ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.3, March 2008, pp.674−686
DOI: 10.3724/SP.J.1001.2008.00674

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

面向中英文混合环境的多模式匹配算法*

孙钦东[1,2+], 黄新波[3], 王 倩[1]

[1](西安理工大学 计算机科学与工程学院,陕西 西安 710048)

[2](西安交通大学 电子与信息工程学院,陕西 西安 710049)

[3](西安工程大学 电子信息学院,陕西 西安 710048)

Multiple Pattern Matching on Chinese/English Mixed Texts

SUN Qin-Dong[1,2+], HUANG Xin-Bo[3], WANG Qian[1]

[1](School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048, China)

[2](School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

[3](College of Electronics and Information, Xi'an Polytechnic University, Xi'an 710048, China)

+ Corresponding author: Phn: +86-29-82312231, E-mail: sqd@xanet.edu.cn

Sun QD, Huang XB, Wang Q. Multiple pattern matching on Chinese/English mixed texts. *Journal of Software*, 2008,19(3):674−686. http://www.jos.org.cn/1000-9825/19/674.htm

Abstract: The characteristics of multiple pattern matching in mixed Chinese and English text and the problem of the existing multiple pattern matching algorithms used for processing mixed Chinese and English text are analyzed. A theorem of multiple pattern matching in mixed Chinese and English text is discovered and proved. A novel multiple pattern matching algorithm based on the threaded trie tree is proposed, which expands the standard trie structure, constructs the hash trie matching machine with the codes of Chinese and English characters, and threads the trie tree according to the characteristic of patterns set. The proposed algorithm does not need complex hash operation, and the matching pointer does not need backdate during matching. Theoretic analysis and experimental results demonstrate that the proposed algorithm efficiently solves the space expansion problem, and process mixed Chinese and English text correctly and efficiently with lower time and space complexity.

Key words: multiple pattern matching; Chinese/English mixed; Hash; Trie

摘 要: 分析了中英文混合环境下多模式匹配的特点,以及已有多模式匹配算法应用于中英文混合环境时的不足,给出并证明了中英文混合环境下多模式匹配算法的性能定理,提出了一种适合于中英文混合环境的基于线索完全哈希 Trie 结构的多模式匹配算法.该算法扩展了标准 Trie 结构,以中英文字符内码为键值构造完全哈希 Trie 匹配机,并利用模式串之间的关系对 Trie 匹配机进行线索化.理论分析与实验结果表明,所提出的算法在匹配中无需复杂的哈希运算,不需要回溯匹配指针,在中英文混合环境下能够进行正确、高效的匹配,而且不存在空间膨胀问题,具有较低的空间与时间复杂度,有较大理论与应用价值.

关键词: 多模式匹配;中英文混合;哈希;Trie

* Supported by the Key Science-Technology Project of Xi'an of China under Grant No.06JK225 (西安市科技攻关项目); the Research Program of the Education Department of Shannxi Province of China under Grant No.06JK225 (陕西省教育厅专项科研计划)

Received 2006-08-21; Accepted 2007-08-24

中图法分类号: TP301 文献标识码: A

目前,互联网络已成为全球范围内信息传播的最主要渠道之一.与此相对应,网络信息检索、网络信息内容审计等信息处理技术的应用也越来越广泛,而多模式匹配算法则是这些技术的核心,与网络信息处理相关的问题最终大多转换成多模式匹配问题.与传统的多模式匹配相比,面向网络信息文本的多模式匹配存在较大差异.由于网络协议编码、不同区域使用语言不同等多方面原因,在网络数据包中,与协议相关的部分通常是英文字符,与内容相关部分则是其他类型字符.因此,待处理的文本通常由多种编码规则不同的字符混合组成.就中文而言,由于汉字有简、繁体之分,此种情况更为突出,文本中会包含两类或两类以上的字符.

已有的多模式匹配算法大都是面向单一字符环境,应用于上述中英文混合环境时,存在漏匹配、误匹配等问题[1−6],研究高效、实用的适合于中英文字符混合环境下的多模式匹配算法有较大的理论与应用价值.针对上述问题,本文提出了一种基于线索完全哈希 Trie 匹配机的多模式匹配算法 THT(threaded Hash Trie).理论分析与实验结果表明,THT 算法完全适合于中英文混合环境的多模式匹配应用,而且具有较低的时间复杂度和可接受的空间复杂度.

1 已有多模式匹配算法分析

英文字符采用 ASCII 编码,一个字符占用一个字节;中文字符则采用双字节编码,简体字采用 GB 编码,每个字节的最高位为 1;繁体字使用 BIG5 编码,高字节最高位必须为 1,低字节的最高位没有要求.这样,在混合文本串中,一个字节的属性可能有以下几种情况:最高位为 0 的字节可能是英文字符或繁体汉字的低字节;最高位为 1 的字节可能是简体或繁体汉字的高、低字节.

这种编码长度及规则的不同,导致在混合环境下进行多模式匹配变得极为复杂.在单一字符环境下,字符的编码长度是一致的,匹配算法只需根据固定长度进行匹配即可,并可以根据规律进行跳跃等来加速匹配.但在混合环境下,由于网络信息的随机性,文本串中不同编码字符出现的概率是随机的,若在匹配过程中错误地确定字节的属性,则会产生错位匹配的问题,导致将文本串汉字的低字节与模式串中汉字的高字节进行对比,从而产生一连串的匹配错误;或者将繁体汉字的低字节误匹配为英文字符等.

例如,对于字符串“〈b〉搜索产品〈/b〉”(3C|62|3E|CBD1|CBF7|B2FA|C6B7|3C|2F|62|3E),假设“产品(B2FA|C6B7)”为关键词,如果在处理字符边界时出现错误,导致匹配从“搜”的低字节 D1 开始,而使字节组合变为“3C|62|3E|CB|D1CB|F7B2|FAC6|B73C|2F|62|3E”,显然会产生一连串的错位,导致漏匹配;如果错位的编码正好组成某个关键词,则会导致误匹配,如图 1 所示.

