漫画：什么是 B- 树?

2017-07-07 算法爱好者

（点击上方公众号，可快速关注）

> 来源：伯乐专栏作者/玻璃猫，微信公众号 - 梦见（dreamsee321）
> 如有好文章投稿，请点击 → 这里了解详情

呃，面试官考你什么了，说来听听？

当时，面试官是这么考我的……

———————————

小灰是吧？请简单介绍一下你自己。

好的！

blah blah blah .....

Mysql 数据库用过吧？里面的索引是基于什么数据结构？

这个我知道，索引主要是基于 Hash 表或者 B+ 树。

很好，请你说一说 B+ 树的实现细节是什么样？ B- 树和 B+ 树有什么区别？联合索引在 B+ 树中如何存储？

呃，让我想一想.....

……

嘿嘿，不会……


哦，没关系，回家等通知去吧！


小灰，B- 树和 B+ 树都是很基础
的概念，一定要掌握呀。

哎，我也看过相关的书，可是没怎么看懂。你给讲讲呗？

好吧，要弄明白 B+ 树，咱们先要了解什么是 B- 树。需要注意的是，B- 树就是 B 树，中间的横线并不是减号。

谁要是把 B- 树读成 B 减树，那可就丢人现眼喽。

哎呀，我以前一直给读成 B 减树，
嘿嘿……

小灰，你来说说，数据库索引为什么要使用树结构存储呢？

这还不简单，树的查询效率高，
而且可以保持有序。

既然这样，为什么索引没有使用二叉查找树来实现呢？

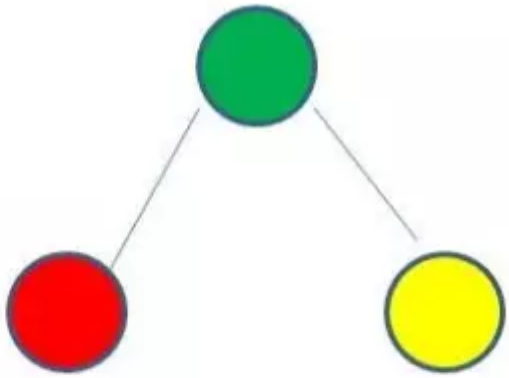
呃，这我就不明白了... 二叉查找树查询的时间复杂度是 O（logN），性能已经足够高了啊，难道 B 树可以比它更快？

其实从算法逻辑上来讲，二叉查找树的查找速度和比较次数都是最小的。但是，我们不得不考虑一个现实问题：磁盘 IO。

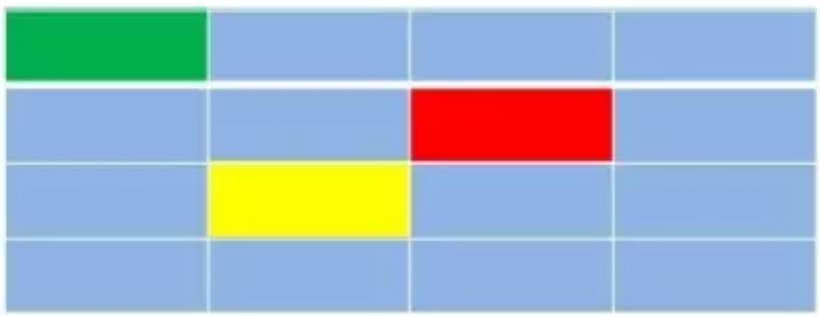
数据库索引是存储在磁盘上的，当数据量比较大的时候，索引的大小可能有几个 G 甚至更多。

