R－树和四叉树的空间索引结构:RQOP_树

刘润涛[1], 郝忠孝[1,2]

(1.哈尔滨理工大学 计算机科学与技术学院,哈尔滨 150080,liurt@hrbust.edu.cn;
2.哈尔滨工业大学 计算机科学与技术学院,哈尔滨 150001)

摘 要: 针对现有的基于R－树和四叉树的空间索引结构中存在的问题,通过建立数据矩形间的序关系对数据空间进行分割,提出了一种新的空间数据索引结构:RQOP树.在此结构中,节点的构造是按照空间数据的分布来进行的而不是像其它基于R－树和四叉树的空间索引结构只是对数据空间进行均匀划分而得到,使树的高度尽可能低,同时使兄弟节点间的交叠相对较小.在区域查询算法中引入了查询窗口包含节点MBR的判断加快了查询的速度.给出了RQOP树的生成、节点插入和区域查询算法,并给出了相应算法的可行性和正确性定理及时间复杂度分析.实验表明:新索引结构的查询速度明显加快.

关键词: 空间数据;索引结构;RQOP树;区域查询
中图分类号: TP311 **文献标志码**: A **文章编号**: 0367－6234(2010)02－0323－05

Spatial index structure based on R-tree and quadtree: RQOP_tree

LIU Run-tao[1], HAO Zhong-xiao[1,2]

(1. College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China, liurt@hrbust.edu.cn; 2. School of Computer Science and Technology, Harbin Institute of and Technology, Harbin 150001, China)

Abstract: A new index structure for spatial data, RQOP_tree, is proposed by setting up the order relation between data rectangles to partition the data spaces, aimed at the existing problems in current index structures based on R-tree and quadtree. In this structure, the middle nodes are constructed according to the distribution of spatial data instead of partitioning the data space evenly, therefore the height of the tree is guaranteed as low as possible and a comparatively small overlap between brother nodes can be kept. In the range query algorithm, the check of query window containing a node's MBR is introduced to speed up the query effectively for a comparatively large query window. The algorithm for constructing the index structure is given, and its time complexity as well as its correctness is presented. The algorithms for node insertion and range query are obtained. The experiment shows that the query speed is increased greatly.

Key words: spatial data; index structure; RQOP_tree; range query

空间数据索引技术是提高空间数据库查询性能的关键技术,直接影响到空间数据库系统的性能.四叉树是一种层次结构,其结构清晰、易于实现,广泛用于很多领域.在空间数据库中,将四叉树与R－树结合得到的空间数据索引结构将用于索引空间数据点的点四叉树、MX－四叉树、PR－四叉树,用于索引空间矩形数据的CIF－四叉树[1]、QR－树[2-3],QR*－树[4]和PMR树[5-6].然而,这些索引结构在生成相应的索引结构时采用的是对数据空间进行四等分的分割或采用超节点策略.因此,当数据分布非均匀时,所产生的相应的空间索引四叉树将会是严重不平衡的,即,树的高度将会变得很高,会严重地影响查询的速度.本文针对这一问题,结合四叉树和R－树,按照数据分布的情况对数据空间进行分割,同时以使中间节点间的交叠尽可能小为目标,建立了一种新的空间数据索引结构:RQOP树.在此结构中,对

收稿日期: 2008－05－29.
基金项目: 国家自然科学基金资助项目(10571037);黑龙江省自然科学基金资助项目(F200601).
作者简介: 刘润涛(1961—),男,博士,教授;
郝忠孝(1940—),男,教授,博士生导师.

数据空间的分割是按照该空间所包含的数据的分布情况进行的,使树的高度为 $\log_4 n$,从而加快了查询的速度.

1 相关定义

采用数据的最小外包矩形[7](MBR)作为空间物体的近似表示.

假设有 n 个空间物体,其 MBR 为:

$\{I_i \mid I_i$ 左上角点为(a_x^i, a_y^i),I_i 右下角点为$(b_x^i, b_y^i)\}_{i=1}^n$.

式中:I_i 为第 i 个物体的最小外包矩形,用其左上角点和右下角点表示.

