索引文件的生成（十三）（Lucene 8.4.0）

本文承接索引文件的生成（十二）之dim&&dii，继续介绍剩余的内容，为了便于下文的介绍，先给出生成索引文件.dim&&.dii的流程图以及流程点`构建BKD树的节点值（node value）`的流程图：

图1：

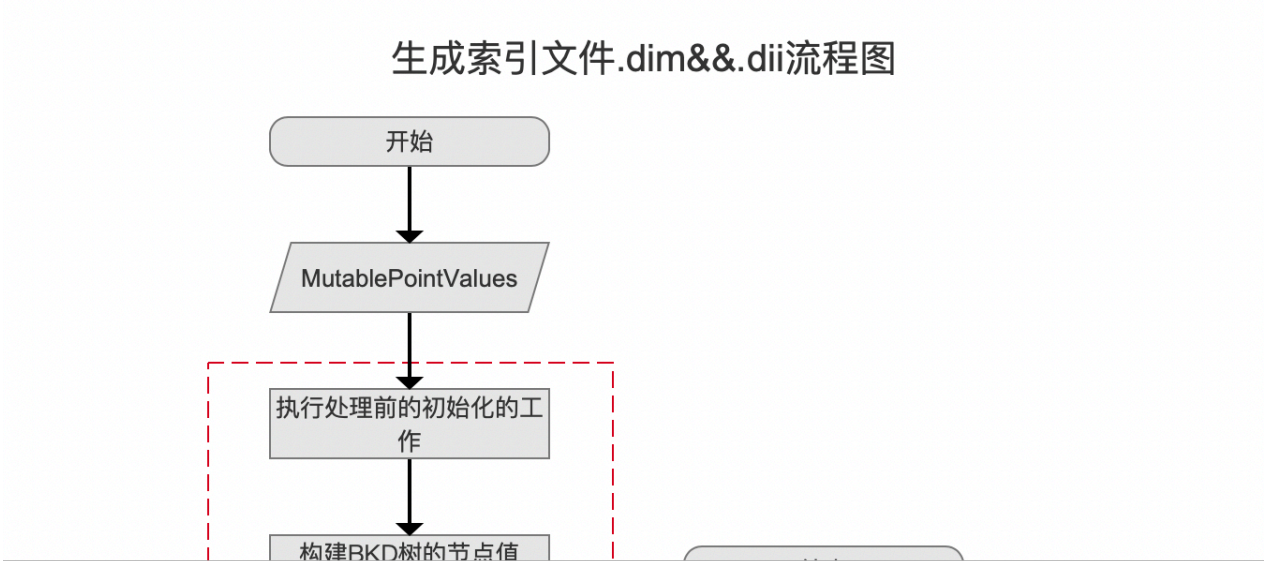
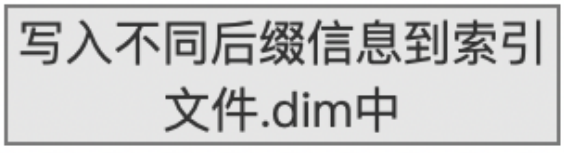


图2：

写入不同后缀信息到索引文件.dim中

图3：



在文章索引文件的生成（十二）之dim&&dii中我们将每个维度的最长公共前缀的信息写入到了索引文件.dim中，同样后缀信息也需要将写进去。

将维度值拆分为相同的前缀跟不相同的后缀两部分并且分别存储的目的在于降低存储开销（reduce the storage cost）,为了能进一步降低存储开销，在当前流程点，会遍历一次叶子节点中的所有点数据，找出一个或者多个区间，在某个区间里面的点数据的排序维度（见在文章索引文件的生成（十二）之dim&&dii）对应的维度值，该维度值的不同后缀的第一个字节是相同的。

图4：

图4中，我们假设排序维度为维度编号2，可以看出排序维度对应的维度值的最长公共前缀的长度为2个字节，随后我们就比较不同后缀的第一个字节（即公共前缀的下一个字节），这个字节相同的点数据集合认为是同一个区间的，那么就可以划分出三个区间，他们包含的点数据如下所示：

```
区间1（红框）：{12, 5, 12}、{23, 1, 13}
区间2（蓝框）：{3, 5, 268}、{20, 3, 270}、{4, 5, 271}
区间3（绿框）：{8, 1, 780}
```

