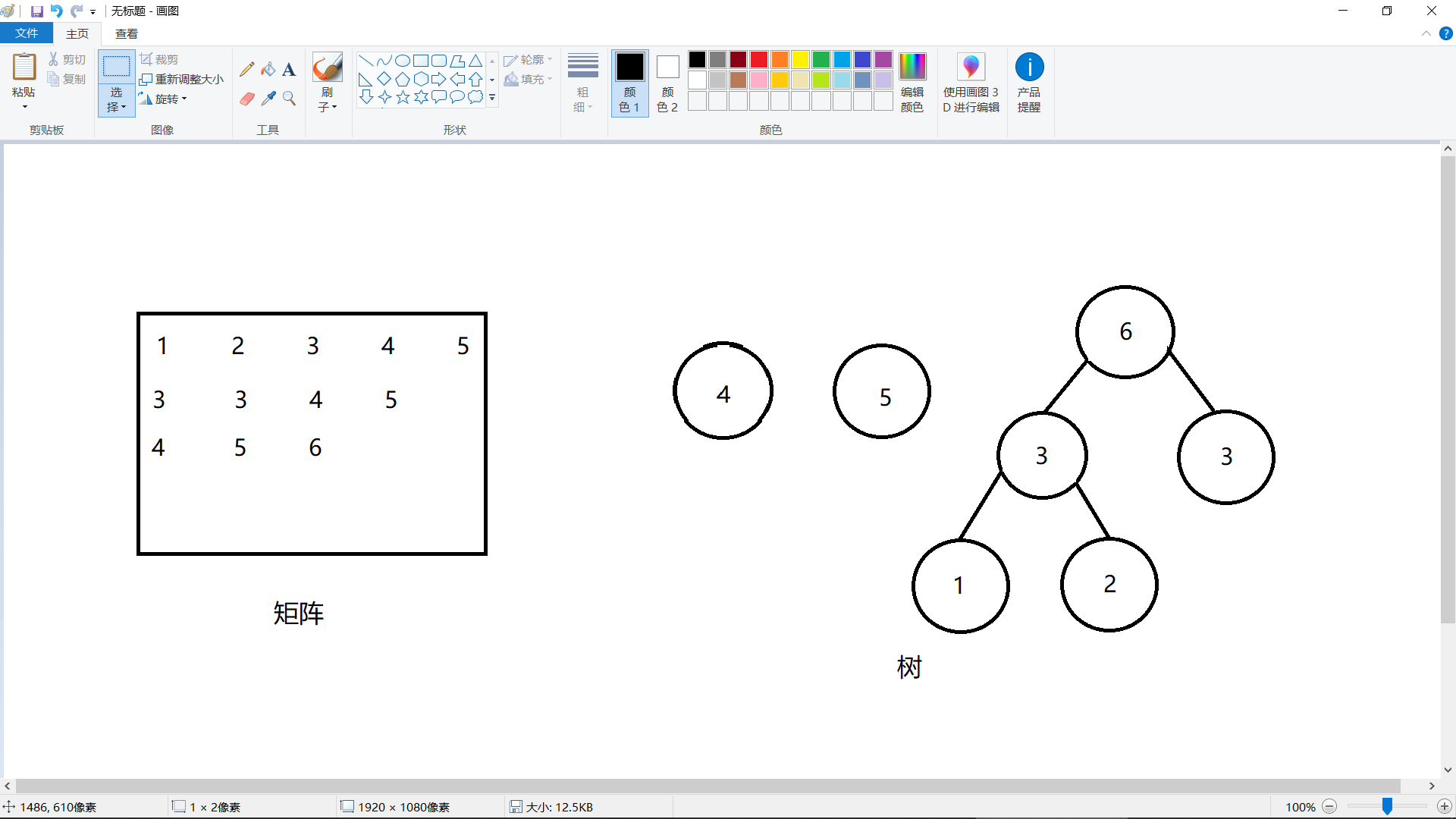
* 每次将矩阵最后一行的前两个数相加，结果放入新的一行的行首
* 将上一行的其它元素照搬下来
* 将新生成的一行从小到大排序
* 若新的一行的元素个数大于2，返回第一步
* 直到最后一行只剩两个数，这两个数就是哈夫曼树的根节点的左右子节点，两个数的和就是哈夫曼树的根节点

