Bkd-Tree

Bkd-Tree作为一种基于K-D-B-tree的索引结构，用来对多维度的点数据(multi-dimensional point data)集进行索引。Bkd-Tree跟K-D-B-tree的理论部分在本篇文章中不详细介绍，对应的两篇论文在附件中，感兴趣的朋友可以自行下载阅读。本篇文章中主要介绍Bkd-Tree在Lucene中的实现，即生成树的过程。

预备知识

如果只是想了解Bkd-Tree生成过程，那么这节内容可以跳过，这块内容是为介绍索引文件.dim、.dii作准备的。

点数据

点数据(Point Data)，源码中又称为点值(Point Value)，它是由多个数值类型组成。 图1：

```
doc.add(new IntPoint( name: "content",  ...point: 3, 5, 99, 12));
```

上图中由4个int类型数值组成一个点数据/点值，并且根据点数据中的数值个数定义了维度个数。上图中即有四个维度。同一个域名的点数据必须有相同的维度个数，并且在当前7.5.0版本中，维度个数最多为8个。

int numPoints

numPoints是一个从0开始递增的值，可以理解为是每一个点数据的一个唯一编号，并且通过这个编号能映射出该点数据属于哪一个文档(document)。映射关系则是通过docIDs[ ]数组实现。

int docIDs[ ]数组

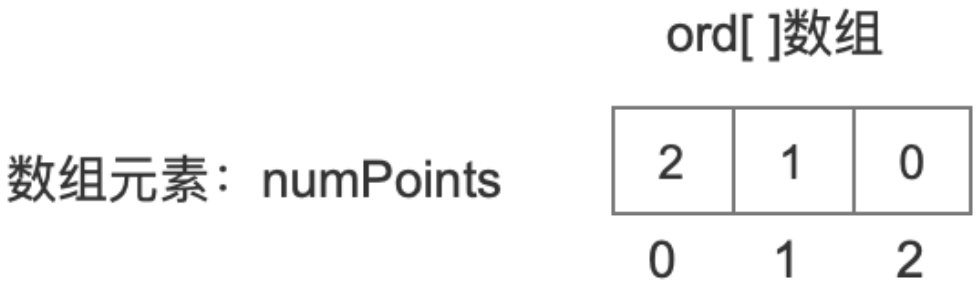
docIDs[ ]数组在PointValuesWriter.java中定义，数组下标是点数据的编号numPoint，数组元素是点数据所属的文档号。由于一篇文档中可以有多个点数据，所以相同的数组元素对应的多个数组下标值，即numPoints，即点数据，都是属于同一个文档。 图2：

```
Document doc;
// 文档号 = 0
doc = new Document();
doc.add(new IntPoint( name: "content",  ...point: 3, 5, 99, 12)); // numPoints = 0
indexWriter.addDocument(doc);
// 文档号 = 1
doc = new Document();
doc.add(new IntPoint( name: "content",  ...point: 1, 5, 23, 12)); // numPoints = 1
doc.add(new IntPoint( name: "content",  ...point: 3, 6, 12, 33)); // numPoints = 2
indexWriter.addDocument(doc);
indexWriter.commit();
```

上图中只添加了2篇文档，处理顺序按照文档号的顺序，所以文档0的点数据的numPoints的值为0，另外一篇文档可以有多个点数，所以numPoints的值分别为1、2。生成的docIDs[]数组如下： 图3：

