基于KMP算法的改进算法KMPP

李 莉[1],江育娥[1],林 劼[1],江秉华[2]
LI Li[1], JIANG Yu'e[1], LIN Jie[1], JIANG Binghua[2]

1.福建师范大学 软件学院,福州 350100
2.南京医科大学 病理系,南京 210029
1.Faculty of Software, Fujian Normal University, Fuzhou 350100, China
2.Department of Pathology, Nanjing Medical University, Nanjing 210029, China

LI Li, JIANG Yu'e, LIN Jie, et al. Improved algorithm KMPP based on KMP. Computer Engineering and Applications, 2016, 52(8):33-37.

Abstract: KMP algorithm and BM algorithm are classical single pattern matching algorithms, because the pointer i in the text can only move one character at each time in KMP algorithm, the overall matching efficiency is not high, the improved algorithm(KMPP) accordingly to combine the advantages of the KMP algorithm with BM algorithm together is proposed. The core of KMPP algorithm is when a mismatch occurs at position j, it calculates the position in the text where the last character of pattern will align with after the pattern moving the value of $next[j]$. If the last character of pattern does not match with the corresponding text position, the bad character rules are applied. The moving distance of pointer i in the text is a two-step jumping distance, thus the pointer can move farthest at each time. The experimental result shows that the comparison number of KMPP algorithm is far less than the KMP algorithm's number and it improves the efficiency of pattern matching algorithm.

Key words: pattern matching; KMP algorithm; BM algorithm; KMPP algorithm

摘 要: KMP算法和BM算法是经典的单模式匹配算法,但KMP算法中文本指针i每次只能移动一个字符,整体的匹配效率并不高,结合KMP算法和BM算法的优点提出一种改进算法(KMPP)。算法的思想是模式串与文本在j处不匹配时,预算出模式串移动$next[j]$后末字符在文本中的位置,当该位置的文本字符与末字符不匹配时,则用该字符进行坏字符匹配,这两步的跳跃距离就是文本指针i移动的距离,从而使指针i每次移动的距离达到最大。实验结果表明,该算法匹配次数远低于KMP算法的匹配次数,提高了模式匹配的效率。

关键词: 模式匹配;KMP算法;BM算法;KMPP算法
文献标志码: A **中图分类号:** TP311 doi:10.3778/j.issn.1002-8331.1405-0426

1 引言

在当前大数据的时代,无论是金融、文学、生物信息还是计算机领域,文本都是必不可少的信息组成元素。面对不断出现的大量文本,如何快速精确地找到文本中需要的信息成为研究的重点。

目前的互联网正面临着越来越严重的网络安全问题,网络入侵涉及到网络信息的保密性、完整性、可用性、真实性和可控性,因此入侵检测技术成为当前的研究热点,而精确字符匹配算法效率的提升对提高网络入侵检测系统(NIDS)的性能起到很大的作用。随着各个

基金项目: 国家自然科学基金重大国际(地区)合作研究项目(No.81320108019);福建省自然科学基金(No.2014J01220)。
作者简介: 李莉(1991—),女,硕士,研究方向为软件工程、数据挖掘,E-mail:529396211@qq.com;江育娥(1970—),女,博士,教授,研究领域为数据挖掘;林劼(1972—),通讯作者,男,博士,副教授,研究领域为序列算法、生物信息学、数据挖掘;江秉华(1962–),男,博士,教授,研究领域为分子生物学、生物信息学。
收稿日期: 2014-06-03 **修回日期:** 2014-08-14 **文章编号:** 1002-8331(2016)08-0033-05
CNKI**网络优先出版:** 2014-12-11, http://www.cnki.net/kcms/detail/11.2127.TP.20141211.1526.043.html

领域的关联性的提高,产生的文本数据量越来越大,很多领域都需要在大量信息中查找特定的信息。例如生物信息领域中的DNA测序定位、航海领域中的海洋数据查询、计算机领域中的高效搜索引擎等。

由此可见,寻找更有效的字符串匹配算法是当务之急。本文结合经典的单模式匹配算法KMP和BM的优点,提出一种更高效率的匹配算法——KMPP(KMP Plus),该算法不仅仅在匹配次数上远低于经典算法BF与KMP,而且在时间性能上也大大提高了,KMPP算法针对文本查找信息的效率有很大的提高。