为什么划分区间的规则只考虑不同后缀的第一个字节并且能降低存储开销

以图4为例，维度值的排序方式为从高到低依次比较一个字节，由于图4中维度编号2的最长公共前缀的长度为2，说明是按照不同后缀的第一个字节排序的，这就意味着，存在连续的一个或多个维度值，这些维度值在这个字节的值一样的，那么这个相同的字节就不用重复存储，提取出来之后存储一次即可，从而降低存储开销。

不同后缀信息对应在索引文件.dim中的位置如下所示：

图5：

我们以图4为例结合图5，图5中的PackedValuesRange的数量就是3，假设PackedValuesRange描述的是图4中蓝框对应的区间，那么PrefixByte的值就是 二进制0B00000001，RunLen的值为3，也就是PackedValue的数量为3。

同时还要记录当前叶子节点中每一个维度的最大值跟最小值，并且只保存后缀值，在读取阶段通过跟commonPrefixes就可以拼出原始值以及排序维度编号，这两个信息在索引文件.dim中的位置如下所示：

图6：

以图4为例，SortedDim的值为2（维度编号）。

在上文中，我们说到，通过区间划分能降低存储开销，但是只考虑了不同后缀的一个字节，如果有一个或者多个（不是全部）点数据是完全一样，点数据的每个维度的值都是相同的，那么即使使用了区间划分，图5中的一个或者PackedValue之间中还是会存储相同的Suffix，在Lucene8.2.0之后，针对这个现象作了一些优化。

Lucene 8.2.0的优化

在生成图5的PackedValues之前，会先计算几个参数，leafCardinality（源码中的变量）、lowCardinalityCost、highCardinalityCost。

leafCardinality

在执行完图2中的流程点`叶子节点的排序`之后，通过遍历一次叶子节点中的所有点数据，将这些有序的点数据划分出n个区间，单个区间内至少包含一个点数据并且区间内的点数据的每个维度的值都是相同，leafCardinality的值即区间的数量。

图7：

图7中，划分出4个区间，那么leafCardinality的值就为4。

在Lucene 8.2.0版本之后，图6中的PackedValues中的信息会根据lowCardinalityCost、highCardinalityCost两个参数的大小关系，使用**不同的数据结构来存储点数据的不同后缀信息，并且对应的数据结构占用存储空间较小**，也就是说lowCardinalityCost、highCardinalityCost两个参数描述了存储点数据的不同后缀信息的需要占用的存储空间大小，其计算方式如下所示：

```
highCardinalityCost = count * (packedBytesLength - prefixLenSum - 1) + 2 * numRunLens;
lowCardinalityCost = leafCardinality * (packedBytesLength - prefixLenSum + 1);
```

这两个参数计算的是图6中PackedValues占用的存储空间，其中highCardinalityCost对应的数据结构即图6中的PackedValues，即Lucene 8.2.0之前的版本，而lowCardinalityCost对应的数据结构在下文中会给出。

- highCardinalityCost

上述公式中，count的值就是叶子节点中点数据的数量，(packedBytesLength - prefixLenSum - 1) 描述的是单个点数据后缀信息占用的字节数，(2* numRunLens) 描述的是图6中每一个PackedValuesRange中PrefixByte跟RunLen的累加和。

图8：

上图中，红框标注的字段占用的存储空间大小即highCardinalityCost。另外蓝框标注的两个字段SortedDim、ActualBounds在Lucene8.2.0之后对调了位置，可以跟图5中的内容（Lucene 7.5.0）比较下。

- lowCardinalityCost

我们先介绍下lowCardinalityCost对应的数据结构：

图9：

在上文的计算lowCardinalityCost公式中，leafCardinality就是划分出的区间数量，即图9中PackedValuesRange的数量，以图7中的数据为例，PackedValuesRange的数量就是4，cardinality描述的是当前区间内点数据的数量，由于单个区间的点数据完全相同，所以只需要存储一个PackedValue即可。

结语

至此，图2的所有流程点都介绍完毕。

点击下载附件