Character string	〈	b	〉	搜		索		产		品		〈	/	b	〉
ISN	3C	62	3E	CB	D1	CB	F7	B2	FA	C6	B7	3C	2F	62	3E
Correct matching	3C	62	3E	CB	D1	CB	F7	B2	FA	C6	B7	3C	2F	62	3E
Wrong matching	3C	62	3E	CB	D1	CB	F7	B2	FA	C6	B7	3C	2F	62	3E

Fig.1 The sketch of misalignment

图 1 错位匹配示意图

发生字节错位后,如果该中文字符后面紧跟着出现一个中文模式串,则会产生漏匹配,直至出现一个非中文字符时才能纠正此种错位;如果某个中文字符的低字节和相邻中文字符的高字节恰好组成某个模式串,则会造成误匹配.

为了防止错位匹配,目前常用的方法是先过滤掉文本串中的非中文字符,将其转换成一个纯中文文本串,再进行相应处理.显然,这需要两次扫描文本串,会浪费较多的系统时间,匹配效率较低.这对网络信息内容审计、内容搜索等实时性较高的处理有极大的影响.

针对上述问题,有少数中国学者对中文及中英文混合环境下的模式匹配进行了一定的研究,这些研究对中英文混合环境下的模式匹配作了非常有益的探索,但均存在一定的缺陷.对于主要使用单字节编码的国家,如美

国、英国等,因为不存在上述讨论的问题,相关文献也较少.

文献[1]提出的经典 DFSA 算法应用于英文字符环境时有很高的效率,但直接应用于中文字符匹配时,存在着存储空间膨胀的问题.DFSA 通常采用存取效率高的完全哈希表存储状态转换函数以获得高匹配速度,该表所占空间为 *sumof*(*state*)×256,其中,*sumof*(*state*)为自动机状态总数,256 为单内码字符集的最大字符数;如果采用同样方法处理中文字符,则空间为 *sumof*(*state*)×256×256,是英文字符的 256 倍.随着状态数的增加,完全哈希表所需的存储空间会快速膨胀,导致算法在实际中无法直接应用.

文献[6]针对经典多模式匹配算法 DFSA 应用于中文字符匹配时存在的存储空间膨胀问题,提出了通过分解中文字符内码构造组合状态自动机,并利用 QS 算法的思想进行加速.该算法解决了中文字符构建完全哈希表时的空间膨胀问题,但它只适用于纯中文字符环境,在中英文混合环境下会导致字节错位问题.

文献[7,8]采用加“标记”方法来防止匹配中的错位问题,即模式串和待匹配文本中的每一位的属性都被标记.每一位的类型可分为以下 3 类:(类型 0)英文的 ASCII、(类型 1)中文字符的高字节、(类型 2)中文字符的低字节.当两个字节被比较时,不仅它们的数值相同,而且还要有相同的标记,才认为是匹配成功.这种方法可以解决中英文混合环境下字节错位的问题,但因需要对待匹配文本串进行预扫描,以对每个字节加注标记,所以匹配效率较低,而且没有考虑 ACSII,GB,BIG5 这 3 种编码混合的情况.

文献[9]通过对中文字符内码的高字节 *Hbyte* 及低字节 *Lbyte* 进行哈希运算,如 256×*Hbyte*+*Lbyte*,将所有中文字符映射到大小为 65 536 的集合中,并在此基础上提出一种基于两级哈希表的 DFSA 算法.该算法以中文字符内码的两个字节作为一个最小匹配单位,能够避免中英文混合环境下的字节错位问题,而且也适用于 3 种编码混合的情形,但在匹配中要对待匹配文本串中的每一个中文字符进行哈希映射运算,无疑会影响算法的匹配速度,在实际应用中也验证了这个问题的存在,而且对算法效率的影响较大.

另外,还有一些文献也对多模式匹配算法进行了研究,但都没有考虑中英文混合环境对匹配算法的准确性及匹配效率的影响,如文献[10,11].

可以看出,研究一种高效、实用的适合于多种编码字符混合环境的多模式匹配算法对网络信息处理相关技术的发展极为重要,具有较大的理论与应用价值.

2 混合环境下多模式匹配相关定理

通过前面的分析可以看出,与单一字符环境相比,混合环境下的多模式匹配有自己的特点.由于英文字符与中文字符的内码所占字节数不同,因此,要匹配两个英文字符还需对两个字节的数据进行比较,而要匹配两个中文字符则需要对 4 个字节的数据进行比较.为了方便后面的论述,这里给出匹配次数和比较次数的定义.

定义 1(匹配次数与比较次数). 无论待匹配字符的内码为单字节还是双字节,每完成一个字符的比较,定义为一次匹配,即匹配次数为 1.要完成一个字符的一次匹配,所需比较的字节数目,即该字符的内码字节数称为该次匹配的比较次数.

定义 2(字节长度与字符长度). 假设 T 为任一包含中英文字符的随机文本串,其中,中文字符的个数为 m,英文字符的个数为 n,则定义 T 的字符长度为 T 中包含的中文字符与英文字符的个数之和,即 $m+n$;字节长度为 T 在内存中所占的字节数,即 $2m+n$.

对于英文字符环境下的模式匹配而言,匹配次数与比较次数是相等的,对于英文文本串,其字符长度与字节长度也是相等的.但对于多字节内码的字符而言,它们之间则存在着内码编码长度的倍数关系.

定理 1. 如果模式串集合中含有中文字符模式串,则对任意一个随机的含有中英文字符的待匹配文本串,在任何情况下都能正确完成匹配所需要的匹配次数不小于该文本串的字符长度.

证明:令 $T[1,2,\ldots,M]$为待匹配的含有中英文字符的随机文本串,其中,M 为其字节长度,N 为其字符长度,$T[i]$ $(1\le i\le M)$为文本串中的一个字节,Σ为模式串集合.

根据是否对待匹配文本 T 进行预处理,匹配算法可分为两类:一类对 T 进行预处理;一类不进行处理,直接进行匹配.与此相对应,定理证明也分为两部分.