2 相关算法分析

在介绍相关的算法前,先作以下的定义:文本表示为 $T=T[0,1,\cdots,n-1]$,长度为 n;模式串表示为 $P=P[0,1,\cdots,m-1]$,长度为 m;并且满足条件 $n\geqslant m$。如果存在 i 使得:$T[i]=P[0],T[i+1]=P[1],\cdots,T[i+m-1]=P[m-1]$,那么模式串 P 就出现在文本 T 的 i 处。单模式匹配问题就是找出文本 T 中所有匹配的模式串 P 的起始位置。

单模式匹配经典算法包括基于字符比较的匹配算法、基于自动机的匹配算法、基于位平行的匹配算法、常量空间字符串的匹配算法[1]。本文论述的算法是基于字符比较的单模式匹配算法。基于字符比较的主要匹配算法有BF(Brute-Force)算法[2]、KMP(Knuth-Morris-Pratt)算法[3]、BM(Boyer-Moore)算法[4]等,其改进算法主要有BMH算法[5]、QS算法[6-7]、AKC算法[8]、针对next函数进行修改的KMP改进算法(下文称为NKMP算法)[9]等。本文主要是在KMP算法的基础上,提出新的改进算法KMPP(KMP Plus),并将KMPP与KMP、BM、NKMP算法的性能做了对比,理论与实验均证明KMPP的性能高于BM、KMP、NKMP算法。

2.1 KMP算法

1969年Knuth、Morris和Pratt提出快速单模式匹配算法——KMP算法[3]。KMP算法消除了BF算法中的指针 i 回溯问题。比较过程中模式串是从左往右移动,字符比较也是从左往右比较。若 $T[i]=P[j]$,则继续比较 $T[i+1]=P[j+1]$;若 $T[i]!=P[j]$,则 i 值不变,j 值等于 $next[j]$,再进行下一轮的匹配。其中 $next[j]$ 的值表示 $P[0,1,\cdots,j-1]$ 中最长后缀的长度,且这个最长后缀等于相同字符序列的前缀,对next数组的定义如下:

$$Next(j)=\begin{cases}-1, \text{当}j=0\text{时}\\ \max\{k|0<k<j, \text{且}P_0P_1\cdots P_{k-1}=\\ \quad P_{j-k}P_{j-k+1}\cdots P_{j-1}\text{存在}\}\\ 0, \text{其他情况}\end{cases} \tag{1}$$

在下一次匹配时只需确定 j 的位置即可,从而提高模式匹配的效率,KMP算法的时间复杂度为 $O(m+n)$。表1为KMP算法匹配过程,其中 T="acbccadbacbacc",P="acbacc"。

表1 KMP算法匹配过程

序号	0	1	2	3	4	5	6	7	8	9	10	11	12	13
文本串	a	c	b	c	c	a	d	b	a	c	b	a	c	c
第一次	a	c	b	a	c	c								
第二次				a	c	b	a	c	c					
第三次					a	c	b	a	c	c				
第四次						a	c	b	a	c	c			
第五次							a	c	b	a	c	c		
第六次								a	c	b	a	c	c	
第七次									a	c	b	a	c	c

2001年,复旦大学朱洪教授主要针对KMP匹配算法的预处理next函数进行修改,在计算 $next[j]$ 值时,多加一步判断 $P[next[j]]\neq P[j]$[10]。这种改进算法主要针对的是特殊的字符串,例如 P="aabaabaac",即在模式串中出现 $P[next[j]]=p[j]$ 的次数越多,且文本出现这种模式串越多,提高的效率就越高。NKMP算法的时间复杂度为 $O(m+n)$,2010年杨俊丽也对此改进算法进行分析研究[9]。2006年华东科技大学鲁宏伟教授根据自定义的特征值 k 提出一种新的模式匹配算法。当此改进算法的 k 值几乎为1时,即 $next[j]$ 的值大部分为0时,该算法的提高效率并不高[11]。综合两种改进算法的比较结果,最终采用运行效率更好的NKMP算法进行比较论述。

