

基于位置索引的中英文模糊检索算法研究^①

王立彬^② 许成谦 袁丽

(燕山大学信息科学与工程学院 秦皇岛 066004)

摘要 信息检索效率的提高可以给用户带来更好的体验。信息检索的实质是字符串匹配,针对当前字符串匹配算法效率低下的问题,本文提出一种基于位置索引的中英文快速模糊检索算法,通过直接获取被匹配串的所有位置信息,在匹配串进行匹配时可根据位置信息达到直接精确匹配,从而提升匹配效率。并在此算法基础上深入分析了基于中文的模糊匹配查找和多关键字查找。通过实验测试,该算法在数据量、数据源长度和待匹配串长度等特性评价指标上均优于一般算法。

关键词 位置索引; 模糊检索; 字符串匹配; 快速检索; 中英文检索

0 引言

随着信息技术的日渐发展,信息化设备和智能化软件在人们生活中所占比重持续升高,越来越多的应用通过提高信息检索效率来改善用户体验。实际应用中,良好的用户体验在于使用者在快速、连续输入检索字符时,能够无感知地获得检索结果。如在手机中输入部分电话号码或联系人姓名可以快速地提示相关联的全部号码或联系人;在证券系统中输入部分证券代码或者简称也可以快速匹配出全部证券代码或者简称信息。由于信息检索的实质是字符串匹配,高效的字符串匹配算法在快速检索中发挥着重要作用,因此,高效匹配算法的研究十分必要。

当前字符串匹配方面有一些常见算法如 KMP 算法^[1]及其优化算法^[2-6],其通过对被匹配串进行匹配值计算,匹配失败后利用匹配串匹配值信息进行再次匹配,尽量减少匹配串与被匹配串的匹配次数以达到快速匹配。BM 算法^[7]及其优化算法^[8-12],其从右往左比较模式串的后缀,从而加大匹配的移动距离。这两类算法的时间复杂度都是基于被匹配

串长度的。文献[13]提出了一种基于位置索引的字符串查找算法(TNA 算法),其根据被匹配串中出现的字符的位置建立了一个表,并根据匹配串的长度拆分成子串,单个子串的匹配过程仍然是逐个匹配,多个子串需要匹配多次。KMP 算法的时间复杂度是 $O(m + n)$, BM 算法的时间复杂度由两部分组成,即预处理阶段和搜索阶段,前者的时间复杂度为 $O(m + n)$,后者最好情况下的时间复杂度为 $O(n/m)$,最坏情况下的时间复杂度为 $O(n \cdot m)$ 。TNA 算法平均时间复杂度仍然为 $O(m + n)$,最好可以达到 $O(n/m)$ 。当被匹配串数量很大时,这几种算法仍然无法满足性能要求。若能直接获取被匹配串的所有位置信息,在匹配串进行匹配时能根据位置信息直接精确匹配,将会大大提升匹配效率。基于此,本文提出一种基于位置索引的中英文快速模糊检索算法,时间复杂度为 $O(m/2)$,并且匹配过程中不存在字符串的移动。

1 算法描述

本文算法采用以空间换取时间的方法来提高检

^① 国家科技重大专项(2017ZX05019001-011)资助项目。

^② 男,1981 年生,博士;研究方向:信息编码理论,信息检索,高可用设计;E-mail: wanglb@neeq.com.cn
(收稿日期:2020-06-10)

索效率。本节以数字、字母组成的长度小于等于 32 字节的字符串为例,对算法进行描述。算法分两部分,首先是被匹配串初始化过程,即把众多的被匹配串经过编码并根据位置信息建立映射表的过程,其次是根据匹配串匹配的过程。

1.1 被匹配串初始化

1.1.1 符串编码

对字符串进行编码时只对字母和数字进行编码,空格和其他字符不进行区分。由于存在 26 个英文字母(不区分大小写)和 10 个数字编码,因此采用一个大小为 37 的 32 位整型数组来存取一个英文字符串,即每个字母的位置信息采用一个 32 位整型数的位信息来表示。首先需要把字符映射成 37 位数组的索引,然后用位置信息标识每一个 32 位整型,为了方便把经过标记位置信息的 37 位数组索引称为基准索引表。

字母和数组索引的映射表见表 1。

表 1 字母数组索引映射表

字符	位置索引	字符	位置索引	字符	位置索引
0	0	a 或 A	10	n 或 N	23
1	1	b 或 B	11	o 或 O	24
2	2	c 或 C	12	p 或 P	25
3	3	d 或 D	13	q 或 Q	26
4	4	e 或 E	14	r 或 R	27
5	5	f 或 F	15	s 或 S	28
6	6	g 或 G	16	t 或 T	29
7	7	h 或 H	17	u 或 U	30
8	8	i 或 I	18	v 或 V	31
9	9	j 或 J	19	w 或 W	32
空格或其他	36	k 或 K	20	x 或 X	33
		l 或 L	21	y 或 Y	34
		m 或 M	22	z 或 Z	35

1.1.2 基准索引表标记过程

每个字符在字符串中都有其对应的位置信息,同一字符在字符串中出现的位置可能存在多个,不同字符在同一字符串中出现的位置各不相同。为保证字符位置信息的准确性,就需记录单个字符在字符串中所有的位置信息。假设字符 0 在字符串中出现 3 次,位于字符串的第 2 位、第 3 位、第 4 位,这里从第 0 位开始计数,且需初始化的被匹配字符串长度不超过 32 位,在进行位置信息获取时采用二进制编码进行,则字符 0 所记录的位置信息为 00000000 00000000 00000000 00011100,转化为十六进制数表示为 0x1C。举例说明,被匹配字符串为 ABCABAAC, 字符 A 出现的位置为 0,3,5,6 位,字符 B 出现的位置为 1,4 位,字符 C 出现的位置为 2,7 位。内存中的编码见表 2。

