超酷算法：用四叉树和希尔伯特曲线做空间索引

转载　2015年10月26日 14:38:24　标签：四叉树 / 空间索引 / 希尔伯特曲线 / 算法 / 数据结构 /

1466

原文出处：http://blog.jobbole.com/81106/

越来越多的数据和应用和地理空间相关，空间索引变得愈加重要。然而，有效地查询地理空间数据是相当大的挑战，因为数据是二维的（有时候更高），不能用标准的索引技术来查询位置。空间索引通过各种各样的技术来解决这个问题。在这篇博文中，我将介绍几种：四叉树，geohash（不要和geohashing混淆）以及空间填充曲线，并揭示它们是怎样相互关联的。

四叉树

四叉树是种很直接的空间索引技术。在四叉树中，每个节点表示覆盖了部分进行索引的空间的边界框，根节点覆盖了整个区域。每个节点要么是叶节点，有包含一个或多个索引点的列表，没有孩子。要么是内部节点，有四个孩子，每个孩子对应将区域沿两根轴对半分得到的四个象限中的一个，四叉树也因此得名。

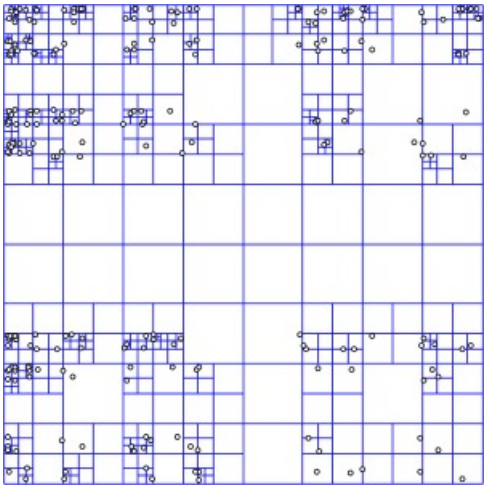

图1　展示四叉树是怎样划分索引区域的 来源：维基百科

将数据插入四叉树很简单：从根节点开始，判断你的数据点属于哪个象限。递归到相应的节点，重复步骤，直到到达叶节点，然后将该点加入节点的索引点列表中。如果列表中的元素个数超出了预设的最大数目，则将节点分裂，将其中的索引点移动到相应的子节点中去。

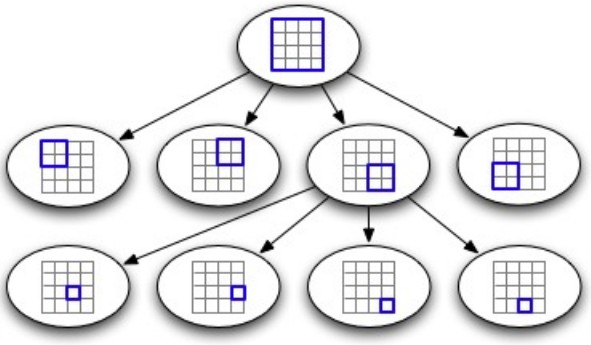

图2　四叉树的内部结构

查询四叉树时从根节点开始，检查每个子节点看是否与查询的区域相交。如果是，则递归进入该子节点。当到达叶节点时，检查点列表中的每一个项看是否与查询区域相交，如果是则返回此项。

注意四叉树是非常规则的，事实上它是一种字典树，因为树节点的值不依赖于插入的数据。因此我们可以用直接的方式给节点编号：用二进制给每个象限编号（左上是00，右上是10等等 译者注：第一个比特位为0表示在左半平面，为1在右半平面。第二个比特位为0表示在上半平面，为1在下半平面），任一节点的编号是由从根开始，它的各祖先的象限号码串接而成的。在这个编号系统中，图2中右下角节点的编号是1101。

