

关于LZ系列算法的分享

文昕颢

分析需要压缩的文件

[illegible]

重复的字符较多， 并且是每隔一段字符就会出现重复

那么， 我的第一想法就是， 能不能将这些重复的长字符串使用短的标记进行标记

这样， 我估计可以很大程度将其压缩，
问题是， 我怎么知道哪些是重复的字符？

我不可能全部进行暴力的遍历，来寻找哪些字符串是经常出现的

我也不可能指定我要替换那些字符，这样的程序只能压缩特定的日志文件

我就去寻找相关的压缩算法，找到的文章大多在各类博客、CSDN、知网或助教给的参考资料
但是有的文章特别是博客上的，讲述会有缺漏、错误或我有点难理解，所以我找了很多来对比着看

博客:

<https://www.cnblogs.com/idreamo/p/9249367.html>

<https://www.cnblogs.com/cliveleo/articles/9759019.html>

知网:

[https://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFD2008&filename=DNKF200805016&uid=WEEvREcwSIJHSlIdRa1FhdXNzY2Z2cUVISjJacFJodjBHZ3J2dk4xZEczOD0=\\$9A4hF_YAuvQ5obgVAqNKPCYcEjKensW4IQMowwHtwkF4VYPoHbKxJw!!&v=MjcxMzNvOUUVZb1I4ZVgxTHV4WVM3RGgxVDNxVHJXTTFGckNVUjdxZlkrUm9GeS9sVmJ2QUITUEFhTEc0SHRuTXE=](https://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFD2008&filename=DNKF200805016&uid=WEEvREcwSIJHSlIdRa1FhdXNzY2Z2cUVISjJacFJodjBHZ3J2dk4xZEczOD0=$9A4hF_YAuvQ5obgVAqNKPCYcEjKensW4IQMowwHtwkF4VYPoHbKxJw!!&v=MjcxMzNvOUUVZb1I4ZVgxTHV4WVM3RGgxVDNxVHJXTTFGckNVUjdxZlkrUm9GeS9sVmJ2QUITUEFhTEc0SHRuTXE=)

我发现LZ77算法非常符合我的要求，而且比较简单

LZ77算法中设定了一个滑动窗口和前向缓冲区，然后滑动窗口做主串，前向缓冲区做模式串，寻找最长的匹配字符串

这基于一个假设：重复的字符都出现在较近的位置

LZ77算法介绍:

- 1、设定滑动窗体大小，前向缓冲区大小，设一变量存储当前压缩到的位置
- 2、在滑动窗口中找到一个**最长的子串**，这个子串与**在前向缓冲区中以当前位置开始的子串**相同
如果找到了长度大于2的子串，当前位置后移此长度加一，转**步骤3**；
否则当前位置后移一格，转**步骤4**
- 3、获得匹配到的子串到当前位置的距离，然后将**距离、长度、匹配完后前向缓冲区的第一个字符**编码好放入压缩缓冲区
- 4、直接将**当前位置的字符**放入压缩缓冲区
- 5、当前位置移动后，滑动窗口和前向缓冲区的内容发生改变，回到步骤2，继续压缩直至完成

A A B C D B (5,3)D A E A (4,3)A B C D

	A	A	B	C	D	B	A	B	C	D	A	E	A	D	A	E	A	B	C	D
--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1	距离 (10位)	长度 (5位)	未匹配字符 (8位)
---	----------	---------	------------

0	未匹配字符 (8位)
---	------------

距离和长度的位数由滑动窗口和前向缓冲区决定

这个想法非常的妙，然后我兴致勃勃地写了出来，但是时间远远超过了15秒，我出去喝了杯水，回到电脑前，再等了很久才压缩完成

我认为的问题

- 一、字符串匹配太慢
- 二、读写文件太慢
- 三、频繁转换二进制

读入的优化

```
·istreambuf_iterator<char>·beg(fin),·end;·····//·设置两个文件指针，指向开始和结束，以char(一字节)为  
·string·content(beg,·end);·····//·将文件全部读入string字符串 LF  
·fin.close();·····//·操作完文件后关闭文件句柄是一个好习惯 LF
```

使用fstream的read进行读， write进行写
读入到char类型数组， 或将char类型数组写入文件

但是我们不知道这个数组要开多少， 那不妨设一个较大的数然后读入， 压缩完再读入， 直至文件结尾

字符串匹配的优化

暴力——对比来匹配显然行不通的

我是使用KMP来优化，看起来应该比暴力的要好

https://blog.csdn.net/dark_cy/article/details/88698736

Microsoft Visual Studio 调试控制台

49964ms



E:\GradeOnePrograme\HomeWork\EasyCompress\Debug\EasyCompress.exe (进程 13552) 已退出，返回代码为：0。
若要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...

