索引文件的读取（一）（Lucene 8.4.0）

本系列的文章会通过例子来介绍索引文件的读取，本篇文章先介绍索引文件.dim&&.dii的读取，为了便于理解，请先阅读索引文件的生成（八）之dim&&dii至索引文件的生成（十四）之dim&&dii的文章。

在生成SegmentReader期间，会生成PointsReader（PointsReader为抽象类，实现的子类就是Lucene60PointsReader对象），它用来描述**某个段**中的点数据信息，下面先列出该对象包含的部分信息：

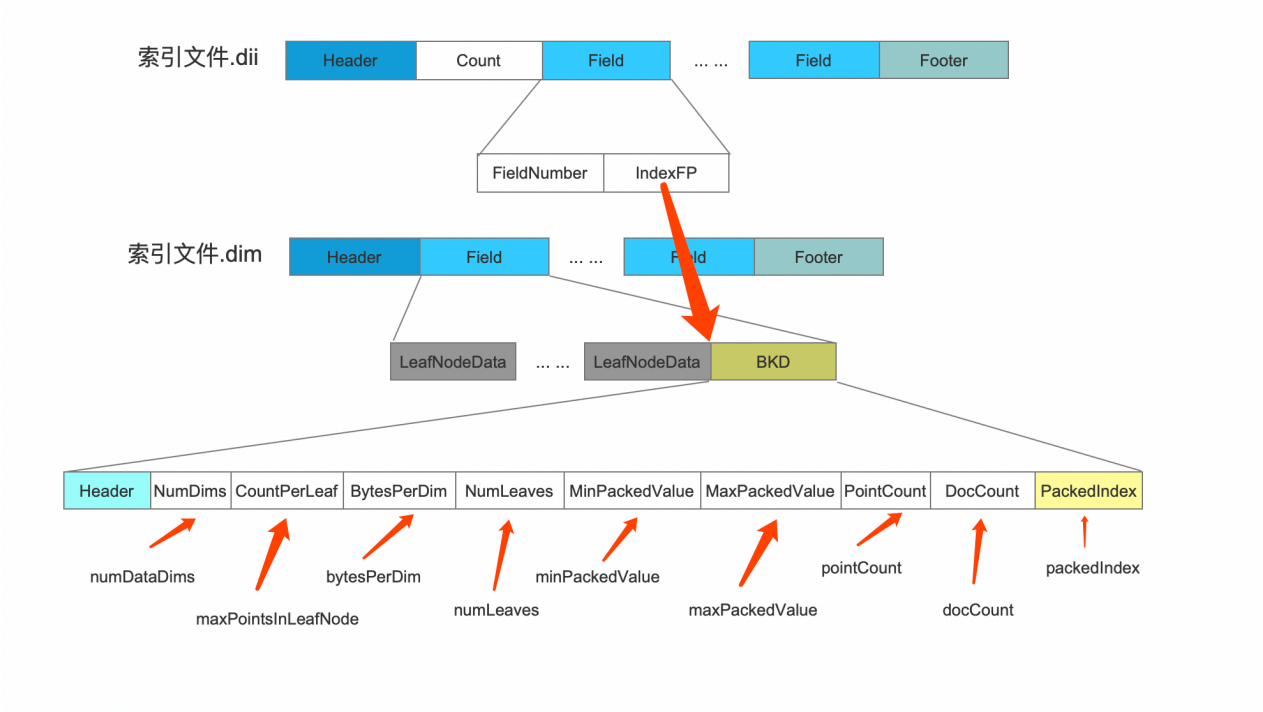
- final Map<Integer,BKDReader> readers = new HashMap<>();

在上述的Map对象中，存放的是不同域名的点数据信息，key为域的编号（FieldNumber），value为域对应的点数据信息，通过读取索引文件.dii来初始化readers对象，BKDReader中包含的主要信息如下所示：

