RM 树:一种支持字符串相似性操作的索引

王金宝[1)] 高 宏[1)] 李建中[1)] 杨东华[2)]

[1)](哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

[2)](哈尔滨工业大学基础与交叉科学研究院高性能计算中心 哈尔滨 150001)

摘 要 字符串相似性操作在很多领域中被广泛应用,如数据清洁、信息集成等.现有研究工作主要为基于 q-Gram 和倒排索引的内存方法,在处理大量数据时具有以下缺点:内存消耗大、更新效率低、支持操作类型有限.现有的外存索引 B^{ed} 树无法将相似的字符串聚类,在查询处理过程中导致了较大的 I/O 代价.该文设计了支持多种字符串相似性操作的 RM 树索引,消除了现有内存方法的缺点,并通过字符串聚类的方法提高了相似性操作的效率.该文通过大量实验结果证明了 RM 树的有效性.

关键词 字符串;相似性;索引;查询处理;连接处理

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2011.02142

RM-Tree: An Index Supporting String Similarity Operations

WANG Jin-Bao[1)] GAO Hong[1)] LI Jian-Zhong[1)] YANG Dong-Hua[2)]

[1)](*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

[2)](*Center for High Performance Computing, Academy of Fundamental and Interdisciplinary, Harbin Institute of Technology, Harbin* 150001)

Abstract String similarity processing is well adopted in various fields such as data cleaning, information integration, spelling check and bioinformatics. With the rapid growth of information, processing strings over massive datasets becomes a significant basic operation. Existing solutions based on q-Gram and inverted lists suffer from the following disadvantages: large storage cost, poor efficiency of updates and limited types of operation. At meanwhile, most of these methods are main memory based. B^{ed}-tree is a recently proposed tree structured, disk resident index which supports diverse operations. B^{ed}-tree fails to cluster similar strings together, thus incurs large I/O costs while string similarity processing. This paper proposes a disk resident index RM-tree which clusters similar strings together while eliminating the shortcomings of q-Gram and inverted list based solutions. RM-tree supports multiple types of string similarity operations such as range query, top-k query and string similarity join. This paper presents the construction method of RM-tree and its query processing algorithms. RM-tree is tested with extensive experiments including index construction and different types of query processing in terms of I/O cost and time consumption with two real world datasets and a synthetic dataset. Experiment results show that RM-tree is efficient for supporting string similarity operations, and provides better performance than B^{ed}-tree.

Keywords string; similarity; index; query processing; join processing

---

收稿日期:2011-08-29;最终修改稿收到日期:2011-09-20.本课题国家“九七三”重点基础研究发展规划项目基金(2012CB316200)、国家自然科学基金(60903016,61003046,60533110,60773063,61173022)、黑龙江省自然科学基金(F201031)、中国博士后科学基金(20110491064)、黑龙江省博士后基金(LBH-Z09140)、哈工大科研创新基金“中央高校基本科研业务费专项资金”(HIT.NSRIF.2010060)、哈工大优秀青年教师培养计划(HITQNJS2009.063)资助.**王金宝**,男,1983 年生,博士研究生,主要研究方向为云计算环境下海量数据索引及查询处理技术.**高 宏**,女,1966 年生,博士,教授,博士生导师,主要研究领域为无线传感器网络、物联网、海量数据管理和数据挖掘.**李建中**,男,1950 年生,教授,博士生导师,主要研究领域为物联网、无线传感器网络、数据库和海量数据处理.**杨东华**(通信作者),男,1976 年生,博士,讲师,主要研究方向为海量数据管理和数据密集型计算等.E-mail:yang.dh@hit.edu.cn.

1 引　言

字符串相似性操作包括相似性查询操作和相似性连接操作.这些操作被广泛应用于众多领域,如数据清洁、信息集成、拼写校验以及生物信息处理等.当前,随着信息生成速度的增加和信息存储能力的增长,大规模数据集合日益普遍.字符串数据是其中的重要组成元素.Google 拥有的 N-Gram 数据集合包含 1T 条元组;在生物信息方面,GeneBank 数据集合大小达到 416 GB,包含超过 100 万条记录.在如此庞大的字符串数据集合中,定位相似性操作涉及的数据是一项具有挑战性的任务.

当前,针对字符串相似性操作的研究工作主要集中在两个方面:字符串相似性查询和字符串相似性连接.大部分研究工作基于 q-Gram 以及倒排索引.这些方法在处理大规模字符串数据集合时有以下劣势:(1)内存消耗过大.现有的基于 q-Gram 和倒排索引的方法都针对内存设计,没有给出有效的外存算法.(2)存储代价过大.即使将 q-Gram 以及倒排索引存储在磁盘上,其存储代价将与原有数据大小不相上下,甚至超过原始数据.(3)更新代价大.对原始数据的每一次更新,都需要修改若干 Gram 对应的倒排链表.在实际应用中,倒排链表的长度与数据集合的大小成正比,因此更新若干倒排链表的代价不容忽视.(4)支持查询的类型有限.只能支持范围查询以及连接操作,其它复杂查询如 top-k 查询等则无法直接支持.B^{ed} 树是一种树形索引结构,用于支持字符串相似性操作,这也是与本文工作最为相关的工作.B^{ed} 树为字符串数据设定一维的顺序,将数据排序并按照该顺序建立 B^+ 树索引存储数据.B^{ed} 树中使用 3 种字符串排序,分别为 Dictionary Order、Gram Counting Order 和 Gram Location Order.在查询过程中,不同的字符串排序方法分别结合了不同的过滤器用以选择包含查询结果的子树.然而,使用一维顺序排列字符串无法有效地将相似的字符串聚类,降低了过滤器的效果.因此,B^{ed} 树在处理查询时,即使查询结果很少,查询处理也会引起很多 I/O 操作.

本文设计了一种支持字符串相似性操作的索引结构 RM 树.RM 树将相似的字符串聚类在一起,减少了 B^{ed} 树中使用一维排序方法引起的昂贵的查询处理 I/O 开销.RM 树通过字符串的 Gram 集合,将相似的字符串组织到相近的叶节点中.字符串首先被转换成固定维度的 Gram 向量,并使用与 R 树类似的方法管理和操作这些 Gram 向量.不同于 R 树,RM 树在每个节点中加入该子树中 Gram 集合的 signature,进一步刻画该子树中字符串的内容,增加查询处理过程中,子树之间的过滤能力.由于在 R 树中加入位图 Bitmap 作为 signature,本文设计的索引命名为 RM 树.RM 树支持多种字符串相似性操作包括范围查询、top-k 查询、连接操作.实验结果表明,RM 树有效地降低了字符串相似性操作过程中的 I/O 代价,减少了时间开销.

本文的主要贡献在于:

(1)本文提出了一种新的索引结构 RM 树,用于支持多种字符串相似性操作.RM 树节点中保存 Gram 向量和位图信息用于削减搜索空间.

(2)本文给出了 RM 树的构建方法,即适用于字符串数据的插入目标选择方法,节点分裂方法.

(3)本文设计了 RM 树中处理多种字符串相似性操作的方法,包括范围查询、top-k 查询以及连接操作.RM 树在查询处理过程中利用节点中存储的 Gram 向量和位图信息有效地削减了搜索空间.

(4)本文通过大量实验测试了 RM 树处理各种字符串相似性操作的代价,包括 I/O 代价和时间开销.实验部分还测试了建立 RM 树的时间开销.

本文第 2 节介绍各种字符串相似性操作的定义.第 3 节给出 RM 树的结构以及 RM 树的插入算法.RM 树处理各种字符串相似性查询的方法在第 4 节中给出,包括范围查询处理、top-k 查询处理以及连接操作的处理方法.第 5 节简介 RM 树基于其它字符串相似性度量的应用.第 6 节通过真实数据和人工合成数据上的实验验证了 RM 树索引结构的有效性.第 7 节介绍 RM 树的相关工作.最后,第 8 节给出本文结论.

2 背景知识

本节介绍编辑距离、q-Gram、字符串相似性操作的定义,包括字符串相似性查询和连接操作.

2.1 编辑距离和 q-Gram

给定字母表 Σ,字符序列 $s\in(\Sigma)^*$ 定义为 Σ 中字符串,记 $s[i]$为字符串 s 的第 i 个字符,$|s|$ 为 s 的长度.编辑距离(也叫做 Edit Distance),是指两个字符串之间,由一个转成另一个所需的最少编辑操作次数.其中,编辑操作包括将一个字符替换成另一个字符,插入一个字符,删除一个字符.字符串 s 和字符串 t 的编辑距离记为 $ED(s,t)$.设字符 $\alpha,\beta\notin\Sigma$,在 s 的开头加上 $q-1$ 个 α,在 s 的末尾加上 $q-1$ 个 β,

得到字符串 s'. 字符串 s' 中所有长度为 q 的子串组成 s 的 q-Gram 集合,简称 Gram 集合,记为 $GS(s)$. 字符串 s 拥有 $|s|+q-1$ 个 q-Gram,如果两个字符串 s 和 t 的编辑距离不大于 k,则它们至少拥有 $\max\{|s|,|t|\}-1-(k-1)\times q$ 个相同的 Gram,这是现有工作中广泛使用的过滤条件之一.

