双数组 Trie 树索引的可操作性研究①

Research on the Operability of the Double-Array Trie Structure

廖 敏 褚颖娜 宋继华 (北京师范大学 信息科学与技术学院 北京 100875)

摘 要: 双数组是组织和实现 Trie 树的一种数据结构。双数组 Trie 树索引实现的是一种线性时间复杂度的搜索机制,因此被广泛的应用于信息检索和中文分词等领域。然而双数组 Trie 树索引建立后不易于更新,限制了这种索引的现实应用。在前人的双数组 Trie 树优化索引构造的基础上,分析了插入和删除操作的所有可能情况,提出了对双数组 Trie 树索引进行相关操作的算法。最后分析了其时间和空间开支,并用实验结果证明了其可行性。

关键词: 信息检索 Trie 树 双数组 稳定词 空间开支

Trie 树(键树)是一种用于快速检索的多叉树结构。Trie 树把要查找的关键词看作一个字符序列,每个结点存储该序列中的一个字符,根据关键词中字符的先后顺序构造的一种树结构[1]。一个 Trie 树的例子如图 1 所示。Trie 树在计算机内可以用矩阵、链表等组织形式存储。双数组 Trie 树数据结构采用了两个一维数组表示一棵 Trie 树,最早由 Aoe 在 1989 年提出,它综合了矩阵的快速存取和链表的空间利用率高的优点[2]。

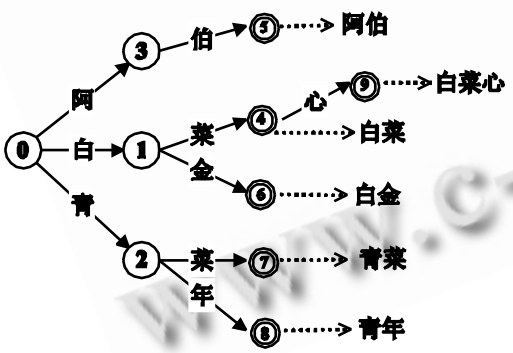

图 1 Trie 结构树

1 引言

针对双数组 Trie 树的空间利用率不高的缺点,前人已经多次研究过这种数据结构,并实现了不同程度的优化[3-5]。目前讨论的比较多的是双数组 Trie 索引树的建立,但对它的插入、删除等更新操作却很少有论述,而这在信息检索领域的许多应用中恰好有必要,比如多条件检索。在中文分词中,由于人们对词的界定的不统一,因此实时添加与删除词语对不同用户来说也是十分必要的。本文在文献[3]提出的双数组的构建方法的基础上,利用 Java 语言实现了其构造算法,并进一步探讨及实现了插入和删除操作。

2 双数组Trie树

2.1 双数组 Trie 树的模型

由图 1 可知,Trie 树可以看成是一个有限状态自动机,其中圆点代表不同的状态,双线圈代表可终止状态,弧代表转移条件。定义函数 g(s, c)=t 为状态之间的转移函数,其中 s 表示当前状态,c 表示转移条件,t 表示下一个可接受状态。当 Trie 树用两个一维数组(设为 base 和 check 数组)表示时,函数 g(s, c)=t 拆分成如下表示:

(1) t = base[s] + c

(2) check[t] = s

第 1 个式子表示 t 的位置可由 s 的 base 值和 c 的序列码来计算。假设 s 的 n 个孩子结点对应的序列码分别为 $s_1, s_2 \cdots, s_n$ $(n \geq 1)$,那么 base[s]的计算过

① 基金项目:国家社科基金(05BYY022)
收稿时间:2009-02-16

程如下：

(1) base[s]=min{k|base[s1+k]=check[s1+k]=base[s2+k] = check[s2+k] =…=base[sn+k] = check[sn+k] =0 且 k≥1}；

(2) 若 s 是可终止结点但不是叶子结点，则 base[s]= - base[s]；

(3) 若 s 是叶子结点，则 base[s]=-∞(-∞表示负无穷大)。

第 2 个式子限定了 t 的前一个状态为 s。已知一个状态和转移条件，便可以唯一确定下一个可接受状态。按照以上公式便可以计算出一棵 Trie 树所对应的唯一双数组。

2.2 双数组 Trie 树的构造过程

以图一中的 Trie 树为例，按照深度优先将所有字符排序为 1：阿 2：白 3：青 4：伯 5：菜 6：金 7：年 8：心，我们称之为字符序列码表。再根据双数组 Trie 树的模型，双数组的构造过程如下：