定义 1 设 I_i、$I_j(i \neq j)$ 是两个不同物体的最小外包矩形,若 $a_x^i > a_x^j$,或当 $a_x^i = a_x^j$ 时,$a_y^i > a_y^j$,或当 $a_x^i = a_x^j$ 且 $a_y^i = a_y^j$ 时,$b_x^i > b_x^j$,或当 $a_x^i = a_x^j$ 且 $a_y^i = a_y^j$,$b_x^i = b_x^j$ 时,$b_y^i < b_y^i$,则称 I_i^y 按 x 坐标大于 I_j,记作 $I_i^x > I_j$.

定义 2 设 I_i、$I_j(i \neq j)$ 是两个不同物体的最小外包矩形,若 $a_y^i > a_y^j$,或当 $a_y^i = a_y^j$ 时,$a_x^i > a_x^j$,或当 $a_y^i = a_y^j$ 且 $a_x^i = a_x^j$ 时,$b_y^i > b_y^j$,或当 $a_y^i = a_y^j$ 且 $a_x^i = a_x^j$,$b_y^i = b_y^j$ 时,$b_x^i < b_x^j$,则称 I_i 按 y 坐标大于 I_j,记作 $I_i^y > I_j$.

给定 n 个空间数据的 MBR,记录于三维数组 I. 排序算法记为 Sorting_MBR(I, n, id). 其中,n 为要排序数据的个数,id 为标识变量,id =0,1 分别为按定义 1 和定义 2 对 I 中数据排序.

2 四叉树和 R - 树的空间索引结构:RQOP 树

2.1 节点构成

在 RQOP 树中,有两种节点:中间节点和叶子节点. 中间节点采用 (I, child(1), child(2), child(3),child(4)) 的形式. 其中,I 为该节点中所包含的所有数据矩形的 MBR,而 child(1) 为指向该节点的右下角孩子节点的指针,child(2) 为指向该节点的左下角孩子节点的指针,child(3) 为指向该节点的左上角孩子节点的指针,child(4) 为指向该节点的右上角孩子节点的指针. 叶子节点采用 (I, $\wedge$, $\wedge$, $\wedge$, $\wedge$) 的形式,其中,I 为空间数据的 MBR.

2.2 RQOP 树的定义

定义 3 一棵 RQOP 树定义为一棵满足下列条件的四叉树:

1) 若根节点不是叶节点,则它至少有两棵子树.

2) 所有中间节点 node 至多有 4 棵子树、至少有 2 棵子树,并且若孩子节点存在,则满足:

node- > child(1)- > I, node- > child(4)- > $I_>^x$

node- > child(2)- > I, node- > child(3)- > I,

node- > child(3)- > $I_>^y$, node- > child(2)- > $I_>^y$

node- > child(4)- > $I_>^y$, node- > child(1)- > I.

3) 中间结点是 RQOP 树.

从定义可以看出:RQOP_树是递归定义的,并且对一个节点的孩子节点的几何位置有限制,即必须按照相对几何位置定位孩子节点.

2.3 RQOP 树的生成

RQOP 树的生成算法是由上向下的贪婪算法. 给定 n 个数据节点的 MBR. 首先,求出包含这 n 个数据的 MBR,作为这棵树的根节点的 I, 同时作为整个数据空间. 然后,依据这些数据的分布用 x 轴方向或 y 轴方向的垂直或平行地将数据空间分成四部分,使得每个部分中包含几乎相等个数的数据,如图 1 所示. 分别将包含这四个部分中的所有数据的 MBR 求出,用于生成该树根节点的四个孩子节点的数据项 I, 并建立与根节点的链接关系,从而完全确定了更节点的所有孩子节点. 对根节点的所有孩子节点递归地实施同样的操作,直至所有数据处理完毕. 就生成了一棵 RQOP 树.

