

关于 LZ77 压缩算法

在看木马代码的时候，涉及到一个 lz77 无损压缩算法的问题，上网搜了好多资料，都没找到特别好的。感觉这篇写的还算完整，贴出来分享给大家。关于该算法的资料来源与网络，版权归原作者所有，如果侵权，请及时告知。之所以这样说，是笔者听说在 LZ 系列算法中还有一部分压缩算法有专利，另一方面也是为了尊重知识产权。

以下内容来自互联网：

全新的思路

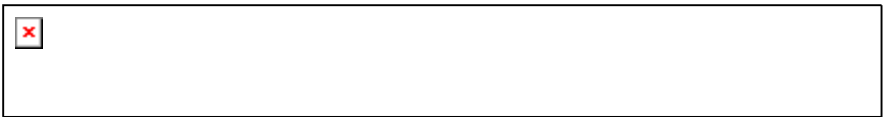
我们过去讨论的压缩模型都是基于对信息中单个字符出现频率的统计而设计的，直到 70 年代末期，这种思路在数据压缩领域一直占据着统治地位。在我们今天看来，这种情形在某种程度上显得有些可笑，但事情就是这样，一旦某项技术在某一领域形成了惯例，人们就很难创造出在思路上与其大相径庭的哪怕是更简单更实用的技术来。

我们敬佩那两个在数据压缩领域做出了杰出贡献的以色列人，因为正是他们打破了 Huffman 编码一统天下的格局，带给了我们既高效又简便的“字典模型”。至今，几乎我们日常使用的所有通用压缩工具，象 ARJ, PKZip, WinZip, LHArc, RAR, GZip, ACE, ZOO, TurboZip, Compress, JAR……甚至许多硬件如网络设备中内置的压缩算法，无一例外，都可以最终归结为这两个以色列人的杰出贡献。

说起来，字典模型的思路相当简单，我们日常生活中就经常在使用这种压缩思想。我们常常跟人谈“奥运会”、“IBM”、“TCP”之类的词汇，说者和听者都明白它们指的是“奥林匹克运动会”、“国际商业机器公司”和“传输控制协议”，这实际就是信息的压缩。我们之所以可以顺利使用这种压缩方式而不产生语义上的误解，是因为在说者和听者的心中都有一个事先定义好的缩略语字典，我们在对信息进行压缩（说）和解压缩（听）的过程中都对字典进行了查询操作。字典压缩模型正是基于这一思路设计实现的。

最简单的情况是，我们拥有一本预先定义好的字典。例如，我们要对一篇中文文章进行压缩，我们手中已经有一本《现代汉语词典》。那么，我们扫描要压缩的文章，并对其中的句子进行分词操作，对每一个独立的词语，我们在《现代汉语词典》查找它的出现位置，如果找到，我们就输出页码和该词在该页中的序号，如果没有找到，我们就输出一个新词。这就是静态字典模型的基本算法了。

你一定可以发现，静态字典模型并不是好的选择。首先，静态模型的适应性不强，我们必须为每类不同的信息建立不同的字典；其次，对静态模型，我们必须维护信息量并不算小的字典，这一额外的信息量影响了最终的压缩效果。所以，几乎所有通用的字典模型都使用了自适应的方式，也就是说，将已经编码过的信息作为字典，如果要编码的字符串曾经出现过，就输出该字符串的出现位置及长度，否则输出新的字符串。根据这一思路，你能从下面这幅图中读出其中包含的原始信息吗？



啊，对了，是“吃葡萄不吐葡萄皮，不吃葡萄倒吐葡萄皮”。现在你该大致明白自适应字典模型的梗概了吧。好了，下面就让我们来深入学习字典模型的第一类实现——LZ77 算法。

滑动的窗口

LZ77 算法在某种意义上又可以称为“滑动窗口压缩”，这是由于该算法将一个虚拟的，可以跟随压缩进程滑动的窗口作为术语字典，要压缩的字符串如果在该窗口中出现，则输出

