【注意】本文档中所有蓝色文字，是教师对内容的要求或补充说明。学生在提交报告前，应将它们全部删除。

**目录**

[1. 系统设计](#_Toc9346)

[1.1. 设计目标](#_Toc4243)

[1.2. 设计方案](#_Toc6454)

[2. 系统实现](#_Toc10810)

[2.1. 信源](#_Toc529)

[2.2. 信道](#_Toc30422)

[2.3. 信源编解码](#_Toc3293)

[2.4. 信道编解码](#_Toc29236)

[3. 系统仿真与分析](#_Toc32408)

[3.1. 理想情况下](#_Toc19489)

[3.2. 信源非理想情况下](#_Toc21400)

[3.3. 信道非理想情况下](#_Toc21610)

[3.4. 一般非理想情况下](#_Toc5634)

[4. 系统改进【可选】](#_Toc10245)

[5. 总结](#_Toc17607)

[5.1. 设计完成情况](#_Toc12429)

[5.2. 经验与收获](#_Toc14280)

[6. 附录——相关技术简介](#_Toc20583)

**要求**：定稿后更新目录，以保证目录的正确

# 系统设计

## 设计目标

使用计算机编程语言，完整实现一个包含信源、信道、信源编解码、信道编解码等模块的信息传输系统模型，并通过与理论推导的结果进行比对，验证系统实现的正确性。再进一步通过多种参数组合下的仿真，模拟不同通信条件下的信息传递，掌握信息论与编码理论的理论和实践基础。

以上“设计目标”段落文字可以直接复制使用。

## 设计方案

**要求**

* 整体系统的原理方框图、文字说明
* 每个模块的文字说明

# 系统实现

**要求**

* 每个模块是一个独立可执行的程序，模块间通过文件进行数据交互
* 数据文件的有效载荷(payload)，必须经过比特打包(bit-packing)
* 报告正文中，每个模块给出关键的代码“片段”，并提供简要的文字解说

## 信源

### 二元离散无记忆信源（DMS）模块

**要求**

* 输入参数：信源消息概率分布*P*(0)
* 输入参数：消息序列的长度（字节为单位，一般至少10K字节长）
* 模块输出：信源输出消息序列文件

### 信源指标计算模块

**要求**

* 模块输入：信源输出消息序列文件
* 模块输出：（输出形式自定）
  + 数据比特概率分布（即二元DMS的概率分布统计）：*P*(0)
  + 字节概率分布（即二元DMS的8次扩展信源的概率分布统计）：*P*(0), ..., *P*(255)
  + 二元DMS的信息熵（信息比特/二元消息）
  + 二元DSM的信源冗余度

### 单元测试

**要求（同时适用于下文相同部分，有特殊要求的再另行补充）**

* 理论推导
  + 为各输入参数假设至少1个数值，推导出相关指标的结果
  + 详细描述假设条件、推导过程、推导结果
* 测试结果
  + 按上述理论推导中的假设条件设定该模块的输入参数
  + 收集相关指标的实测结果
* 对比理论推导的结果和测试结果，分析异同及原因
* byteSource部分：

理论推导：

设信源消息概率分布，消息序列的长度为1MB，则，即二元离散无记忆信源生成0和1的概率相等。而每个字节相当于二元DMS的8次扩展，设该二元DMS的8次扩展信源为Y，则Y的概率空间为：



其中分别由8位二元离散无记忆信源符号（0和1）组成，取十进制值0~255；为对应的概率，设中二元离散无记忆信源符号0的个数为，则

故当二元离散无记忆信源生成0和1的概率相等时，其8次扩展信源的每一个值的取值概率都相等，即信源输出消息序列文件的字节概率是均匀分布的。

测试结果：

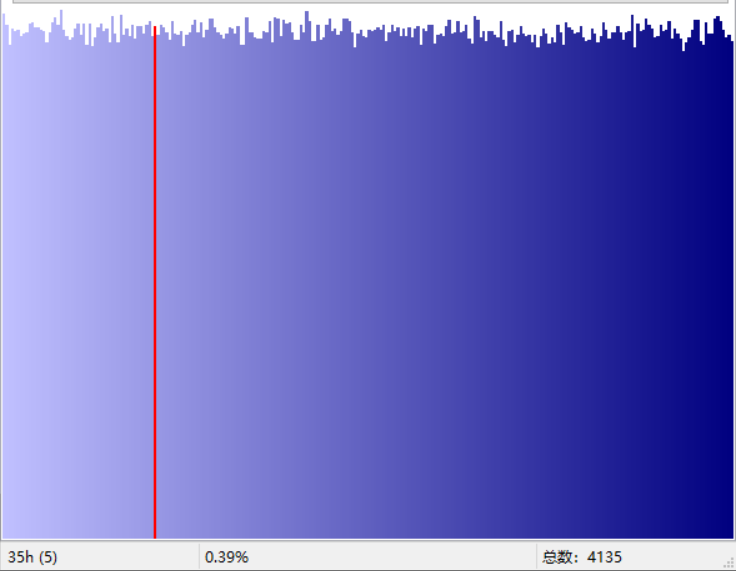
在cmd中输入命令：

python byteSource.py 0.5 1048576 unit-test\P0=0.5.LEN=1MB.OUTPUT.dat

生成的信源输出消息序列文件如下：



可以看到文件大小确实为1MB，在16进制文件编辑器HxD中打开该文件，使用统计功能显示字节概率分布如下：



可以看到字节概率分布基本均匀，均在0.39%~0.40%左右，与理论推导结果比较接近；而不能完全符合理论推导的均匀分布的原因是byteSource程序是根据蒙特卡罗法随机生成字节的，因此在一定文件大小下实际字节概率分布与理论会有微小的差距，而随着文件大小的增加，实际字节概率分布会趋于理论的均匀分布。

* byteSource\_calc部分：

理论推导：

根据byteSource部分设定的输入参数，信源指标理论计算结果应如下：

1. 数据比特概率分布（即二元DMS的概率分布统计）：
2. 字节概率分布（即二元DMS的8次扩展信源的概率分布统计）：
3. 二元DMS的信息熵：
4. 二元DSM的信源冗余度：，其中为信源的实际熵，则；为信源最大熵，此二元离散无记忆信源生成0和1的概率相等，故信息熵即为最大熵，，故

