多维数据索引结构回顾*)

Review of Multi-Dimensional Index Structures

董道国　薛向阳　罗航哉

(复旦大学计算机科学系　上海 200433)

Abstract Similar to conventional relation databases, fast search and query operations in multi-dimensional databases require efficient index structures. The traditional index structures can't be applied to the multi-dimensional databases because of the special characteristics. Many multi-dimensional index structures have been proposed. In this paper, we classify these index structures by their intrinsic features, and introduce some representative index structures in detail.

Keywords Multi-dimensional index structure, Database, Similarity-based query

一、引言

最初,多维数据库主要用于计算机图形学、地理信息系统等。现在,多维数据库的应用扩展到医学图像处理、计算机视觉和多媒体数据库等领域。尤其是在多媒体数据库中,对多媒体对象的描述更加复杂,例如:对图像来说,常用颜色直方图、主色调、Tamura 纹理等特征描述图像;对文本来说,常用矢量空间模型来描述文档;对视频来说,常用颜色、纹理、形状和运动等特征来描述视频镜头。但是,不管采用哪一种描述方法,都需要用多维空间中的点、线段或区域等来表示这些多媒体对象。

很多实际应用需要从多维数据库中快速查找到特定数据,例如:在图像数据库中查找与给定图像最相似的图像;在地理信息系统中需要查找离某个城市最近的河流。为了支持这些快速查找操作,必须借助高效索引结构。由于传统数据索引结构(如 B-树等)无法适用于多维数据,因此,人们相继提出了大量的多维数据索引结构,本文将回顾多维数据索引结构的发展进程。

本文将介绍多维数据以及多维数据索引结构的特点;定义了多维数据库上的种种查询方式;给出这些索引结构的分类方法;详细讨论了一些具有代表性的索引结构;介绍了几种最新提出的索引结构;最后给出了本文结论。

二、多维数据及多维数据索引结构的特点

所谓多维数据,是指多维空间中的数据,例如二维空间中的点、线段、矩形,三维空间的球、立方体以及高维空间中的点数据等。一般来说,多维数据有以下一些特点[14]:

1)复杂的结构:对于多维数据,它有可能是一个多维空间的点数据,也有可能是复杂的多边形或多面体,一般不能象传统的关系型数据库一样用固定大小的条目来保存。

2)动态的特性:在插入和删除的过程中,还往往伴随着对数据本身的修改。

3)数据的海量:多维数据库往往是比较大的,例如,一般的地图大概需要几千兆字节的存储空间。

4)多样化的操作:对多维数据库而言,没有标准操作,往往要根据实际应用的需要确定。

5)时间代价大:尽管对多维数据库的操作所花费的时间各不相同,但一般远高于传统的关系型数据库的操作。

6)不能排序:无法对空间数据进行线形排序以使得那些在多维空间中相邻的数据仍然能够相邻。

正是由于多维数据具有以上各种特点,因此要求索引结构也相应具有以下一些特征[14]:

1)动态构造:由于数据可以在数据库以任意顺序插入或删除,其索引也应该与之保持一致,即索引结构也必须支持动态的插入和删除。

2)二级/三级存储管理:尽管主存容量日益增大,仍不能将一个完整的数据库保存在主存中,因此索引结构要充分考虑到二级以及三级的存储管理。

3)支持尽量多的操作:不能以牺牲其它的操作而支持某一种特定的操作。

4)独立于输入数据及插入顺序:支持各种多维数据以及任意的插入顺序。

5)可增长性:索引结构要能够适应数据库大小的增长。

6)时间的有效性:查找速度必须是快速的。

7)空间的有效性:一个索引结构相对于其原数据应是比较小的,要保证一定的空间利用率。

8)并行性及可恢复性。

三、多维数据库的查询方式

根据应用领域的不同,多维数据库的查询方式也各不相同。对于给定的数据库 DB,常用的查询方式有[14]:

1)精确匹配查询(EMQ:Exact Match Query):对于给定的查询数据 q,从 DB 中找出所有与 q 相同的数据:

$$EMQ(q)=\{o\in DB \mid o=q\}$$

2)点查询(PQ:Point Query):给定点数据 q,从数据库中找出所有包含点 q 的数据:

$$PQ(q)=\{0\in DB \mid q\cap o=q\}$$

3)窗口查询(WQ:Window Query):给定一个 d 维的矩形查询区间 $I^d=[l_1,u_1]\times[l_2,u_2]\times\cdots\times[l_d,u_d]$,从数据库中找出至少包含一个 I 中的点的所有数据:

*)本文得到自然科学基金课题(60003017)、自然科学基金重点课题(69935010)、863 计划、上海市教委资助基金等资助。**董道国**　硕士研究生,主要从事高维数据索引结构研究。**薛向阳**　教授,博士生导师,主要从事基于内容的多媒体信息检索技术研究。**罗航哉**　助教,硕士,主要从事基于内容的多媒体信息检索技术研究。