我们定义了树的最大深度，不需通过树就可以计算数据点所在节点的编号：只要把节点的坐标标准化到适当的整数区间中（比如32位整数），然后把转化后x, y坐标的比特位交错组合。每对比特指定了假想的四叉树中的一个。（译者注：不了解的读者可看看Z-order，它和下文的希尔伯特曲线都是将二维的点映射到一维的方法）

eohash

上述编号系统可能看起来有些熟悉，没错，就是geohash！此刻，你可以把四叉树扔掉了。节点编号，或者说geohash，包含了对于节点在树中位置我们需要的全部信息。全高树中的每个叶节点是个完整的geohash，每个内部节点代表从它最小的叶节点到最大的叶节点的区间。因此，通过查询所需的节点覆盖的数值区间中的一切（在geohash上索引），你可以有效地定位任意内部节点下的所有数据点。

一旦我们丢掉了四叉树，查询就变得复杂一点了。我们需要事先构建搜索集合而不是在树中递归地精炼搜索集合。首先，找到完全覆盖查询区域的最小前缀（或者说四叉树节点 译者注：注意在我们的编号系统中节点由比特串表示）。在最坏情况下，这可能远大于实际的查询区域，比如对于在索引区域中心、和四个象限都相交的小块地方，查询将要从根节点开始。

现在的目标是构建一组完全包含查询区域的前缀，并且尽可能少包含区域外的部分。如果没有其他约束，我们可以简单地选择与查询区域相交的叶节点，但这会造成大量的查询。所以要加一个约束：使得要查询的不同区间最少。

一种达到这个目的的方法是先设置我们愿意承受的查询区间的最大数目。构建一组区间，最开始都设为我们之前指定的前缀。从中选择可以再分裂而不超出最大区间数并将从查询区域删除最不受欢迎区域的节点。重复这个过程直到集合中再没有区间可以细分。最后，检查得到的集合，如果可能的话合并相邻的区间。下面的图说明了这对于查询一个圆形区域且限制最大5个查询区间是如何工作的。

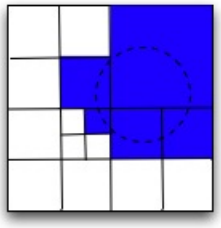

图3　一个对区域的查询是怎样分解成一连串geohash前缀/区间的

这个方法工作地很好，它使我们避免了递归查找。我们执行的一整套区间查找都可以并行完成。由于每次查找都预期要一次硬盘搜索，将查询并行化大大减少了返回结果需要的时间。

然而，我们还可以做得更好。你可能注意到上图中我们要查询的所有区域都是相邻的，但我们却只能将其中两个合并（选择区域的右下角的两个）成一个单独的查询，进而只要4次单独查询。（译者注：这两个区域可以合并是因为它们在geohash以Z字形遍历区域的路径上是相邻的）这个后果部分是由于geohash访问子区域的顺序，在每个象限中从左到右，从上到下。从右上角象限到左下角象限的不连续性使得我们不得不将本可以使之连续的区间分裂。如果以不同的顺序访问区域，可能我们就可以最小化或者消除这些不连续性，使得更多的区域可以被看做是相

邻的，一次查询就可得到结果。通过这样效率上的提升，对于同样的覆盖区域，我们可以做更少的查询，或者相反地，同样的查询次数的情况下包含更少的无关区域。

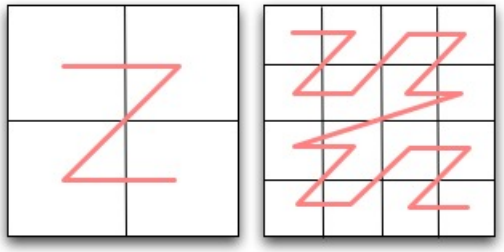

图4　geohash访问象限的顺序

尔伯特曲线

现在假设我们以U字形来访问区域。在每个象限中，我们同样以U字形来访问子象限，但是要调整好U字形的朝向使得和相邻的象限衔接起来。如果我们正确地组织了这些U字形的朝向，我们就能完全消除不连续性，不管我们选择了什么分辨率，都能连续地访问整个区域，可以在完全地探访了一个区域后才移动到下一个。这个方案不仅消除了不连续性，而且提高了总体的局域性。按照这个方案得到的图案看起来有些熟悉，没错，就是希尔伯特曲线。