1.1.3 批量被匹配字符串基准索引表映射过程

根据被匹配字符串的数量(*STR_CNT*),建立二维位置映射数组 *init_search[STR_CNT][37]*,分别对编码后的所有被匹配字符串组成的数据源进行基准索引表标记。在进行字符检索时只需根据检索字符的下标即可在数据源快速检索。

1.2 匹配算法

算法基于下面的定理和推论。

定理 一个从 *n* 位起始的 *t* 个连续字符组成,如果第 *n+1* 个字符左移 1 位,第 *n+2* 个字符左移 2 位,依次处理,直到第 *n+t-1* 个字符左移 *t-1* 位,那么经过移位后的字符均和第 *n* 位字符位置相同。

定理的证明是显而易见的,反过来可以得到下面的推论。

推论 一个字符串中任意选择 *t* 个字符,第 1 个字符的位置均左移 1 位,第 2 个字符的位置均左移 2 位,依次处理,直到第 *t-1* 个字符的位置均左移

表 2 基准索引表

字符	位置	31	...	8	7	6	5	4	3	2	1	0	HEX 值
A	10	0	0	0	0	1	1	0	1	0	0	1	0x69
B	11	0	0	0	0	0	0	1	0	0	1	0	0x12
C	12	0	0	0	1	0	0	0	0	1	0	0	0x84

$t - 1$ 位,如果经过左移的字符的位置均和第 0 个字符的位置存在交集,则这几个字符组成的子串在原字符串中必定相邻。

1.2.1 基于基准索引表的查找过程

在查找前,首先将待匹配字符串转换成基准索引表中的位置索引,由此对待匹配串的匹配过程即转换成了基于位置索引的与或运算。假定待匹配的字符为多个字符,先获取首字符的位置信息,若被匹配串中存在首字符(位置信息大于 0),则进行第一顺位字符的匹配。根据上面推论将第一顺位字符的位置向右(位置信息在 32 位整形中从低位向高位存放)移 1 位,并与首字符的位置信息进行与(&)操

作,当第一顺位字符存在时(位置信息大于 0),则继续与第二顺位字符右移 2 位后的位置信息进行相与,直到所有的待匹配字符串处理完成。如果最后得到的相与结果仍然大于 0,则说明待匹配字符匹配成功,否则匹配失败。

举例说明,假设被匹配字符串为 ABCABAAC,匹配字符串为 CAB,字符 C 的位置信息不变,字符 A 的位置信息右移 1 位,字符 B 的位置信息右移 2 位,将经过移位的位置信息求与,如下表所示,经过移位后字符位置信息 2 的值均为 1,因此求与后的值大于 0。具体实现见表 3。

表 3 匹配成功演示表

字符	位置	31	...	8	7	6	5	4	3	2	1	0	HEX 值
C	12	0	0	0	1	0	0	0	0	1	0	0	0x84
A	10	0	0	0	0	0	1	1	0	1	0	0	0x34
B	11	0	0	0	0	0	0	0	0	1	0	0	0x04

假如匹配字符串为 CAA,字符 C 的位置信息不变,第一顺位字符 A 的位置信息右移 1 位,第二顺位字符 A 的位置信息右移 2 位,经过移位的信息求与,如下表所示,经过移位后在任何位置没有出现均

为 1 的情况,因此求与后的值等于 0,说明被匹配字符串中不含有匹配字符串为 CAA。具体实现见表 4。

表 4 匹配失败演示表

字符	位置	31	...	8	7	6	5	4	3	2	1	0	HEX 值
C	12	0	0	0	1	0	0	0	0	1	0	0	0x84
A	10	0	0	0	0	0	1	1	0	1	0	0	0x34
A	10	0	0	0	0	0	0	1	1	0	1	0	0xA1

1.2.2 批量被匹配字符串匹配过程

对 $init_search[STR_CNT]$ [37] 中各被匹配串进行循环获取,并将其与待匹配串进行匹配,匹配结束后,返回数组中所有匹配成功的被匹配串信息,完成批量匹配。

