48 | 多路查找树：B+树的插入与删除操作详解

2023-06-02 王健伟 来自北京

《快速上手C++数据结构与算法》

你好，我是王健伟。

上节课我们详细讲解了多路查找树中的 B 树，这节课我们来聊一聊 B+ 树。B+ 树有人理解为 B 树的升级，有人理解为 B 树的变形（变体），都可以。性质上来看，B+ 树与 B 树基本相同，但还是有一些不同点的。

B+ 树的所有非叶子节点中的数据都会包含在叶子节点中。

B+ 树的所有叶子节点相连，从而构成了一个数据从小到大排列的链表（虽然绘制时常绘制成一个单链表，但实际应用中，往往实现成一个双链表以方便对数据的查找）。

下面，我带你看一看 B+ 树都有哪些操作。

B+ 树的插入操作

B+ 树的插入操作和 B 树非常类似。在这里看一个 B+ 树节点插入的步骤。假如要按顺序给出如下数据创建一棵 4 阶 B+ 树：

11，12，6，5，13，7，3，4，2，1，9，8，10

4 阶 B+ 树同样遵循每个节点最少有 1 个数据，最多有 3 个数据。创建的步骤如下：

因为当前并不存在 B+ 树，所以以 11 为根创建一棵 B+ 树。

插入 12，因为 12 大于 11，所以 12 位于 11 的右侧，与 11 共用一个节点。

插入 6，因为 6 小于 11，所以 6 位于 11 的左侧，与 11、12 共用一个节点。

插入 5，因为 5 小于 6，所以 5 位于 6 的左侧，此时注意，5、6、11、12 共用一个节点。但因为 4 阶 B+ 树最多有 3 个数据，因此这个节点必须要进行拆分（分裂），拆分的原则与 B 树一样—取这几个数据中间（[m/2]）的那个数据并作为根节点，剩余的数据分别做这个节点的左孩子和右孩子节点。讲解 B 树时取了第 2 个数据作为根节点，在这里取第 3 个数据作为根节点（4 个数据中，第 2 个或者第 3 个数据都可以看成是中间的数据）。对于 5、6、11、12，取第 3 个数据 11 作为根节点，将数据 5、6 所代表的节点作为 11 的左孩子，将 12 所代表的节点作为 11 的右孩子。

这里必须强调的两点：

1. B+ 树需要叶子节点存放所有非叶子节点的数据，所以数据 11 也要保存在叶子节点中，把数据 11 放到右孩子（数据 12 所代表的节点）中。换句话说，11 如果在叶子节点中，并且要作为分拆后的根节点，那么 11 就要在分拆后的叶子节点中留一份拷贝。在编写代码时，要遵循这句描述以免代码出现错误。
2. B+ 树需要叶子节点相连。

目前的 B+ 树如图 1 所示：

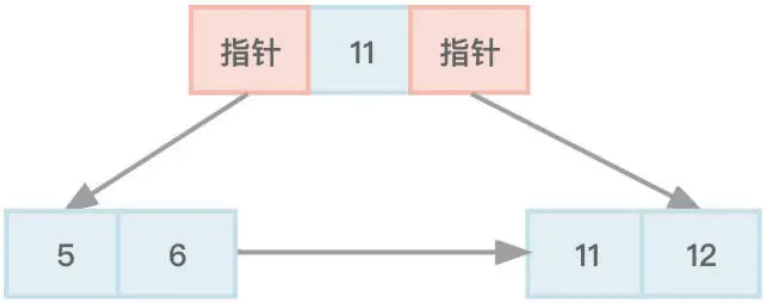

图1 包含5个数据的4阶B+树

插入 13，从根节点 11 开始，因为 13 大于 11，因此沿着 11 的右指针向下找，找到 11、12 这个节点，因为 13 大于 12，因此按照排列顺序，11、12、13 三个数据被放到一起作为一个节点。