$WQ(I^d)=\{o\in DB|I^d\cap o\neq\phi\}$

4)相交查询(IQ:Intersection Query):给定具有一定形状的空间数据 q,从数据库中找出至少包含 q 中一点的所有数据:

$IQ(q)=\{o\in DB|q\cap o\neq\phi\}$

5)包含查询(EQ:Enclosure Query):给定查询数据 q,从数据库找出所有包含 q 的数据:

$EQ(q)=\{o\in DB|q\cap o=q\}$

6)被包含查询(CQ:Containment Query):给定查询数据 q,从数据库中找出所有被 q 包含的数据:

$CQ(q)=\{o\in DB|q\cap o=o\}$

7)邻接查询(AQ:Adjacency Query):给定查询数据 q,从数据库中找出所有同 q 邻接的数据:

$AQ(q)=\{o\in DB|q\cap o\neq\phi\wedge q'\cap o'=\phi\}$

其中 q'、o' 分别表示 q 和 o 的内部区域。

8)范围查询(RQ:Range Query):给定查询数据 q 与查询距离门限 T,从数据库中找出所有与 q 距离小于 T 的数据:

$RQ(q,T)=\{o\in DB|\ \|o-q\|\leqslant T\}$

9)k-近邻查询(kNNQ:k Nearest-Neighbor Query):给定查询数据 q 及正整数 k,从数据库中找出距离 q 最近的 k 个数据:

$kNNQ(q,k)=\{o_0\cdots o_{k-1}\in DB|\forall\ e\in DB\backslash\{o_0\cdots o_{k-1}\},\|o_i-q\|\leqslant\|e-q\|,o\leqslant i\leqslant k-1\}$

当 k=1时,又称为'最近邻查询'。

四、多维数据索引结构的分类

在近三十年对多维数据索引的研究过程中,已经提出了大量的索引结构,如图1所示。根据这些索引结构的特点分类如下:

1)根据处理数据的类型,可以分为点数据类和空间数据类。点数据类是指那些只能处理点数据的索引结构,如 k-d-树[2]、TV-树[8]等;空间数据类指既能处理点数据,又可以处理线、矩形等具有一定形状的数据的索引结构,如 R-树[5]、R*-树[1]等。

2)根据索引创建时数据组织形式,可以分为静态结构类和动态结构类。静态结构类是指用批处理的方式将全部数据构建成索引,不能支持动态的插入和删除,如 packed R-树[7];而动态结构类指数据依次插入而生成的索引结构,如动态 R-树[5]、动态 R*-树[1]等。

3)根据划分方法的不同,可以分为数据划分方法类、空间划分方法类以及混合型划分方法类。数据划分方法是根据数据所在的位置进行层次划分,如 R-树等;空间划分方法则是将整个数据空间逐渐划分为互相邻接的子空间,如 k-d-树等。混合型是指既根据空间进行划分又根据数据进行划分,如 Hybrid-树[26]等。