(1) 对 T 进行预处理.无论是对 T 中字节加标记,还是去除 T 中的英文或中文字符,将其转化成单一字符的文本串,即使只对中文字符的高字节进行判别(确定最高位是否为 1),预扫描也至少要进行 N 次比较才能确定 T 中每个字节 $T[i](1\le i\le M)$的正确属性.由于预处理后还要进行匹配,所以,此类算法能完成的正确匹配次数显然大于 T 的字符长度 N.因此,对于此类算法,定理成立.

(2) 对 T 不进行预处理.假设存在一种不进行预处理的匹配算法 A,对于(Σ,T)在任何情况下正确完成匹配所需要的字符匹配次数 Cmp 都小于 T 的字符长度 N,即 $Cmp<N$.由于 $Cmp<N$,显然在匹配过程中,算法 A 对 T 内的部分字符没有进行匹配,即算法 A 在匹配过程中至少进行了 1 次跳跃.不妨令算法 A 跳过的字符为 $T[i,j](1\le i\le j\le M)$,并且假设在跳跃之前没有发生错位.根据假设可知,$j-i\ge 1$,$T[i]$, $T[i+1],\ldots,T[j]$均为随机字符,跳跃后从 $T[j+1]$开始下一次匹配,则有如下两种情况:

(a) $j-i$ 为偶数.由于 $T[i],T[i+1],\ldots,T[j-1]$为随机字符,那么,其中出现的英文字符个数 $enum$ 也是随机的,既可能为奇数也可能为偶数.当 $enum$ 为奇数时,则 $T[j]$必为英文字符或某个中文字符的高字节.然而当 $T[j]$为某个中文字符的高字节时,$T[j+1]$为该中文字符的低字节,从 $T[j+1]$处开始匹配显然会带来一连串的匹配错位,所以在此情况下,算法不能保证不出现字节错位,即不能保证在任何情况下都进行正确的匹配.

(b) $j-i$ 为奇数.在此情况下,要保证在 $T[j+1]$开始进行匹配不产生错位,则在 $T[i],T[i+1],\ldots,T[j-1]$之中必须存在奇数个英文字符,这样,$T[j+1]$才有可能是英文字符或中文字符的高字节,从 $T[j+1]$开始匹配才不会造成字节错位的问题,这同样与 T 为随机文本串的前提相矛盾.因此,在此种情况下,同样不能保证进行正确的匹配.

综合上述两种情况,假设不成立,即不存在这样的一种匹配算法,使得对于(Σ,T),在任何情况下正确完成匹配所需要的字符匹配次数小于 T 的字符长度.因此,对于直接匹配类算法,定理成立.

综合上述分析,定理成立. □

定理 2. 在中英文混合环境下,在不对待匹配文本串进行预扫描的情况下,基于跳跃匹配的多模式匹配算法存在漏匹配或误匹配的情况.

证明:在中英文混合环境下,待匹配文本串均可能为同时含有中英文字符的随机文本串.基于跳跃匹配的算法在匹配过程中会跳过文本串中的一些字符,属于亚线性匹配算法,匹配过程中的匹配次数会小于文本中的字符长度.根据定理 1,在中英文混合环境下,匹配次数小于待匹配文本串的字符长度时会出现匹配字节错位问题,造成漏匹配或误匹配. □

3 基于完全哈希 Trie 的多模式匹配算法

本节给出 THT 算法的具体实现.Trie 结构是一种深度可变的多层树型索引结构,它采用宽度优先搜索法,在同一层叶子结点上从左到右逐个查找,查到相匹配项后,再转入下一层继续查找.在匹配过程中,查找路径为从根到叶子的一次查找,具有与 DFSA 相似的查找效率,但常规 Trie 匹配结构在叶子结点处要进行遍历查找,会降低匹配速度.本文对常规 Trie 结构进行扩展,将所有 Trie 叶子结点均设成大小为 256 的完全哈希表,以模式串字符的内码为键值构造完全哈希 Trie 匹配机.在完全哈希 Trie 匹配机中,中文字符用两级相邻的叶子结点表示,英文字符用一级叶子结点表示,用特殊字符表示模式串的结束.如汉字“华”的内码为 0xBBAA,所对应的完全哈希 Trie 匹配机结点如图 2 所示.由于单字节的最大值为 255,在查找过程中可以实现完全哈希查找,不需要对汉字的高低字节进行任何附加运算,因而具有极高的查找效率;由于分别对中文字符的高低字节进行构造,因而不存在空间膨胀问题.

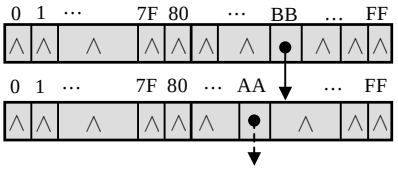

Fig.2 Tree node of Hash trie

图 2 完全哈希 Trie 树结点

完全哈希 Trie 匹配机的构造以字节为最小单位,能够将中文模式串与英文模式串构造在同一 Trie 匹配机中,具有良好的中英文兼容性.例如,根据模式串集合{Chinese,中华,中国人}构造的完全哈希 Trie 匹配机如图 3 所示,其中,“中”、“华”、“国”与“人”的内码分别为 0xD6D0,0xAABB,0xB9FA 与 0xC8CB.

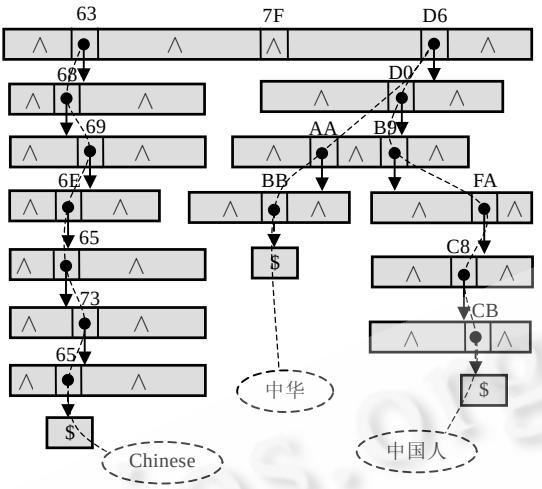

Fig.3 Sketch of Hash Trie matching machine

图 3 全哈希 Trie 匹配机示意图

与英文字符不同,中文字母表内字符的个数较多,在匹配过程中,首字符匹配失败的概率较大,可以使用首字符完全哈希表进行匹配加速,即只有首字符匹配成功以后,再进入后面的完全哈希 Trie 匹配机进行匹配.首字符完全哈希表使用一个 256×256 的两维数组 *head_index*[*Hbyte*][*Lbyte*]来实现,在匹配过程中,以中文字符内码的高低字节直接作索引,获得了较快的匹配速度.

