

一种串匹配的快速 Boyer-Moore 算法*

李雪梅¹, 代六玲², 童新海¹, 李 莉¹

(1. 北京电子科技学院 电子信息工程系, 北京 100070; 2. 南京理工大学 计算机科学系, 江苏 南京 210094)

摘 要: 在对经典的 Boyer-Moore 和 Quick Search 串匹配算法进行分析的基础上, 提出了一种更加快速的串匹配算法 Quick Boyer-Moore(QBM)。QBM 算法利用当前尝试中的已匹配子串、匹配失败字符信息以及当前窗口下一个字符的位置信息, 以在每一次跳跃中获得更大的跳跃距离, 从而使算法具有更高的效率。在真实语料上的实验结果表明, QBM 算法的效率较显著地高于原始的 BM 算法及其改进算法 Improved Boyer-Moore(IBM)。
关键词: 串匹配; Boyer-Moore 算法; Improved Boyer-Moore 算法; Quick Boyer-Moore 算法
中图法分类号: TP18 **文献标识码:** A **文章编号:** 1001-3695(2005) 09-0049-03

Quick Boyer-Moore Algorithm for String Matching

LI Xue-mei¹, DAI Liu-ling², TONG Xin-hai¹, LI Li¹

(1. Dept. of Electronic Information & Engineering, Beijing Electronic Science & Technology Institute, Beijing 100070, China; 2. Dept. of Computer Science, Nanjing University Science & Technology, Nanjing Jiangsu 210094, China)

Abstract: This paper suggests a very efficient algorithm for string matching, Quick Boyer-Moore(QBM) algorithm, based on the ideas of the Boyer-Moore(BM) algorithm and the Quick Search(QS) algorithm. Besides the match and mismatch information inside the current window used by the Boyer-Moore algorithm, QBM also uses the information carried by the character immediately after the current window. The good-suffix shift distance of QBM is larger than that of BM algorithm in most circumstances. The tests on actual corpus show that QBM is more efficient than BM and the Improved Boyer-Moore(IBM) algorithm.
Key words: String Matching; Boyer-Moore Algorithm; Improved Boyer-Moore Algorithm; Quick Boyer-Moore Algorithm

串匹配是计算机技术领域的一个基本问题。在文档编辑、内容管理、计算机病毒检测、DNA 序列检测以及入侵检测等多个应用领域中都有重要应用。快速的串匹配算法一直都是一个研究热点, 已经有多种基于不同思想的算法被提出。

在本文中, 我们使用 $x = x[0..m-1]$ 表示要匹配的模式串(关键词), 长度为 m 。使用 $y = y[0..n-1]$ 表示正文文本, 长度为 n 。字母表以 Σ 表示, 大小为 σ 。串匹配的任务就是要在给定的 y 中发现所有的 x , 并报告所有 x 出现的位置。将匹配过程中 x 与 y 中长度为 m 的子串的一次比较称为一次尝试, 并将当前尝试中与关键词对齐的 y 的子串称为当前窗口。

在实际应用中, 通常认为 Boyer-Moore (BM)^[1] 和 Quick Search(QS)^[2] 算法具有最高的效率^[3]。BM 采用从后向前的比较方式, 并利用当前尝试中的已匹配信息和匹配失败的字符, 查找预处理好的良好后缀跳跃表(Good Suffix Shift Table) 和不良字符跳跃表(Bad Character Shift Table), 使尝试跳跃式的进行, 最大可能跳跃距离为 m 。QS 对匹配顺序没有要求, 且只使用正文中位于当前窗口的下一个字符查找预先生成的不良字符跳跃表, 最大可能的跳跃距离为 $m+1$ 。贺龙涛等人提出的改进 BM 算法(Improved Boyer-Moore, IBM)^[4] 在 BM 算法的基础上, 利用不匹配位置和已匹配字符的相邻位置关系, 只使用一个良好后缀跳跃表, 使得算法的效率得到很大提高。本文提出的快速 BM 算法(Quick Boyer-Moore, QBM) 除了利用当

收稿日期: 2004-09-04; 修返日期: 2004-10-10
基金项目: 国家自然科学基金资助项目(60272088)

