合肥工业大学

《信息论与编码》

课程设计

专业班级：信息安全

姓　　名：Martin

学　　号：

指导教师：

课设选题：

2021-2022 学年第二学期

目录

一、设计要求

1. 必做题

对任意输入的字符串序列分别进行二元霍夫曼编码、fano 编码、游程编码和算术编码，给出编码结果、编码效率；并实现相应的译码操作。

2. 提升题

对一幅 BMP 格式的灰度图像先进行二元霍夫曼编码和游程编码，并根据霍夫曼编码结果将游程编码变换成二进制序列。(象素用霍夫曼编码,游程用等长码)。并设计相应的译码。

二、开发环境与工具

编辑工具：Visual Studio Code

编译工具：python 3.9.7

界面工具：PyQt5 designer

用到的库：math; Pillow 8.4.0; PyQt5 5.15.4;

三、设计原理

1. Huffman 编码原理：

哈夫曼编码的基本思想是以字符的使用频率作为权值构建一颗哈夫曼树,然后利用哈夫曼树对字符进行编码。构造的一棵哈夫曼树,是将要编码的字符作为叶子节点,将该字符在文件中的使用频率作为叶子节点的权值,以自底向上的方式,通过 n-1 次合并运算后构造出的树。其核心思想是让权值大的叶子离根最近。以二元霍夫曼编码为例题，下图为求解过程。

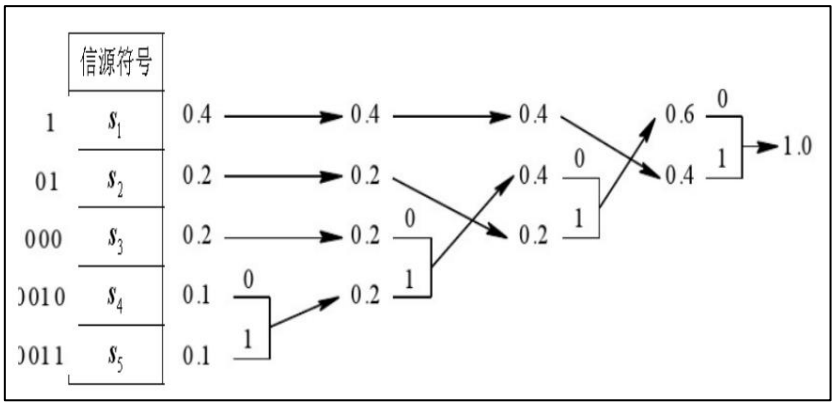


图 1 二元霍夫曼编码示例图

说明：1、每次对缩减信源两个概率最小的符号分配“0”″和“1”是任意的，所以可得到不同的码字。2、缩减信源时，若合并后的新符号概率与其他符号概率相等，从编码方法上来说，这几个符号的次序可任意排列，编出的码都是正确的，但得到的码字不相同。

平均码长计算公式为：

$$\bar{L} = \sum p(a_i) * l_i \qquad (1)$$

编码效率计算公式为

$$\eta = \frac{H(S)}{\bar{L}} \qquad (2)$$

式中，$p(a_i)$ 为对应字符出现的概率，l_i 为相应字符的码长，H(S)为信源的熵，η 为编码效率。

2. Fano 编码原理

费诺编码属于统计匹配编码，但它一般不是最佳的编码方法。编码步骤为：(1)将信源消息(符号)按其出现的概率由大到小依次排列；(2)将依次排列的信源符号按概率值分为两大组，使两个组的概率之和近于相同，并对各组分别赋予一个二进制码元“0”和“1”；(3)将每一大组的信源符号进一步再分成两组，使划分后的两个组的概率之和近于相同，并又分别赋予一个二进制符号“0”和“1”；(4)如此重复，直至每个组只剩下一个信源符号为止；(5)信源符号所对应的码字即为费诺码。费诺码考虑了信源的统计特性，使经常出现的信源符号能对应码长短的编码字。显然，费诺码仍然是一种相当好的编码方法。但是，这种编码方法不一定能使短码得到充分利用。尤其当信源符号较多，并有一些符号概率分布很接近时，分两大组的组合方法就很多。可能某种分配结果，会出现后面小组的“概率和”相差较远，因而使平均码长增加，所以费诺码不一定是最佳码。

<table>
<tr><th>信源符号</th><th>概率</th><th colspan="4">编码</th><th>码字</th><th>码长</th></tr>
<tr><td>x_1</td><td>0.32</td><td rowspan="2">0</td><td>0</td><td colspan="2" rowspan="3"></td><td>00</td><td>2</td></tr>
<tr><td>x_2</td><td>0.22</td><td>1</td><td>01</td><td>2</td></tr>
<tr><td>x_3</td><td>0.18</td><td rowspan="4">1</td><td>0</td><td>10</td><td>2</td></tr>
<tr><td>x_4</td><td>0.16</td><td rowspan="3">1</td><td>0</td><td></td><td>110</td><td>3</td></tr>
<tr><td>x_5</td><td>0.08</td><td rowspan="2">1</td><td>0</td><td>1110</td><td>4</td></tr>
<tr><td>x_6</td><td>0.04</td><td>1</td><td>1111</td><td>4</td></tr>
</table>

图 2 二元 Fano 编码示例图

平均码长计算公式为：

$$\bar{L} = \sum p(a_i) * l_i \quad (3)$$

编码效率计算公式为

$$\eta = \frac{H(S)}{\bar{L}} \quad (4)$$

式中，$p(a_i)$ 为对应字符出现的概率，l_i 为相应字符的码长，H(S)为信源的熵，η 为编码效率。

3. 游程编码原理

游程编码是相对简单的编码技术，主要思路是将一个相同值的连续串用一个代表值和串长来代替。例如，有一个字符串“aaabccddddd”，经过游程编码后可以用“3a1b2c5d”来表示。对图像编码来说，可以定义沿特定方向上具有相同灰度值的相邻像素为一轮，其延续长度称为延续的行程，简称为行程或游程。例如，若沿水平方向有一串 M 个像素具有相同的灰度 N，则行程编码后，只传递 2 个值（N,M）就可以代替 M 个像素的 M 个灰度值 N.

4. 算数编码原理

计算信源符号序列的累积分布函数，使每组符号序列对应于累积分布函数上不同的区间，再在区间内取一点，将其二进制小数点后 1 位作为符号序列的码字。计算信源符号序列的累积分布函数，使每组符号序列对应于累积分布函数上不同的区间，再在区间内取一点，将其二进制小数点后 1 位作为符号序列的码字.