1.3 算法整体流程

完整的算法匹配流程如下(见图 1)。

(1) 系统初始化时对被匹配串进行字符编码并记录各字符位置信息,建立基准索引表。

(2) 数据检索时按照相同编码方式对被匹配串

进行编码处理,即将被匹配字符串转换成基准索引表中的位置索引。

(3) 对字符位置信息进行移位与操作,完成字符匹配。

2 算法应用分析

2.1 汉字检索

实际检索中,常常存在被匹配串为中文字符^[14-16]的数据源,如果算法支持对汉字的检索则会

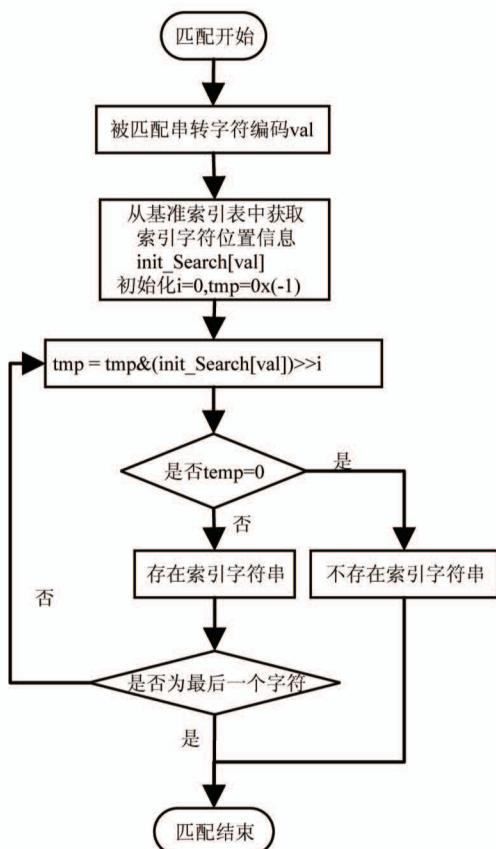


图 1 算法匹配流程

得到更加广泛的应用。手机中姓名的模糊查找、证券行业中对上市公司名称的模糊查找等,均是在有限的集合中进行查找,而本文算法也是在被匹配串初始化集合中对待匹配串进行检索,因此该算法可实现汉字检索。

汉字可以转化成汉语拼音,而汉语拼音是指用规定的字母和拼法拼成一个现代汉语的标准语音,通过对汉语拼音应用本文阐述的算法即可完成汉字检索。本节对汉字转换成拼音全拼或声母韵母的组合所建立的基准索引表进行了研究。

2.1.1 基于全拼的汉字检索

本文算法进行汉字检索时,检索结果提示的字符结果集会增加扩展音汉字的提示,如查找 ni 时,会同时提示 ni、nia、nian 等以 ni 开头的汉字,因此需要在对被匹配串建立映射时增加排序功能,即 ni、nia、nian 组成的汉字依次排列。

汉字通常由声母韵母组成,由表 5 可知,一个汉字平均可以转化成 4 个字母的组合,本文算法对单个汉字进行匹配时相当于完成了多次英文字符匹

配。多汉字时,初始化所占用的内存空间和匹配运算复杂度比单个字符所占内存空间和匹配运算复杂度增加 4 倍。由于存在几千个拼音各不相同的汉字,如果按照英文和数字编码的方式对中文进行编码则对内存的消耗巨大,使其在实际工程中没有应用价值。

表 5 声母韵母表

声母表	韵母表
b,d,g,j,zh,z,p,t,k,q,ch,c,	a ai ao an ang
m,n,h,x,sh,s,f,l,r,y,w	o ouong
	e ereieneng
	iiaianiangiaoieiouiong in ing
	u uauaiuoueiuauenuangueng
	v ve van vn

2.1.2 基于声母韵母的汉字检索

一个汉语拼音由多个字母组成,根据全拼进行中文汉字查找时需要多次匹配才能查找到需要的内容。为了改善匹配效率,需对汉语拼音编码进行优化。考虑到声母韵母的数量有限,若对其进行独立编码,这样一个汉字只需要声母和韵母的两次匹配即可以找到需要的汉字,匹配效率更高。

在实际的工程实践中,无须对所有的韵母建立索引映射,多个韵母可以共享索引值。如 c 或 ch 可以编为同一索引值,在对 c 进行匹配时可模糊匹配出 ch,提高用户体验。

扩展声母韵母后的基准索引见表 6。

2.2 多关键字检索

在实际应用中,多个关键字常常表示相同的内容,比如在证券行业,999999 表示上证指数,因此在检索里面通过 999 或者首字母 SZS 或者上证均应该能匹配到上证指数,通常称此类查找为多关键字查找。通过本文定义的算法很容易实现此类查找。

将多个关键字映射到同一个基准表中(见表 7),即可实现在一次匹配中完成所有关键字查找。以 999999、SZS、shangzhengzhishu 均表示上证指数为例,需要注意的是 shangzhengzhishu 会根据声母韵母拆分成 sh、ang、zh、eng、zh、i、sh、u。

表 6 数字声母韵母数组索引映射表

字符	位置索引	字符	位置索引	字符	位置索引
0	0	a 或 A	10	n 或 N	23
1	1	b 或 B	11	o 或 O	24
2	2	c 或 C 或 ch	12	p 或 P	25
3	3	d 或 D	13	q 或 Q	26
4	4	e 或 E	14	r 或 R	27
5	5	f 或 F	15	s 或 S 或 sh	28
6	6	g 或 G	16	t 或 T	29
7	7	h 或 H	17	u 或 U	30
8	8	i 或 I	18	v 或 V	31
9	9	j 或 J	19	w 或 W	32
空格或其他	36	k 或 K	20	x 或 X	33
		l 或 L	21	y 或 Y	34
		m 或 M	22	z 或 Z 或 zh	35
ai	37	ve	45	ua 或 uai 或	53
ao	38	va 或 van	46	uo	54
an 或 ang	39	vn	47	ue	55
ou	40	ia	48	uen 或 ueng	56
on 或 ong	41	ian 或 iang	49	uan 或 uang	57
er	42	iao	50	uei	58
ei	43	ie	51	ion 或 iong	59
en 或 eng	44	io 或 iou	52	in 或 ing	60