插入 7，从根节点 11 开始，通过比较大小，将 7 放到 5、6 所在的节点中，注意顺序，现在 5、6、7 在一个节点中。如图 2 所示：

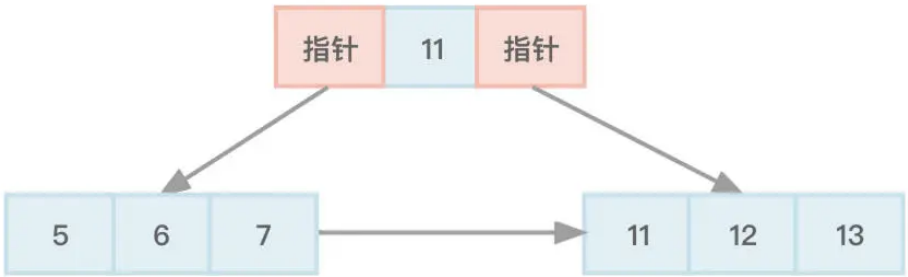

图2 包含7个数据的4阶B+树

插入 3，从根节点 11 开始，通过比较大小，将 3、5、6、7 这 4 个数据放到一起作为一个节点，因为 4 阶 B+ 树最多有 3 个数据，因此这个节点必须要进行拆分，将 6 作为根节点，将 3、5 这两个数据所代表的节点作为 6 的左孩子，将 6、7 这两个数据所代表的节点

作为 6 的右孩子。再将 6 这个节点并到数字 11 所代表的根节点中（因为根节点还不超过 3 个数据），注意因为 11 大于 6，因此 11 排在 6 后面，如图 3 所示：

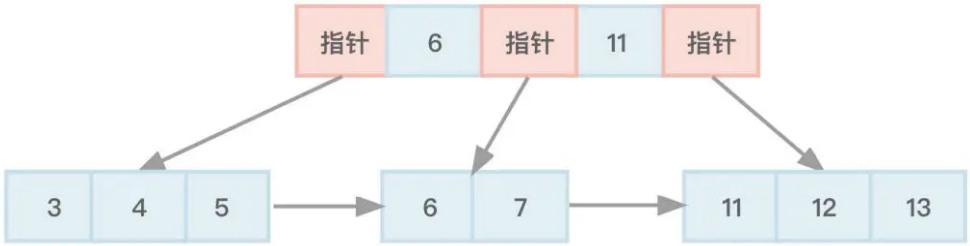

图3 包含9个数据的4阶B+树

插入 4，放到 3、5 所在的节点中，注意顺序，现在 3、4、5 在一个节点中。

插入 2，将 2、3、4、5 这 4 个数据放到一起作为一个节点，必须要对该节点进行拆分，将 4 作为根节点，将 2、3 作为 4 的左孩子，将 4、5 作为 4 的右孩子，将 4 并到 6、11 所在的节点，如图 4 所示：