当我们利用索引查询的时候，能把整个索引全部加载到内存吗？显然不可能。能做的只有逐一加载每一个磁盘页，这里的磁盘页对应着索引树的节点。

索引树：

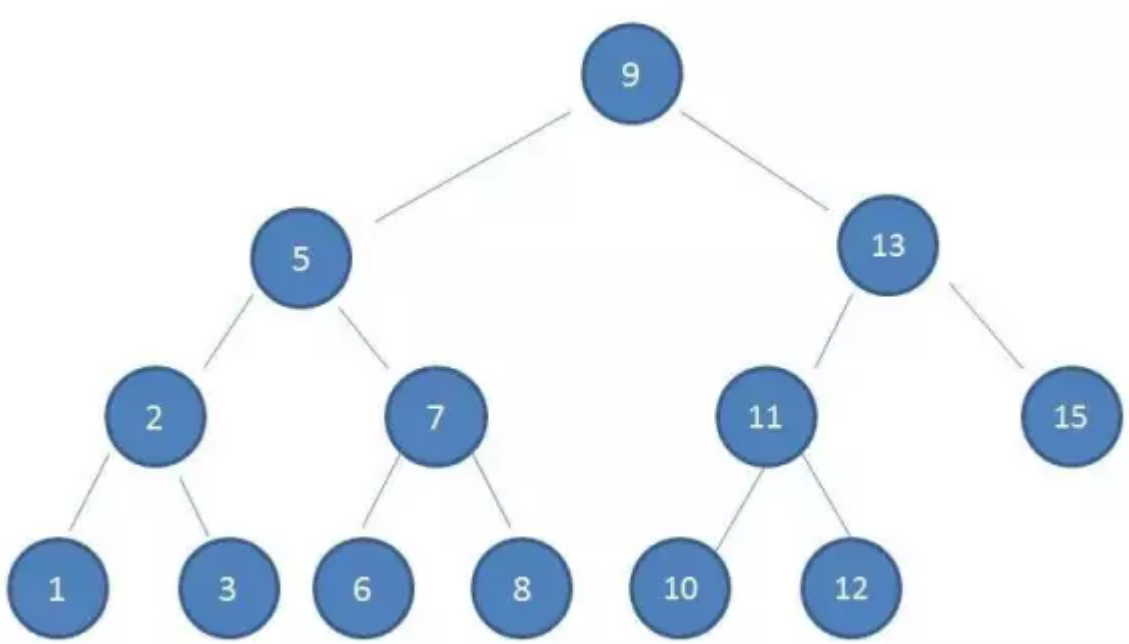


磁盘页：