假设有一段数据需要编码，统计里面所有的字符和出现的次数。将区间 [0,1) 连续划分成多个**子区间**，每个子区间代表一个上述字符， 区间的大小正比于这个字符在文中出现的概率 p。概率越大，则区间越大。所有的子区间加起来正好是 [0,1)。编码从一个初始区间 [0,1) 开始，设置:，不断读入原始数据的字符，找到这个字符所在的区间，比如 [L, H)，更新。

设原字符串长度为 lenstr,编码后的长度为 lenstr2,累计概率为 P

$$P = \prod p_i \quad (5)$$

$$lenstr2 = \left\lceil \log_2 \frac{1}{P} \right\rceil \quad (6)$$

平均码长计算公式为：

$$\bar{L} = \frac{lenstr}{lenstr2} \quad (7)$$

编码效率计算公式为

$$\eta = \frac{H(S)}{\bar{L}} \qquad (8)$$

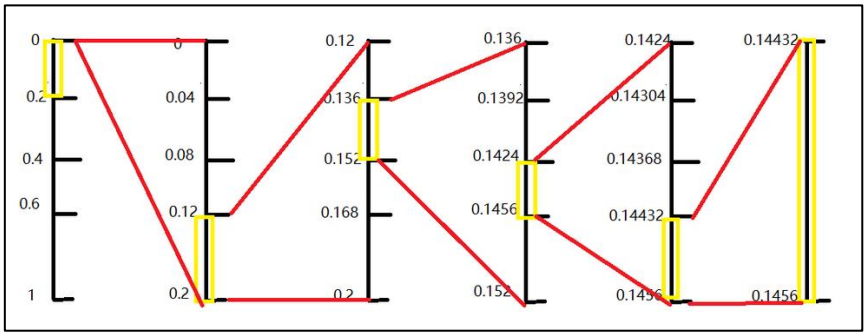


图 3 二元算数编码示例图

5. 灰度图像 Huffman+游程编码原理

像素值用 Huffman 编码，连续出现的次数用游程编码，以下图为例：

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

初步思想为统计像素值和出现个数：（5，3）（3，1）（6，4）（6，2）（4，2）（2，3）（3，1）。进一步，将像素值用哈夫曼编码，长度用等长游程编码。

四、系统功能描述及软件模块划分

由下图可见，有五个按钮，分别实现五个题目对应的功能，分别调用五个 python 文件来进行对应的编码。在文本框中输入被编码的字符串，点击不同的按钮，即可得到字符及其对应的编码，编码结果，译码结果和编码效率。Huffman 编码为 Huffman.py，Fano 编码为 fano.py，游程编码为 run_length2.py，算数编码为 signal.py，灰度图像编码为 bmp_huffman.py，主界面源文件为 main.ui 和 UI_main.py，主函数为 main.py 负责将编码程序和界面程序统一管理。下图为软件主窗口图：

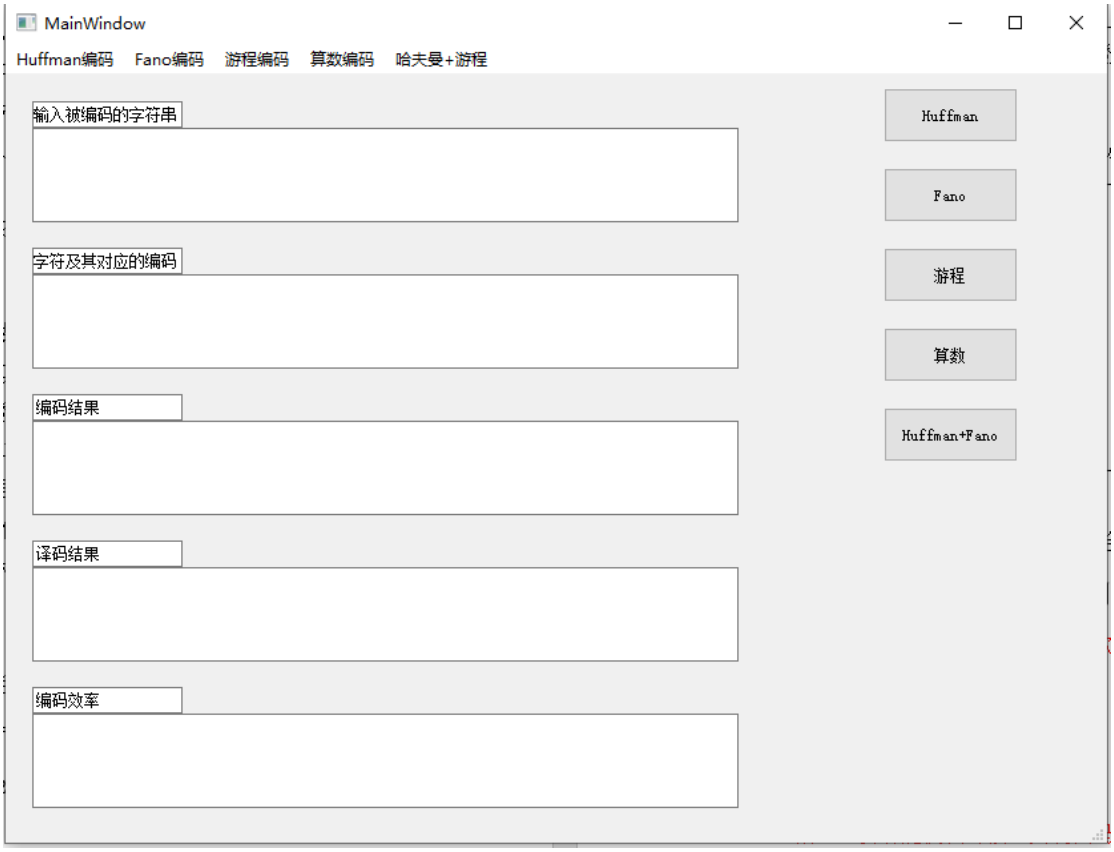


图 4 软件主窗口图

以下为各个子文件的模块划分：

1. Huffman.py 模块的划分

函数名称	函数输入	函数输出
Class node	Huffman 结点参数	一个结点
get_text_count()	被编码的字符串	字符和相应概率
create_nodes()	字符串的字符	结点列表
create_huffman_tree()	结点列表	Huffman 树和根结点
huffman_encoding()	字符频率和 Huffman 树	每个结点被编成的字符串和平均码长
encode_str()	文本和结点列表	被编码的字符串
decode_str()	被编码的字符串和结点列表	解码后的字符串
get_H()	字符和相应的概率	信源的熵

表 1 Huffman.py 模块的划分

2. fano.py 模块的划分

函数名称	函数输入	函数输出
Class node	fano 结点参数	一个结点
get_text_count()	被编码的字符串	字符和相应概率
create_nodes()	字符串的字符	结点列表
fano_encode()	结点列表和字符相应概率	每个结点被编成的字符串
encode_str()	文本和结点列表	被编码的字符串
decode_str()	被编码字符串结点列表	解码后的字符串
get_H()	字符和相应的概率	信源的熵
get_average_code_length()	结点列表	平均码长

表 2 fano.py 模块的划分

3. run_lenggth2.py 模块划分

函数名称	函数输入	函数输出
get_text_count()	被编码的字符串	字符和相应概率
get_max_bin_length()	被编码的字符串	游程码的最大二进制长度
encode_str()	被编码的字符串和游程列表	被编码的字符串
decode_str()	被编码字符串和游程列表	解码后的字符串
get_H()	字符和相应的概率	信源的熵

表 3 run_lenggth2.py 模块划分

4. signal.py 模块的划分

函数名称	函数输入	函数输出

get_text_count()	被编码的字符串	字符及相应概率和信源的熵和平均码长
encode_str()	被编码的字符串和字符相应的概率	被编成的十进制小数
ten2bin()	十进制小数	二进制小数
decode_str()	十进制小数和字符的相应概率	解码后的字符串

表 4 signal.py 模块的划分

5. bmp_huffman.py 模块的划分

函数名称	函数输入	函数输出
Class node	Huffman 结点参数	一个结点
get_image_list()	图像的路径（图像）	像素值列表，游程列表，最大二进制长度
get_image_list_count()	像素值列表	一个字符，频率字典和相应的字符概率
create_nodes()	字符串的字符	结点列表
create_huffman_tree()	结点列表	Huffman 树和根结点
huffman_encoding()	字符频率和 Huffman 树	每个结点被编成的字符串和平均码长
encode_str()	像素值列表，结点列表，游程表	被编码的字符串
decode_str()	被编码的字符串和结点列表	解码后的字符串
get_H()	字符和相应的概率	信源的熵
get_average_length()	游程列表和结点列表	平均码长
compare()	图像路径，解码字符串和像素值列表	是否解码译码一致