其出现位置和长度。使用固定大小窗口进行术语匹配,而不是在所有已经编码的信息中匹配,是因为匹配算法的时间消耗往往很多,必须限制字典的大小才能保证算法的效率;随着压缩的进程滑动字典窗口,使其中总包含最近编码过的信息,是因为对大多数信息而言,要编码的字符串往往在最近的上下文中更容易找到匹配串。

参照下图,让我们熟悉一下 LZ77 算法的基本流程。



- 1、从当前压缩位置开始,考察未编码的数据,并试图在滑动窗口中找出最长的匹配字符串,如果找到,则进行步骤 2,否则进行步骤 3。
- 2、输出三元符号组 (off, len, c) 。其中 off 为窗口中匹配字符串相对窗口边界的偏移, len 为可匹配的长度, c 为下一个字符。然后将窗口向后滑动 $len + 1$ 个字符,继续步骤 1。
- 3、输出三元符号组 $(0, 0, c)$ 。其中 c 为下一个字符。然后将窗口向后滑动 $len + 1$ 个字符,继续步骤 1。

我们结合实例来说明。假设窗口的大小为 10 个字符,我们刚编码过的 10 个字符是: `abddbcca`,即将编码的字符为: `abacaaabae`

我们首先发现,可以和要编码字符串匹配的最长串为 `ab` ($off = 0, len = 2$), `ab` 的下一个字符为 `a`,我们输出三元组: $(0, 2, a)$

现在窗口向后滑动 3 个字符,窗口中的内容为: `dbbccaaba`

下一个字符 `e` 在窗口中没有匹配,我们输出三元组: $(0, 0, e)$

窗口向后滑动 1 个字符,其中内容变为: `bbccaabae`

我们马上发现,要编码的 `aaabae` 在窗口中存在($off = 4, len = 6$),其后的字符为 `e`,我们可以输出: $(4, 6, e)$

这样,我们将可以匹配的字符串都变成了指向窗口内的指针,并由此完成了对上述数据的压缩。

解压缩的过程十分简单,只要我们向压缩时那样维护好滑动的窗口,随着三元组的不断输入,我们在窗口中找到相应的匹配串,缀上后继字符 c 输出(如果 off 和 len 都为 0 则只输出后继字符 c)即可还原出原始数据。

当然,真正实现 LZ77 算法时还有许多复杂的问题需要解决,下面我们就来对可能碰到的问题逐一加以探讨。

编码方法

我们必须精心设计三元组中每个分量的表示方法,才能达到较好的压缩效果。一般来讲,编码的设计要根据待编码的数值的分布情况而定。对于三元组的第一个分量——窗口内的偏移,通常的经验是,偏移接近窗口尾部的情况要多于接近窗口头部的情况,这是因为字符串在与其接近的位置较容易找到匹配串,但对于普通的窗口大小(例如 4096 字节)来说,偏移值基本还是均匀分布的,我们完全可以用固定的位数来表示它。

编码 off 需要的位数 $bitnum = upper_bound(\log_2(MAX_WND_SIZE))$

由此,如果窗口大小为 4096,用 12 位就可以对偏移编码。如果窗口大小为 2048,用 11 位就可以了。复杂一点的程序考虑到在压缩开始时,窗口大小并没有达到 `MAX_WND_SIZE`,而是随着压缩的进行增长,因此可以根据窗口的当前大小动态计算所需

要的位数，这样可以略微节省一点空间。

对于第二个分量——字符串长度，我们必须考虑到，它在大多数时候不会太大，少数情况下才会发生大字符串的匹配。显然可以使用一种变长的编码方式来表示该长度值。在前面我们已经知道，要输出变长的编码，该编码必须满足前缀编码的条件。其实 Huffman 编码也可以在此处使用，但却不是最好的选择。适用于此处的好的编码方案很多，我在这里介绍其中两种应用非常广泛的编码。