- numDataDims：点数据的维度数量
- maxPointsInLeafNode：每个叶子节点中的最多包含的点数据数量
- bytesPerDim：一个维度值占用的字节数量
- numLeaves：叶子节点的数量
- minPackedValue：MinPackedValue中每个维度的值都是所在维度的最小值
- maxPackedValue：MinPackedValue中每个维度的值都是所在维度的最大值
    - minPackedValue跟maxPackedValue两者描述了BKD树中点数据的数值范围（见下文）
- pointCount：当前域中的点数据的数量
- docCount：包含当前域中的点数据域的文档数量。 一篇文档中可以包含多个相同域名的点数据域，但是docCount的计数为1
- packedIndex：PackedIndex存放了非叶节点的信息

上述的字段在索引文件.dim中的位置如下所示：

图1：



上图中，根据索引文件.dii中的Count获得不同域名的点数据的数量，随后依次遍历每一个Field，通过下面的两个信息读取某个点数据域在索引文件.dim中的信息：

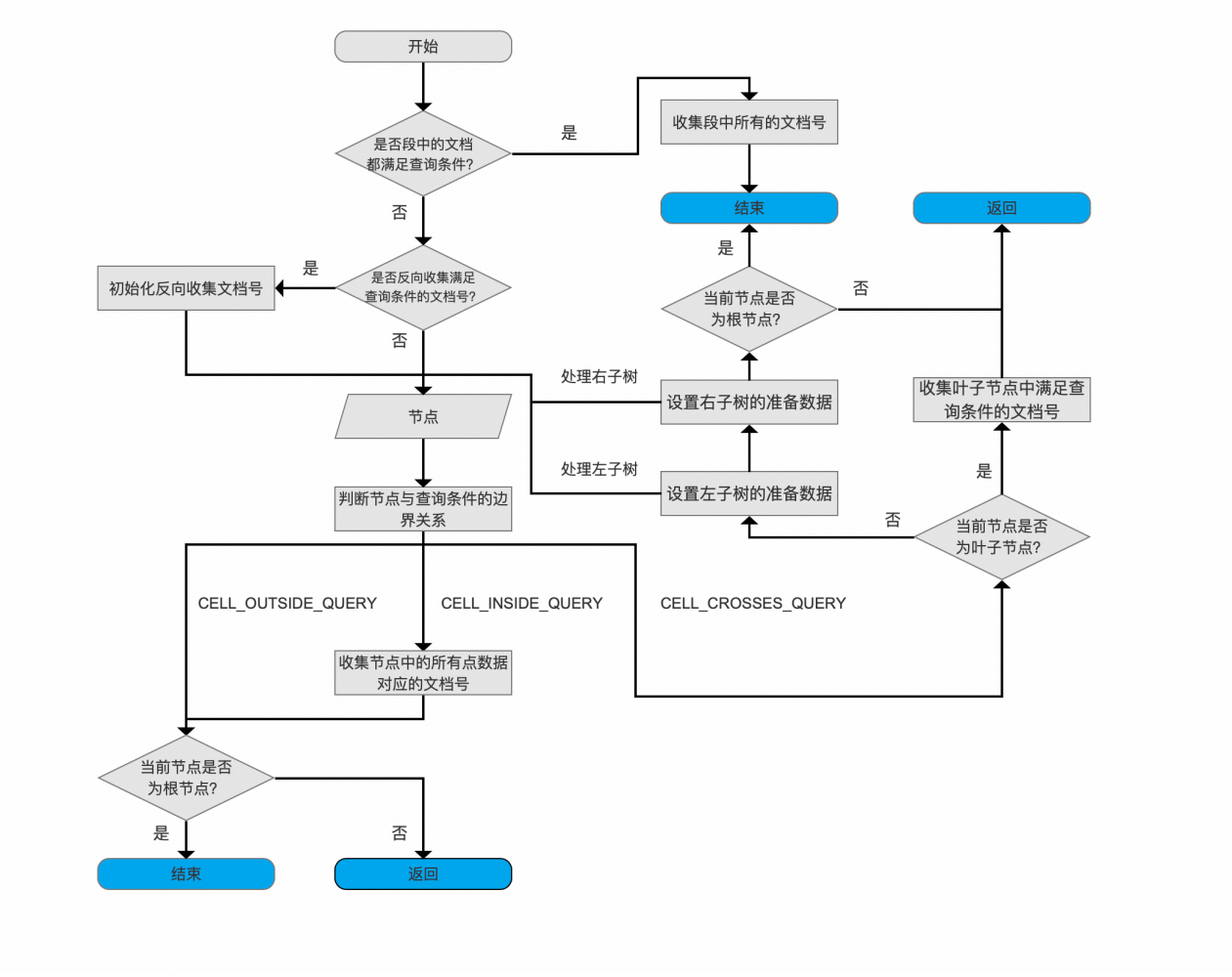
- FiledNumber：域的编号，它作为Map对象readers的key
- IndexFP：该字段指向了该点数据域对应的BKD信息在索引文件.dim中起始读取位置，随后BKD字段的信息被封装到BKDReader对象中，最后作为Map对象readers的value