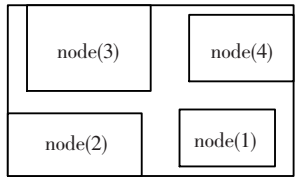

图 1 节点的几何划分

生成算法描述如下:

算法 1 Tree_Creation(I, n, head)

/* RQOP 树生成 */

输入:n 个数据矩形,存储于数组 I 中;

输出:头节点指针 head;

Begin

call Sorting_MBR(I, n, 0);/* 对 I 中的 n 个数据按定义 1 排序 */

计算节点 node 的 MBR,记为 I_1;

node- > I = I_1;

$n_1 = \lceil n \rceil$; $n_2 = n - n_1$;

把 I 中前面的 n_1 个 MBR 存入 I_1 中;

把 I 中后面的 n_2 个 MBR 存入 I_2 中;

call Sorting_MBR(I_1, n_1, 1);

call Sorting_MBR(I_2, n_2, 1);

$n(3) = \lceil n_1/2 \rceil$; $n(2) = n_1 - n(3)$;

$n(4) = \lceil n_2/2 \rceil$; $n(1) = n_2 - n(4)$;

把 I_1 中前面的 $n(3)$ 个 MBR 存入 II_3 中;
把 I_1 中后面的 $n(2)$ 个 MBR 存入 II_2 中;
把 I_2 中前面的 $n(4)$ 个 MBR 存入 II_4 中;
把 I_2 中后面的 $n(1)$ 个 MBR 存入 II_1 中;
for j =0 to 4 do
call Tree_Creation(II_j, $n(j)$, node->child(j));/* 递归构造中间节点 */
return head;
End

定理 1　对于给定的 n 个数据矩形,算法 1 能在有限步内正确地构造出相应的 RQOP－树,且算法的时间复杂度为 $O(n\log_2 n)$.

证明　1)可终止性. 在算法 1 中,每个中间节点依其所包含的数据矩形的分布情况被划分为四个部分. 至多有 $1+4+4^2\cdots+4^{\lceil\log 4n\rceil}=(4^{\lceil\log 4n\rceil+1}-1)/3=4(n-1)/3$ 个中间节点. 因此,只需进行有限步的划分即可完成树的构造,即,算法 1 在有限步内完成.

2)正确性. 在算法 1 中,根节点的数据域被划分成 4 个数据域,每个域中含有几乎相等数量的数据矩形,由此生成 4 个孩子节点的数据域的值. 对这四个孩子节点递归的进行同样的操作,直至只含有不超过 4 个数据矩形. 由 1)可知,至多有 $(4n-3)/3$ 个中间节点,故至多进行 $(4n-3)/3$ 次节点划分即可正确完成 RQOP 树的构造. 正确性是显而易见的.

3)时间复杂度. 划分根节点需 $n\log_2 n+2\lceil n/2\rceil\log_2\lceil n/2\rceil$ 次比较运算,而划分其四个孩子节点 $4\lceil n/4\rceil$ 次比较,…,算法 1 至多花费 $n\log_2 n+2\lceil n/2\rceil\log_2\lceil n/2\rceil+4\lceil n/4\rceil+4^2\lceil n/4^2\rceil+\cdots+4^{\lceil\log_4 n\rceil}\lceil n/\log_4 n\rceil\approx 2n\log_2 n+n\log_4 n\approx 3n\log_2 n$ 次比较运算即可完成,即该算法的时间复杂度为 $O(n\log_2 n)$.

从算法 1 的描述可以看出:叶子节点均位于最后两层上,树的高度为 $\lceil\log_4 n\rceil$.

2.4　RQOP 树的节点插入

假设:head 为已知一棵 RQOP 树的头节点的指针,II 为待插入数据的 MBR,则节点插入算法描述为:

算法 2 Node_Insertion(n, head, II)

输入:II 是待插入的数据;head 是树的根节点的指针,n 是树上包含数据的个数;

输出:树的头节点的指针 head;

Begin
node = head;
if II^x > node->child(1)->I 并且 node->child(1)->$I^y_>$ II then
call Node_Insertion(n, node->child(1), II);/* 插入到 node->child (1)中 */
else if $II_>{}^x$ node->child(4)->I 并且 $II^y_>$ node->child(1)->I then
call Node_Insertion(n, node->child(4), II);/* 插入到 node->child (4)中 */
else if $II_>{}^x$ node->child(2)->I 并且 node->child(2)->$I_>{}^y II$ then
call Node_Insertion(n, node->child(2), II);/* 插入到 node->child (2)中 */
else
call Node_Insertion(n, node->child(3), II);/* 插入到 node->child (3)中 */
node->I = MBR containing I and node->I;
/* 修改节点 node 的 MBR */;
$n=n+1$; /* 修改数据个数 */
return head;
End

定理 2　对给定的具有 n 个数据的 RQOP 树,算法 2 可在有限步内正确地完成节点的插入,其时间复杂度为 $O(\log_4 n)$.