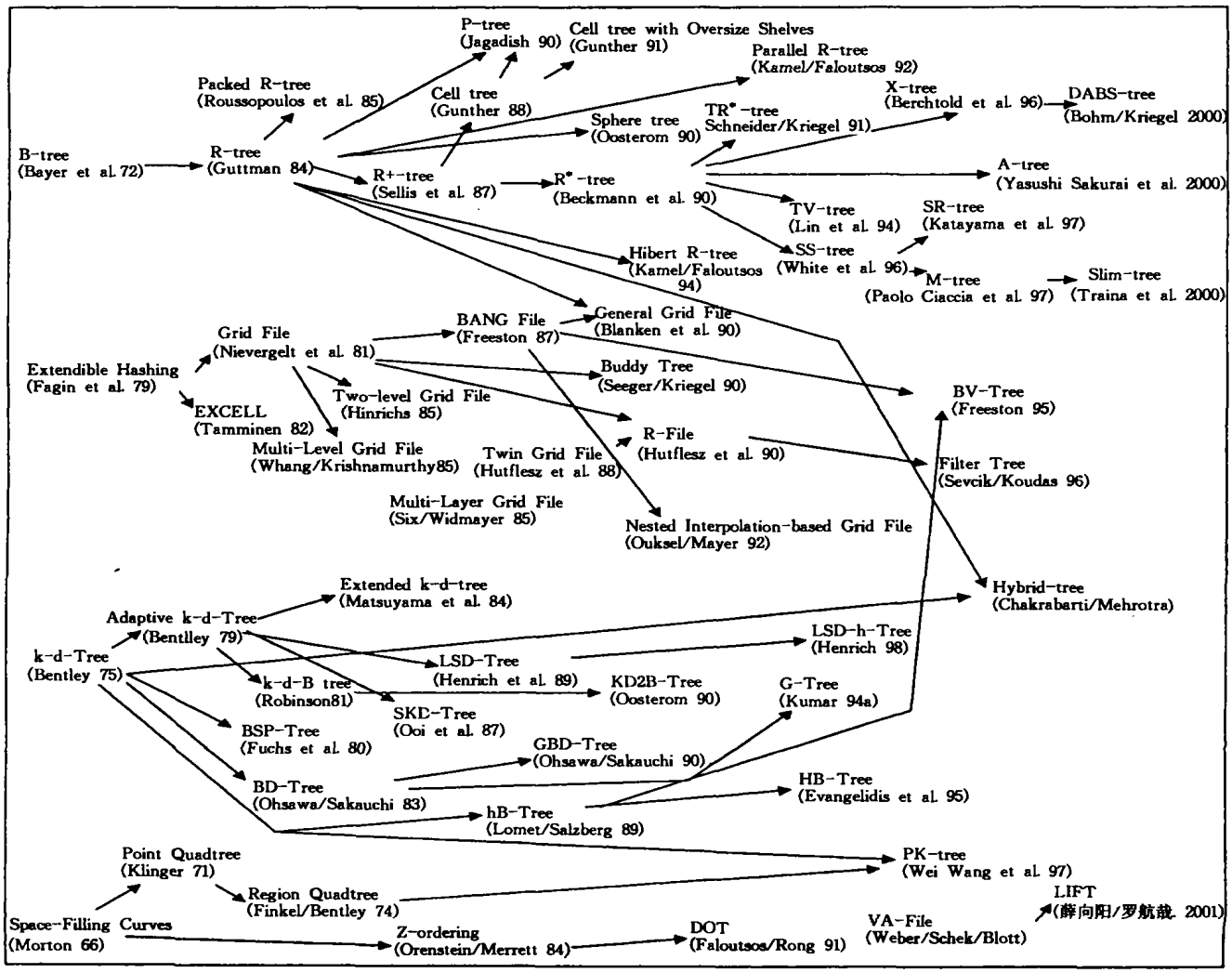


图1 多维数据索引结构的发展历史[14]

4)根据索引的组织形式,可以分为树形结构类和非树形结构类。树形结构类指索引结构是按照树的形式组织的,如k-d-树、R-树等;非树形结构类是指索引结构不是按照树的形式组织的,如VA-File[17]等。

5)根据目录节点的形状,可以分为矩形、球形和混合形。在构造索引结构的时候,目录节点形状的选择直接影响检索的效率,选择为矩形的有k-d-树、R-树、R*-树等;选择为球形的有SS-树[15]、TV-树[8]等;将矩形和球形结合起来的称为混合形,如SR-树[16]等。

五、几种代表性的多维数据索引结构

限于篇幅,无法对已经提出的众多索引结构作一一介绍,因此我们从中选择了一些具有代表性的结构给予详细描述。

5.1 R-树类

R-树类的索引结构类似于B-树,一般都具有层次性的数据结构,最先出现的R-树类索引结构是在1984年由Guttman提出的R-树[5]。此后人们对它的算法和结构进行了一些改进,不过大多都保持了与R-树相同或类似的结构特点。

5.1.1 *R-树* 是一种平衡树,具有类似于B-树的层次结构。在R-树中存放的数据并不是原始数据,而是这些数据的最小外接矩形(MBR),它具有以下的一些性质:

·如果用M来表示一个节点中可存放的项数目的最大值,用m(m≤M/2)来表示一个节点中可存放的项数目的最小值,则除了根节点外,每个节点中必须包含m—M个项。

·根节点中至少要包括两个项,除非它是个叶子节点。

·节点中的每个项的结构为(rect,pointer),在内部节点中,rect是其子节点中所有数据的MBR,pointer指向该子节点;在叶子节点中,rect是实际数据的MBR,pointer指向实际数据存放的位置。

·所有的叶子节点出现在同一层上。

图2是一个简单R-树的例子,它反映了R-树的结构特点。

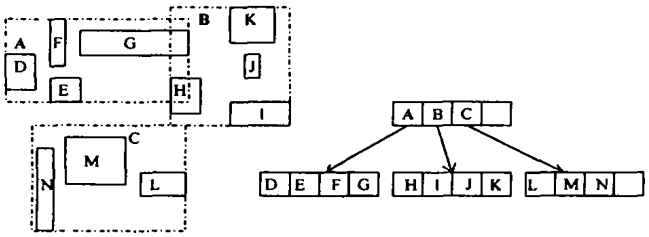


图2 R-树的例子

R-树的查找算法类似于B-树,它是从根节点出发,对内部节点,检查其每一个项是否与要查找的区域相重叠,如果重叠的话,则查找该项所指向的子节点,直至查找到叶子节点。由于在R-树中存放的是MBR,这些MBR可能相互重叠(如图2中的A和B),因此无法保证查找操作只搜索一个路径即可成功,在最坏的情况下,一次查找操作可能要遍历所有的路径。