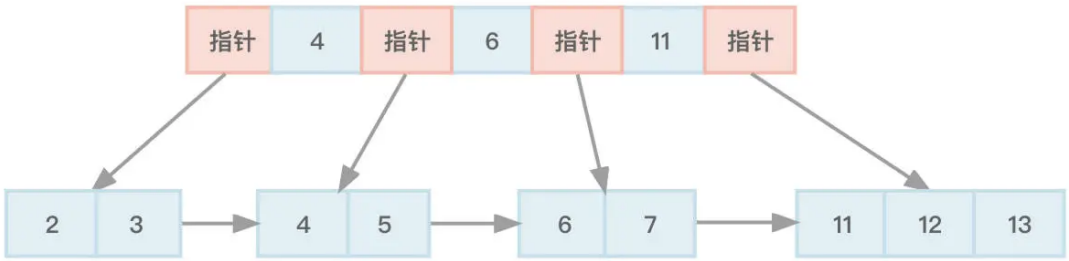

图4 包含12个数据的4阶B+树

插入 1，放到 2、3 所在的节点中，注意顺序，现在 1、2、3 在一个节点中。

插入 9，放到 6、7 所在的节点中，注意顺序，现在 6、7、9 在一个节点，如图 5 所示：

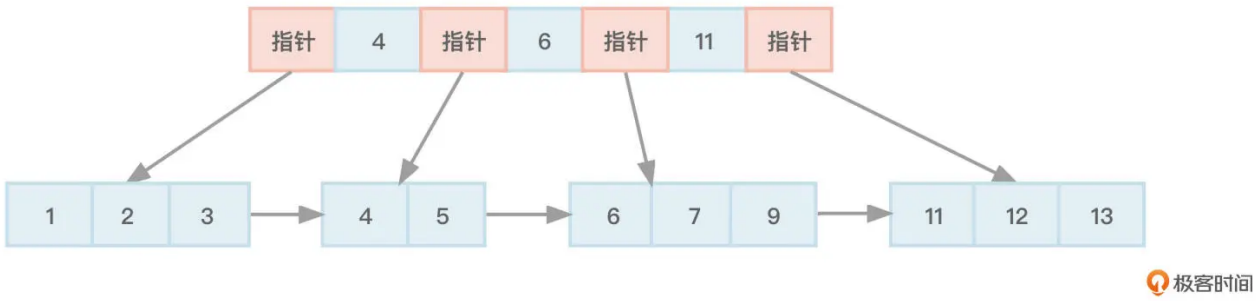

图5 包含14个数据的4阶B+树

插入 8，放到 6、7、9 所在的节点中，注意顺序，现在 6、7、8、9 这 4 个数据放到一起作为一个节点，必须要对该节点进行拆分，将 8 作为根节点，将 6、7 作为 8 的左孩子，将 8、9 作为 8 的右孩子，将 8 并到 4、6、11 所在的节点，如图 6 所示：

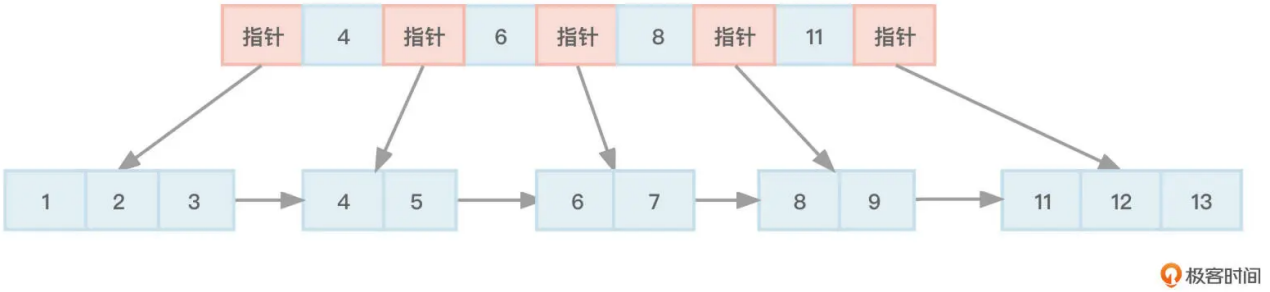

图6 包含16个数据的4阶B+树（中间状态）

图 6 是这棵 B+ 树的中间状态，因为 4、6、8、11 所在节点需要继续进行拆分。将 8 作为根节点，将 4、6 作为 8 的左孩子，将 11 作为 8 的右孩子。注意这里仅仅将 11 作为 8 的右孩子，因为 8 已经不是叶子节点，8 在叶子节点中已经存在了，所以不能将 8 和 11 放在一起作为 8 的右孩子。如图 7 所示：