构造完全哈希 Trie 匹配机所需要的数据结构及构造算法伪码见算法 1.由定理 2 可知,在中英文混合环境下,带有跳跃匹配的算法都可能产生误匹配或漏匹配,因此,首先给出基于完全哈希 Trie 匹配机的蛮力(brute force)多模式匹配算法 HT(Hash Trie).算法在匹配前事先判断字符的种类,匹配失败或成功后,匹配指针仅比上一次匹配的初始位置前进 1 个字符.由于匹配指针在匹配中要进行回溯,因此,蛮力匹配算法效率不高,但可以避免误匹配或漏匹配的情况.HT 算法伪码见算法 2.

算法 1. 完全哈希 Trie 匹配机构造算法.

```
typedef struct Head_Hash_Table {      //首字符哈希表
    struct Trie_Node   *head;
} Head_Index[CODE_LEN][CODE_LEN];
typedef struct Trie_Node {            //完全哈希 Trie 结点
    struct Trie_Node   *next[CODE_LEN];
} Trie_Node;
for (i=0;i<key_sum;i++){
    if (kw[i][0]>128) {               //中文模式串
        len=strlen(kw[i]);
        if (!head_index[kw[i][0]][kw[i][1]].head) {
            p_1=malloc(sizeof(struct Trie_Node));
            head_index[kw[i][0]][kw[i][1]].head=p_1;}
        else
            p_1=head_index[kw[i][0]][kw[i][1]].head;
        for (k=2;k<len;k++){
            if (k<len−1){
```

```
                    if (!p_1→next[kw[i][k]]){
                        p_2=malloc(sizeof(struct Trie_Node));
                        p_1→next[kw[i][k]]=p_2;p_1=p_2;}
                    else p_1=p_1→next[kw[i][k]];}
                else p_1→next[kw[i][k]]=END_FLAG;}}
    else {                          //英文模式串,处理与中文模式串相似}
}
```

算法 2. TH 匹配算法伪码.

```
for (i=0;i<strlen(Text);){
    if (T[i]>128){                  //待匹配字符为中文字符
        fwd_num=2;
        p=head_index[T[i]][T[i+1]].head;
        if (p&&p→next[T[i+2]]) {
            p=p→next[T[i+2]]→next[T[i+3]];
            m=4;
            if (End_FLAG==p) PRINT kw;      //匹配成功
            else
                while (NULL!=p){
                    if (End==p) {PRINT kw;break;}
                    if (p→next[T[i+m]]) {p=p→next[T[i+m]]→next[T[i+m+1]];m+=2;}
                    else break;     //匹配失败}}}
    else {                          //英文模式串匹配,与中文相似
        fwd_num=1;}
    i+=fwd_num;
}
```

4 完全哈希 Trie 匹配机的线索化

上一节给出的蛮力匹配算法在匹配成功或失败后,匹配指针要进行回溯.也就是说,待匹配文本串中的一些字符需要进行两次或两次以上的匹配,这无疑会降低匹配速度.从匹配过程可以看出,这些回溯是可以避免的,因为无论匹配成功还是失败,都可以利用已经得到的部分匹配结果来判断下一次的匹配情况,从而使匹配指针可以从当前位置继续向后匹配,而不需要回溯.此种处理类似于 KMP 算法[12],但由于 KMP 为单模式匹配算法,因此又与其有所区别.由于模式串集合是已知固定的,可以通过预处理对完全哈希 Trie 匹配机进行线索化来记录本趟匹配结束后下一趟匹配的情况,从而使得匹配指针不需要回溯,提高了匹配速度.

以模式串集合{服务社会,人民日报,为人民服务}为例,其线索化如图4所示,图中弯曲的细线为线索.当模式串“为人民服务”在第 4 个字符即“服”处匹配失败后,应该转移至模式串“人民日报”的第 3 个字符“日”处继续进行匹配;当模式串“为人民服务”匹配成功后,应转移到模式串“服务社会”的第 3 个字符“社”处继续进行匹配.可以看出,对完全哈希 Trie 匹配机的线索化可以避免匹配指针的回溯.线索化主要有两个方面:一方面是匹配失败的情况,另一方面是匹配成功的时候.

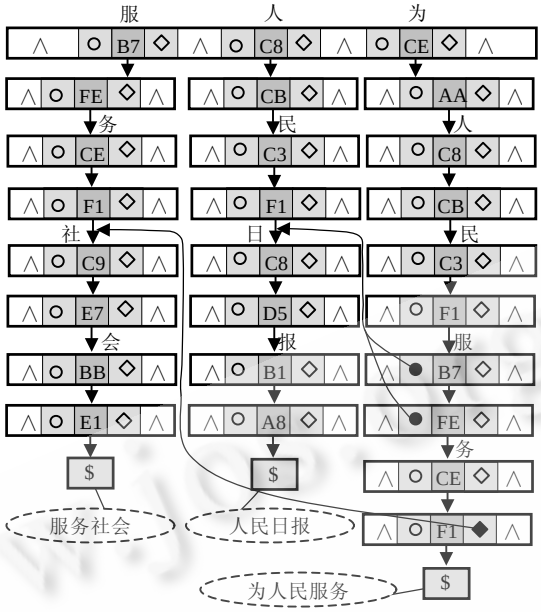

Fig.4 Sketch of threaded Hash Trie matching machine

图4 线索化完全哈希Trie树示意图

下面给出完全哈希 Trie 匹配机的线索化过程.中英文模式串的 Trie 匹配机构造及线索化过程是相同的,在此仅以中文模式串为例进行论述.假设中文模式串集合$\Sigma=\{p_1,p_2,\ldots,p_{sum}\}$,其中,模式串的长度分别为 $len_1,len_2,\ldots,len_{sum}$, p_k^j 表示模式串 p_k 内的第 j 个字符.假设当前匹配字符为 p_k^j,令 *findex*[*k*][*j*]与 *sindex*[*k*][*j*]分别表示在 p_k^j 处匹配失败与匹配成功后,需要转向继续进行匹配的位置.线索化过程实质上就是*findex*[*k*][*j*]与 *sindex*[*k*][*j*]的求值过程.