但还是不能在15s内完成

频繁地转二进制

由于额外需要一个比特位来标记是否是匹配字符，
而且位数都不是8的倍数
我当时就想为了节约空间，压缩得更多，我就全部
转二进制，这个耗费的时间还是蛮多的

最大的瓶颈就在于字符串匹配，如果能极大地优化这个地方，对日志文件压缩效率还是很可观的
因为日志文件总是隔一段字符串就有重复的

 OUT	2020/4/26 14:01	文本文档	1,705 KB
 test	2020/4/16 16:35	文本文档	10,073 KB

我就百度有什么办法可以优化LZ77，首先是找到了哈希表优化，发觉这应该是个对的方向，然后就搜索最快的压缩算法，有网友提LZ4，恰好也是用哈希表来匹配字符串的，我就将其一起对比着看了

利用哈希表优化的介绍：

https://blog.csdn.net/jison_r_wang/article/details/52073517

https://blog.csdn.net/jison_r_wang/article/details/52073710

<https://blog.51cto.com/14239789/2472031>

LZ4介绍：<https://www.cnblogs.com/z-blog/p/8860799.html>

<https://www.jianshu.com/p/824e1cf4f920>

<https://blog.csdn.net/zhangskd/article/details/17282895>

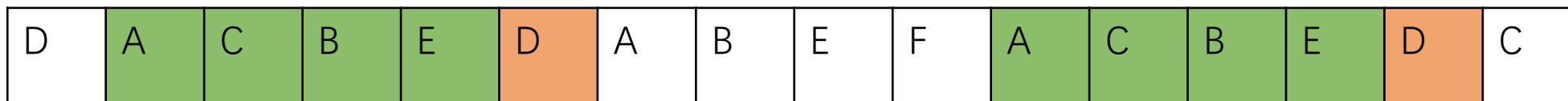
说实在我找到关于LZ4的文章讲的很简略，基本就在扔代码，所以我就是了解了一下主要思想就去看哈希表怎么写了

LZ4与LZ77的不同

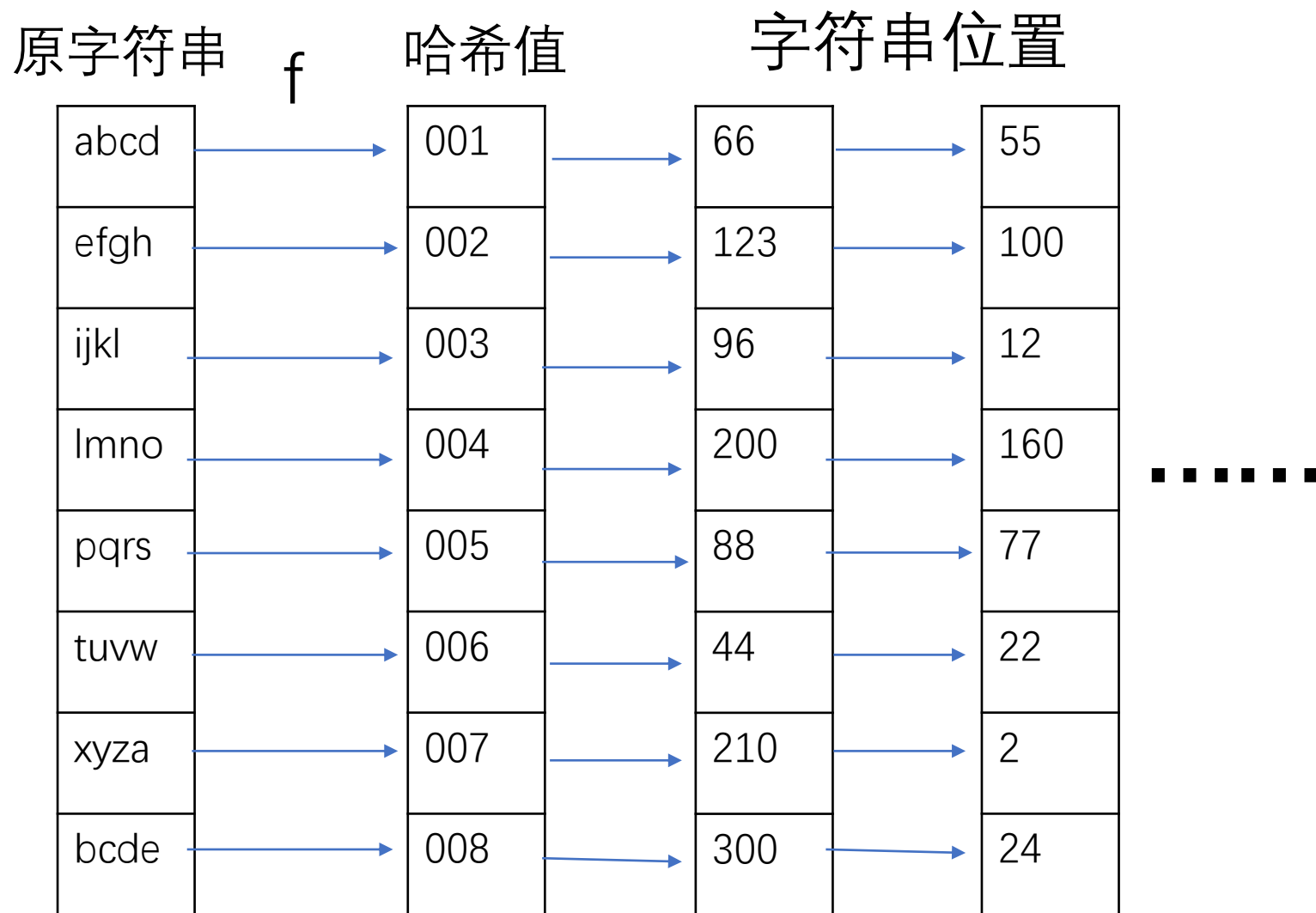
在匹配字符串时，不再将整个前向缓冲区作为模式串，
仅仅读入四个字符来匹配。但是滑动窗口还是保留了