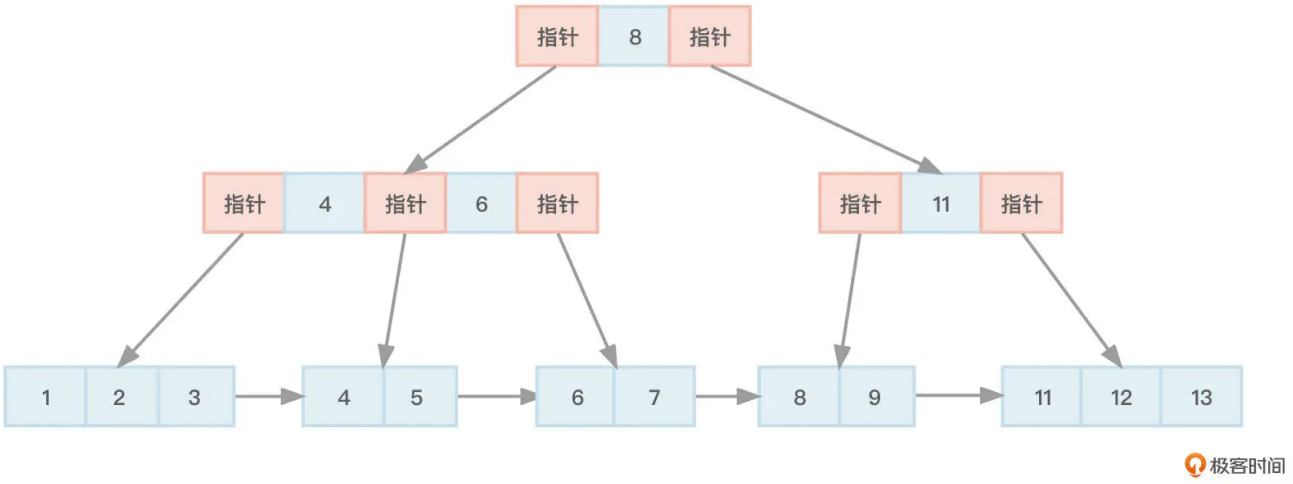

图7 包含16个数据的4阶B+树（最终状态）

插入 10，放到 8、9 所在的节点中，注意顺序，现在 8、9、10 在一个节点中，如图 8 所示：

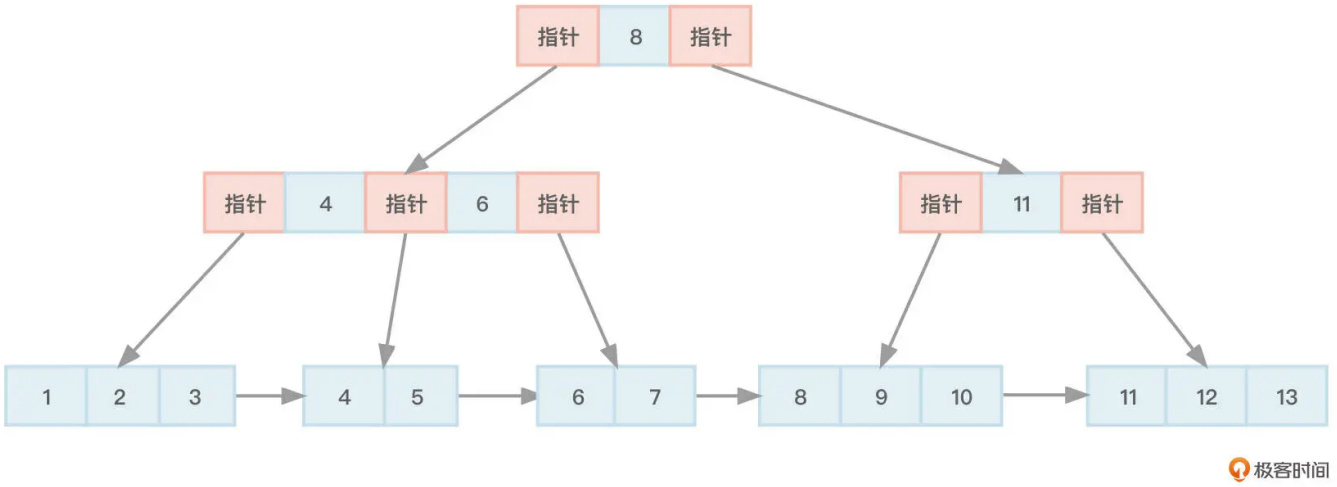

图8 包含17个数据的4阶B+树

特别提示，B+ 树在不同的资料中表述和实现存在差异。前面谈到的 B+ 树，非叶子节点的子节点数目比该节点的数据数目多 1。但有的资料中对 B+ 树的表述是非叶子节点的子节点数目是和数据数目相等的。这两种表述得到的 B+ 树外观和实现代码都会有些不同，但都是 B+ 树的正确实现方式。这里我就采用“非叶子节点的子节点数目是比该节点的数据数目多 1”的表述和实现方式来实现 B+ 树，因为这种实现方式与前面所实现的 B 树最接近。

B+ 树的插入操作实现代码与 B 树的插入操作实现代码大同小异。考虑到已经完整的实现过 B 树的插入代码，这里我就不实现 B+ 树的插入操作代码了。你如果有兴趣，可以自行实现相关的代码。如果在代码编写中遇到问题，可以参考前面的“可视化数据结构算法演示网站”页面并单击其中的“B+ Trees”链接，或者直接访问页面对 B+ 树中插入、删除等操作进行演示，这对更深入的理解 B+ 树和对正确的写出 B+ 树的插入、删除等操作将起到至关重要的作用。

B+ 树的删除操作

B+ 树的删除操作其实和 B 树也非常类似。因为 B+ 树的所有非叶子节点中的数据都会包含在叶子节点中，所以删除 B+ 树中的任何数据在终端节点（叶子节点）中都可以找到。明确一下 B+ 树每个节点有多少个数据。以一棵 5 阶 B+ 树为例：