首先是匹配失败线索 *findex*[*k*][*j*]的计算.在 p_k^j 处匹配失败后,有如下的情况:

(1) 当 $j=1$ 时,此处的匹配失败实际就是首字符匹配失败,*findex*[*i*][*j*]的值为(0,0).此处值为(0,0)表示不需要进行转移,直接从当前匹配位置的下一个相邻字符重新开始匹配.

(2) 当 $1<j\le len_k$ 时,需要判断已经匹配成功的 $[p_k^1, p_k^2,..., p_k^{j-1}]$ 的后缀子串集合内是否存在模式串 p_m $(1\le m\ne k\le sum)$的前缀子串.如果不存在,则 *findex*[*k*][*j*]=(0,0);如果存在 $[p_k^n, p_k^{n+1},..., p_k^{j-1}](1\le n\le j-1)$ 为 p_m 的最长前缀子串,则 *findex*[*k*][*j*]=(*m*,*j*−*n*),其中,(*m*,*n*)表示从模式串 p_m 的第 *n* 个字符开始匹配.

综合上述分析,*findex*[*k*][*j*]的值可以用公式(1)进行计算.计算 *findex*[*k*][*j*]的伪码见算法 3.

$$findex[k][j]=\begin{cases}(0,0), & j=1\\ (0,0), & 1<j\le len_k \text{且}\\ & [p_k^n, p_k^{n+1},..., p_k^{j-1}]\ne[p_m^1, p_m^2,..., p_m^{j-n}]\\ (m, j-\min(n|[p_k^n, p_k^{n+1},..., p_k^{j-1}]=[p_m^1, p_m^2,..., p_m^{j-n}])), & 1<j\le len_k\end{cases} \tag{1}$$

算法 3. 匹配失败线索计算算法.

```
for (k=1;k<=sum;k++){
    for (n=1;n<len_k;n++){
        if (n==1)findex[k][n]=(0,0);
        flag=NOFIND;
        for (t=2;t<n;t++){
```

```
            stemp=[p_k^t, p_k^{t+1},..., p_k^{n-1}];
            for (m=1;m<=sum;m++){
                if (m==k) continue;
                dtemp=[p_k^t, p_k^{t+1},..., p_k^{n-1}];
                if (0==strncmp(stemp,dtemp)){
                    findex[k][n]=(m,n-t);
                    flag=FIND;break;}}
            if (flag==FIND) break;}
        if (flag==NOFIND)findex[k][n]=(0,0);}
}
```

其次是匹配成功的情况,当模式串 p_k 匹配成功后,后续的匹配存在如下两种情况:

(1) p_k 的任一后缀都不是 Σ 中其余任一模式串的前缀,应直接从当前匹配位置的下一个字符重新开始匹配;

(2) p_k 的长度为 n 的最长后缀为模式串 p_m 的前缀,则转向 p_m 的第 n+1 个字符处进行匹配.

sindex[*k*]的值可以用以下公式进行计算:

$$sindex[k]=\begin{cases}(0,0), [p_k^n, p_k^{n+1},..., p_k^{len_k}]\neq[p_m^1, p_m^2,..., p_m^{len_k-n+1}] \mid (1\le n\le len_k)\\ (m, len_k-\min(n\mid[p_k^n, p_k^{n+1},..., p_k^{len_k}]=[p_m^1, p_m^2,..., p_m^{len_k-n+1}]))\end{cases} \tag{2}$$

计算 *sindex*[*k*]的算法与计算 *findex*[*k*][*j*]的算法 3 相似,在此不再给出算法伪码.

THT 的匹配过程伪码见算法 4.

值得说明的是,由于算法 4 中的 *findex*[*kw.num*].*n* 是以字节为度量最小单位,而在式(1)、式(2)中的 *n* 是以字符为最小单位,因此,在具体程序中对 *findex* 进行初始化时应进行转换,并要区分中英文模式串.

算法 4. THT 匹配算法伪码.

```
for (i=0;i<strlen(T);){
  if (T[i]>128) {       //Chinese word
    p_1=head_index[T[i]][T[i+1]].head;
    if (p_1&&p_2=p_1→next[T[i+2]]){
      if (!p_1=p_2→next[T[i+3]])   i+=2;
      else {
        m=4;
        while (p_1) {
          if (END_FLAG==p_1) {
            PRINT kw;
            If (sindex[kw.num].index==NULL) {i+=m;break;}
            Else {
              p_1=sindex[kw.num].index;i=i+m-sindex[kw.num].n;
              m=sindex[kw.num].n;}}
          if (p_2=p_1→next[text[i+m]]){
            p_1=p_2→next[text[i+m+1]];
            if (p_1==NULL) {
              if (findex[kw.num].index==NULL){i=i+m;break;}
              else {p_1=findex[kw.num].index;i=i+m-findex[kw.num].n;m=findex[kw.num].n;}}
            m+=2;}
          else {
```

```
                if (findex[kw.num].index==NULL) {i=i+m;break;}
                else {p_1=findex[kw.num].index;i=i+m–findex[kw.num].n;m=findex[kw.num].n;}
    }}}}
    else i+=2;}
else {      //English char}
}
```

5 算法性能分析

假设待匹配中英文混合的随机文本串 T 的字符长度为 $Twlen$,字符长度为 $Tblen$;模式串集合$\Sigma=\{p_{e1},p_{e2},\ldots,p_{eM},p_{c1},p_{c2},\ldots,p_{cN}\}$,其中,$p_{em}(1\le m\le M)$为英文模式串,其字符长度为 len_{em};$p_{cn}(1\le n\le N)$为中文模式串,其字符长度为 len_{cn};Ψ为所有模式串中字符的集合.对 THT 算法而言,Trie 匹配机构造算法的空间复杂度与匹配算法的时间复杂度至关重要,两者决定了算法的可行性与可用性,性能分析主要集中在这两个方面.

