一种在汉语文本中抽取重复字串的快速算法

马颖华，王永成，苏贵洋
（上海交通大学计算机系，上海 200030）

摘 要： 词典未登录词的处理是自然语言处理不可或缺的研究方向.抽取文本中重复出现的字串是抽取未登录词最为直接简便的方法.以往算法运行速度较慢，无法满足海量文本快速处理的要求.遵循“左结合优先”和“最长匹配”原则，本文提出一种快速算法：位置记忆跳跃匹配.该方法最差情况下时间复杂度为 $o(t^2)$，其中 t 为重复字串的重复次数.比较实验表明，本方法速度提高明显，数据结构简单，处理过程一次扫描完成.

关键词： 重复字串抽取；自动抽词；汉语文本处理

中图分类号： TP391.1 **文献标识码：** A **文章编号：** 0372-2112 (2002) 12A-2177-04

A Fast Approach of Extracting Repeated String from Chinese Text

MA Ying-hua, WANG Yong-cheng, SU Gui-yang
(*Dept. Computer Science, Shanghai Jiao Tong University; Schanghai* 200030, *China*)

Abstract: The processing of words unlisted in dictionaries is necessary in natural language processing. Extraction of repeated string appearing in text is the most direct, convenient method, and it is rather effective. Existing algorithms can not meet the requirement of high speed in vast text processing system. Aceording to principles of "left first" and "longest first", a fast approch named "Postitional Remembering and Jump Matching" which works in worst condition $o(t^2)$ time, where is repeating times of substring, is put forwards. Results of experments show that compared with previous methods, this method gains advantages such as high speed, simple data structures, and simultaneous scanning and matching processing.

Key words: repeated string extraction; automatic word extraction; Chinese text processing

1 引言

分词、抽词是文本处理的基础工作，可分为基于词典的方法及无词典方法两类.即使是最大的词典不会也不可能囊括自然语言的所有词汇[1]，所以词典外未登录词的处理是自然语言处理中不可缺少的部分.

目前在未登录词抽取研究中，有的基于统计和规则方法自动识别地名[2]、姓名[3]、机构名[4]；文献[5]采用 x2——统计量和广义似然比方法计算字之间的相关度，以抽取未登录词用于自动分词和自动编制词典；文献[6]引入支持度、置信度等概念来筛选词条，在无需词典支持和利用语料库学习的前提下抽取中、高频词条.在其他应用领域也有类似的寻找重复模式的方法，例如文献[9]利用关联矩阵和 R-P 树发现音乐乐段的重复模式.但这个算法适合小字符集、重复率高的情况，不适合汉语文本大字符集、重复率低的特点.

基于重复字串统计角度进行的研究[7,8]，也称为“串频统计”.串频统计的方法是这些方法中最简便直接的方法.文献[7,8]中介绍的算法基本相同，后者进行了算法上的改进，使得运算速度有所加快.但是串频统计运行速度仍然远远慢于基于词典的方法[7]，妨碍了无词典方法的广泛应用.

采用串频统计进行未登录词抽取的过程一般可分为三步.一，使用切割标记对文本进行预切割；二，在切割好的文本中找出重复字串；第三，对重复字串进行统计加权，判断重复字串是否为“词”.步骤二是最为耗时的.

文献[8]的做法是将文本倒排映射到汉字 Hash 表中：通过遍历倒排结构，找出位置相邻的字所构造的所有字串；排序并计算字串频次，输出高频字串.该算法对文本进行了两次扫描，第一次形成了倒排 Hash，第二次进行了高频统计.

在以往研究的基础上，本文提出了一种快速串频统计算法，称为位置记忆跳跃匹配算法.该算法对文本进行一次扫描，扫描过程中根据以往的匹配成功数据进行跳跃匹配.

2 位置记忆跳跃匹配算法

2.1 重复字串

在设计从文本中找出所有的重复字串的算法前，首先定义几个概念.

定义 1 设 $C_1C_2\cdots C_{n-1}C_n$ 为长度为 n 的文本，字串 $C_i\cdots C_{i+l}$ 与字串 $C_j\cdots C_{j+l}$ 完全相同，即 $C_{i+k}=C_{j+k}$，其中 $j>i+1$，

收稿日期：2002-05-07；修回日期：2002-09-09
基金项目：国家自然科学基金（No.60082003）

$l \geq 1, k > 0$ 且 $k \leq l$. 那么字串 $C_i \cdots C_{i+l}$ 就称为文本中的重复字串.

定义 2 存在字串 $C_m \cdots C_{m+l}$, 那么任何满足以下条件的字串都为该字串的子串; $C_{m+i} \cdots C_{m+j}$ (其中 $i \geq 0$, 且 $j \leq l$). 字串是其自身的字串.