例 1. 表 1 给出了包含 3 个字符串的集合 S. 其中,字符串 s_1 = "Jim Grey" 和 s_2 = "Jim Gray" 的编辑距离 $ED(s_1,s_2)=1$,字符串 s_1 中的第 7 个字符 e 替换成字符 a 就可以得到字符串 s_2.

表 1 字符串集合 S

Id	*Content*	*Id*	*Content*
1	Jim Gray	3	StoneBreaker
2	Jim Grey		

2.2 字符串相似性操作

给定字符串数据集合 $S=\{s_1,s_2,\cdots,s_N\}$,范围查询 $Q(s,\theta)$ 返回结果为 $S'=\{s'|s'\in S,ED(s,s')\leq\theta\}$,即数据集合中与给定字符串 s 的编辑距离不大于阈值 θ 的所有字符串. 在 S 中执行范围查询 $Q(s_1,1)$ 将得到结果 $\{s_1,s_2\}$.

Top-k 查询 $Q(s,k)$ 返回 S 中与给定字符串 s 编辑距离最小的 k 个字符串组成的集合. 在集合 S 中执行 top-2 查询 $Q(s_2,2)$ 的结果为 $\{s_1,s_2\}$.

2.3 字符串相似性连接

给定字符串集合 $S=\{s_1,\cdots,s_N\}$ 和 $R=\{r_1,\cdots,r_M\}$,字符串相似连接操作 Join(R,S,θ) 返回 $J=\{(s,r)|s\in S,r\in R,ED(s,r)\leq\theta\}$. 即所有满足以下条件的字符串对:(1) 字符串对中一个来自 S,另一个来自 R;(2) 二者的编辑距离不大于给定的阈值 θ. 执行表 1 中集合 S 与表 2 中集合 R 的连接操作 Join$(R,S,1)$ 的结果为空集,执行连接操作 Join$(R,S,3)$ 的结果为 {(J. Gray, Jim Gray)}.

表 2 字符串集合 R

Id	*Content*
1	J. Gray
2	J. Jones

3 RM 树索引

本节介绍 RM 树索引结构及其构建方法. 3.1 节介绍 RM 树的索引结构,3.2 节给出 RM 树的构建方法. 将字符串 s 插入 RM 树前,需要计算 Gram 向量、MBR 及 Bitmap,它们的定义如下.

定义 1(字符串 s 的 d 维 Gram 向量). 给定正整数 q 和从字符串集合到正整数集合 $[1,d]$ 的 Hash 函数 h_V,字符串 s 的 q-Gram 集合记为 $GS(s)$. 字符串 s 的 Gram 向量 $\boldsymbol{gv}(s)$ 是一个 d 维向量,该向量的第 i 个分量 $\boldsymbol{gv}(s)[i]$ 的值为 $|\{g\in GS(s),h_V(g)=i\}|$.

定义 2(字符串集合 S 的 MBR). 字符串集合 S 中所有字符串的 d 维 Gram 向量组成的 d 维区间称为 S 的 MBR(Minimal Bounding Region).

定义 3(字符串 s 的 Bitmap). 给定从字符串集合到整数集合 $[1,L]$ 的 Hash 函数 h_B、字符集 $\Sigma'\subseteq\Sigma$ 和字符 c,字符串 s 的 Bitmap 为一串 0-1 序列,记为 $B(s)$,其中第 i 位为 1 当且仅当以下两个条件中至少有一个成立,(1) $\exists g\in GS(s)\cap(\Sigma')^*$ 而且 $h_B(g)=i$;(2) $\exists g\in GS(s),g\notin(\Sigma')^*$,将 g 中不属于 Σ' 的字符替换成 c 得到 g',$h_B(g')=i$. 将 g 中不属于 Σ' 的字符替换成 c 的转换记为 $T(g)=g'$. RM 树中,Σ' 为字母集合,不包含标点和特殊符号,这样可以大幅降低 Bitmap 中 1 的个数,增加 Bitmap 过滤的作用. L 为 Bitmap 的长度.

定义 4(字符串集合 S 的 Bitmap). 字符串集合 S 的 Bitmap 记为 $B(S)$,定义为其中所有字符串的 Bitmap 的或("|")操作结果. 即 $B(S)$ 的第 i 位为 1 当且仅当 S 中存在 s,满足 $B(s)$ 的第 i 位为 1.

3.1 RM 树索引结构

RM 树具备高度平衡性质,即所有叶子节点都在同一层上. RM 树节点与磁盘页一一对应,并被存储在磁盘上. RM 树节点分为内节点和叶节点两类. RM 树通过字符串的 Gram 向量来组织数据存储,将相似的字符串存储在相同或相近的叶节点中. 在查询处理过程中,RM 树使用 Gram 向量、MBR 和 Bitmap 作为过滤信息,削减搜索空间,提高查询处理效率.

RM 树内节点的存储格式为 (*PageId*, *MBR*, *Bitmap*, *LengthRange*, *ChildrenSet*),其中 *PageId* 是内节点对应磁盘页的编号,*MBR* 是以该节点为根的子树中存储的字符串集合的 MBR. *LengthRange* 为该子树中字符串长度的范围. *Bitmap* 为该子树中存储的字符串集合的 Bitmap. 内节点的 *ChildrenSet* 包括若干儿子记录,儿子记录格式为 (*PageId*, *MBR*, *Bitmap*, *LengthRange*). 其中,*PageId* 为儿子节点对应的磁盘页编号,*MBR*, *Bitmap*, *LengthRange* 分别为以该儿子为根的子树中包含的字符串集合的 MBR、字符串所有 Bitmap 以及长度范围. 图 1 所示为 RM 树内节点的结构.

RM 树叶子节点的存储格式为 (*PageId*, *MBR*, *Bitmap*, *LengthRange*, *DataSet*). 其中,*PageId* 是

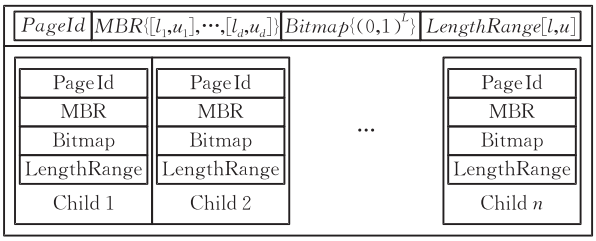


图 1　内节点结构

叶子节点对应的磁盘页的编号，*MBR* 是该节点包含的字符串集合的 MBR，*LengthRange* 为该节点内字符串的长度范围，*Bitmap* 为该叶子节点中包含的字符串集合的 Bitmap，*DataSet* 是该叶子节点包含的字符串数据集合.

在 RM 树中内节点 N_I 的 *MBR*、*Bitmap*、*LengthRange* 通过该节点的所有儿子的 *MBR*、*Bitmap*、*LengthRange* 计算得到. 其中，

$$N_I.MBR = \bigcup_{n \in N_I.ChildrenSet} n.MBR.$$

$N_I.LengthRange$ 的下界为 N_I 所有儿子的 *LengthRange* 的下界的最小值，其上界为 N_I 所有儿子上界的最大值.

$N_I.Bitmap[i]=1$ 当且仅当存在 N_I 的儿子 N，使得 $N.Bitmap[i]=1$.

在 RM 树中叶子节点 N_L 的 *MBR*、*LengthRange* 和 *Bitmap* 通过 N_L 中包含的字符串计算获得. 其中，

$$N_L.MBR = \bigcup_{s \in N_L.DataSet} \boldsymbol{gv}(s),$$

$$N_L LengthRange = [\min_{s \in N_L.DataSet} \{|s|\}, \max_{s \in N_L.DataSet} \{|s|\}].$$

$N_I.Bitmap[i]=1$ 当且仅当存在 $s \in N_L.DataSet$ 使得 $B(s)[i]=1$.

在 RM 树节点中存储字符串集合的 MBR 和 Bitmap 信息，有助于查询处理过程中的搜索空间削减. 增加 MBR 的维度和 Bitmap 的长度可以提高其过滤能力，但是也降低了 RM 树节点的度，增加查询处理的 I/O 代价，因此取值需要折中选取，本文实验部分测试了 MBR 维度和 Bitmap 长度对 RM 树性能的影响. 定理 1 和定理 2 描述了 MBR 和 Bitmap 削减搜索空间的作用. 在介绍定理 1 和定理 2 之前，首先介绍以下定义.

定义 5（字符串 s 与字符串 t 的 Gram 向量距离）. 字符串 s 和字符串 t 的 d 维 Gram 向量距离定义为 $D^G(s,t)=\sum_{i=1}^{d} |\boldsymbol{gv}(s)[i]-\boldsymbol{gv}(t)[i]|$. 即 s 和 t 的 d 维 Gram 向量各个位之差的和.