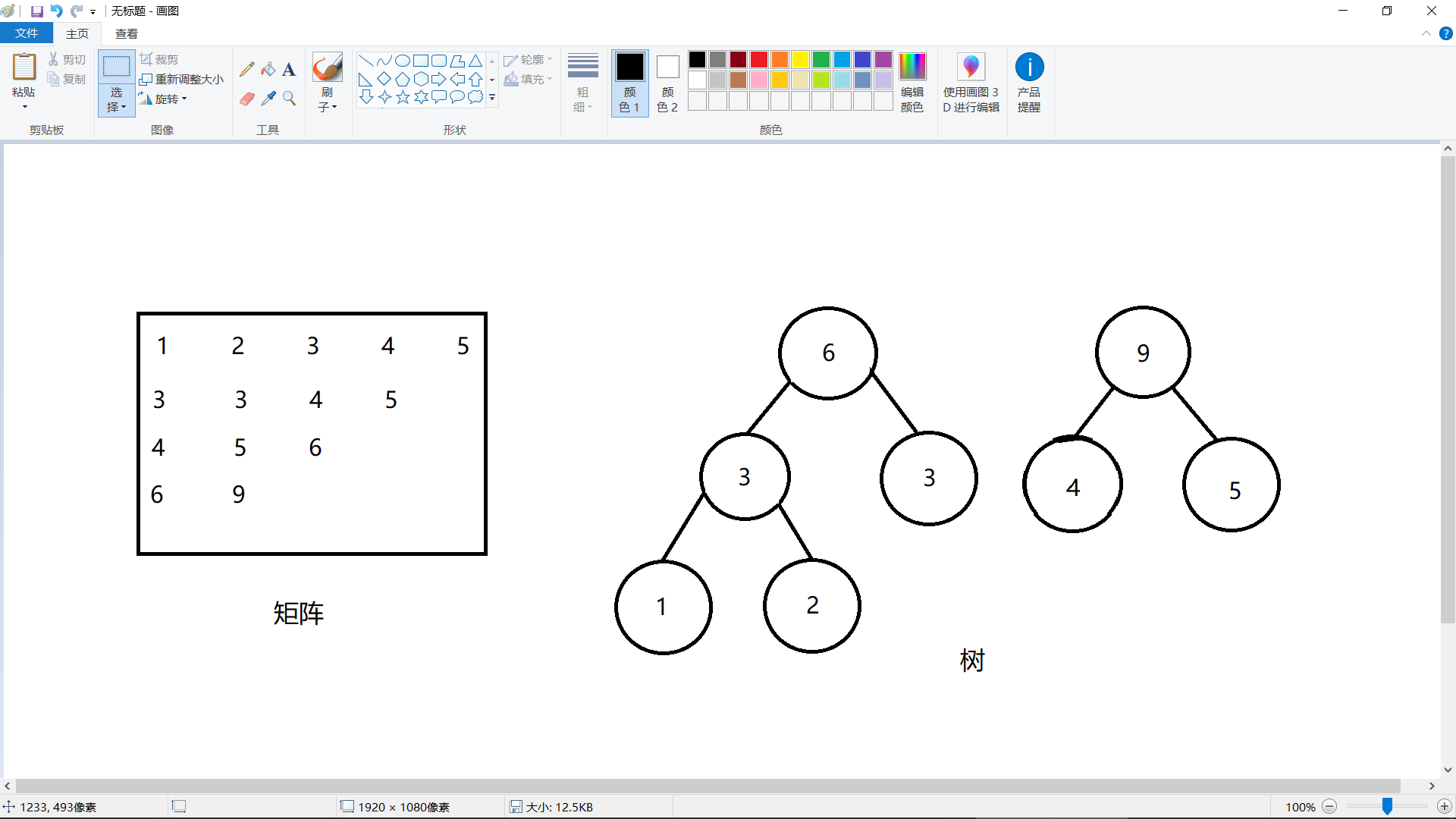
**相应代码**：

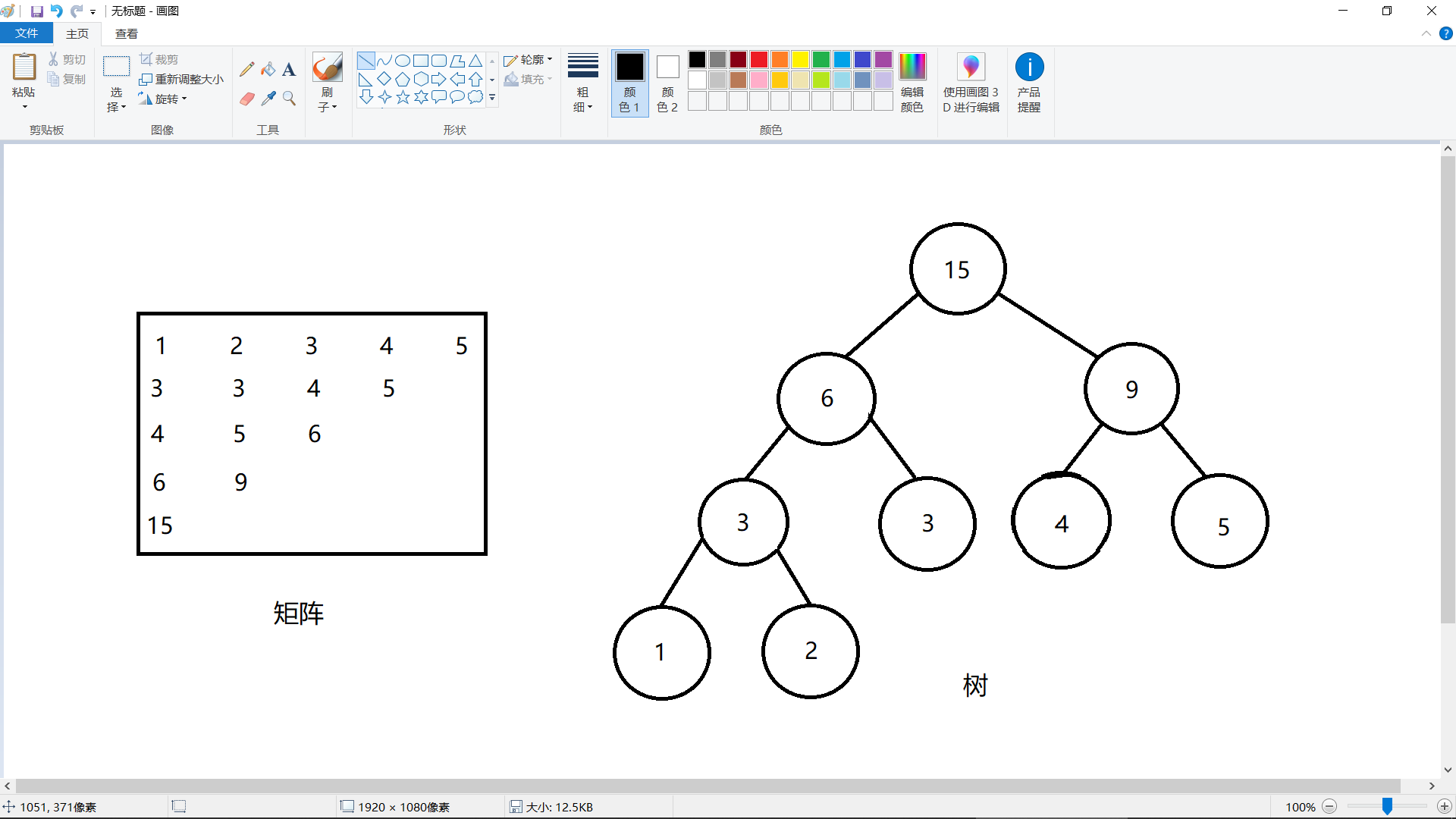


**演示图例**：









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：