(1) 将首结点(图中的结点 0)入队后出队，并将其所有孩子结点入队。初始化以它们的序列码为下标的 base 和 check 的值为 0；

(2) 由文献[3]提出的优化方法，我们选择拥有孩子结点数目最多的一个结点出队(孩子数目相同时按照入队顺序排序)，这里选择结点 1 作为下一个出队的结点；

(3) 将结点 1 出队，并将其所有孩子结点(结点 4 和 6)入队，此时计算结点 1 的 base 值。由于‘菜’和‘金’的序列码分别为 5 和 6，因此 base(1)可取最小值为 1，因为 base(5+1)=check(5+1)=base(6+1)=check(6+1)=0。更改结点 4 和 6 的 check 值为结点 1 的数组下标值，即 check(5+1)=check(6+1)= 1；

(4) 为了解决冲突，当结点为可终止状态(即可成为一个词)时，修改其 base 值为原值的相反数，若是叶子结点，则修改其 base 值为负无穷大；

(5) 选择下一个拥有孩子节点数目最多的结点出队，循环步骤(3)和(4)，直到队空。

按照上面的方法构造的双数组如下：

	1	2	3	4	5	6	7	8	9	10
BASE	1	1	3	0	-∞	-1	-∞	-∞	-∞	-∞
CHECK	0	0	0	0	1	2	2	3	6	3
STATES	阿	白	青		(阿)伯	(白)菜	(白)金	(青)菜	(白)(菜)心	(青)年

其中 states 是程序运行过程中的临时状态信息，双数组的每个下标对应一个状态，base 和 check 值均为 0 时表示空状态。

2.3 双数组 Trie 树的检索机制

当我们判断一个词是否在索引中时，可以由 base 和 check 数组的简单运算得到结果。比如确定“白菜心”是否在索引中的过程如下：

(1) 指针指向第一个字符‘白’，其序列码为 2，得到 base(2)=1；

(2) 指针移向下一个字符‘菜’，其序列码为 5，检验 check(base(2)+5) =2，与状态点‘白’的数组下标相同，说明它的前一个状态是‘白’；

(3) 指针移向下一个字符‘心’，其序列码为 8，检验 check(|base(6)| + 8) = 6，正好是状态点‘(白)菜’的数组下标，因此指针继续移动，但此时到达了词尾，检验 base(9) = -∞<0，即当前状态为可终止状态，所以最终确定“白菜心”在索引中。

3 双数组Trie树索引的可操作性分析

通过分析双数组 Trie 树的构造过程，不难得出该数据结构的特点如下：

(1) 确定一个字符串是否是一个词时，首先要找到首字的状态点，因此首字必须独占一个状态点，即下标为首字的序列码的位置存放该首字状态点；

(2) 任一状态点的移动，会影响其 Trie 树中父节点的 base 值的选择以及兄弟结点位置的变动，而兄弟结点的移动又须变更相应的子节点的 check 值；

(3) 双数组中存在状态为空的结点，降低了数组的空间利用率。

根据以上几个特点，我们分析并设计了双数组的添加和删除操作。

3.1 插入操作

设待插入的词或其子串为‘$C_1C_2C_3\cdots$’。由双数组的结构可以看出，当索引中已经存在以单个字符 C_1 为状态的状态点时，所需的操作与建立双数组时的相同，不影响双数组的整体结构，我们把符合这种情况的词或其子串统称为“稳定词”。根据首字符 C_1 的不同可将插入操作分为以下四大类：

(1) 字符 C_1 不在序列码表中，这时只需把 C_1 加入序列码表中，设定其码值为数组的大小(即最大下标加 1)，以保证其拥有首字状态。此时‘$C_1C_2\cdots$’便成

为了一个“稳定词”，后续插入操作便是对该“稳定词”的简单操作；

(2) 索引中存在以字符 C_1 以及‘$(C_1\cdots C_{i-2})C_{i-1}$’为状态的状态点，但字符 C_i 不在序列码表中。这时要把 Ci 加入序列码表，调整其兄弟结点(诸如‘$(C_1\cdots C_{i-2}C_{i-1})O$’的状态点，其中 O 是一个字符)的插入位置，修改父节点‘$(C_1\cdots C_{i-2})C_{i-1}$’的 base 值，并修改其兄弟结点的孩子的 check 值。做了这些调整之后，这时词的子串‘$C_iC_{i+1}\cdots$’便是一个“稳定词”，后续插入操作便是对该“稳定词”的简单操作；

(3) 索引中存在以字符 C_1 以及‘$(C_1\cdots C_{i-2})C_{i-1}$’为状态的状态点，字符 C_i 也在序列码表中，但状态点‘$(C_1\cdots C_{i-2}C_{i-1})C_i$’不存在索引中。此时需要做的是调整其兄弟结点(与上同)的插入位置，修改父节点‘$(C_1\cdots C_{i-2})C_{i-1}$’的 base 值，并修改兄弟结点的孩子的 check 值，此时子串‘$C_iC_{i+1}\cdots$’成为了一个“稳定词”，后续插入操作便是对该“稳定词”的简单操作；