定义 6（字符串 s 与字符串集合 S 的 Gram 向量距离）. 字符串 s 的 Gram 向量记为 $\boldsymbol{gv}(s)$，字符串集合 S 中的所有字符串的 Gram 向量确定了 S 的 MBR. 字符串 s 与字符串集合 S 的 Gram 向量距离定义为，$D^G(s,S)=\sum_{\boldsymbol{gv}(s)[i]>S.MBR^{UP}[i]} (\boldsymbol{gv}(s)[i]-S.MBR^{UP}[i])$. 即通过 MBR 可以判断不属于 $GS(S)$，但属于 $GS(s)$ 的 Gram 个数.

定义 7（字符串 s 与字符串集合 S 的 Bitmap 距离）. 字符串 s 的 Gram 集合记为 $GS(s)$，字符串集合 S 的 Bitmap 记为 $B(S)$，s 与 S 的 Bitmap 距离 $D^B(s,S)=|\{g|g \in GS(s), g'=T(g), B(S)[h_B(g')]=0\}|$，即通过 Bitmap 可以确定 s 中不存在的 $GS(S)$ 中的 Gram 的个数.

下面介绍定理 1 和定理 2，其中定理 1 说明了若两个字符串的编辑距离小于给定的阈值，则它们的 Gram 向量距离也小于某一阈值. 由此，Gram 向量距离可以用来在 RM 树中，判断某一子树中是否包含与目标字符串编辑距离小于某阈值的字符串. 定理 2 说明了如果一个字符串 s 和一个字符串集合 S 的 Bitmap 距离大于给定阈值，则 S 中的所有字符串与 s 的编辑距离不小于某一阈值. 由此，Bitmap 距离也可以用来判断子树中是否包含与 s 编辑距离不超过给定阈值的字符串.

定理 1.　如果字符串 s_1 和字符串 s_2 的编辑距离不大于 θ，则 s_1 和 s_2 的 Gram 向量距离不大于 $2\theta \times q$，其中 q 为 Gram 长度.

证明. 由于字符串 s_1 和字符串 s_2 的编辑距离不大于 θ，可知 s_1 的 q-Gram 集合与 s_2 的 q-Gram 集合的差中至多有 $\theta \times q$ 个元素，即 $|GS(s_1)/GS(s_2)| \leq \theta \times q$. 对称地，不等式 $|GS(s_2)/GS(s_1)| \leq \theta \times q$ 也成立.

由 Gram 向量距离的定义可知，在计算字符串 s 的 d 维 Gram 向量时，不同的 Gram 可能被散列到同一个维度进行计数. 记 $GS^i(s)$ 为字符串 s 的被映射到第 i 个维度的 Gram 组成的集合. 字符串 s_1 和字符串 s_2 的 Gram 向量在第 i 个维度上的差为 $|\boldsymbol{gv}(s_1)[i]-\boldsymbol{gv}(s_2)[i]|$，它的数值不大于 $|GS^i(s_1)/GS^i(s_2)|+|GS^i(s_2)/GS^i(s_1)|$. 对这两个数值在维度从 1 到 d 上取和，可以得出两个字符串的 Gram 距离向量满足如下不等式：

$$D^G(s_1,s_2) \leq |GS(s_1)/GS(s_2)| + |GS(s_2)/GS(s_1)|.$$

由此得证 $D^G(s_1,s_2) \leq 2\theta \times q$.　　证毕.

定理 2.　如果字符串 s 和字符串集合 S 的 Bitmap 距离大于 $\theta \times q$，则不存在 $t \in S$ 使得字符串 s 和 t 的编辑距离不大于 θ.

证明. 由字符串 s 和字符串集合 S 的 Bitmap 距离大于 $\theta\times q$,可知字符串 s 的 Gram 集合 $GS(s)$ 中至少有 $\theta\times q$ 个 Gram 组成集合 G,满足 $\forall g\in G$,必不存在 $t\in S$ 使得 $g\in GS(t)$. 因此,对于 $\forall t\in S$,字符串 s 中包含多于 $\theta\times q$ 个字符串 t 中不存在的 Gram,因此 s 与 t 的编辑距离不小于 θ. 即 S 中不存在与 s 编辑距离不大于 θ 的字符串. 证毕.

算法 1. ChooseChild.

输入:字符串 s,RM 树内节点 N

输出:作为插入目标的儿子编号 j

1. for $i=1$ to $|N.ChildrenSet|$
2. i's $ENLARGE(i)=(MOEnlarge(s),BEnlarge(s))$
3. j is the child with least enlarge value
4. return j

3.2 RM 树插入方法

RM 树的插入过程从根节点开始,从上而下选择子树插入,直到最终选择一个叶子节点作为插入目标. 随后,字符串 s 被插入到叶子节点中,叶子节点判断是否需要处理溢出,并更新叶子节点的 MBR,Bitmap 等信息. 如果作为插入目标的节点产生分裂或者形状变化,则分裂和形状变化作为插入结果,返回给父亲节点,并在父亲节点中进一步处理,如果父亲节点也由此产生分裂,则该父亲节点也向它的父亲节点返回插入结果. 根节点的分裂将产生新的根节点,并增加树的高度,这与 B 树等的插入过程相似. RM 树的插入过程与其它索引树的不同在于插入目标的选择和节点分裂的过程. 算法 1 的 ChooseChild 和算法 2 的 SplitIndex 分别描述了插入目标选择和内节点的分裂过程,算法 4 的 SplitLeaf 描述了 RM 树叶子节点的分裂过程.

算法 1 用于在节点 N 中选择字符串 s 插入的子树. 对于 N 的每个儿子节点 i,ChooseChild 计算在 i 中插入字符串 s 产生的 $ENLARGE$ 数值(第 1～2 行),并选择该数值最小的节点来完成 s 的插入(第 3～4 行). 儿子节点 i 的 $ENLARGE$ 数值由两部分组成,分别为 $MOEnlarge$ 和 $BEnlarge$. 其中,$MOEnlarge$ 描述在 i 中插入 s 后 MBR 与其它儿子节点的 MBR 相交部分的增量,而 $BEnlarge$ 描述了在 i 中插入 s 后,节点 i 的 Bitmap 中从 0 变成 1 的位的数量. 使用 MBR 的相交增量作为首要排序元素,减小了插入目标节点的 MBR 与其它儿子节点的 MBR 的相交之和,以此减少查询过程中多个儿子被访问的可能性,从而达到减低 I/O 操作的目的. 由于 Bitmap 在 RM 树中用作 MBR 之后的过滤条件,因此次要排序元素使用 Bitmap,其中 Bitmap 从 0～1 的位的数量描述了 Bitmap 过滤能力的损失大小. 为了提高查询处理的效率,$ENLARGE$ 二元组越小越好.

算法 2. SplitIndex.

输入:RM 树内节点 N

输出:索引项集合 $p1,p2$

1. $dim=$chooseSplitDimension(N)
2. sort $N.ChildrenSet$ on $\langle MBR[dim].low, MBR[dim].up\rangle$
3. for $i=m$ to $M-m$
4. partition $N.entryList$ into $p1$and $p2$
5. $cost[i]=\langle MbrOverlap(p1,p2),BSum(p1)+BSum(p2)\rangle$
6. choose partition strategy j with minimal cost
7. partition $N.entryList$ into $p1$ and $p2$
8. return $(p1,p2)$

RM 树节点的 MBR 为多维区域,在节点分裂过程中,将多维区域划分到两个集合的方法,在 R 树及其诸多改进工作中被广泛研究. 与 R 树不同,RM 树节点中包含字符串集合的 MBR 和 Bitmap 两类信息,而且这两类信息不能组成一个多维区域. 算法 2 描述了 RM 树内节点分裂的过程. 与 R* 树节点分裂的过程类似,算法 SplitIndex 首先调用 chooseSplitDimension 方法(第 1 行)确定划分维度,该算法与 R* 树中选择划分维度的算法相似(算法 3 描述了选择划分维度的过程,与 R* 树的不同,每种划分的代价的值为一个二元组,两个维度分别为分裂后产生的两个 MBR 的相交体积和两个 Bitmap 中 1 的个数之和(第 5 行)). 随后,算法 2 在选中的维度上把所有索引项排序(第 2 行),并计算所有可行划分的代价,从中选择代价最小的划分(第 3～6 行),其中 m 和 M 为节点中索引项的最小、最大数目. 最终,算法 2 按照代价最小的划分策略将所有索引项划分成两组(第 7 行). 在节点分裂算法中,RM 树在选择划分维度,以及生成可行划分策略时使用的排序使用 MBR,由于相似的字符串集合的 MBR 也会相似,通过 MBR 划分节点中的索引项可以将相似的字符串组织到相近的子树中. 不同的 Gram 可能被哈希到同一个 Gram 向量的维度,而被视作相同的 Gram. 在查询处理过程中,Bitmap 中包含的更精确的 Gram 信息可以用作进一步过滤来排除不相干的 Gram.

算法 3. chooseSplitDimension.