从节点插入算法的描述可以看出:容易得到定理 2,故证明略.

2.5　RQOP 树的区域查询

对于区域查询,建立如下的剪枝规则:

规则 1 对于 RQOP 树的中间节点 node 和查询窗口 W,若 node－>$I\cap W=\varnothing$,则说明节点 node 中没有数据与查询窗口相交,故将该节点剪枝掉.

规则 2 对于 RQOP 树的中间节点 node 和查询窗口 W,若 node->$I\subset W$,则说明节点 node 中所有数据均与查询窗口相交,即,它们均为要查询的结果,故将该节点中的所有数据放到结果集合中,而不必到其孩子节点中做进一步的查询.

给定一棵 RQOP 树,其头节点为 head,查询窗口 W. 区域查询就是查找出该树上所有位于 W 内部或与 W 相交的数据矩形. 为此,从根节点开始,检查其数据域 head->I,是否被 W 包含或与 W 相交. 若不是,则说明 W 中没有该树中的物体;否则,若 head->$I\subseteq W$,则该树中的所有数据都为要查询的结果;不然的话,到其每个孩子节点中进行递归查找. 如果孩子节点 node 的数据域 node->$I\subseteq W$,则按照规则 1 将该节点中的所有叶子节点中所含的数据取出并进行记录;如果

node - > $I \cap W \neq \emptyset$ 且节点 node 不是叶子节点，则到其所有孩子节点中查找，如果是叶节点，则记录该数据项. 这个过程递归进行直至所有节点遍历完毕.

算法描述为：

算法 3 Range _ Query(W, head, Q)

输入：查询窗口 W，一棵 RQOP 树，头节点指针为 head；

输出：该树中所有与 W 相交的数据集 Q；

```
Begin
    node = head;
    if head = null then
      输出“RQOP 树中无数据”;return;
    if node- > I ∩ W = Φ then
      输出“窗口中没有 RQOP 树的数据”;
      return;
    else if W ⊃ head- > I then
      将树上的所有数据取出放到 Q 中，算法结束;
    else
      for j = 1 to 4 do
    if head- > child( j ) - > I ⊂ W then
              将 head- > child( j ) 中的所有数据放到 Q 中;
          else if head- > child( j ) 为叶子节点，且 node- > I ∩ W ≠ ∅ then
          将 head- > child( j ) 中的数据放到 Q 中;
    else
          call Range _ Query( W , head- >
          child( j ), Q );/ * 到节点 head 的第 j 个孩子节点中查找 * /
          endif
    enddo
  endif
  return Q ;
End
```

定理 3 对于给定的一棵头节点指针为 head 的 RQOP 树和查询窗口 W，算法 3 能在有限步内正确地找到该树中与 W 相交或位于其内部的所有数据.

定理 3 的证明显而易见，故省略. 在该算法中引入了查询窗口与节点的 MBR 的包含关系的判别，当查询窗口较大时可加快查询的速度.

3 实验评估

用 90 000 个随机生成的矩形对本文给出的 RQOP 树与 CIF 树、QR - 树、QR* - 树上进行区域查询效率对比.

3.1 测试环境

1）硬件环境：CPU 为 P4 1.8 GHz、内存为 256 MB、硬盘为 60 G 的个人计算机.

2）软件环境：Window's Xp Professional sp2 操作系统，用 Microsoft Visual Studio. NET2003 编程实现.

3）用于测试算法性能的查询区域：以数据空间的面积/体积为基准，取占数据空间分别为 0.000 05，0.002，0.001 25，0.005，0.011 25，0.02，0.125，0.5，1.125，2，12.5 和 50 的 12 个不同百分比的查询区域为测试，分别给其编号为 1 ~ 12.

3.2 4 种索引结构上对不同查询区域的查询情况

对 4 种索引结构，运用相同的 90 000 个数据矩形建立相应的数据结构，并分别在不同区域上进行区域查询，具体的情况如图 2 所示.