前窗口中已匹配字符信息和匹配失败位置信息外, 还充分利用正文中与当前窗口近邻的后一个字符带来的信息, 生成改进的良好后缀跳跃表, 以期在每一次跳跃中跳跃尽量大的距离。在真实语料上的实验结果表明, QBM 算法对匹配效率的提高是明显的。

1 Boyer-Moore 和 Quick Search 算法分析

Boyer-Moore 算法^[1] 由 R. S. Boyer 和 J. S. Moore 设计实现的一个经典单关键词匹配算法, 到目前仍是应用中最为常用、效率较高的单模式串匹配算法之一。BM 算法在当前窗口中从后向前进行匹配, 直到找到匹配失败的位置或发现成功的匹配, 并根据匹配的情况查找良好后缀跳跃表和不良字符跳跃表来使窗口位置向后跳跃。

Boyer-Moore 算法如图 1 所示。假设正文中当前窗口的位置为 j , 匹配失败发生在模式串字符 $x[i] = a$ 和正文字符 $y[i+j] = b$ 之间, 即 $x[i+1..m-1] = y[i+j+1..j+m-1] = u$ 且 $x[i] \neq y[i+j]$ 。良好后缀跳跃将 x 中最右的前导字符不为 a 的子串 u 与 y 当前窗口中的子串 u 对齐(Case 1)。如果在 x 中不存在这样的子串, 良好后缀跳跃将 x 的前缀与 u 的后缀中最长的相同子串 v 对齐(Case 2), 良好后缀转移表的大小为 m 。

不良字符跳跃是将 $x[0..m-2]$ 中最右的字符 b 与 y 中匹配失败位置 $y[i+j] = b$ 对齐(Case 3)。如果 b 不出现在 x 中, 则跳跃距离为模式长度 $m - |u|$ (Case 4)。BM 算法采用不良字符跳跃和良好后缀跳跃中的较大者作为跳跃距离。

贺龙涛等人^[4]指出,利用 y 中匹配失败字符 $y[i+j] = b$ 与后缀 u 的相邻关系,可以在当前尝试匹配失败后获得更大的跳跃距离。改进后的 IBM 算法只使用一个良好后缀转移表,工作过程与 BM 算法基本相同,只是不再区分好后缀与坏字符,而是将它们一起考虑。对于在位置 j 的尝试,当得到 $a = x[i] \neq y[i+j] = b$ 时,IBM 在 $x[0 \dots m-2]$ 中从后向前查找最右边的子串 au ,并通过跳跃使 x 中的子串 au 和 y 中的子串 au 对齐。如果 x 中没有这样的子串,则与 BM 算法相同,查找 x 的前缀和后缀中最长的相同子串 v ,并通过跳跃使之对齐。对于同样的窗口,IBM 的跳跃距离不会小于 BM 的跳跃距离。由于 IBM 只使用一个良好后缀跳跃表,在每次尝试失败后只比 BM 少一次访存操作和一次比较操作,使得 IBM 比 BM 的效率更高。

Quick Search 算法是一个非常精巧的算法,只使用一个不良字符转移表而不使用良好后缀转移表,从而只有和字符表 Σ 大小相同的内存需求。设正文中当前窗口为 $y[j \dots j+m-1]$,由于在当前窗口匹配完成之后跳跃的距离至少为 1,字符 $y[j+m]$ 被考虑是非常合理的。QS 算法利用字符 $y[j+m]$ 来生成不良字符跳跃表,最大可能的跳跃距离为 $m+1$ 。在模式串较短或者相对于字符表模式串中出现的字符较少的情况下, QS 算法在速度上超过 BM 算法^[3]。

2 Quick Boyer-Moore 算法

进一步提高串匹配算法的效率的主要途径是利用当前尝试中可以获取的信息以进一步的增大跳跃距离。Quick Search 算法指出,在当前尝试之后,不管匹配成功与否,当前窗口的后一个字符 $y[j+m]$ 都能为跳跃距离的确定带来信息。本节提出的 Quick Boyer-Moore 算法考虑字符 $y[j+m]$ 与当前窗口的位置相邻关系,将部分匹配的后缀 u 和字符 $y[j+m]$ 作为一个整体考虑,以期获得更大的跳跃距离。