输入:RM 树内节点 N

输出:划分维度 dim

1. for $i=1$ to d
2. sort $N.entryList$ on$\langle MBR[d].low,MBR[d].up\rangle$
3. for $j=m$ to $M-m$
4. partition $N.entryList$ into $p1,p2$

5.　cost=(*MOverlap*(*p*1,*p*2),*BSum*(*p*1)+*BSum*(*p*2))
6.　choose j with minimal cost
7.　$cost[i]=subcost[j]$
8. choose dim with minimal cost
9. return dim

算法 4 描述了叶子节点的分裂过程，与内节点分裂方法类似，RM 树的叶子节点分裂时同样将字符串排序，然后计算可能的划分策略的代价，最后选择代价最小的划分策略进行分裂操作.

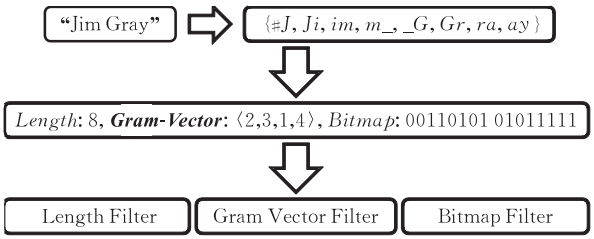


图 2　RM 树过滤功能框架

算法 4.　splitLeaf.

输入：RM 树内节点 N
输出：字符串集合 $p1$ 和 $p2$
1. dim=dimension with largest length
2. sort strings on the dim-th value of Gram vector
3. for $i=m$ to $M-m$
4.　partition strings into $p1$ and $p2$
5.　$cost[i]=\langle Area(p1)+Area(p2), BSum(p1)+BSum(p2)\rangle$
6. choose partition strategy j with minimal cost
7. partition strings into $p1$ and $p2$

RM 树叶子节点的分裂过程与内节点分裂过程的不同之处在于：(1) 在选择划分维度的时候，叶子节点使用长度最长的维度进行划分，这是因为每个字符串都是 Gram 向量空间中的点，任何维度上的划分都能将叶子节点的 MBR 分裂成两个不相交的划分. 选择长度最长的维度划分，可以得到各个维度长度更为平衡的叶子节点. (2) 每种划分方法的代价是一个二元组，由新生成的两个叶子的 MBR 体积之和以及 Bitmap 中 1 的数量之和组成. 在划分过程中选择产生体积小、Bitmap 中 1 的数量小的策略，有助于提高查询处理过程中的剪枝能力，减少查询处理的 I/O 代价.

4　RM 树中的查询处理算法

本节介绍 RM 树中的查询处理算法. 4.1 节给出字符串相似性查询的处理算法，4.2 节介绍字符串相似性连接的处理算法. 在介绍具体查询处理算法前，本节首先给出 RM 树使用的节点过滤方法.

图 2 描述了 RM 树在处理字符串相似性操作时，节点过滤的功能框架. 目标字符串 s 首先被转换成 Gram 集合 $GS(s)$，随后生成 s 的 Gram 向量 $\boldsymbol{gv}(s)$ 和 Bitmap $B(s)$. 对于给定的编辑距离阈值 θ，RM 树通过 3 个过滤器来判断以给定节点 N 为根的子树 T_N 中是否可能包含字符串 t，使得 $ED(s,t)\leq\theta$. 3 种过滤器及其使用方法如下：(1) 长度过滤器 (Length Filter)，记 s 的长度为 Ls，T_N 中最短的字符串长度为 $L_{T_N}^{\mathrm{LOW}}$，T_N 中最长的字符串的长度为 $L_{T_N}^{UP}$，如果 $Ls<L_{T_N}^{\mathrm{LOW}}-\theta$ 或者 $Ls>L_{T_N}^{\mathrm{LOW}}+\theta$，则 T_N 中所有字符串与 s 的编辑距离都大于 θ. (2) Gram 向量过滤器 (Gram Vector Filter)，由定理 1 可知，如果字符串 s 与 N 的 Gram 距离大于 $q\times\theta$，则 T_N 中所有字符串与 s 的编辑距离都大于 θ. (3) Bitmap 过滤器 (Bitmap Filter)，由定理 2，如果字符串 s 与 N 的 Bitmap 距离大于 $q\times\theta$，则 T_N 中所有字符串与 s 的编辑距离都大于 θ. 通过上述 3 种过滤器，RM 树可以判断给定子树中是否可能存在与目标字符串的编辑距离不大于 θ 的字符串.

算法 5.　filter.

输入：字符串 s，RM 树节点 N，编辑距离阈值 θ
输出：是否通过过滤器
1. if ($Ls<L_{T_N}^{\mathrm{LOW}}-\theta$ or $Ls>L_{T_N}^{\mathrm{UP}}+\theta$)
2.　return false
3. if ($D^G(s,N)>\theta\times q$)
4.　return false
5. if($D^B(s,N)>\theta\times q$)
6.　return false
7. return true

将 Gram 向量过滤器与节点 N 的 Bitmap 结合使用，取得优于单独使用 Gram 向量过滤器时的过滤效果. 首先计算 s 与节点 N 的 Gram 向量距离，记为 $Dist_1$. 随后，对于目标字符串 s 的 Gram 集合 $GS(s)$ 中的每个元素 g，如果 Bitmap 中对应的数值为 0，而且 s 的 Gram 向量 $\boldsymbol{gv}(s)$ 对应的位的数值不大于 N 的 MBR 中对应维度的上界，则 $Dist_1$ 的数值加 1. 最后，如果 $Dist_1$ 的数值大于 $q\times\theta$，则 T_N 中不包含与 s 的编辑距离不大于 θ 的字符串. 算法 5 用于判断 T_N 中是否可能存在字符串 t，使得 $ED(s,t)\leq\theta$，不可能存在时返回 false. 其中 $D^G(s,N)$ 为使用结合 N 的 Bitmap 的 Gram 向量过滤器计算而得的数值.

RM 树的过滤功能不会随字符串长度的增加而减弱. 首先，RM 树使用长度过滤器，在长字符串集合中，字符串长度差异较大，因此长度过滤器效果会

更好.其次,RM 树使用的 Gram 过滤器在长字符串组成的数据集合中也会取得更好的效果,这是因为不同的 Gram 向量个数增加了.

4.1 字符串相似性查询处理算法

RM 树支持字符串相似性范围查询和 top-k 查询,本小节首先介绍 RM 树处理字符串相似性范围查询 $Q(s,\theta)$ 的算法,然后介绍处理 top-k 查询 $Q(s,k)$ 的算法.RM 树中的查询处理算法使用先过滤后验证模式,算法首先找到查询结果的候选字符串,然后通过计算编辑距离来验证该候选字符串是否属于查询结果集合.

算法 6. RangeQueryProcessing.

输入:RM 树内节点 N,字符串 s,阈值 θ

输出:字符串集合 $Result$

```
1. Result=∅
2. if(N is an internal node)
3.    for i in N.childrenSet
4.       if(filter(s,i,θ) )
5.          Result=Result∪RangeQueryProcessing(i,s,θ)
6. else
7.    for i in N.dataSet
8.       if(i passes filter)
9.          if(ED(i,s)≤θ)
10.            Result=Result∪{i}
11. return Result
```

4.1.1 范围查询处理算法

RM 树的范围查询处理从根节点开始,逐层向下选择可能包含查询结果的子树,最终查找到包含查询结果的叶子节点,并验证这些叶子节点中的字符串是否属于查询结果集合.算法 6 描述了 RM 树节点 N 处理范围查询的过程,算法 6 调用算法 5 filter 来排除不可能包含查询结果的子树.

如果节点 N 是一个内节点(第 2 行),算法 6 调用 filter 方法考察 N 的每个儿子(第 4 行),排除不可能含有查询结果的儿子节点.对于可能含有查询结果的儿子节点,算法 6 递归调用自身,并合并结果(第 4～5 行).如果 N 是一个叶子节点,则算法 6 逐个检查 N 中包含的字符串是否与目标字符串拥有足够多的公共 Gram(第 8 行),如果是则计算它们之间的编辑距离,并返回编辑距离不大于阈值 θ 的字符串(第 9～10 行).根据之前介绍的 RM 树使用的过滤器的功能可知,在根节点调用算法 6 能找到 RM 树中与目标字符串 s 的编辑距离不大于阈值 θ 的所有字符串.

4.1.2 Top-k 查询处理算法

RM 树处理字符串 top-k 查询时采用分支界限策略,并维护长度为 k 的结果缓存.假设当前结果缓存 $ResultBuffer$ 中保存了当前搜索到的与字符串 s 的编辑距离最小的 k 个结果,top-k 查询处理算法确定下一步需要搜索的子树,并在其中搜索与 s 的编辑距离小于 $ED(s,ResultBuffer[k])$ 的字符串,若找到若干满足条件的结果,查询处理算法更新 $ResultBuffer$,并开始下一轮确定子树及搜索过程,直到未被搜索过的子树中都不可能包含与 s 的编辑距离小于 $ED(s,ResultBuffer[k])$ 为止.算法 7 的 Top-KQueryProcessing 描述了 RM 树处理 top-k 查询 $Q(s,k)$ 的过程,RM 树调用在根节点调用算法 7 来处理 top-k 查询 $Q(s,k)$,其中 $root$ 是 RM 树的根节点.