读入四个字符后，就利用哈希表搜索这四个字符曾经出现的位置
如果存在，则贪心地往后匹配延伸，看看匹配长度能不能比4更大
如果不存在，跳过

下图是个简要示例图，绿色是四字符成功匹配，然后往后又有一个
字符相同，那匹配长度就是5了



哈希表



这里的f算出的哈希值
只是举例子，不代表真正实现结果

由于是可以通过哈希值直接取得字符串位置，理论上能在常数时间内完成

具体哈希表的实现，可以使用STL下的
unordered_map，就比较方便

或者自己手动来实现，手动实现时就用两个数组，
一个数组的下标是哈希值，储存字符串出现位置，
另一个数组就与之配合储存链表，从而实现一个哈希值跟着一长串**字符串位置**

第一个数组的长度要根据哈希值最大的大小确定
第二个数据的长度就是滑动窗口的大小

LZ4 Sequence

Token : ==> 4-high-bits : literal length / 4-low-bits : match length

Token	<i>Literal length+ (optional)</i>	Literals	Offset	<i>Match length+ (optional)</i>
1-byte	0-n bytes	0-L bytes	2-bytes (little endian)	0-n bytes

这是LZ4中规定的一个压缩后的单位数据结构。

- 1、**Token** 高四位储存字面值长度，低四位储存匹配长度，共一字节
这意味着这里储存的长度最多只有15
- 2、**扩展的字面值长度**，可选。如果Token处不够放，放到这就行了
- 3、**字面的字符**，就是没被压缩的字符，都是原字符
- 4、**偏移量**，匹配的字符串到现在位置的距离
- 5、**扩展的匹配长度**，可选。如果Token处不够放，放到这就行了

看到这个基本的结构，我们看到最小的单位都是字节，没有像LZ77需要额外添加一比特来标记，从而也不需要将字符转为二进制，再将二进制转换为字符。避免之前说到的频繁转换二进制。

那我们在编码数据的时候

- 1、如果字面值长度**大于等于15**，Token高四位全1；对匹配长度同理。
如果是小于15的，可用 **或运算** 将其加到高四位；然后对于低四位，用**加法**就行了
- 2、如果是有扩展长度，一个字节存**255**，直到存完长度为止
- 3、字面的字符，直接复制到压缩输出缓冲区即可
- 4、距离，写入即可
- 5、扩展的匹配长度与步骤2类似

加快速度的办法

我认为，每次只选择读四个字符，可以将unsigned char* 强行转换为 unsigned int* 就可以一次性读取完四个字符

前面提到的复制字符，其实也不必一个字符一个字符地复制，通过上述强转的办法可以四个字节四个字节复制，即便不是4的倍数也不用担心，稍微多复制一点没问题，只要当前位置指针移动正确就可以了，多出的字符自然后续会被覆盖

LZ4算法介绍:

- 1、设一变量储存**当前位置**；设一变量作**锚点**，锚点是储存成功匹配后的第一个字符的位置
- 2、如果最大的可能匹配长度**大于等于4**，转步骤3；否则转步骤6
- 3、用哈希表获取当前四字符出现的上一位置
往后**逐一**对比字符，看看匹配长度最长能多长
用哈希表**再往前**获取四字符出现的位置，循环本步骤，直到超出滑动窗口范围
- 4、如果最后最长长度**大于等于4**，编码压缩后的数据，放入缓冲区，再逐一将**每一个四字符**放入哈希表，当前位置跳到匹配后位置，锚点也如此。
若最长长度小于4，转步骤5
- 5、当前位置后移一格，锚点不变，当前的四字符放入哈希表
- 6、如果最后还有剩下未匹配的字符，全部作为原字符存入即可

循环着做，直到没得压缩

从结果来看，这个算法还算比较优秀的

Windows PowerShell

```
PS E:\GradeOnePrograme\HomeWork\SimpleCompression\Debug> .\SimpleCompression.exe c test.log out.log
压缩率: 5.02592%
消耗时间: 89ms
PS E:\GradeOnePrograme\HomeWork\SimpleCompression\Debug> .\SimpleCompression.exe d out.log test1.log
消耗时间: 15ms
PS E:\GradeOnePrograme\HomeWork\SimpleCompression\Debug>
```

out	2020/4/26 15:18	文本文档	507 KB
test	2020/4/16 16:35	文本文档	10,073 KB
test1	2020/4/26 15:20	文本文档	10,073 KB

谢谢