信息论与编码——信源编码实验指导

1 概要

本次实验的目的是让同学们熟悉信源编码的不同方法，将其应用于实际程序，从而更好地理解信源编码；此外，也锻炼一下同学们的编程能力。

考虑到同学们编程基础与空闲时间都有所差别，因此实验拥有**不同的套组供选择**，具体来说：

（1）拥有多个必选模组，你必须选择必选模组之一实现。

（2）拥有一个可选模组，在完成必选模组的基础上，你可以选择完成可选模组。

（3）拥有几个实践问题，你需要完成所有实践并完成问题的回答。

你可以使用你喜欢的语言实现功能，选择的语言不会影响你的分数。对核心功能以外的部分，你可以自由使用不同的库/包。

你可以参考不同来源的各种实现，但请务必确保代码由你自己编写。

我们会编译并运行你的代码，请确保你测试运行了你的代码。

2 计分

实验总计 15 分，由几个部分组成：

（1）必选模组的完成：5~10 分。

（2）实践问题的完成：0~5 分。

（3）可选模组的完成：1~4 分。

如果你的计算分数高于 15 分，会记为 15 分。

2.1 必选模组

必选模组的分数细则如下：

（1） 不同模组的难度并不相同，但是无论何种模组都有机会获得高分；然而，公平起见，难度较低的模组会有更严苛的评分标准，具体如下：

模组	功能实现	代码风格	用户交互	代码鲁棒性与安全性
必选模组 1——Huffman 编码	5	1	3	2
必选模组 2——Huffman、Shanno-Fano、LZ77 与 LZ78	7	1	2	1
必选模组 3——算术编码	8	1	2	1

如果分数高于 10 分，会按 10 分计算。

（2） 功能实现：实现了对任意文件编码、解码的基本功能，**你应该确保编码再解码文件后，其 SHA256 值与原文件相同，并在实验报告中展示这一点**。

（3） 代码风格：代码风格决定了代码的可维护性。你的代码应该有清楚的变量命名、恰当的注释和合理的结构设计。

（4） 用户交互：缺少用户交互的终端应用并不完整，这部分分数来源于设计一个可用的用户交互。

我们不鼓励你设计 GUI（当然，设计 GUI 也能拿到这部分分数），而鼓励你设计 CLI 交互，一个来自 tar 的良好示例如图：

```
[21:10:26] moonmagian@moonarch /home/moonmagian/tartest
> tar -czvf out.tar.gz a.txt b.txt
a.txt
b.txt
[21:10:42] moonmagian@moonarch /home/moonmagian/tartest
> tar -xzvf out.tar.gz
a.txt
b.txt
[21:11:20] moonmagian@moonarch /home/moonmagian/tartest
> tar --help
Usage: tar [OPTION...] [FILE]...
GNU 'tar' saves many files together into a single tape or disk archive, and can
restore individual files from the archive.

Examples:
  tar -cf archive.tar foo bar  # Create archive.tar from files foo and bar.
  tar -tvf archive.tar         # List all files in archive.tar verbosely.
  tar -xf archive.tar          # Extract all files from archive.tar.

 Main operation mode:
  -A, --catenate, --concatenate   append tar files to an archive
  -c, --create               create a new archive
      --delete               delete from the archive (not on mag tapes!)
  -d, --diff, --compare      find differences between archive and file system
  -r, --append               append files to the end of an archive
      --test-label           test the archive volume label and exit
```

当然，tar 是一个十分复杂而强大的工具，此处只作为举例，你只需要设计一个可用的、完整的用户交互接口便能拿到这部分分数。

根据语言的不同，你可以使用一些工具辅助你设计，这里举几个常见的库/包（当然，还有许多不同的选择）：

语言	包
Python	argparse（Built-in 包）
C	getopt（位于 unistd.h）
C++	Boost.Program_options （https://www.boost.org/doc/libs/1_63_0/doc/html/program_options.html）
Java	Commons CLI（http://commons.apache.org/proper/commons-cli/）
Rust	clap（https://github.com/clap-rs/clap）

