

一种基于全 Hash 的整词二分词典机制

彭焕峰¹, 丁宋涛²

(1. 南京工程学院计算机工程学院, 南京 211167; 2. 南京大学软件学院, 南京 210093)

摘 要: 为提高整词二分词典机制的分词效率, 分析现有分词词典机制, 提出一种基于全 Hash 的整词二分词典机制。该机制将首字相同的词条按字数分组, 并进行全词 Hash, 对 Hash 值相同的词条进行二分查找, 从而减少词条匹配的次數。理论分析和实验结果表明, 该机制的分词效率较高。

关键词: 中文分词; Hash 函数; 整词二分; 逐字二分; 最大匹配

Binary-seek-by-word Dictionary Mechanism Based on All-Hash

PENG Huan-feng¹, DING Song-tao²

(1. School of Computer Engineering, Nanjing Institute of Technology, Nanjing 211167, China;

2. Software Institute, Nanjing University, Nanjing 210093, China)

【Abstract】 According to the low efficiency of the traditional binary-seek-by-word dictionary mechanism for word segmentation, this paper gives a binary-seek-by-word dictionary mechanism for word segmentation based on all-Hash by analyzing many old dictionary mechanisms. The new mechanism divides the dictionary entry into some groups by character number the entry has, it uses the Hash value of word to reduce the number of string finding. Theoretical analysis and experiment results show that the new mechanism improves the efficiency of word segmentation.

【Key words】 Chinese segmentation; Hash function; binary-seek-by-word; verbatim binary search; maximum match

DOI: 10.3969/j.issn.1000-3428.2011.21.014

1 概述

目前中文分词算法大致可分为 3 类: 机械分词方法, 基于理解分词方法和基于统计分词方法^[1]。机械分词即把待分解的汉字串与词典中的词条进行匹配来判断是否一个词; 基于理解分词方法是在海量语言知识的基础上进行智能分词, 该类方法大多处于实验阶段; 基于统计分词方法即根据相邻字同时出现的概率作为依据进行分词, 但该方法局限性较大。目前大量采用的分词方法主要是基于词典的机械分词方法, 有 3 种典型的分词词典机制: 整词二分, TRIE 索引树及逐字二分^[1]。这 3 种分词词典机制各有优缺点, 本文针对整词二分词典机制分词效率较低的缺点, 提出一种基于全 Hash 的整词二分词典机制。

2 现有分词词典机制

2.1 传统词典机制

传统的 3 种分词词典机制的空间效率相差不多, 基于 TRIE 索引树和逐字二分的分词词典机制的时间效率大致相同, 较基于整词二分的词典机制均有大幅度的改善^[1]; 整词二分词典机制分词效率较低, 但词典维护简单; TRIE 索引树构造维护比较复杂, 而且都是单树枝, 有一定的空间浪费, 逐字二分由于仍然沿用整词二分词典机制, 因此, 并非严格意义上的逐字匹配, 分词的实现和词典维护不方便^[2]。

2.2 改进的分词词典机制

在 3 种经典分词机制的基础上, 很多研究者提出了改进的分词词典机制。

(1)改进的整词二分分词词典机制

文献[2]提出了一种改进整词二分法的中文分词词典设计。针对最大匹配分词算法, 将以某一汉字开头的词条按词条的长度分组, 每次进行匹配时, 只需要在对应的词组中进

行二分查找即可, 缩小了查找范围, 提高了查询效率。

(2)改进的逐字二分分词词典机制

文献[3]提出了一种改进逐字二分中文分词词典设计。针对词典中二字词、三字词居多的特点, 对首字相同的词条的次字按拼音首字母进行分组, 在进行次字匹配时缩小了查找范围, 提高了查询效率。第三字及其以后的字串匹配仍然采用传统的逐字二分分词词典机制。通过实验测试, 该机制在增加少量存储空间的情况下, 时间效率提升 26%^[3]。