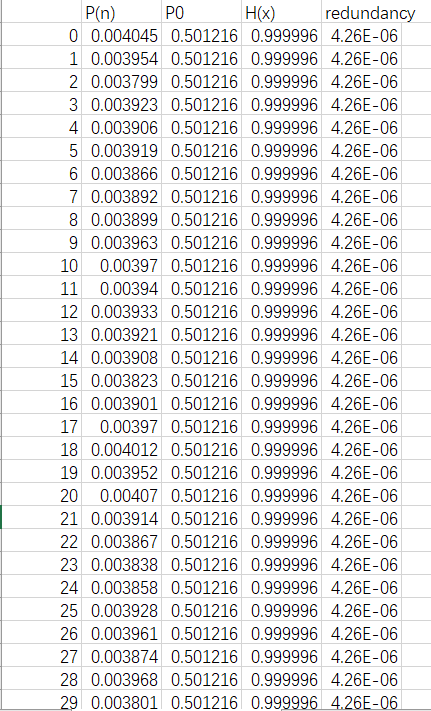
测试结果：

在cmd中输入命令：

python byteSource\_calc.py unit-test\P0=0.5.LEN=1MB.OUTPUT.dat unit-test\P0=0.5.LEN=1MB.calc.CSV

生成的信源指标计算结果如下：





其中第一列为信源输出消息序列文件字节取值，第二列为字节概率分布，第三列为数据比特概率分布，第四列为二元DMS的信息熵，第五列为二元DSM的信源冗余度。可以看到四个实际信源指标均与理论推导结果相接近，在一定误差允许范围内可以认为byteSource\_calc程序能正确计算信源指标，通过单元测试。

## 信道

### 二元对称信道（BSC）模块

**要求**

* 模块输入：信道输入消息序列文件
* 输入参数：错误传递概率*p*
* 模块输出：信道输出消息序列文件

### 信道指标计算模块

**要求**

* 模块输入：信道输入消息序列文件、信道输出消息序列文件
* 模块输出：（输出形式自定）
  + 输入消息序列的信息熵（信息比特/二元消息）
  + 输出消息序列的信息熵（信息比特/二元消息）
  + 平均互信息量（信息比特/二元消息）

### 单元测试

* byteChannel部分：

理论推导：

设错误传递概率，信道输入消息X为的概率分布，信道输出消息为Y，重用【实验3.1 二元对称信道（BSC）仿真】中制作的信道指标计算excel表格，在自变量中输入，得到理论推导结果如下红框所示：



其中输入消息序列的信息熵H(X)=1，输出消息序列的信息熵H(Y)=1，X和Y的联合熵H(XY)=2，损失熵H(X|Y)=1，噪声熵H(Y|X)=1，平均互信息量I(X;Y)=0。

测试结果：

运行byteChannel.py程序，在cmd中输入如下命令：

python byteChannel.py unit-test\P0=0.5.LEN=1MB.OUTPUT.dat 0.5 unit-test\p=0.5.P0=0.5.LEN=1MB.Channel\_OUTPUT.dat

生成的信道输出消息序列文件如下：



重用【实验3.1 二元对称信道（BSC）仿真】中教师提供的信道指标计算程序calcBSCInfo.exe，其程序API如下：

calcBSCInfo.exe X Y OUTPUT

X path to the channel input file

Y path to the channel output file

OUTPUT path to the output file to append results

在cmd中输入如下命令：

calcBSCInfo.exe ..\P0=0.5.LEN=1MB.OUTPUT.dat ..\p=0.5.P0=0.5.LEN=1MB.Channel\_OUTPUT.dat ..\Channel\_calcBSCInfo\_result.CSV

程序运行后信道指标计算结果如下：



可以看到实际测试的计算结果和理论推导的结果相接近，在一定误差允许范围内可以认为是正确的，byteChannel程序单元测试完成。

* byteChannel\_calc部分：

理论推导：

同上byteChannel部分。

测试结果：

运行byteChannel\_calc.py程序，将在cmd中输入如下命令：

python byteChannel\_calc.py unit-test\P0=0.5.LEN=1MB.OUTPUT.dat unit-test\p=0.5.P0=0.5.LEN=1MB.Channel\_OUTPUT.dat unit-test\Channel\_calc\_result.CSV

得到如下CSV文件和结果：





该计算结果与使用程序calcBSCInfo.exe计算得到的结果相同，且与理论推导的结果相接近，可以认为该结果是正确的，byteChannel\_calc单元测试完成。

## 信源编解码

### 霍夫曼编码模块

**要求**

* 模块输入：编码前的文件
* 输入参数：编码前的文件的字节概率分布
* 模块输出：编码后的文件，包含码书、码字序列

### 霍夫曼解码模块

**要求**

* 模块输入：解码前的文件（格式与霍夫曼编码模块的输出相同）
* 模块输出：解码后的文件

### 信源编码指标计算模块

**要求**

* 模块输入：编码前的文件、编码后的文件
* 模块输出：（输出形式自定）
  + 压缩比（编码前文件长度:编码后文件长度）
  + 平均码长（码字数据比特/信源字节）
  + 编码效率
  + 编码前的文件的信息熵（信息比特/字节）
  + 编码后的文件的信息熵（信息比特/字节）

