周磊超, 彭展. 基于字符块跳转技术的藏文字符串匹配算法[J]. 智能计算机与应用,2024,14(12):90-95. DOI:10.20169/j.issn.2095-2163.24071801

基于字符块跳转技术的藏文字符串匹配算法

周磊超[1,2,3], 彭 展[1,2,3]

(1 西藏民族大学 信息工程学院, 陕西 咸阳 712082; 2 西藏网络空间治理研究基地, 陕西 咸阳 712082;
3 西藏自治区 光信息处理与可视化技术重点实验室, 陕西 咸阳 712082)

摘 要: 字符串匹配算法要求在文本串中查找模式串的出现位置。现有的字符串匹配算法大多针对 ASCII 字符集,由于藏文结构的特点,若将已有的字符串匹配算法直接应用于藏文上,则会导致其效率出现一定程度的下降。本文分析目前几种较快的字符串匹配算法和藏文结构特点后,提出了一种基于字符块跳转技术的藏文字符串匹配算法-BMH2T 算法。BMH2T 算法的核心思想是"先跳转,后匹配",该算法根据文本串中双字符块在模式串的出现位置进行跳转,提高了算法效率。实验结果表明,在处理藏文时,该算法相较于对比算法快约 20%~50%,具有更好的性能。

关键词: 藏文处理; 双字符块; 字符串匹配算法
中图分类号: TP391.1 **文献标志码**: A **文章编号**: 2095-2163(2024)12-0090-06

Tibetan string matching algorithm based on character block jump technology

ZHOU Leichao[1,2,3], PENG Zhan[1,2,3]

(1 School of Information Engineering, Xizang Minzu University, Xianyang 712082, Shaanxi, China;
2 Xizang Cyberspace Governace Research Base, Xianyang 712082, Shaanxi, China; 3 Key Laboratory of
Optical Information Processing and Visualization Technology of Tibet Autonomous Region, Xianyang 712082, Shaanxi, China)

Abstract: The string matching algorithm requires the position of the pattern string in the text string. The existing string matching algorithms are mostly for ASCII character sets. Due to the characteristics of the Tibetan structure, if the existing string matching algorithm is directly applied to Tibetan, there is a certain degree of reduction in its efficiency. This article analyzes several faster string algorithms and Tibetan structure characteristics, and proposes a Tibetan string algorithm - BMH2T algorithm based on character block jumping technology. The core idea of the BMH2T algorithm is "jump first, then match". The algorithm jumps according to the position of the mode string in line with the text string, therefore improves the algorithm efficiency. The experimental results show that when dealing with Tibetan, the algorithm is about 20%~50% faster than the comparison algorithm, which has better performance.

Key words: Tibetan language processing; double character block; string matching algorithm

0 引 言

藏语属汉藏语系藏缅语族藏语支,除了在中国的西藏、青海、四川、甘肃和云南等地区广泛使用外,在印度、尼泊尔和不丹等地也有广泛应用[1]。藏文是一种具有 1 400 多年历史的拼音文字,是藏语的书面表达形式[2]。随着藏文信息化技术不断普及[3],藏文文本数据在互联网上呈现出指数级的增长趋势[4]。如何从海量的藏文文本数据中检索关键信息,对于藏文敏感信息过滤、藏文文献检索和藏文搜索引擎等应用都至关重要[5]。字符串匹配算法作为文本检索的核心算法,其性能高低将直接影响上述应用的效率[6]。

当前,面向 ASCII 字符集的字符串匹配算法研究相对成熟,但面向藏文字符集的字符串匹配算法研究相对较少,因此研究藏文字符串匹配算法具有重大的意义。本文旨在研究一种高效的藏文字符串匹配算法。

基金项目: 西藏自治区自然科学基金(XZ202101ZR0089G)。

作者简介: 周磊超(1999—),男,硕士研究生,主要研究方向:文本处理。
通信作者: 彭 展(1986—),男,博士,讲师,主要研究方向:文本处理,网络安全,网络舆情。Email:284270220@qq.com。