(3)多次 Hash 分词词典机制

该类分词词典机制的改进大多是基于词典中二字词居多的特点, 对次字的查找也采用 Hash 查找的方法, 提高了分词效率。文献[4]针对逐字二分分词词典机制进行了改进, 增加了词次字 Hash 表, 采用链地址法解决次字 Hash 表中地址冲突问题, 词典正文中有序保存三字及三字以上词的剩余字部分。文献[5]虽然在具体的实现方法有一定区别, 但改进的思想上与文献[4]大同小异。文献[6]也是针对词典中二字词居多的特点, 直接针对双字词建立 Hash 索引, 将多字词也建立长词 Hash 索引表, 减少了字符串匹配的次數, 提高了词条匹配效率。

(4)基于双数组 TRIE 的分词词典机制

TRIE 索引树词典机制虽然词条匹配效率高, 但词典构造繁琐, 占用空间较多, 文献[7]提出用 2 个线性数组来表示 TRIE 索引树。双数组本质是一个确定的有限状态自动机, 每

基金项目: 南京工程学院科研基金资助项目“基于 Lucene 的全文搜索引擎研究”(QKJB2009026)

作者简介: 彭焕峰(1978—), 男, 讲师、硕士, 主研方向: 大数据量处理, 搜索引擎; 丁宋涛, 讲师、硕士

收稿日期: 2011-05-16 **E-mail:** penghf@njit.edu.cn

个节点代表自动机的一个状态, 根据变量的不同, 进行状态迁移, 当到达结束状态或者无法迁移时查询结束。文献[8]对双数组 TRIE 机制进行了优化和改进。

3 全 Hash 整词二分词典机制

3.1 词典设计

根据上文对现有分词词典机制的研究, 发现每一种机制都有优缺点, 且都有其存在的价值, 整词二分词典机制虽然分词效率较低, 但是词典维护方便, 分词程序便于扩展, 本文针对正向最大匹配算法设计并实现一种基于全 Hash 的分词词典机制。

全 Hash 整词二分词典机制由首字 Hash 表、词 Hash 节点、词 Hash 表和词碰撞表构成, 如图 1 所示。

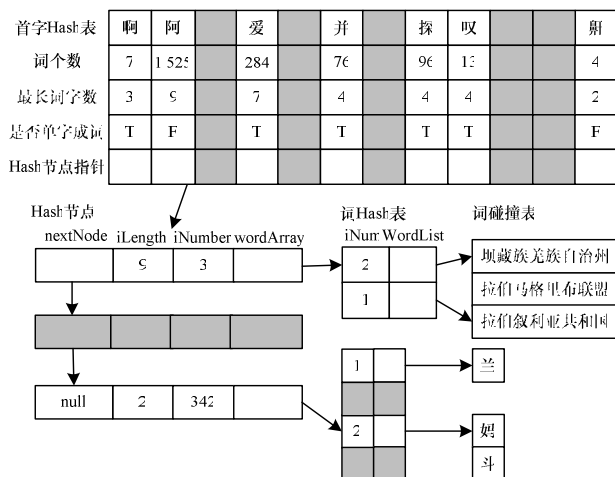


图1 词典组织形式

(1)首字 Hash 表

汉字在计算机中是以内码的形式存储, 可以根据内码获取汉字对应的区位码, 因此, 给定一个汉字, 可以通过 Hash 函数直接得到其在首字 Hash 表中的位置。Hash 函数设计如下: $Index = ((highByte - 160) \times 100 + lowByte - 160) - 1601$, 其中, lowByte 是汉字内码的低字节; highByte 是汉字内码的高字节。1)词个数: 记录以该字为首字的词的个数。2)最长词字数: 记录以该字为首字的最长词所包含的字数。3)是否单字成词: 标识该字是否可以单独成词。4)Hash 节点指针: 指向以该字为首字的第一个 Hash 节点。

(2)Hash 节点

对相同首字的词条按包含的字数进行分组, 然后进行全词 Hash(除首字外), Hash 节点记录了词长(iLength)、相同词长的词的个数(iNumber)、词 Hash 表的地址(wordArray)以及下一个 Hash 节点的指针(nextNode)。Hash 节点按词长倒序排列, 例如在词典中以“阿”为首字的最长词字数为 9, 则首字 Hash 表中“阿”字的 Hash 节点指针指向词长为 9 的 Hash 节点, 词长为 9 的 Hash 节点再指向词长为 8 的 Hash 节点。

