信息论与编码——信源编码实验报告

【实验环境】

Python3.9.10 64bit.

【实验内容】

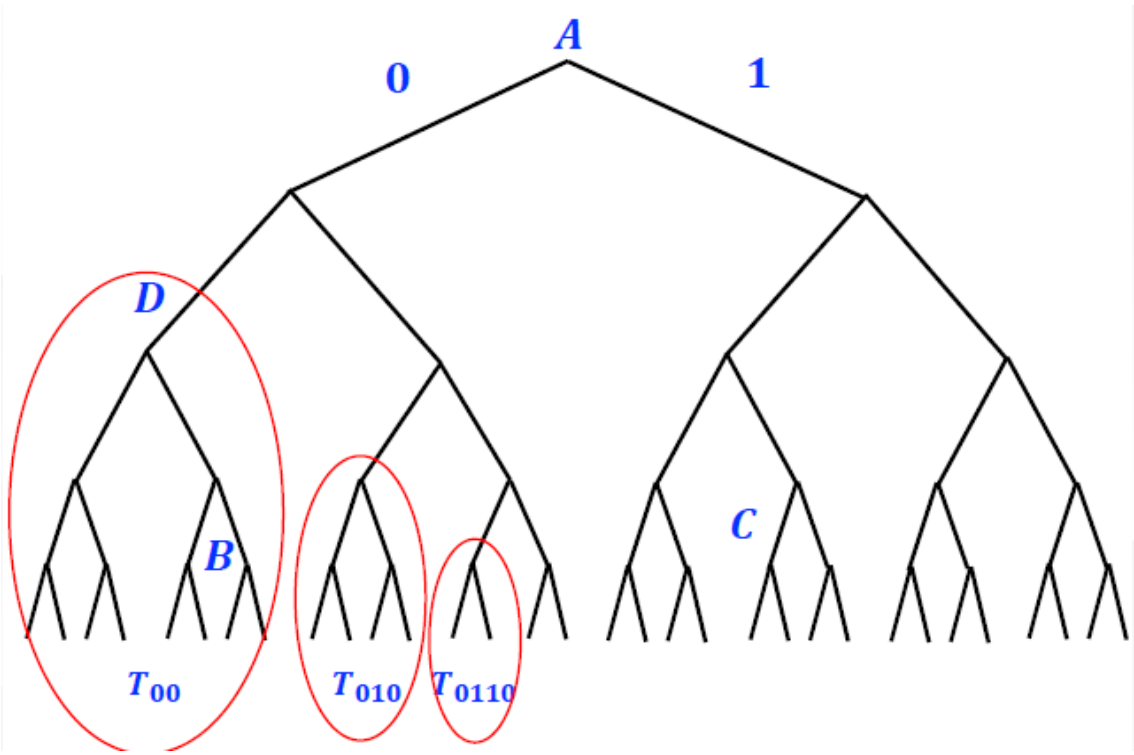
一、 必选模组3——算术编码

1. 编码原理

算术编码是对Shannon-Fano-Elias编码方案的推广。也是基于信源概率分布$p(x^n)$和累计分布函数$F(x^n)$的编码。

采用码树理解，将算术编码的信源码放在一个无限深度的满二叉码树上，信源序列总是可以对应于码树上的某一个节点，每个节点根据单符号概率$p(x)$可以得到序列概率$p(x^n)$,然后前缀和可以得到对应的$F(x^n)$。