表 7 多关键字匹配原理表

字符	位置	31	...	8	7	6	5	4	3	2	1	0	HEX 值
9	10	0	0	0	0	0	1	1	1	1	1	1	0x3F
S/sh	11	0	0	0	0	1	0	0	1	0	0	1	0x49
Z/zh	12	0	0	0	0	0	0	1	0	1	1	0	0x16
ang	39	0	0	0	0	0	0	0	0	0	1	0	0x02
eng	44	0	0	0	0	0	0	0	1	0	0	0	0x08
i	18	0	0	0	0	0	1	0	0	0	0	0	0x20
u	30	0	0	0	1	0	0	0	0	0	0	0	0x80

在进行字符查找时程序依次取各个被匹配串进行匹配,匹配规则与单字符串匹配相同,当到达被匹配串数组下边界时匹配结束,返回匹配成功的被匹配串。以处理一行数据源为例,若第一个字符需要 n 次匹配,其余字符均只需进行 1 次移位操作匹配即可快速得到匹配结果,相较于 KMP 算法中在完成首字符匹配后,目标串其余每个字符都需根据匹配值进行多次匹配,计算量大为下降。

3 算法实验分析

由于检索算法对数字和字母的处理逻辑相同,

因此实验使用随机产生的数字和字母作为数据源,用来评估本文算法、TNA 算法、KMP 算法和 BM 算法的性能。在对比 3 种算法的性能时,把数据源分成了全部匹配、最前面 4 位均可以匹配其他位数不可以匹配、最后面 4 位可以匹配其他位数不可以匹配、随机生成数据源和全部不匹配的情况分别进行测试和对比。并在最后对准确率进行了简要说明。

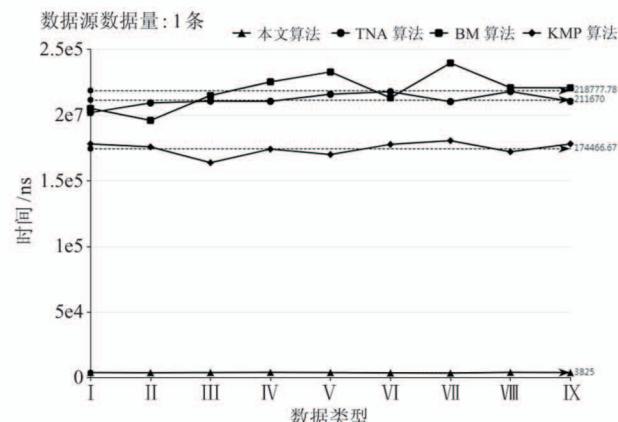
3.1 数据源数据量不同时耗时对比

在被匹配串数据源长度(20 字节)、待匹配串长度(4 字节)不变的情况下,实验中把数据源数据量分成 1 条、1 万条、10 万条和 100 万条的情况下进行

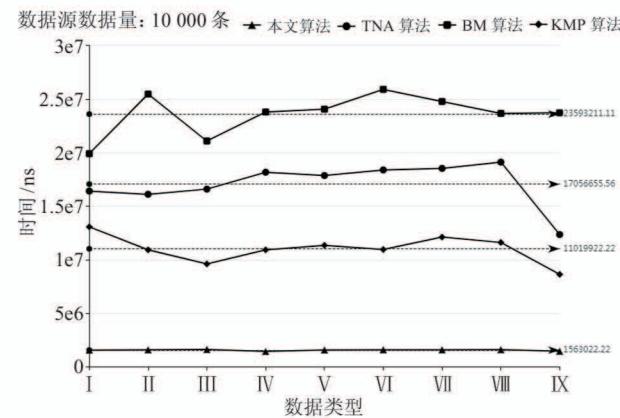
测试,实验结果见图 2。表 8 为平均执行时间,从实验数据可知本文算法匹配效率更高。

将实验数据类型分为:全部匹配(I)、最前面 4

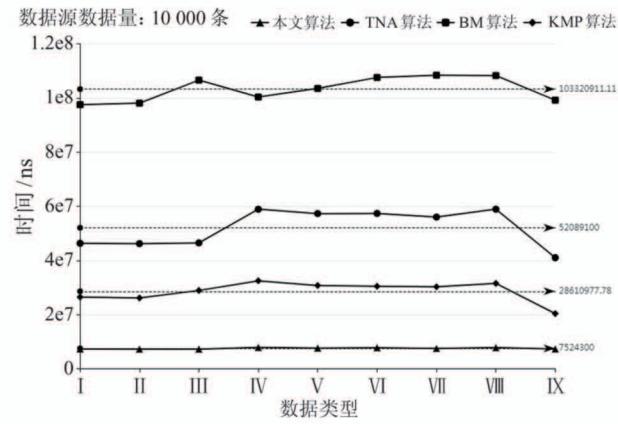
位均可以匹配其他位数不可以匹配(II)、最后面 4 位可以匹配其他位数不可以匹配(III)、随机生成数据源(IV、V、VI、VII、VIII)和全部不匹配(IX)。