(3)词 Hash 表

词 Hash 表的大小设置为词长相同的词的数量, 要选取一个的尽可能简单的 Hash 函数, 否则会增加分词的时间, 而且 Hash 结果尽量均匀分布, 产生尽量少的碰撞。

词 Hash 表由 iNum 和 wordList 两部分组成, 其中 iNum 记录了 Hash 值相同的词条的个数; wordList 指向保存词条的动态数组, 即词碰撞表。

(4)词碰撞表

词碰撞表是一个动态数组, 对于词典中首字相同且词长

相同的词条, 如果 Hash 值相同, 则以动态数组的形式保存, 动态数组中保存的是除首字之外的剩余字串, 以节省一定的空间。例如以“阿”为首字的词长为 9 的词条中, “坝藏族羌族自治州”和“拉伯马格里布联盟”的 Hash 值相同, 则动态数组有 2 个元素。之所以设计为用动态数组解决碰撞, 而不用链地址法, 是考虑到若相同 Hash 值的词条较多的话, 采用链地址法只能顺序查找, 而采用动态数组实现, 可以按顺序保存碰撞的词条, 查找时采用二分查找。

该分词词典机制的出发点是通过对待匹配字串进行 Hash 查找, 以便尽可能的直接定位词条, 减少字符串匹配的次, 因此词条的碰撞率不能过高, 否则会影响分词效率。

实验采用的词典共 266 233 个词条, 词 Hash 表的大小设置为词长相同的词的数量, 假设为 N , 通过调用 Java 语言词条字符串对象的 hashCode()方法获取词条对应的 Hash 值 H , 如果 H 为负, 则求其绝对值, 假设为 $hValue$ 。则词条的 Hash 函数可以简单的表示为: $Index = hValue \% N$ 。通过此 Hash 函数在内存中建立词典结构, 共产生 51 314 次碰撞, 碰撞率为 19.28%。若要进一步降低碰撞率, 可以词 Hash 表的长度设置的大一些, 例如 $2 \times N$, 但会增加一定的内存开销。

3.2 分词算法

该分词词典机制适用于正向最大匹配算法和逆向最大匹配算法, 本文以正向最大匹配算法为例使用该词典机制进行分词。

(1)从 $S = C_1C_2 \dots C_n$ 中读取第一个字 C_1 , 从首字 Hash 表中获取以该字为首字的最长词的字数 m , 则取字符串 $Str = C_1C_2 \dots C_m$ 为待匹配字符串, 如果 S 剩余待分词的字数不足 m , 则取 Str 为 S 的剩余待分词的字符串。

(2)如果 Str 的长度大于 1, 转第(3)步; 如果 Str 的长度为 1, 则在首字 Hash 表中找到对应的位置, 若是否单独成词标志为 F, 说明 Str 不是一个词, 否则说明 Str 是一个词, 分出该词。设置待切分语句 $S = C_2C_3 \dots C_n$, 转第(1)步。

(3)在 C_1 对应的 Hash 节点链表中查找词长为字串 Str 长度的节点, 如果没有找到, 则 Str 字串不是一个词, 则去掉 Str 的最后面的一个字, 转第(2)步。如果找到对应的 Hash 节点, 则计算 Str (除去首字, 因为首字不保存)的 Hash 值, 得到在词 Hash 表中的位置:

1)如果对应的 iNum 等于 0, 说明 Str 不是一个词;

2)如果 iNum 等于 1, 则比较 Str (去掉首字)与词碰撞表中的词进行比较, 如果相等, 则 Str 成词, 否则不成词;

3)如果 iNum 大于 1, 则在词碰撞表中进行二分查找, 若找到则成词, 否则不成词。

上述 3 种情况若不成词, 则去掉 Str 的最后面的一个字, 转第(2)步; 若成词, 则在 S 中分出一个词 Str , 将语句 S 设置为除去 Str 的剩余字串, 转第(1)步。

