基于字频的单模式匹配算法

刘功申,王永成,许欢庆
(上海交通大学计算机科学与工程系,上海 200030)

摘 要: 通过模仿人类查找字符串的过程,本文提出了一种新的单模式匹配算法(MBF 算法).该算法利用 QS 算法的思想和已经成功匹配的前、后缀信息对模式进行预处理.在查找阶段,MBF 算法利用了字符使用频率和连续跳跃的查找思想.实验表明,MBF 算法比同类的其它算法更加高效.

关键词: 字符使用频率; 模式匹配; 字符串
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2002) 12A-2079-04

A Single Pattern Matching Algorithm Based on Character Frequency

LIU Gong-shen, WANG Yong-cheng, XU Huan-qing
(*Department of Computer Science, Shanghai Jiaotong University, Shanghai* 200030, *China*)

Abstract: Based on the study of single pattern matching, MBF algorithm is proposed by imitating the string searching procedure of human. The algorithm preprocesses the pattern by using the idea of Quick Search algorithm and the already-matched pattern prefix and suffix information. In searching phase, the algorithm makes use of the character using frequency and the continue-skip idea. The experiment shows that MBF algorithm is more efficient than other algorithms.

Key words: character using frequency; pattern matching; string

1 引言

在单模式匹配算法的研究过程中,有三个占有重要地位的算法:1977 年,Knuth、Morris 和 Pratt 提出的 KMP 算法[2];1975 年,R S Boyer 和 J S Moore 提出的著名的 Boyer-Moore 算法(以下简称 BM 算法)[5];1990 年,Daniel M Sunday 提出的 Quick Search 算法(以下简称 QS 算法)[6].

单模式匹配的算法执行过程,有的从左至右(正向),有的从右至左(反向),有的从两个方向同时进行(双向).正向的方法有 KMP、QS 算法;反向的有 BM 算法及其一些变种;双向的有 M Crochmore 和 D Perrin 在 1991 年提出的"Two-way string-matching"[1].本文算法的特点是基于字频的,这就决定了本文算法的匹配方向是非常独特的,即跳跃式的匹配方式(无向).

正向的匹配算法在匹配过程中易于记住已成功匹配模式前缀,反向的匹配算法在匹配过程中易于利用已成功匹配模式后缀.这就是 KMP 和 BM 算法的成功的重要特点.本文算法(以下简称 MBF)充分利用了使用率最低的字符失败概率最大的特点,以提高算法的效率.MBF 算法同时也考虑了已匹配成功的模式前缀和后缀,并引入了连续跳跃的思想.

2 MBF 算法的特点

为了便于描述,给出一些符号约定和定义:已成功匹配的模式前缀(*mprefix*);已成功匹配的模式后缀(*msuffix*);模式字串 *pstring*[0: $m-1$],长度 *length* = m;模式字串对应的频率排序表 *pfrequent*[0: $m-1$],长度 *length* = m;文本字串 *tstring*[0: $n-1$],长度 *length* = n;设待处理的文本(如中英文等自然语言文本)定义在一个有限的字母表 A 上,可令 A 的大小为 σ;待比较字符(a);函数 *uchar*(a),用于根据待比较字符 a 在模式串中的位置信息来确定偏移量.

定义 1 待比较字符 *pstring*[0: $m-1$]和 *tstring*[i, $i+m-1$]匹配过程结束后,为确定下一次匹配的偏移量需要考察的字符.本文算法的待比较字符是 *tstring*[$i+m$].

MBF 算法的提出利用了下列特点:

2.1 按使用率次序匹配

定义 2 不可能成功的匹配 对任意 $j(0 \leqslant j \pi m)$,如果 *pstring*[j] $\neq$ *tstring*[$i+j$],那么 *pstring*[0: $m-1$]和 *tstring*[i, $i+m-1$]是不可能成功的匹配.

使用率低的字符在文章中出现的可能性也较低.先比较使用率较低的字符,加速不可能成功匹配的失败过程.