收稿日期: 2024-07-18

1　字符串匹配算法

字符串匹配是计算机科学研究中最基础、最重要的问题之一，其研究相对成熟。经典的字符串匹配算法包括：暴力匹配算法[7]、KMP 算法[8]和 BM 算法[9]等，其中 BM 算法在实际场景中的表现最为出色，得到了学术界的重视关注，也引发诸多学者的研究兴趣，并对其陆续做出优化改进[10]。下面将介绍 BM 算法及其主要改进算法。令文本串为 $T[0\cdots n-1]$，长度为 n；模式串为 $P[0\cdots m-1]$，长度为 m；指针 k 指向模式串尾字符在匹配过程中所对应的文本串字符所在的位置。

1.1　BM 算法

BM 算法是最常用的字符串匹配算法之一。该算法从右向左匹配，在发生失配时，分别根据坏字符规则[9]和好后缀规则[9]计算出模式串的跳转距离，选取其中最大值，让模式串尽可能多地向右跳转。然而，将 BM 算法应用于藏文时，藏文中某些字符高频率的出现（如音节点‘ · ’在藏文文本中出现的频率很高，约占藏文文本全字符的 30%[11]），对 BM 算法的坏字符规则影响较大，这将直接影响其匹配效率。

图 1 是文本串 T 为“[illegible]”（意为：“名单里”），模式串 P 为“[illegible]”（意为：“里”）时，使用 BM 算法的匹配过程。BM 算法从右向左开始匹配，蓝色字符表示已经匹配成功的字符，红色字符表示匹配失败的字符。当匹配成功时，继续匹配，直到整个模式串完全匹配。当匹配失败时，根据算法的跳转规则进行跳转。

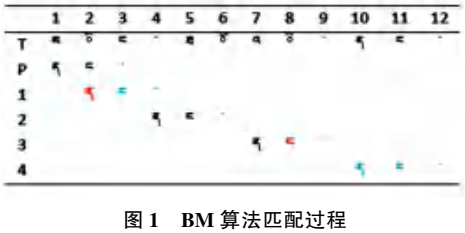

图 1　BM 算法匹配过程

Fig. 1　Matching process of BM algorithm

1.2　BMHS 算法

BMHS 算法是 Sunday[12]于 1990 年提出的对 BM 算法的改进算法。该算法较 BM 算法效率有所提升[13]，是速度最快的单模式匹配算法之一。BMHS 算法是从左向右匹配，在发生失配时，根据 $T[k+1]$ 决定模式串的跳转距离。当 $T[k+1]$ 在模式串中没有出现，则将模式串向右跳转 $m+1$ 个字符；当 $T[k+1]$ 在模式串中出现，则将 $T[k+1]$ 在模式串中最右端出现的位置和 $T[k+1]$ 对齐。类似于 BM 算法，将 BMHS 算法直接应用于藏文时，一些高频出现的字符，会导致 BMHS 算法平均跳转距离较短，对 BMHS 算法的效率有较大的影响。