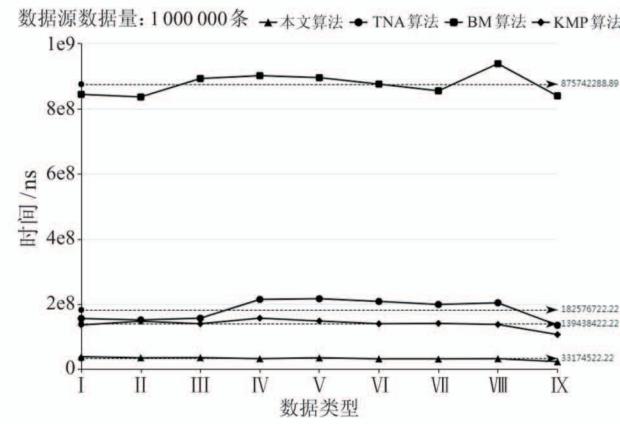
(a) 数据量 1 条



(b) 数据量 10 000 条



(c) 数据量 100 000 条



(d) 数据量 1 000 000 条

图 2 算法数据量时间对比图

表 8 算法数据量平均时间对比表

数据量条数	不同算法花费时间/ns			
	本文算法	TNA 算法	BM 算法	KMP 算法
1	3825	211670	218777	174466
10 000	1 563 022	17 056 655	23 593 211	11 019 922
100 000	7 524 300	52 089 100	103 320 911	28 610 977
1 000 000	33 174 522	182 576 722	875 742 288	139 438 422

3.2 数据源长度不同时耗时对比

在被匹配串数据源数据量(100 万条数)、待匹配串长度(4 字节)不变的情况下,实验中把被匹配串数据源长度分成 20 字节、30 字节、40 字节和 50 字节的情况下进行测试,实验结果见图 3。表 9 为平均执行时间,从实验数据可知本文算法匹配效率

更高。

将实验数据类型分为:全部匹配(I)、最前面 4 位均可以匹配其他位数不可以匹配(II)、最后面 4 位可以匹配其他位数不可以匹配(III)、随机生成数据源(IV、V、VI、VII、VIII)和全部不匹配(IX)。

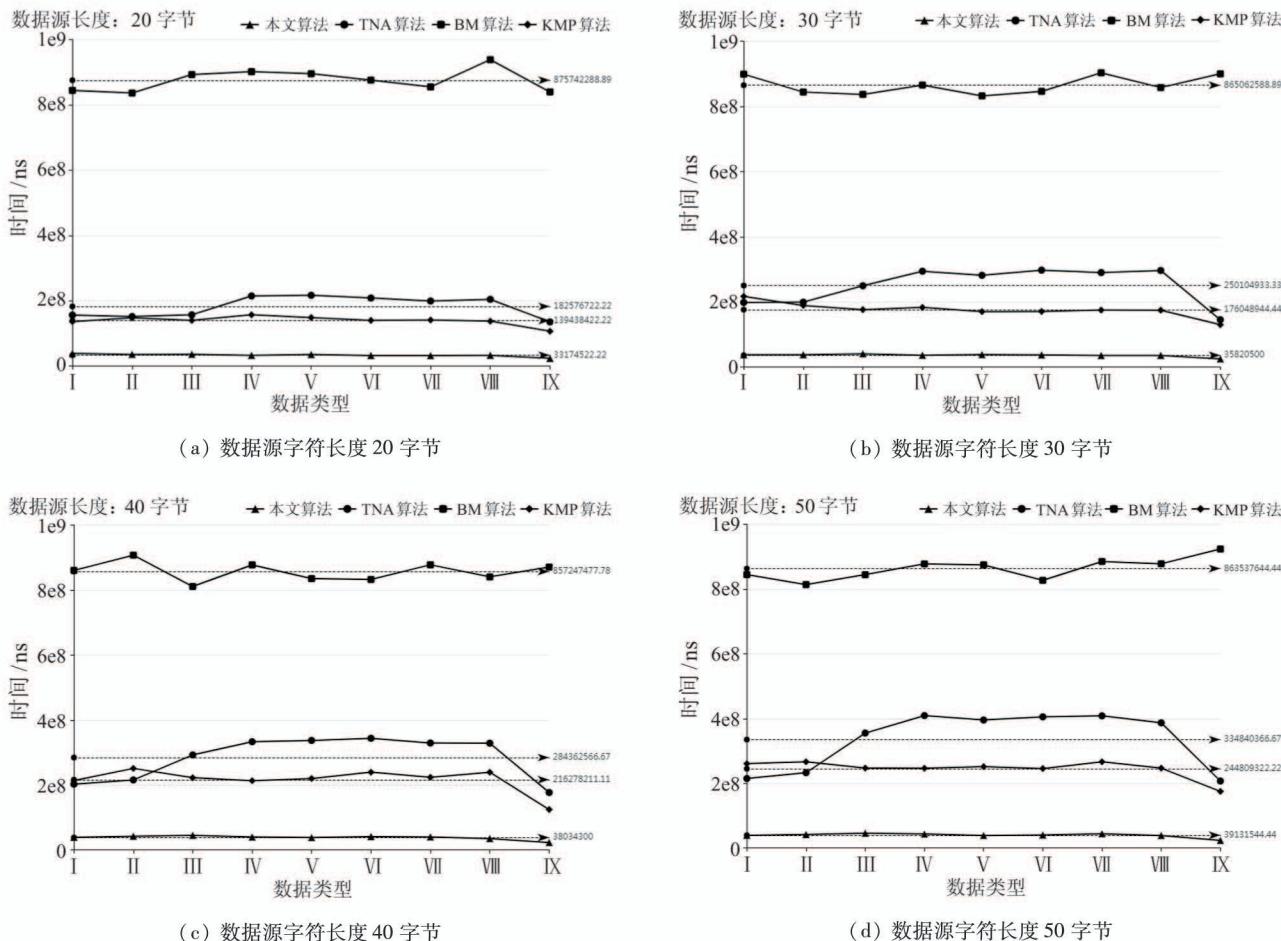


图 3 算法数据源字符长度时间对比图

表 9 算法数据源字符长度平均时间对比表

数据源长度 /字节	不同算法花费时间/ns			
	本文算法	TNA 算法	BM 算法	KMP 算法
20	33 174 522	182 576 722	875 742 288	139 438 422
30	35 820 500	250 104 933	865 062 588	176 048 944
40	38 034 300	284 362 566	857 247 477	216 278 211
50	39 131 544	334 840 366	863 537 644	244 809 322