计算$l = \lceil \log(\frac{1}{p(x^n)}) \rceil$，得到码长。对$F(x^n)$取上取整的$l$位，即为对应序列的编码。



算术编码码树图

2. 编码方法

首先将文件中的01概率分布求出，得到p(x)。

设立区间，设上界L（初始为0）、下界R（初始为1）。

然后读取文件，按每个比特模拟区间划分的过程，对于每个比特，设分界值为M = L + p * (R - L)。

如果当前位为0，那么进入左区间R = M。如果当前为1，进入右区间L = M。

不断划分之后，在[L,R]中选取一个数输出即可

细节：

a. 由于区间不断细分，小数精度有要求。需要写高精度小数。我这里是限制了小数位数为Decimal_Number，然后存储整数m，表示m右移Decimal_Number位的一个小数。

b. 在L，R变化过程中，如果L，R前几个比特位相等，就直接可以输出到输出流里。

c. 为了做好算术编码，必须使用流压缩方法，因为算术编码小数位数过多会效率下降，过少会出错，比如：有时候丢精度会导致L=R的情况。所以采取每次读一小块文件，进行编码，不断调整每块读取长度block_size和高精度小数位数Decimal_Number两个参数，取到一个合适值。我这里block_size为256字节，Decimal_Number为512位。

d.需要添加头部信息：编码后块长度、01分布（0的概率）。

尾部信息：最后一字节有效位数、编码前块长度。

代码在Arithmetic_code.py中的LB_encode()函数中。

3. 解码方法

已经得到01分布、取得小数，跟编码一样，不断区间划分，还原文件。

代码在Arithmetic_code.py中的LB_decode()函数中。

二、 可选模组——流压缩

每次从文件中读取block_size个字节，进行处理后输出。

一些细节在编码方法中讲到，不多赘述。

三、 交互方式

非常容易，在终端运行“python Arithmetic_main.py”,然后按照指示输入编解码模式然后输入文件路径即可。运行一段中会显示进度条，结束后会显示运行时间和压缩率。

```
F:\data\class\code\ArithmeticCode2>python Arithmetic_main.py
--------------------------WELCOM TO LB COMPRESSION TOOL--------------------------
------If you find any bug or have suggestions, welcome to wechat me: Lb_CarryT-------
-----------------------------------NOW LET'S START-----------------------------------
| decode or encode(d/e):r
Hey!!! print true mode!!!
| decode or encode(d/e):e
| path of your file to encode:F:\data\class\code\ArithmeticCode2\picasso.bmp
processing:[**************************************************] 100.00% (8295478B / 8295478B)
time: 128.08s
completed!
your compression ratio: 72.39%

-------------------------------copy right: @BUAA_CARRYT--------------------------------
------If you find any bug or have suggestions, welcome to wechat me: Lb_CarryT-------
--------------------------------THANK YOU VERY MUCH---------------------------------
```

交互图例

四、 鲁棒性与边界处理

首先，需要判断文件路径合法，通过对错误try处理。

```
try:
    ifn = input("| path of your file to decode:")
    LB_decode(ifn)
except:
    print("\n| error!!!")
    bug = 1
if bug == 0:
    print("\ncompleted!")
break
```

当文件全1或者全0的时候，区间编码不再合适，也容易在L，R相同前缀的时候死循环，所以需要特殊处理。判断全1、全0，输出全1全0的个数就可以了。

其中0的概率我保留了p0_len位小数，为了防止由于1过少而误判，p0_len必须大于等于$\log_2(block_size * 8)$。

在L，R相同前缀的时候容易死循环，但调整小数精度之后这个发生的概率非常小，但以防万一，还是进行了判断，防止死循环出错。

```
if dl == dr:
    print('error')
    exit(0)
```