### 单元测试

* byteSourceEncoder/ byteSourceDecoder部分：

理论推导：

注意测试时避免将未经验证的channelEncoder和channelDecoder程序串联起来自我验证。

测试结果：

待编码的原文件如下：



1. 编码测试：

在cmd中输入如下命令进行霍夫曼编码：

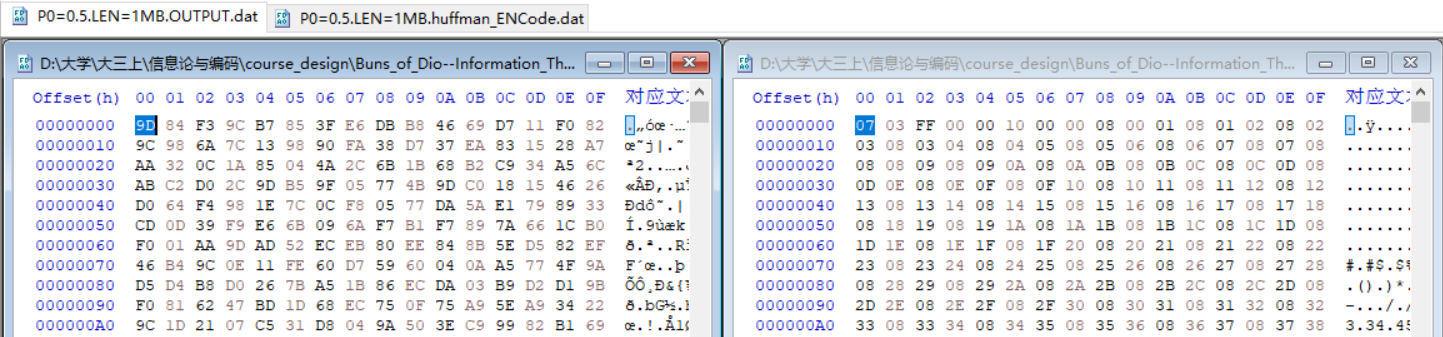
python byteSourceEncoder.py unit-test\pmf.byte.p0=0.5.csv unit-test\P0=0.5.LEN=1MB.OUTPUT.dat unit-test\P0=0.5.LEN=1MB.huffman\_ENCode.dat

程序运行完毕后得到如下文件：



可以看到文件大小与编码前的原文件大小不同，甚至比原文件要大一些，这是因为原文件是由等概率的二元DMS生成，字节概率分别是比较均匀的，而霍夫曼编码基于变长码的思想，因此对信源符号分布较均匀的序列的压缩效果微弱甚至反而会增大文件大小。

使用HxD的文件比较功能对两者的数据进行比较：



可以看到两者数据亦不相同，说明编码器是对原文件进行了处理的。

1. 解码测试：

在cmd中输入如下命令，利用解码器对编码后的文件进行解码：

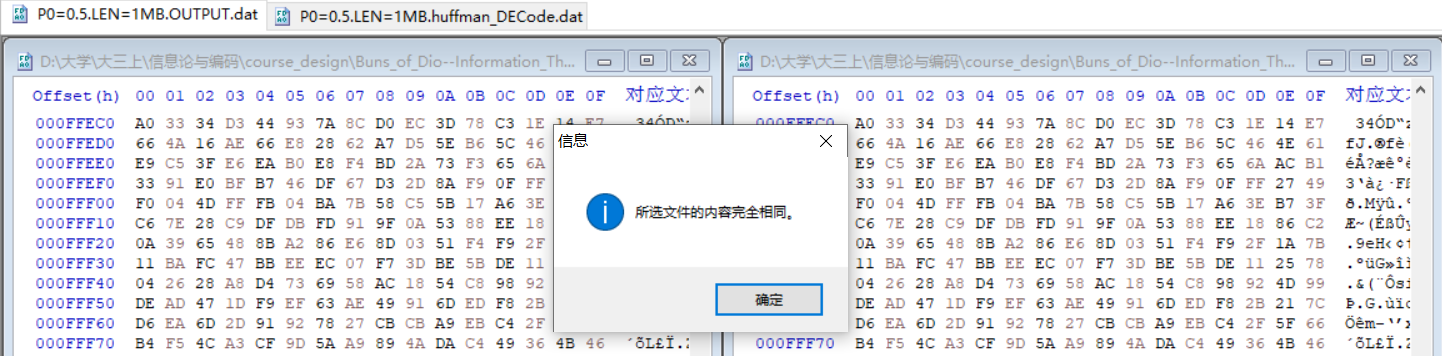
python byteSourceDecoder.py unit-test\P0=0.5.LEN=1MB.huffman\_ENCode.dat unit-test\P0=0.5.LEN=1MB.huffman\_DECode.dat

运行完毕后得到的文件如下：



可以看到文件大小与编码前的原文件大小相同。

使用HxD的文件比较功能对解码后的文件和原文件进行比较：



HxD弹窗提示所选文件的内容完全相同，说明解码后可以完整无失真地还原原文件，单元测试通过。

* codding\_effect部分：

理论推导：

1. 压缩比

编码前文件长度为l=1048576字节，编码后文件长度为L=1049351字节，故压缩比

1. 编码前的文件的信息熵

信源消息概率分布，故每二元消息的信息熵为

每字节有8数据比特，则编码前的文件的信息熵为

1. 平均码长

因为霍夫曼码是在信源给定情况下的最佳码，所以其平均码长的界限为

其中代表r元信源的信息熵，单元测试中为二元信源即r=2，且单元测试中编码前文件的每个字节相当于二元DMS的8次扩展（拓展次数N=8），同样采取霍夫曼编码方法时，因为霍夫曼码是紧致码，所以编码后单个数据比特所编得的平均码长随拓展次数N的增加很快接近于极限值——二元DMS的信息熵H，故理想条件下编码后单个数据比特的平均码长为

进而每个信源符号（8个数据比特）编码后的理论平均码长为

1. 编码效率

根据编码效率计算公式，编码效率为

1. 编码后的文件的信息熵

编码后的每二元消息的信息熵

编码后的文件的信息熵为

测试结果：

在cmd中输入如下命令，运行codding\_effect：

python codding\_effect.py unit-test\P0=0.5.LEN=1MB.OUTPUT.dat unit-test\P0=0.5.LEN=1MB.huffman\_ENCode.dat unit-test\huffman\_calresult.CSV

运行完毕后得到如下结果：





可以看到实际测试结果与理论推导结果相接近，在一定误差允许范围内可以认为是正确的，codding\_effect程序单元测试完成。

## 信道编解码

### 信道编码模块

**要求**

* 模块输入：编码前的文件
* 输入参数：编码方案选择，允许用户自由选择以下6种方案中的一种
  + N长的重复码：N是5、7、9
  + (*n*, *k*)线性分组码：(7,4)、(15,11)、(31,26)
* 模块输出：编码后的文件，包含编码方案相关参数、码字序列