3.3 待匹配串长度不同时耗时对比

在被匹配串数据源数据量(100万条数)、数据源长度(20字节)不变的情况下,实验中把待匹配串长度分成2字节、4字节、8字节和16字节的情况下进行测试,实验结果见图4。表10为平均执行时间,从实验数据可知本文算法匹配效率更高。

实验数据类型分为全部匹配(I)、随机生成数据源(II、III、IV、V、VI)和全部不匹配(VII)。

3.4 算法存储消耗情况分析

本算法初始化时,需要直接获取被匹配字符串

的所有位置信息,因此需要对存储空间消耗的变化进行分析。实验中把待匹配串长度分成8字节、16字节、32字节和64字节,并针对同一待匹配串长度把数据源分成50万条、100万条、150万条、200万条、250万条、300万条时进行测试,实验结果见图5。从实验数据可知本文算法内存使用率与被匹配串数据源数量、被匹配串数据源长度呈线性增长关系。

3.5 算法时间复杂度分析

情形1 在被匹配串数据源数据量(100万条数)

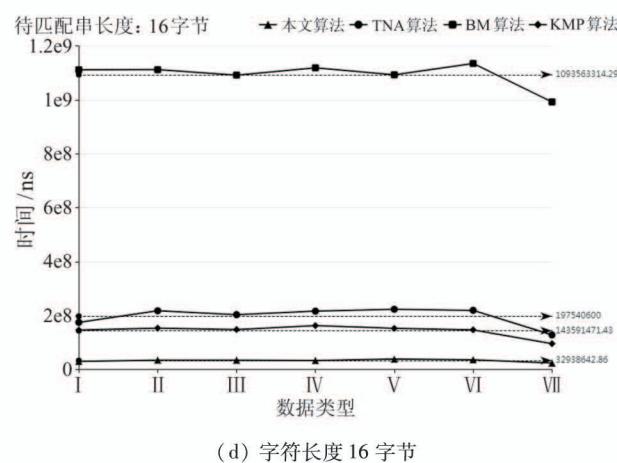
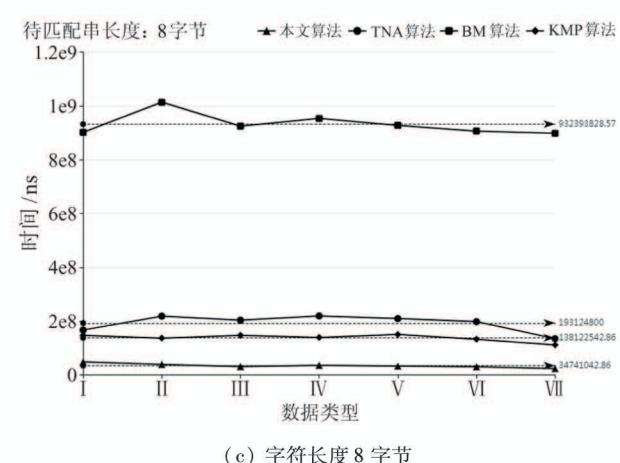
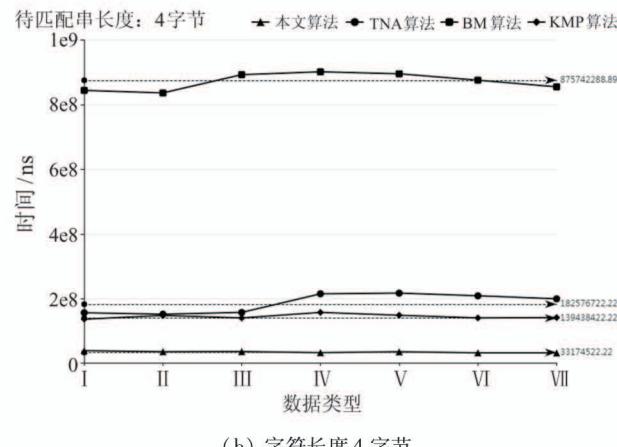
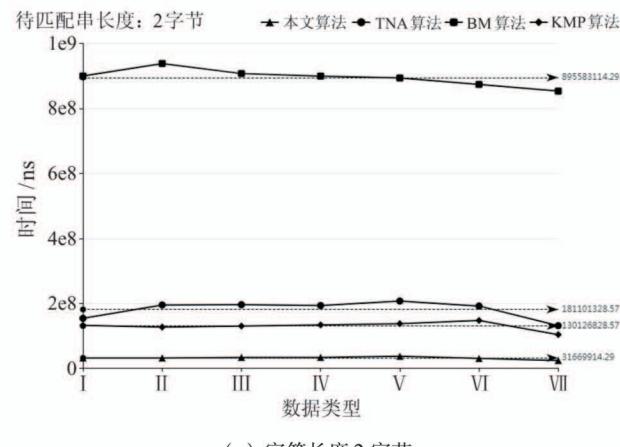