判断代码图

五、 可维护性

参数单独设置，方便调参（doge）

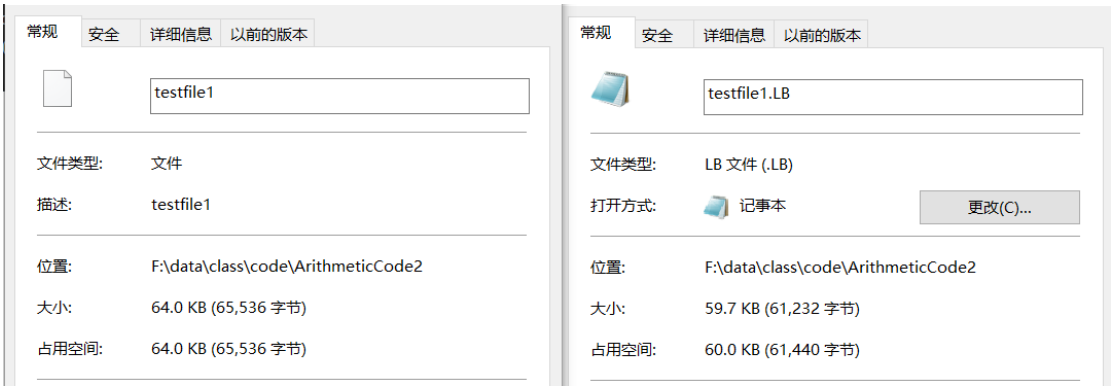
```
Decimal_Number = 512
p0_len = 16
block_size = 256
```

参数代码图

【实验问题】

一、重复性的文件结构

对test1编码后文件大小对比如下：



test1编码前后对比图

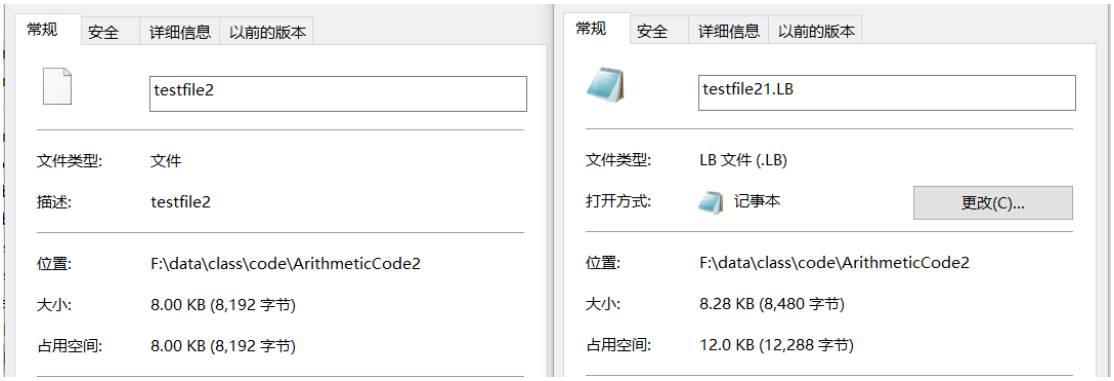
对test1还原之后SHA256值对比：

```
F:\data\class\code\ArithmeticCode2>certutil -hashfile testfile1 SHA256
SHA256 的 testfile1 哈希:
173444ecfa293433329a333289983a665c481d913e9fd1c2778b55380ca4dd31
CertUtil: -hashfile 命令成功完成。

F:\data\class\code\ArithmeticCode2>certutil -hashfile testfile11 SHA256
SHA256 的 testfile11 哈希:
173444ecfa293433329a333289983a665c481d913e9fd1c2778b55380ca4dd31
CertUtil: -hashfile 命令成功完成。
```