图 2 是 BMHS 算法的匹配过程，BMHS 算法从左向右匹配，其中 T、P 和字符标记方式同前。

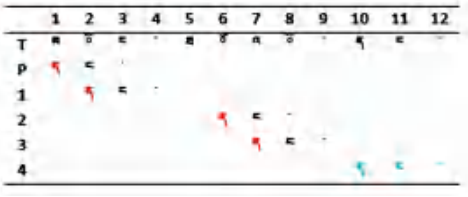

图 2　BMHS 算法匹配过程

Fig. 2　Matching process of BMHS algorithm

1.3　BMH2C 算法

BMH2C[14]算法是对 BMHS 算法的改进，该算法较 BMHS 算法效率提升较大。BMH2C 算法的思想和 BMHS 算法类似，只不过 BMH2C 算法使用双字符块决定模式串的跳转距离。BMH2C 算法从左向右匹配，在发生失配时，根据 $T[k]$ 和 $T[k+1]$ 组成的双字符块 $T[k]T[k+1]$ 决定模式串的跳转距离。当 $T[k]T[k+1]$ 在模式串中出现，将其在模式串中最右端的出现位置和 $T[k]T[k+1]$ 对齐；而当 $T[k]T[k+1]$ 在模式串中没有出现：若 $T[k+1]=P[0]$，模式串跳转距离为 m；否则，跳转距离为 $m+1$。BMH2C 算法使用双字符块决定模式串的跳转距离，由于双字符块相比单字符在模式串中出现的概率更低，因此 BMH2C 具有更长的跳转距离，故 BMH2C 算法的效率相比 BMHS 算法提升较大。

图 3 是 BMH2C 算法的匹配过程，BMH2C 算法从左向右匹配，其中 T、P 和字符标记方式同前。

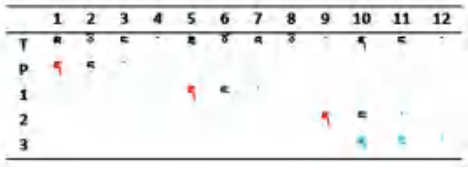

图 3　BMH2C 算法匹配过程

Fig. 3　Matching process of BMH2C algorithm

2　藏文结构及其编码分析

藏文是一种拼音型文字，共有 30 个辅音字母和 4 个元音字符[15]。藏文字形结构以一个辅音字母为核

心,其余字母以此为基础前后和上下拼写。藏文字形结构最少为 1 个辅音字母,最多由 6 个辅音字母构成,元音字符在辅音字母的上方或下方[16]。藏文的拼写顺序是前加字、上加字、基字、下加字、元音字符、后加字和再后加字。藏文字形结构如图 4 所示。藏文文字以音节为基本单位,各个音节之间用音节点‘ ’隔开,各个藏文句子之间用单垂符‘ ’分隔[17]。

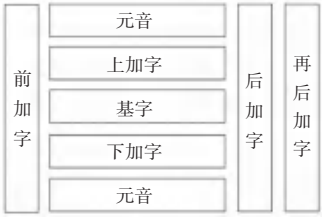

图 4 藏文字形结构

Fig. 4 Tibetan glyph structure

藏文的编码可以分为大字符集编码和小字符集编码[18]。其中,大字符集编码是将藏文的纵向结构按照一个单元进行处理,这种方法需要将上加字、基字、下加字和元音组合成为一个字丁,这种方式的编码方案需要预先知道所有的字丁组合,从而增加了系统处理的复杂性。小字符集编码是基于 ISO/IEC 10646 标准的编码方案,将藏文看作拼音文字,将一个藏文音节拆分成多个不同的部件,对每一个部件进行单独编码[19]。小字符集编码方案也是目前国际上主流的藏文编码方案,本文的算法以小字符集编码方案为准。

3 BMH2T 算法

3.1 基本思想

受到 BMHS 算法基本思想及 BMH2C 字符块技术的启发,本文提出了一种基于字符块跳转技术的藏文字符串匹配算法—BMH2T 算法。但该算法与 BMHS 算法和 BMH2C 算法有较大的不同,后两者在发生失配时,模式串才发生跳转,增加了字符的比较次数。而 BMH2T 算法的核心思想是“先跳转,后匹配”:将模式串跳转操作置于字符比较操作之前,且尽可能地向后跳转,以降低字符比较次数。为了提高模式串的平均跳转距离,在算法中使用了 $T[k-1]$ 和 $T[k]$ 组成的双字符块 $T[k-1][k]$ 决定模式串的跳转距离。这里,T 为文本串;指针 k 指向模式串尾字符在匹配过程中所对应的文本串字符所在的位置。