2.2 待比较字符在模式串中的位置信息

QS 算法主要是利用待比较字符在模式串中的位置信息来确定下一次匹配的偏移量.同时,当模式的长度 m 较小时,QS 算法的查找性能是非常优越的[6].

收稿日期:2002-06-10;修回日期:2002-11-15
基金项目:国家自然科学基金(No.60082003);科技部中小企业创新基金资助课题

2.3 已成功匹配部分的记忆

定义 3 已成功匹配的模式前缀 *pstring*[0: $m-1$]和 *tstring*[i, $i+m-1$]匹配过程结束后.如果存在 $j(0\leqslant j\pi m)$,使得 *pstring*[0: $j-1$]和 *tstring*[i, $i+j-1$]完全匹配,并且 *pstring*[j]$\neq$ *tstring*[$i+j$],那么,称 *pstring*[0: $j-1$]为已成功匹配的模式前缀.

定义 4 已成功匹配的模式后缀 与定义 3 类似.

当模式的长度 m 较大时,利用 BM 已成功匹配的模式后缀信息和 KMP 已成功匹配的模式前缀信息,提高了算法的性能.

2.4 连续跳跃

当文本字符与模式字符不匹配且偏移量为 $m+1$ 时,我们可以直接往后跳,如果跳转后字符的偏移量又为最大值时,则可继续往后跳.连续跳跃可以提高算法效率.

3 MBF 算法描述

MBF 算法分为两阶段:模式的预处理阶段和文本的查找阶段.

3.1 预处理阶段

预处理阶段的任务是初始化字符比较次序和计算 *uchar*、*msuffix* 以及 *mprefix* 三个偏移量函数.

3.1.1 字符比较次序及成功前后缀的初始化

本文算法需要对模式中字符的比较次序进行初始化,也就是初始化数组 *pfrequent* [0: m-1],该数组的每一个元素都是一个三元组(模式字符索引,成功前缀索引,成功后缀索引). *pfrequent* 的定义如下:

```
struct element{
        int index;
        int preindex;
        int sufindex;
} pfrequent[m];
```

例如,*pstring* = {中华人民共和国}.它们对应的(频率,索引)二元组分别是:中(58800.5,0),华(6291.8,1),人(68793.0,2),民(25259.9,3),共(7773.9,4),和(71372.8,5),国(52433.1,6).

模式字符索引对应为,*pfrequent*[j]. *index* = {1,4,3,6,0,2,5},也就是说,在匹配过程中各个字符的比较次序为,"华共民和中人国".

成功前缀索引对应为,*pfrequent*[j]. *preindex* = {0,0,0,0,0,2,5}.也就是说,当比较到 *pstring* 的第 *pfrequent*[j]. *index*($0\leqslant j\pi m$)个字符时,如果此时匹配失败,子串 *sub*[0: (*pfrequent*[j]. *preindex* -1)]已经匹配成功.

成功后缀索引对应为,*pfrequent*[j]. *sufindex* = {7,7,7,7,6,6,6},也就是说,当比较到 *pstring* 的第 *pfrequent*[j]. *index*($0\leqslant j\pi m$)个字符时,如果此时匹配失败,子串 *sub*[(*pfrequent*[j]. *sufindex*): (m-1)]已经匹配成功.

3.1.2 *uchar* 的计算

uchar 计算字母表 A 中每个字符所对应的偏移量[6].

3.1.3 *msuffix* 的计算

在 BM 算法中,*msuffix* 用来计算模式中的某个后缀被匹配成功时文本指针可以右移的偏移量[5].

3.1.4 *mprefix* 的计算

根据文献[2]可以计算出模式 *pstring* 的 *kmp* 向量,则. *mprefix*[j] = j - *kmp*[j].

3.2 查找阶段

本文算法的比较次序是根据模式串中字符的频率由小到大进行的.如图 1 所示.第一行为文本串 *tstring*[0: $n-1$],第二行为字频排序表 *pfrequent*[0: $m-1$].

