|  |
| --- |
| 空间索引是对存储在介质上的数据位置信息的描述，用来提高系统对数据获取的效率。[GIS](http://baike.baidu.com/view/5094.htm)涉及的各种海量复杂数据存储于外存，如果对磁盘上的数据的位置不加以记录和组织，每查询一个数据项都要扫描整个数据文件，则这种访问磁盘的代价将严重影响系统的效率。因此索引的建立与处理至关重要。此外GIS所表现的地理数据多维性使得传统的[B树](http://baike.baidu.com/view/298408.htm)索引不再适合，因为B树所针对的字符、数字等传统数据类型是在一个[良序集](http://baike.baidu.com/view/657759.htm)之 中，即都是在一个维度上，集合中任给两个元素，都可以在这个维度上确定其关系只可能是大于、小于、等于三种，若对多个字段进行索引，必须指定各个字段的优 先级形成一个组合字段，而地理数据的多维性，在任何方向上并不存在优先级问题，因此B树并不能对地理数据进行有效的索引，所以需要研究特殊的能适应多维特 性的空间索引方式。  1984年Guttman发表了《R树:一种空间查询的动态索引结构》[1]一种高度[平衡树](http://baike.baidu.com/view/1991667.htm)，由中间节点和叶节点组成，实际数据对象的最小外接矩形存储在叶节点中，中间节点通过聚集其低层节点的外接矩形形成，包含所有这些外接矩形。其后，人们在此基础上针对不同空间运算提出了不同改进，才形成了一个繁荣的索引树族，是目前流行的空间索引。  R树是一种采用对象界定技术的高度平衡树,是B 树在 k 维空间上的自然扩展，它将空间对象按范围划 分，每个结点都对应一个区域和一个磁盘页，非叶结点的磁盘页中存储其所有子结点的区域范围，非叶结点的所有子结点的区域都落在它的区域范围之内；叶结点的 磁盘页中存储其区域范围之内的所有空间对象的外接矩形。每个结点所能拥有的子结点数目有上、下限，下限保证对磁盘空间的有效利用，上限保证每个结点对应一 个磁盘页，当插入新的结点导致某结点要求的空间大于一个磁盘页时，该结点一分为二。R树是一种动态索引结构，即：它的查询可与插入或删除同时进行，而且不 需要定期地对树结构进行重新组织。  R-Tree数据结构  （1）R-Tree是n叉树，n称为R-Tree的[扇（fan）](http://www.hudong.com/wiki/%E6%89%87%E5%87%BA)。  （2）每个结点对应一个矩形。  （3）叶子结点上包含了小于等于n的对象，其对应的矩为所有对象的外包矩形。  （4）非叶结点的矩形为所有子结点矩形的外包矩形。  R-tree具有以下性质：  （1）除根节点外，每个节点的项数介于最小项数m和最大项数M之间；  （2）根节点至少有两个孩子，除非它是叶子节点；  （3）所有叶子节点位于同一层；  （4）同一节点中项，其排列没有顺序要求  R-Tree的的评价标准为：  （1）位置上相邻的结点尽量在树中聚集为一个父结点。  （2）同一层中各兄弟结点相交部分比例尽量小。  R树是一种用于处理多维数据的数据结构，用来访问二维或者更高维区域对象组成的空间数据.R树是一棵平衡 树。树上有两类结点：叶子结点和非叶子结点。每一个结点由若干个索引项构成。对于叶子结点，索引项形如(Index，Obj\_ID)。其中，Index表 示包围空间数据对象的最小外接矩形[MBR](http://en.wikipedia.org/wiki/Minimum_bounding_rectangle)，Obj\_ID 标识一个空间数据对象。对于一个非叶子结点，它的索引项形如(Index，Child\_Pointer)。 Child\_Pointer指向该结点的子结 点。Index仍指一个矩形区域，该矩形区域包围了子结点上所有索引项MBR的最小矩形区域。一棵R树的如图1所示。 |
| |  | | --- | |  |   R-Tree算法描述  （I）插入算法       基本思想：找到合适的叶子节点，插入之，若需分裂，则由下至上调整MBR值。算法如下：       I1: 调用ChooseLeaf来选择一个合适的叶子节点L以容纳需插入项E       I2: 若L中还能容纳E，则加入之；否则调用SplitNode来获取两个节点L和LL，它们包含E和L中原有的所有项       I3: 调用AdjustTree,传递参数L,LL(若产生了分裂)       I4: 若节点分裂向上传播导致根节点的分裂，则生成新的根节点。  **算法ChooseLeaf:** 选择一个合适的叶子节点以放置新项E。合适的评价标准是插入E后的节点MBR面积增加度最少。       C1: 设N指向根结点root       C2: 若N是叶子节点，返回N       C3: 若N不是叶子节点，让F表示N中的一项，该项F容纳E后，则N在面积上只需作面积最小扩展       C4: 设N指向叶子节点，则返回C2.  **算法AdjustTree:**从叶子节点向根节点进行调整       A1: 设N=L,若L进行了分裂，则设NN=LL       A2: 若N为根节点，则返回       A3: 设P为N的双亲节点，EN为节点P中指向N的项，调整项EN的MBR       A4: 若NN存在，创建一个新项ENN,使其指向NN，同时计算出ENN的MBR.将ENN加入P，若不能容纳则调用SplitNode产生节点P和PP,包含ENN和原来P中所有项       A5: 设N=P, NN=PP, 转至A2.  **算法SplitNode:**将M+1项分成两组，将它们加入到两个新节点。  判断节点分裂好坏的一个标准为：分裂后，两个新节点对应的MBR的面积之和最小。下图展示了节点分裂的一个例子。  