BMH2T 算法使用二维数组 *next* 存储模式串的跳转距离。*next* 数组的计算分为多种情况。研究可知,当双字符块 $T[k-1]T[k]$ 在模式串中出现时,将 $T[k-1]T[k]$ 在模式串中最右端出现的位置和文本串中 $T[k-1]T[k]$ 所在的位置对齐;当双字符块 $T[k-1]T[k]$ 在模式串中没有出现时,模式串的跳转距离分为多种情况:模式串的跳转距离为 $m-1$ 时,不会漏配,但跳转距离较短;若跳转距离为 m,可能会出现漏配。

本文结合藏文的特点做进一步的改进,使模式串在某些情况下的跳转距离可以增大到 m 或 $m+1$,且不会漏配。藏文文字以音节为基本单位,音节点‘ ’和‘ ’为藏文音节或句子之间的结束标志,本文利用此特点增大模式串的跳转距离。当 $T[k-1]$ 和 $T[k]$ 都不为音节点‘ ’或单垂符‘ ’时,模式串的跳转距离为 $m+1$;当 $T[k]$ 为音节点‘ ’或单垂符‘ ’时,模式串跳转距离为 m。

3.2 算法框架

BMH2T 算法使用 *next* 数组来确定匹配过程中模式串的跳转距离。数组大小为 256×256(256 为藏文字符集的大小),因跳转距离最大不超过 $m+1$(m 为模式串长,且通常不超过 100),因此数组元素使用 1 字节进行存储,使得 *next* 数组占用空间总大小仅为:256 × 256/1 024 = 64 KB。在算法开始前,根据下面 3 种情况对 *next* 数组进行初始化。

设 i 和 j 表示藏文字符集中的藏文字符。当 i 和 j 都不为‘ ’和‘ ’时,$next[i][j]=m+1$;当 j 为‘ ’或‘ ’时,i 为其他藏文字符时,$next[i][j]=m$;i 和 j 为其他情况时,$next[i][j]=m-1$。

设文本串 $T[0\cdots n-1]$,长度为 n;模式串 $P[0\cdots m-1]$,长度为 m。BMH2T 算法匹配过程的伪代码如下。

算法 BMH2T 算法

输入 长为 n 的文本串 T,长为 m 的模式串 P,初始化后的 *next* 数组

输出 s //若 P 是 T 的子串,则返回 P 在 T 中的起始位置 s;否则,返回 -1

1. for $i \leftarrow 0$ to $m-2$ do
2. $next[P[i]][P[i+1]] \leftarrow m-i-2$ //计算 *next* 数组
3. $s \leftarrow 0$ //s 为文本串 T 的指针
4. while $s <= n-m$
5. $j \leftarrow 0$//j 为模式串 P 的指针

```
6.  jump = next[T[s + m - 2]][T[s + m - 1]]
// 跳转距离
7.  while jump ! = 0  // 如果可能,则持续跳转
8.      s ← s + jump
9.      if s > n - m
10.         return -1 //匹配失败,返回-1.
11.  jump = next[T[s + m - 2]][T[s + m - 1]]
12.  while T[s + j] = P[j] // 逐个字符比较
13.     j ← j + 1
14.     if j = m// 匹配成功
15.         return s //返回 P 首字符在 T 中的位置
16.  s ← s + 1
17. return -1//匹配失败,返回-1
```

3.3　实例分析

图 5 是使用 BMH2T 算法的匹配过程实例，BMH2T 算法从左向右匹配，其中 T、P 和字符标记方式同前。图 6 是根据模式串 P 建立的 *next* 数组。在算法执行时，模式串根据 *next* 数组进行跳转。与上述算法不同的是，BMH2T 算法在进行匹配前，先根据 *next* 数组进行跳转，直到跳转距离为 0 时，才开始匹配，大幅减少了匹配次数。

