|  |
| --- |
| **k-d 树** |
|  |
| K-D树，K表示空间的维数。 它的每一层通过检测不同的属性（关键字）值以决定选择分枝的方向。在二维空间中（也就是2-D树）在根和偶数层比较X坐标值（假设根的深度为0），在奇数层比较Y坐标值。  每个数据点用K-D树中一个结点来表示，每个记录是通过结点中的6个域表现出来。开始的两个域是指向结点两个孩子的指针，各自相对应方向是左和右。XCOORD和YCOORD各自保存数点X和Y的坐标值。NAME域用来保存结点描述信息（例如城市名）。DISC域表示结点识别的坐标名（也就是比较坐标名）。  当结点P是一个x识别器。那么所有具有x坐标值小于P的结点将放在左树中，而x坐标值大于或等于P的结点将放到P的右子树中。对于一个Y识别器的结点有同样的约定。实际上DISC域并不是必须的，因为随着树的下降，很容易跟踪被访问的结点类型。表2.18给出了一个2-D树它对应于表2.1中的相同的8个结点。  http://202.114.200.211:7310/cug/space_data_struct/zhishixuexi/content/new/content/tupian/2.181.jpg  图2.18 一个k-d树及其记录的表示  在识别器的定义中，关键字相等的问题的解决是靠规定相等的关键字在右子树中（也就是HISON）实现的。另一种方法是 通过超关键字定义一个结点。给定一个结点P，用K0（P），K1（P）等指代它的K个关键字。假设j的值代表DISC（P）那么任何满足条件Kj（Q）〈Kj（P）的结点Q在P的左子树中，同样任何满足条件Kj（R）>Kj(P)的结点R在P的右子树中。在相等的情况下，定义一个超关键字,它是通过一个从Kj(P)开始所有关键字循环的相关值形成的.换言之Sj(P)=Kj(P)Kj+1(P)…Kk-1(P)K0(P)…Kj-1(P) ，现在比较P和Q的2个关键字,当Sj(Q)<Sj(P)转向左子树,当Sj(Q)>Sj（P）则转向右子树，如果Sj（Q）=Sj（P）,那么所有K个关键字相等则返回一个特殊的值。  这里定义K-D树的子分割线是垂直于分割线的。在BSP树中，每个结点靠它的线性半空间等式来表示，在二维空间中它是一条直线，在三维空间中它是一个平面。同样在二维空间中每个结点块是个凸多边形。而在三维空间中它是一个凸多面体。 |