一个时间复杂度为二次（Quadratic）的分裂算法如下：        S1： 调用PickSeeds选出两项，将它们分别作为两组的第一个元素；        S2： 若所有项都已分配完，则返回；若一组中的项如此之少，以至于将剩下的所有项添至其中才能满足项数达到m的要求，则进行分配且返回；        S3： 调用PickNext选择下一项，将其分配到某组中，该组在容纳该项后，MBR只需作最小面积扩展，转至S2.  算法PickSeeds：        PS1： 对每一项E1和E2，计算d=area(E1和E2合并之后的MBR) – area(E1) – area(E2)；        PS2：选择d值最大的一对项。  算法PickNext：        PN1：对每一项E，计算d1=<将E加入组1后MBR增加的面积>，同理计算d2；        PN2：选出d1和d2值差距最大的项。  **删除**  R树的删除操作与B树的删除操作会有所不同，不过同B树一样，会涉及到压缩等操作。相信读者看完以下的伪代码之后会有所体会。R树的删除同样是比较复杂的，需要用到一些辅助函数来完成整个操作。  伪代码如下：  Function：Delete  描述：将一条记录E从指定的R树中删除。  D1：[找到含有记录的叶子结点] 使用FindLeaf方法找到包含有记录E的叶子结点L。如果搜索失败，则直接终止。  D2：[删除记录] 将E从L中删除。  D3：[传递记录] 对L使用CondenseTree操作  D4：[缩减树] 当经过以上调整后，如果根结点只包含有一个孩子结点，则将这个唯一的孩子结点设为根结点。    Function：FindLeaf  描述：根结点为T，期望找到包含有记录E的叶子结点。  FL1：[搜索子树] 如果T不是叶子结点，则检查每一条T中的条目F，找出与E所对应的矩形相重合的F（不必完全覆盖）。对于所有满足条件的F，对其指向的孩子结点进行FindLeaf操作，直到寻找到E或者所有条目均以被检查过。  FL2：[搜索叶子结点以找到记录] 如果T是叶子结点，那么检查每一个条目是否有E存在，如果有则返回T。    Function：CondenseTree  描 述：L为包含有被删除条目的叶子结点。如果L的条目数过少（小于要求的最小值m），则必须将该叶子结点L从树中删除。经过这一删除操作，L中的剩余条目必 须重新插入树中。此操作将一直重复直至到达根结点。同样，调整在此修改树的过程所经过的路径上的所有结点对应的矩形大小。  CT1：[初始化] 令N为L。初始化一个用于存储被删除结点包含的条目的链表Q。  CT2：[找到父条目] 如果N为根结点，那么直接跳转至CT6。否则令P为N 的父结点，令EN为P结点中存储的指向N的条目。  CT3：[删除下溢结点] 如果N含有条目数少于m，则从P中删除EN，并把结点N中的条目添加入链表Q中。  CT4：[调整覆盖矩形] 如果N没有被删除，则调整EN.I使得其对应矩形能够恰好覆盖N中的所有条目所对应的矩形。  CT5：[向上一层结点进行操作] 令N等于P，从CT2开始重复操作。  CT6：[重新插入孤立的条目] 所有在Q中的结点中的条目需要被重新插入。原来属于叶子结点的条目可以使用Insert操作进行重新插入，而那些属于非叶子结点的条目必须插入删除之前所在层的结点，以确保它们所指向的子树还处于相同的层。          R树删除记录过程中的CondenseTree操作是不同于B树的。我们知道，B树删除过程中，如果出现结点的记录数少于半满（即下溢）的情况，则直接把这些记录与其他叶子的记录“融合”，也就是说两个相邻结点合并。然而R树却是直接重新插入。    同样，我们用图直观的说明这个操作。  http://hi.csdn.net/attachment/201106/8/8394323_1307512429351R.jpg  http://hi.csdn.net/attachment/201106/8/8394323_1307512428Ii2z.jpg    假 设结点最大条目数为4，最小条目数为2。在这张图中，我们的目标是删除记录c。首先使用FindLeaf操作找到c所处在的叶子结点的位置——R11。当 c从R11删除时，R11就只有一条记录了，少于最小条目数2，出现下溢，此时要调用CondenseTree操作。这样，c被删除，R11剩余的条目 ——指向记录d的指针——被插入链表Q。然后向更高一层的结点进行此操作。这样R12会被插入链表中。原理是一样的，在这里就不再赘述。  http://hi.csdn.net/attachment/201106/8/8394323_13075124166AX3.jpg  http://hi.csdn.net/attachment/201106/8/8394323_1307512414AUi3.jpg  有 一点需要解释的是，我们发现这个删除操作向上传递之后，根结点的条目R1也被插入了Q中，这样根结点只剩下了R2。别着急，重新插入操作会有效的解决这个 问题。我们插入R3，R12，d至它原来所处的层。这样，我们发现根结点只有一个条目了，我们把这个根结点删除，它的孩子结点，即R5，R6，R7，R3 所在的结点被置为根结点。至此，删除操作结束。  如何将一个矩形集分裂成合适的两部分，是影响R树检索效率的一个重要因素。  1．以面积作为标准:即分裂后两部分的MBR的和最小。但是算法基于穷举，时间复杂度很大(指数级)。  2. 平方耗费算法:（时间复杂度为平方的近似算法）  (1)  首先从要分裂的矩形集中选取在分裂后最不可能在同一类中的两个矩形作为种子,作为两类中的第一个矩形  (2)  将剩余的矩形依次的分配到这两个类中。  该算法不保证分裂后的面积和最小。  来源： <<http://blog.csdn.net/zhouxuguang236/article/details/7898272>> |