int ord[ ]数组

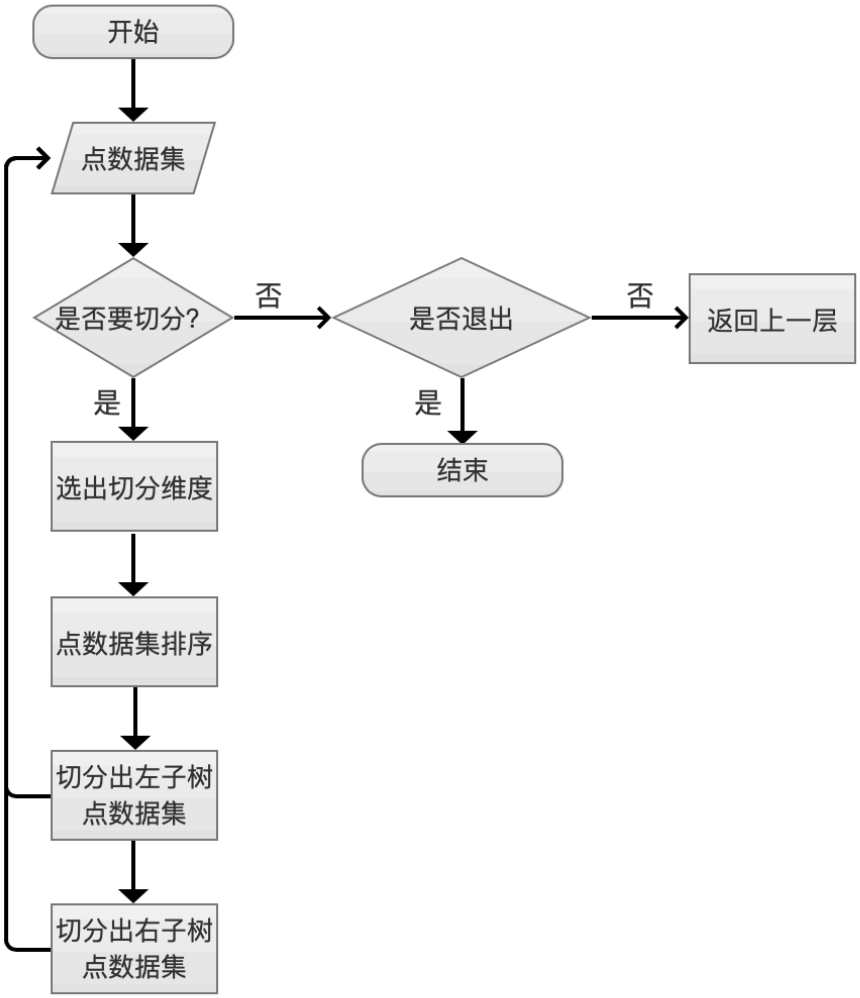
ord数组的数组元素为numPoints，下面的一句话很重要：ord数组中的元素是有序的，排序规则**不是**按照numPoints的值，而是按照numPoints对应的点数据的值。这里ord数组的用法跟SortedDocValues中的sortedValues[]数组是一样的用法。例如根据图2中的点数据，如果我们按照第三个维度的值，即"99"、"23"、"12"来描述点数据的大小关系，那么ord数组如下图所示： 图4：



这里先提一句，在生成BKD-Tree之后，叶子节点中的点数据会根据某个维度进行排序的，并且所有叶子节点中的点数据的大小关系就存放在ord[]数组中，后面的内容会详细介绍这过程。

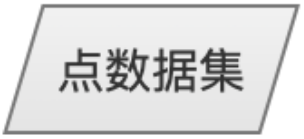
流程图

一句话概括整个流程的话就是：根据某一个维度将点数据集划分为两部分，递归式将两部分的点数据子集进行划分，最终生成一个**满二叉树**。 图5：



点数据集

图6：



点数据集即为待处理的点数据集合。

是否要切分?

图7：

如果数据集的个数大于1024个，那么需要进行拆分。在源码中并不是通过判断数据集的个数，而是在建立Bkd-Tree之前就预先计算出当前数据会对应生成节点(node)的个数（可以认为每个节点中的数据都是空的），然后采用深度遍历方式处理每一个节点，通过节点编号来判断是否为叶子节点。如果不是叶子节点，说明要切分(节点赋值)。