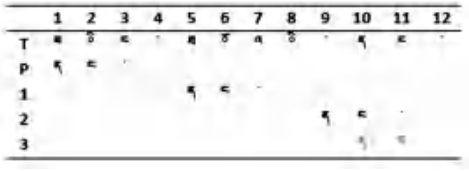

图 5　BMH2T 算法匹配过程

Fig. 5　Matching process of BMH2T algorithm

i	j	next[i][j]
		1
		0
除e外其他字符		3
任意字符		3
都不为或时		4
其他		2

图 6　*next* 数组

Fig. 6　*next* array

4　实验结果及分析

为了验证 BMH2T 算法的性能，本文选取 BM 算法、BMHS 算法和 BMH2C 算法来进行对比。实验从 3 个方面评估算法的性能：算法的运行时间、模式串的平均跳转距离、字符的比较次数。模式串的平均跳转距离的计算公式为：

$$\bar{D} = Length(Num)/Frequency \tag{1}$$

其中，$\bar{D}$ 表示模式串的平均跳转距离；*Length* 表示文本长度；*Num* 表示文本总的字符个数；*Frequency* 表示模式串的跳转次数。

字符的比较次数为在算法执行过程中总共进行了多少次字符比较。算法的运行时间直接反映了算法性能，字符的比较次数和模式串的平均跳转距离都对算法的执行效率有较大的影响[14]。

实验环境为 Windows11，配置为 Core(TM) i7-10700 CPU 2.90 GHz，运行内存为 16 GB。对比算法均使用 Python 语言实现，Python 版本为 3.9.13。实验所用文本数据来源为西藏语言文字网(http://tb.zyw.xizang.gov.cn/)、人民网藏文版网页(http://tibet.people.com.cn/)和中国西藏网(http://www.tibet.cn/cn/religion/)上的藏文文本。模式串从文本中随机抽取，长度 2~18 音节不等。

实验 1 中，文本串的大小固定为 16MB，测试各算法在不同长度模式串下的运行时间。实验 2 和实验 3 中，文本串的大小固定为 60 KB，分别测试在不同长度的模式串下，各算法字符的比较次数和模式串的平均跳转距离。为保证实验严谨性，所有实验结果，均为 50 次运行后的平均值。

图 7 和表 1 是实验 1 的运行结果。不难看出，模式串长度对各算法都有影响，模式串越长，算法执行速度越快。由于 BMH2T 和 BMH2C 算法是基于双字符块进行跳转，随着模式串长度的增加，其跳转距离更大、执行速度更快。在模式串到达一定长度时，串长的增加对各算法的影响有限，算法性能趋向平稳。具体来讲，BMH2T 算法在模式串长度较短时，执行速度比 BM 算法快约 36%，比 BMHS 算法快约 18%，比 BMH2C 算法快约 27%。在模式串长度较长时，执行速度比 BM 算法快约 59%，比 BMHS 算法快约 55%，比 BMH2C 算法快约 28%。

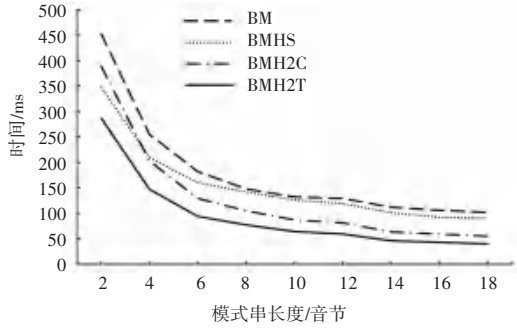

图 7　各算法的运行时间

Fig. 7　The running time of each algorithm

表 1 各算法的运行时间

Table 1 The running time of each algorithm

模式串长	时间/ms			
	BM	BMHS	BMH2C	BMH2T
2	453	348	389	287
4	255	210	203	147
6	182	160	129	94
8	148	142	105	77
10	132	126	87	64
12	129	119	81	59
14	112	101	63	46
16	106	92	59	43
18	102	90	55	40

