一种新的多模式快速匹配算法*

王若梅，张绮雯，周 凡
（中山大学计算机科学系，广东 广州 510275）

摘 要：提出了一种针对多模式的快速模式匹配算法。算法分为预处理阶段和匹配阶段两个部分，预处理阶段对所有待匹配的模式进行分析，构造一个关于这些模式的树型有限状态自动机，匹配阶段利用这个模式自动机，对文本串进行一次性的搜索，查找文本是否包含模式集中的模式。为了提高了匹配速度，算法利用已匹配的字符串信息实行跳跃式的比较，避免了文本扫描指针的回溯。

关键词：模式匹配；多模式；有限状态自动机；内容过滤
中图分类号：TP301 文献标识码：A 文章编号：0529-6579（2005）S2-0107-04

在模式匹配技术的实际应用中，通常对匹配过程的时间要求十分严格，如内容过滤防火墙[1]、网络入侵检测系统等[2,3]。KMP 算法[4]，Boyer-Moore（BM）算法[5]，Karp-Rabin（KR）算法[6]等单模式匹配算法虽然在单次匹配时速度较快，但若模式数目不只一个，则要对文本进行多次扫描匹配，模式数目越多效果越差。文献[7]提出了基于模式匹配的网络入侵检测系统，检测引为基于模式匹配的方法。在模式匹配算法方面为了适应一些模式数量较多、对匹配速度要求较高的应用环境，本文提出了一种快速的多模式匹配算法。

1 一种新的多模式快速匹配算法

算法分为预处理阶段和匹配阶段两个部分。预处理阶段对所有待匹配的模式进行分析和处理，构造一个关于这些模式的树型有限状态自动机。匹配阶段则利用前面建立的有限自动机对文本进行一次性的搜索。对于固定的待匹配模式集，预处理过程只需进行一次，就可以对不同的文本进行搜索。

1.1 预处理阶段

预处理阶段首先要构造一个有限自动机[7]，也可以看作是由所有模式的前缀构成的一棵前缀树。设待匹配模式的集合为 $P=\{p_1, p_2, \cdots p_k\}$，则这些模式的所有前缀可以构成一个集合 $Q=\{q_{11}, q_{12}, \cdots q_{21}, q_{22}, \cdots\}$。前缀树每个结点表示一个状态，而每个状态都代表一个模式的前缀或多个模式的共同前缀；反之，集合 Q 中的每一个前缀都可以用唯一的状态来表示。例如，待匹配模式集合为 $P=\{$shine，shimmy，hitter$\}$，构成的有限自动机如图 1 所示。

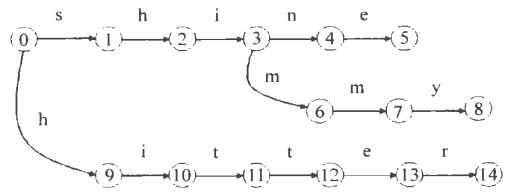

图 1 有限自动机示例

Fig. 1 The finite state automata

1.2 模式匹配阶段

在匹配的过程中，利用预处理阶段构造的模式自动机，对文本串进行一次性的搜索，查找文本是否包含模式集中的模式。例如在图 1 的有限自动机对文本进行搜索，从状态 0 开始，根据当前的状态和从文本中读入的字符决定下一个状态。若进入终止状态，表示已找到一个匹配的模式。若根据当前状态和读入的字符找不到下一个状态，表示出现失配情况。

出现失配后，最简单的方法是把这个自动机向前移动一位，然后从状态 0 开始重新搜索。为了加快匹配的过程，本算法利用已匹配的信息，通过向前看一个字符实现跳跃式的比较的方法，避免了文本扫描指针的回溯，减少了比较的次数。

例如，在上面的自动机中，若在状态 3 失配，则文本中已出现’shi’这一字符串，这时，可直接进入状态 10，因为状态 10 表示已匹配’hi’这一前缀，从而避免了从状态 0 开始重复比较’hi’