### 信道解码模块

**要求**

* 模块输入：解码前的文件（格式与信道编码模块的输出相同）
* 模块输出：解码后的文件

### 信道编解码指标计算模块

**要求**

* 模块输入：编码前的文件、编码后的文件、解码后的文件
* 模块输出：（输出形式自定）
  + 误码率（汉明失真，错误二元消息数/总二元消息数）
  + 编码前的信源信息传输率（信息比特/二元消息）
  + 编码后的信源信息传输率（信息比特/二元消息）
  + 压缩比（编码前文件长度:编码后文件长度）

### 单元测试

* channelEncoder/channelDecoder部分：

理论推导：

注意测试时避免将未经验证的channelEncoder和channelDecoder程序串联起来自我验证。

测试结果：

待编码的原文件如下：



对6种方案分别进行测试，以5长的重复码为例

1. 编码测试：

在cmd中输入如下命令进行5长的重复码编码：

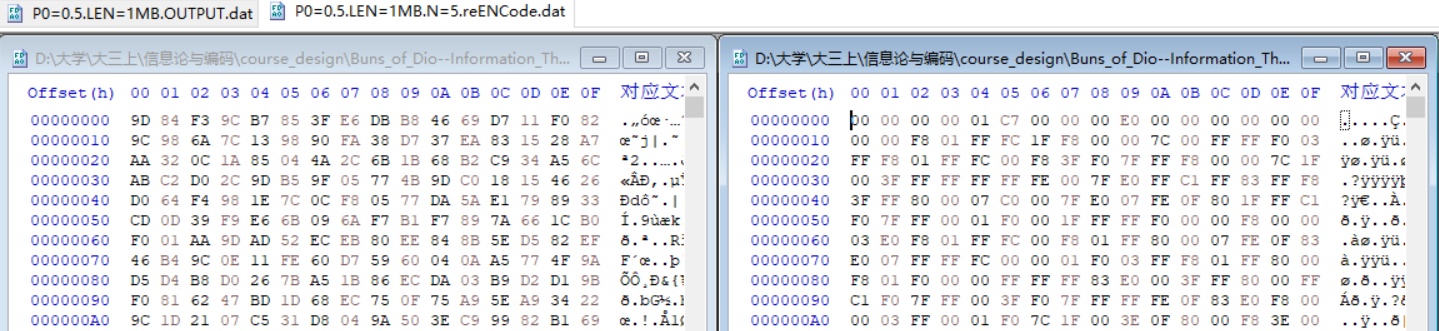
python channelEncoder.py -r unit-test\P0=0.5.LEN=1MB.OUTPUT.dat unit-test\P0=0.5.LEN=1MB.N=5.reENCode.dat 5

程序运行完毕后得到如下文件：



可以看到文件大小与编码前的原文件大小不同，N=5长的重复编码后的文件约比原文件大5倍。

使用HxD的文件比较功能对两者的数据进行比较：



可以看到两者数据亦不相同，说明编码器是对原文件进行了处理的。

1. 解码测试：

在cmd中输入如下命令，利用解码器对编码后的文件进行解码：

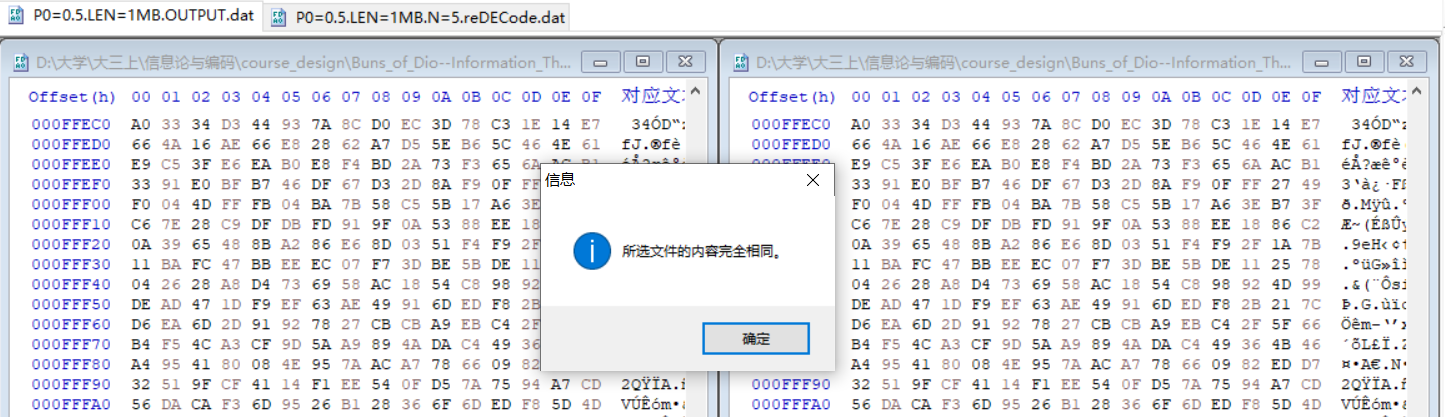
python channelDecoder.py unit-test\P0=0.5.LEN=1MB.N=5.reENCode.dat unit-test\P0=0.5.LEN=1MB.N=5.reDECode.dat