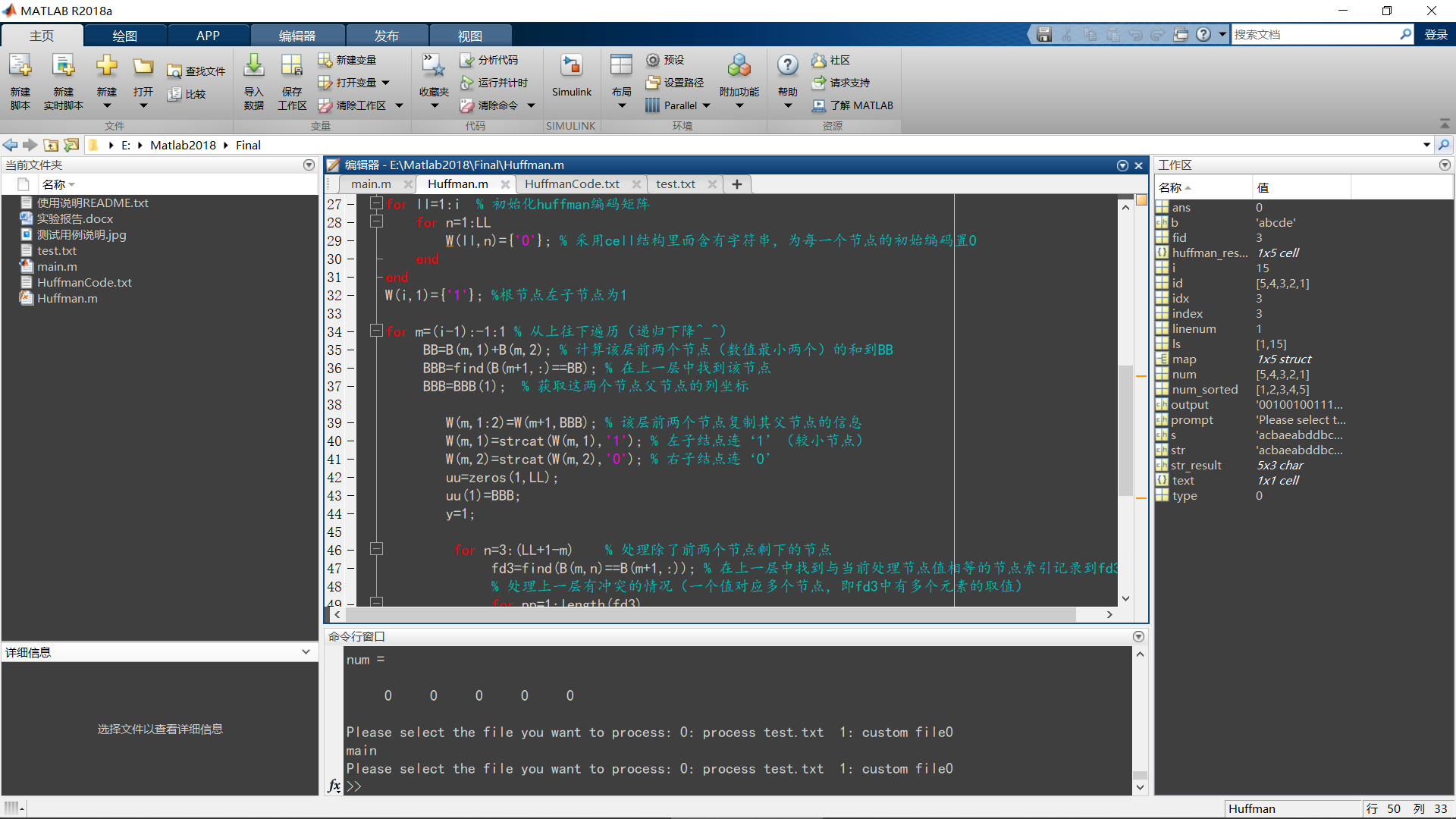
* 每次选取最后一行的前两个字符相加相当于为当前森林中最小的两个根节点生成一个父节点并且将两棵树连接起来，这个新树的根节点的值就是原来两个根节点值的和。
* 排序是为了使得下一次要取的根节点（当前森林中最小的两个根节点）就位于最后一行的前两个，省去了遍历下一行选出两个最小节点的过程。
* 将最后一行第三个及之后的元素照搬到下一行代表对当前森林中不是最小两个根节点的其他根节点不做处理
* 矩阵并没有包括最后的根节点（最后一行有两个树），因为到最后一步时没有必要计算出根节点的值并加进矩阵，只要比较最后一行两个数的大小就可以明白哪一个是左子树哪一个是右子树