代码鲁棒性与安全性：你的程序应该考虑到一些边界情况和错误，例如：对空文件的处理；对仅含有一种字节的文件的处理（在 Huffman 编码中需要特别考虑）；对错误的解码输入的部分处理（欲解码的文件不是由该程序编码，可以使用校验、MagicNumber 等方式）；涉及到序列化、反序列化时的任意代码执行（Python 的 pickle 需要特别考虑）；对缓冲区溢出的处理等。

对安全性和鲁棒性的考量是相当困难的，即使强如 openssl 也有许多严重的安全漏洞，因此，你只需要按自己的理解尽可能使代码鲁棒，**并且在代码或报告中说明所做的工作**，便可以拿到这部分分数。

2.2 可选模组

正确实现可选模组会拿到全部的可选模组成绩。即使是部分实现，也能拿到部分成绩，你应该在代码中保留你的部分工作并在实验报告说明。

2.3 实践问题

在实践问题中，你需要使用你在必选模组中编写的程序处理一些实际的任务，并回答一些相关的问题。问题的答案有时候并不是显然，请善用搜索引擎。

你的成绩与对实践问题的解答（这体现了你对编码方式的理解）相关。

3 模组

此处介绍不同的模组，你需要**从必选模组中选取一个进行实现**，在完成必选模组的基础上可以选择完成可选模组。

3.1 必选模组 1——Huffman 编码的实现

Huffman 编码是课程中重点学习的编码方式之一，在必选模组 1 中，你需要实现使用二元 Huffman 编码对文件的编码和解码。

你的解码操作应该仅依赖于编码后的文件，而不依赖于内存中的任何量。（即，不能将编码时在内存构建的码树直接用于解码，而需要以恰当的方式存储在编码后的文件，在解码时将其复原出来）。

需要注意，选择实现 Huffman 的同学除了实现常见的对以 byte 为单位作为符号进行编码，还需要实现对 n byte 单位符号的编码（至少实现 2byte），应用在第 4 部分中的实践。

3.2 必选模组 2——Huffman 编码、Shannon–Fano 编码、LZ77 编码与 LZ78 编码四选二

LZ77 和 LZ78 编码相比 Huffman 编码、Shannon-Fano 编码在实际的压缩、解压场景中应用要多得多。

在必选模组 2 中，你需要选择 Huffman 编码或 Shannon-Fano 编码实现其中之一，并选择 LZ77 编码或 LZ78 编码实现其中之一（如果你愿意，也可以选择 LZW 或者 LZMA 编码）。

之后，你需要综合你选择的两种方式，实现对文件的编码与解码。（可以先使用 LZ 编码，再使用剩下的编码）

同样的，你的解码操作应该仅依赖于编码后的文件，而不依赖于内存中的任何量。（额外地，对于 LZ 编码，你可以依赖于一个与你的程序一同分发的预构建的字典，当然也可以不用）

3.3 必选模组 3——算术编码

算术编码有着不少的优秀性质，但它的计算机实现并非那么容易。

在必选模组 3 中，你需要使用算术编码（或者一些实现变种，例如区间编码）实现对文件的编码与解码。

同样的，你的解码操作应该仅依赖于编码后的文件，而不依赖于内存中的任何量。

3.4 可选模组——流压缩与解压缩

设想一个情景：你的硬盘还剩 10GiB 空间，而你想要立即下载一份 12GiB 的数据保存起来（不会立即使用，仅仅是保存），你的硬盘中全是不能删除的重要文件，你的内存也只有 1GiB，怎么办？

一个可行的方法是去借一块 16GiB 的 U 盘，下载到 U 盘后将其压缩，之后转存到硬盘里。

设想另一个场景：你从服务器接收 100GiB 的压缩文本，其中只有几行含有关键的字符串，需要从中找到这一行。你的硬盘只剩 1MiB 可用，你的内存也相当有限，怎么办？

这个情景似乎比上一个情景更头大。

好在，这种情景在 90 年代过于常见了，以至于操作系统都使用了管道和输入输出流解决它。

想像一个未经过滤的水源，我们希望得到饮用水，一种做法是将所有水积蓄到一个大的蓄水池，使用不同的设施处理蓄水池中的水，一次性得到所有饮用水。

比起这种方式，一个更有效率的方式是从水源建立一个管道，一次只让数量不多的水流过各个设施，得到一部分饮用水，使用这种方式，我们不再需要建立一个极其庞大的蓄水池，而仅仅需要一些管道。