接着在Search阶段，我们就可以通过PointsReader来读取每一个点数据域的点数据信息了。

读取索引文件.dim&&dii

下面的流程图基于数值类型的范围查询IntPoint.newRangeQuery(String field, int[ ] lowerValue, int[ ] upperValue)方法，流程图中每个流程点描述的内容依赖BKDReader对象中提供的信息：

图2：



点击查看大图

在介绍每一个流程点之前，我们先介绍下图2中CELL_OUTSIDE_QUERY、CELL_INSIDE_QUERY、CELL_CROSSES_QUERY的概念：

先给出源码中的注释：

```
public enum Relation {
    /** Return this if the cell is fully contained by the query */
    CELL_INSIDE_QUERY,
    /** Return this if the cell and query do not overlap */
    CELL_OUTSIDE_QUERY,
    /** Return this if the cell partially overlaps the query */
    CELL_CROSSES_QUERY
  }
```

在索引文件的生成（十一）之dim&&dii文章中我们提到，在生成BKD树的过程中，每生成一个内部节点，该节点的父节点会分别提供给子节点两个值：minPackedValue、maxPackedValue。

- minPackedValue：内部节点中每个维度的最小值
- maxPackedValue：内部节点中每个维度的最大值

minPackedValue跟maxPackedValue用来描述该节点中的点数据的数值范围，另外读取BKD树的方式为深度遍历，即从根节点开始，逐个节点查询，在读取节点信息前，先用查询条件跟节点的minPackedValue、maxPackedValue进行比较，计算出节点中的点数据范围跟查询条件的边界关系，即上文中的Relation，我们通过例子来介绍上述的三种边界关系，为了便于介绍以及图形化，点数据的维度数量为2，并且我们比较的节点为根节点。