2.1 算法描述

为方便描述,设 $y[j+m] = t$ 。在当前尝试结束后,如果进行跳跃以后 u 仍在下一个窗口中且匹配成功,则子串 ut 也必定在匹配窗口中。或者如果跳跃后 u 的某一长度的后缀 v 仍然在下一个窗口中且匹配成功,则子串 vt 也必定在该匹配窗口中。我们将子串 vt 称为串 xt 的边界,因为它同时出现在 xt 的首端和末端。

在匹配过程中,在当前尝试结束后, QBM 在 $x[0 \dots m-1]$ 中自后向前地寻找最右的子串 ut ,并将该子串和 y 中的子串 ut 对齐,如图 2 所示。

如果在 $x[0 \dots m-1]$ 中不存在这样的子串,则在 $x[0 \dots m-2]$ 中寻找最长的前缀,使之与 xt 的后缀 vt 相同,并使之对齐,获得跳跃,如图 3 所示。

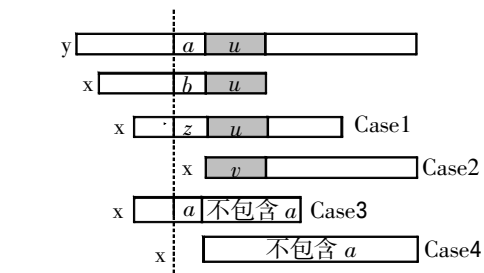


图 1 BM 算法示意图

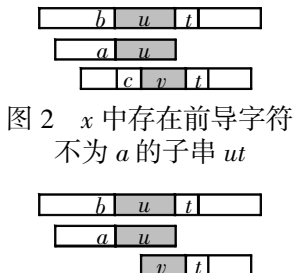


图 2 x 中存在前导字符不为 a 的子串 ut

在 x 中查找最右的前导字符不为 a 的子串 ut 或 vt 在大多数情况下会比在 x 中查找前导字符不为 a 的子串 u 或 v 获得更大的跳跃距离。最好的情况下,在 x 中既不存在子串 ut ,也不存在任意长度的边界 vt ,此时的安全跳跃距离为 $m+1$ 。

与 BM 算法相似,还可以使用坏字符跳跃来保证最大的跳跃距离,并选择坏字符跳跃距离和良好后缀跳跃距离的较大者作为安全跳跃距离。本文中将同时使用坏字符跳跃策略和良好后缀跳跃策略的 QBM 算法记为 QBM1,并测试其效率。

2.2 预处理和匹配过程

QBM 算法只使用一个跳跃表,即良好后缀跳跃表 $qbmGs$ 。该表在预处理阶段被生成。与 BM 算法相比, QBM 算法的良好后缀转移表 $qbmGs$ 的生成多使用了一个字符的信息。在匹配的时候该字符出现在当前尝试窗口的下一个位置。为使 $qbmGs$ 表的生成过程更加直观,可以假设模式串 x 的长度为 $m+1$ 。模式 x 的最后一个字符 $x[m]$ 的可能取值为字符表 Σ 中的任意字符,在匹配过程中由当前尝试窗口的紧邻下一字符决定并与之相同。每次跳跃的跳跃距离由 $x[m]$ 的取值和匹配失败的位置共同决定。

设字符表的大小为 σ 。因为匹配失败的可能位置个数为 m (从 0 到 $m-1$),从而 $qbmGs$ 的大小为 $m\sigma$ 。借助于 BM 算法的预处理函数 $preBmGs$, QBM 使用如下 C 代码完成 $qbmGs$ 的预处理:

```
qbmGs = new int[ ASIZE * m]
for ( unsigned int sigma = 0; sigma < ASIZE; sigma ++ ) {
    x[ m] = sigma;
    preBmGs( x, m + 1, qbmGs + sigma * m );
}
```

其中 ASIZE 等于字符表的大小 σ , $preBmGs$ 为 BM 算法中初始化良好后缀跳跃表的函数。 $preBmGs$ 接收三个参数,即关键词、关键词长度和良好后缀表地址(长度为 m)。

对于 QBM1 算法,除了使用 $qbmGs$ 表之外,还使用一个不良字符转移表,该表与 BM 算法中的不良字符转移表完全相同。即对于字符表 Σ 中的任意字符 c :

$$bmBc[c] = \begin{cases} \min\{i | i < m-1 \text{ 且 } x[m-1-i] = c\}, & \text{如果 } c \text{ 出现在 } x \text{ 中} \\ m, & \text{如果 } c \text{ 不出现在 } x \text{ 中} \end{cases}$$