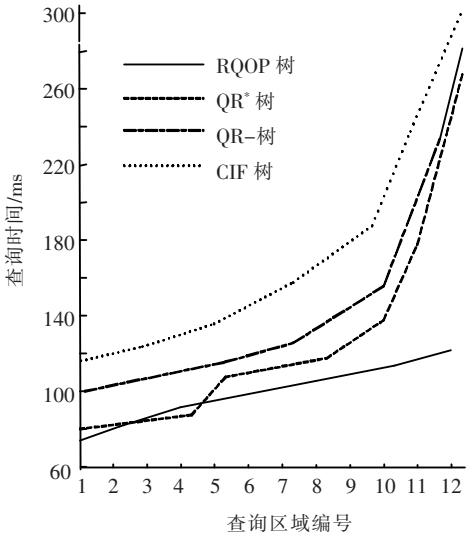

图 2 四种索引结构上不同区域查询的性能对比

从图 2 中可以看出：给出的索引结构上的区域查询效率有较大的提高，特别是对较大查询窗口的情况更是突出，明显好于其他 3 种索引结构. 图 3 表明了给出的索引结构中剪枝规则 2 的作用.

从图 3 可以看出：对于较小的区域查询，剪枝规则 2 对查询速度起到了阻碍的作用；而对较大的区域查询，剪枝规则 2 对查询速度起到了加快的作用. 因此，可先测算出对多大的查询区域（占数据空间的百分比），剪枝规则 2 对查询速度基本上不起作用，然后，将其存储在树中，当有新的查询要进行时，可先将新的窗口与存储的窗口相比较，如果比存储的大，则在查询算法中采用剪枝规则 2；否则，不采用. 这样，还会加快查询的速度.

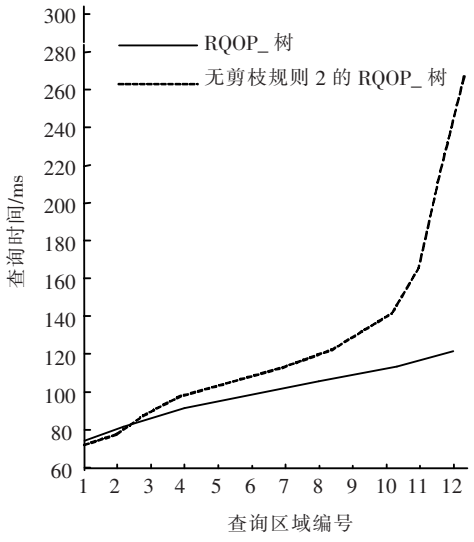

图 3　剪枝规则 2 在 RQOP_树的区域查询算法中的作用

4　结　论

1)在此结构中,对数据空间的分割是按照该空间所包含的数据的分布情况进行的,使树的高度尽可能的低,同时使同层兄弟节点间的交叠相对小.

2)在新的查询算法中引入剪枝规则,有效地加快了查询的速度. 通过实验证明了本文给出的索引结构的有效性.

参考文献:

[1] GUO Wei, GUO Jing, HU Zhiyong. Spatial Database Index Technique[M]. Shanghai:Shanghai Jiaotong University Press, 2006.

[2] YU Chenfu,HU Zhiyong, GUO Wei, *et al.* QR_tree: A hybrid spatial index structure[C]// Proceedings of the Second International Conference on Machine Learning and Cybernetics. Xi'an:IEEE, 2003: 459－463.

[3] ZHANG Qin, WANG Zhen-zhen. QR-tree:A kind of spatial index structure based on R-tree and quadtree [J]. Computer Engineering and Its Applications (in Chinese), 2004, 40(9):100－103.

[4] QIU Jianhua, TANG Xuebing, HUANG Huaguo. An index structure based on quad-tree and R*-tree-QR*-tree[J]. Computer Application (in Chinese), 2003, 23(8):124－126.

[5] 周巧临. PMR 树四叉树空间索引优化的应用研究[J]. 微计算机信息,2008, 24(3):175－177.

[6] 蒋华. 一种 PMR 四叉树空间索引效率分析模型的研究[J]. 计算机工程与应用,2006,42 (35):166－168.

[7] GUTTMANN A. R-trees: A Dynamic Index Structure for Spatial Searching[D]. SIGMOD: Proceedings of ACM SIGMOD International Conference on Management of Data, 1984:45－57.

(编辑　张　红)