## KDTree

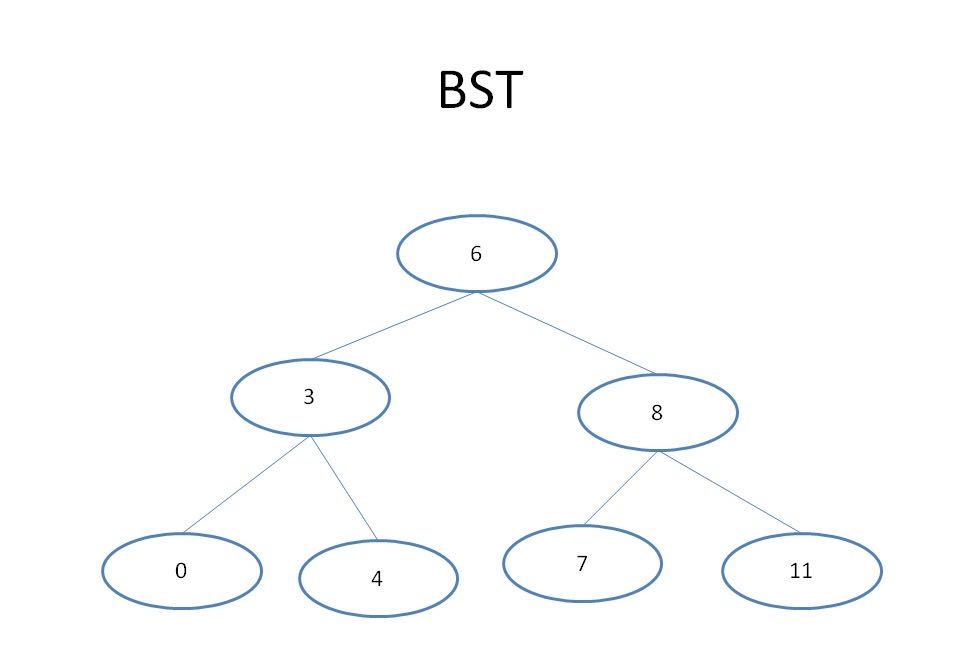
### 1.1二分查找树（BST）

假设数组A为[0, 6, 3, 8, 7, 4, 11]，有一个元素x，要找到数组A中距离x最近的元素

#### 1.1.1步骤

1. 我们把数组A进行升序排列，得到[0, 3, 4, 6, 7, 8, 11]；
2. 令x = 2，数组中间的元素是6，2小于6，所以2只可能存在于6的左边，我们只需要在数组[0, 3, 4]中继续查找；
3. 左边的数组中间的元素是3，2小于3，所以2只可能存在于3的左边，即数组[0]；
4. 由于数组[0]无法再分割，查找结束；
5. x需要跟我们最终找到的0，以及倒数第二步找到的3进行比较，发现2离3更近，所以查找结果为3。

#### 1.1.2图片展示



### 1.2KDTree 解释

是一种分割k维数据空间的数据结构，能快速搜索最近点。

### 1.3 建立KD-Tree步骤

1. 建立根节点；

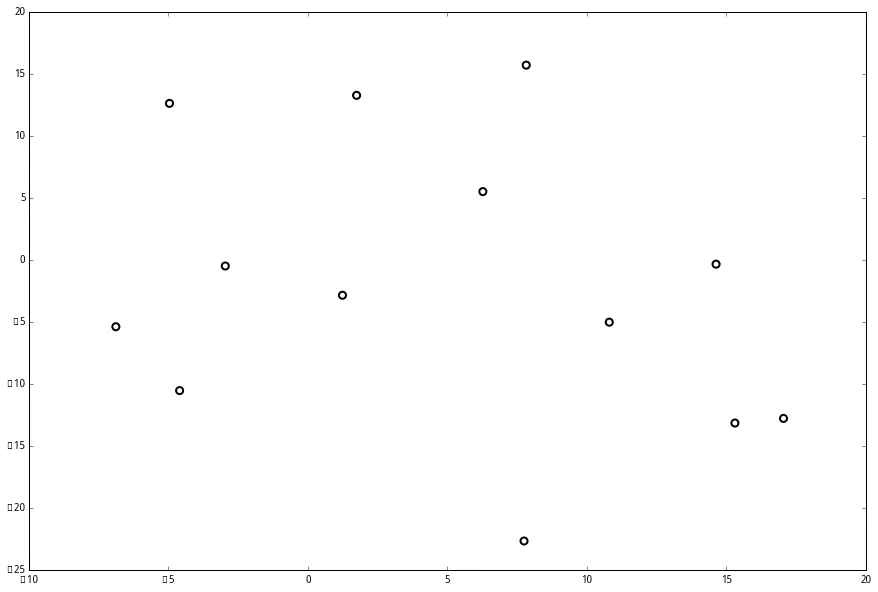
2. 选取方差最大的特征作为分割特征；（方差大，数据相对“分散”，选取该特征来对数据集进行分割，数据散得更“开”一些）