* 收稿日期：2005－08－08
基金项目：广东科技计划资助项目（20042060274）
作者简介：王若梅（1961 年出生），女，副教授；E-mail：isswrm@mail.sysu.edu.cn

这两个字符。因此，改进的方法是，若在状态 j 失配，则直接进入状态 k，其中状态 k 表示的字符串是状态 j 表示的字符串的后缀，且状态 k 表示的字符串是所有满足条件的最长的字符串。

另一方面，因为失配后，模式自动机至少要向前移动一个位置，若要匹配成功，前方的字符必须在成功匹配的模式中出现。因此，进一步改进的方法是，以最短模式为标准向前看文本的一个字符，计算该字符在所有模式中最后出现的位置，然后把整个自动机移动若干位，使该字符与其在模式中最后出现的位置对齐，从而跳过一些不必要的比较。

1.3 算法的描述

1.3.1 预处理过程 预处理过程主要是构造模式自动机，以及计算失配后文本指针跳跃的位置和转入的新状态。

首先，读取模式集合中的所有模式，并用 Aho Corasick 算法中建立有限自动机的子算法[4] 构造一个包含所有模式的前缀信息的自动机，得到自动机的状态转移函数 GOTO 和输出函数 OUTPUT，然后把结果保存在如下两个表中（其中 s 表示自动机中的状态，c 表示字符表 Σ 中的字符）：

$$GOTO[s][c]=\begin{cases} s1, & \text{（若 } s1 \text{ 是在状态 } s \text{ 下读入字符 } c \text{ 后的下一个状态）}\\ fail, & \text{（若自动机中不存在从状态 } s \text{ 读入字符 } c \text{ 的状态转移）}\end{cases}$$

$$OUTPUT[s]=\begin{cases} \text{状态 } s \text{ 所表示的字符串}, & \text{（若状态 } s \text{ 是终结状态）}\\ NULL, & \text{（若状态 } s \text{ 不是终结状态）}\end{cases}$$

然后，计算失配后的文本指针的跳跃和新状态，计算结果用一个表 shift 来保存，表中的元素是一个二元组（jump，newstate），即 shift[s] [c] =（jump，newstate），表示当在状态 s 失配，且向前看的文本字符为 c，则文本扫描指针向前跳跃 jump 个位置，并从状态 newstate 开始继续比较。计算失配后的文本指针跳跃和新状态算法描述如算法 1。

算法 1 计算失配后的文本指针跳跃和新状态

输入：保存有模式自动机信息的 GOTO 表

输出：保存失配后文本指针的跳跃和新状态的 shift 表

1）minlen←最短模式的长度

2）对每个 $c\in\Sigma$，last[c] 表示 c 在所有模式中的最末出现位置，并初始化为 0

3）建立队列 queue 并置空

4）对每个满足 GOTO[0] [c] ≠fail 的字符 c（即位于自动机的第一层的字符），进行以下操作：

（1）last[c] ←1

（2）s ← GOTO[0] [c]

（3）next[s] ← 0

（4）s 进入队列 queue

5）当队列 queue 不为空，进行以下循环（计算 next[ ] 和 last[ ]）：

（1）s ← 队列 queue 的队头元素出队

（2）对每个满足 GOTO[s] [c] ≠fail 的字符 c 进行以下操作：

① s1← GOTO[s] [c]

② next_ s:= next[s]

③ 若 GOTO[next_ s] [c] = fail，重复 next_ s ← next[next_ s] 直到 GOTO[next_ s] [c] ≠fail

④ next[$s1$] ← GOTO[next_ s] [c]

⑤ $s1$进入队列 queue

⑥ 若 $s1$在自动机中所处的层次＜ minlen，Last [c] ←$s1$ 所处的层次

6）对自动机中的每个状态 s 和每个 $c\in\Sigma$，进行以下操作（计算 shift[s] [c] ）：

（1）k ← 状态 s 在自动机中所处的层次

（2）shift[s] [c] . jump←minlen -last[c] − k +1

（3）若 shift[s] [c] . jump≥0，则 shift[s] [c] . nextstate ← 0

（4）否则，shift[s] [c] . nextstate ← next[s] ，shift [s] [c] . jump ← 0

例如对于图 1 所示的例子，用算法 1 建立的 shift[ ] [ ] 表的内容如表 1 所示，其中表格的第一列表示状态，第一行表示向前看的字符，表格体中的二元组表示（jump，newstate）：