向R-树中插入一个数据,实际上是插入该数据的MBR。和查找操作不同的是,一次插入操作只需要遍历一个从根节点到叶子节点的路径,首先从根节点开始,在经过的每一个内部节点中选择这样的项:它所指向的子节点为了包纳下要插入的数据,其MBR的面积要求扩大最小,如果出现相同的情况,则选择MBR最小的一个。当到达叶子节点后,如果该节点中还有剩余的空间,就将数据插入到节点中,并按原路径返回,依次修改其祖先节点中相应项的MBR;假如在叶子节点中已经没有足够的空间存放下要插入的数据,则要发生分裂,生成一个新的叶子节点,并在其父节点中插入一个指向新叶子节点的项,如果在其父节点同样出现了溢出,也要产生分裂,如此下去直至根节点,如果根节点也发生了分裂,则生成一个新的根节点,树的高度增加1。

要在R-树中删除一个数据,首先要进行一次精确查询操作,如果找到了该数据,则从节点将其删除,在删除之后如果节点中有足够的项(即项数目不小于m),就依次修改其祖先节点中相应项的MBR,删除结束;如果在删除后节点中项的数目小于m,则将该节点所有的项保存到一个临时缓冲区内,并将该节点从树中删除,如果节点删除后影响到其父节点也出现相同的情况,则也要做相同的操作,如此直至根节点,然后再将临时缓冲区内所有的项重新插入到树中。如果在删除完成之后根节点只有一个子节点的话,则这个子节点就成为新的根节点,树的高度减1。

5.1.2 *其它几种R-树类索引结构* 在R-树提出之后,人们对它的结构和算法进行了一些改进,以期望得到更好的性能,下面列出主要的几种:

·R*-树[1]:R*-树对R-树的插入算法和分裂算法进行了一些改进,主要体现在以下两点:第一,提出了强制重新插入的概念,即当一个节点在插入过程中发生了溢出,并不急于进行分裂,而是首先看一下该层节点在这次插入过程中有没有进行过重新插入,如果没有的话,则选择一定比例的项从该节点中删除并重新插入到树中,而如果该层已经有节点进行过重新插入,才对该节点进行分裂;第二,当节点进行分裂的时候,不仅要考虑分裂后两个新节点的面积,还要考虑分裂后节点周长以及该层节点的重叠面积等因素。

·X-树[3]:X-树中引入了超节点的概念,当节点发生溢出的时候,首先考虑对节点选择合适的分裂算法,以使节点分裂以后重叠区域小到一定程度,假如无法避免分裂后出现较严重的区域重叠,则不分裂节点,而是扩大节点大小以放入更多的项,形成超节点。

·SS-树[15]:SS-树中采用超球代替了原来的进行数据划分的超矩形,以更好地支持相似性查询。

·SR-树[16]:它是在分析了超矩形和超球两种不同的数据划分方法的优缺点后,将两者结合起来,形成了SR-树,从而取得更好的性能。

5.2 网格文件类

5.2.1 *网格文件*[9] 是一种典型的基于哈希的存取方式,它是由包含着很多与数据桶相联系的单元的网格目录来实现的,一般一个数据桶为硬盘上一个磁盘页。每个单元只对应着一个数据桶,而一个数据桶往往可以包含着几个相邻的单元。随着数据的增多,网格目录可能会慢慢变大,所以往往将其保存在硬盘上,但为了保证在进行精确查询的时候能仅用两次I/O操作就可找到相应的记录,一般将网格本身保存在主存中,网格是用d(数据的维数)个一维的数组来表示的,这些数组称为刻度。当我们进行精确查询的时候,首先用这些刻度来定位包含要查找的记录的单元,假如这个单元不在主存中,那么将进行一次I/O操作,将这个单元从硬盘调入主存,在这个单元中包含着一个指向可能找到记录的页的指针,取这个指针所指的页又需要一次I/O操作。而对于范围查询,需要检查所有与要查询的区域相交的单元。图3就是一个