运行完毕后得到的文件如下：



可以看到文件大小与编码前的原文件大小相同。

使用HxD的文件比较功能对解码后的文件和原文件进行比较：



HxD弹窗提示所选文件的内容完全相同，说明解码后可以完整无失真地还原原文件，5长重复码单元测试通过。

7/9长的重复码和(n, k)线性分组码的测试步骤与5长重复码的步骤相同，不再赘述，经测试均通过。

* channelCoder\_calc部分：

理论推导：

1. 误码率

由于原文件编码后未经信号传输而直接解码，没有噪声干扰，因此相当于无失真编解码，误码率

1. 编码前的信源信息传输率

信源为等概率的二元DMS，消息个数，因此编码前的信源信息传输率

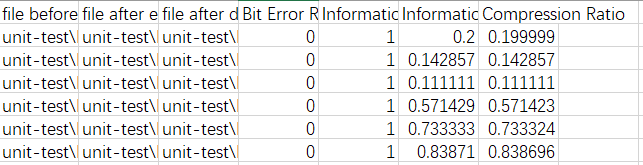
1. 编码后的信源信息传输率

设编码是将每n位二元消息编成N位码长的码字，则

1. 对于5长的重复码，n=1，N=5，故编码后的信源信息传输率
2. 对于7长的重复码，n=1，N=7，故编码后的信源信息传输率
3. 对于9长的重复码，n=1，N=9，故编码后的信源信息传输率
4. 对于(7, 4)线性分组码，n=4，N=7，故编码后的信源信息传输率
5. 对于(15, 11)线性分组码，n=11，N=15，故编码后的信源信息传输率
6. 对于(31, 26)线性分组码，n=26，N=31，故编码后的信源信息传输率
7. 压缩比
8. 对于5长的重复码，编码前文件长度为l=1048576字节，编码后文件长度为L=5242898字节，故压缩比
9. 对于7长的重复码，编码前文件长度为l=1048576字节，编码后文件长度为L=7340050字节，故压缩比
10. 对于9长的重复码，编码前文件长度为l=1048576字节，编码后文件长度为L=9437202字节，故压缩比
11. 对于(7, 4)线性分组码，编码前文件长度为l=1048576字节，编码后文件长度为L=1835026字节，故压缩比
12. 对于(15, 11)线性分组码，编码前文件长度为l=1048576字节，编码后文件长度为L=1429895字节，故压缩比
13. 对于(31, 26)线性分组码，编码前文件长度为l=1048576字节，编码后文件长度为L=1250245字节，故压缩比

测试结果：

根据不同编码类型输入相应参数，运行channelCoder\_calc，运行完毕后得到如下结果：



前三列分别为：编码前文件路径、编码后文件路径、解码后文件路径；后四列分别为：误码率、编码前的信源信息传输率、编码后的信源信息传输率、压缩比。可以看到实际测试结果与理论推导结果相接近，可以认为是正确的，channelCoder\_calc程序单元测试完成。

# 系统仿真与分析

要求

* 在本节推导以下待测指标的计算公式
  + 信源的信息率*RS*（信息比特/秒）
  + 信道的数据率*rc*（数据比特/秒）
  + 信道的输入信息率*Rci*（信息比特/秒）
  + 信道的输出信息率*Rco*（信息比特/秒）
  + 信宿关于信源的信息率*RI*（信息比特/秒）
  + 信宿的误码率*er*
* 在以下每种情况下
  + 在“条件设定”小节中，详细描述固定条件、可变条件的具体数值
    - 所有情况下的“信源的数据率”固定为*rs*=1（数据比特/秒）
  + 在“理论推导”小节中，计算所有待测指标的理论数值
  + 在“仿真”小节中，描述仿真的操作过程和待测指标的仿真结果
  + 在“分析”小节中，对比待测指标的理论数值和仿真结果，分析异同及原因

## 理想情况下

### 条件设定

**要求**

* 固定条件
  + 信源的数据率*rs*（数据比特/秒）
  + 信源为等概率二元DMS
  + 信道为无噪无损的离散无记忆信道
  + 不使用信源编码
  + 不使用信道编码
* 可变条件
  + 无

理想情况下，信源为等概率二元DMS，信源概率分布为，

则其信息熵（每数据比特）为

当其信源数据率为*v*（数据比特每秒）时，其每秒信源信息率为，单位为信息比特每秒,由于信道为无噪无损的离散无记忆信道，即噪声熵和损失熵都为0，本题定义信源数据率为1数据比特/秒。

**信源概率分布****，即等概率二元DMS，信道为无噪无损的离散无记忆信道，不使用信源编码，不使用信道编码**

由于信源概率分布，信息熵为:



定义信源数据率为1数据比特/秒，所以信源信息率为：

。

无信源编码，则信道的输入信息率等于信源信息率；

无信道编码，信道无噪无损，所以信道的输出信息率与等于信道的输入信息率，不存在误码率。理论推导结果为:

a.信源的信息率:



其中M为是总字节数，由于本题中设定信源的数据率为1数据比特/秒，传输时间为1s，故M=1信息比特。

b.信道的数据率数据比特/秒

c.信道的输入信息率=1信息比特/秒

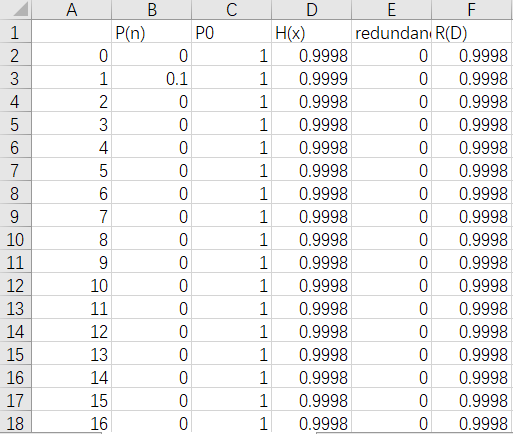
d.信道的输出信息率=1信息比特/秒

e.信宿的误码率=0

### 仿真

**信源概率分布****，即等概率二元DMS，信道为无噪无损的离散无记忆信道，不使用信源编码，不使用信道编码：**

先用程序byteSource.py生成一个概率分布为离散无记忆信源(大小为1KB)，调用程序byteSource\_calc.py可以计算其信源信息率的大小(数据率为1bit/s)(文件名为：P0=0.5.LEN=10KB.calc.CSV)：

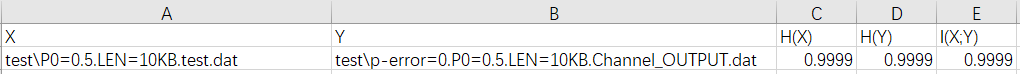


从上图可以知道计算的信源信息率为1信息比特/秒。

再通过程序byteChannel.py生成过信道的序列：



再调用byteChannel\_calc.py的函数进行对其过信道的序列进行信道的输入与的信息熵的计算，如下(文件名：P0=0.5.Channel\_not\_source\_calc\_result.CSV)：



由于取信源的数据率为1数据比特/秒，由于不使用信源编码，因此，信道的数据率为数据比特/s。再与上图中的信息熵分别相乘，即可得到信道的输入与输出的信息率，经过计算可以得到信道的输出信息率=输入信息率=1.0信息比特/秒。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 信源信息率  （信息比特/秒） | 信道数据率  （数据比特/秒） | 信道输入信息率（信息比特/秒） | 信道输出信息率（信息比特/秒） | 误码率 |
| 理论值 | 1 | 1 | 1 | 1 | 0 |
| 仿真值 | 0.9998 | 0.9998 | 0.9999 | 0.9999 | 0 |