定理 1 字串 S 是文本 T 的重复字串, 则该字串的所有长度大于 1 的子串也为文本 T 的重复字串. 同理, 字串 S 重复出现的频率必定小于或等于其子串重复出现的频率.

定义 3 设有字串 S, 其长度大于 1 的子串 C 在文本 T 中重复出现的频率等于字串 S 重复出现的频率, 则子串 C 称为字串 S 的完全覆盖子串.

定义 4 如果重复字串 S 是另一个重复字串的完全覆盖子串, 则 S 是平凡的, 否则 S 是非平凡的.

串频统计即要在文本中抽取非平凡的重复字串.

2.2 位置记忆跳跃匹配算法

位置记忆跳跃匹配算法中应用两个数据结构:

(1)字符在文本中第一次以及最后一次出现的位置数组, 该数组采用 Hash 结构, 称为位置 Hash 数组.

(2)与文本长度大小一致的匹配跳跃信息数组, 保存字符下一个出现位置、当前位置匹配成功最大长度以及与之匹配成功的字串位置.

2.2.1 位置 Hash 数组 汉语字符数量多, 一级和二级字符总数达到 6763 个, 这些汉字覆盖了 99.99% 的常用汉字[1]. 位置记忆跳跃匹配算法应用 Hash 结构保存字符在文本中出现的第一位置和最后位置. 具体 Hash 算法与文献[8]中一致. Hash 函数如下:

```
int pos(char 汉字字符 ){
    return(字符内码高字节 - 0xa0) × 94
          (字符内码低字节 - 0xa0);  }
```

把整个汉字表无冲突地 Hash 到了一个大小为 8200 的数组中, 其中一、二级以外字符的入口点都为 8180. 表中数据单元中保存两个数据: 第一位置号, 最后出现位置号.

```
struct HashItem{
    long lFirstPosition = 0;//字符第一次出现位置号
    long lLastPosition = 0;//字符最后出现的位置号  };
HashItem hashArray[8200];
```

如图 1 右左侧 Hash 结构示意图所示, 字符 A 出现的第一位置号为 1, 此时扫描位置为 9, 在已扫描的文本中字符 A 最后出现的位置是 8.

2.2.2 匹配跳跃信息数组 位置记忆跳跃匹配算法中使用的另一个数据结构是大小等于文本长度的数组, 称为匹配跳跃信息数组. 该数组单元保存三个数据: 字符下一个出现位置、当前位置匹配成功的字串最大长度以及与之匹配成功的字串位置.

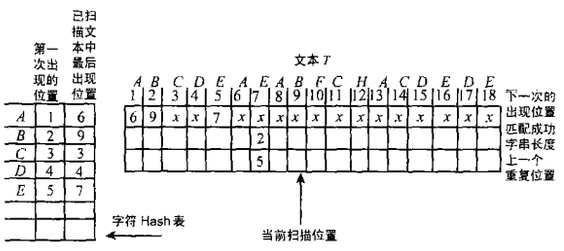

图 1 位置跳跃匹配算法示意图(扫描到文本中)

```
struct TxtLengthItem{
    long lNextPosition = 0;      //下一次出现的位置号
    int iLenRepeatedStr = 0;     //匹配成功的重复字串长度
    log lPrevMatchSuccess = 0; //与之重复的字串位置};
TxtLengthItem * pTxtLenArray;
pTxtLenArray = new TxtLengthItem(文本长度);
```

2.2.3 扫描和匹配过程 以上两种数据结构中的数据都是在文本一次扫描过程中不断填入的. 例如, 扫描到位置 i, 如果该字符是第一次出现, 则更新位置 Hash 数组的 lFirstPosition 的值. 如果不是第一次出现, 则首先则更新该字符 lLastPosition 指向位置上匹配跳跃信息数组中的 lNextPosition 数据, 然后填入 Hash 数组中的 lNextPosition 数据.

```
if(hashArray[pos(Text[i])].lFirstPosition = =0){
    hashArray[pos(Text[i])].lFirstPosition = i;}
else{pTxtLenArray[hashArray[pos(Text[i])]].
        lLastPosition].lNextPosition = i;      }
hashArray[pos(Text[i])].lLastPosition = 1;
```

本算法中有两个跳跃过程. 一是扫描过程中的跳跃; 二是匹配过程中的跳跃. 下面先来讨论匹配的跳跃过程.

如果扫描位置上字符曾多次出现(lFirstPosition 不为零), 算法就进入了匹配过程.