希尔伯特曲线属于一类被称为空间填充曲线的一维分形，因为它们虽然是一维的线，却可以填充固定区域的所有空间。它们相当有名，部分是由于XKCD把它们用于互联网地图。如你所见，对于空间索引它们也是有用的，因为它们展现的正是我们需要的局域性和连续性。再看看之前用一组查询来覆盖圆的例子，我们发现（应用希尔伯特曲线）还可以减少一次查询：左下方的小区域现在和它右边的区域连起来了（减少一次），虽然底部的两块区域不再连续了（增加一次），右下角的区域现在却和它上方的连续了（减少一次）。

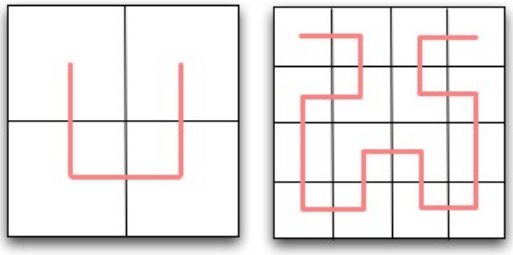

图5　希尔伯特曲线访问象限的顺序

到目前为止，我们优雅的系统还缺一样东西：将(x,y)坐标转换为希尔伯特曲线上相应位置的方法。对于geohash，这是简单而明显的–只需将x, y坐标交错，但没有明显的方法修改这个方案使之对希尔伯特曲线也适用。在网上搜索，你很可能遇到很多关于希尔伯特曲线是怎样画出来的描述，但很少有关于找到任意点（在曲线上）位置的。为了搞定它，我们需要更仔细看看希尔伯特曲线是怎么递归构建的。

首先要注意到虽然大多数关于希尔伯特曲线的文献都关注曲线是怎么画出来的，却容易让我们忽略曲线的本质属性以及其重要性：曲线规定了平面上点的顺序。如果我们用这顺序来表达希尔伯特曲线，画曲线就不值一提了：仅仅是把点连起来。忘记怎么把子曲线连起来吧，把注意力集中在怎么递归地列举点上。

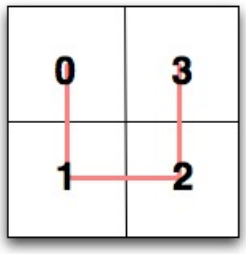

图6　希尔伯特曲线规定了二维平面上的点的顺序

在根这一层，列举点很简单：选定一个方向和一个起始点，环绕四个象限，用0到3给他们编号。当我们要确定访问象限的顺序同时维护总体的邻接属性，困难就来了。通过检查我们发现，子象限的曲线是原曲线的简单变换，而且只有四种变换。自然地，这个结论也适用于子子象限，等等。对于一个给定的象限，我们在其中画出的曲线是由象限所在大的方形的曲线以及该象限的位置决定的。只需要费一点力，我们就能构建出如下概况所有情况的表。