4 实验结果及分析

选择传统的整词二分词典机制、文献[2]中提出的改进整词二分词典机制(为便于表述, 下文称为改进整词二分)、以及本文提出的机制进行分词实验, 实验采用的词典共 266 233 个词条, 经实验统计, 词典中共有汉字 754 317 个。

4.1 内存占用对比

(1)传统整词二分词典

传统整词二分词典所占空间为首字散列表、词索引表、词典正文三项所占空间之和。

1)首字散列表: 每单元占 8 Byte, 共有 7 614 个单元, 所

占内存为: $M_{th}=7\ 614 \times 8=60\ 912$ Byte。

2)词索引表: 每单元占 4 Byte, 共有 266 233 个单元, 所占内存为: $M_{wi}=266\ 233 \times 4=1\ 064\ 932$ Byte。

3)词典正文: 每个汉字占 2 Byte, 共有 754 317 个汉字, 所占内存为: $M_{wd}=754\ 317 \times 2=1\ 508\ 634$ Byte。

则传统整词二分词典占内存为:

$$M_{cls}=M_{th}+M_{wi}+M_{wd}=2634\ 478 \text{ Byte}$$

(2)改进整词二分词典

根据文献[2]的设计所占空间由首字散列表、链节点和动态数组 3 项组成。

1)首字散列表: 每单元占 10 Byte, 共有 7 614 个单元, 所占内存为: $M_{imth}=7\ 614 \times 10=76\ 140$ Byte。

2)链节点: 每单元占 12 Byte, 实验统计共 14 521 个节点, 所占内存为: $M_{imnd}=14\ 521 \times 12=174\ 252$ Byte。

3)词典正文: 由于词典正文中不保存首字, 词典中共有汉字数用 $wTotal$ 表示, 词典中的单字词用 $wSingle$ 表示, 词典中的词条数用 $wNum$ 表示, 词典正文所占空间为:

$$M_{imwd}=(wTotal-(wNum-wSingle)) \times 2 = (754\ 317-(266\ 233-2\ 730)) \times 2=970\ 708 \text{ Byte}。$$

则改进整词二分词典占内存为:

$$M_{imcls}=M_{imth}+M_{imwi}+M_{imwd}=1\ 221\ 100 \text{ Byte}$$

传统整词二分词典机制词索引表占用了较多空间, 且将词条的所有汉字都存储, 因此, 改进的整词二分词典占用空间较传统整词二分词典占用空间小很多。

(3)全 Hash 整词二分词典

全 Hash 整词二分词典所占空间由首字散列表、Hash 节点、词 Hash 表、词碰撞表 4 项组成。

1)首字散列表: 每单元占 12 Byte, 共有 7 614 个单元, 所占内存为: $M_{hsth}=7\ 614 \times 12=91\ 368$ Byte。

2)Hash 节点: 每单元占 11 Byte, 实验统计共 14 521 个节点, 所占内存为: $M_{hsnd}=14\ 521 \times 11=159\ 731$ Byte。

3)词 Hash 表: 每个单元占 6 Byte, 由于单字词(共 2 730 个)并不存储, 因此共有 263 503 个单元, 所占内存为: $M_{hshs}=263\ 503 \times 6=1\ 581\ 018$ Byte。

4)词碰撞表: 词碰撞表用来保存词典正文, 由于首字不存储, 因此所占空间与改进的整词二分词典相同, 所占内存为: $M_{hswd}=970\ 708$ Byte。

则全 Hash 整词二分词典占内存为:

$$M_{hscsls}=M_{hsth}+M_{hsnd}+M_{hshs}+M_{hswd}=2\ 802\ 825 \text{ Byte}$$

全 Hash 整词二分词典所占内存比改进的整词二分词典要多, 与传统整词二分词典差别不大。

4.2 时间复杂度对比

采用正向最大匹配算法, 分别使用传统整词二分词典、改进整词二分词典和全 Hash 整词二分词典进行分词, 比较 3 种机制的分词效率。采用 3 种类型的测试: (1)对词典中的

所有词都查询一遍; (2)按词频对词典中的所有词查询一遍; (3)选择 2 个中文文本进行分词, 其中测试集 1 和测试集 2 的大小分别为 354 KB 和 4 328 KB。表 1 为不同机制词条比较次数对比, 表 2 为不同机制分词时间对比。