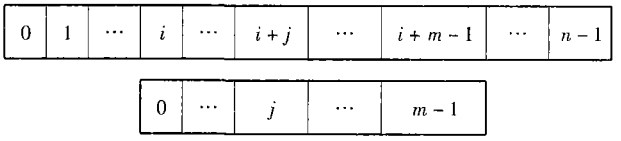

图 1 某次匹配示意图

假设模式和文本在文本的 i 处对齐.令 $j=0$,用字符 *pstring*[*frequent*[j]. *index*](频率最低的字符)和文章中相应位置的字符进行比较,成功,则 j 递增并继续进行比较.不成功,分下列步骤处理:

- 取 *uchar*[*tstring*[$i+m$]]、*msuffix*[*frequent*[j]. *sufindex*]和 *mprefix*[*frequent*[j]. *preindex*]的大者作为偏移量.
- 连续跳跃.如果该偏移量为最大值($m+1$),我们并非立即比较字符,而是将 i 再往后移动,如此往复,一直到偏移量小于最大值($m+1$).图 2 为一种连续跳跃的情形.

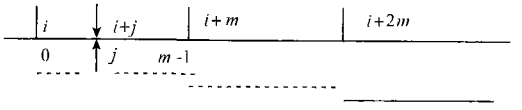

图 2 指针 i 的连续跳跃过程

如果模式被扫描完,则找到了模式的一次出现.以上述方式处理文本直至文本末尾,可以找出模式的所有出现位置.整个查找算法可描述如下:

算法 MBF Search

输入:文本 *tstring*[0: $n-1$],模式 *pstring*[0: $m-1$],文本长度 n,模式长度 m

输出:模式的所有出现位置

计算 *pfrequent*, *uchar*, *msuffix* 和 *mprefix*, $i-0$

循环以下各步骤直到全文结束:

pfrequent 对齐于文本 i 处;

按字频从小到大比较各个字符,直到匹配失败或成功;如果匹配成功,则找到模式的一次匹配;否则,选择 *uchar*[*tstring*[$i+m$]], *msuffix*[*frequent*[j]. *sufindex*]和 *mprefix*[*frequent*[j]. *preindex*]的大者作为偏移量;如果此次循环的偏移量是最大值($m+1$),则连续跳跃直到偏移量小于 $m+1$.

4 算法的分析

像 BM 算法一样,本文算法查找阶段的复杂性分析是一件比较困难的事情.Richard Cole 在文[3]指出,当模式为非周期模式($P!=\omega v^k$,ω 为 v 的后缀,$k\geqslant 2$)时,BM 算法的紧上界(Tight Upper Bound)为 $3n-3(n-m+1)/(m+2)$.下面给出本文算法相关分析和几个结论.

4.1 加速失败过程

对模式串 *pstring* 中的字符按使用率从小到大排序后,再进行比较.QS 算法实际上是默认字符出现的概率相等,可以说是本文比较方式的一个特例.由于很难用公式证明排序后比排序前效率高,本文采用实验的方法加以验证.实验过程中使用"文汇报"资料(总计 350 篇,306870 字).

表 1　排序前后比较次数对照表

长度	项目　排　序	未排序	比　率(%)
2	10284	5832	56.7
3	4012	2424	60.4
4	4607	2545	55.2
7	10171	7161	66.8
8	5345	1885	35.3
9	7667	4638	60.5

如表 1 所示,本试验随机选取了 6 组字符串.每组 10 个,每组长度分别是 2,3,4,7,8,9 个字.表中比较次数是某字符串比较失败时已经成功比较过的字符个数.表 1 是对详细实验数据的总结.从表 1 中可以看出,排序后的比较次数仅仅是排序前的百分之五十左右.通过上述试验,可以从实践上说明按频率排序的思想可以提高模式匹配的效率.

4.2 MBF 算法

在最优情况下的时间复杂度为 $O(n/(m+1))$,在最坏情况下的时间复杂度为 $O(nm)$.