第一种叫 Golomb 编码。假设对正整数 x 进行 Golomb 编码，选择参数 m ，令

$$b = 2m$$

$$q = \text{INT}((x - 1)/b)$$

$$r = x - qb - 1$$

则 x 可以被编码为两部分，第一部分是由 q 个 1 加 1 个 0 组成，第二部分为 m 位二进制数，其值为 r 。我们将 $m = 0, 1, 2, 3$ 时的 Golomb 编码表列出：

值 x	$m = 0$	$m = 1$	$m = 2$	$m = 3$
1	0	0 0	0 00	0 000
2	10	0 1	0 01	0 001
3	110	10 0	0 10	0 010
4	1110	10 1	0 11	0 011
5	11110	110 0	10 00	0 100
6	111110	110 1	10 01	0 101
7	1111110	1110 0	10 10	0 110
8	11111110	1110 1	10 11	0 111
9	111111110	11110 0	110 00	10 000

从表中我们可以看出，Golomb 编码不但符合前缀编码的规律，而且可以用较少的位表示较小的 x 值，而用较长的位表示较大的 x 值。这样，如果 x 的取值倾向于比较小的数值时，Golomb 编码就可以有效地节省空间。当然，根据 x 的分布规律不同，我们可以选取不同的 m 值以达到最好的压缩效果。

对我们上面讨论的三元组 len 值，我们可以采用 Golomb 方式编码。上面的讨论中 len 可能取 0，我们只需用 $len + 1$ 的 Golomb 编码即可。至于参数 m 的选择，一般经验是取 3 或 4 即可。

可以考虑的另一种变长前缀编码叫做 γ 编码。它也分作前后两个部分，假设对 x 编码，令 $q = \text{int}(\log_2 x)$ ，则编码的前一部分是 q 个 1 加一个 0，后一部分是 q 位长的二进制数，其值等于 $x - 2^q$ 。 γ 编码表如下：

值 x	γ 编码
1	0
2	10 0
3	10 1
4	110 00
5	110 01
6	110 10
7	110 11
8	1110 000
9	1110 001

其实，如果对 `off` 值考虑其倾向于窗口后部的规律，我们也可以采用变长的编码方法。但这种方式对窗口较小的情况改善并不明显，有时压缩效果还不如固定长编码。

对三元组的最后一个分量——字符 `c`，因为其分布并无规律可循，我们只能老老实实地用 8 个二进制位对其编码。

根据上面的叙述，相信你一定能写出高效的编码和解码程序了。

另一种输出方式

LZ77 的原始算法采用三元组输出每一个匹配串及其后续字符，即使没有匹配，我们仍然需要输出一个 `len = 0` 的三元组来表示单个字符。试验表明，这种方式对于某些特殊情况（例如同一字符不断重复的情形）有着较好的适应能力。但对于一般数据，我们还可以设计出另外一种更为有效的输出方式：将匹配串和不能匹配的单个字符分别编码、分别输出，输出匹配串时不同时输出后续字符。

我们将每一个输出分成匹配串和单个字符两种类型，并首先输出一个二进制位对其加以区分。例如，输出 0 表示下面是一个匹配串，输出 1 表示下面是一个单个字符。

之后，如果要输出的是单个字符，我们直接输出该字符的字节值，这要用 8 个二进制位。也就是说，我们输出一个单个的字符共需要 9 个二进制位。

如果要输出的是匹配串，我们按照前面的方法依次输出 `off` 和 `len`。对 `off`，我们可以输出定长编码，也可以输出变长前缀码，对 `len` 我们输出变长前缀码。有时候我们可以对匹配长度加以限制，例如，我们可以限制最少匹配 3 个字符。因为，对于 2 个字符的匹配串，我们使用匹配串的方式输出并不一定比我们直接输出 2 个单个字符（需要 18 位）节省空间（是否节省取决于我们采用何种编码输出 `off` 和 `len`）。