算法 7. Top-KQueryProcessing.

输入:RM 树内节点 N,字符串 s,整数 k

输出:字符串集合 R

```
1. R=∅
2. insert n' children into min-heap H
3. while (H≠∅)
4.    pop minimal n from H
5.       if (n is an internal node)
6.          for i=1 to number of N's children
7.             i's key=max{D^G(s,n)/θ,D^B(s,n)/θ}
8.                if(|R|<k or key<ED(s,R[k]))
9.                   insert i into H according to key
10.      else
11.         refine R,H with strings in n
12. return R
```

算法 7 中,1～2 行初始化结果缓存 R 和最小堆 H.其中 R 用于保存当前与查询目标字符串 s 编辑距离最小的 k 个结果;H 用于保存处理查询时需要访问的 RM 树节点.H 不为空时算法逐轮运行(3～4 行),返回结果.如果当前处理节点为内节点,5～9 行计算每个儿子包含的字符串与 s 的最小编辑距离,并以此为 key,将需要访问的儿子节点插入最小堆 H 中.如果当前处理节点是叶子节点,算法(10～11 行)通过叶子节点中的字符串完善当前 top-k 结果,并去除 H 中无需访问的节点.最后,在 H 中没有节点的情况下,算法返回 R,此时,R 包含了以 N 为根的子树中的 top-k 查询处理结果.

4.2 字符串相似性连接算法

假设两个字符串集合 R,S 被存储在两个 RM 树中,记为 T^R 和 T^S.RM 树字符串相似连接操作从 T^R 和 T^S 的根节点开始,递归地在可能产生连接结果的子树之间进行连接操作.具体执行过程由算法 8 的 Join 所描述,R 和 S 之间的字符串相似连接需要

调用 Join($T^R.root$, $T^S.root$, θ).

算法 8. Join.

输入：RM 树内节点 R、S，编辑距离阈值 θ

输出：字符串对集合 *Result*

```
1. Result=∅
2. if (R,S are both internal nodes)
3.    for i in R.childrenSet, j in S.childrenSet
4.       if (D^G(i,j)≤θ×q and D^B≤θ×q )
5.          Result=Result∪Join(i,j,θ)
6. else if (R is internal node)
7.    for i in R.childrenSet
8.       if (D^G(i,S)≤θ×q and D^B(i,S)≤θ×q )
9.          Result=Result∪Join(i,S,θ)
10. else if (S is internal node)
11.   for j in S.childrenSet
12.      if ( D^G(R,j)≤θ×q and D^B(R,j)≤θ×q)
13.         Result=Result∪Join(i,S,θ)
14. else
15.   for i in R.dataSet, j in S.dataSet
16.      if ((i,j) passes filter)
17.         if (ED(i,j)≤θ)
18.            Result=Result∪{(i,j)}
19. return Result
```

算法 8 描述了以节点 R 和 S 为根的子树之间进行字符串相似连接的过程，针对 R 和 S 的不同节点类型进行处理. 第 1 行初始化结果集合 *Result* 为空集. 当 R 和 S 都是内节点的时候(2～5 行)，所有 R 的儿子和 S 的儿子组成的子树对(i,j)都被检查，如果可能产生连接结果，即子树 i 和子树 j 中存在字符串对(s,r)，满足 $r\in R, s\in S$ 且 $ED(s,r)\leq\theta$，则算法递归调用 Join(i,j,θ)，并将递归调用的结果加入 *Result*.

如果 R 和 S 中有一个为内节点，另一个为叶子节点(6～13 行)，则检查内节点的每个儿子和叶节点组成的子树对，如果可能产生连接结果，则在该子树对中递归调用 Join 算法并将结果加入连接结果集合.

如果 R 和 S 都是叶子节点(14～18 行)，则检查 R 中的字符串和 S 中的字符串组成的每一个字符串对(r,s)，如果 r 和 s 的编辑距离可能小于 θ，则计算 r 和 s 的编辑距离，如果 $ED(s,r)\leq\theta$，字符串对(r,s)被加入结果集合 *Result*. 算法 8 在检查两个子树 R 和 S 是否能产生连接结果时，计算 R 和 S 的 Gram 向量距离和 Bitmap 距离(第 4 行，第 8 行，第 12 行)，并根据定理 1 和定理 2 中编辑距离和 Gram 向量距离、Bitmap 距离的关系进行剪枝，排除不可能生成连接结果的子树对.

5　RM 树应用扩展

本节介绍使用其它字符串相似性度量时，RM 树支持各种相似性操作的方法，包括 Dice Similarity，Cosine Similarity，Jarccard Similarity 和 Normalized Edit Distance. 给定字符串 s_1 和 s_2 这些字符串相似性度量如下所述：

$$DICE(s_1,s_2)=\frac{2\times|GS(s_1)\cap GS(s_2)|}{|GS(s_1)|+|GS(s_2)|},$$

$$COSINE(s_1,s_2)=\frac{|GS(s_1)\cap GS(s_2)|}{\sqrt{|GS(s_1)|\times|GS(s_2)|}},$$

$$JAD(s_1,s_2)=\frac{|GS(s_1)\cap GS(s_2)|}{|GS(s_1)\cup GS(s_2)|},$$

$$NED(s_1,s_2)=\frac{ED(s_1,s_2)}{\max\{|s_1|,|s_2|\}}.$$

对于上述相似性度量，RM 树的使用在查询处理过程中存在区别，而插入过程没有区别，只需修改 RM 树中字符串和索引节点的相似性界限估计方法即可. 对于 Dice Similarity，使用目标字符串与索引节点 MBR 和 Bitmap 估计分子的上界，而后用 s_1 和 s_2 的长度估计分母的下界，最终计算得到 Dice Similarity 的上界，以此作为剪枝的依据. 其它相似性度量的估计方法 Dice Similarity 类似，RM 树中节点的 MBR、Bitmap 和字符串集合长度范围使得 RM 树能够支持上述相似性度量.

6　性能评价和分析

6.1　实验设置

本节通过实验测试 RM 树的性能，我们用 Java 语言实现了 RM 树索引结构以及查询处理算法，JDK 版本为 Java(TM) SE Runtime Environment (build 1.6.0_22-b04). 实验平台为一台 PC 机，配置为双核 CPU，主频 2.93 GHz，2 GB 内存，320 GB 硬盘. 实验使用的真实数据从 DBLP 数据集合中提取，分别为 Title 数据集和 Author 数据集. 其中，Title 数据集中包含 1644932 条字符串记录，Author 数据集包含 967810 条字符串.

对 RM 树的性能测试内容包括查询处理性能和 RM 树维度、Bitmap 长度的关系，RM 树处理范围查询、top-k 查询和连接操作的性能以及建立 RM 树的时间开销. 实验部分比较了 RM 树和 B^{ed} 树的性能，其中 B^{ed} 树使用 Gram Counting Order，这是由于 Gram Counting Order 在文献[13]的众多测试中都能获得较好的性能，因此本文将其作为比较对

象.测试过程中使用的参数如表3所示,其中粗体的数值为参数默认值,测试中Gram的长度为2.

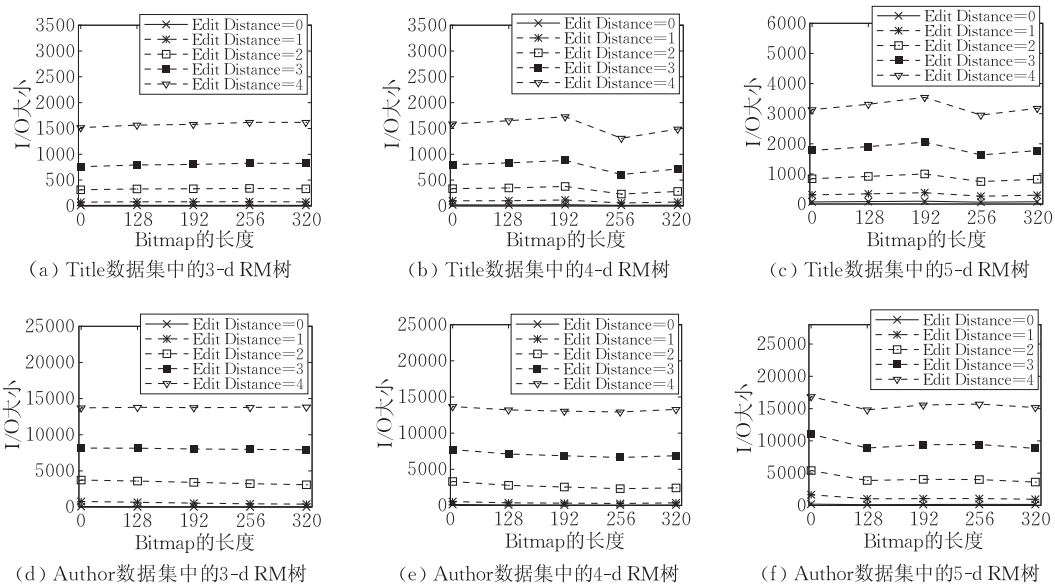
表3 实验参数

参数	取值
Distance Threshold θ	0,1,**2**,3,4
Top-k Parameter k	1,2,**4**,8
Page Size/KB	**4**
Cache Size/MB	4,**16**,64,256
Length of Bitmap	0,128,192,**256**,320
Dimension of MBR	3,**4**,5

6.2 维度和Bitmap长度对剪枝能力的影响

本部分测试RM树的性能与RM树的MBR维度以及Bitmap的长度之间的关系,通过范围查询的性能评价不同MBR维度和Bitmap长度组合.实验在Title和Author数据集合中测试了维度为3、4、5,Bitmap长度为0、128、192、256和320时,测试了RM树处理范围查询的I/O代价,本部分实验缓存大小设置为0.

