链滴

图解 B 树 B+ 树算法

作者：jianzh5

原文链接：https://ld246.com/article/1589245190037

来源网站：链滴

<h2 id="1--B树">1. B 树</h2>
在介绍 B+ 树之前， 先简单的介绍一下 B 树，这两种数据结构既有相似之处，也有他们的区别最后，我们也会对比一下这两种数据结构的区别。
<h3 id="1-1-B树概念">1.1 B 树概念</h3>
B 树也称 B-树,它是一颗多路平衡查找树。二叉树我想大家都不陌生，其实，B 树和后面讲到的 B 树也是从最简单的二叉树变换而来的，并没有什么神秘的地方，下面我们来看看 B 树的定义。

每个节点最多有 m-1 个关键字（可以存有的键值对）。

根节点最少可以只有 1 个关键字。

非根节点至少有 m/2 个关键字。

每个节点中的关键字都按照从小到大的顺序排列，每个关键字的左子树中的所有关键字都小于它而右子树中的所有关键字都大于它。

所有叶子节点都位于同一层，或者说根节点到每个叶子节点的长度都相同。

每个节点都存有索引和数据，也就是对应的 key 和 value。

所以，根节点的关键字数量范围：<code>1 <= k <= m-1</code>，非根节点的关键字量范围：<code>m/2 <= k <= m-1</code>。
另外，我们需要注意一个概念，描述一颗 B 树时需要指定它的阶数，阶数表示了一个节点最多有少个孩子节点，一般用字母 m 表示阶数。
我们再举个例子来说明一下上面的概念，比如这里有一个 5 阶的 B 树，根节点数量范围：1 <= k <= 4，非根节点数量范围：2 <= k <= 4。
下面，我们通过一个插入的例子，讲解一下 B 树的插入过程，接着，再讲解一下删除关键字的过。
<h3 id="1-2-B树插入">1.2 B 树插入</h3>
插入的时候，我们需要记住一个规则：判断当前结点 key 的个数是否小于等于 m-1，如果满足直接插入即可，如果不满足，将节点的中间的 key 将这个节点分为左右两部分，中间的节点放到父节中即可。
例子：在 5 阶 B 树中，结点最多有 4 个 key,最少有 2 个 key（注意：下面的节点统一用一个节表示 key 和 value）。

插入 18，70，50,40

插入 22

插入 22 时，发现这个节点的关键字已经大于 4 了，所以需要进行分裂，分裂的规则在上面已经讲了分裂之后，如下。

ogfile.com/file/2020/05/image-5fe13404.png?imageView2/2/interlace/1/format/jpg">

接着插入 23，25，39

分裂，得到下面的结果：

更过的插入的过程就不多介绍了，相信有这个例子你已经知道怎么进行插入操作了。
<h3 id="1-3-B树的删除操作">1.3 B 树的删除操作</h3>
B 树的删除操作相对于插入操作是相对复杂一些的，但是，你知道记住几种情况，一样可以很轻的掌握的。

现在有一个初始状态是下面这样的 B 树，然后进行删除操作。

删除 15，这种情况是删除叶子节点的元素，如果删除之后，节点数还是大于 <code>m/2</cod>，这种情况只要直接删除即可。

接着，我们把 22 删除，这种情况的规则：22 是非叶子节点，**对于非叶子节点的删除，我们需用后继 key（元素）覆盖要删除的 key，然后在后继 key 所在的子支中删除该后继 key。**对于删除 2，需要将后继元素 24 移到被删除的 22 所在的节点。

此时发现 26 所在的节点只有一个元素，小于 2 个（m/2），这个节点不符合要求，这时候的规则（兄弟节点借元素）：删除叶子节点，如果删除元素后元素个数少于（m/2），并且它的兄节点的元素大于（m/2），也就是说兄弟节点的元素比最少值 m/2 还多，将先将父节点的元素移到节点，然后将兄弟节点的元素再移动到父节点。 这样就满足要求了。

接着删除 28，删除叶子节点，删除后不满足要求，所以，我们需要考虑向兄弟节点借元素，但，兄弟节点也没有多的节点（2 个），借不了，怎么办呢？如果遇到这种情况，首先，还是将先将父点的元素移到该节点，然后，将当前节点及它的兄弟节点中的 key 合并，形成一个新的节点。

移动之后，需要跟兄弟节点合并：

删除就只有上面的几种情况，根据不同的情况进行删除即可。
上面的这些介绍，相信对于 B 树已经有一定的了解了，接下来的一部分，我们接着讲解 B+ 树，相信加上 B+ 树的对比，就更加清晰明了了。

<h2 id="2-B-树">2 B+ 树</h2>
<h3 id="2-1-B-树概述">2.1 B+ 树概述</h3>
B+ 树其实和 B 树是非常相似的，我们首先看看相同点。