网格文件的例子。

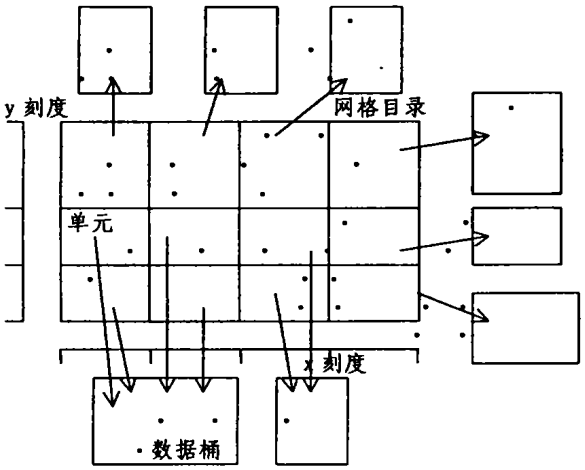


图3 网格文件的例子

当向网格文件插入一个数据的时候，首先要进行一次精确查询以确定该数据项应当插入的单元以及对应的数据页（数据桶所存放的磁盘页）。如果在这个页中还有足够的空间的话，将数据插入即可。假如已经没有足够的空间，则要根据与该页相联系的单元的数目来决定处理的方法，当有几个单元指向该页的时候，则检查现在的刻度中是否有合适的超平面能够成功地将该页分开，如果有的话，就产生一个新的页并根据这个超平面将数据分配在两个页中；如现存的超平面没有合适的，或者只有一个单元指向该页时，我们将引入一个新的超平面并产生一个新的页，然后根据这个超平面将数据分配，同时要将这个超平面插入到相应的刻度中，而所有与此超平面相交的单元也要发生分裂。

在网格文件中删除一个数据，也要先进行一次精确查询确定该数据所在的单元及对应的数据页，并将其删除。假如在删除之后，这个页中存储的数据低于一定的数目后，我们需要做相应的处理，根据当前刻度对空间的分隔情况，或者选择同其相邻的页合并，或者选择将刻度中的一个超平面取消。

5.2.2 *其它几种网格文件类索引结构* 与网格文件类似的还有 EXCELL、两层网格文件以及"twin"网格文件等：

·EXCELL[22]与网格文件不同之处在于其所有的网格单元是相同大小的，因此它的每次分裂都将导致目录大小的成倍增长。

·两层网格文件[23]的基本思想是再增加一个网格文件，形成两层网格来管理目录，其中第一层被称为根目录，它是第二层目录的一个大致描述，它具有指向第二层目录的指针，而第二层才是真正的目录，它包含有指向数据页的指针。这种结构的好处是，当发生分裂的时候，所产生的影响被限制在第二层目录的范围内，从而没有过多影响其它数据。

·"twin"网格文件[6]也是引入了另外一个网格文件，不过与两层网格文件不同的是，这两个文件的关系是对等的，而且每个文件都覆盖了整个空间，数据在这两个文件中的分布是动态的，当在插入过程中某个文件的单元所指向的页出现溢出的时候，将在这两个网格文件之间重新分配数据。

5.3 k-d-树类

5.3.1 k-d-树[2] 是一种在 k 维空间中的二叉查找树，它主要存储的是点数据，在每一个内部节点中，它用一个 k－1维的超平面将节点所表示的 k 维空间分成两个部分，这些超平面在 k 个可能的方向上交替出现，而且在每一个超平面中至少要包括一个点数据，图4就是一个 k-d-树的例子。

在 k-d-树的查找和插入操作是很简单的，而删除操作则有点复杂，因为一个点的删除可能会导致它的子树的重建。由于 k-d-树只能处理点数据，我们对其它具有一定形状的数据只好用它们的中心点来代替。需要指出的是当数据插入的顺序不同的时候，k-d-树的结构也不同，而且数据会分散出现在树的任何地方，而不是仅出现在叶子节点上。

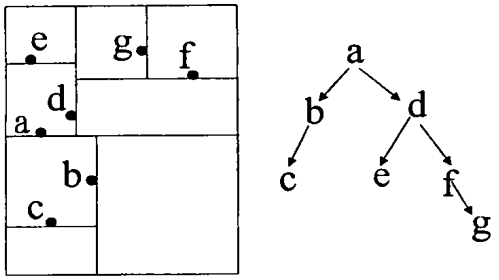


图4 k-d-树的例子

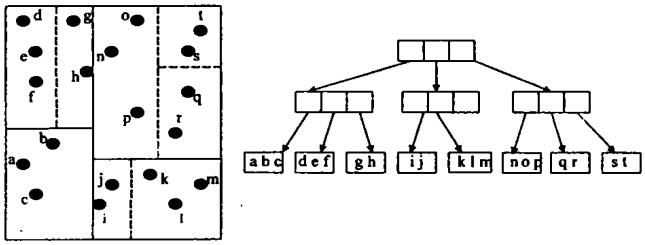


图5 k-d-B-树的例子

5.3.2 *其它几种 k-d-树类索引结构*

·Adaptive k-d-树[21]：它对 k-d-树的结构做了少许改变，在分裂的时候选择合适的超平面使分裂后的两部分包含相同数量的数据，而且在这些超平面上不一定非要包含数据以及它们的方向也不一定严格地在这 k 个可能的方向上交替出现。在这种结构里，内部节点表示的是分裂的超平面，所有的数据都出现在叶子节点中，每个叶子只能包含一定量的数据，当超过这个数量的时候要发生分裂。