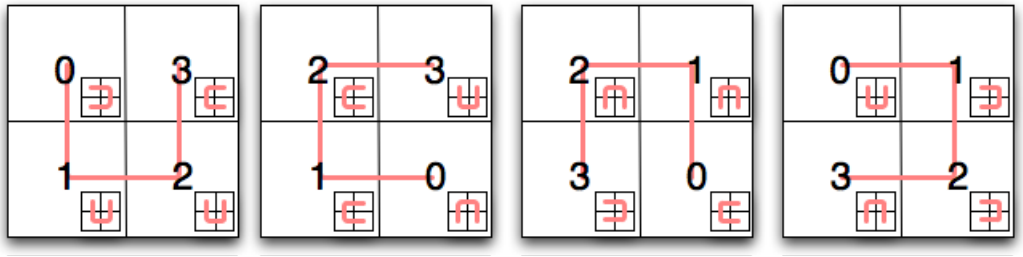

图7

假设我们想用这个表来确定某个点在第三层希尔伯特曲线上的位置。在这个例子中，假设点的坐标是(5,2)。（译者注：请参照图8）从上图的第一个方形开始，找到你的点所在的象限。在这个例子中，是在右上方的象限。那么点在希尔伯特曲线上的位置的第一部分是3（二进制是11）。接着我们进入象限3里面的方块，在这个例子中，它是（图7中的）第二个方块。重复刚才的过程：我们的点落在哪个子象限？这次是左下角，意味着位置的下一部分是1（二进制01），我们将进入的小方块又是第二个。最后一次重复这个过程，发现点落在右上角的子子象限，因此位置的最后部分是3（二进制11）。把这些位置连接起来，我们得到点在曲线上的位置是二进制的110111，或者十进制的55。

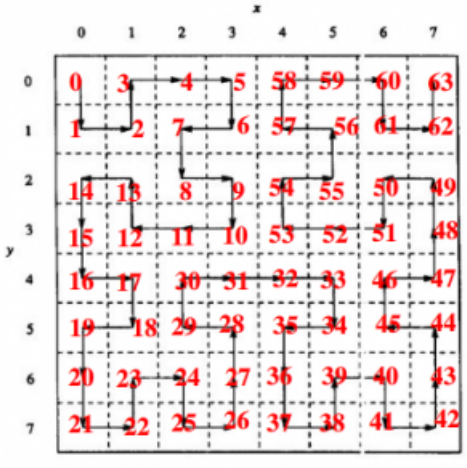

图8　三阶希尔伯特曲线

让我们更系统一些，写出从x, y坐标到希尔伯特曲线位置转换的方法。首先，我们要以计算机看得懂的形式表达图7：

```
hilbert_map
=
{
'a':
 {(0,
0):
 (0,
'd'),
 (0,
1):
 (1,
'a'),
 (1,
0):
 (3,
'b'),
 (1,
1):
 (2,
'a')},
'b':
 {(0,
0):
 (2,
'b'),
 (0,
1):
 (1,
'b'),
 (1,
0):
 (3,
'a'),
 (1,
1):
 (0,
'c')},
'c':
 {(0,
0):
 (2,
'c'),
 (0,
1):
 (3,
'd'),
 (1,
0):
 (1,
'c'),
 (1,
1):
 (0,
'b')},
'd':
 {(0,
0):
 (0,
'a'),
 (0,
1):
 (3,
'c'),
 (1,
0):
 (1,
'd'),
 (1,
1):
 (2,
'd')}}
```

上面的代码中，每个hilbert_map的元素对应图7四个方形中的一个。为了容易区分，我用一个字母来标识每个方块：’ a'是第一个方块，’ b'是第二个，等等。每个方块的值是个字典，将(子)象限的x, y坐标映射到曲线上的位置（元组值的第一部分）以及下一个用到的方块（元组值的第二部分）。下面的代码展示了怎么用这个来将x, y坐标转换成希尔伯特曲线上的位置：

```
def
point_to_hilbert(x, y, order=16):
```

```
    current_square = 'a'
    position = 0
    for i in range(order - 1, -1, -1):
        position <<= 2
        quad_x = 1 if x & (1 << i) else 0
        quad_y = 1 if y & (1 << i) else 0
        quad_position, current_square = hilbert_map[current_square][(quad_x, quad_y)]
        position |= quad_position
    return position
```

