信息论及其应用实验一：数据压缩
实验报告

史泽宇

2020 年 4 月 3 日

1 实验内容

将附件中的英文文献用最佳不等长编码进行压缩。即利用霍夫曼编码（Huffman Code）、算术编码（Arithmetic Code）和 LZ 编码（Lempel-Ziv Code）任一编码算法对文献实现无损压缩。

2 实验环境

2.1 硬件环境

阿里云服务器，Ubuntu 18.04 LTS

- System:

  Host: iZbp15f6uou33mcqggu2flZ

  Kernel: 4.15.0-91-generic x86_64 bits

  Console: tty 4

  Distro: Ubuntu 18.04.4 LTS

- Machine:

  Device: kvm System

  Alibaba Cloud product: Alibaba Cloud ECS

  N/A BIOS: SeaBIOS

- CPU: Single core Intel Xeon Platinum 8269CY (-MCP-) speed: 2500 MHz (max)
- Graphics: Card: Cirrus Logic GD 5446 Display Server: N/A driver: cirrus tty size: 120x30 Advanced Data: N/A for root out of X
- Network: Card: Red Hat Virtio network device driver: virtio-pci
- Drives: HDD Total Size: 42.9GB (25.3% used)

- Info: Processes: 124 Uptime: 5 days Memory: 1291.8/1911.9MB Init: systemd runlevel: 5 Client: Shell (fish) inxi: 2.3.56

2.2 运行环境

- 程序设计语言：Rust
- Linux 操作系统 Ubuntu 18.04.4 LTS

3 实验过程

使用 Rust 程序设计语言实现了 LZ 编码，Huffman 编码以及算术编码。并提供了简单的 CLI 接口，可以在命令行下完成各种编码操作。其中 LZ 编码与 Huffman 编码部分可以应对所有输入，但是算术编码部分由于实现过程中使用了语言内置的 f64 浮点数类型，对于较长的输入会导致数据溢出的问题。

3.1 LZ Code

编码过程中首先读取提供的字母表集合，按照字母表中字母的顺序生成对应的编码映射。接着读取文本内容，按照 LZ 算法对文本进行分段，并生成分段的编码映射，最后将编码输出到文件中，分别使用文本模式与二进制模式进行处理。

解码过程中首先读取提供的字母表集合，按照字母表中字母的顺序生成对应的编码映射。接着读取文本内容，按照输入的编码长度对编码后的数据进行分割，并将分割后的数据映射到字母表中的字母，并进行输出。

3.2 Huffman Code

编码过程首先遍历一次输入文档，计算文档中各个字母对应的概率，生成字母表对应的概率分布。接着将概率分布转化为树中的叶子节点并将这些叶子节点存储到优先队列中。按照输出码元的个数从优先队列中取出叶子节点并构建树，构建完成整个 Huffman 树之后对树进行层序遍历，层序遍历中会产生叶子节点的 Huffman 编码。最后使用字母表以及对应的编码对输入文档编码输出。除了文本模式与二进制模式的输出之外，还会以 JSON 序列化输出字母以及对应的概率，可以用于之后进行解码。

解码过程会读取编码输出的概率分布，然后在输入的编码后文档进行字符串匹配以解码产生输出。

3.3 Arithmetic Code

算术编码中的编码与解码过程主要还是书上的迭代算法，没有使用另外的数据结构。但是存在的问题是由于编程语言中内置的 f64 浮点数类型的精度有限，所以不能对较长的输入进行编码。可以考虑使用第三方高精度浮点数库来辅助生成编码，但是可能不是最佳解决方案，效率比较低下。简单查阅了一些资料后发现存在一些优化方案来辅助生成编码，但是都比较复杂，估计了一下复杂度后决定放弃算术编码的完美实现。