·k-d-B-树[11]：它是将 Adaptive k-d-树同 B-树的特点结合到了一起，首先，它采用了 Adaptive k-d-树中用超平面来划分节点中的空间的方法，不过对于一个内部节点所表示的空间，可以用多于一个的超平面来划分，形成了几个相邻的子空间，每个子空间对应着一个内部节点，所有的数据均存储在相应的叶子节点中。从图5中可以看出，它的结构类似于 B-树，不过在 k-d-B-树中不能保证最小空间利用率。

5.4 四叉树(Quadtree)类

同 k-d-树类似，四叉树[4]也是用超平面的方法来划分整个空间的，不过不同的是，四叉树不再是二叉树，在维数为 k 的空间中，每一个内部节点具有 2^k 个孩子，每个孩子对应着一个子空间。例如，对于一个在二维空间的四叉树，每个内部节点拥有4个孩子，每个孩子对应着一个矩形，这四个矩形一般用 NW、NE、SW 以及 SE 来表示（NW 表示西北方向的那个矩形）。用这种方法将整个空间逐渐划分到每一个矩形中所包含的数据的个数低于一定的数目为止，显然这样得到的四叉树不能保证是平衡树，在数据比较密集的地方树的深度将高于其它的地方，如图6所示。

四叉树的查找操作类似于普通的二叉查找树，在每一层中我们要从四个中选择要继续查找的子树，对于点查询，只需

要选择其中的一个合适的就行了，而对于范围查询可能需要几个。按照这个方法递归执行下去直到查找到了树的叶子节点为止。

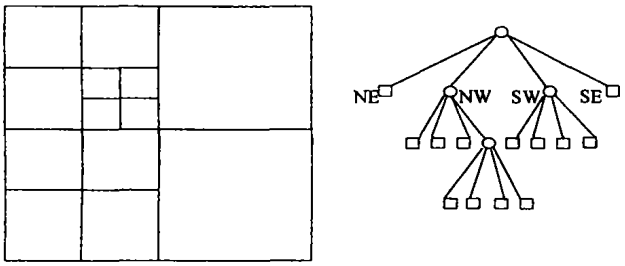


图6 四叉树的例子

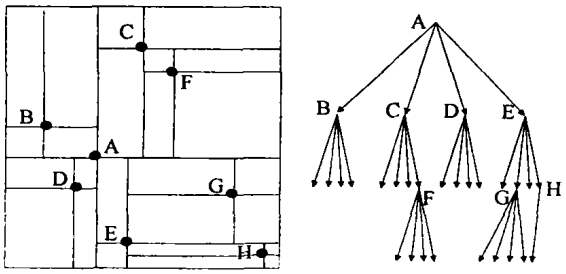


图7 点四叉树的例子

点四叉树(Point Quadtree)[4]是四叉树的一个变种，它是由点数据一个接一个地连续插入而构造出来的，对于每一个点，我们首先进行一个点查找操作，假如我们没有在树中发现这个点，我们就将这个点插入到刚才的查找操作终止的叶子节点，而这个节点所表示的区域以新插入的点为中心分为2^k个子空间。如果想从点四叉树中删除一个点的话，则会引起相应的子树的重建，一个简单的方法是将所有子树上的数据重新插入。图7是一个点四叉树的例子。

5.5 空间填充曲线

这一类索引结构的特点就是希望找到某种方法对多维空间中的数据进行近似的排序，使得原来在空间中较为接近的数据能在排序后以比较高的概率靠在一起，那么就可以用一维数据对它们进行索引。用这种方法在点查询操作中能够取得良好的效果，但进行范围查询时就会比较麻烦了。

根据这种思路，人们就提出了几种将多维空间中的点数据影射到一维空间并进行排序的方法，图8中列出了四种，详细的描述请参看文[12]。

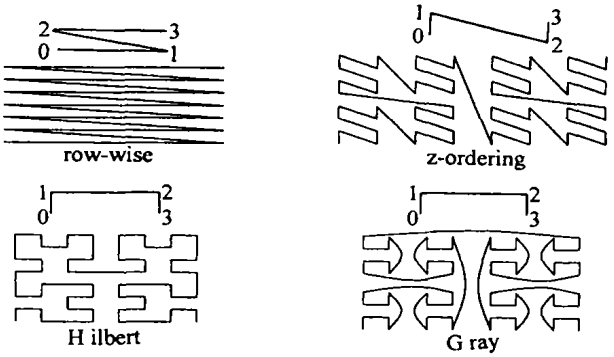


图8 四种空间填充曲线

所有的空间填充曲线都有一个重要的优点，就是对任何维数的数据都可以处理，前提是影射到的一维空间的键值可以任意大。但这种方法也有一个明显的缺点，当将两个不同区域的索引组合到一起的时候，至少要对其中的一个进行重新编码。