2.2 Boyer-Moore(BM)算法

1977年Boyer和Moore提出了著名的BM算法[4],BM算法在进行匹配的时候,从文本 T 左边向右移动模式串 P,而模式串 P 与文本 T 在字符比较的时候是从右往左比较。BM算法在匹配的时候要同时根据坏字符跳跃表和好后缀跳跃表,取两者中的最大值作为模式串的右移量。BM算法的最坏时间复杂度为 $O(m\times n)$,最好时间复杂度为 $O(n/m)$。

坏字符跳转规则定义如下,其中 ch 为文本 T 与模式 P 比较时出现的不匹配文本字符:

$$BC_p(ch)=\begin{cases}m; P[j]\neq ch(1\leqslant j\leqslant m), \text{即}ch\text{在}P\text{中未出现}\\ m-j; j=\max\{j|P[j]=ch, 1\leqslant j\leqslant m-1\},\\ \quad \text{其他情况}\end{cases} \tag{2}$$

以文本 T="acbccadbacbacc"和模式串 P="acbacc"为例,文本 T 与模式串 P 从右往左比,当模式串 P 的 j 处字符与文本的 $i+j$ 处不匹配时,即 $T[i+j]!=P[j]$,则移动模式串的距离为 $BCp(T[i+j])$。

好后缀跳跃规则定义如下:

$$GSp(j)=\min\begin{cases}s|(P[j+1,\cdots,m-1]=P[j-s+1,\cdots,\\ \quad m-1-s])\&\&P[j]\neq P[j-s](j>s)\\ P[s+1,\cdots,m-1]=P[1,\cdots,m-s](j\leqslant s)\end{cases} \tag{3}$$

例子的匹配过程见表2所列。

表2　BM算法匹配过程

序号	0	1	2	3	4	5	6	7	8	9	10	11	12	13
文本串	a	c	b	c	c	a	d	b	a	c	b	a	c	c
第一次	a	c	b	a	c	c								
第二次			a	c	b	a	c	c						
第三次						a	c	b	a	c	c			
第四次									a	c	b	a	c	c

3　KMPP算法

3.1　KMPP算法思路

KMP算法消除了BF算法中的文本指针 i 回溯问题,但指针 i 每次只能移动一个字符,而且模式串每次的跳转量都小于当前 j 值,当字符不匹配概率大时, j 值往往很小,所以KMP的整体匹配效率并不高[12]。本文结合KMP算法和BM算法坏字符跳转规则的优点,提出一种改进算法KMPP,该算法可以增加指针 i 每次移动的距离,从而提高匹配效率[13-15],与BM不同,该算法的最坏时间复杂度为 $O(m+n)$,理论上也优于BM算法。

KMPP算法思路如下:

预处理阶段得到两个数组,分别是next数组和bmBc数组。

匹配阶段在遇到 T , P 不匹配时进行两个部分操作。第一部分是计算模式串移动 $next[j]$ 后末字符在文本中的位置;第二部分是比较文本和模式串在该位置的字符是否匹配,如果两字符不匹配,则文本该位置的字符应用bmBC数组进行坏字符跳转,否则按照KMP的匹配过程继续比较。