5.1 Trie构造算法性能分析

从算法 1 可以看出,构造Σ的完全哈希 Tire 匹配机算法的时间复杂度为 $O(\Sigma len_{em}+\Sigma len_{cn})$.构造算法运行于整个算法的初始化阶段,其空间复杂度是主要的,它决定了算法在实践中是否可行.为分析构造算法的空间复杂度,进行以下定义.

最大模式串长度 MAXLEN,

$$\text{MAXLEN}=\max(len_{em}|1\le m\le M,2\times len_{cn}|1\le n\le N) \tag{3}$$

两个辅助参数 fe_{mi},fc_{ni},

$$fe_{mi}=\begin{cases}1, & 1\le i\le len_{em}\\ 0, & i>len_{em}\end{cases},\ fc_{ni}=\begin{cases}1, & 1\le i\le 2len_{cn}\\ 0, & i>2len_{cn}\end{cases} \tag{4}$$

构造算法采用首字符双字节完全哈希表,占用 64K 空间,完全哈希 Trie 匹配机从模式串的第 2 个字符开始构造,因此,算法 1 所需要的空间Ω为

$$\Omega=64K+\sum_{i=2}^{\text{MAXLEN}}\left(256\times\sum_{m=1}^{M}fe_{mi}\right)+\sum_{i=3}^{\text{MAXLEN}}\left(256\times\sum_{n=1}^{N}fc_{ni}\right) \tag{5}$$

由式(5)可以看出,在最坏情况下,即所有模式串的字符长度均为 MAXLEN,且任两个模式串之间不存在相同前缀子串,算法 1 使用的空间为

$$\Omega_{\max}=256\times256+256\text{M}(\text{MAXLEN}-1)+256\text{M}(\text{MAXLEN}-2) \tag{6}$$

定理 3. 在中文环境下,对于相同的模式串集合,完全哈希 Trie 构造算法在最坏情况下的使用空间远小于 DFSA 算法中状态转换完全哈希表所需空间.

证明:由于空间膨胀问题主要针对中文字符而言,因此不妨假设模式串集合内只有中文模式串,则最坏情况下完全哈希 Trie 构造算法使用空间$\Omega_{\max}$为

$$\Omega_{\max}=256\times(256+N\times\text{MAX}LEN-2N) \tag{7}$$

DFSA 算法中状态转换函数完全哈希表使用空间Ω_{DFSA}为 $256\times256\times sumof(state)$,则有

$$\frac{\Omega_{\max}}{\Omega_{DFSA}}=\frac{N\times\text{MAX}LEN}{256\times sumof(state)}+\frac{256-2N}{256\times sumof(state)} \tag{8}$$

当模式串数量 N 较大时,显然有 $N<<sumof(state)$,又因为 MAXLEN<<256,所以在模式串数量较多时有$\Omega_{\max}/\Omega_{DFSA}<<1$,即$\Omega_{\max}<<\Omega_{DFSA}$. □

不妨假设有 2 500 个中文模式串,500 个英文模式串,最长模式串为 10 个中文字符,即最长为 20 字节,则最坏情况下使用空间为 13.37M.根据文献[13],已知中文词出现频率依次为单字词占 12.11%,双字词占 73.16%,三字词占 7.16%,四字词占 6.14%,多字词占 0.12%,在实际应用中又以中文模式串为主,实际使用关键字的长度远小于 MAXLEN.可以看出,本文算法不存在空间膨胀问题,在实际应用中完全可行.

完全哈希 Trie 匹配机线索化算法的时间复杂度为 $O(sum\times len^2)$,通常,模式串的长度 len 比较小,而且线索化运行于算法的初始化阶段,对整个匹配算法的性能基本上没有影响.

5.2 匹配算法性能分析

通过算法 4 可以看出,THT 算法在匹配过程中匹配指针不需要回溯,最坏情况下,完成对 T 的匹配需要进行的匹配次数为 $Twlen$+MAXLEN,比较次数为 $Tblen$+MAXLEN,因此,算法的时间复杂度为 $O(Tblen$+MAXLEN). 根据定理 1,在中英文混合环境下,THT 算法的匹配次数已经逼近正确匹配所需最少匹配次数的理论下限,因此,算法具有较好的匹配效率.而且,由于中文字符为大字符集,经过首字符高字节的匹配后,匹配失败的概率很大,因此,在实际的使用中,THT 算法的比较次数一般会远远小于待匹配文本的字节长度.

另外,THT 算法的匹配速度与以下因素有关:

(1) 模式串之间相同字符的数量,即集合 Ψ中字符在模式串中出现的次数.相同字符数量越少,所构造 Trie 匹配机内的线索密度越小,在匹配中需要转移的次数也越少,匹配效率也就越高.

(2) 集合 Ψ中字符在文本串 T 中出现的概率.显然,出现的概率越低,经过首字哈希匹配后匹配失败的概率就越大,匹配指针直接后移,匹配所花费的时间也就越少;

(3) 中英文模式串在 Σ中各占的比例以及 T 内中英文各占的比例.中英文语言存在比较明显的差异,如中文语言是大字符集语言,字母表数量庞大,词语长度较短;英文语言的字母表小,词比较长等.这些差异使得在大多数情况下,T 内中文字符都不属于 Ψ,而且即使属于 Ψ,匹配成功或失败后需要匹配转移的概率也小;对于英文则恰恰相反.所以,当模式串集合内英文模式串较少、待匹配文本串内英文字符较少时,算法匹配速度较快.

6 实验与分析

6.1 实验文本与结果

本节给出了实验结果.实验所使用的服务器为 IBM 的 eServer,处理器为 Intel(R) Xeon(TM) CPU 2.00GHz,内存为 512M,硬盘为 40G,操作系统为 RedHat 7.2,内核版本为 2.4.20.