函数的输入是为整数的x, y坐标和曲线的阶。一阶曲线填充2×2的格子，二阶曲线填充4×4的格子，等等。我们的x, y坐标应该先标准化到0到2^{order}-1的区间。这个函数从最高位开始，逐步处理x, y坐标的每个比特位。在每个阶段中，通过测试对应的比特位，可以确定坐标处于哪个（子）象限，还可以从我们之前定义的hilbert_map中取得在曲线上的位置以及下一个要用的方块。在这阶段取得的位置，加入到目前总的位置的最低两位。在下一次循环的开头，总的位置左移两位以便给下一个位置腾出地方。

让我们运行一下之前的例子来检验一下函数写对了没有：

```
>>> point_to_hilbert(5,2,3)
55
```

对了！为了进一步测试，我们可以用这个函数生成一条希尔伯特曲线的有序点的完整列表，然后用电子制表软件把它们画出来看我们是否真的得到了一条希尔伯特曲线。在Python交互解释器中输入如下代码：

```
>>> points = [(x, y) for x in range(8) for y in range(8)]
>>> sorted_points = sorted(points, key=lambda k: point_to_hilbert(k[0], k[1], 3))
>>> print 'n'.join('%s,%s' % x for x in sorted_points)
```

将输出的文本粘贴到文件中，保存为hilbert.csv，用你最喜欢的电子制表软件打开，将数据画成一个散点图。结果当然是一条漂亮的希尔伯特曲线！

将hilbert_map做简单的反转就能实现point_to_hilbert的逆向功能（将希尔伯特曲线上的位置转换为x, y坐标），把这个留给读者作为练习吧。

场景管理：四叉树算法C++实现

简单实现了游戏中场景管理用到的四叉树算法 代码结构： object.h，object.cpp被管理的对象类quad_tree_node.h，quad_tree_node.cpp四叉树类ma...


u012234115 2015年07月30日 21:04 2622

ROAM地形渲染算法的核心

M地形渲染算法的核心是： 1.二叉三角面树的构造。这个构造过程即细化分割三角面的过程(Tessellation)。具体的讲，我们将自己的地形分成若干个菱形(Diamond),每个Diamond由...


pizi0475 2011年04月12日 13:35 2376

超酷算法：用四叉树和希尔伯特曲线做空间索引

转自http://blog.jobbole.com/81106/ 随着越来越多的数据和应用和地理空间相关，空间索引变得愈加重要。然而，有效地查询地理空间数据是相当大的挑战，因为数据是二维的（...


dhtx_wzgl 2015年11月14日 15:37 366

基于卷积定理和希尔伯特变换的模拟调制分析

2012年02月13日 20:06 129KB 下载

使用java绘制希尔伯特曲线（hilbert curve）

绘制希尔伯特曲线 Hilbert Curve


Half_open 2016年10月26日 11:58 1359

python 画希尔伯特曲线

给你一个问题： 让你在一个N×NN\times N的点阵，让你画一条连续曲线，使得这条曲线经过这个点阵中的每个点，并且每个点只经过一次，N满足条件：N=2k，k⊂ZN=2^k， k\subset \...


WhoisPo 2015年11月04日 21:52 2087

希尔伯特空间

2016年12月08日 13:41 1.12MB 下载

L系统_重写系统_希尔伯特曲线_谢尔宾斯基三角形_科赫曲线

2016年08月08日 12:24 141KB 下载

再生核希尔伯特空间（RKHS）和核函数

之前看SVM核函数相关的问题，总是会碰到再生核希尔伯特空间（Reproducing Kernel Hilbert Space, RKHS）不过一直没有太仔细了解过到底是指的什么，前几天研究了一下。 希...

haolexiao 2017年05月15日 17:48 1678

图像处理中的数学原理详解23——详解希尔伯特空间

0

有一段时间没继续更新我的“图像处理中的数学原理详解”专栏了。因为前面基础的部分已经发布的差不多了，现在已经进入“深水区”。一方面现在文章的长度都有所增加，所以我写起来就更加麻烦了。另一方面，现在的话题...

aimafujinji 2016年01月16日 16:38 8928