图3(a),(b)和(c)为Title数据集合中,RM树的维度分别为3,4和5时,范围查询处理的I/O代价.I/O代价的大小反映了RM树在处理范围查询时的剪枝能力,I/O代价越小,剪枝能力越强.如图3(a)所示,3维的RM树在Bitmap长度增加时,剪枝能力没有提高,反而略有下降.这是因为RM树的维度过低,导致过多的字符串具有相同的Gram向量,因此在同一个叶子节点中,存在不同Gram的个数较大,这使得Bitmap的过滤能力大幅下降甚至失效.因此,3维的RM树中,加入Bitmap导致节点中包含条目数量的降低,从而降低了索引的效率.从图3(b)中可见,在4维的RM树中增加Bitmap的长度,剪枝能力先降低后增长,并在Bitmap长度为256时取得最优的剪枝能力.这是由于4维的RM树中,叶子节点包含的不同Gram数量大多少于200个,因此长度256的Bitmap可以提供较好的剪枝能力.长度为128、192的Bitmap由于剪枝能力不足,又降低了内节点中索引条目的个数,导致了索引效率的降低.长度320的Bitmap具有更高的剪枝能力,但是过多地降低了内节点的度,因此效率低于长度256的Bitmap,但是由于其它长度的Bitmap.图3(c)中所示内容与(b)相似.



(a) Title数据集中的3-d RM树 (b) Title数据集中的4-d RM树 (c) Title数据集中的5-d RM树

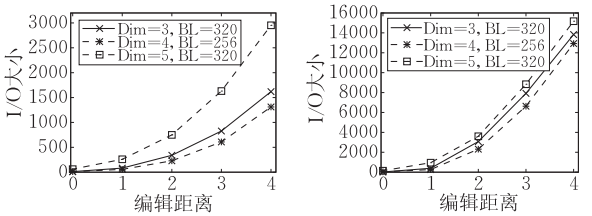
(d) Author数据集中的3-d RM树 (e) Author数据集中的4-d RM树 (f) Author数据集中的5-d RM树

图3 位图长度的影响

图3(d)~(f)给出了Author数据集合中,不同维度的RM树,使用不同长度的Bitmap时的剪枝能力.从图3(d)~(f)中可见,在各个维度的RM树中加入Bitmap都增加了RM树的剪枝能力,这是由于Author数据集合中的字符串由名字组成,RM树的叶子中含有不同的Gram数量较少,故较短的Bitmap也可以增加RM树的剪枝能力.

对于Title和Author两个数据集合,我们选出3维、4维和5维性能最好的RM树进行比较,以此确定后续实验使用的RM树维度和Bitmap长度.如图4(a)所示,在Title数据集合中构建的3维、4维和5维的RM树,取得最好性能的Bitmap长度分别为320、256和256.通过比较3种组合可见,4维的RM树使用长度为256的Bitmap可以获得最好的剪枝效果.类似的,图4(b)中列出了Author中3维、4维和5维的性能最好的RM树,其中4维RM树使用长度为256的Bitmap可以取得最好的剪枝效果.在后续实验中,RM树的维度和Bitmap

长度分别确定为 4 和 256.

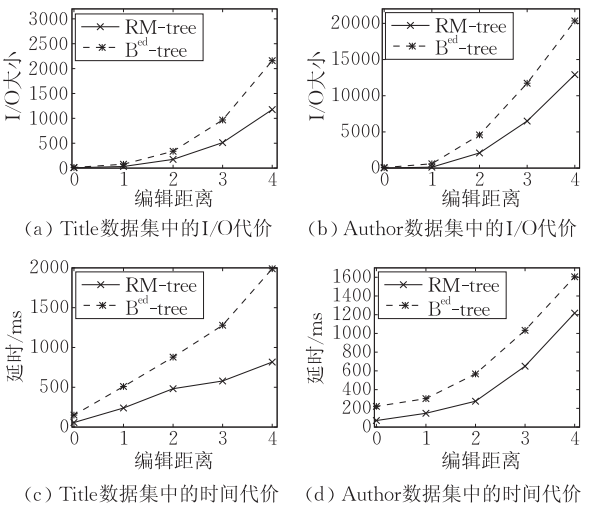


(a) Title数据集中的RM树　(b) Author数据集中的RM树

图 4　RM 树维度的影响

6.3　范围查询性能分析

本部分测试 RM 树处理范围查询的 I/O 代价和时间代价，从 Title 和 Author 中随机抽取 100 个字符串作为查询目标，并计算查询处理的平均 I/O 代价和时间开销. 如图 5 所示，RM 树处理范围查询的 I/O 代价和时间开销随编辑距离阈值的增加而增加. 在 Title 和 Author 中的实验表明，与 B^{ed} 树相比，RM 树具有更好的剪枝能力，减少了索引节点访问次数，有效地降低了 I/O 代价并减少了查询处理时间开销，查询处理延时降低了 25%～59%.



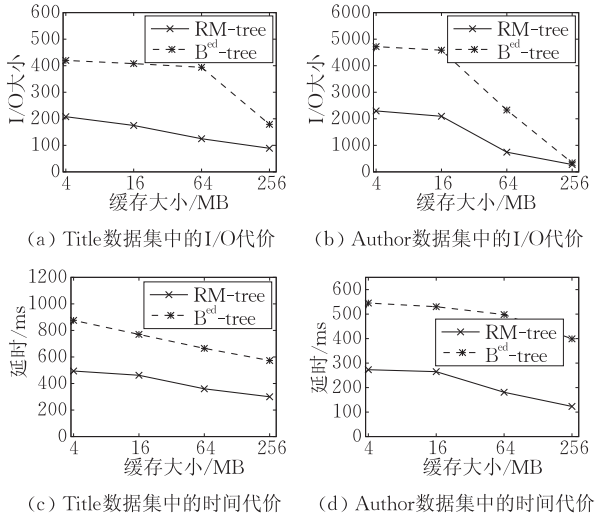
(a) Title数据集中的I/O代价　(b) Author数据集中的I/O代价

(c) Title数据集中的时间代价　(d) Author数据集中的时间代价

图 5　范围查询性能与编辑距离的关系(16 MB buffer)

图 6 中列出了 RM 树处理范围查询的 I/O 代价和时间开销与缓存大小的关系，其中编辑距离阈值为 2. I/O 代价和时间开销随缓存的增加而减少，并在缓存大小达到 256 MB 时显著减小. 这是由于缓存页面减少了磁盘读取次数，因此也降低了时间开销. 从图 6(a)～(d)可见，RM 树在不同缓存大小情况下，性能都优于 B^{ed} 树.

对比 Title 和 Author 对应的 I/O 代价和时间开销可见，时间开销降低幅度小于 I/O 代价. 原因为加大缓存可以降低磁盘读取次数，但是不能减少编辑距离的计算次数，因此时间开销不能获得与 I/O 代价同幅度的降低.