一棵 5 阶 B+ 树，每个节点最多有 4 个数据。

根节点可以只有 1 个数据，非根节点至少有 2 个数据。

删除终端节点中的数据，一棵 5 阶 B+ 树为例，分为两种情况。如图 9 所示的一棵 5 阶 B+ 树：

图9 一棵5阶B+树

情况一：删除数据后节点中的数据数量不低于下限

只要删除数据后节点中的数据数量不低于下限，就可以直接在该节点中将数据删除。然后判断被删数据在原节点中是否位于最左侧（在原节点中值最小），若是意味着该数据很可能在其前辈节点中也存在，此时需要继续向树根方向回溯，查看该被删除的数据在中间节点或根节点是否存在，若存在则用叶子节点中该数据的后继数据（该数据右侧的数据）取代在中间节点或根节点中的该值。

在图 9 中，若要删除数据 75，则直接在叶子节点中删除即可。而若要删除数据 70，则除在叶子节点中删除数据 70，还要用 70 的后继数据 75 来取代根节点中的数据 70。所以，删除数据 70 后得到的结果如图 10 所示：

图10 一棵5阶B+树删除数据70后的情形

情况二：如果将数据删除后节点的数据数量低于 2，就会导致节点的合并

如果兄弟节点数据的个数是超过数量 2 的，就可以从兄弟中借一个数据过来。还是以图 9 为例。现在要删除数据 50，删除 50 后所在的节点只剩余数据 60，低于 2 个数据，此时左侧兄弟节点（30、35、40）数据个数超过 2 个，可以借一个数据。具体步骤是把左侧兄弟中最大值 40 拿来和数据 60 放到一起，现在这个节点中最小的值就是 40。又因为删除数据 50 之前，50 是所在节点最左侧的数据，所以继续向树根方向回溯，查看 50 在中间节点或根节点是否存在，若存在则用 40（被删除数据所在节点中当前最小值）取代。最终得到的 B+ 树如图 11 所示：