表 1 shift[ ] [ ] 的值

Tab. 1 The value of shift[ ] [ ]

状态＼字符	e 或 m	n 或 t	i	h	s	其他
0	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(5, 0)	(6, 0)
1	(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(5, 0)
2	(0, 9)	(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)
3	(0, 10)	(0, 10)	(0, 0)	(1, 0)	(2, 0)	(3, 0)
⋮	⋮	⋮	⋮	⋮	⋮	⋮

1.3.2 匹配过程 匹配过程利用预处理过程构造的模式自动机和失配转移表 shift，对文本串进行一次性的搜索，查找文本是否包含模式集中的模式。模式匹配算法描述如算法 2。

算法 2 模式匹配算法

输入：文本串 T [1：n]，GOTO 表，OUTPUT 表，shift 表

输出：匹配成功的模式串

1）minlen ← 最短模式的长度

2）初始化文本扫描指针 i← 1

3）初始化当前状态 s← 0

4）当 $i \leqslant n$，进行以下循环：

（1）c ← $T[i]$

（2）若 GOTO[s] [c] ≠ fail，则：

① s ← GOTO[s] [c]

② i ← i+1

③ 若 OUTPUT[s] ≠ NULL，则输出 OUTPUT[s]

（3）若 GOTO[s] [c] ＝fail（即出现失配情况），则：

① 计算向前看的字符的位置 k ← minlen－s 所处的层次＋1

② 向前看的字符为 ch ←T[i＋k]

③ 文本指针跳跃到新位置 i← i ＋ shift[s] [ch] . jump

④ 新状态为 s ← shift[s] [ch] . nextstate

2　算法分析

设共有 k 个模式，长度分别为 L_1，…L_k，最大长度为 m，字符表的大小为 δ 对于预处理过程：建立有限自动机要扫描所有模式一遍，时间与模式的总长度成线性关系，即 $O(\sum_{i=1}^{k} L_i)$；算法 1 中计算 last[ ] 和 next[ ] 时，每个状态入队和出队各一次，对某个状态计算 next 的内层循环执行的次数小于该状态的所处的层次，而状态总数≤$\sum_{i=1}^{k} L_i$，任一个状态所处的层次≤模式最大长度 m，所以这一步的时间为 $O(m\sum_{i=1}^{k} L_i)$，接下来计算 shift 表的外层循环的次数为状态总数，内层循环次数为字符表的字符总数即 δ 所以这一步的时间为 $O(\delta \sum_{i=1}^{k} L_i)$。综上所述，预处理过程的时间为 $O(\sum_{i=1}^{k} L_i + m\sum_{i=1}^{k} L_i + \delta\sum_{i=1}^{k} L_i) = O((m+\delta)\sum_{i=1}^{k} L_i)$。

设文本的长度为 n，在匹配过程（算法 2）中，文本扫描指针 i 是无回溯的，即使不考虑扫描指针的跳跃，最坏情况下的比较次数也不超过 $2n$ 次。再把文本指针的跳跃考虑进去，设每次比较后指针的平均跳跃距离为 k，则匹配过程的时间为 $O(n/k)$，最好情况下 k 可以达到（minlen＋1），其中 minlen 为最短模式的长度，最好时间可以达到 $O(n/(\text{minlen}+1))$，最坏情况下为 $O(n)$。

选取的目标文本大小为 15M，与朴素的单模式匹配算法和 AC 算法进行对比，结果如表 2 所示：

表 2　运行时间比较

Tab. 2　The comparison of time　/ms

模式数目	朴素算法		AC 算法		本文算法	
	预处理	匹配过程	预处理	匹配过程	预处理	匹配过程
5	0	195	10	120	25	30
10	0	382	20	141	48	37
20	0	750	38	173	95	48
50	0	1818	83	207	220	72
100	0	3534	150	215	400	98
200	0	7140	281	222	781	102