匹配是扫描当前位置开始字串和以往相同首字的字串(待匹配位置)进行匹配. 匹配过程中利用以往的匹配成功信息进行适当的跳跃. 例如文本"CDECDEFCDEF", 位置 4 上的 C 字符可以与位置 1 上的 C 字符成功匹配串长为 3 的重复字串. 位置 4 上的匹配跳跃信息数组的 iLenRepeatedStr 和 lPrevMatchSuccess 数据表示为(3,1), 1 表示该重复字串是在位置 1 匹配成功的. 扫描到位置 8, 字符为"C", 该字符既往出现 2 次. 首先与位置 1(待匹配位置)进行匹配; 位置 1 的跳跃信息数组数据为(0,0), 表明此处之前未有匹配成功字串. 在该处匹配成功串长为 3 的重复字串, 位置 8 数据更新为(3,1). 其次与位置 4 进行匹配, 位置 4 的(待匹配位置)的跳跃信息数组数据为(3,1)与位置 8(扫描当前位置)相同, 无须匹配前三个字符, 直接匹配第 4 个字符(位移为 3), 匹配成功, 位置 4 的跳跃信息数组数据改为(4,1).

匹配中的跳跃过程是把以往匹配成功的数据作为参考进行的. 有下面几种可能性:

(1)扫描位置的跳跃信息数组数据为(0,0), 则匹配过程从位移量 1(当前位置的下一个位置)开始进行;

(2)待匹配位置和扫描位置的跳跃信息数据相同, 匹配从位移量为匹配成功长度处开始继续匹配;

(3)待匹配位置和扫描位置的跳跃信息数据不同, 则此处无须匹配, 直接跳跃到下一位置继续匹配.

如果某一个匹配位置上的匹配过程结束后, 待匹配位置和扫描位置的跳跃信息数据仍然相同, 则扫描位置上跳跃信

息数据中 iLenRepeatedStr 置 0,lPrevMatchSuccess 置为现匹配位置号.这一步骤是为了防止在不同的位置记录了相同的重复字串匹配数据,这样,所有的匹配成功数据都被保存在第一次重复的位置.

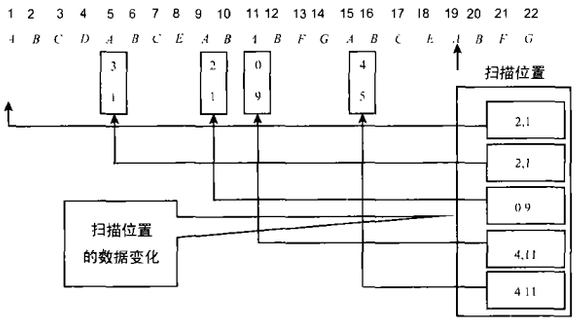

图 2　匹配过程跳跃示意图

图 2 中说明了文本串"ABCDABCEABABFGABCEABFG",扫描到位置 19 后的匹配过程.位置 1 匹配结束后,iLenRepeatedStr 和 lPrevMatchSuccess 数据为(2,1).位置 5 匹配过程由于匹配数据不同,所以跳过.位置 9 匹配过程,从位移为 2 开始.匹配结束后,匹配长度是 2,与匹配位置数据重复,改为(0,9).位置 11 匹配过程与位置 9 类似,但匹配结束后,成功长度增至 4,匹配数据改为(4,11).位置 15 的匹配过程由于匹配数据不符,跳过.整个匹配过程结束,结果为(4,11).

扫描过程的跳跃过程是在扫描过程中进行匹配成功长度的跳跃.如图 1 所示,在完成位置 7 的全部匹配后,跳过位置 8,直接开始进行位置 9 的扫描过程.

扫描过程结束后最终数据如示意图 3 所示.由图可知文本 T 的重复字串有:"CDE"重复次数 1、"DE"重复次数 1 和"EA"的重复次数.

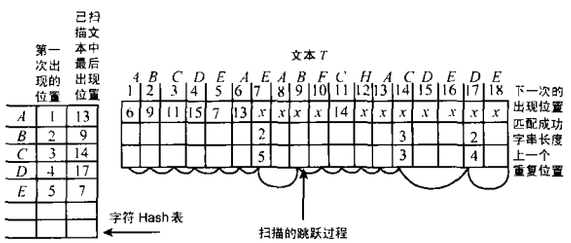

图 3　位置跳跃匹配算法示意图
(扫描的跳跃过程及最终结果)

3　算法性能分析

3.1　性能分析

与以往算法相比,本文算法中使用的数据结构都是固定大小和长度的数组,没有采用任何链表结构.算法不需要额外的排序过程,且是一次扫描,以往算法都为多次扫描过程.

本算法共使用了两个数据结构,其中 Hash 数组的空间占用为:8200 × 2 × size(long) = 64k.跳跃信息数组占用空间为 $10 \cdot N$,其中 N 为文本长度.算法的空间总占用为:64K + $10 \cdot N$.另外这两个数据结构相当简单,易于编程实现.