表 5 bmp_huffman.py 模块的划分

五、设计步骤

因任务 5 用 Huffman+游程编码 bmp 灰度图像是任务 1huffman 编码和任务 3 游程编码的综合，其核心思想是一样的，所以将其合并到一起。

1. Fano 编码

首先在主界面输入任意的字符串，然后统计字符串中的字符出现频率，返回一个（字符名，频率）的字典,（计算频率时，可用字符串的方法 count）。根据将字典按照其中元素频率的大小，从小到大排序（排序时，可用队列的方法 sort(),然后利用关键词 key 达到排序的效果），进行 Fano 编码，得到字符对应的编码。然后按照 Fano 编码对程序开始输入的字符串从头开始编码，得到编码结果。译码也是同样的道理，只是编码的逆过程，注意每次都是从字符串的头开始，此处可用字符串的函数，index()来判断它的位置。最终将译码结果和初始字符串比对，查看是否正确，正确则计算编码效率。程序主要流程图如下：

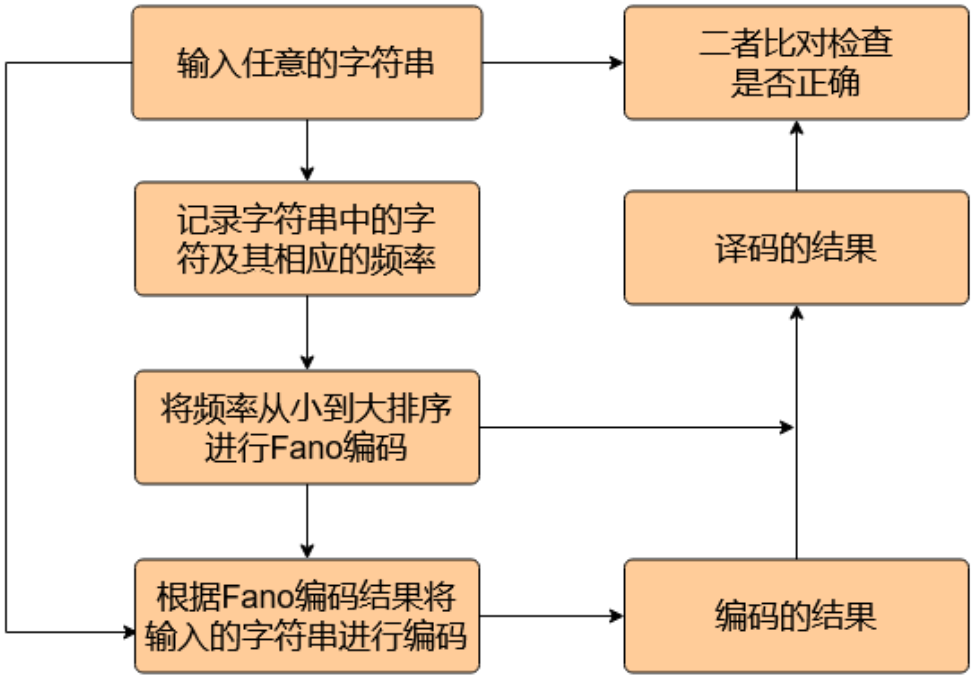


图 5 Fano 编码总流程图

Fano 编码的主要难点是 Fano 树的构建，它和 Huffman 树略有不同。在构建树时，我采用如下思想，将信源符号按出现的频率由小到大排列，使其分为两个小组，两个小组频率之差的绝对值最小，然后对其中一个分组置 1，另一个分组置 0。进一步使用递归的思想，将每个小的分组，分成更小的两个分组。再进行如上操作，直到小分组只有一个元素时，跳出递归，此时便得到了每个元素对应的 Fano 编码。构建 Fano 树的流程图如下：

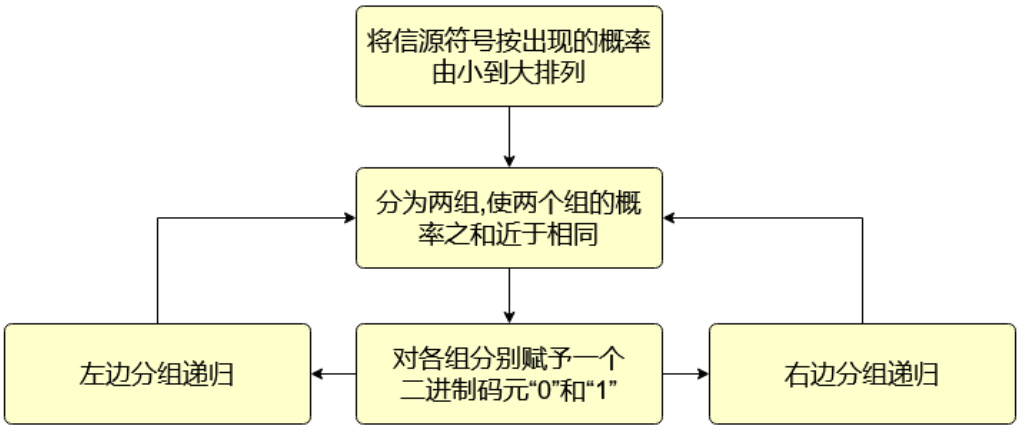


图 6 构建 Fano 树

首先将信源符号按出现的概率由小到大排序，排序可用列表内置函数 sort，在找中间位置时，我采用的思想是将当前列表分为两组，两组差值绝对值最小的位置即为分组的位置，此时要记录分组的位置和上一次分组的位置，否则在递归赋值时会引起混乱。实现的代码如下：

```
def fano_encode(nodes,pi_list,positon):
    if len(pi_list) <= 1:
        return 0
    # 最佳分组位置
    mini_difference = 1
    find_position = 1
    for i in range(len(pi_list)):
        sum1 = 0
        sum2 = 0
        for i in range(i+1):
            sum1 += pi_list[i]
        for j in range(i+1,len(pi_list)):
            sum2 += pi_list[j]
        difference = abs(sum1 - sum2)
        if difference < mini_difference:
            mini_difference = difference
            find_position = i+1
            # print(find_position)
    print(find_position)
    # 编码
    for i in range(len(pi_list)):
        if nodes[i+positon].flag==0: #可写入
            if i < find_position:
                nodes[i+positon].sign_str = nodes[i+positon].sign_str+ '0'
                print(i+positon,nodes[i+positon].sign_str)
            else:
                nodes[i+positon].sign_str = nodes[i+positon].sign_str+ '1'
                print(i+positon,nodes[i+positon].sign_str)
            if len(pi_list)==2:  #最终值
                nodes[i+positon].flag=1
                print(i+positon,'之1')
    # 编码分组
    leftgroup = []
    rightgroup = []
    for i in range(find_position):
        leftgroup.append(pi_list[i])
    for i in range(find_position, len(pi_list)):
        rightgroup.append((pi_list[i]))

    # 递归编码
    fano_encode(nodes,leftgroup,0+positon)
    fano_encode(nodes,rightgroup,find_position+positon)
```

图 7 Fano 编码的代码实现

2. 算数编码

首先在主界面输入任意的字符串，然后统计字符串中的字符出现频率，返回一个（字符名，频率）的字典,（计算频率时，可用字符串的方法 count）。根据该字典,计算得到它在 0-1 概率区间上的对应区间,根据概率区间进行算数编码，此处将字符串的每个字符进行遍历，每次调用一个字符时，根据它的概率区间来更新总的概率区间。字符串遍历结束，也就得到了最终的概率区间，此处取左端点作为最终值，将其转为二进制 01 比特流，则可得到编码结果。译码也是同样的道理，只是编码的逆过程，注意每次都是判断某个字符是否在该概率区间，是的话就输出，并更新概率区间，最终得到译码结果。将译码结果和初始字符串比对，查看是否正确，正确则计算编码效率。程序主要流程图如下：