算法流程如图1。

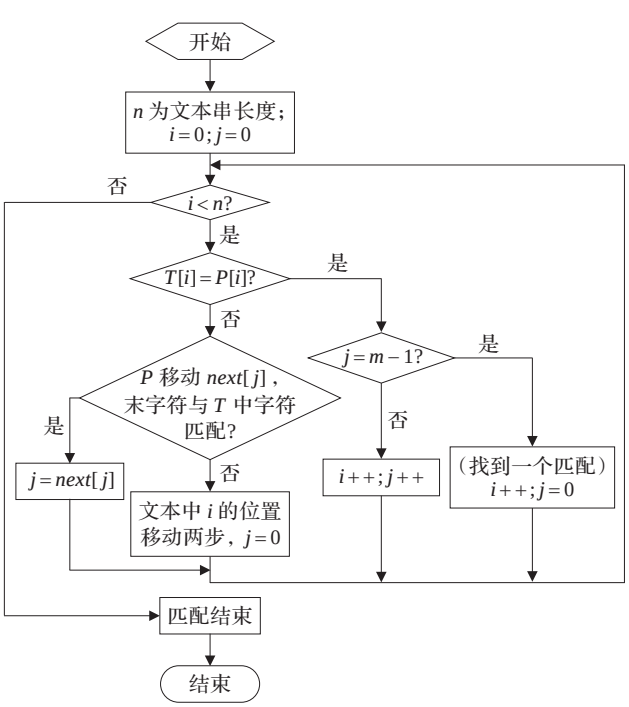

图1　KMPP算法流程图

3.2　预处理阶段

预处理阶段包含两部分:KMP算法预处理和BM算法坏字符规则预处理。KMP算法预处理得到的是next数组,next数组在KMP算法中是用来改变下一轮比较窗口的 j 值,在KMPP算法中,它还起到确定模式串移动 $next[j]$ 后末字符在文本中的位置的作用;运用BM算法中的坏字符跳转规则得到的是bmBc数组,在KMPP算法中匹配阶段的第二部分中,如果两字符不匹配,则利用bmBc数组作文本该位置字符的坏字符跳转。

由于KMP算法预处理的时间复杂度为 $O(m)$,坏字符的时间复杂度为 $O(m+|\Sigma|)$, $|\Sigma|$ 为字符集大小,所以KMPP预处理阶段具有线性的时间复杂度 $O(m+|\Sigma|)$ 。

KMP预处理核心代码如下:

```
while(i<strlen(P))
{
    if(j==-1||(j>=0 && P[i]==P[j]))
    {
        i++;j++;next[i]=j;
    }
    else
        j=next[j];
}
```

坏字符预处理核心代码如下:

```
for(i=0;i<ASIZE;i++)
{
    bmBc[i]=m;
}
for(i=0;i<m-1;i++)
{
    bmBc[x[i]]=m-1-i;
}
```

3.3　匹配阶段

匹配函数核心代码如下:

```
while(i<length)
{
    if(j==-1||T[i]==P[j])
    {
      if(匹配成功)
      {//找到一个匹配
        i++;j=0;
      }
      else
        {i++;j++;}
        }
        else
        {//计算模式串移动next[j]后末字符在文本中的
位置
            tLast=T[i+pLen-next[j]-1];
//该位置的文本字符与末字符不匹配
```

```
            if(tLast!=pLast)
            {
                j=0;
//进行坏字符跳转
                i=i+bmBc[tLast] – next[j];
            }
            else
//按照KMP的匹配过程继续比较
                j=next[j];
        }
}
```

以文本串 T="acbccadbacbacc",模式串 P="acbacc"为例,匹配过程如表3所示;预处理后得到的两个数组值分别为:

$next[6]=[-1,0,0,0,1,2]$

$bmBc[a]=2, bmBc[b]=1, bmBc[c]=3$

表3 KMPP算法匹配过程

序号	0	1	2	3	4	5	6	7	8	9	10	11	12	13
文本串	a	c	b	c	c	a	d	b	a	c	b	a	c	c
第一次	a	c	b	a	c	c								
第二次						a	c	b	a	c	c			
第三次									a	c	b	a	c	c

在第一次遇到不匹配时,先计算出该位置的文本字符为 $tLast=T[i+m-next[j]-1]=T[3+6-0-1]=T[8]=$'a',模式串的末字符为c,a与c不匹配,则用字符a进行坏字符匹配,第一步next数组跳的距离为 $j-next[j]=3-next[3]=3$,第二步用bmBc数组跳的距离 $bmBc[a]=2$,两步的跳跃距离和就是 i 移动的距离,即 i 值等于 $i+bmBc(a)-next[j]=5$,此时 j 值重新赋值为0,再进行下一次窗口的匹配,如表4所示;若两字符匹配,则按照KMP的匹配过程进行比较,即 j 值等于 $next[j]$ 值,再进行下一轮的比较。

表4 字符不匹配的情况

0	1	2	3	4	5	6	7	8	9	10	11	12	13
a	c	b	c	c	a	d	b	a	c	b	a	c	c
a	c	b	a	c	c								
					a	c	b	a	c	c			