图 4 算法字符长度时间对比图

表 10 算法字符长度平均时间对比表

字符长度 /字节	不同算法花费时间/ns			
	本文算法	TNA 算法	BM 算法	KMP 算法
2	31 669 914	181 101 328	895 583 114	130 126 828
4	33 174 522	182 576 722	875 742 288	139 438 422
8	34 741 042	193 124 800	932 393 828	138 122 542
16	34 938 642	197 540 600	1 093 563 314	143 591 471

● 数据源长度: 8 字节 ▲ 数据源长度: 16 字节 ■ 数据源长度: 32 字节 ◆ 数据源长度: 64 字节

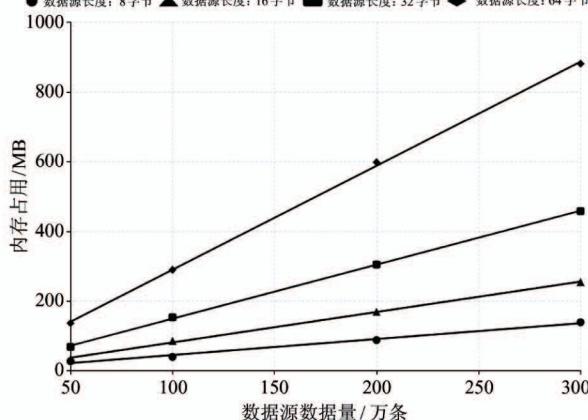


图 5 存储空间消耗图

不变情况下,实验中把待匹配串分成全部匹配和全部不匹配的情形,并把数据源被匹配串长度分成 8 字节、16 字节、32 字节和 64 字节,把待匹配串的长度分成 2 字节、4 字节、8 字节和 16 字节,分别进行测试验证,实验结果见图 6。

在图 6 中,2e7 ns 附近的 4 条线在全部不匹配的情形下基本重合,说明数据量相同的情况下,如果全部不匹配,耗时和待匹配串的长度无关。

图 6 中在 3e7 ns 上方的曲线为全部匹配的情形下待匹配串不同时的情况,说明待匹配串越长耗时

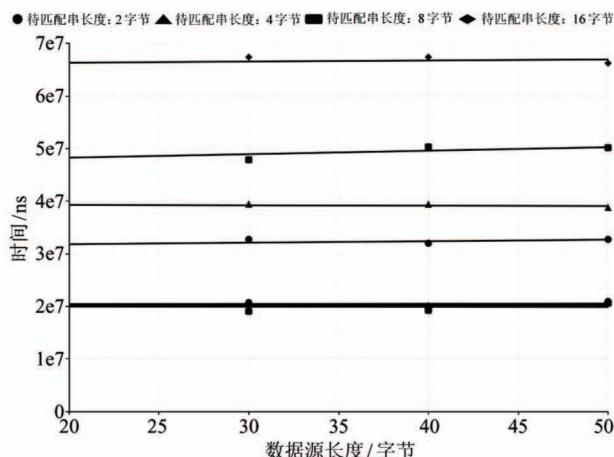


图 6 时间复杂度情形 1 验证图

越大。

情形 2 在被匹配串数据源长度(20 字节)不变的情况下,实验中变化待匹配串的长度和数据量,实验结果见图 7。从图中可知本文算法花费时间与待匹配串长度、被匹配串数据源数据量呈线性关系。待匹配串长度越长,算法花费时间越长;被匹配串数据源数量越大,算法花费时间越长。

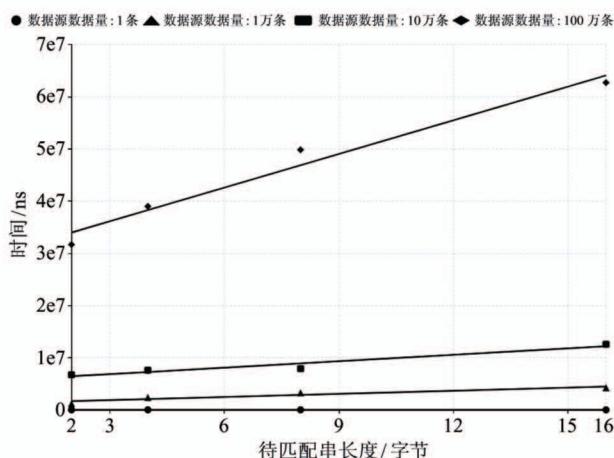


图 7 时间复杂度情形 2 验证图

情形 3 在待匹配串长度(4 字节)不变的情况下,变化被匹配串的长度和数据量,实验结果见图 8。从实验数据可知本文算法花费时间与被匹配串数据源长度无关,与被匹配串数据源数量有关,且被匹配串数据源数量越大,算法花费时间越长。