根节点中包含的点数据如下所示：

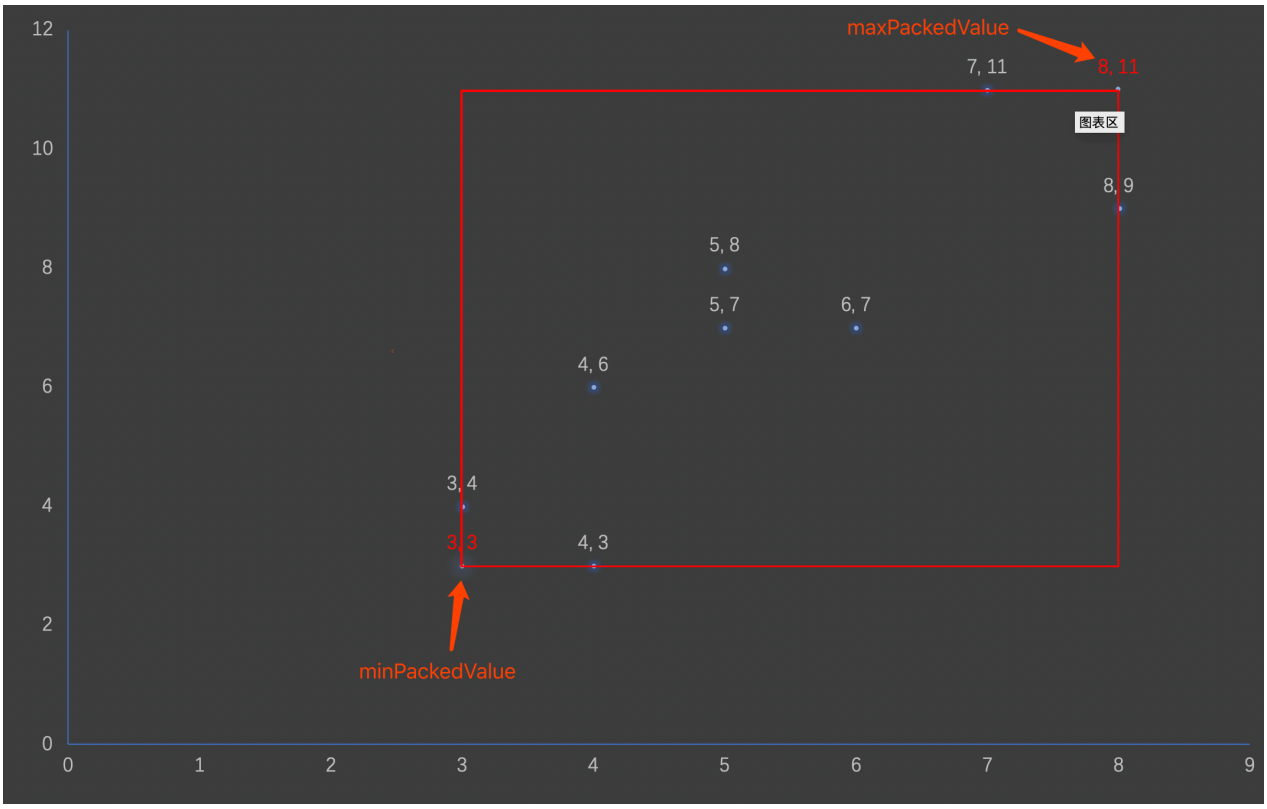
```
{5, 7}, {5, 8}, {4, 6} -> {4, 3} -> {3, 4} -> {7, 11} -> {8, 9} -> {6, 7}
```

那么根节点中的minPackedValue、maxPackedValue（根节点中如何获得minPackedValue、maxPackedValue见文章索引文件的生成（九）之dim&&dii）如下所示：

```
minPackedValue: {3, 3}
maxPackedValue: {8, 11}
```

根据上述的minPackedValue、maxPackedValue，根节点中的点数据的数值范围如下所示红框，由于维度数量是2，所以数值范围可以用一个平面矩形表示，同理如果维度数量是3，那么数据范围就是一个立方体：

图3：



CELL_OUTSIDE_QUERY

CELL_OUTSIDE_QUERY描述的是查询条件的数值范围跟节点的数值范围没有交集，即没有重叠（overlap），在IntPoint.newRangeQuery(String field, int[ ] lowerValue, int[ ] upperValue)方法中，如果lowerValues跟upperValue的值如下所示：

```
lowerValues: {1, 1}
upperValue: {2, 2}
```