3.4 KMPP算法分析

KMPP算法的目的是尽量使文本中的指针 i 每次移动的距离达到最大,从而减少移动窗口次数,字符比较次数,从而降低运行时间。

BM算法的最大移动量是 m,而KMPP算法的最大移动量可以接近 $2m$ 通过大量实验证明当字符集 Σ 较大且字符 ($T[tLast]$) 出现在Pattern中的概率较低的情况下,可以显著提高KMPP算法的最大移动量。当该字符与模式串的末字符不匹配时,i 移动的距离为两步的跳跃距离和,第一步移动的距离由next数组决定,第二步跳跃的长度由bmBc数组决定。在第一步的移动距离接近 m 且第二步跳跃的距离为 m 的条件下,i 移动的长度将接近 $2m$。从以上分析可以得出KMPP算法在最好情况下的时间复杂度应该接近 $O(n/2m)$,是BM算法最好情况下的两倍,但是KMPP算法在搜索过程要多一步匹配工作,所以要考虑到 KMPP算法的平均比较次数。假设If语句的两个分支出现的概率相同,平均出现次数为 $1/2m$,两字符的匹配部分出现在else分支中,所以它的平均出现次数为 $1/4m$,在最好情况下KMPP算法的平均比较次数为 $(n/2m+n/4m)$,接近BM算法的1.33倍。综上所述,在字符集 Σ 的大小较大的情况,KMPP算法在运行性能上高于KMP算法和BM算法。

4 实验结果

本文针对BM算法、KMP算法、KMPP算法和NKMP算法进行对比实验分析。算法是在VC6.0编译器上实现,并运行在CPU为Intel® 2.30 GHz,内存为4 GB的计算机上,算法采用C++语言实现。

本文实验的文本是"100M128.txt"和"bible.txt"。"100M128.txt"是随机生成的100 MB大小的文本,字符集 Σ 大小为128;"bible.txt"文本来源于坎特伯雷语料(http://corpus.canterbury.ac.nz/),字符集 Σ 大小为63。实验随机构建长度为3、5、10、17、25、50的模式串,每组测试的次数为10次,执行上述算法程序,统计出模式串不同长度时各算法的运行时间和比较次数。

从图2~3,表5~6可以看出,KMPP算法在比较次数和运行时间上都明显低于KMP算法和NKMP算法,NKMP算法的运行效率高于KMP算法,但效果并不是很明显,此算法主要针对查找特殊的模式串,它的应用领域较小;KMPP算法的运行效率高于BM算法,说明在

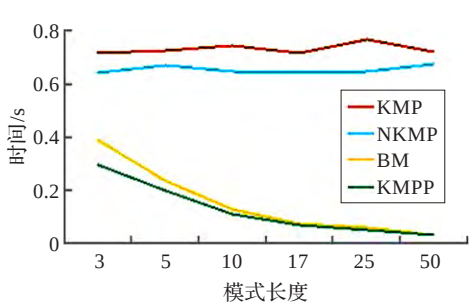

图2 四种算法运行时间对比("100M128"文本)

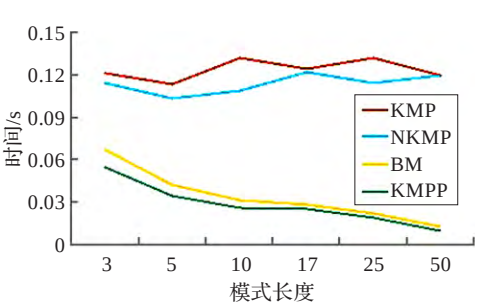

图3 四种算法运行时间对比("Bible"文本)

表5　四种算法的比较次数("100M128"文本)

模式长度	KMP算法	NKMP算法	BM算法	KMPP算法
3	100 781 150	91 071 595	33 862 576	25 888 133
5	100 781 440	97 259 867	20 478 723	17 397 069
10	100 781 223	99 907 374	10 439 582	9 653 551
17	100 781 184	99 904 345	6 317 392	6 084 360
25	100 781 893	99 999 979	4 411 801	4 294 815
50	100 781 204	99 999 999	2 418 864	2 411 032

表6　四种算法的比较次数("Bible"文本)