表 1 不同机制词条比较次数对比

词典机制	测试 1	测试 2	测试 3(测试集 1)	测试 3(测试集 2)
传统整词二分词典	1 737 485	6 949 523	2 576 524	31 028 153
改进整词二分词典	1 358 633	5 434 617	1 423 190	17 295 388
全 Hash 整词二分词典	364 872	1 459 764	217 467	2 649 849

表 2 不同机制分词时间对比

词典机制	测试 1	测试 2	测试 3(测试集 1)	测试 3(测试集 2)
传统整词二分词典	367	974	739	7 236
改进整词二分词典	281	763	553	6 015
全 Hash 整词二分词典	215	539	391	4 423

可以看出, 采用全 Hash 整词二分词典进行分词时, 词条比较的次数大大低于另外 2 种机制。虽然在分词时, 增加了 Hash 函数的计算时间, 但分词的时间仍比另外 2 种机制有了较大提高。

5 结束语

本文对整词二分词典机制进行改进, 提出了一种基于全 Hash 的整词二分词典机制。全 Hash 整词二分词典所占内存比经典的整词二分词典略高, 但分词效率提高了约 40%, 且词典维护方便。因此, 本文提出的词典机制具有一定的实用价值, 目前使用本机制实现的分词程序已成功应用到相关课题项目中。下一步工作将研究如何提高词典的加载速度。

参考文献

- [1] 孙茂松, 左正平, 黄昌宁. 汉语自动分词词典机制的实验研究[J]. 中文信息学报, 1999, 14(1): 1-6.
- [2] 谭骏珊, 吴惠雄. 一种改进整词二分法的中文分词词典设计[J]. 信息技术, 2009, (5): 40-42, 45.
- [3] 杨毅, 王禹桥. 一种改进逐字二分中文分词词典设计[J]. 湘潭大学: 自然科学学报, 2009, (4): 124-128.
- [4] 张科. 多次 Hash 快速分词算法[J]. 计算机工程与设计, 2007, 28(7): 1716-1718.
- [5] 李庆虎, 陈玉健, 孙家广. 一种中文分词词典新机制-双字哈希机制[J]. 中文信息学报, 2002, 17(4): 13-18.
- [6] 吴晶晶, 荆继武, 聂晓峰, 等. 一种快速中文分词词典机制[J]. 中国科学院研究生院学报, 2009, 26(5): 703-711.
- [7] Aoe, J. An Efficient Digital Search Algorithm by Using a Double-array Structure[J]. IEEE Transactions on Software Engineering, 1989, 15(9): 1066-1077.
- [8] 王思力, 张华平, 王斌. 双数组 Trie 树算法优化及其应用研究[J]. 中文信息学报, 2006, 20(5): 24-30.

编辑 金胡考

(上接第 39 页)

参考文献

- [1] 夏秀峰, 谢光宇, 石祥滨, 等. 基于置信区间的偏离群数据检测方法[J]. 计算机工程, 2008, 34(21): 12-14.
- [2] He Zengyou, Xu Xiaofei. FP-outlier: Frequent Pattern Based Outlier Detection[J]. Computer Science and Information Systems, 2005, 2(1): 103-118.
- [3] Skowron A. The Discernibility Matrices and Functions in Information Systems[M]. [S. l.]: Kluwer Academic Publishers, 1992.
- [4] 葛浩, 李龙澍, 杨传健. 新的可分辨矩阵及其约简方法[J]. 控制与决策, 2010, 25(12): 1891-1895.
- [5] Chen Yumin, Miao Duoqian, Zhang Hongyun. Neighborhood Outlier Detection[J]. Expert Systems with Application, 2010, 37(12): 8749-8750.
- [6] Knorr E, Ng R, Tucakov V. Distance-based Outliers: Algorithms and Applications[J]. The VLDB Journal, 2002, 8(3): 237-253.
- [7] Bay S D. The UCI KDD Repository[DB/OL]. [2010-06-20]. <http://kdd.ics.uci.edu>.

编辑 顾姣健