实验所使用的测试文本有 4 组,分别为 2004 年 5 月《人民日报》原文(4.64M,字节长度为 4 869 842,字符长度为 2 575 632,http://www.people.com.cn/GB/paper464/review/200405.html,以下简称文本 1)及将原文内的英文字符去除后得到的纯中文文本(以下简称文本 2,4.38M);搜狐研发中心搜狗实验室的文本分类语料(211.96M,字节长度 222 259 691,字符长度为 129 371 202,http://www.sogou.com/labs/dl/c.html,以下简称文本 3)及将文本 3 中的英文字符去除后得到的纯中文文本(以下简称文本 4,177.17M).

从文本 1 中切分出 5 组中文关键词,关键词的数量分别为 500,1 000,1 500,2 000 和 2 500,每组词平均长度均为 3.7 个汉字;切分出 5 组英文关键词,数量分别为 10,20,30,40,50.由于单字汉字的语义大多不明确,在实际应用中设置单字关键词的情况较少,因此,在实验过程中没有设置单字关键词.

3 种算法所使用的空间见表 1,其中,Shen,Gao 分别代表文献[6]、文献[9]中的算法.由于文献[7]中加标记的算法需要对待匹配文本进行预扫描处理,然后才进行匹配,匹配时间明显大于其他算法,因此,本文没有与文献[7]算法进行对比实验.为了对本算法的理论值与实际使用的空间进行对比,假定最长关键词长度为 10 个汉字,英文字符为 20 个字符,在表 1 中同时给出了 THT 算法的理论使用空间.

Table 1 Space used by three algorithms

表 1 各算法空间性能比较

Number of keywords		Space used (Mb)			
Chinese	English	Gao	Shen	THT	THT theoretical value
500	10	1.26	0.78	0.64	2.31
1000	20	3.22	1.43	1.17	4.55
1500	30	5.69	2.06	1.67	6.79
2000	40	8.26	2.67	2.15	9.04
2500	50	10.66	3.26	2.61	11.28

首先使用文本 2、文本 4 做对比实验.由于算法 Shen 只适用于纯中文环境,使用文本 2、文本 4 可以保证每种算法都能进行正确匹配,在此种情况下进行效率对比才有意义.文本 2、文本 4 为纯中文文本,在匹配中不需要区分字符的种类,在算法 Gao,HT 及 THT 中将不进行字符种类的判断.这样,4 种算法所进行的字符处理是一致的,匹配使用的时间反映了算法匹配效率的高低.在实验过程中,4 种算法均能正确完成查找,文本 2 的 5 组实验匹配次数为 11 067,18 525,22 987,30 210,50 181;文本 4 的 5 组实验匹配次数为 379 016,451 942,609 310,941 736,1 636 960;各种算法所使用的时间见表 2.

Table 2 Matching time of algorithms on Chinese texts

表 2 纯中文字符环境下各算法时间性能比较

Number of keywords	Text two (s)				Text four (s)			
	Gao	Shen	HT	THT	Gao	Shen	HT	THT
500	0.144	0.080	0.062	0.067	5.467	2.825	2.101	2.508
1 000	0.189	0.113	0.082	0.089	6.943	3.908	2.926	3.481
1 500	0.218	0.136	0.096	0.099	8.169	4.741	4.159	3.937
2 000	0.245	0.156	0.113	0.109	9.124	5.580	4.679	4.401
2 500	0.259	0.184	0.130	0.121	9.850	6.412	5.411	4.926

使用文本 1、文本 3 评测 Gao,Shen,HT 及 THT 这 4 种算法在中英文混合环境下的性能,性能对比见表 3.Gao,HT 及 THT 算法均能正确完成查找,文本 1 的 5 组实验匹配次数为 11 132,18 714,23 078,30 507,50 639;文本 4 的 5 组实验匹配次数为 379 361,452 426,609 903,942 643,1 638 205.在中英文混合环境下,算法 Shen 不能进行正确匹配,文本 1 的 5 组实验匹配次数为 5 683,9 394,11 805,15 426,31 092;文本 3 的 5 组实验匹配次数为 189 328,224 678,314 580,489 099,1 053 692.

Table 3 Matching time of algorithms on Chinese/English mixed texts

表 3 中英文混合环境下各算法性能比较

Number of keywords	Text one (s)			Text three (s)		
	Gao	HT	THT	Gao	HT	THT
500	0.188	0.089	0.100	6.012	2.602	3.057
1000	0.252	0.128	0.146	7.803	3.418	3.502
1500	0.300	0.189	0.165	9.129	4.791	3.925
2000	0.337	0.228	0.206	10.172	5.279	4.432
2500	0.359	0.274	0.256	10.887	6.038	5.174

表 4 为算法 HT 与 THT 算法在实际运行中的比较次数.

Table 4 Comparing times of HT and THT

表 4 HT 与 THT 实际运行中的比较次数

Number of Chinese keywords	Text one		Text three	
	HT	THT	HT	THT
500	2866058	2948690	143601794	145337870
1000	3169592	3211740	150431290	151107295
1500	3327754	3359030	154951714	153798596
2000	3597419	3486754	160450458	158135882
2500	3947920	3558062	176503172	163137132

6.2 实验结果分析

(1) 空间使用方面

从表 1 可以看出,本文算法使用空间小于算法 Gao 及算法 Shen;随着模式串数量的增加,算法 Gao 使用空间增加,成超线性关系;本文算法与算法 Shen 的空间增加基本上与模式串数量增加,成亚线性关系.这主要是因为算法 Gao 使用状态完全哈希表,随着模式串数量的增加,状态机的状态数量增加较快.本文算法与算法 Shen 优于算法 Gao,不存在存储空间增长过快的问题;对 THT 算法而言,实际运行使用空间远远小于理论计算值,与性能分析的结论相吻合.