证明　在预处理阶段,计算 *uchar* 时,对于不在模式中出现的字符,其偏移值为 $m+1$,考虑一种特殊的情况,在 n 个 b 中找 m 个 a,显然每次都可跳跃 $m+1$ 个字符,此时的时间复杂度为 $O(n/(m+1))$.

又每次跳跃的字符数为 *msuffix*、*mprefix* 与 *uchar* 中的最大值,不会超过 $m+1$,故本节算法在最优情况下的时间复杂度为 $O(n/(m+1))$.

在证明 MBF 算法在最坏情况下的时间复杂度为 $O(nm)$ 这个结论之前,我们先讨论一条性质.

性质 1　在 MBF 算法中,每个字符被匹配的次数不超过 m.

通过算法的预处理阶段,我们可以看出,$uchar \geqslant 1$, $msuffix \geqslant 1$, $mprefix \geqslant 1$,因此每次文本指针的偏移量(不管匹配是否成功),由于模式的长度为 m,因此文本中某个字符被比较的最大次数为 m.

现在我们来证明结论 4.2.在查找过程中,由于每个字符被匹配的次数不超过 m,因而总的匹配次数不超过 mn.另外,我们考虑在 n 个 a 中找 m 个 $a(n>m)$,除了开始的 $m-1$ 个字符和末尾的 $m-1$ 个字符之外,中间的每个字符都比较了 m 次,显然时间复杂度为 $O(nm)$,故最坏情况下的时间复杂度为 $O(nm)$.

4.3 平均时间复杂度的探讨

在速度方面,算法的平均时间复杂度是算法性能中最重要的指标.由于证明难度大,在此只做一些简单近似的讨论.

假设字母表 A 中每个字符出现的概率均相等,即 $pl = 1/\sigma$,则其不出现的概率为 $1-pl$,故我们获得如下的一些推论.

推论 1　某个字符不在长度为 m 的模式中出现的概率为 $(1-pl)^m$.

利用乘法原理即可得出.

推论 2　某个字符在长度为 m 的模式中出现(一次或多次)的概率为 $1-(1-pl)^m$.

根据推论 1 易证.

推论 3　记长度为 $k(k \leqslant \lfloor m/2 \rfloor)$的某个后缀在长度为 m 的模式中重复出现两次(因为 *msuffix* 只考虑两次)的概率为 p_s,则

$$p_s = \frac{(m-2k+1)}{(m-k)^\sigma}$$

证明　模式(证明过程中不考虑字符的排序)中去掉两个长度为 k 的子串,还剩 $m-2k$ 个字符,固定长度为 k 的后缀,将另一个长度为 k 且等于该后缀的子串视为一个整体,在组合意义上,该子串与剩余的 $m-2k$ 个字符可进行全排列,如图 3 所表示:

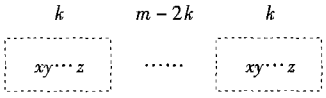

图 3　后缀 $xy\cdots z$ 的两次出现

故排列数为$(m-2k+1)!$.又根据乘法原理,$m-k$ 个字符可出现$(m-k)^\sigma$ 种组合,故 $p_s = \frac{(m-2k+1)!}{(m-k)^\sigma}$ 成立.我们可以就此推出,长度为 $k(k \leqslant \lfloor m/2 \rfloor)$的某个后缀在长度为 m 的模式中不重复出现两次的概率为:$1-p_s = 1-\frac{(m-2k+1)!}{(m-k)^\sigma}$

在上述假设下,我们来推导一下平均的字符比较次数.如果字符不在模式中出现,则每次跳过的字符数为$(m+1)$.对于字符在模式中出现的情形,有两种情况,一种是有重复出现的后缀,偏移量从 k 到$m-k$,取平均值 $m/2$,匹配的字符数为 $k+1$;另外一种情况是没有任何后缀重复出现,匹配的字符数为 $k+1$,偏移量为 m;对于 k 大于 $m/2$ 且小于 m 的情况,匹配的字符数为 $k+1$,偏移量为 m.在上述情况下,取 *msuffix*$[j+1]$与 *uchar*$[tstring[i+m]]$中的大者,因此我们可近似得到平均的匹配次数 $\bar{Z}$:

$$\bar{Z} = \begin{cases} (1-p_l)^* n/(m+1) + [l-(1-p_l)^*]\min\{(k+1) \\ \quad \cdot(2ps+(l-ps))n/m, n\}, & l \leqslant k \leqslant \lfloor m/2 \rfloor \\ (1-p_l)^* n/(m+1) + [l-(1-p_l)^*]\min\{(k+1) \\ \quad \cdot n/m, n\}, & \lceil m/2 \rceil \leqslant k \leqslant \lfloor m/2 \rfloor \end{cases}$$

由于 p_l 一般都很小,因此 $\bar{N}$ 主要由$(1-p_l)^m \frac{n}{m+1}$这一项决定.这也说明了模式越长查找越快的原因.

推论 4　记长度为 $k(k \leqslant \lfloor m/2 \rfloor)$的某个前缀在长度为 m 的模式中重复出现两次的概率为 ps,则

$$ps = \frac{(m-2k+1)!}{(m-k)^\sigma}$$

推论 4 的证明同推论 3.这也证明了模式较长时,*mprefix*

对算法效率的贡献.

5 对照实验和结论

表 2 几种算法的对比实验结果(时间单位为 ms)实验条件:PII 300,128MB RAM

长度 \ 算法	KMP T/ms	BM T/ms	QS T/ms	RF T/ms	AG T/ms	MBF T/ms
2	195	230	130	161	861	110
4	143	130	90	90	440	80
6	138	90	75	75	320	70
8	142	80	65	65	250	60
10	141	70	55	60	220	52
12	157	65	53	60	190	51
14	164	60	51	50	180	50
22	172	50	40	50	140	40

为了测试算法的实际性能,参考文献[4]的实验结果,我们选取几种具有代表性的同类算法进行比较:BM[5],QS[6],Reverse Factor(RF)[7]和 Apostolico-Giancarlo(AG)算法[8].所获实验数据如表 2 所示.

从表 2 可以看出,在利用 7 种模式长度进行测试的过程中,本节所提出的算法都属于性能最好的算法,特别是在模式长度较小的情况下,优势更为明显.

由于查找问题的普遍性,故该方法具有广阔的应用前景.对于字符使用频率的应用决定了本文算法更加适用于大字符集语言.

参考文献:

[1] M Crochemore, D Perrin. Two-way string-matching[J]. J. Assoc. Comput, 1991, 38(3): 651 - 675.

[2] Knuth D E, Morris J H, Pratt V R. Fast pattern matching in strings[J]. SIAM Journal on Computing, 1977, 6(1): 323 - 350.

[3] Richard Cole. Tight bounds on the complexity of the Boyer-Moore string matching algorithm[J]. SIAM J. Comput, 1994, 23(5): 1075 - 1091.

[4] Thierry Lecroq. Experimental results on string matching algorithms[J]. Software Practice and Experience, 1995, 25(7): 727 - 765.

[5] R S Boyer, J S Moore. A fast string searching algorithm[J]. Commun. ACM, 1977, 20(10): 762 - 772.

[6] Daniel M Sunday. A very fast substring search algorithm[J]. Commun. ACM, 1990, 33(8): 132 - 142.

[7] Ricardo Baeza-Yates, Gaston H Gonnet. A new approach to text searching[J]. Commun. ACM, 1992, 35(10): 74 - 81.

[8] A Apostolico, R Giancarlo. The Boyer-Moore-Galil string searching strategies revisited[J]. SIAM J. Comput, 1986, 15(1): 98 - 105.

作者简介:

刘功申 男,1974 年生于山东聊城,上海交通大学计算机系博士研究生,研究方向:网络信息智能处理.

王永成 男,1939 年生于江苏扬州,上海交通大学计算机系教授,博士生导师,研究方向:中文信息处理.

许欢庆 男,1973 年生于江西新余,上海交通大学计算机系博士研究生,研究领域:智能信息检索.