利用哈夫曼编码实现压缩与解压

高士杰

1014186239@qq.com

2020 年 3 月 31 日

Contents

1 编译环境与使用方法

1.1 编译环境

本文代码编译于 macOS 10.13。

1.2 使用方式

1. **编译**

   gcc -o compressor huffman_compressor.c

   gcc -o decompressor huffman_decompressor.c

2. **压缩，程序会生成 *.txt.compress 压缩文件**

   ./compressor ./usconstitution.txt

3. **解压**

   ./decompressor ./usconstitution.txt.compress us.txt

2 程序结构

程序主要分为两部分，即压缩器和解压器。

2.1 压缩器

压缩器包括多个部分和流程。简要介绍即：首先，压缩器读入欲压缩文档，按照字符读入，随后将字符通过哈希表记录于字典之中。随着文件不断读取，程序会统计出不同字符出现的频率。

随后，将不同字典键值构成的哈夫曼树节点，根据其权重从下到上构建起哈夫曼树。

之后再利用哈弗曼树推导出各叶子结点上字符对应的编码。

然后，再次读入欲压缩文档，按照字符读入，根据字典映射找到对应编码方式。将编码按照单 bit 的方式，按位与插入长 u_char 数组。

最后，将该长数组写回文件。

2.1.1 哈希映射字典

鉴于 C 语言不提供哈希表，需要手动实现。实现的方式如下：

首先建立一张类型为 int 的 hash_table。这张表的用途是：用于存放真正索引数据所在数组的下表。当用户将键值经过自制 hash 映射得到地址后，就利用该地址对应的下标到真正的表中 hash_list 进行检索。该表各元素初始化为-1；

自制 hash 函数，手工设置种子值，以混淆 hash 值，随后将键值按位转整数后相加并累乘，上溢则取余。

hash 插入，键值经过 hash 运算得到地址值后，检索其值，如果是-1，说明可以插入。如果不是-1，则地址递增直到有空位插入。

hash 查询，键值经过 hash 运算得到地址值后，检索其值，并根据该值到 hash_list 中找到真实字符，如果不同则地址递增直到找到。如果遇到-1，说明表中不存在该键值。

2.1.2 哈夫曼树构建

通过统计得到各字符对应的频率即该叶子结点的权重后，将所有叶子结点指针存放于数组当中。

随后进行循环，循环的条件是数组中元素超过一个。这时，找到现存于数组中节点权重最小的两个的下标。随后生成新的节点，节点的权重为两个节点的权重之和。将该节点作为上述两节点的父节点，即父节点中左右孩子指针分别指向上述两个节点，上述两个节点父指针指向新节点。随后新节点侵占两个节点中的一个，并删除另一个。

在每一轮循环中数组减少一个节点，当只剩下一个节点时，即根节点，返回该节点。

2.1.3 产生哈夫曼编码

根据上节得到的哈夫曼树，利用前序递归方式，标记每个叶子结点的编码值。

具体实现中，是将编码对应 0、1 作为字符串写入节点中，并记录编码长度方便后续使用。

2.1.4 压缩数据

重新读入欲压缩文档，记录文档长度。随后以处理每个字符为主，根据该字符查找字典，得到对应的编码。由于编码以字符串方式存储，所以建立新一级循环，在循环内，以 bit 为单位，分别根据 “0、1” 及位长左移数位，并与目标 char 数组中的某个 char 进行交运算。如果目标 char 填满，则行进到下一级。

该方法以 bit 为单位依次填充，最后一个 byte 很可能产生空白，空白则补充 0。同时记录 bit 总数。

得到被压缩的数据后，就可以写入文件。为了方便解码，哈夫曼树的结构也将写入文件。文件按照以下方式组织：

哈弗曼树节点数	节点结构编码 1	...	文本编码 bit 长度	文本编码

其中节点结构编码为：

字符 1	编码长度	编码 1	编码 2	...

2.2 解压器

解压器中，hash 表与压缩器相反，其键值为相应的编码。

解压器首先需要解析文件头的哈夫曼树。只需要根据压缩器写入的数据格式，依次将树结构读出，并写入字典即可。

读入被压缩的文件后，以 byte 为单位，分别利用位并操作从左到右依此读取 0、1 数据。每读入一位即转为相应字符串，并在字典中进行比对。如果匹配，则将对应字符写入目标 char 数组，随后推动指针向前继续读取数据。

3 实现效果

本文进行了一些简单实验，罗列如下：

原始数据	原始大小	压缩后大小	压缩率
usconstitution.txt	119,306 字节 (123 KB)	72,794 字节 (74 KB)	38.98%
1.txt	1,670,294 字节 (1.7 MB)	1,001,099 字节 (1 MB)	40.06%
2.txt	637,853 字节 (639 KB)	384,026 字节 (385 KB)	39.79%

4 存在的问题

1. 经实验，本程序并不能压缩非文本文件。经压缩、解压后得到的文件是损坏的。同样的，两次压缩（即对第一次压缩产物再压缩）两次解压，文件同样损坏。

2. 程序中使用的定长数组，为求方便，使用的空间定的偏大，浪费空间。

3. 对压缩器来说，以但 char 字符为编码对象，事实上不需要进行 hash 操作，最大不过 256 个空间而已，可以直接索引。

4. 构建哈夫曼树中用到求两个最小权重，可以实现一种通用的堆排序。但对于本文的用途来说，也够用。

5. 存于哈夫曼树节点中的字符的编码，不应该使用字符串，而应该使用二进制方式保存。如果不担心浪费空间，直接用 long long 存储未尝不可。

6. 压缩数据中对单 bit 的操作过于粗暴简单，效率低下。

7. 解码器运行效率很低，主要是匹配编码的方式依靠将编码按位转为字符串后检索字典，导致每一位都要进行大量比较操作。应该探索更高效的解码匹配方式

8. 代码组织混乱，函数命名混乱，变量命名混乱，缺乏多平台支持，调试方式暴力原始，缺少异常处理。