(2) 匹配时间方面

从表 2 可以看出,在纯中文环境下,THT 使用时间少于算法 Gao,Shen,5 组实验使用的时间分别为算法 Gao 的 46.53%,47.09%,45.41%,44.49%,46.72%以及算法 Shen 的 83.75%,78.76%,72.79%,69.87%,65.76%,其时间性能优于其他两种算法;对于不同的测试文本,HT 与 THT 两种算法时间的变化规律是一致的,即在关键词数量较少时,HT 算法所使用的时间较少,但随着关键词数量的增加,THT 算法的性能优于 HT.其主要原因是,在关键词数量较少的情况下,由于关键词之间的重合并不太多,因此线索化的效果并不明显,而且线索化所产生的额外判断导致所使用时间反而比 HT 算法要稍高;但随着关键词数量的增加,线索化的作用逐渐明显,相应地,THT 的时间会小于 HT 算法.因此对 THT 而言,最适合于关键词数量非常大的情况.表 4 显示,由于中文为大字符集,首字符高字节匹配失败的概率很大,因此在实际运行过程中,HT 与 THT 的实际比较次数远远小于待匹配文本的字节长度,这与理论分析相吻合.

(3) 混合环境下的匹配性能

如表 3 所示,在混合环境下,本文算法与算法 Gao 均能进行正确的匹配,所使用的时间均略高于纯中文环境下的匹配时间,这是因为在算法中增加了区分字符种类的运算;算法 Shen 不能进行正确的匹配,文本 1 的 5 组匹配次数仅为正确匹配次数的 51.05%,50.20%,51.15%,50.57%,61.40%,文本 3 的 5 组匹配次数仅为正确匹配次数的 49.91%,49.66%,51.58%,51.89%,64.32%;本文算法 THT 使用时间与算法 Gao 相比,呈现出与纯中文环境下相同的规律;在线索化方面,HT 与 THT 相比所呈现的规律与纯中文环境一致,结果表明,在关键词集较大时,THT 的性能优于 HT,能够充分发挥线索化的作用.

综合上述分析,本文算法在空间与时间性能方面优于算法 Gao,Shen,适合于中英文混合的环境.

7 结 论

本文分析了在中英文混合环境下进行多模式匹配的特点,给出并证明了适用于中英文混合环境的多模式匹配算法的性能定理,并分析了已有多模式匹配算法应用于中英文混合环境时的不足,提出了一种基于线索完全哈希 Trie 匹配机的适合于中英文混合环境的多模式匹配算法.该算法以汉字内码中的高低字节为键值构造完全哈希 Trie 匹配机,并根据模式串集合的特点对 Trie 匹配机进行线索化,使得匹配算法在匹配过程中指针不需要回溯,有效地提高了算法的匹配效率,降低了算法的时间与空间复杂度.理论分析和实验结果表明,本文算法能够在中英文混合的环境中避免误匹配和漏匹配,匹配速度明显优于已有的算法,而且不存在空间膨胀问题.

在后续工作中,我们将会在以下两个方面作进一步研究:如何对模式串进行前期处理,确定模式串的最佳排列顺序,使得构造的线索完全哈希 Trie 匹配机性能最好;将完全哈希 Trie 结构应用于中英文混合环境下的多模式相似匹配.

致谢 在此,我们向对本文提出有益建议的审稿专家们表示感谢!

References:

[1] Aho AV, Corasick MJ. Efficient string matching: An aid to bibliographic search. Communications of the ACM, 1975,18(6): 333−340.

[2] Fan JJ, Su KY. An efficient algorithm for matching multiple patterns. IEEE Trans. on Knowledge and Data Engineering, 1993,5(2): 339−351.

[3] Muth R, Manber U. Approximate multiple string search. In: Proc. of the 7th Annual Combinatorial Pattern Matching Symp. Berlin: Springer-Verlag, 1996. 75−86.

[4] Wu S, Manber U. A fast algorithm for multi-pattern searching. Technical Report, TR-94-17, University of Arizona, 1994.

[5] Sunday DM. A very fast substring search algorithm. Communications of the ACM, 1990,33(8):132−142.

[6] Shen Z, Wang YC, Xu YZ. A fast multiple pattern algorithm for chinese string matching. Journal of Shanghai Jiaotong University, 2001,35(9):1286−1289 (in Chinese with English abstract).

[7] Wong KF. String matching on Chinese/English mixed texts. Int'l Journal of Computer Processing of Chinese and Oriental Languages, 1996,10(1):115−126.

[8] Wong KF, Lum VY, Lam W. Chicon — A Chinese text manipulation language. Software—Practice & Experience, 1998,28(7):681−701.

[9] Gao P, Zhang DY, Sun QD, Zhai YH, Lu WC. A multiple approximate string matching algorithm of network information audit system. Journal of Software, 2004,15(7):1074−1080 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1074.htm

[10] Yin LH, Fang BX. An improved algorithm for multiple patterns matching. Journal of Huazhong University of Science and Technology, 2005,33(S1):300−303 (in Chinese with English abstract).

[11] Wang GG, Qin ZhG. Improved AC-BM algorithm for matching multiple strings. Journal of University of Electronic Science and Technology of China, 2006,35(4):531−533 (in Chinese with English abstract).

[12] Knuth DE, Morris JH, Pratt VR. Fast pattern matching in strings. SIAM Journal on Computing, 1977,6(2):323−350.

[13] Wang YC. The Technique and Basis of Chinese Information Processing. Shanghai: Shanghai Jiao Tong University Press, 1990. 30−31 (in Chinese).

附中文参考文献:

[6] 沈洲,王永成,许一震.一种面向中文的快速字串多模式匹配算法.上海交通大学学报,2001,35(9):1286−1289.

[9] 高鹏,张德运,孙钦东,翟亚辉,卢伍春.网络信息审计系统中的多模式相似匹配算法.软件学报,2004,15(7):1074−1080. http://www.jos.org.cn/1000-9825/15/1074.htm

[10] 殷丽华,方滨兴.一种改进的多模式匹配算法.华中科技大学学报(自然科学版),2005,33(S1):300−303.

[11] 万国根,秦志光.改进的 AC-BM 字符串匹配算法.电子科技大学学报,2006,35(4):531−533.

[13] 王永成.中文信息处理技术及其基础.上海:上海交通大学出版社,1990.30−31.

孙钦东(1975－),男,山东莒南人,博士,副教授,CCF 会员,主要研究领域为网络安全,嵌入式系统.

黄新波(1975－),男,博士,副教授,主要研究领域为无线网络.

王倩(1979－),女,助理工程师,主要研究领域为网络安全,图像处理.