**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。**

1. **利用哈夫曼树进行编码**

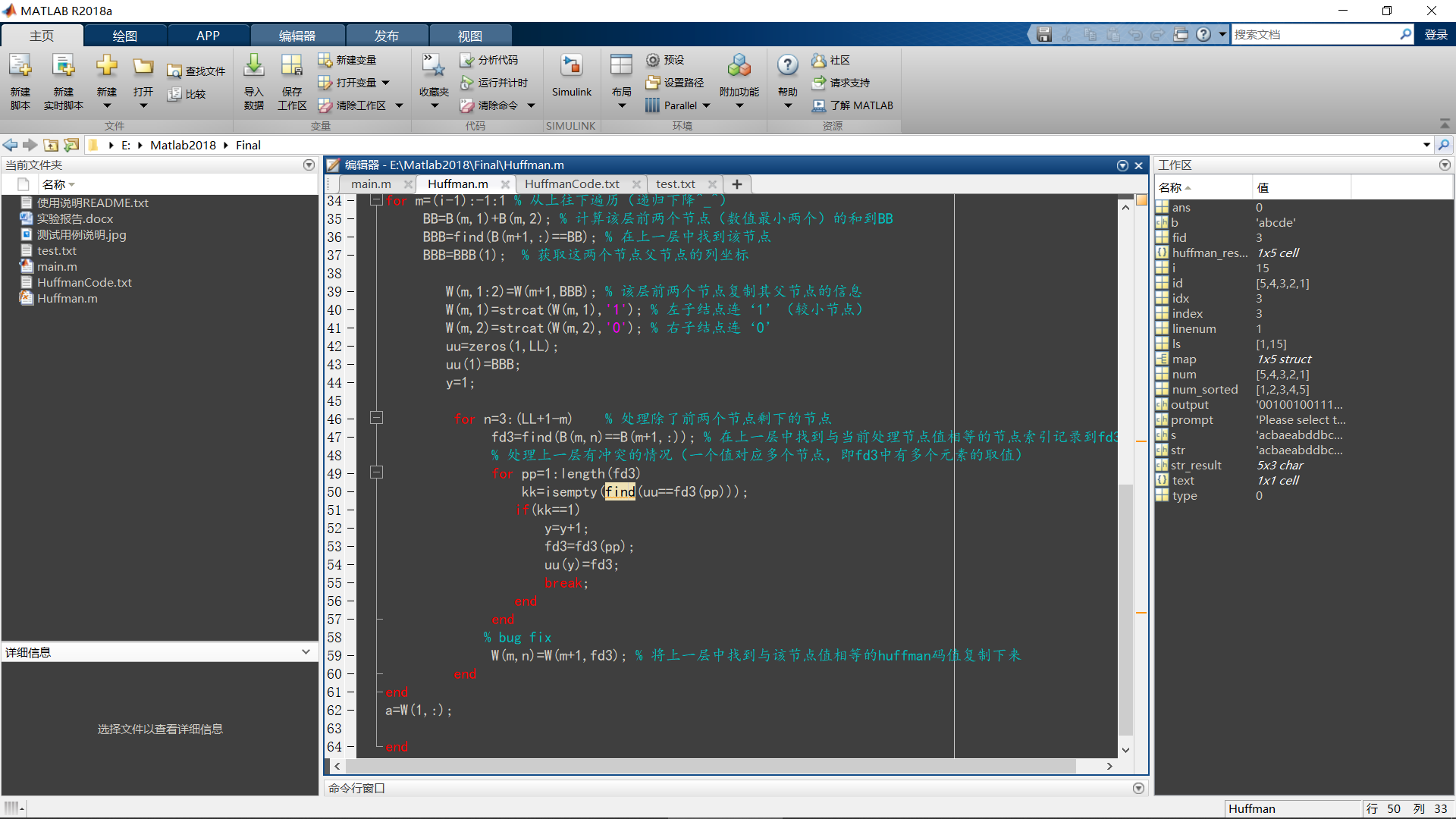
完成哈夫曼树的构建以后，下一步就是利用哈夫曼树生成每一个字符的哈夫曼编码。刚才生成的哈夫曼矩阵记为B