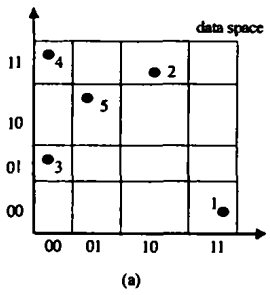
5.6 VA-File

VA-File[17]的基本思想是将高维点数据进行压缩和近似存储。首先，对每一维 j 分配一定的比特数 b_j，对于给定总的比特数 b 以及维数 d，则：

$$b_j=\left[\frac{b}{d}\right]+\begin{cases}1 & j\leqslant b \bmod d\\ 0 & \text{其它}\end{cases}$$

每一维上的比特数决定了在这一维上的刻度数，如图9(a)所示，一般来说，第 j 维需要$2^{b_j}+1$个刻度来得到2^{b_j}个区间，其中第一个刻度对应着该维上的最小值，最后一个刻度对应着该维上的最大值，其余的刻度值对该维进行等分，在插入和删除过程中这些刻度值一般保持不变，只有在区间之间的点数的差别超过了一定的门限后才进行重新计算。

对每一个点数据，将它每一维所在区间对应的比特串串联起来即为它的近似值，如图9(b)所示，可以看到，用两个浮点数表示的点现在可以用4个比特来表示，从而对原来的数据实现了压缩。当进行查询的时候，VA-File 起到过滤的作用，也就是说首先对这些比特串顺序扫描，根据比特串所对应区间的边界来确定哪些可能是查询结果的数据，然后再对得到的结果进行精确的查询。



	Point data	VA-File
1	0.9 0.1	11 00
2	0.6 0.8	10 11
3	0.1 0.4	00 01
4	0.1 0.9	00 11
5	0.3 0.7	01 10

(b)

图9 VA-File 的例子

六、最新进展

最近，又有人提出了一些新的索引结构，它们将多种索引思想结合起来以取得更好的性能，如 A-树等。

A-树[18]的特点是将近似的想法应用到了 R-树类索引结构上。对于 R-树类索引结构，当维数逐渐增大的时候，每一个节点中可以存放的项数迅速减少，这必然会降低查询时的性能，因此在 A-树中提出了 VBR 的概念。

VBR 是根据子 MBR 在父 MBR 中的相对位置而得到的。设 A 为 d 维空间中的父 MBR，则 A 可以用主对角线上的两个点来表示 $A=(a,a')$，其中 $a=(a_1,a_2,\cdots,a_d)$，$a'=(a'_1,a'_2,\cdots,a'_d)$，B 为子 MBR，$B=(b,b')$，其中 $b=(b_1,b_2,\cdots,b_d)$，$b'=(b'_1,b'_2,\cdots,b'_d)$。由于 B 包含在 A 中，因此有 $a_i\leqslant b_i\leqslant b'_i\leqslant a'_i$，(i=1,2…,d)。可以给定一个合适的正整数 q，用下面的方法来求出 b_i,b'_i 的近似值：

$$Q_i(b_i)=a_i+\frac{(a'_i-a_i)h_i(b_i)}{q}$$

其中 $h_i(b_i)=\begin{cases}q-1 & (b_i=a'_i)\\ \left\lfloor\left(\frac{b_i-a_i}{a'_i-a_i}\right)\cdot q\right\rfloor & \text{其它}\end{cases}$

$$Q_i(b'_i)=a_i+\frac{(a'_i-a_i)h_i(b'_i)}{q}$$

其中 $h_i(b'_i)=\begin{cases}1 & (b'_i=a_i)\\ \left\lceil \left(\frac{b'_i-a_i}{a'_i-a_i}\right)\cdot q \right\rceil & \text{其它}\end{cases}$

显然，$a_i\leqslant Q_i(b_i)\leqslant b_i\leqslant b'_i\leqslant Q_i(b'_i)\leqslant a'_i$，那么由 $v=(Q_1(b_1),Q_2(b_2),\cdots,Q_d(b_d))$ 以及 $v'=(Q_1(b'_1),Q_2(b'_2),\cdots,Q_d(b'_d))$ 所表示的矩形 $V_B=(v,v')$ 就称为 B 的 VBR，从上面可以看出 V_B 可以用2d 个整数 $h_i(b_i)$、$h_i(b'_i)$(i=1,…,d)来表示，而 $h_i(b_i)\in\{0,1,\cdots,q-1\}$，因此我们可以用长度为$[\log_2 q]$的二进制串来表示每一个数字，同理 $h_i(b'_i)$也是如此。这样就大大减少了存储每一个项所占用的空间。