位置跳跃匹配算法中总的匹配次数小于 $l + \sum(1/2) \cdot t \cdot (t+1)$,其中 1 为文本总长度,$t$ 为每一个重复字串的重复次数,t 小于重复字串总个数.即该算法的时间复杂度最差情况下为 $o(t^2)$,其中 t 为重复字串的重复次数.与文献[8]的 $o(n \cdot \log(n))$相比,在文本字符重复次数少的情况下,具有更良好的运行性能.汉语文本字符重复次数相对较少,所以该算法非常适用于在汉语文本中寻找重复字串.

3.2　实验结果比较

实验随机选择长度不等的汉语文本,应用本文算法和文献[8]的算法分别进行了重复字串抽取实验,并从算法耗用时间以及抽取数量两个方面进行了比较.实验中两个算法采用了同样的预处理过程和停用字符集.实验结果表明两个算法所抽取重复字串结果一致,这是由于两个方法都遵循"最长匹配"和"左结合原则"的原则.

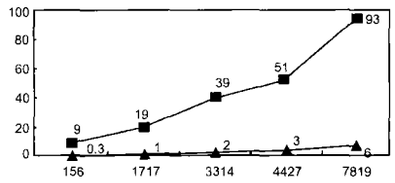

图 4　算法运行时间比较图

实验在一台 PⅢ800、256M 的 PC 上进行,算法耗用时间比较如图 4 所示.图中横坐标为文本长度,纵坐标为耗用时间(单位:毫秒).正方形符号线条为文献[8]算法,三角形符号线条为位置记忆跳跃匹配算法.可以看出本文的算法明显提高了处理速度.该算法非常适用于应用在对文本分类、检索等对文本处理速度有较高要求的自然语言处理领域.

3.3　存在的问题

字串的交叉组合在分词过程中造成的歧分现象是汉语文本自动处理中的难点.这种情况在重复字串抽取过程中同样也可能造成两难的选择困境.

位置匹配跳跃算法扫描文本的过程是从左到右,按匹配最大长度进行跳跃,这造成了交叉字串中部分重复字串的遗漏.例如图 3 中字串"AB"也是文本的重复字串,但由于它的重复位置与字串"EA"交叉,所以被遗漏.这说明本算法是不完备的抽取算法.在汉语自然语言的真实语料环境中这种歧分字串的现象可以从统计的角度来消除.

4　结论

在现有汉语文本串频统计算法的研究基础上,本文提出了位置记忆跳跃匹配算法,与以往算法相比,本算法具有更快的速度,以及更少的空间占用.该方法可以被广泛地应用于自动标引、分类、信息检索、信息抽取等各种信息处理系统中,对处理速度要求高的系统尤其适用.

参考文献:

[1] 王永成.中文信息处理技术及其基础[M].上海:上海交通大学出版社,1991.

[2] Tan Hong-ye. Research on method of automatic recognition of chinese place name based on transformation[J]. Journal of Software, 2001, 12

(11):1608 - 1613.
[3] 孙茂松,等.中文姓名的自动辨识[J].中文信息学报,1995,9(2):16 - 27.
[4] 张小衡,等.中文机构名称的识别与分析[J].中文信息学报,1997,11(4):21 - 32.
[5] 黄萱菁,等.基于机器学习的无需人工编制词典的切词系统[J].模式识别与人工智能,1996,9(4):297 - 303.
[6] 金翔宇,等.一种中文文档的非受限无词典抽词方法[J].中文信息学报,2001,15(6):33 - 39.
[7] 刘挺,等.串频统计和词性匹配相结合的汉语自动分词系统[J].中文信息学报,1998,12(1):17 - 25.
[8] 韩客松,等.无词典高频字串快速提取和统计算法研究[J].中文信息学报,2001,15(2):23 - 30.
[9] Jia-Lien Hsu, Chih-Chin Liu, L P Chen. Discovering nontrivial repeating patterns in music data[J]. IEEE Transactions on MULTIMEDIA, 2001,3(3):311 - 325.

作者简介:

马颖华 女,1973 年生于山东省济南市,博士研究生,现为上海交通大学计算机系博士研究,1993 年和 1996 年分别在山东大学获得学士和硕士学位,主要研究兴趣为概念标引及算法设计.

王永成 男,1939 年 10 月生于江苏省杨州市,上海交通大学计算机系教授、博士生导师,曾负责承担 748 工程、自然科学基金、863 项目等十多项,获得近十项国家及省部级大奖,发表论著 200 多篇,目前主攻方向为"网络信息智能处理".

CORRESPONDENCE