选出切分维度

图8：

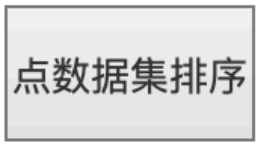
一个点数据中有多个维度，例如图1中就有四个维度。

1. 先计算出切分次数最多的那个维度，切分次数记为maxNumSplits，如果有一个维度的切分次数小于 (maxNumSplits / 2) ，并且该维度中的最大跟最小值不相同，那么令该维度为切分维度。
2. 计算出每一个维度中最大值跟最小值的差值，差值最大的作为切分维度(篇幅原因，下面的例子中仅使用了这种判定方式)。

条件1优先条件2。

点数据集排序

图9：



当确定了切分维度后，我们对当前节点中的点数据集进行排序，排序规则根据该每个点数据中的该维度的值，排序算法使用最大有效位的基数排序(MSB radix sort)。

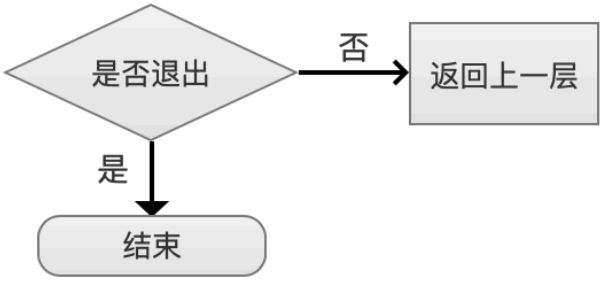