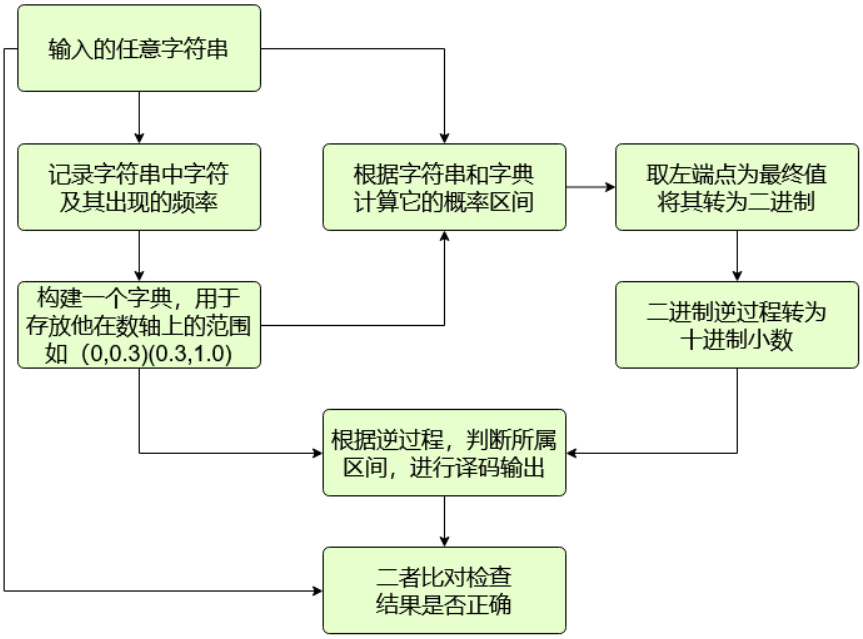


图 8 算数编码的主要流程图

算数编码的主要难点是编码时概率区间的选择和译码时概率区间的反推。在编码时，我采用如下思想：首先记录每个字符在 0-1 这个概率区间上的具体子区间，然后根据每一个字符串中的字符，根据其概率区间更新当前的子区间，更新方法采用如下思想： 当前区间的右端点=当前区间的左端点+字符对应的区间长度*字符对应的右端点，当前区间的左端点=当前区间的左端点+字符对应的区间长度*字符对应的左端点，当字符串的所有字符被顺序遍历完成之后，即可得到十进制小数的编码结果，将其转为二进制即可得到最终的编码结果。二进制的位数取决于输入符号的频率，计算见公式（5）（6），算数编码的核心流程图如下：

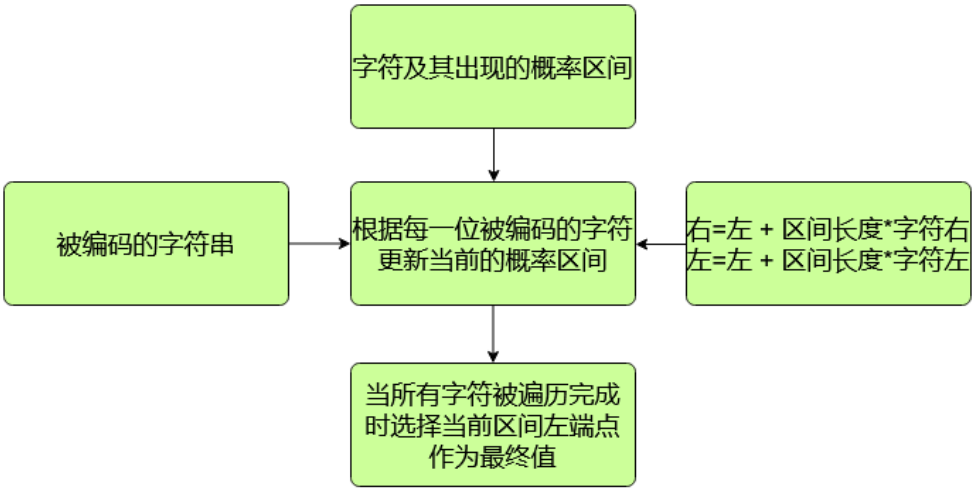


图 9 对字符串算数编码的核心思想

```
# 编码，返回字符串(左端点)
def encoder(text, char_dict):
    # 根据字符和字典进行编码，返回区间左端点的值
    left = 0
    right = 1
    for s in text:
        span_length = right - left
        right = left + span_length * char_dict[s][1]  # 右端点+区间长度*
        left = left + span_length * char_dict[s][0]  #这个放下面，否则会left，right不是同时变
        #print(left)
    ##print('left',left)
    # print(type(left)) # float
    return left
```

图 10 对字符串算数编码的核心思想的代码实现

3. 灰度图像 Huffman+游程编码

程序首先选择图片的绝对路径，然后读取图片的像素值，存放到一维数组 image_list 中（例如[127,123,125]），并读取游程列表，存放到二维数组 run_list 中（例如[[127,3]],代表 127 这个像素值连续出现 4 次（此处存在偏移，因为不会有 0 游程，所以 0 代表 1，以此类推））。**此处可以优化**，因为后续游程码会用等长码表示,所以要计算等长码的长度,读取游程列表中的最大长度，用 log 向上取整，注意当最大长度为 2 的幂次方时，取整后+1(因为 0 代表 1，会偏移)。然后记录 image_list 中的像素值出现的频率，构造 Huffman 树，对每个像素值进行编码。最终，根据游程二维列表进行图像的编码，像素值用 Huffman 编码，游程用等长码表示，得到最终的编码结果。译码时为编码的逆过程，此处不赘述。最后，将译码结果和编码结果进行比对，观测是否两次结果一致，若一致，利用译码结果生成一幅图像，保存到原路径下，命名为 after_lennag.bmp，并在主界面展示。

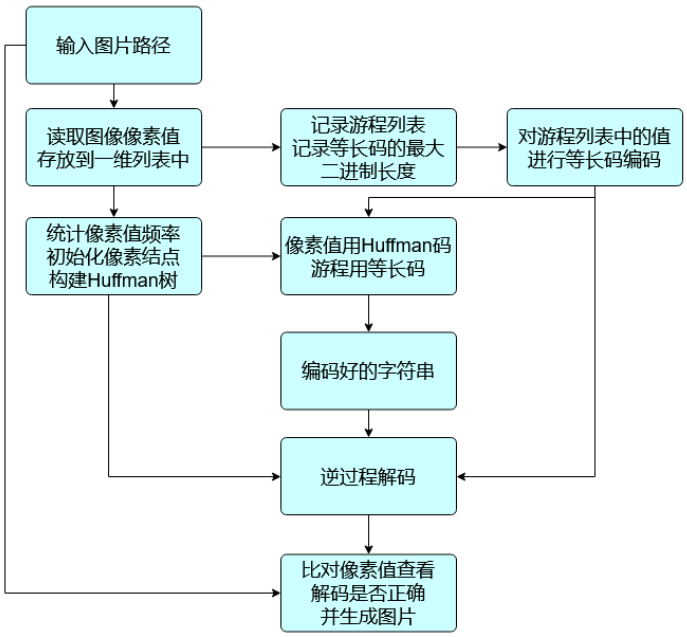


图 11 1. 灰度图像 Huffman+游程编码总流程图

在进行 Huffman 树构建时，我采用如下思想：将每个像素值建立结点，并存放到一个队列中。每个符号的权值设为其频率，每次出队列两个权值最小的结点，作为左子树和右子树，生成的新结点的权值为两个子树的权值之后，然后入队了。判断是否只剩一个结点，是则退出，否则继续循环。构建 Huffman 树的流程如下：