(4) 字符 C_1 在序列码表中但不构成首字状态结点。查找以 C_1 的序列码为下标的状态点 S，修改 S 的父节点的 base 值，移动 S 及其所有孩子，同时修改其孙子结点的 check 值，最后将 C_1 作为首字状态结点插入原来状态点 S 的位置。则‘$C_1C_2\cdots$’便转换成为了一个“稳定词”，后续插入操作便是对该“稳定词”的简单操作。

以情况(2)为例，加入词“青壮年”，其操作过程如下：

第 1 步 由于字符‘壮’不在序列码中，因此需要首先修改序列码表。

在序列码表中添加字符‘壮’，此时序列码表为 1：阿 2：白 3：青 4：伯 5：菜 6：金 7：年 8：心 9：壮

第 2 步 更新‘青’状态点的 base 值，移动其所有孩子(这里正好可以在后面直接插入状态点‘(青)壮’，因此不需移动)。得到如下的结果：

	1	2	3	4	5	6	7	8	9	10	11	12
BASE	1	1	3	0	-∞	-1	-∞	-∞	-∞	-∞	0	0
CHECK	0	0	0	0	1	2	2	3	6	3	0	3
STATES	阿	白	青		(阿)伯	(白)菜	(白)金	(青)菜	(白)(菜)心	(青)年		(青)壮

第 3 步 此时“壮年”是一个稳定词，按照建立索引时的方法将状态结点‘(青壮)年’加入便可：

	1	2	3	4	5	6	7	8	9	10	11	12	13
BASE	1	1	3	0	-∞	-1	-∞	-∞	-∞	-∞	0	6	-∞
CHECK	0	0	0	0	1	2	2	3	6	3	0	3	12
STATES	阿	白	青		(阿)伯	(白)菜	(白)金	(青)菜	(白)(菜)心	(青)年		(青)壮	(青)(壮)年

如再加入词“年糕”，则属于情况(4)，其过程如下：

第 1 步 找到‘年’的序列码为 7，发现状态数组 base(7)不为空状态，此时必须腾出该状态。原数组中 base(7)所对应的状态是“(白)金”，父结点是‘白’，调整所有以‘白’为父结点的所有结点的位置，并将状态点‘年’插入 base(7)这个位置：

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BASE	1	9	3	0	-∞	0	0	-∞	-∞	-∞	0	6	-∞	-1	-7
CHECK	0	0	0	0	1	0	0	3	14	3	0	3	12	2	2
STATES	阿	白	青		(阿)伯		年	(青)菜	(白)(菜)心	(青)年		(青)壮	(青)(壮)年	(白)菜	(白)金

第 2 步 此时“年糕”已是一个稳定词，只需插入状态点‘年’的后续结点‘(年)糕’(并将字符‘糕’插入序列码表)，此时序列码表为 1：阿 2：白 3：青 4：伯 5：菜 6：金 7：年 8：心 9：壮 10：糕

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BASE	1	9	3	0	-∞	0	6	-∞	-∞	-∞	0	6	-∞	-1	-7	-∞
CHECK	0	0	0	0	1	0	0	3	14	3	0	3	12	2	2	7
STATES	阿	白	青		(阿)伯		年	(青)菜	(白)(菜)心	(青)年		(青)壮	(青)(壮)年	(白)菜	(白)金	(年)糕

3.2 删除操作

设待删除的词为‘$C_1C_2\cdots C_n$’(假设该词在索引中存在)，分为两种情况：

(1) 如果状态‘$(C_1C_2\cdots C_{n-1})C_n$’的 base 值为负无穷，则说明该状态对应着一个叶子结点，因此只需将其 $base$ 值修改为 0 即可；

(2) 否则说明该词属于某个词的前半子串（比如“白菜”是“白菜心”的前半字串），将其 base 值修改为原值的相反数即可。

以删除“青年”为例，因为 $base(10)=-\infty$，修改 $base(10)=check(10)=0$。如要删除“白菜”，因为 $base(14)=-1$，修改 $base(14)=|base(14)|=1$。

4　实验结果及分析

本文所有实验的硬件环境为 CPU CoreDuo2.0，内存 1G；软件环境为 Ubuntu7.10(Linux)，Eclipse3.3，JRE1.6，所有字符采用 GB18030 编码(向下兼容 GBK 和 GB2312 编码)。