切分出左子树点数据集、切分出右子树点数据集

图10：

执行完排序操作后，当前节点中的点数据集数量为N，那么将前 (N / 2)个的点数据集划分为左子树，剩余的划分为右子树。 **这么划分的目的使得无论初始的点数据集是哪种数据分布，总是能生成一颗满二叉树。**

是否退出

图11：



当前节点不需要切分，需要判断下算法是否需要退出。

结束

- 当前节点是满二叉树的最右子树，那么算法结束，可以退出。
- 当前树中只有一个节点，且该节点不需要切分，那么算法结束，可以退出。

返回上一层

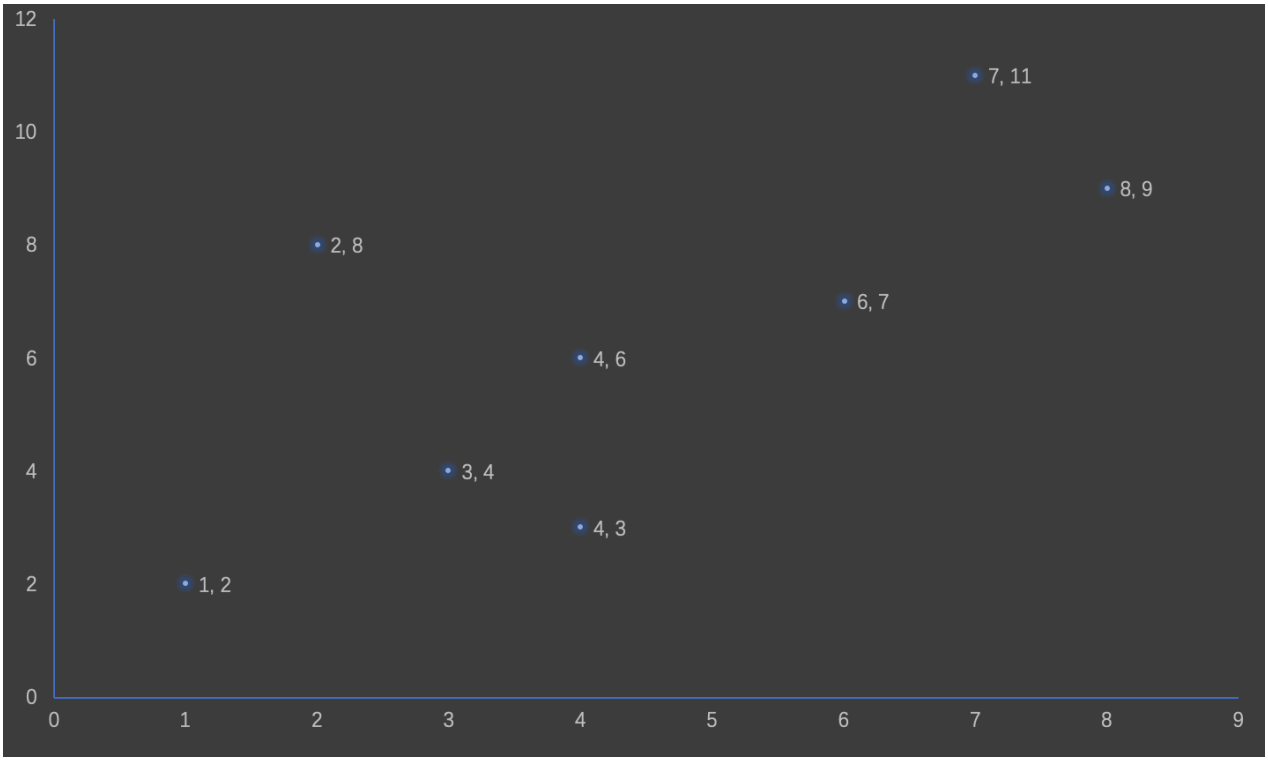
- 当前处理的节点是**左子树节点**或者是**非最右子树节点**，说明该节点是由 划分左右子树生成的，即算法还处在递归中，当不需要划分后，返回到递归的上一层。