lowerValues跟upperValue描述的数值范围如下所示：

图4：

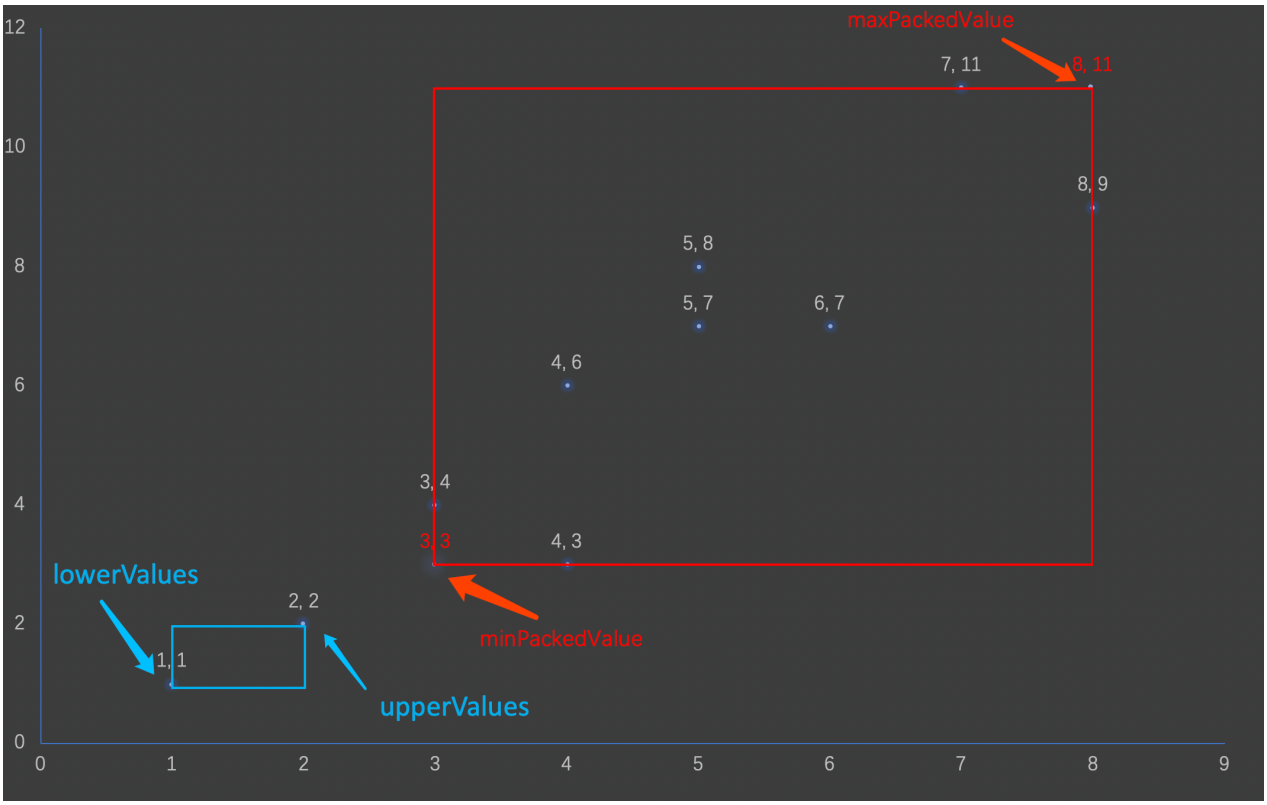


图4可以明显看出，查询条件的数值范围跟根节点没有重合，即CELL_OUTSIDE_QUERY

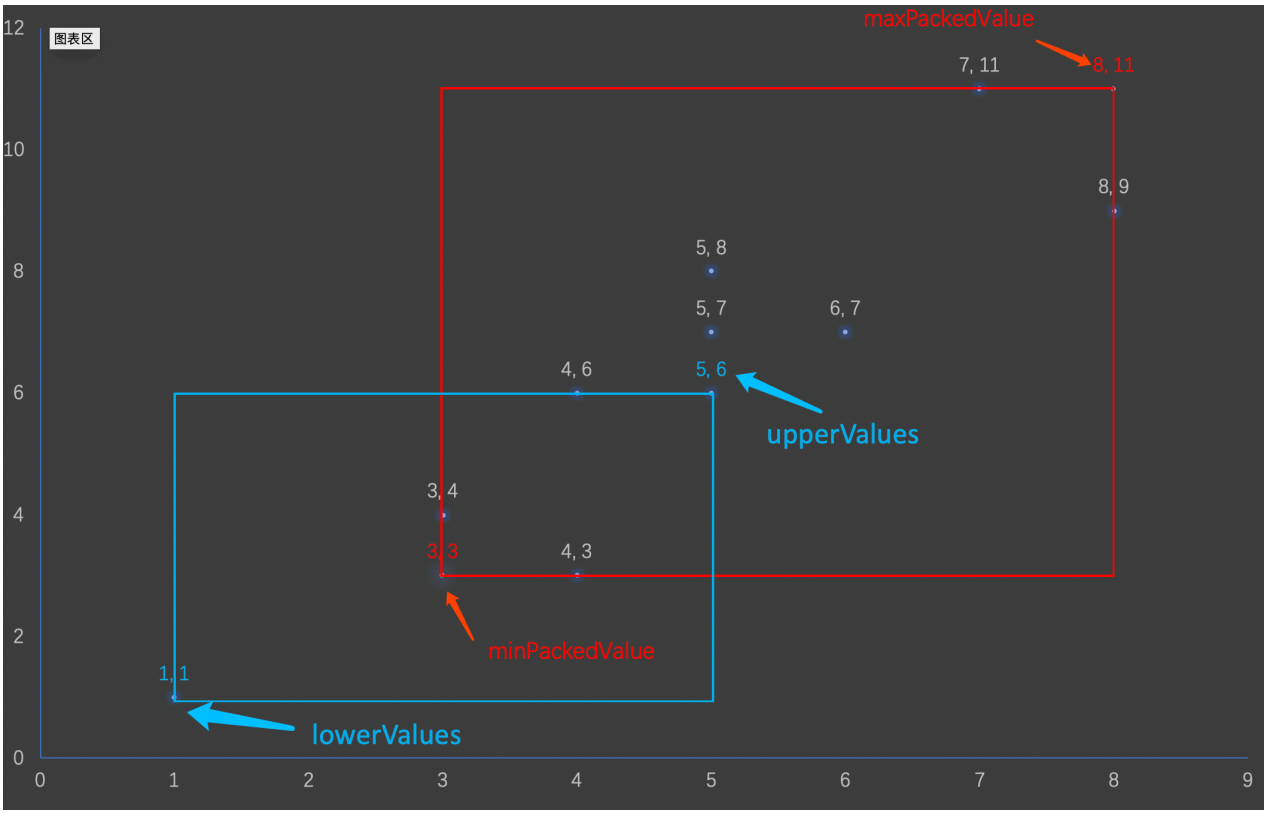
CELL_CROSSES_QUERY

CELL_CROSSES_QUERY描述的是查询条件的数值范围跟节点的数值范围部分重叠（partially overlaps），如果我们以下的lowerValues跟upperValue：

```
lowerValues: {1, 1}
upperValue: {5, 6}
```