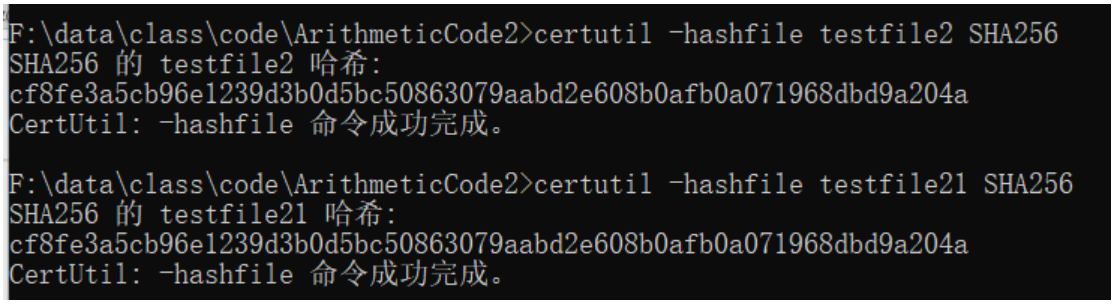
test1编解码SHA256对比图

对test2编码后文件大小对比如下：



test2编码前后对比图

对test2还原之后SHA256值对比：



Test2编解码SHA256对比图

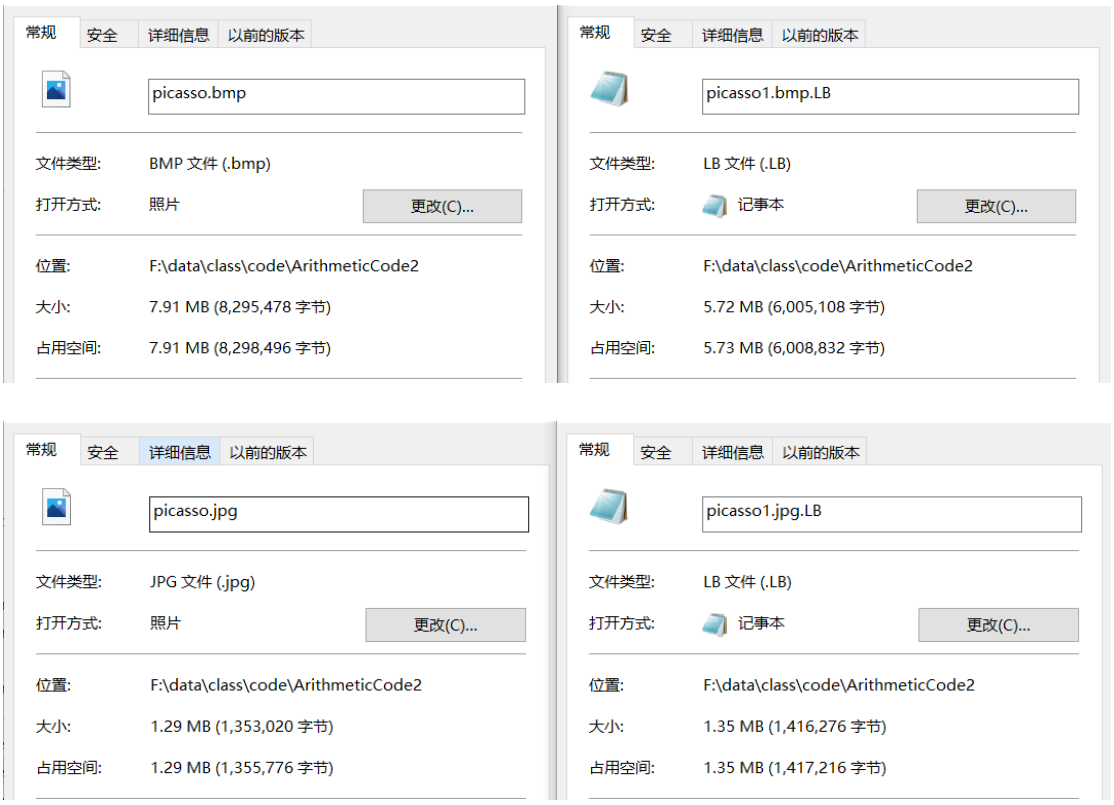
现象&原因分析：

可以看到testfile1中由于流压缩，连续一样的字节分到了一起，能进行一定的压缩。而testfile2中字节内容分布均匀，由于算术编码是利用01分布的不对称性进行的编码，因此基本没有压缩，而因为加入了首部尾部信息，导致文件变大了。