例子

Lucene 7.5.0版本源码中当一个节点中的点数据个数大于1024才会进行切分，为了能简单示例，例子中假设一个节点中的点数据个数大于2个才会进行切分，并且点数据的维度为2。

点数据集

图12：



上图中一共有8个点数据，每个点数据有两个维度。为了描述方便，下面统称为x维度，跟y维度。

处理节点1

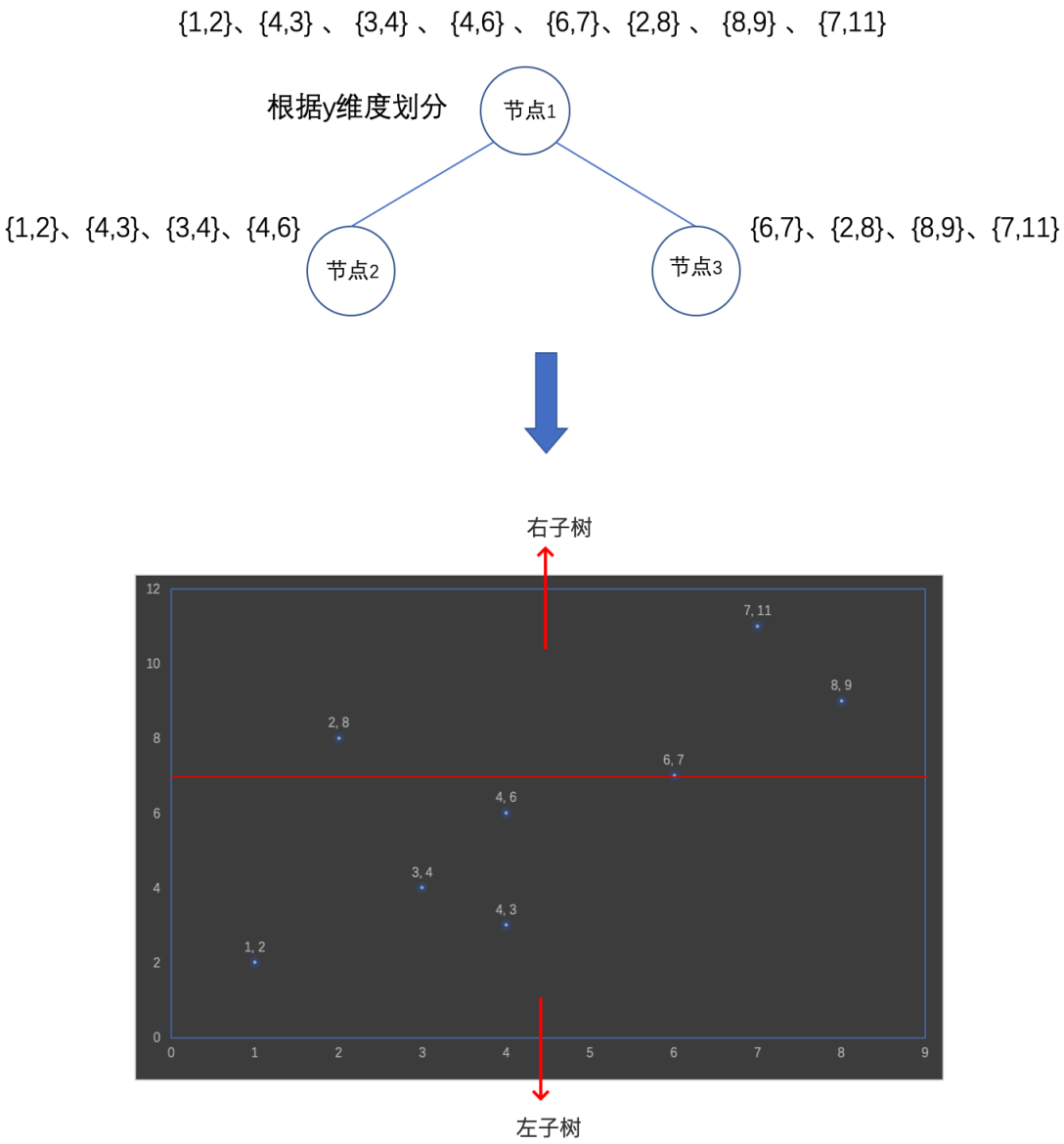
- 是否要切分：初始的数据集作为第一个节点，即节点1开始进行切分，该节点中有8个数据，大于节点切分的条件值2，所以需要切分。
- 选出切分维度：x维度的最大值跟最小值的差值为7 ，而y维度的最大值跟最小值的差值为9，所以当前节点的切分维度为y维度。
- 点数据排序：对8个点数据按照y维度的值进行排序，排序后的结果如下:

```
{1,2} -> {4,3} -> {3,4} -> {4,6} -> {6,7} -> {2,8} -> {8,9} -> {7,11}
```

- 切分出左子树数据集、切分出右子树数据集：当前节点个数为8，从排序后的点数据中取前一半的点数据划为左子树(节点2)，剩余的划为右子树(节点3)。

```
左子树：{1,2}、{4,3}、{3,4}、{4,6}
右子树：{6,7}、{2,8}、{8,9}、{7,11}
```