lowerValues跟upperValue描述的数值范围如下所示：

图5：



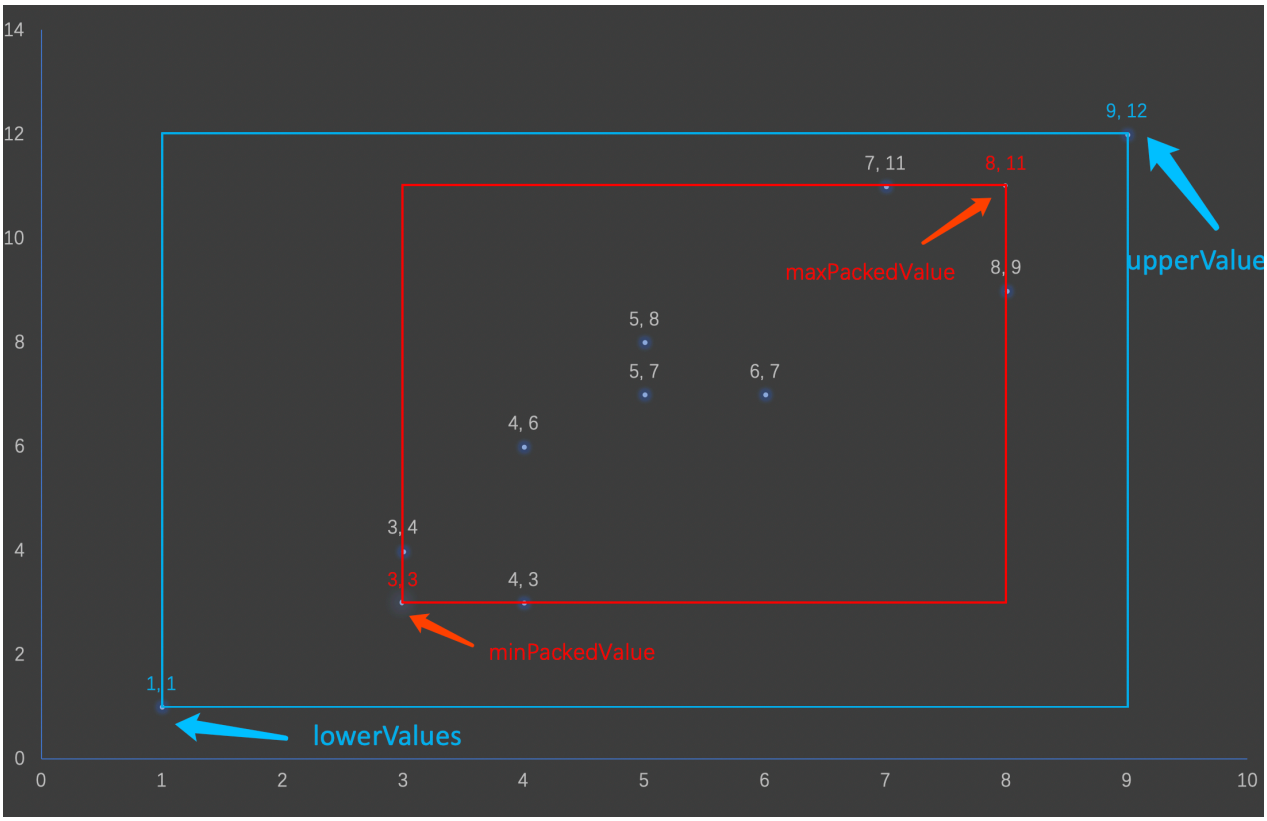
CELL_INSIDE_QUERY

CELL_INSIDE_QUERY描述的是查询条件的数值范围包含节点的中的所有点数据，如果我们以下的lowerValues跟upperValue：

```
lowerValues: {1, 1}
upperValue: {9, 12}
```

lowerValues跟upperValue描述的数值范围如下所示：

图6：



计算CELL_OUTSIDE_QUERY、CELL_INSIDE_QUERY、CELL_CROSSES_QUERY的目的是什么：

目的就是为了以读取性能最高的方式读取完BKD树的信息，例如如果根节点跟查询条件计算出的边界为CELL_OUTSIDE_QUERY，那么就可以不用递归的去读取左右子树的信息，特别是索引目录中有多个段时，那么就可以跳过该段；又例如如果叶子节点的内部节点与查询条件计算出的边界为CELL_INSIDE_QUERY，那么直接读取叶子节点中的所有点数据对应的文档号即可，而对于CELL_CROSSES_QUERY的情况，我们需要递归的去该节点的左右子树中继续处理，处理流程相对复杂，我们将在后面的流程中展开介绍。

结语

无。

点击下载附件