图 8 和表 2 是实验 2 的运行结果。BMH2T 算法的字符比较次数相比其他算法有明显差别,在图 8 中可以看出 BMH2T 算法趋近于平稳,因为 BMH2T 算法相比其他算法有较大的不同。其他算法在字符匹配失败后进行跳转,BMH2T 算法一直进行跳转,除非跳转距离为 0 才开始进行匹配,故与其他算法相比,则大幅减少了字符比较次数。

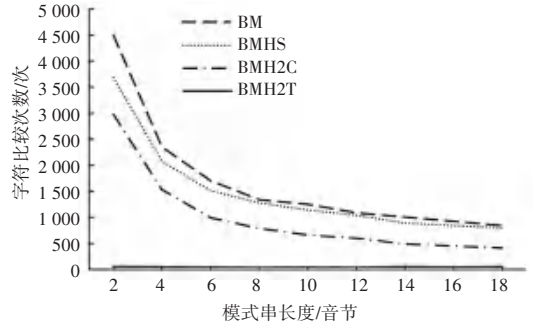

图 8 各算法在执行过程中字符的比较次数

Fig. 8 The number of characters comparison during the execution of each algorithm

表 2 各算法在执行过程中字符的比较次数

Table 2 The number of characters comparison during the execution of each algorithm

模式串长	字符比较次数/次			
	BM	BMHS	BMH2C	BMH2T
2	4 503	3 690	2 982	61
4	2 351	2 077	1 535	51
6	1 703	1 510	993	46
8	1 340	1 276	791	41
10	1 250	1 142	661	44
12	1 089	1 039	601	40
14	1 003	889	489	56
16	925	844	450	45
18	841	800	412	56

图 9 和表 3 是实验 3 的运行结果。表 3 中,藏文模式串的基本单位为音节,1 个音节由 1~7 个字符组成。模式串的平均跳转距离为 1 次跳转平均跳过了多少个字符。可以看出,虽然 BMHS 算法模式串的平均跳转距离要逊色于 BM 算法,但 BM 算法在计算最大跳转距离时开销较大,且字符比较次数较多。较大时间开销及其较多的字符比较次数,导致其比 BMHS 算法慢。当模式串较长时,BM 算法和 BMHS 算法中模式串的平均跳转距离均在增长,但 BM 算法增长的幅度更大,可以弥补上述 BM 算法的不足。因此,在模式串较长时,BM 算法运行时间趋近于 BMHS 算法。

BMH2T 算法相比 BM 算法和 BMHS 算法,有更大的模式串平均跳转距离和更少的字符比较次数,同时 BMH2T 算法的匹配过程相当精炼,没有复杂的分支判断,这些都使得 BMH2T 算法有更高的执行效率。另一方面,尽管 BMH2C 算法的模式串平均跳转距离优于 BMH2T 算法,但是其字符比较次数却远高于 BMH2T,导致其综合性能不如 BMH2T。

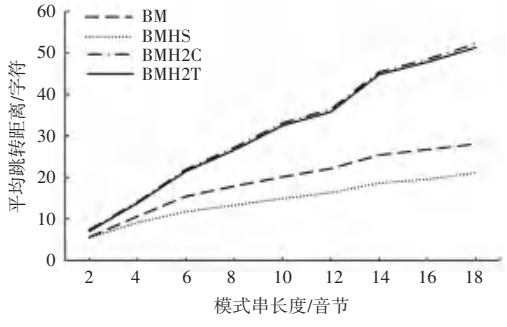

图 9 不同长度模式串的平均跳转距离

Fig. 9 The average jump distance of pattern strings of different lengths

表 3 不同长度模式串的平均跳转距离

Table 3 The average jump distance of pattern strings of different lengths

模式串长	平均跳转距离			
	BM	BMHS	BMH2C	BMH2T
2	5.6	5.4	7.3	7.0
4	10.6	9.1	14.1	13.8
6	15.4	11.7	21.9	21.5
8	17.9	13.3	27.2	26.6
10	20.1	14.9	33.1	32.5
12	22.1	16.3	36.4	35.8
14	25.4	18.6	45.4	44.8
16	26.7	19.5	48.5	47.8
18	28.0	21.1	52.2	51.3