### 分析

对比上述理论分析和仿真结果可以看到，仿真结果大致与理论推导结果一致，对于相应的仿真值与理论值相比，理论值较大，原因可能与计算时的保留小数位有关。因为理论值计算得到的信息熵为1，但是实际上由于是生成一定概率分布的信源，长度也只有10KB，故计算出来的信息熵只会趋近1，但不完全等于1，导致后面许多计算相对理论值小一点，但这结果基本在可接受的误差范围以内，故可判断该理论推导和仿真结果基本正确。

## 信源非理想情况下

### 条件设定

**要求**

* 固定条件
  + 信源的数据率*rs*（数据比特/秒）
  + 信道为无噪无损的离散无记忆信道
  + 不使用信道编码
* 可变条件
  + 2种信源概率分布：0.1, 0.3
  + 无信源编码、有信源编码

### 理论推导

假设信源概率分布为,则其信息熵（每数据比特）为。当其信源数据率为v（数据比特每秒）时，其信源信息率为，单位为信息比特每秒,由于信道为无噪无损的离散无记忆信道，即噪声熵和损失熵都为0，本题定义信源数据率为1数据比特/秒。

**(1).信源概率分布，不使用信源编码**

由于信源概率分布****，信息熵为:



定义信源数据率为1数据比特/秒，所以信源信息率为

。

无信源编码，则信道的输入信息率等于信源信息率；

无信道编码，信道无噪无损，所以信道的输出信息率与等于信道的输入信息率，不存在误码率。理论推导结果为:

a.信源的信息率:



其中M为是总字节数，由于本题中设定信源的数据率为1数据比特/秒，传输时间为1s，故M=1信息比特。

b.信道的数据率=数据比特/秒

c.信道的输入信息率=0.4690信息比特/秒

d.信道的输出信息率=0.4690信息比特/秒

e.信宿的误码率=0

**(2).信源概率分布，使用信源编码**

由于信源概率分布****，信息熵为:



定义信源数据率为1数据比特/秒，所以信源信息率为

。

无信道编码，信道无噪无损，所以信道的输出信息率与等于信道的输入信息率，不存在误码率。

有信源编码：由于我们是在每次的信源生成消息后，根据当前信源概率分布生成霍夫曼码书，所以每次的霍夫曼码书可能略有不同，我们输出对应的每次的平均码长，由于信道传输消息的时间是取决于信源生成消息序列的时间，所以信道数据率为，即8×信源信息率/输出序列的平均码长。

由于采用了霍夫曼编码，信源实际上被压缩，信息率提高，所以信道的输入信息率会大于等于信源的信息率。

理论推导结果为:

a.信源的信息率:



其中M为是总字节数，由于本题中设定信源的数据率为1数据比特/秒，传输时间为1s，故M=1信息比特。



**(3).信源概率分布，不使用信源编码**

由于信源概率分布****，信息熵为:



定义信源数据率为1bit/s，所以信源信息率为

。

无信源编码，则信道的输入信息率等于信源信息率；

无信道编码，信道无噪无损，所以信道的输出信息率与等于信道的输入信息率，不存在误码率。理论推导结果为:

a.信源的信息率:



其中M为是总字节数，由于本题中设定信源的数据率为1数据比特/秒，传输时间为1s，故M=1信息比特。

b.信道的数据率数据比特/秒

c.信道的输入信息率=0.8813信息比特/秒

d.信道的输出信息率=0.8813信息比特/秒

e.信宿的误码率=0

**(4).信源概率分布，使用信源编码**

由于信源概率分布****，信息熵为:



定义信源数据率为1数据比特/秒，所以信源信息率为

。

无信道编码，信道无噪无损，所以信道的输出信息率与等于信道的输入信息率，不存在误码率。

有信源编码：由于我们是在每次的信源生成消息后，根据当前信源概率分布生成霍夫曼码书，所以每次的霍夫曼码书可能略有不同，我们输出对应的每次的平均码长，由于信道传输消息的时间是取决于信源生成消息序列的时间，所以信道数据率为，即8×信源信息率/输出序列的平均码长。

由于采用了霍夫曼编码，信源实际上被压缩，信息率提高，所以信道的输入信息率会大于等于信源的信息率。

理论推导结果为:

a.信源的信息率:



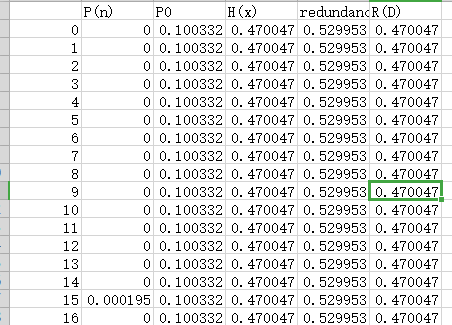
其中M为是总字节数，由于本题中设定信源的数据率为1数据比特/秒，传输时间为1s，故M=1信息比特。



### 仿真

**(1).信源概率分布时，不使用信源编码：**

先用程序byteSource.py生成一个概率分布为离散无记忆信源(大小为1KB)，调用程序byteSource\_calc.py可以计算其信源信息率的大小(数据率为1bit/s)(文件名为：P0=0.1.LEN=10KB.calc.CSV)：

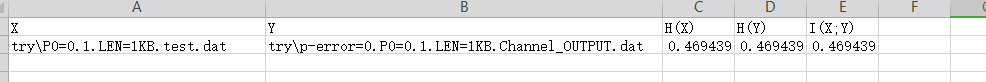


从上图的第六列可以知道计算的信源信息率为0.4700信息比特/秒(保留四位小数)。

再通过程序byteChannel.py生成过信道的序列，如下：



再调用byteChannel\_calc.py的函数进行对其过信道的序列进行信道的输入与的信息熵的计算，如下(文件名：P0=0.1.Channel\_not\_source\_calc\_result.CSV)：