（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。



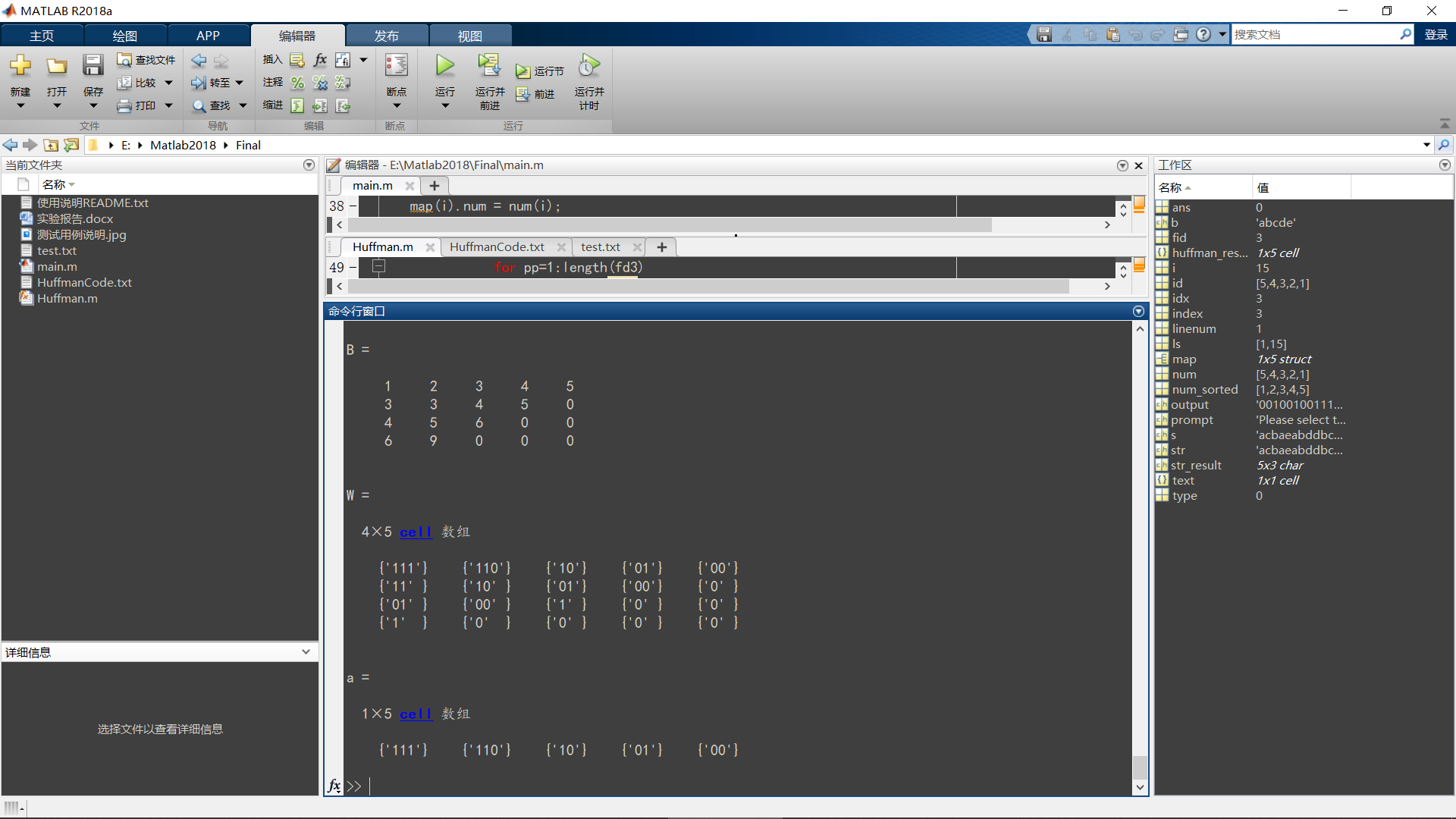
第i行为根的左右子节点

（2）遍历哈夫曼树生成编码



* 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

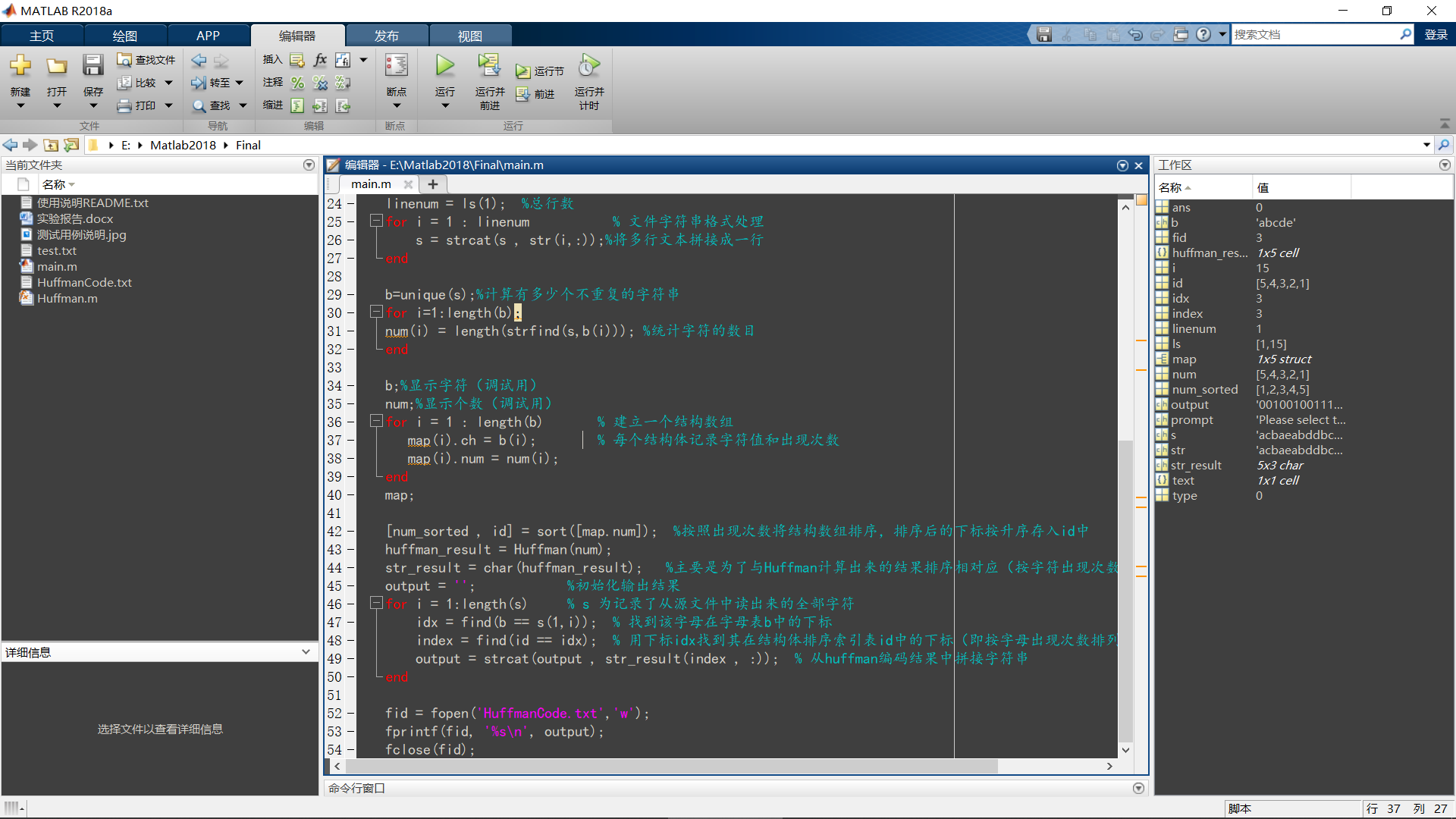
验证结果：



结果符合预期，说明生成哈夫曼编码函数编写正确