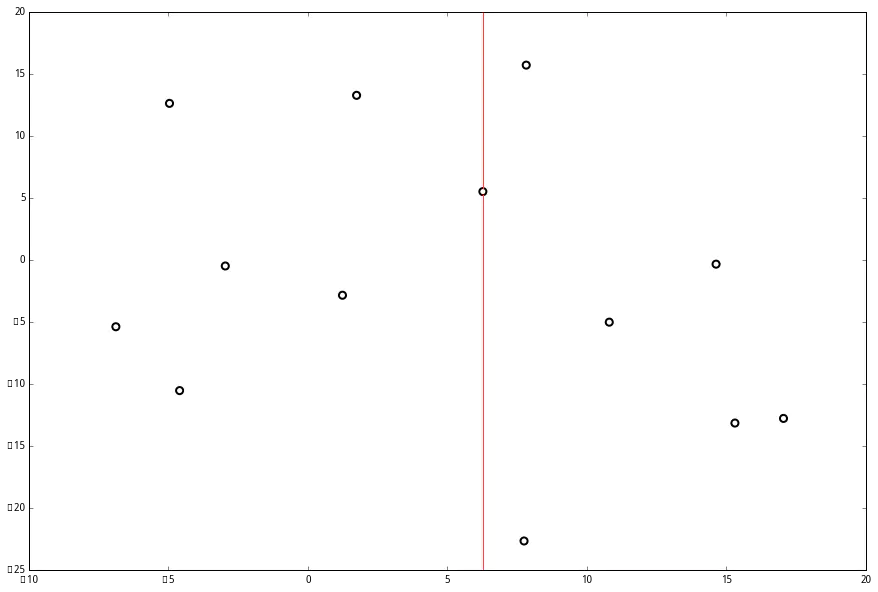
3. 选择该特征的中位数作为分割点；（选取中位数，让左子树和右子树的数据数量一致，便于二分查找）

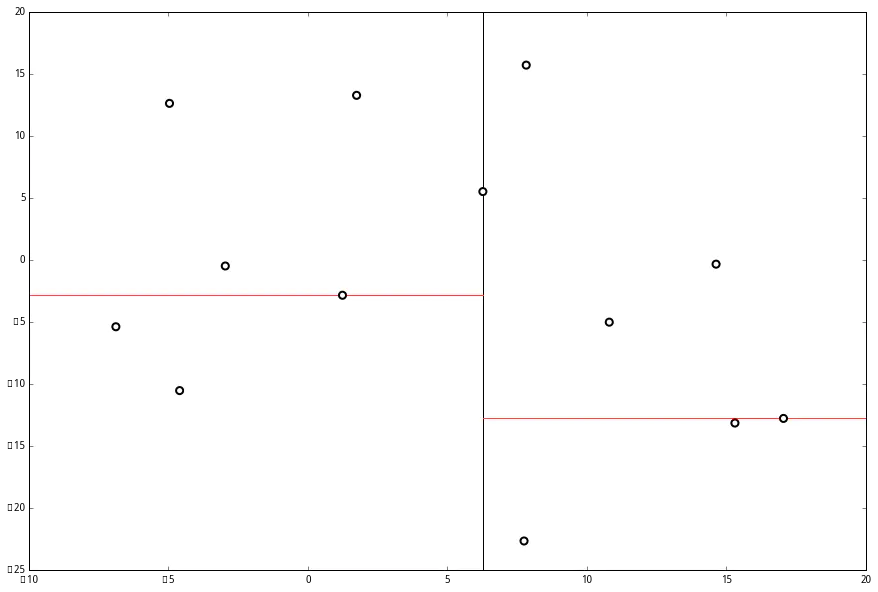
4. 将数据集中该特征小于中位数的传递给根节点的左儿子，大于中位数的传递给根节点的右儿子；