(a) Title数据集中的I/O代价　(b) Author数据集中的I/O代价

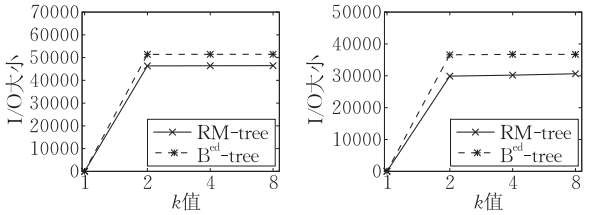
(c) Title数据集中的时间代价　(d) Author数据集中的时间代价

图 6　范围查询性能与缓存大小的关系

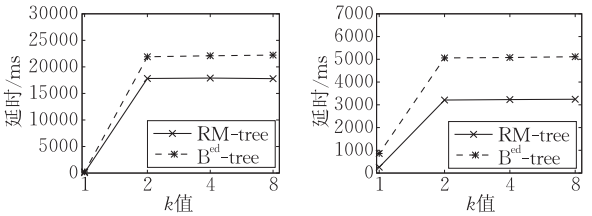
6.4　Top-k 查询性能分析

本部分测试 RM 树处理 top-k 查询的效率，实验从 Title 和 Author 中随机抽取 100 个字符串作为查询目标，并测试参数 k 的值为 1、2、4、8，缓存大小为 16 MB 时，查询处理的 I/O 代价和时间开销. 如图 7 所示，在 Title 中建立的 RM 树在回答 top-k 查询时，$k=1$ 的查询处理代价小于其它情况，而 k 的取值为 2、4、8 时，处理代价基本不变. 这是由于 Title 数据集合中，字符串之间相差较大，选取 top-2、top-4 和 top-8 的字符串集合也会访问很多索引节点. 在 Author 数据集合上的测试结果与 Title 数据集合中的测试结果相似，k 取值从 2～8 时，查询处理代价都保持稳定. 这是因为 Author 数据集合中，字符串长度差距较小，导致与目标字符串具有相似 Gram 向量的字符串大量存在，因此 top-k 查询在 k 较小的时候也会访问大量索引节点以获取足够的字符串进行剪枝. 由于树剪枝能力不及 RM 树，在 k 的取值变化时，B^{ed} 树处理 top-k 查询的效率在 Title 和 Author 两个数据集合上都不及 RM 树. 在真实的数据集合中，与 B^{ed} 树相比，RM 树降低了 21%～39%的查询处理延时.

图 8 给出了 RM 树处理 top-k 查询的代价与缓存大小的关系，其中参数 k 的值为 4. RM 树和 B^{ed} 树处理 top-k 查询的 I/O 代价和时间开销都随缓存的增加而减少. Title 数据集合中的 top-k 处理 I/O 代价大于 Author 数据集合中的 I/O 代价，这是由于 Title 中不同字符串相差更大，获取 k 个最相似的字符串需要访问更多的索引节点，造成更大的 I/O 代价. 时间开销的降低幅度不及 I/O 代价的降低幅度，这也是因为增加缓存只能降低 I/O 代价，而无法降低字符串之间编辑距离的计算次数.

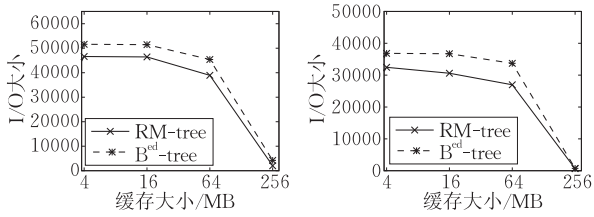


(a) Title数据集中的I/O代价 (b) Author数据集中的I/O代价

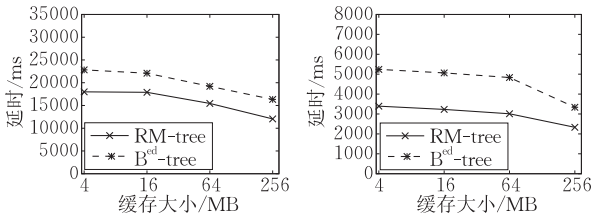


(c) Title数据集中的时间代价 (d) Author数据集中的时间代价

图 7 Top-k 查询处理性能与 k 值大小的关系(16 MB buffer)



(a) Title数据集中的I/O代价 (b) Author 数据集中的I/O代价

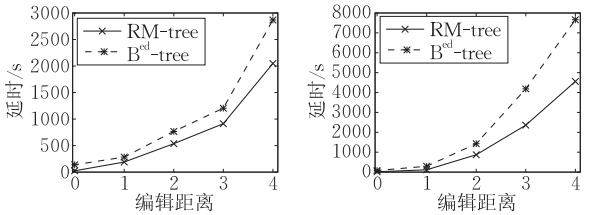


(c) Title数据集中的时间代价 (d) Author数据集中的时间代价

图 8 Top-k 查询处理性能与缓存大小的关系

6.5 连接操作性能分析

本部分测试 RM 树处理字符串相似性连接的效率,缓存大小设为 16 MB. 直接使用 Title 和 Author 进行相似性连接会产生过多的连接结果(与原始数据同量级),为了控制生成结果的大小,本部分实验使用的数据集合为 Title 和 Author 的子集. 实验分别从 Title 和 Author 中抽取的 10^5 条字符串与 5×10^5 条字符串,作为测试数据. 如图 9 所示,在 Title 和 Author 中,随着编辑距离阈值的增加,RM 树和 B^{ed} 树处理字符串相似性连接操作的时间开销也增加. 在 Title 和 Author 中的测试结果显示,RM 树在处理连接操作时,性能优于 B^{ed} 树,RM 树降低了 29%～41%的查询处理延时.

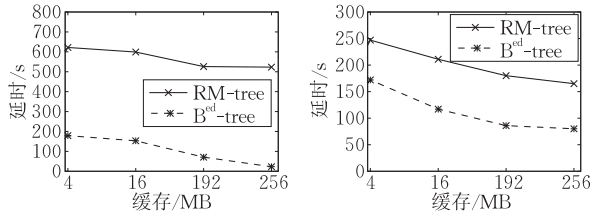


(a) Title数据集中的时间代价 (b) Author数据集中的时间代价

图 9 连接操作性能与编辑距离的关系(16 MB buffer)

6.6 索引构件时间

本部分测试了 RM 树索引构建时间,分别测试了不同缓存大小情况下,使用 Title 和 Author 两个数据集合构建 RM 树的时间开销. 其中,Title 数据集中包含 1644932 条字符串记录,Author 数据集包含 967810 条字符串. 如图 10 所示,RM 树在 Title 和 Author 中的构建时间都长于 B^{ed} 树的构建时间. 通过加大缓存,可以减少 RM 树的构建时间,但是减小的幅度小于 B^{ed} 树的减小幅度,这是因为 RM 树在插入节点选择、节点划分过程中需要大量计算,加大缓存只能减少磁盘读取时间,无法减少计算时间. B^{ed} 树构建过程中计算次数远小于 RM 树,因此可以通过加大缓存的方法有效地减少构建时间开销. 虽然 RM 树的构建时间长于 B^{ed} 树,但是 RM 树在处理查询和连接操作时,效率均优于 B^{ed} 树,因此一次性的构建过程不影响 RM 树的有效性.



(a) Title数据集中的时间代价 (b) Author数据集中的时间代价

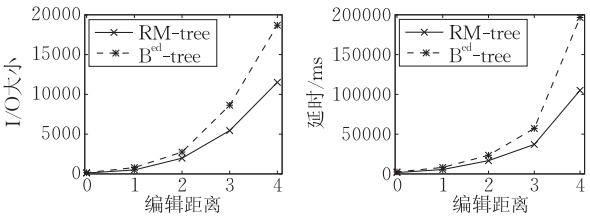
图 10 索引构建性能与缓存大小的关系(16 MB buffer)

6.7 合成数据上的测试结果

本文在人工合成的数据集合中测试了 RM 树的性能. 人工合成数据集合包括 Author 和 Title 中的字符串,以及随机修改其中字符串内容所得的合成数据. 对 Author 和 Title 中的每个字符串,我们随机修改若干字符,生成与原有字符串的编辑距离为 1～5 的合成数据. 合成数据的大小是原有数据大小的 6 倍,在合成数据上建立的索引大小为 3.1 GB,超过了内存的大小. 本部分实验使用的缓存大小为 16 MB.

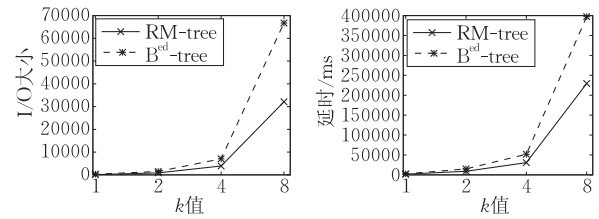
由图 11 可见,RM 树在合成数据集合中处理范围查询的 I/O 开销和时间代价随编辑距离增加而变大. 增加编辑距离阈值导致更多的索引节点被访问,这与真实数据集合上的实验结果一致. 图 12 描述了 RM 树在合成数据集合中处理 top-k 查询的性能. RM 树处理 top-k 查询的 I/O 代价和时间开销随查询参数 k 增长. 当 k 的取值在 1～4 时,查询代

价增长非常缓慢. 当 k 的取值为 8 时，查询代价显著大于 k 值为 1，2 和 4 时的查询代价. 这是由于合成数据集合中，对于每个字符串 s，都存在与之编辑距离为 0～5 的字符串数据. 因此，获取与之最近的 1 个，2 个和 4 个字符串不需要访问大量节点，而只需访问与 s 存储位置相近的叶子节点即可. 当 $k=8$ 时，查询结果中包含与 s 的编辑距离较大的字符串，因此查询处理将访问大量的索引节点，造成较大的查询代价. 如图 11 和图 12 所示，在人工合成数据集合中，RM 树的性能也优于 B^{ed} 树，降低了查询的 I/O 代价，并减少了 35%～50%的时间开销.