由于取信源的数据率为1数据比特/秒，由于不使用信源编码，因此，信道的数据率为数据比特/s。再与上图中的信息熵分别相乘，即可得到信道的输入与输出的信息率，经过计算可以得到信道的输出信息率=输入信息率=0.4694信息比特/秒(保留四位小数)。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 信源信息率  （信息比特/秒） | 信道数据率  （数据比特/秒） | 信道输入信息率（信息比特/秒） | 信道输出信息率（信息比特/秒） | 误码率 |
| 理论值 | 0.4690 | 0.4690 | 0.4690 | 0.4690 | 0 |
| 仿真值 | 0.4700 | 0.4694 | 0.4694 | 0.4694 | 0 |

**(2).信源概率分布时，使用信源编码：**

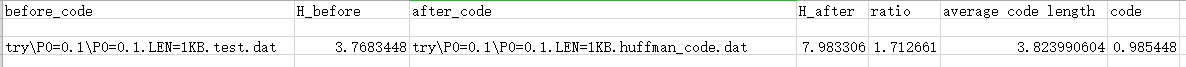
将之前得到的信源文件先进行霍夫曼编码，即通过程序：byteSourceEncoder.py实现，(其中的霍夫曼码书由程序Doublelisan.exe生成)得到使用编码后的信源文件(调用命令见程序API)：



将得到的霍夫曼编码不进行信道编码后，通过信道，即调用程序：byteChannel.py可以获得一个通过信道后的文件：



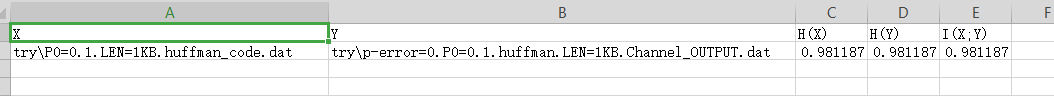
调用程序codding\_effect.py可以计算8次扩展后的所得未编码前的信源信息熵，编码后的信源信息熵(即输入信道前的信息熵)以及输出序列的平均码长：

 根据上图数据，可以计算下列实际值：

仿真的信源信息率：

仿真的信道数据率：

调用程序byteChannel\_calc.py可以得到下列数据(文件：P0=0.1.Channel\_source\_huffman\_calc\_result.csv)：



从上式可得信道输入与输出的信息熵，可以认为是仿真的输入与输出的信道的输入信息率以及信道的输出信息率。

理论推导的结果如下：

信源信息率=

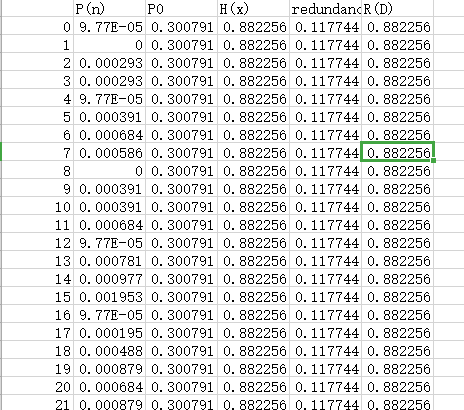
信道数据率=

信道输入信息率=信道输出信息率=信道数据率=0.9812数据比特

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 信源信息率  （信息比特/秒） | 信道数据率  （数据比特/秒） | 信道输入信息率（信息比特/秒） | 信道输出信息率（信息比特/秒） | 误码率 |
| 理论值 | 0.4690 | 0.9812 | 0.9812 | 0.9812 | 0 |
| 仿真值 | 0.4710 | 0.9979 | 0.9979 | 0.9979 | 0 |

**(3).信源概率分布时，不使用信源编码：**

先用程序byteSource.py生成一个概率分布为离散无记忆信源(大小为1KB)，调用程序byteSource\_calc.py可以计算其信源信息率的大小(数据率为1bit/s)(文件名为：P0=0.3.LEN=10KB.calc.CSV)：

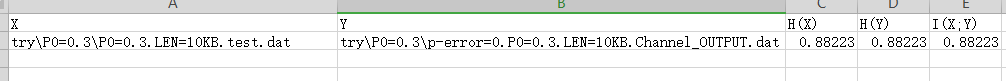


从上图的第六列可以知道计算的信源信息率为0.8823信息比特/秒(保留四位小数)。

再通过程序byteChannel.py生成过信道的序列，如下：



再调用byteChannel\_calc.py的函数进行对其过信道的序列进行信道的输入与的信息熵的计算，如下：



由于取信源的数据率为1数据比特/秒，由于不使用信源编码，因此，信道的数据率为数据比特/s。再与上图中的信息熵分别相乘，即可得到信道的输入与输出的信息率，经过计算可以得到信道的输出信息率=输入信息率=0.8823信息比特/秒(保留四位小数)。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 信源信息率  （信息比特/秒） | 信道数据率  （数据比特/秒） | 信道输入信息率（信息比特/秒） | 信道输出信息率（信息比特/秒） | 误码率 |
| 理论值 | 0.8813 | 0.8813 | 0.8813 | 0.8813 | 0 |
| 仿真值 | 0.8823 | 0.8823 | 0.8823 | 0.8823 | 0 |

**(4).信源概率分布时，使用信源编码：**

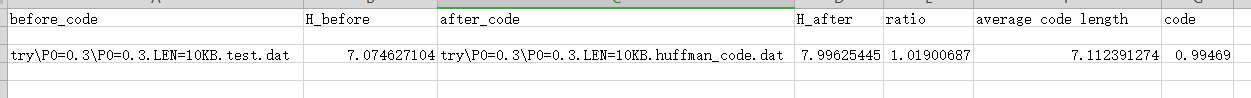
将之前得到的信源文件先进行霍夫曼编码，即通过程序：byteSourceEncoder.py实现，(其中的霍夫曼码书由程序Doublelisan.exe生成)得到使用编码后的信源文件(调用命令见程序API)：



将得到的霍夫曼编码不进行信道编码后，通过信道，即调用程序：byteChannel.py可以获得一个通过信道后的文件：



调用程序codding\_effect.py可以计算8次扩展后的所得未编码前的信源信息熵，编码后的信源信息熵(即输入信道前的信息熵)以及输出序列的平均码长：

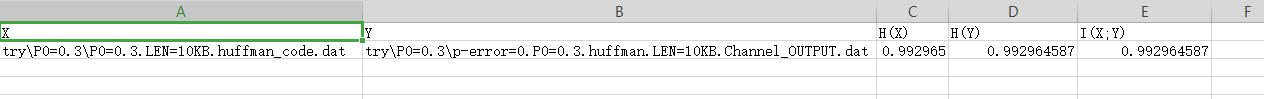


根据上图数据，可以计算下列实际值：

仿真的信源信息率：

仿真的信道数据率：

调用程序byteChannel\_calc.py可以得到下列数据(文件：P0=0.1.Channel\_source\_huffman\_calc\_result.csv)：