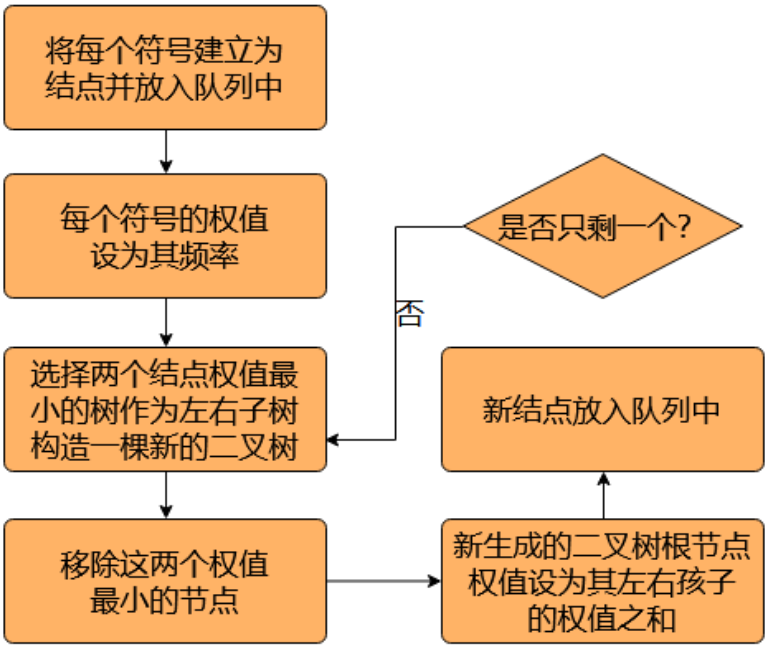


图 12 Huffman 树的构建

```python
# 创建Huffman树,返回根节点
def create_huffman_tree(nodes):
    '''创建Huffman树,返回根节点'''
    queue = nodes[:]

    while len(queue) > 1:
        queue.sort(key=lambda item: item.pi) #队列排序，由小到大
        node_left = queue.pop(0)
        node_right = queue.pop(0)
        node_father = Node('','',node_left.pi + node_right.pi)
        node_father.left = node_left
        node_father.right = node_right
        node_left.father = node_father
        node_right.father = node_father
        queue.append(node_father)

    queue[0].father = None
    print('成功构建Huffman树')
    return queue[0]
```

图 13 Huffman 树的代码实现

六、关键问题及其解决方法

1. Fano 编码时，递归会出现各种错误，例如分割位置正确，但赋值错误，或分割位置不正确，只是盲目的使用 1/2。

解决方法：在递归时，不盲目使用 1/2，而是使用在当前的数组内，分成两个数组，从小到大排序，穷举可能出现的位置，记录在对应位置时，两个小数组的差值绝对值是否最小，最小则为正确的分组位置。其次，在递归时，要正确传入参数，否则会导致赋值错误，在本次课程设计中，我传入的参数增加了当前的位置和找到的位置，这样在赋值时就不会发生紊乱。实现的核心代码如下：

```python
def fano_encode(nodes,pi_list,positon):
    if len(pi_list) <= 1:
        return 0
    # 最佳分组位置
    mini_difference = 1
    find_position = 1
    for i in range(len(pi_list)):
        sum1 = 0
        sum2 = 0
        for i in range(i+1):
            sum1 += pi_list[i]
        for j in range(i+1,len(pi_list)):
            sum2 += pi_list[j]
        difference = abs(sum1 - sum2)
        if difference < mini_difference:
            mini_difference = difference
            find_position = i+1
            # print(find_position)
    print(find_position)
    # 编码
    for i in range(len(pi_list)):
        if nodes[i+positon].flag==0: #可写入
            if i < find_position:
                nodes[i+positon].sign_str = nodes[i+positon].sign_str+ '0'
                print(i+positon,nodes[i+positon].sign_str)
            else:
                nodes[i+positon].sign_str = nodes[i+positon].sign_str+ '1'
                print(i+positon,nodes[i+positon].sign_str)
            if len(pi_list)==2:  #最终值
                nodes[i+positon].flag=1
                print(i+positon,'之1')
    # 编码分组
    leftgroup = []
    rightgroup = []
    for i in range(find_position):
        leftgroup.append(pi_list[i])
    for i in range(find_position, len(pi_list)):
        rightgroup.append((pi_list[i]))

    # 递归编码
    fano_encode(nodes,leftgroup,0+positon)
    fano_encode(nodes,rightgroup,find_position+positon)
```

图 14 Fano 编码的代码实现

2.构建 Huffman 树时，新的结点为合并两个权值较小的结点，如何区分新产生的结点和正常字符对应的结点？

解决方法：在构建之前，先将正常字符对应的结点保存到一个结点列表中，后续构建 Huffman 树时虽然会产生新的结点，但是不会影响正常字符对应的结点，并且在编码时，可以更高效的编码。（不用编码新产生的结点）。

3.如何对 Huffman 树译码？

解决方法：此处，我在构建 Huffman 树时，选择将新生成的结点和两个子结点进行链接到一起。在译码时，对结点列表中的结点进行译码。宏观上来说，是从 Huffman 树的底层叶子结点向上开始译码（此处也可以从根节点开始译码），每次判断他是否是父结点的左节点，如是则在它编码的字符串前面+ ‘0’，如不是则判断是否是父节点的右节点，如是则在它编码的字符串前面+ ‘1’，如果到了根节点，则代表编码结束，需退出。实现的代码如下：

```
# 根据Huffman树，从下向上编码，返回哈夫曼的平均码长
def huffman_encoding(nodes, root,):
    average_code_length=0
    for node in nodes:
        temp_node = node      #递归的，改变的temp_node
        while temp_node != root:
            if temp_node.is_left(): # 判断是不是父节点的左节点
                node.sign_str = '0' + node.sign_str
            else:
                node.sign_str = '1' + node.sign_str
            temp_node = temp_node.father

    print('成功构建Huffman编码')
```

图 15 Huffman 编码的代码实现

4.算数编码时，如何进行编码和译码操作？

解决方法：算在编码时，我采用如下思想：首先记录每个字符在 0-1 这个概率区间上的具体子区间，然后根据每一个字符串中的字符，根据其概率区间更新当前的子区间，更新方法采用如下思想： 当前区间的右端点=当前区间的左端点+字符对应的区间长度*字符对应的右端点，当前区间的左端点=当前区间的左端点+字符对应的区间长度*字符对应的左端点，当字符串的所有字符被顺序遍历完成之后，即可得到十进制小数的编码结果，将其转为二进制即可得到最终的编码结果。二进制的位数取决于输入符号的频率，计算见公式（5）（6）。译码为编码的逆过程，注意每次都是判断某个字符是否在该概率区间，是的话就输出，并更新概率区间，最终得到译码结果。算数编码的核心代码如下：

```python
# 编码，返回字符串(左端点)
def encoder(text, char_dict):
    # 根据字符和字典进行编码，返回区间左端点的值
    left = 0
    right = 1
    for s in text:
        span_length = right - left
        right = left + span_length * char_dict[s][1]  # 右端点+区间长度*
        left = left + span_length * char_dict[s][0]  #这个放下面，否则会left，right不是同时变
        #print(left)
    ##print('left',left)
    # print(type(left)) # float
    return left

# 解码
def decoder(singal_str, char_dict, len_text):
    text = []
    #print(char_dict)
    while len_text:
        for k, v in char_dict.items():
            if v[0] <= singal_str < v[1]:      # 在对应区间内
                print(v[0],v[1])
                text.append(k)
                range = v[1] - v[0]         #在对应区间求子区间
                singal_str -= v[0]          #
                singal_str /= range         #
                break
        len_text -= 1
    ret = ''
    for item in text:
        ret+=item
    return ret
```