这种输出方式的优点是输出单个字符的时候比较节省空间。另外，因为不强求每次都带一个后续字符，可以适应一些较长匹配的情况。

如何查找匹配串

在滑动窗口中查找最长的匹配串，大概是 LZ77 算法中的核心问题。容易知道，LZ77 算法中空间和时间的消耗集中于对匹配串的查找算法。每次滑动窗口之后，都要进行下一个匹配串的查找，如果查找算法的时间效率在 $O(n^2)$ 或者更高，总的算法时间效率就将达到 $O(n^3)$ ，这是我们无法容忍的。正常的顺序匹配算法显然无法满足我们的要求。事实上，我们有以下几种可选的方案。

- 1、限制可匹配字符串的最大长度（例如 20 个字节），将窗口中每一个 20 字节长的串抽取出来，按照大小顺序组织成二叉有序树。在这样的二叉有序树中进行字符串的查找，其效率是很高的。树中每一个节点大小是 $20(\text{key}) + 4(\text{off}) + 4(\text{left child}) + 4(\text{right child}) = 32$ 。树中共有 $\text{MAX_WND_SIZE} - 19$ 个节点，假如窗口大小为 4096 字节，树的大小大约是 130k 字节。空间消耗也不算多。这种方法对匹配串长度的限制虽然影响了压缩程序对一些特殊数据（又很长的匹配串）的压缩效果，但就平均性能而言，压缩效果还是不错的。

- 2、将窗口中每个长度为 3（视情况也可取 2 或 4）的字符串建立索引，先在此索引中匹配，之后对得出的每个可匹配位置进行顺序查找，直到找到最长匹配字符串。因为长度为 3 的字符串可以有 256³ 种情况，我们不可能用静态数组存储该索引结构。使用 Hash 表是一个明智的选择。我们可以仅用 $\text{MAX_WND_SIZE} - 1$ 的数组存储每个索引点，Hash 函数的参数当然是字符串本身的 3 个字符值了，Hash 函数算法及 Hash 之后的散列函数很容易设计。每个索引点之后是该字符串出现的所有位置，我们可以使用单链表来存储每一个位置。值得注意的是，对一些特殊情况比如 `aaaaaa...` 之类的连续字符串，字符串 `aaa` 有很


```
        return (_MAX_WINDOW_SIZE - (nLeftOff - off));  
    }
```

这样，解码程序无需考虑环形偏移系统就可以顺利高速解码了。

资源

结合上面的讨论，典型的 LZ77 算法应当不难实现，我们本章给出的源码是一个较为特殊的实现。

示例程序 `lz77.exe` 使用对匹配串和单个字符分类输出的模型，输出匹配串时，`off` 采用定长编码，`len` 采用 γ 编码。索引结构采用 2 字节长字符串的索引，使用 $256 * 256$ 大小的静态数组存储索引点，每个索引点指向一个位置链表。链表节点考虑了对 `aaaaa...` 之类的重复串的优化。

示例程序的独特之处在于使用了 64k 大小的固定长度窗口，窗口不做滑动（因此不需要环形偏移系统，也节省了删除索引点的时间）。压缩函数每次只对最多 64k 长的数据进行压缩，主函数将原始文件分成 64k 大小的块逐个压缩存储。使用这种方法首先可以增大匹配的概率，字符串可以在 64k 空间内任意寻找最大匹配串，以此提高压缩效率。其次，这种方法有利于实现解压缩的同步。也就是说，利用这种方法分块压缩的数据，很容易从原始文件中间的任何一个位置开始解压缩，这尤其适用于全文检索系统中全文信息的保存和随机读取。

结合上述示例程序，王笨笨开发了可压缩多个文件并可同步（随机）解压缩的文件级接口，但此接口并非自由代码(free code)。如果需要可以和王笨笨联系。