综上，BMH2T 算法在所有场景下表现均为最佳，相较于对比算法，更加适用于藏文字符串匹配任务。

5　结束语

本文介绍了 BM 算法、BMHS 算法和 BMH2C 算法，分析这 3 种算法的特点。同时，针对藏文的特征，提出了一种基于字符块跳转技术的藏文字符串匹配算法—BMH2T 算法。实验结果表明，在处理藏文时，该算法具有更好的性能表现。未来可对其*next*数组及模式串的跳转距离做进一步优化。

参考文献

[1] 格桑加措，阿卜杜热西提·热合曼，尼玛扎西，等. Bi-LSTM 和 CRF 结合的藏文分词方法研究[J]. 中央民族大学学报(自然科学版)，2024，33(3)：40-46.

[2] 王蒙，彭展，杨涵刈. 基于藏文元音构件的字符串匹配算法[J]. 电子技术与软件工程，2022(18)：137-142.

[3] 李果，陈晨，杨进，等. 基于 DAN 与 FastText 的藏文短文本分类研究[J]. 计算机科学，2024，51(S1)：115-119.

[4] 胥桂仙，刘兰寅，张廷，等. 基于预训练模型和图神经网络的藏文文本分类研究[J]. 东北师大学报(自然科学版)，2023，55(1)：52-64.

[5] HAKAK S，KAMSIN A，SHIVAKUMARA P，et al. A new split based searching for exact pattern matching for natural texts[J]. PloS One，2018，13(7)：e0200912.

[6] 焦文欢，冯兴杰. 一种改进的字符串匹配模型研究[J]. 计算机仿真，2022，39(3)：319-324.

[7] 巫喜红，文张斌. BF 模式匹配算法的改进[J]. 计算机测量与控制，2018，26(5)：173-176.

[8] KNUTH D E，MORRIS，JR J H，et al. Fast pattern matching in strings[J]. SIAM Journal on Computing，1977，6(2)：323-350.

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

[10] 张欢，胡勇. 一种改进的 BMHS 模式匹配算法[J]. 计算机时代，2015 (1)：8-12.

[11] 春燕，曲珍. 藏文文本编码识别方法研究[J]. 计算机工程与应用，2013，49(1)：141-144.

[12] SUNDAY D M. A very fast substring search algorithm[J]. Communications of the ACM，1990，33(8)：132-142.

[13] 李明月，张善卿，陆剑锋，等. 一种改进的 Sunday 匹配算法[J]. 杭州电子科技大学学报(自然科学版)，2015，35(1)：93-96.

[14] 钱屹，侯义斌. 一种快速的字符串匹配算法[J]. 小型微型计算机系统，2004，25(3)：410-413.

[15] 尼玛扎西，完么扎西. 藏语自然语言处理基本理论和方法[M]. 北京：科学出版社，2020.

[16] 珠杰. 藏文信息处理中若干关键技术研究[D]. 成都：西南交通大学，2016.

[17] 陈小莹. 现代藏文音节结构分析研究[J]. 智能计算机与应用，2019，9(2)：89-90.

[18] 陈小莹，艾金勇. 基于小字符集藏文拉丁转写系统的设计与实现[J]. 中文信息学报，2016，30(3)：74-78.

[19] 春燕. 基于藏文音节特征的模式匹配算法的研究[J]. 计算机光盘软件与应用，2014，17(15)：119-120.

[20] 王艳霞，江艳霞，王亚刚，等. BMH2C 单模匹配算法的研究与改进[J]. 计算机工程，2014，40(3)：298-302.