结合图 6、图 7 和图 8 的结论,说明本算法的时间复杂度为 $O(m/2)$, 即仅仅与待匹配串长度 m 成线性关系,与被匹配串长度 n 无关。

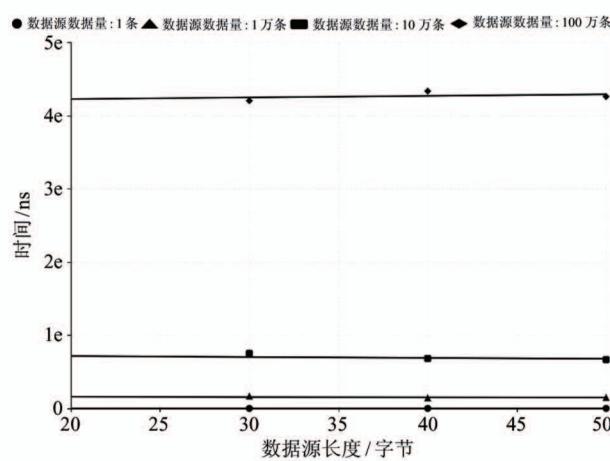


图 8 时间复杂度情形 3 验证图

3.6 准确率说明

本算法是基于基准索引表的精确匹配算法,这点在实验中也得到了很好的验证,只有匹配串在基准索引表中值在被匹配串中存在且顺序完全一致的情形才能检索出来。如表 1 所列的基准表中是不区分大小写的,在匹配过程中 abc 是完全可以匹配被匹配串中存在任意大小写 ABC 组合的字符串的。

4 结论

本文首先分析了目前主流的 KMP、BM 算法和近年基于索引的 TNA 算法,然后对本文提出的算法进行了详细的描述和测试验证,并对 4 种算法进行了性能测试,说明本文算法具有更好的检索效率,并且在实验中本文算法能对匹配串进行基于基准索引表的精确匹配。此算法在中文和关键字查找中所涉及的检索部分是完全相同的,唯一区别的是检索表的建立。从基准检索表的建立说明了此方法的快速查找是牺牲了非关键字符的精确匹配的,如在字符串查找算法中只关注英文和数字,而对于标点符号等一些特殊字符并不区分,但这在有限字符作为被匹配串的查找过程中仍然是非常有意义的,如手机号码的字符集合、身份证号码的字符集合和证券代码字符的集合等等。

本文算法在实际应用中可以根据不同的字符分布规律进行进一步优化,如每一次的查找基于上一次的查找结果、引入基于多线程的并行查找技术等。

参考文献

- [1] Knuth D, Morris J, Pratt V. Fast pattern matching in strings[J]. *SIAM Journal on Computing*, 1977, 6(2): 323-350
- [2] 姚秀情. 关于 KMP 算法改进的探讨[J]. 数字技术与应用, 2020, 38(4): 102-103
- [3] 孙娟红. 一种基于 KMP 算法思想的字符串匹配算法的研究与实现[J]. 电脑知识与技术, 2019, 15(26): 196-197
- [4] 李云辉. 基于 Hash 建立索引和 KMP 快速匹配算法的 DNA 序列查找方法[J]. 数学的实践与认识, 2016, 46(23): 173-179
- [5] 王淑礼. KMP 算法的一种新的简化算法[J]. 数学的实践与认识, 2016, 42(12): 230-234
- [6] Lu H W, Wei K, Kong H F. An improved KMP efficient pattern matching algorithm [J]. *Journal of Huazhong University of Science and Technology (Natural Science)*, 2006, 34(10): 41-43
- [7] Boyer R, Moore J. A fast string searching algorithm[J]. *Communications of the ACM*, 1977, 20(10): 762-772
- [8] Ying D, Hua L, Yu Q Q. Application of improved BM algorithm in string approximate [J]. *Procedia Computer Science*, 2020, 166: 576-581
- [9] 韦安垒, 李开科, 张榆. 一种快速单模式匹配算法的设计与实现[J]. 网络空间安全, 2018, 9(1): 86-92
- [10] 蔡婷, 杨卫帅. 一种改进的字符串模式匹配算法[J]. 物联网技术, 2017, 7(7): 89-91, 95
- [11] Jens A. Exact pattern matching: adapting the Boyer-Moore algorithm for DNA searches[J]. *PeerJ PrePrints*, 2016, doi: 10.7287/peerj.preprints.1758v1
- [12] 陈伟. 基于 BM 窗口竞争的高效单模式匹配算法[J]. 计算机工程, 2015, 41(12): 144-149
- [13] Tania I, Kamrul H. An improving algorithm for string matching using index based shifting approach[C] // The 20th International Conference of Computer and Information Technology, Dhaka, Bangladesh, 2017: 22-24
- [14] 邵清, 叶琨. 基于编辑距离和相似度改进的汉字字符串匹配[J]. 电子科技, 2016, 29(9): 7-11
- [15] Koloud A, Shadi A. A survey of string matching algorithms[J]. *International Journal of Engineering Research and Applications*, 2014, 4(7): 144-156
- [16] 常建秋, 沈炜. 基于字符串匹配的中文分词算法的研究[J]. 工业控制计算机, 2016, 29(2): 115-116

Research on fuzzy retrieval algorithm of Chinese and English based on location index

Wang Libin, Xu Chengqian, Yuan Li

(College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004)

Abstract

The improvement of information retrieval efficiency can bring better user experience. The essence of information retrieval is string matching. In view of the low efficiency of the current string matching algorithm, this paper proposes a fast fuzzy retrieval algorithm based on location index. By directly obtaining all the position information of the matched string, the direct and accurate matching can be achieved according to the position information when matching the matched string, thus greatly improving the matching efficiency. Based on this algorithm, the fuzzy matching search and multi keyword search based on Chinese are analyzed. The experimental results show that the algorithm is superior to the ordinary algorithm in terms of data quantity, data source length and string length to be matched and other characteristics.

Key words: location index, fuzzy retrieval, string matching, quick retrieval, Chinese and English search