根节点至少一个元素

非根节点元素范围：m/2 <= k <= m-1

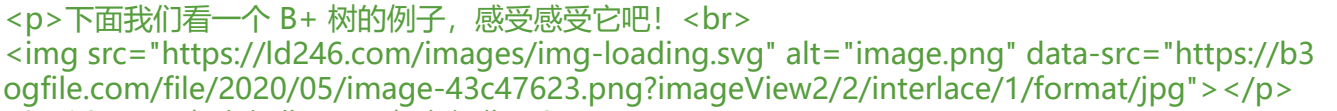
不同点。

B+ 树有两种类型的节点：内部结点（也称索引结点）和叶子结点。内部节点就是非叶子节点，部节点不存储数据，只存储索引，数据都存储在叶子节点。

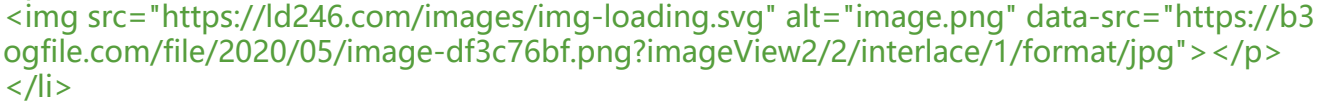
内部结点中的 key 都按照从小到大的顺序排列，对于内部结点中的一个 key，左树中的所有 key 都小于它，右子树中的 key 都大于等于它。叶子结点中的记录也按照 key 的大小排列。

每个叶子结点都存有相邻叶子结点的指针，叶子结点本身依关键字的大小自小而大顺序链接。</
>

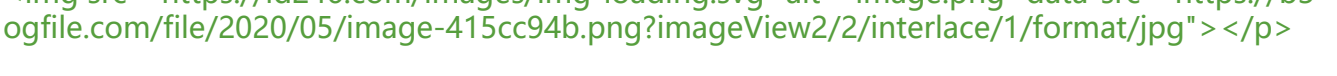
父节点存有右孩子的第一个元素的索引。

下面我们看一个 B+ 树的例子，感受感受它吧！

<h3 id="2-2-插入操作">2.2 插入操作</h3>
对于插入操作很简单，只需要记住一个技巧即可：当节点元素数量大于 m-1 的时候，按中间元分裂成左右两部分，中间元素分裂到父节点当做索引存储，但是，本身中间元素还是分裂右边这一部的。
下面以一颗 5 阶 B+ 树的插入过程为例，5 阶 B+ 树的节点最少 2 个元素，最多 4 个元素。

插入 5，10，15，20

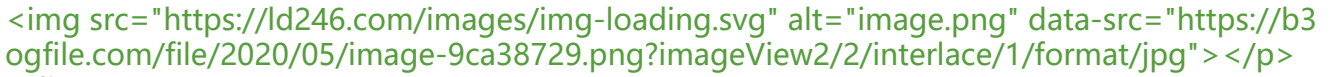

插入 25，此时元素数量大于 4 个了，分裂


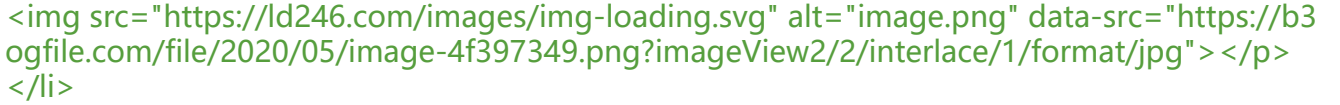
接着插入 26，30，继续分裂

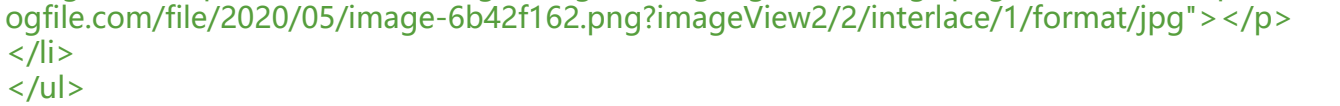


有了这几个例子，相信插入操作没什么问题了，下面接着看看删除操作。
<h3 id="2-3-删除操作">2.3 删除操作</h3>
对于删除操作是比 B 树简单一些的，因为叶子节点有指针的存在，向兄弟节点借元素时，不需要过父节点了，而是可以直接通过兄弟节移动即可（前提是兄弟节点的元素大于 m/2），然后更新父节的索引；如果兄弟节点的元素不大于 m/2（兄弟节点也没有多余的元素），则将当前节点和兄弟节点并，并且删除父节点中的 key，下面我们看看具体的实例。

初始状态


删除 10，删除后，不满足要求，发现左边兄弟节点有多余的元素，所以去借元素，最后，修改节点索引


删除元素 5，发现不满足要求，并且发现左右兄弟节点都没有多余的元素，所以，可以选择和兄节点合并，最后修改父节点索引


发现父节点索引也不满足条件，所以，需要做跟上面一步一样的操作。


这样，B+ 树的删除操作也就完成了，是不是看完之后，觉得非常简单！
<h2 id="3-B树和B-树总结">3 B 树和 B+ 树总结</h2>
B+ 树相对于 B 树有一些自己的优势，可以归结为下面几点。

单一节点存储的元素更多，使得查询的 IO 次数更少，所以也就使得它更适合做为数据库 MySQL 的底层数据结构了。

所有的查询都要查找到叶子节点，查询性能是稳定的，而 B 树，每个节点都可以查找到数据，所不稳定。

所有的叶子节点形成了一个有序链表，更加便于查找。