(a) 合成数据中的 I/O代价　　(b) 合成数据中的时间代价

图 11　范围查询性能与编辑距离的关系



(a) 合成数据中的 I/O代价　　(b) 合成数据中的时间代价

图 12　Top-k 查询处理性能与 k 值大小的关系

7　相关工作

现有研究工作设计了多种字符串相似性度量指标，并基于这些度量指标进行字符串相似性操作. 本文主要使用编辑距离度量字符串的相似性，因为它不需要对语言理解的直观度量. 文献[1]设计了计算两个字符串 s_1 和 s_2 之间编辑距离的算法，其时间复杂度和空间复杂度都是 $O(|s_1||s_2|)$，$|s_1|$ 和 $|s_2|$ 分别为字符串 s_1 和 s_2 的长度. 其后续研究工作将提高计算编辑距离的效率或者通过近似计算降低时间代价[2-4].

现有基于编辑距离的字符串相似性操作的工作主要使用 q-Gram 和倒排索引结构，将字符串相似性转换为两个 Gram 集合的相似性. 这些工作针对相似性查询和相似性连接操作，使用 filter-verify 框架. 它们首先通过过滤技术构建结果候选集合，然后计算相应的编辑距离，得到最后的结果. 文献[5]针对字符串相似性连接操作，提出了 counting-filter、positional-filter 和 length-filter 技术，用于识别连接操作的候选集. 其它过滤技术包括 prefix-filter[8]，mismatch-filter[9] 等. 文献[6-7]使用堆来合并多个 Gram 对应的倒排链表，从中选取相似性操作结果的候选集合. 文献[10]通过去除部分 Gram 的倒排链表并改变过滤条件来减少空间代价. 文献[11]研究字符串索引的增量式更新方法. 以上方法在处理大量字符串数据时，有以下劣势：(1) 空间代价巨大，Gram 对应的倒排表大小与原始数据大小相当；(2) 更新代价大，对一个字符串的更新引起多个 Gram 链表的更新. (3) 支持查询类型有限，无法直接支持 top-k 等复杂查询. 与上述工作不同，本文设计的 RM 树在磁盘上索引字符串数据，支持多种字符串相似性操作，减少了数据更新代价，并避免了存储 Gram 对应的倒排表所需昂贵的存储代价.

现有处理字符串相似性操作的树形索引包括 MHR 树[12] 和 B^{ed} 树[13]. 其中，MHR 树用于处理谓词中包括空间条件和字符串相似性条件的搜索. MHR 树在 R 树节点中加入字符串集合的 Min-Hash 签名，支持近似地空间字符串相似性搜索，并可以估计查询的选择度大小. 文献[13]提出了 B^{ed} 树，B^{ed} 树将字符串排序后存储在 B^+ 树中，减少了 q-Gram结合倒排索引引起的空间代价和更新代价，并能支持多种相似性操作. 然而，一维的排序无法有效的将字符串按照相似性聚类，不能有效减少查询处理的 I/O 代价. 本文设计 RM 树，通过字符串的 Gram 向量进行聚类，并使用节点中存储的 Bitmap 进一步过滤，以此减少字符串相似性操作的 I/O 代价.

8　结　论

本文提出了一种支持字符串相似性处理的树形索引 RM 树，RM 树通过字符串的 Gram 向量将相似的字符串组织到相近的叶子节点中，并使用位图作为过滤工具，能够有效地支持字符串相似性范围查询、top-k 查询和相似性连接操作. 本文给出了 RM 树的构建方法、范围查询处理、top-k 查询处理和连接处理算法. 在真实数据上的实验结果表明，与 B^{ed} 树相比，RM 树在处理各种字符串范围查询、top-k 查询和连接操作时有效地降低了磁盘 I/O 操作次数，并分别将时间开销降低了 25%～59%、21%～39%和 29%～41%. 在人工合成的数据集合上，与 B^{ed} 树相比，RM 树减少了 35%～50%的查询处理时间开销. 真实数据集合和人工合成数据集合中的实验结果验证了 RM 树的有效性.

参 考 文 献

[1] Wagner R A, Fischer M J. The string-to-string correction problem. Journal of the ACM, 1974, 21(1): 168-173

[2] Cormen T H, Leiserson C E, Rivest R L. Introduction to Algorithms. 2nd Edition. Cambridge, Massachusetts, USA: The MIT Press, 2002

[3] Masek W J, Paterson M. A faster algorithm computing edit distances. Journal of Computer and System Sciences, 1980, 20(1): 18-31

[4] Cornode G, Muthukrishnan S. The string edit distance matching problem with moves. ACM Transactions on Algorithms, 2007, 3(1)

[5] Gravano L, Ipeirotis P G, Jagsdish H V et al. Approximate string join in a database (almost) for free//Proceedings of the 27th International Conference on Very Large Data Bases. Roma, Italy, 2001: 491-500

[6] Li Chen, Lu Jia-Heng, Lu Yi-Ming. Efficient merging and filtering algorithms for approximate string searches//Proceedings of the 24th International Conference on Data Engineering. Cancún, México, 2008: 257-266

[7] Sarawagi S, Kirpal A. Efficient set joins on similarity predicates//Proceedings of the ACM SIGMOD International Conference on Management of Data. Paris, France, 2004: 743-754

[8] Chaudhuri S, Ganti V, Kaushik R. A primitive operator for similarity joins in data cleaning//Proceedings of the 22nd International Conference on Data Engineering. Atlanta, USA, 2006: 5-15

[9] Xiao Chuan, Wang Wei, Lin Xue-Min. Ed-join: An efficient algorithm for similarity joins with edit distance constrains//Proceedings of the 34th International Conference on Very Large Data Bases. Auckland, New Zealand, 2008: 933-944

[10] Behm A, Ji Sheng-Yue, Li Chen, Lu Jia-Heng. Pace-constrained gram-based indexing for efficient approximate string search//Proceedings of the 25th International Conference on Data Engineering. Shanghai, China, 2009: 204-215

[11] Hadijieleftheriou M, Koudas N, Srivastava D. Incremental maintenance of length normalized indexes for approximate string matching//Proceedings of the ACM SIGMOD International Conference on Management of Data. Rhode Island, USA. 2009: 429-440

[12] Yao Bin, Li Feifei, Hadjieleftheriou M, Hou Kou. Approximate string search in spatial databases//Proceedings of the 26th International Conference on Data Engineering. California, USA, 2010: 545-556

[13] Zhang Zhen-Jie, Hadjieleftheriou M, Ooi Beng-Chi, Srivastava D. B^{ed}-tree: An all-purpose index structure for string similarity search based on edit distance//Proceedings of the ACM SIGMOD International Conference on Management of Data. Indianapolis, USA, 2010: 915-926

WANG Jin-Bao, born in 1983, Ph. D. candidate. His research interests include index and query processing over massive data in cloud computing systems.

GAO Hong, born in 1966, Ph. D., professor, Ph. D. supervisor. Her research interests include wireless sensor networks, cyber-physical systems, massive data management and data mining.

LI Jian-Zhong, born in 1950, professor, Ph. D. supervisor. His research interests include wireless sensor networks, cyper-physical systems, database, massive data processing etc.

YANG Dong-Hua, born in 1976, lecturer, Ph. D.. His research interests include massive data processing and data intensive computing.

Background

This work focuses on research of building string similarity index. String similarity processing is a basic operation in various applications such as data cleaning, information integration, spell checking and bioinformatics. Existing research work focus on main memory based methods with q-Gram and inverted lists, and they adopt filter and verify framework. They design filters to reduce the size of candidate set, and then calculate edit distance between the target string and candidate string to get query results. However, such solutions produce disadvantages as below. First, memory consumption is very large while processing massive datasets, and it even takes as large as the size of original dataset. Second, the update cost is really high since a single update of string causes multiple updates of inverted lists. Third, it supports limited types of operations, only range query and joins, and they're unable to process complex queries such as top-k query. Recently proposed B^{ed}-tree index designs string orders and stores strings according to such orders in a B^{+}-tree index. Such solution fails to cluster similar strings together thus incurs large I/O cost. To avoid the large memory consumption and reduce I/O costs while supporting diverse string similarity operations, we propose RM-tree index, which targets at better efficiency of locating data for string similarity processing.

This work is supported in part by the State Key Development Program of Basic Research of China, the Key Program of National Natural Science Foundation of China, the NSF of China and the NSFC-RGC of China. These foundations focus on the research of various areas of data intensive super computing. Our group has been working on the research of database for many years, and many high quality papers have been published in worldwide conferences and transactions, such as SIGMOD, VLDB, ICDE, KDD, INFOCOM, TKDE, VLDB Journal et al.

This paper proposes RM-tree, a tree structured index to support various types of string similarity operations. Compared to existing solutions, RM-tree eliminates shortcomings of q-Gram and inverted lists, and takes much less I/O cost and latency than B^{ed}-tree while processing string similarity operations.