图 16 算数编码和译码的核心代码实现

5．图像的游程编码时，如何确定等长码的长度。

首先读取图片的宽和高，若是横着读取图片，则其最大的二进制长度为宽度值取 log。此处可以**进一步优化**，根据宽度来决定最大二进制长度，会造成大量的浪费，增加平均码长，所以，此时可以采用自适应码长。首先读取图片中的游程列表，游程列表的最大值，代表了此图像最多有多少个像素值是连续的，由其来 log 向上取整，得到的二进制长度才不会被浪费。此处还可**进一步优化**，因为没有 0 游程，所以为了利用 0 这个编码，要产生偏移，例如：0 代表 1 以此类推。此时用 log 向上取整时，注意游程列表最大长度为 2 的幂次方时，取整后+1(因为 0 代表 1，会偏移，不进行+1 会造成溢出)。

6．图像像素值如何读取？

解决方法：可以用 python 库 PIL 中的 Image.open().getdata(),返回一个对象，用 list（）方法将其转为一维数组即可。注意在读取图像时，有的图像时 8 位位深度，有的图像是 24 位位深度，此处可用一个 try,except 语句即可。

```python
try:
    image_list = list(x[0] for x in im.getdata()) #横着读
except:
    image_list = list(im.getdata()) #横着读
```

七、设计结果

以测试数据‘2020210593aaaaaabbbbdaeqtjklczxjioxzxjl’为例来测试四种编码的编码效率以及译码结果。运行 main.py，得到编码效率从大到小依次是，算数编码：99.65%，Huffman 编码：99.02%，Fano 编码：98.39%，游程编码：76.92%。