在匹配阶段,对于当前尝试,如果匹配成功,报告一次成功匹配,并根据当前窗口的紧邻下一个字符从 $qbmGs$ 中取得跳跃距离。如果当前尝试失败, QBM 算法根据匹配失败的位置和当前窗口的紧邻下一个字符一起从 $qbmGs$ 中取得良好后缀跳跃距离。 QBM1 则还根据正文中匹配失败位置的字符从 $bmBc$ 中获取不良字符跳跃距离,并取这两个跳跃距离的较大者作为实际的跳跃距离。

匹配阶段的 C 代码如下, QBM 和 QBM1 算法的唯一差别在于当前尝试失败后计算跳跃距离的方式:

```
while ( j <= n - m )
{
    for ( i = m - 1; i >= 0 && x[i] == y[i+j]; --i );
    if ( i < 0 ) // 匹配成功
    {
        j += qbmGs[ y[j+m] ];
    }
    else // 匹配失败
    {
        j += qbmGs[ y[j+m] * m + i ]; // for QBM1
    }
}
```

```
        j + = max( qbmGs[ y[ j + m] * m + i] , bmBc[ y[ i + j] ] - m +
1 + i); // for QBM1
    }
}
```

3 算法分析

QBM 算法对空间的需求与 IBM 相同, 比 BM 算法高。由于良好后缀跳跃表大小为 $m\sigma$, 算法整体空间复杂度为 $O(m\sigma)$ 。不良后缀跳跃表大小为 σ , QBM1 算法整体空间复杂度为 $O((m+1)\sigma)$ 。在预处理阶段, 生成良好后缀跳跃表的时间复杂度为 $O(\sigma)$, QBM 生成良好后缀跳跃表的时间复杂度为 BM 算法生成良好后缀跳跃表的 σ 倍, 为 $O(m\sigma)$ 。QBM1 预处理阶段的时间复杂度为 $O((m+1)\sigma)$ 。

在匹配阶段, QBM 及 QBM1 在最理想情况下的时间复杂度优于 BM 及 IBM 算法, 为 $O(n/(m+1))$ 。在最坏情况下 QBM 及 QBM1 算法的时间复杂度和 BM 算法相等, 为 $O(m(n-m))$ 。

4 实验结果及分析

4.1 实验结果

算法的实际运行情况比较复杂, 在真实语料上的测试数据将更能说明各种算法的效率。分别在一个中文语料和一个英文语料上对各算法的效率进行对比测试。中文测试语料主要从各个政府门户网站上获得, 包含 2 054 个网页, 总量约为 60MB。网页已经成为串匹配的主要处理对象, 而且网页中既包含中文文本, 又包含英文文本(通常是 HTML 等), 是很好的串匹配算法测试语料。英文测试语料为 reuters 21 578^[5]。该语料通常被用来作为文本分类的测试语料, 共包含 21 578 篇文章, 总量为 27MB, 各文章之间用 SGML 语言标记。

我们分别以 BM 和 IBM 算法为对照, 对 QBM 和 QBM1 算法的效率进行测试。对于串匹配算法来说, 效率的最好体现是 CPU 运行时间。由于四个算法工作模式相似, 所以总的尝试次数和总的字符比较次数也是算法效率的很好体现。在中文语料中, 对中文的单字、双字、三字、四字和五字以上的每一长度分别取不同的 10 个关键词进行实验。在英文语料和中文语料上, 则分别取长度为 2~11 的关键词各五个进行实验。分别记录所有关键字在语料中的总出现次数, 消耗的 CPU 时间, 总尝试次数和总字符比较次数。实验环境为 Athlon 2500 + CPU, 512MB 内存, Windows XP。编译环境为 Microsoft Visual C++ 6.0, 算法所耗 CPU 周期使用 C 库函数 clock() 得到, 不包含 I/O 时间。表 1 和表 2 给出实验数据。