4.1 建立双数组索引

实验数据为 15000 条成语词典、从搜狗实验室网站(http://www.sogou.com/labs/)上下载的 18 万多常用词的词典中筛选出的 123211 个词，总共 138211 个词条，所有词条按字典的拼音排序。得出的实验结果如表 1 和图 2 所示。

表 1　双数组 Trie 树索引建立的部分实验结果

词数目	平均词长	单字数	数组大小	耗时(ms)
1000	2.835	773	1887	86
10000	2.771	2883	21370	2882
50000	2.703	4859	104717	64542
100000	2.692	5911	199220	268784
138211	2.693	6388	270879	497605

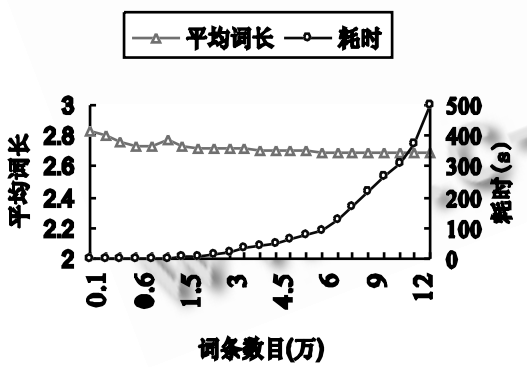

图 2　双数组 Trie 索引建立耗时图

4.2 双数组的操作

由于删除操作的速度非常快(查找和赋值操作都只需各一次)，所以我们主要分析了插入操作的情况。这里的“原索引”与 4.1 中建立的双数组 Trie 索引对应，插入的 200 个词条是从不在原索引中的词中随机产生的。其中平均每个词的插入时间包含产生随机数的时间。实验结果如表 2 和图 3 所示，其中“空间开支”指的是插入 200 词条的前后索引数组大小的变化率。设原索引中的数组大小为 S1，插入 200 词条后的索引数组大小为 S2，则空间开支 exp 由下式得到：

$$exp=(S_2-S_1)/S_2*100\%$$

表 2　双数组 Trie 索引插入操作的部分实验结果

插入词条		原索引中		插入后	空间开支	插入耗时(词/ms)
数目	平均词长	词数	数组大小	数组大小		
200	2.59	1000	1887	3507	46.2%	8.985
200	2.69	10000	21370	23494	9.04%	19.26
200	2.71	50000	104717	106667	1.83%	20.85
200	2.63	120000	237352	239575	0.93%	22.785

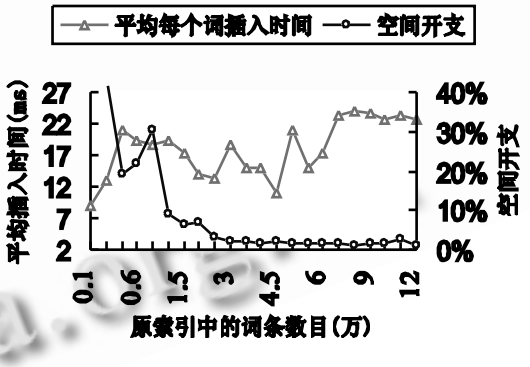

图 3　双数组 Trie 索引插入词条效率图

由上图可知，当原索引的词条数目在 3 万以上时，平均每插入 200 词的空间开支低于 5%。平均每插入一个词的时间稳定在 24ms 以内，当原索引的词条数目大于 7 万时，插入时间稳定在 23ms 左右，空间开支控制在 1%以内。

5　总结

双数组Trie树索引广泛地应用于信息检索和中文分词等领域，在双数组索引建立好了之后，对双数组

(下转第 52 页)

(上接第 56 页)

的操作具有十分重要的现实意义。双数组 Trie 树索引的建立是本文的基础，而双数组 Trie 索引的操作是本文的重点。

本文采用 Java 语言实现了双数组 Trie 索引的构造，并进一步提出和实现了一种对双数组 Trie 树索引进行操作的方法。实验结果表明当索引到一定规模(3 万词条以上)时，插入和删除操作在时间和空间上都是允许的、可行的。

参考文献

1 邓俊辉.数据结构与算法 Java 语言描述.北京:机械工业出版社, 2006:89－102.

2 Aoe J.An Efficient Digital Search Algorithm by Using a Double-Array Structure. IEEE Transactions on Software Engineering. 1989,15(9):1066－1077.

3 王思力,张华平,王斌.双数组 Trie 树算法优化及其应用研究.中文信息学报, 2006,20(5):24－30.

4 Oono M, Atlam ES. A fast and compact elimination method of empty elements from a doublearray structure. Software-Practice and Experience, 2003, 33(8):1229－1249.

5 Yata S, Oono M, A compact static double- array keeping character codes. Information Processing and Management, 2007,43(3):237－247.