具体程序运行结果如下图 17,18,19,20：

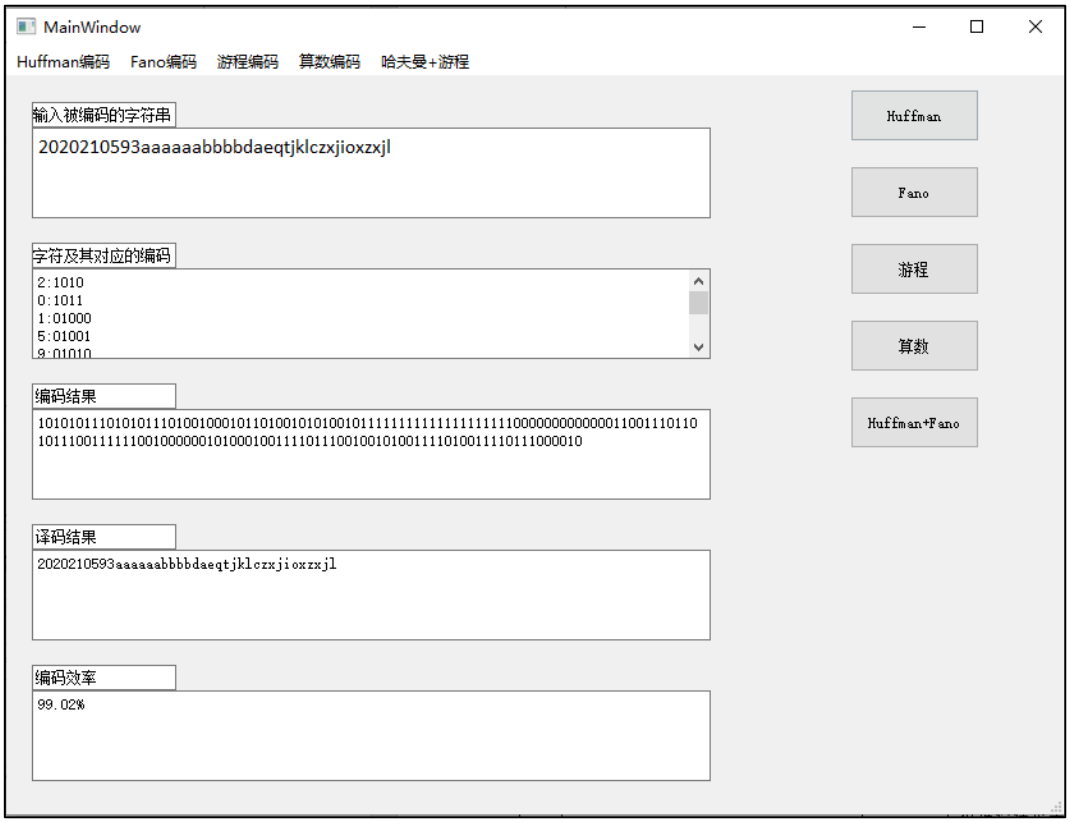


图 17 Huffman 编码测试

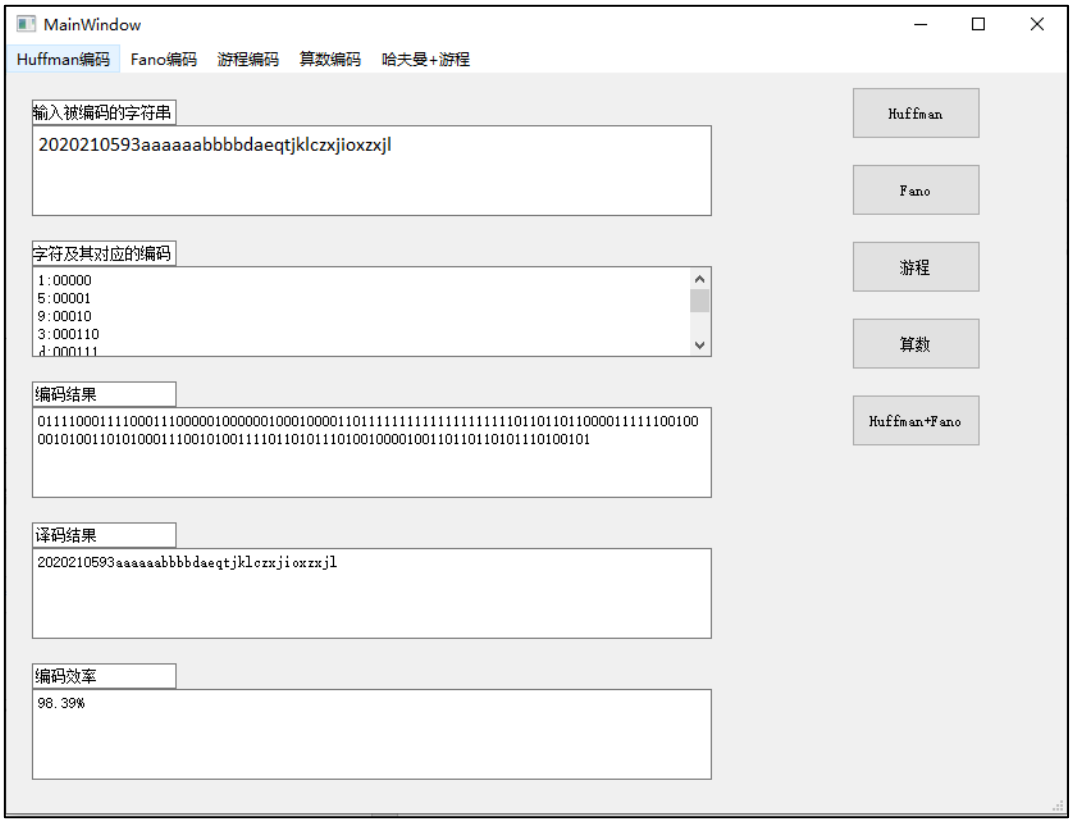


图 18 Fano 编码测试

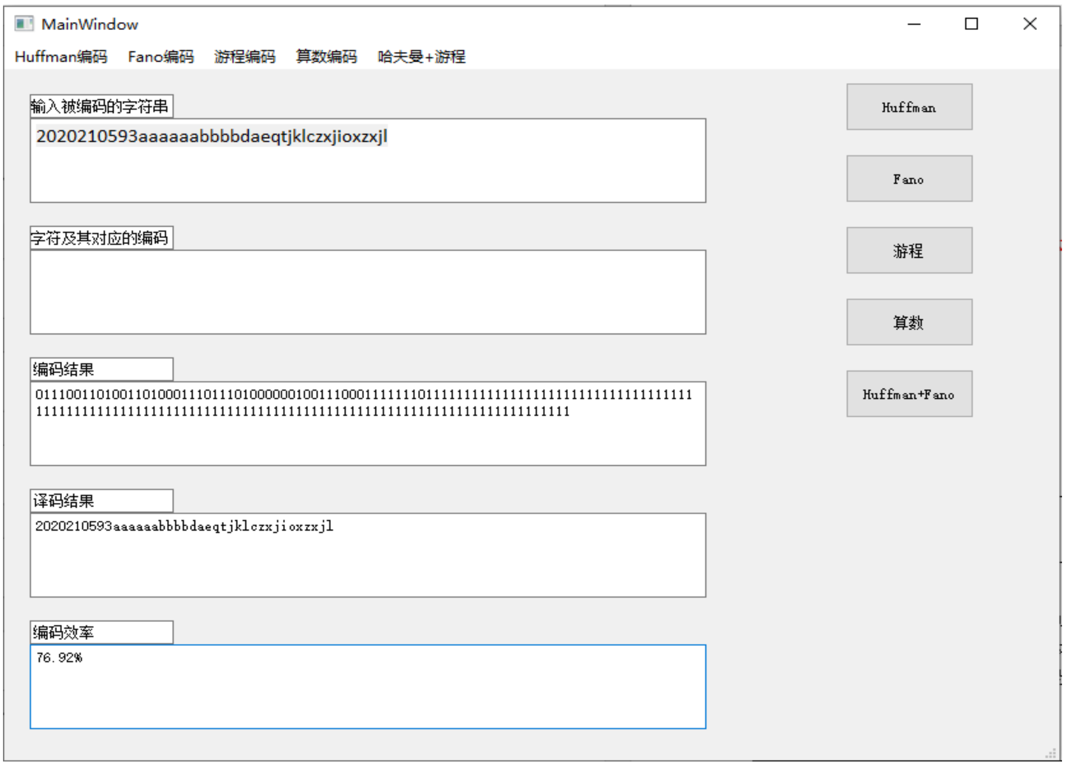


图 19 游程编码测试

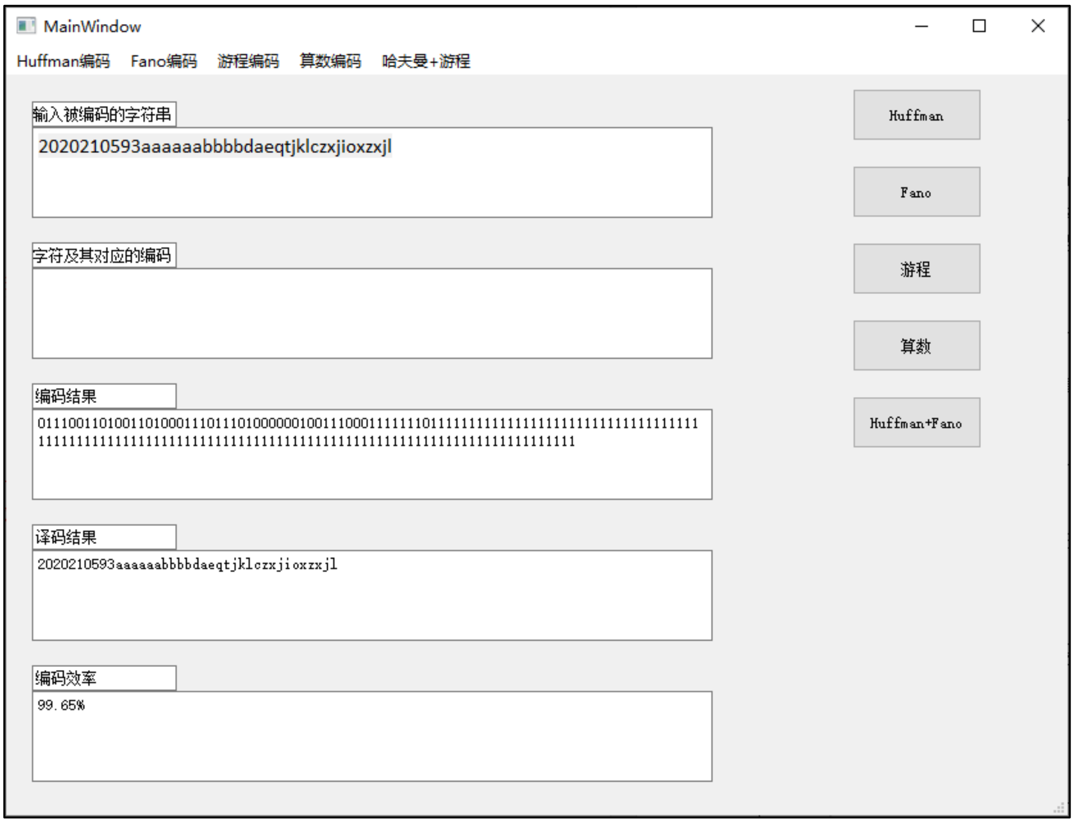


图 20 算数编码测试

提升题以 256*256 的 bmp 灰度图像为例，运行结果如下图，被编码的字符串和译码结果是一样的。

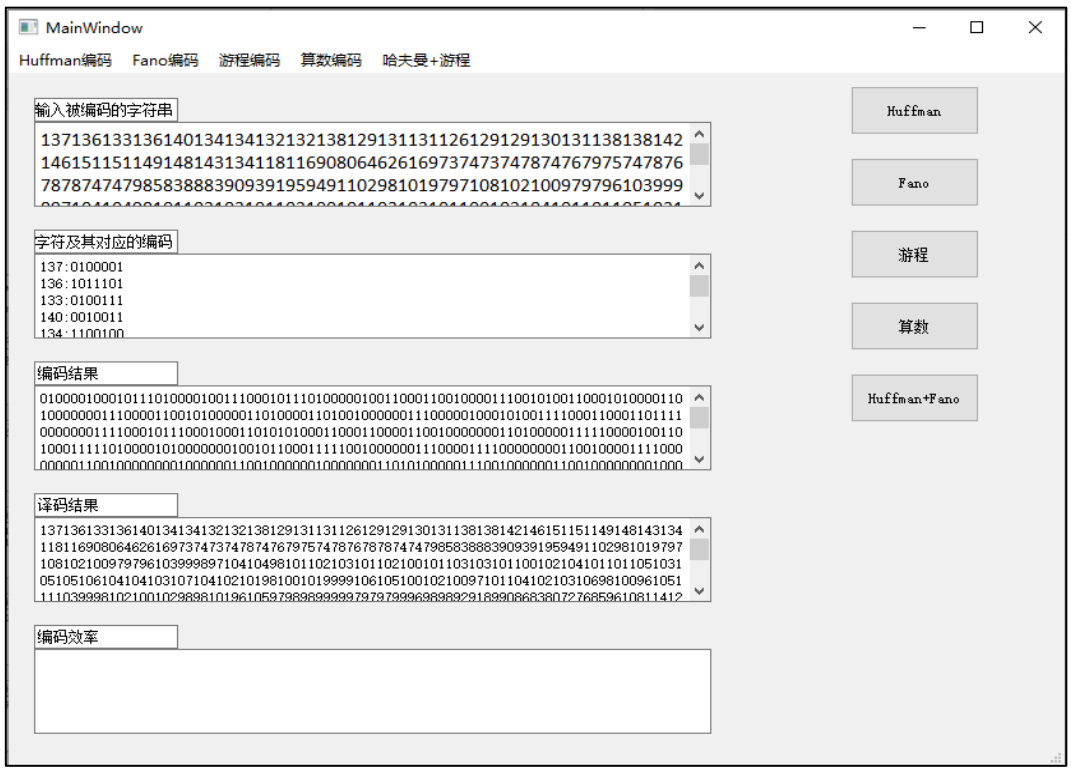


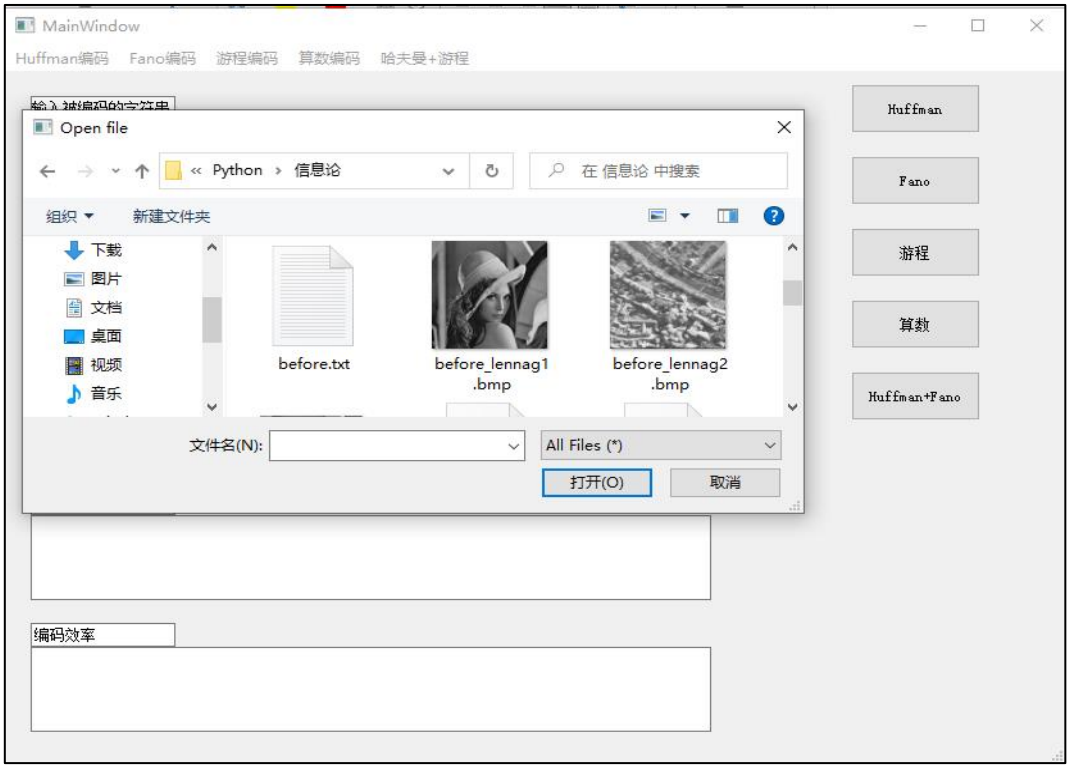
图 21 灰度图像测试

图 22 测试使用图像

八、软件使用说明

主界面共有 5 个按钮，分别实现五个题目对应的功能，在文本框中输入被编码的字符串，点击不同的按钮，即可得到字符及其对应的编码，编码结果，译码结果和编码效率。其中，第五个按钮按下，会提醒您选择文件，之后会进行相应

的编码和译码操作。如下图所示



九、参考资料

[1] 傅祖芸，信息论基础理论与应用（第 4 版），电子工业出版社，2015

十、验收时间及验收情况

验收时间为：2022 年 7 月 1 日

验收情况为：验收通过。

十一、设计体会

通过本次课程设计，我收获颇丰，使我更加扎实的掌握了有关信息论与编码的知识，通过编写 Huffman，Fano，游程，算数这四种无失真信源编码方法，能更深刻体会他们的应用场景以及优缺点。在设计过程中虽然遇到了大量的问题，但经过一次一次的思考，一遍一遍的讨论，一次一次的检查，最终找到了原因所在，也暴露出了前期我在这方面的知识欠缺和经验不足，实践出真知，通过本次亲手设计，才发现了成功的不易，必须要不厌其烦的发现问题所在，然后一一进行解决，只有这样，才能成功的做成想做的事，才能在今后的道路上披荆斩棘，收获喜悦，否则永远不可能得到社会及他人对你的认可！