从上式可得信道输入与输出的信息熵，可以认为是仿真的输入与输出的信道的输入信息率以及信道的输出信息率。

理论推导的结果如下：

信源信息率=

信道数据率=

信道输入信息率=信道输出信息率=信道数据率=0.9912数据比特

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 信源信息率  （信息比特/秒） | 信道数据率  （数据比特/秒） | 信道输入信息率（信息比特/秒） | 信道输出信息率（信息比特/秒） | 误码率 |
| 理论值 | 0.8813 | 0.9912 | 0.9912 | 0.9912 | 0 |
| 仿真值 | 0.8843 | 0.9995 | 0.9995 | 0.9995 | 0 |

### 分析

从以上4种情况的理论推导与仿真结果可以看出，仿真结果大致与理论推导结果一致，对于相应的仿真值与理论值相比，仿真值较大，原因可能与计算时的保留小数位有关。可以从以上情况看出，对于存在信源编码时，通过无噪无损离散无记忆信道且不经过信道编码的信道输入信息率(输出信息率)要大于信源信息率，原因在于huffman编码为非等长编码，计算时需要考虑平均码长。

## 信道非理想情况下

### 条件设定

**要求**

* 固定条件
  + 信源的数据率*rs*（数据比特/秒）
  + 信源为等概率二元DMS
  + 不使用信源编码
* 可变条件
  + 3种信道错误传递概率：0.01, 0.2, 0.5
  + 无信道编码、重复码、(*n*,*k*)线性分组码

### 理论推导

### 仿真

### 分析

## 一般非理想情况下

### 条件设定

**要求**

* 固定条件
  + 信源的数据率*rs*（数据比特/秒）
* 可变条件
  + 2种信源概率分布：0.1, 0.3
  + 3种信道错误传递概率：0.01, 0.2, 0.5
  + 无信源编码、有信源编码
  + 无信道编码、重复码、(*n*,*k*)线性分组码

### 理论推导

### 仿真

### 分析

# 系统改进【可选】

**要求**

* 针对仿真所出现的现象，提出对系统设计、实现、仿真等方面的改进设想。
* 分析说明改进的目的、原理和可行性依据。
* 实施对系统的改进，并通过仿真验证改进的效果。

# 总结

|  |  |  |
| --- | --- | --- |
| 序号 | 已完成的内容 | 经验与收获 |
| 2.1 | 信源单元测试：  可自由设定信源消息概率分布  可自由设定生成的消息序列的长度  信源概率分布统计表的生成  离散无记忆信源的信息熵的计算  信源冗余度的计算 | 在实际运行该模块时，发现离散无记忆信源的信息熵与生成消息序列的长度无关，只与各字节的概率分布相关。 |
| 2.2 | 信道单元测试：  可自由设定信源消息概率分布  可自由设定生成的消息序列的长度  可自由设定信道的错误传递概率  信道的输入信息熵的计算  信道的输出信息熵的计算  平均互信息量的计算  信道容量的计算 | 当信源等概率分布时，平均互信息量最大，此时的平均互信息量为信道容量。 |
| 2.3 | 信源编解码单元测试：  可自由设定信源消息概率分布  可自由设定生成的消息序列的长度  霍夫曼码书构造  霍夫曼编码  霍夫曼解码  平均码长的计算  编码效率的计算 | 霍夫曼编码是变长码，考虑了信源的统计特性，而且是最佳码，对于同一个信源，编码并非唯一，但平均码长都相同。 |
| 2.4 | 信道编解码单元测试：  可自由设定信源消息概率分布  可自由设定生成的消息序列的长度  可自由设定信道的错误传递概率  汉明码编码  汉明码解码  误码率*R*error（汉明失真）的计算 | 汉明码编解码误码率较低 |
| 3.1 | 理想情况下的系统仿真与设计：  可自由设定生成的消息序列的长度  信源的信息率的计算  信道的数据率的计算  信道的输入信息率的计算  信道的输出信息率的计算  信宿的误码率的计算 | 理想情况下信道的输入输出信息率相等，误码率为0 |
| 3.2 | 信源非理想情况下的系统仿真与设计：  可自由设定信源消息概率分布  可自由设定生成的消息序列的长度  可自由设定是否使用信源编码  信源的信息率的计算  信道的数据率的计算  信道的输入信息率的计算  信道的输出信息率的计算  信宿的误码率的计算 | 信源非理想时，无论是否使用信源编码，信道的输入输出信息率相等，误码率为0 |
| 3.3 | 信道非理想情况下的系统仿真与设计：  可自由设定生成的消息序列的长度  可自由设定信道的错误传递概率  可自由设定是否使用信道编码  信源的信息率的计算  信道的数据率的计算  信道的输入信息率的计算  信道的输出信息率的计算  信宿的误码率的计算 | 信道非理想时，在信源分布概率和错误传递概率相同的条件下，使用信道编码比不使用信道编码会提高信道的输入信息率，错误传递概率越小，误码率越低 |
| 3.4 | 一般非理想情况下的系统仿真与分析：  可自由设定生成的消息序列的长度  给定两种信源概率分布  给定三种信道的错误传递概率  无信源编码、有信源编码  无信道编码、重复码、(*n*,*k*)线性分组码  信源的信息率的计算  信道的数据率的计算  信道的输入信息率的计算  信道的输出信息率的计算  信宿的误码率的计算 | 一般非理想情况的通讯条件设定较复杂， |

# 附录——相关技术简介

**要求：**简要介绍本设计中所使用的

* 计算机语言
  + 官方名称、版本号、官网URL
* 主要函数包/模组
  + 只需包含非标准函数包/模组等（不是计算机语言自带的）
  + 官方名称、版本号、官网URL
  + 在本设计中的用途
* 工具软件
  + 编程开发、报告撰写、作图、其他辅助工具等
  + 官方名称、版本号、官网URL
  + 在本设计中的用途