图13：



处理节点2

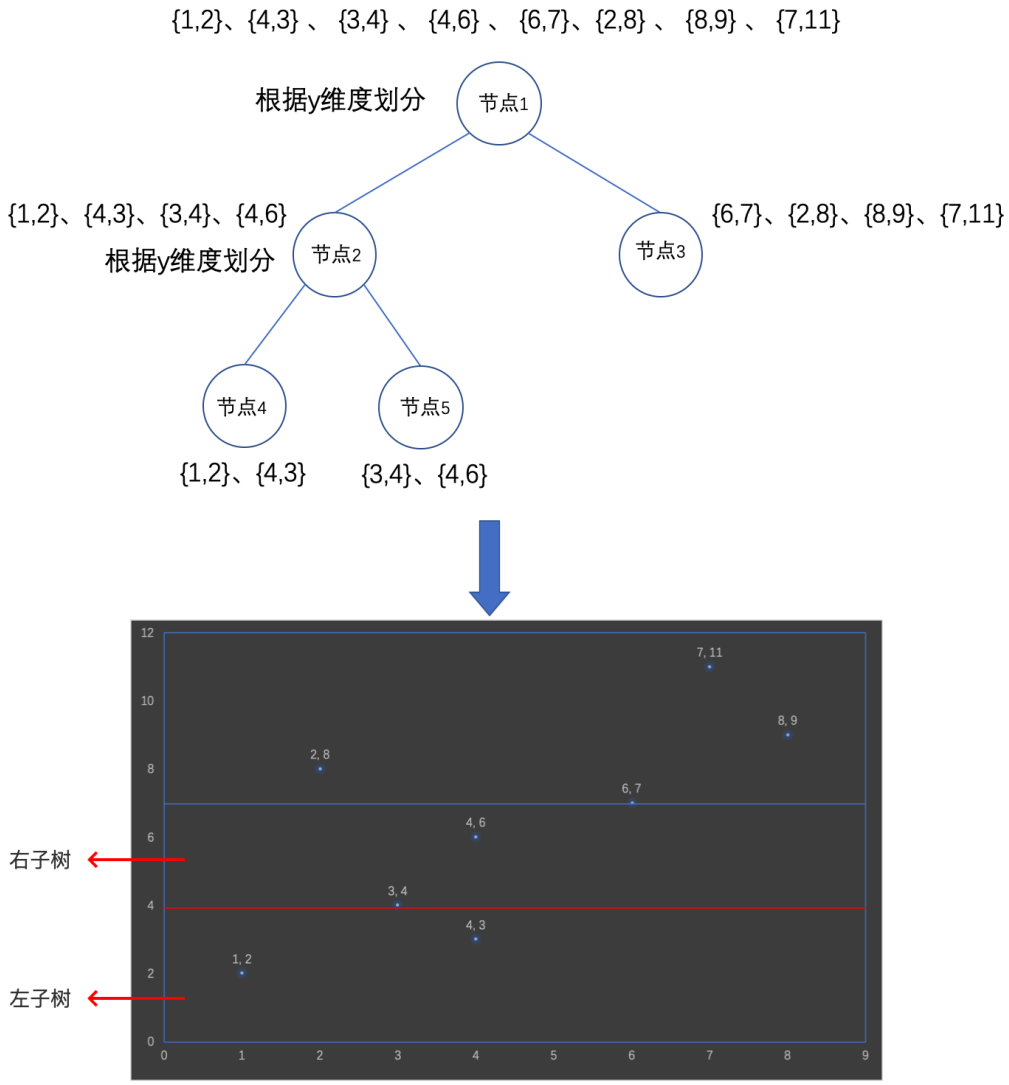
- 是否要切分：节点2中有4个数据，大于节点切分的条件值2，所以需要切分。
- 选出切分维度：x维度的最大值跟最小值的差值为3 ， 而y维度的最大值跟最小值的差值为4，所以当前节点的切分维度为y维度。
- 点数据排序：对4个点数据按照y维度的值进行排序，排序后的结果如下:

```
{1,2}、{4,3}、{3,4}、{4,6}
```

- 切分出左子树数据集、切分出右子树数据集：当前节点个数为4，从排序后的点数据中取前一半的点数据划为左子树(节点4)，剩余的划为右子树(节点5)。

```
左子树：{1,2}、{4,3}
右子树：{3,4}、{4,6}
```

图14：



处理节点4、5

源码中对叶子结点还有一些处理，目的是为了生成索引文件作准备，在随后的介绍索引文件.dii、.dim时候会介绍跟叶子节点相关的知识，这篇文章主要介绍生成Bkd-Tree的过程。