从表 2 中可以看出，朴素的匹配算法的时间随着模式数目的增加而迅速增加，所需时间与模式数据成正比，这是因为该算法需要根据模式的数目对文本进行多次的扫描。AC 算法和本文算法的匹配过程的时间对模式数目的增加远远没有朴素算法那样敏感，而只有缓慢的增长，并且模式数目越多，这两种算法相对于朴素算法的优势就越明显，这是因为这两种算法都不需要根据模式的数目多次扫描文本，但是当模式数目增加时，自动机的规模也增大，所以匹配的时间也略有增长。AC 算法和本文的算法都需要进行预处理，预处理的时间随着模式数目的增长而成比例地增长，本文提出的算法的预处理时间比 AC 算法的多，这是为了实现加速措施而增加预处理阶段的工作量而造成的。但在匹配过程的时间上，因为本文的算法在匹配时采用了加速措施，所以匹配时间明显小于 AC 算法，这正是本文算法的最大优点。

3　应　用

本文提出的算法适用于很多需要模式匹配的领域，例如目前应用广泛的基于关键字的内容过滤防火墙。内容过滤防火墙中的模式匹配过程有如下的特点和要求：第一，防火墙系统对时间性能的要求很高，所以对模式匹配过程的速度要求也很高；第二，要检查的关键字通常不止一个，数目比较多；第三，要检查的关键字集合确定后，就要不断对大量经过防火墙的数据进行检查，也就是说，用相同的模式集对大量不同的目标文本进行匹配。对于应用环境中的以上特点和要求，本文提出的算法可以很好地适应，并显示出优势：算法分预处理和匹配两个阶段，并将尽可能多的工作放在预处理阶段进行，使得匹配过程的速度加快，而根据防火墙的特点，预处理过程可以在防火墙运行之前完成，它的时间不包含在对数据包内容的检测时间当中，而且对于相同的模式集合，预处理过程只需进行一次，把结果保存起来，就可以对所有经过防火墙的数据

包进行高效的匹配过程。

参考文献：

[1] 亿阳信通股份有限公司. 防火墙内容过滤的关键——高速度、设置灵活[J]. 计算机安全, 2003(6): 20－21.

[2] 蒋建春, 卿斯汉. 网络安全入侵检测: 研究综述[J]. 软件学报, 2000, 11(11): 1460－1467.

[3] COIT C J, STANIFORD S, MCALERNEY J. Towards faster string matching for intrusion detection or exceeding the speed of Snort[C] //DARPA Information Survivability Conference and Exposition, 2001.

[4] KNUTH D E, MORRIS J H, PRATT V R. Fast pattern matching in strings[J]. SIAM Journal on Computing, 1977, 6(2): 323－350.

[5] BOYER R S, MOORE J S. A fast string searching algorithm [J]. Communications of the ACM, 1977, 20(10): 762－772.

[6] KARPR, RABIN M. Efficient randomized pattern-matching algorithms[J]. IBM Research and Development, 1987, 31(2): 249－260.

[7] 田大新, 刘衍衍, 魏达, 等. 基于模式匹配的网络入侵检测系统. 吉林大学学报: 信息科学版, 2003, 21(4): 412－416.

A New Fast Multiple Pattern Matching Algorithm

WANG Ruo-mei, ZHANG Qi-wen, ZHOU Fan

(Department of Computer Science, Sun Yat-sen University, Guangzhou 510275, China)

Abstract: To present a new multiple-pattern matching algorithm. The algorithm includes pre-processing and matching processing. Pre-processing analyzes all the patterns to build finite state automata. Matching processing uses these automata to search the patterns. The algorithm uses jump comparison to improve the speed.

Key words: pattern matching; multi-pattern; finite state automata; content filtering