二、不同格式的压缩

1. bmp 与 jpeg

将picasso.bmp与picasso.jpeg压缩，结果如下：



压缩对比图

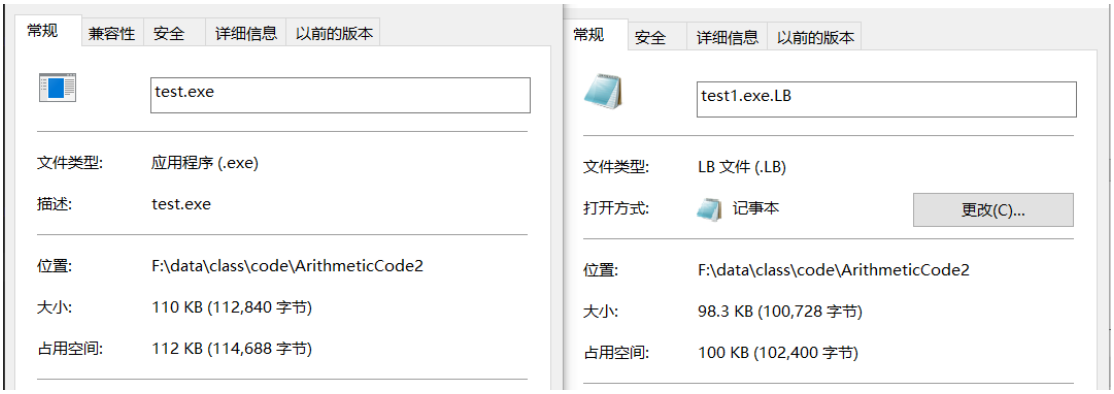
现象&原因分析：

bmp文件记录像素，具有高分辨率。jpeg已经对图像进行了有损压缩，因此图片反而增大。

2. 可执行文件

将test.exe编码，稍有压缩。

可执行文件的01分布也比较均匀。

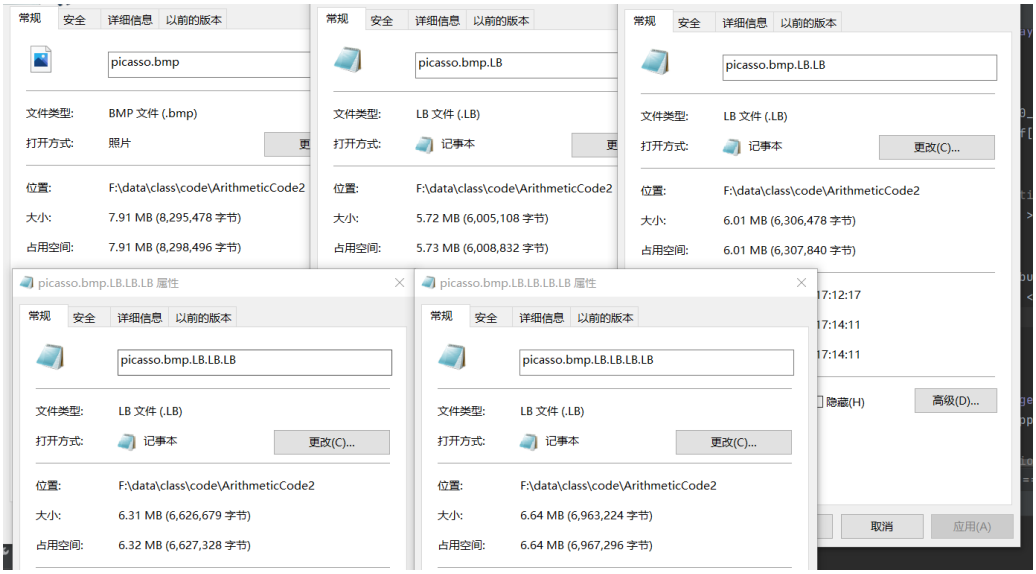


压缩对比图

三、 黑洞问题

不正确。

4次压缩体积对比如下：



因为信源编码是基于信号概率分布的无损编码压缩，是具有熵界的。压缩最多也只能达到熵界，多次迭代会趋向稳定。我这里由于头部尾部信息，越压越大了……