模式长度	KMP算法	NKMP算法	BM算法	KMPP算法
3	16 995 085	16 177 472	5 905 912	4 875 824
5	17 070 900	16 187 412	3 653 874	3 442 479
10	17 306 621	16 180 548	2 424 218	2 418 140
17	19 253 064	16 189 564	1 920 981	1 923 084
25	17 711 114	16 187 364	1 611 879	1 484 449
50	16 893 845	16 189 564	872 628	834 592

这个区域内KMPP算法每次移动的距离高于BM算法每次移动的距离。

5　结语

本文通过分析BM算法和KMP算法并结合两者的优点,提出一种改进算法——KMPP算法,该算法与KMP相同,最坏时间复杂度为 $O(m+|\Sigma|+n)$,明显优于BM的最差时间复杂度 $O(m\times n)$,并且通过实验证明在字符集较大的情况下,在窗口比较次数和运行时间上都低于KMP算法、BM算法和NKMP算法。在理论上,最好情况下KMPP算法运行效率接近BM算法的1.33倍。综上所述,KMPP算法提高了匹配效率,具有广阔的应用前景。

参考文献:

[1] Faro S,Lecroq T.The exact online string matching problem:a review of the most recent results[J].ACM Computing Surveys,2013.

[2] 王成,刘金刚.一种改进的字符串匹配算法[J].计算机工程,2006,32(2):62-64.

[3] Knuth D E,Morris J H,Pratt V R.Fast pattern matching in string[J].SIAM Journal on Computing,1977,20(6):323-350.

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

[5] Horspool R N.Practical fast searching in strings[J].Software:Practice and Experience,1980,10:501-506.

[6] Daniel M S.Very fast substring search algorithm[J].Communications of the ACM,1990,33(8):132-142.

[7] 万晓榆,杨波,攀自甫.改进的Sunday模式匹配算法[J].计算机工程,2009,35(7):125-129.

[8] Ahmed M,Kaykobad M,Chowdhury R A.A new string matching algorithm[J].International Journal of Computer Mathematics,2003,80(7):825-834.

[9] 杨俊丽,吕晓燕,满晰.基于改进的KMP算法的词频统计[J].微计算机信息,2010,26(9):161-162.

[10] 苏德福,钟诚.计算机算法设计与分析[M].2版.北京:清华大学出版社,2001:57-59.

[11] 鲁宏伟,魏凯,孔华峰.一种改进的KMP高效模式匹配算法[J].华中科技大学学报,2006,34(10):41-43.

[12] 严蔚敏,吴伟民.数据结构[M].2版.北京:清华大学出版社,1998.

[13] Rajesh S,Prathima S,Reddy L S S.Unusual pattern detection in DNA database using KMP algorithm[J].International Journal of Computer Applications,2010,1(22):1-5.

[14] 赵森严,黄伟,李阳铭.一种改进的KMP入侵检测的模式匹配算法[J].井冈山大学学报:自然科学版,2013,34(1):55-58.

[15] 解晨,王瑜.KMP算法研究与实现[J].电脑知识与技术,2013,9(20).

(上接32页)

[7] 谢箭,刘国良.基于神经网络的不确定性空间机器人自适应控制方法研究[J].宇航学报,2010,31(1):123-129.

[8] Guo Y S,Chen L.Adaptive neural network control of free-floating space manipulator system in joint space[C]//China Control Conference,2007:117-120.

[9] 张文辉,齐乃明.自由漂浮空间机器人神经网络自适应补偿控制[J].宇航学报,2011,32(6):1312-1317.

[10] 刘福才,高娟娟.不同重力环境下空间机械臂神经自适应鲁棒控制[J].宇航学报,2013,34(4):503-510.

[11] Dubowsky S,Papadopoulos E G.The kinematics,dynamics and control of free-flying space robotic systems[J].IEEE Trans on Robotics and Automation,1993,9(5):531-543.

[12] Xu Y S,Kanade T.Space robotics:dynamics and control[M].[S.l.]:Kluwer Academic Publishers,1992.

[13] 魏承,赵阳.空间机器人捕获漂浮目标的抓取控制[J].航空学报,2010,31(3):632-637.

[14] 韩力群.人工神经网络理论、设计及应用[M].北京:化学工业出版社,2007.

[15] Newton R T,Xu Y.Neural network control of a space manipulator[J].IEEE Control Systems Magazine,1993,13(6):14-22.