表 1 中文语料上的实验结果

算法	关键词出现次数	CPU 时间	尝试次数	字符比较次数
BM	284 811	23 781	713 043 297	715 615 671
IBM	284 811	21 452	713 039 964	715 611 222
QBM	284 811	18 718	548 675 073	551 947 816
QBM1	284 811	21 781	546 765 233	548 960 164

表 2 英文语料上的实验结果

算法	关键词出现次数	CPU 时间	尝试次数	字符比较次数
BM	37 611	7 002	162 603 833	168 445 929
IBM	37 611	5 387	162 573 294	168 413 953
QBM	37 611	4 816	136 103 019	144 618 742
QBM1	37 611	6 430	130 561 674	135 449 764

4.2 结果分析

从表 1 和表 2 中可以看出, 对于尝试次数和字符比较次数, 具有关系 $BM > IBM > QBM > QBM1$ 。QBM 在中文语料上的尝试次数为 IBM 的 76.9%, 字符比较次数为 IBM 的 77.1%。在英文语料上的尝试次数为 IBM 的 83.7%, 字符比较次数为 IBM 的 85.9%。而对于 CPU 时间, 则具有关系 $BM > QBM1 > IBM > QBM$ 。QBM 在中文语料上的运行时间为 BM 的 78.7%, 为 IBM 的 87.3%。在英文语料上的运行时间为 BM 的 68.8%, 为 IBM 的 89.4%。QBM 对效率的提高是明显的。

实验结果中一个有趣的现象是, 虽然 QBM1 算法的尝试次数和字符比较次数在所有算法中最少, 但是其实际运行效率却并不高, 其消耗的 CPU 时间甚至高于 IBM 算法。而尝试次数和字符比较次数与 BM 算法相当的 IBM 算法却显著地减少了 BM 算法的运行时间。对于此, 我们给出如下解释:

就串匹配算法来讲, 提高算法运行效率的途径除了充分利用已知信息来尽力提高跳跃距离外, 还应尽力减少匹配过程中生成跳跃距离的复杂度。除此之外, 还应考虑计算机在运行算法时的 Cache 命中率。QBM1 使用不良字符跳跃表以保证其跳跃距离不小于 QBM。但是 QBM1 在当前尝试失败后生成跳跃距离要比 QBM 算法多一次访存操作和一次比较操作, 以及几次加法操作, 其复杂度大于 QBM。而 IBM 算法虽然对尝试次数和字符比较次数的减少并不明显, 但是其生成跳跃距离的复杂度低于 BM 和 QBM1。这导致 QBM1 的运行效率甚至不如 IBM。而 QBM1 的运行效率相对与 BM 也没有显著提高, 这是因为 QBM1 使用的内存要比 BM 大很多(BM 的空间复杂度为 $O(m + \sigma)$), 导致运行时的 Cache 命中率大大降低, 从而影响了运行效率。

5 结论

本文提出的 QBM 串匹配算法在计算良好后缀跳跃表的时候使用了比 BM 算法更多的信息。利用与当前窗口紧邻的后一个字符带来的信息使得在绝大多数情况下获得比 BM 更大的跳跃距离。实验结果表明, QBM 算法对匹配效率的提高是显著的。在实际语料上的实验结果还说明要提高串匹配算法的效率, 应该综合考虑跳跃距离、生成跳跃距离的复杂度以及 Cache 命中率。这为今后串匹配算法的设计提供了指导。今后的研究包括将 QBM 的思想用于多字符串匹配等。

参考文献:

[1] S Boyer, J S Moore. A Fast String Searching Algorithm[J]. Communications of the ACM, 1977, 20: 762-772.

[2] D M Sunday. A Very Fast Substring Search Algorithm[J]. Communications of the ACM, 1990, 33(8) : 132-142.

[3] T Lecroq. Experimental Results on String Matching Algorithms[J]. Software Practice & Experience, 1995, 25(7) : 727-765.

[4] 贺龙涛, 方滨兴, 胡铭曾. 对 BM 串匹配算法的一个改进[J]. 计算机应用, 2003, 23(3) : 6-9.

[5] <http://www.research.att.com/~lewis/reuters21578.html>[EB/OL].

作者简介:

李雪梅(1975-), 女, 讲师, 硕士, 主要研究方向为网络安全、信息处理; 代六玲(1977-), 男, 博士研究生, 主要研究领域为自然语言处理、网络内容管理。