处理节点3

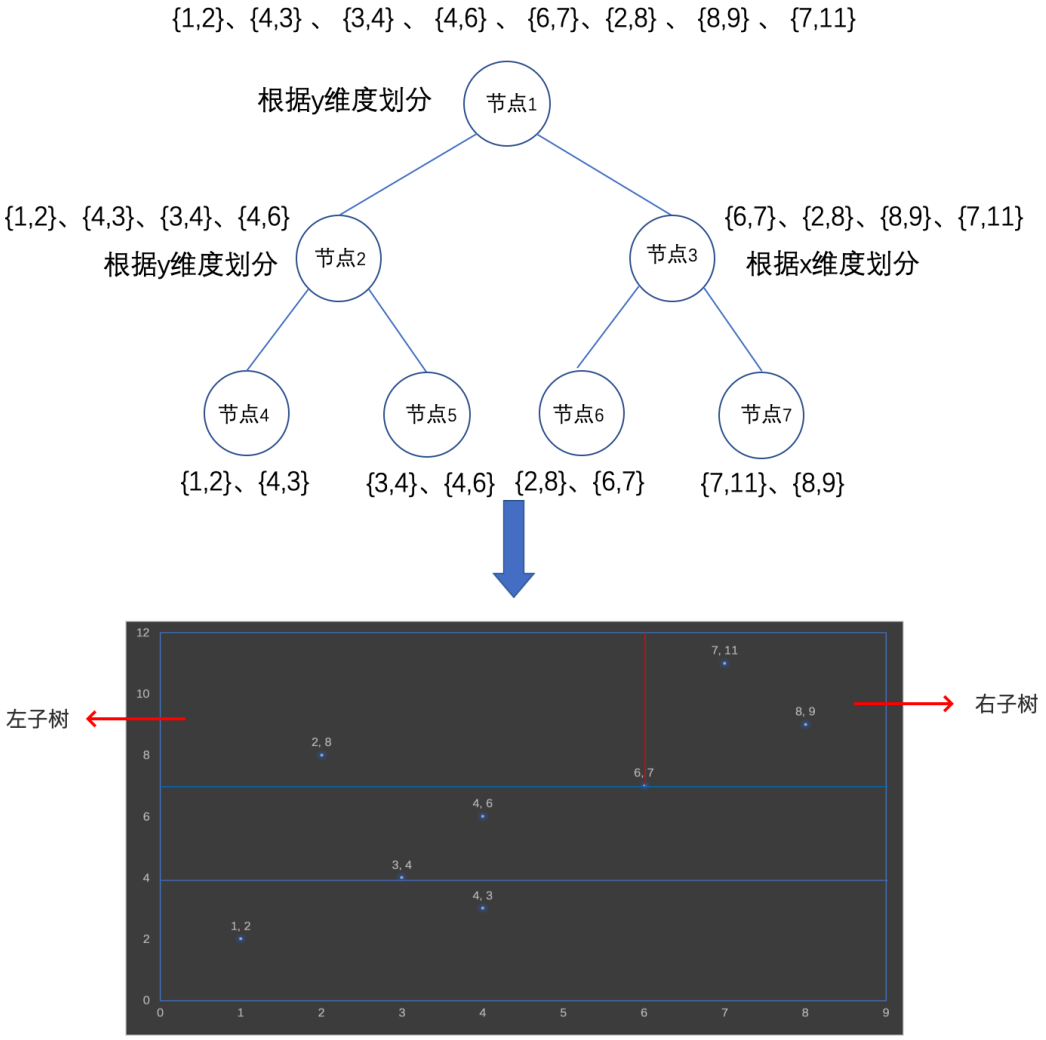
- 是否要切分：节点3中有4个数据，大于节点切分的条件值2，所以需要切分。
- 选出切分维度：x维度的最大值跟最小值的差值为6 ， 而y维度的最大值跟最小值的差值为4，所以当前节点的切分维度为x维度。
- 点数据排序：对4个点数据按照x维度的值进行排序，排序后的结果如下:

```
{2,8}、{6,7}、{7,11}、{8,9}
```

- 切分出左子树数据集、切分出右子树数据集：当前节点个数为4，从排序后的点数据中取前一半的点数据划为左子树(节点6)，剩余的划为右子树(节点7)。

```
左子树：{2,8}、{6,7}
右子树：{7,11}、{8,9}
```

图15：



处理节点6、7

同节点4、5

结语

本篇文件介绍了Bkd-Tree在Lucene中的实现，即生成满二叉树的过程，再以后介绍索引文件.dii、.dim中会继续讲一些细节的东西。另外在随后的文章中会介绍Bkd-Tree插入和更新的内容。

点击下载Markdown文件