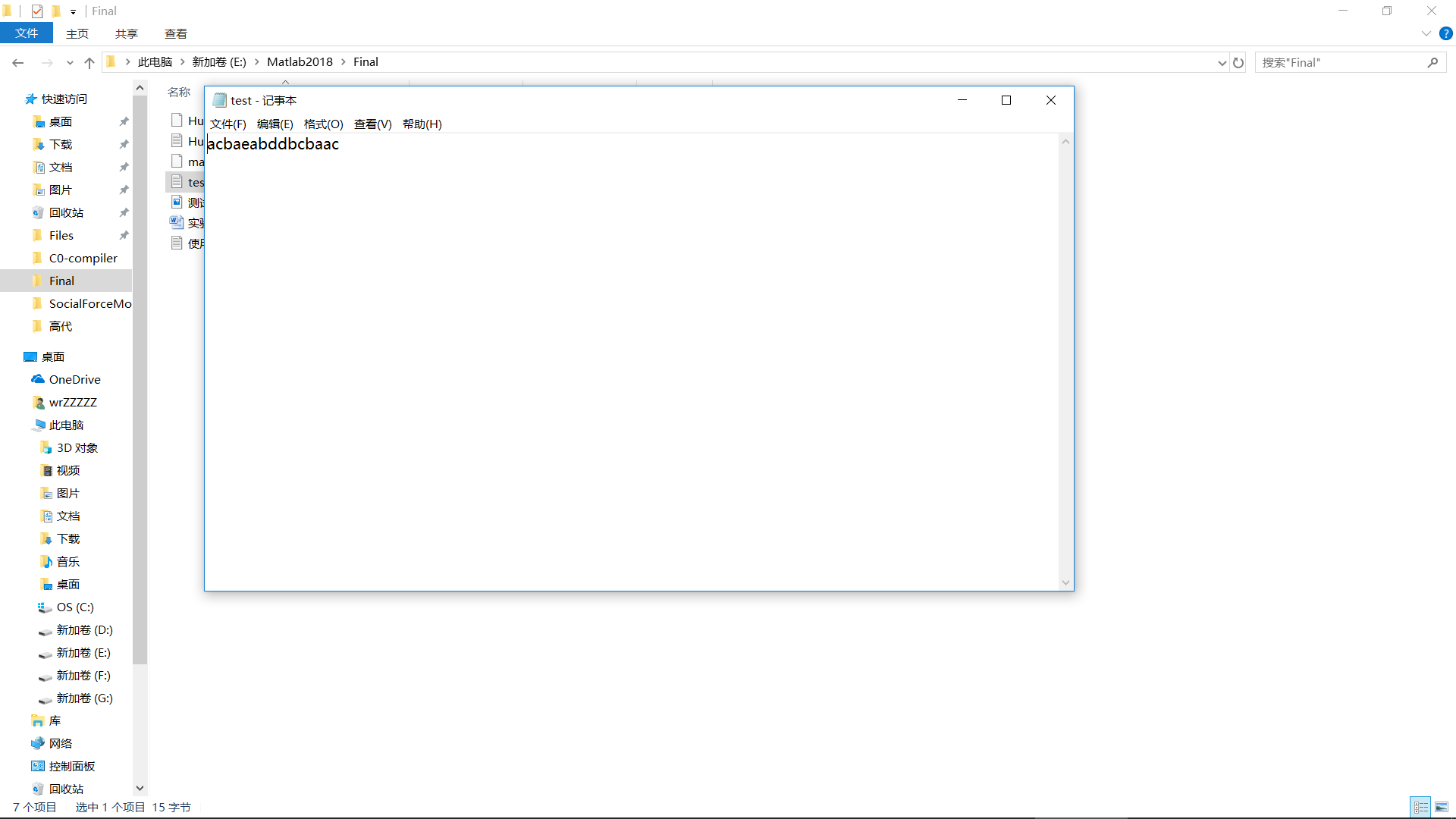
1. **利用哈夫曼编码生成压缩后文本**

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

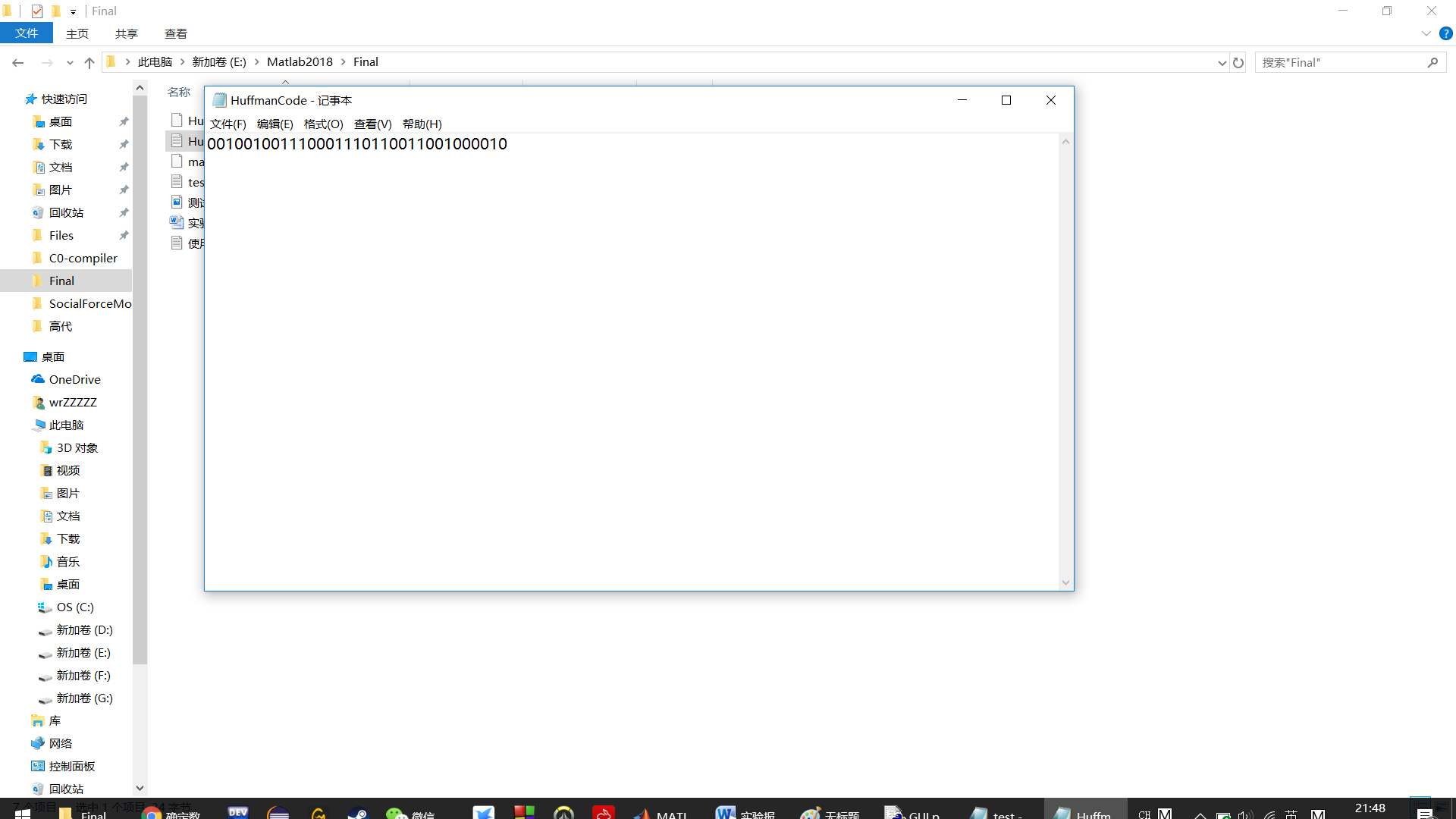


* line 43: huffman\_result保存了Huffman函数的返回值
* line 44:由一个元胞向量变成一个char型矩阵，按行从上到下为出现次数从少到多的字符的哈夫曼编码
* line 45-50: s为一行文本’ acbaeabddbcbaac’，b为’abcde’,在b中找到s中每一个字符的位置记为index，str\_result的index行即为该字符对应的huffman编码，把这个编码粘贴到已有的output末尾。
* line 52-54: 输出到指定文件

1. **哈夫曼编码程序最终运行结果**

程序输入：test.txt，然后在控制台输入0

程序输出：HuffmanCode.txt



结果符合预期，说明程序正确

1. **利用哈夫曼编码进行解码**
2. ***人员分工***