图11 一棵5阶B+树删除数据50后的情形

如果兄弟节点数据的个数不超过数量 2，就无法从兄弟中借数据。此时，被删除了数据的节点就会和兄弟节点、父节点进行合并。继续以图 9 为例。如果删除数据 15，则该节点只剩余数据 18，这就不满足 5 阶 B+ 树非根节点至少有 2 个数据的要求，于是，父节点中的 15 也用 18 替代，我们将“18 所在节点”与“父节点 18、20、30、50”以及“18 的左兄弟节点 5、10”进行合并，如图 12 以及图 13 所示。

图12 一棵5阶B+树删除数据15后的情形（中间状态）

图13 一棵5阶B+树删除数据15后的情形（最终状态）

合并时注意的是因为 18 在叶子节点中已经不是最左侧的数值，不应该出现在非叶子节点中，因此父节点中应该将 18 删除，新的父节点是 20、30、50。

再看一看图 14 这棵 3 阶 B+ 树，现在要删除数据 29：

图14 一棵3阶B+树

叶子节点 29 删除后，所在的节点就没有数据了，此时该节点与 24 所在节点以及父（29）所在节点就要合并，合并前发现父节点也包含 29，将这个 29 删除。如图 15 所示：

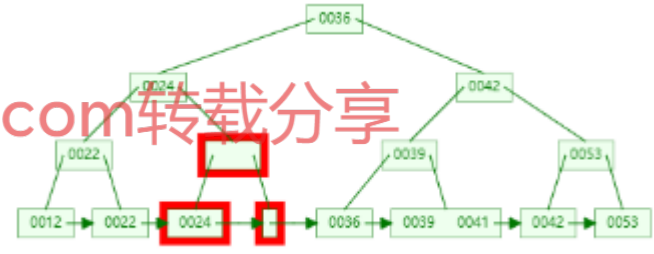

图15 一棵3阶B+树删除数据29后的情形_1（中间状态）

合并之后子节点仍旧只有数据 24，父节点依旧为空，如图 16 所示：

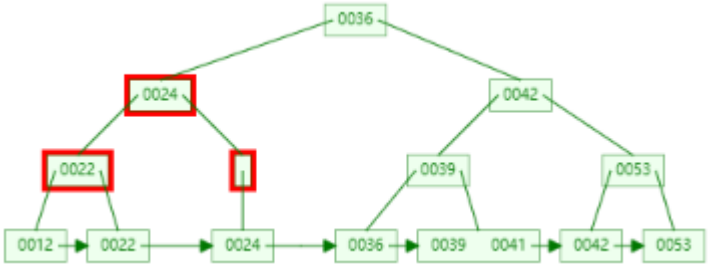

图16 一棵3阶B+树删除数据29后的情形_2（中间状态）

如图 16，因为此时父节点（节点 24 的父亲）为空，不满足 3 阶 B+ 树节点数据最少为 1 的情形，所以该父节点和其左兄弟 22 以及该父节点的父节点 24 要继续合并。合并后如图 17 所示：