总结 从上面的介绍可以看出，在三十多年的时间里，人们对多维数据索引进行了大量的研究，提出了众多的索引结构和算法，它们各有优缺点，本文只是对这些索引结构进行概括性的介绍，而不去评价它们孰优孰劣。事实上，到目前为止多维数据索引结构仍未得到广泛的应用，这是因为它们都具有下面两个问题：

1)无法支持高维的数据。不管是哪种索引结构，随着维数的逐渐增大，性能都迅速下降，这种现象称为'维数灾难'。在对这些索引结构进行测试的时候，一般都选择低于64维的数据进行测试，对于更高维数的数据很难取得比较好的性能。

2)无法对空间中任意分布方式的数据都能取得比较好的性能。

正是因为如此，多维数据索引结构仍存在很多需要研究的问题，今后的工作主要集中在以下几个方面：1)能够支持对高维数据的索引；2)能够支持各种分布方式的数据；3)能够支持各种相似度计算方法的相似性检索；4)能够取得实际的应用。

参 考 文 献

1 Beckmann N, Kriegel H-P, Schneider R, Seeger B. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: Proc. ACM SIGMOD Intl. Conf. on Management of Data, 1990. 322～331

2 Bentley J L. Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, 1975, 18(9): 509～517

3 Berchtold S, Keim D, Kriegel H-P. The X-tree: An index structure for high-dimensional data. In: Proc. of the 22nd Intl. Conf. on Very Large DataBases, (Bombay) 1996. 28～39

4 Finkel R, Bentley J L. Quad trees: A data structure for retrieval of composite keys. Acta Inf. 4, 1, 1-9, 1974

5 Guttman A. R-tree: A dynamic index structure for spatial searching. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, 1984. 47～54

6 Hutflesz A, Siz H-W, Winmayer P. Twin grid files: Space optimizing access schemes. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, 1988. 183～190

7 Kamel I, Faloutsos C. On packing R-trees. In: Proc. of the 2nd Intl. Conf. on Information and Knowledge Management, 1993. 490～499

8 Lin K I, Jagadish H, Faloutsos C. The TV-tree: An index structure for high-dimensional data. VLDB J. 3, 4, 1994. 517～543

9 Nievergelt J, Hinterberger H, Sevcik K. The grid file: An adaptable, symmetric multikey file structure. In: Proc. of the Third ECI Conf. 1981. 236～251

10 Ooi B C, McDonell K J, Sacks-Davis R. Spatial k-d-tree: An indexing mechanism for spatial databases. In: Proc. of the IEEE Computer Sofware and Applications Conf. 1987. 433～438

11 Robinson J T. The K-D-B-tree: A search structure for large multidimensional dynamic indexes. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, 1981. 10～18

12 Sagan H. Space-Filling Curves. Springer-Verlag, Berlin/Heidelberg/New York, 1994

13 Sagan H. The quadtree and related hierachical data structure. ACM Comput. Surv., 1984, 16(2): 187～260

14 Gaede V, Gunther O. Multidimensional Access Method. ACM Computing Surveys, 1998, 30(2)

15 White D A, Jain R. Similarity indexing with the SS-tree. In: Proc. 12th Int Conf on Data Engineering, New Orleans, LA, 1996

16 Katayama N, Satoh S. The SR-tree: An index structure for high dimensional nearest neighbor queries. In: Int. Proc. of the ACM SIGMOD Int. Conf. on Management of Data, Tucson, Arizon USA, 1997. 369～380

17 Weber R, Schek H-J. Blott S. A Quantitative Analysis and Performance study for Similarity-search Methods in High-dimensional Spaces. In: Proc. of the 24th VLDB Conf. New York, 1998

18 Sakurai Y, et al. The A-tree: An index structure for high-dimensional spaces using relative approximation. In: Proc. of the 26th VLDB Conf. Cairo, Egypt, 2000

19 Tamminen M. The extendible cell method for closest point problems. BIT 22, 27～41

20 Hinrichs K. Implementation of the grid file: Design concepts and experience. BIT 25, 569～592

21 Bentley J L, Friedman J H. Data structures for range searching. ACM Comput. Surv., 1979, 11(4): 3997～4009

22 Tamminen M. The extendible cell method for closest point problems. BIT 22, 27-41

23 Samet H. The quadtree and related hierarchical data structure. ACM Comput. Surv., 1984, 16(2): 187～260

24 薛向阳，罗航哉，吴立德. LIFT：一种用于高维数据的索引结构. 电子学报，2001

25 薛向阳，罗航哉，吴立德. Index Point Data Using Algebraic Lattice. SPIE Electronic Imaging 2000: Storage and Retrieval for Media Database, 2000, 3972: 271～283

26 Chakrabarti K, Mehrotra S. The Hybrid Tree: An index structure for high-dimensional feature space. In: Proc. Int. Conf. on Data Engineering, Feb. 1999. 440～447