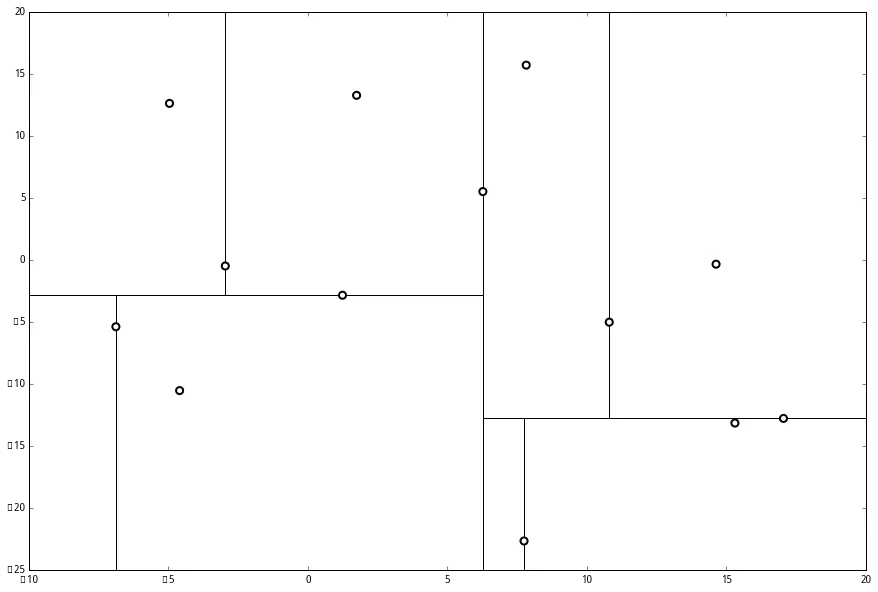
5. 递归执行步骤2-4，直到所有数据都被建立到KDTree的节点上为止。

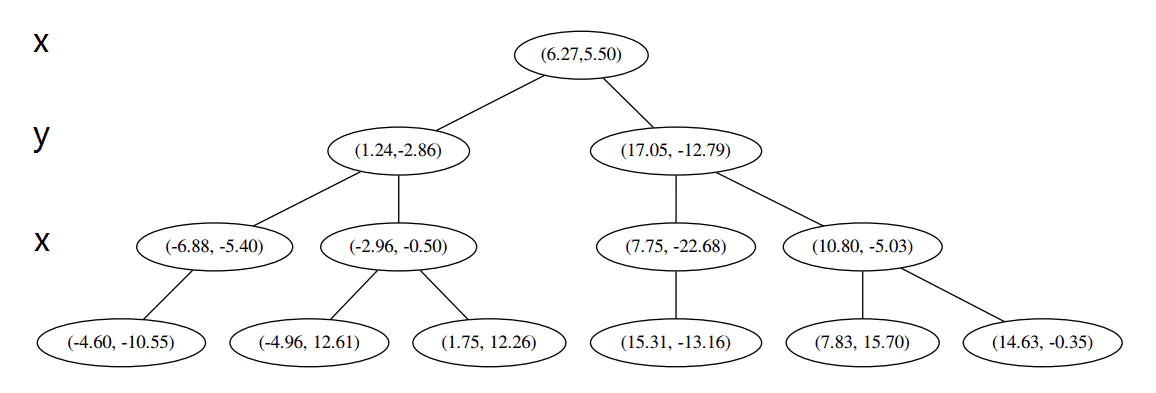
### 1.4图片展示











### 1.5 利用KD-Tree查找步骤

查找的方式在BST的基础上又增加了一些难度，如下：

1. 从根节点开始，根据目标在分割特征中是否小于或大于当前节点，向左或向右移动。

2. 一旦算法到达叶节点，它就将节点点保存为“当前最佳”。

3. 回溯，即从叶节点再返回到父节点

4. 如果当前节点比当前最佳节点更接近，那么它就成为当前最好的。

5.如果目标距离当前节点的父节点所在的将数据集分割为两份的超平面的距离更接近，说明当前节点的兄弟节点所在的子树有可能包含更近的点。因此需要对这个兄弟节点递归执行1-4步。

### 补：超平面

以[0, 2, 0], [1, 4, 3], [2, 6, 1]的举例：

1. 如果用第二维特征作为分割特征，那么从三个数据点中的对应特征取出2, 4, 6，中位数是4；

2. 所以[1, 4, 3]作为分割点，将[0, 2, 0]划分到左边，[2, 6, 1]划分到右边；

3. 从立体几何的角度考虑，三维空间得用一个二维的平面才能把空间一分为二，这个平面可以用y = 4来表示；

4. 点[0, 2, 0]到超平面y = 4的距离就是 sqrt((2 - 4) ^ 2) = 2；

5. 点[2, 6, 1]到超平面y = 4的距离就是 sqrt((6 - 4) ^ 2) = 2。