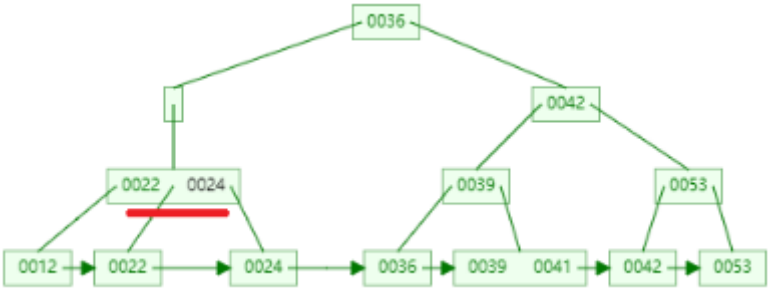

图17 一棵3阶B+树删除数据29后的情形_3（中间状态）

此时 22、24 合并为一个新节点，但该新节点的父亲又为空节点，所以这个空节点要继续与其右兄弟节点 42 和其父节点 36 合并，如图 18 所示：

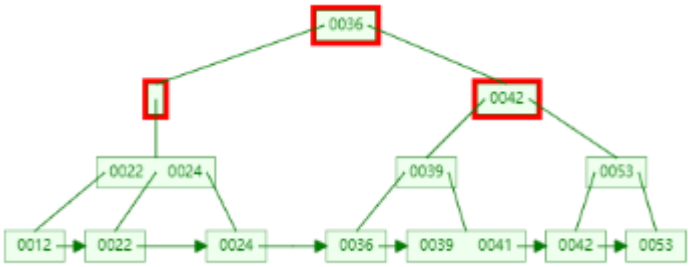

图18 一棵3阶B+树删除数据29后的情形_4（中间状态）

合并后如图 19 所示：

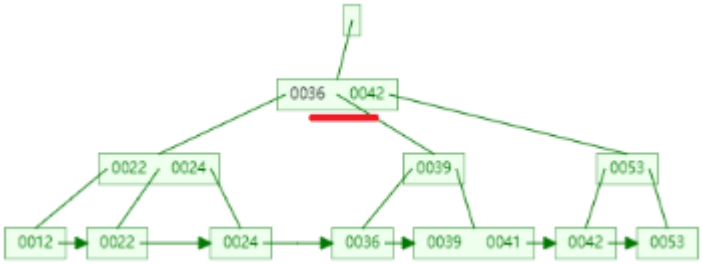

图19 一棵3阶B+树删除数据29后的情形_5（中间状态）

此时 36、42 合并为一个新节点，但该新节点的父亲又为空节点，而该空节点恰恰是根节点，所以只需要把这个空的根节点删除，让 36、42 所在的节点作为整棵 B+ 树的根节点即可，如图 20 所示：

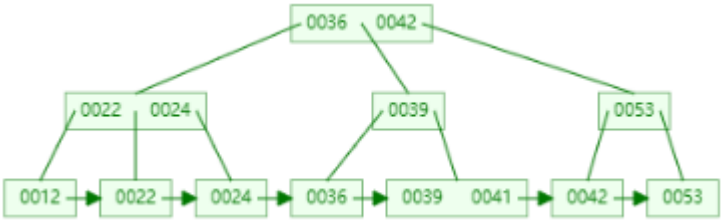

图20 一棵3阶B+树删除数据29后的情形_6（最终状态）

从代码实现的角度来讲，B+ 树的删除代码与 B 树大同小异但要稍微复杂一点，所以要注意非叶子节点中也存在要删除或者替换的数据，也要注意维持叶子节点之间的相连关系。考虑到已经完整的实现过 B 树的删除代码，这里我就不实现 B+ 树的删除操作代码了。

小结

本节我带你详细学习了多路查找树中的 B+ 树。B+ 树是 B 树的升级或变体。它的性质与 B 树基本相同，但也有如下的不同点：

B+ 树的所有非叶子节点中的数据都会包含在叶子节点中。
B+ 树的所有叶子节点相连，从而构成了一个数据从小到大排列的链表。

这节课的重点在于 B+ 树的插入和删除操作的具体步骤。考虑到相关的实现代码与 B 树非常类似，所以我没有为你提供 B+ 树的具体实现代码，如果你有兴趣的话，可以参考 B 树的实现代码来自行实现。

你可能会有疑惑，B+ 树无论从代码实现复杂程度上还是从节点需要保存的数据数量上都高于 B 树，那 B+ 树为什么要这样组织数据呢？这就涉及下节课所讲解的 B+ 树的具体应用了。

思考题

1. B+ 树作为 B 树的变体，它在 B 树的基础之上进行了哪些改变，试着阐述 B+ 树相对于 B 树的优势有哪些？
2. 尝试描述如何利用 B+ 树实现数据记录的范围查询？

欢迎你在留言区和我交流。如果觉得有所收获，也可以把课程分享给更多的朋友一起学习。我们下节课见！

精选留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。