现在，将数据比喻成水流，程序比喻成各个设施，我们用一组管道连接各个程序，将上个程序的标准输出连接到下个程序的标准输入，之前的两个场景似乎可以解决了。

对第一个情景，我们连接这样一组程序：

下载程序（将下载的数据输出到标准输出流）→压缩程序→文件

我们不借助额外的硬盘空间，在下载时便将数据压缩到文件。

```
# moonmagian @ moonarch in ~ [22:28:18]
$ curl "www.buaa.edu.cn/donna_donna.iso" | gzip > donna_donna.iso.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 23.8M  100 23.8M    0     0  34.8M      0 --:--:-- --:--:-- --:--:-- 34.8M

# moonmagian @ moonarch in ~ [22:28:19]
$ ls -lh donna_donna.iso.gz
-rw-r--r-- 1 moonmagian moonmagian 21M Apr  7 22:28 donna_donna.iso.gz
```

如图展示了使用 curl 和 gzip 建立一个这样的流（仅供演示使用）

对第二个情景，我们连接这样一组程序：

下载程序→解压缩程序→按行检索数据的程序→输出

我们一边下载一边检索信息，从而不再需要预先完全下载文件。

```
# moonmagian @ moonarch in /usr/share/nginx/html [22:35:33]
$ curl "www.buaa.edu.cn/never_gonna_give_you_up.txt.gz" | gunzip | grep -yn "never gonna"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   444  100   444    0     0   433k      0 --:--:-- --:--:-- --:--:--  433k
14:Never gonna give you up
15:Never gonna let you down
16:Never gonna run around and desert you
17:Never gonna make you cry
18:Never gonna say goodbye
19:Never gonna tell a lie and hurt you
31:Never gonna give you up
32:Never gonna let you down
33:Never gonna run around and desert you
34:Never gonna make you cry
35:Never gonna say goodbye
36:Never gonna tell a lie and hurt you
37:Never gonna give you up
38:Never gonna let you down
39:Never gonna run around and desert you
40:Never gonna make you cry
41:Never gonna say goodbye
42:Never gonna tell a lie and hurt you
48:Never gonna give, never gonna give (Give you up)
50:Never gonna give, never gonna give (Give you up)
62:Never gonna give you up
63:Never gonna let you down
64:Never gonna run around and desert you
65:Never gonna make you cry
66:Never gonna say goodbye
67:Never gonna tell a lie and hurt you
68:Never gonna give you up
69:Never gonna let you down
70:Never gonna run around and desert you
71:Never gonna make you cry
72:Never gonna say goodbye
73:Never gonna tell a lie and hurt you
74:Never gonna give you up
75:Never gonna let you down
76:Never gonna run around and desert you
77:Never gonna make you cry
78:Never gonna say goodbye
79:Never gonna tell a lie and hurt you
```

如图展示了使用 curl,gzip 和 grep 建立一个这样的流（仅供演示使用）

你的任务是修改你在必选模组中完成的程序（请做好备份），使其能接受标

准输入，将结果输出到标准输出。

输出到标准输出是十分简单的，但是，处理输入并不那么容易。

对于标准输入：你无法通过 fseek 一类的函数重读以前的数据了（可以认为，从标准输入读入的数据被“消耗”了）；此外，你也无法用 ftell 和 fseek 拿到文件大小了（你会得到 0 或者-1）；你也不应该将标准输入读到一个超大的缓冲区再操作（这失去了使用流的意义）。

因此，你需要设立一个大小恰当的缓冲区（gzip 压缩时默认使用 36K 的输入缓冲和 8K 的输出缓冲，在这个实验你可以不考虑用于提高性能的输出缓冲），每当数据填满缓冲区，便单独地将当前块编码、解码（因此，在编码时每一个输出缓冲大小就应该保存一份码表，在解码时最易于实现的缓冲区大小是输入缓冲+码表尺寸（这样读满一个缓冲便是读入了一个完整分块）），你应该特别考虑流的总长度不是缓冲大小的整数倍的情况。