十二、考核及成绩

《信息论与编码课程设计》评分表

序号	课程目标	指标点	评价观察点	评价方式		得分（百分制）
				依据/评价人	权重	
1	CO1 掌握信息安全相关的信息编码基本知识、信息编码的具体方法，独立设计简单的信息编码系统。	GR3	根据设计报告情况和验收情况来综合评判： 1）调试基本流程是否熟练； 2）任务完成的技术路线是否合理有效； 3）是否增加新的设计内容和功能； 4）演示效果是否达标； 5）能否正确回答老师提问。	实验验收 / 教师	0.2	
2	CO2 能够撰写格式规范、逻辑清晰、结构合理的设计报告。	GR5	根据设计报告情况来评判： 1）是否完成任务要求； 2）设计报告撰写是否规范合理。	设计报告 / 教师	0.4	
3	CO3：通过项目锻炼，培养自主学习、适应发展的能力。	GR12	根据验收和设计报告情况评判： 1）代码编写是否规范高效； 2）能够流畅表达技术思路、分析可能存在的问题以及解决思路。 3）设计报告是否正确反应了其阐述的设计技术思路	实验验收 / 教师	0.3	
				设计报告 / 教师	0.1	

评分方法：评价人根据观察点要求对各项课程目标的完成情况进行评估并给出得分（百分制）。

评价标准：1）观察点任务完成、能力达成，且表现突出：评为优秀，得分范围为 85~100；
2）观察点任务完成、能力达成，表现良好：评为良，得分范围为 75~84.9；
3）观察点任务完成、能力达成一般，表现较好：评为中，得分范围为 66~74.9；
4）观察点任务基本完成、能力基本达成，表现一般：评为及格，得分范围为 60~65.9；
5）观察点任务未完成、能力未达成，表现差：评为不及格，得分范围为 0-59；

总评成绩：_____ 主讲教师：苏兆品 张国富 日期：2022.7.5