为了测试你的程序，在终端或 cmd 下分别输入：

```
your_program_in_encode_mode < some_file > some_file.enc
your_program_in_decode_mode < some_file.enc > some_file.dec
```

some_file 是已经存在的一个文件。

之后，为了验证程序的正确性，对 macOS 或 Linux，在终端输入：

```
Openssl dgst -sha256 some_file some_file.dec
```

对 Windows，在 cmd 输入：

```
certUtil -hash filesome_file SHA256
certUtil -hash filesome_file.dec SHA256
```

两个文件的 SHA256 值应该相同。

4 实践问题

4.1 重复性的文件结构

使用你在必选模组实现的程序，编码 lab1_data/testfile1，lab1_data/testfile1 的文件内容是：

256 字节 0x00，256 字节 0x01，256 字节 0x02，...，256 字节 0xff，总计 64KiB。

观察编码后的文件大小，解释为什么文件体积会发生这种变化（为什么会这样（模组 1 和模组 3）/是什么发挥了作用（模组 2））。

模组 1 和模组 2 中实现 Huffman 编码的同学还需要在不同长度符号下，编码 lab1_data/testfile2，lab1_data/testfile2 的文件内容是：
16 字节 0x00，16 字节 0xFF，16 字节 0x00，...，总计 8KiB。

观察编码后的文件大小，解释一下对符号的不同定义如何影响压缩效率？思考一下符号定义的最优解？（可以思考编码表和编码文件大小之间的制约）

实验文件下载：

https://bhpan.buaa.edu.cn:443/link/71FED9B61D0DD99F7383815580D9681B

4.2 不同格式的压缩

使用画图或者其他工具进行一些简单的艺术创作（推荐使用三四种颜色，不要太多，尽量使用较大的分辨率（3840x2160）），分别将图片保存为 bmp 和 jpeg，尝试使用你在必选模组实现的程序编码图片，对比地解释为什么文件体积会发生这种变化。（可以图像化频度表以更好地展示与解释）

类似的，尝试编码一个可执行文件，解释文件体积的变化。

更进一步，可以选择继续探究不同格式文件（.txt，.mp4，.avi，.zip 等），或者不同内容（中文文本和英文文本）的压缩效果。

4.3 黑洞！

如果我将一个超大的文件压缩几百次，他就会变得越来越小，最后达到几 KB，这便是时间换空间！

这种说法正确吗？

对 2 中的 bmp 图片迭代的编码 10 次，观察结果的体积变化。

为什么会发生这种变化？结合你学过的知识解答。

5 实验提交

你需要提交包含实验报告、程序源代码和过程文件（用于测试你的程序的编码后、解码后的文件）的压缩包。

请使用 7z、rar 等格式，或在 utf-8locale 下使用 tar.gz，不要使用 zip。

实验报告只需要包含运行源代码所需要的依赖；对必选模组代码的必要解释、运行测试截图；对可选模组代码的必要解释、运行测试截图；实践问题的运行截图和解答。

实验报告并无特定的格式，但**需要转换成 pdf**。

只要实验报告包含所有必要的内容和解释，它便不会影响你的分数，字数多的实验报告不会给你带来任何加分。

你需要将压缩包命名为 `2022 信息论与编码-实验 1-学号-姓名`，例如 `2022 信息论与编码-实验 1-18371111-神秘人.7z`，你可以运行 `lab1_data/check.py` 检查你的文件名是否合乎规范。

在截止日期前，你可以无限次地修改并重新上传压缩包（只要保持文件名一样，便可覆盖上传）。

请不要覆盖别人（例如你的室友）的作业。

在确认无误后，将你的压缩包上传至：

https://bhpan.buaa.edu.cn:443/link/FAE6CEB521367E0837E7BF414EA7AD1E

此外，还需要在雨课堂提交对应的实验作业（不需要在雨课堂上传任何内容，仅仅为了方便评分）。

你应该在北京时间 2022 年 4 月 30 日 23:59 前完成上述的提交。