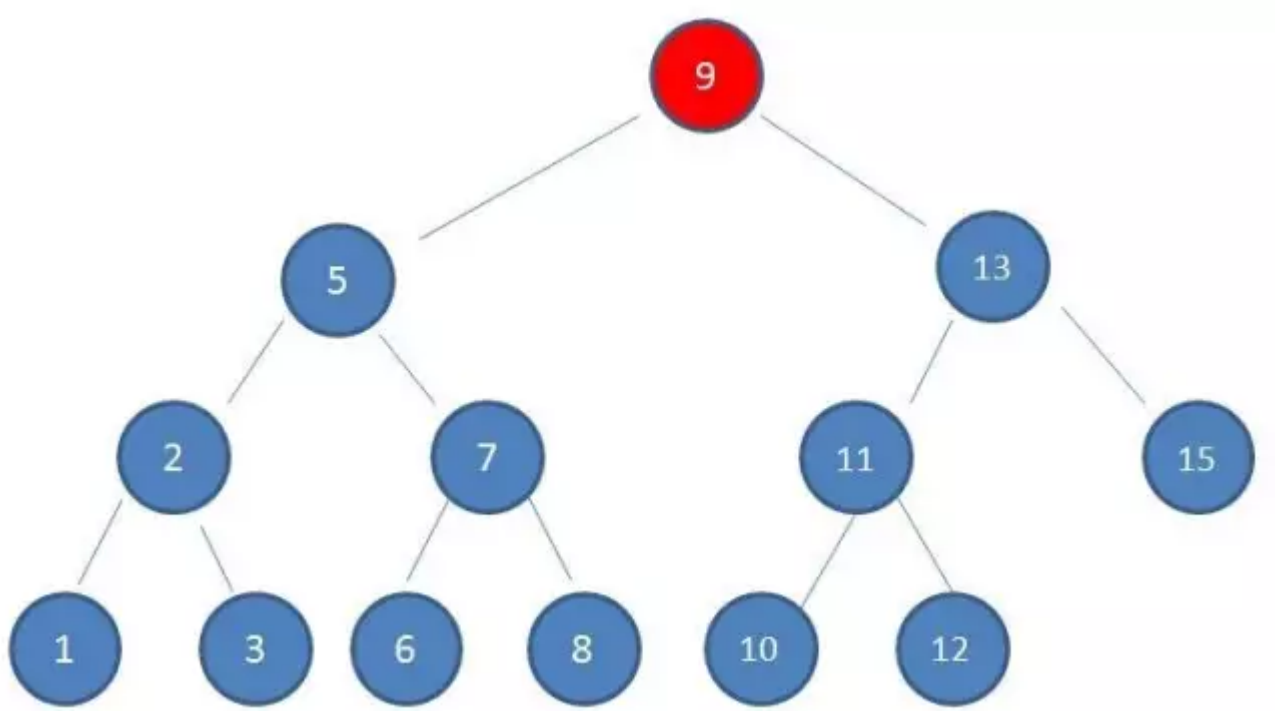


如果我们利用二叉查找树作为索引结构，情形是什么样呢？假设树的高度是4，查找的值是 10，那么流程如下：

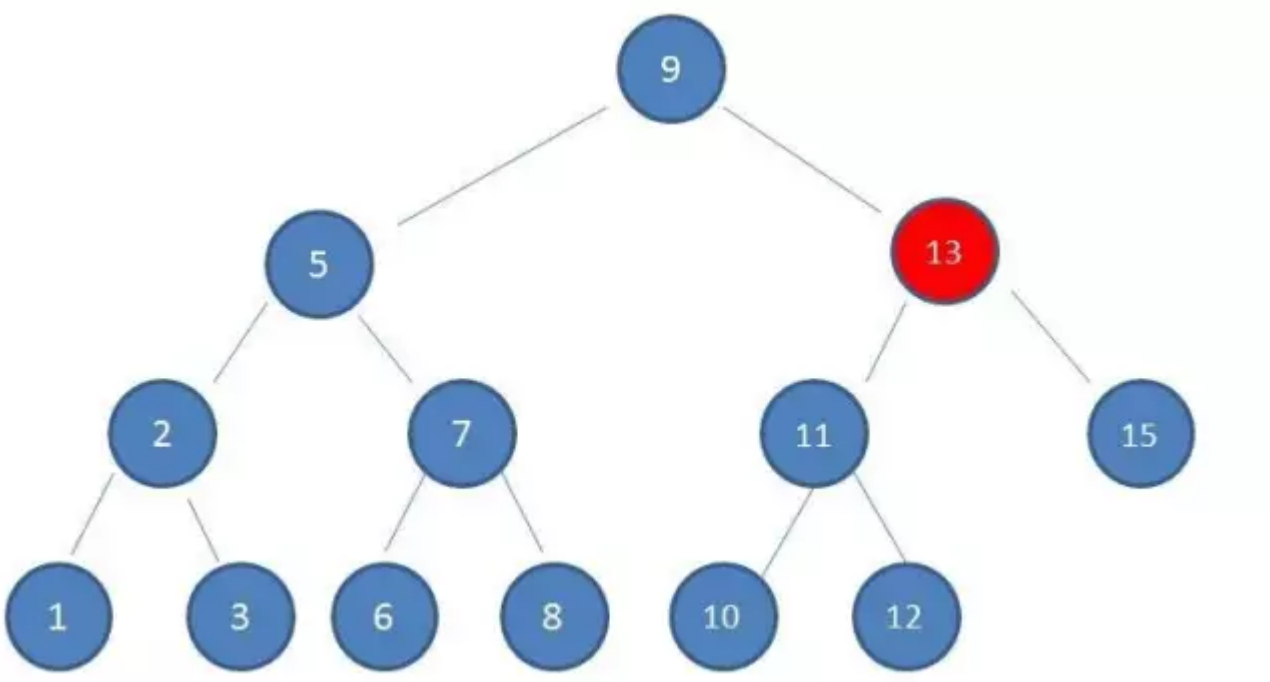
二叉查找树的结构：



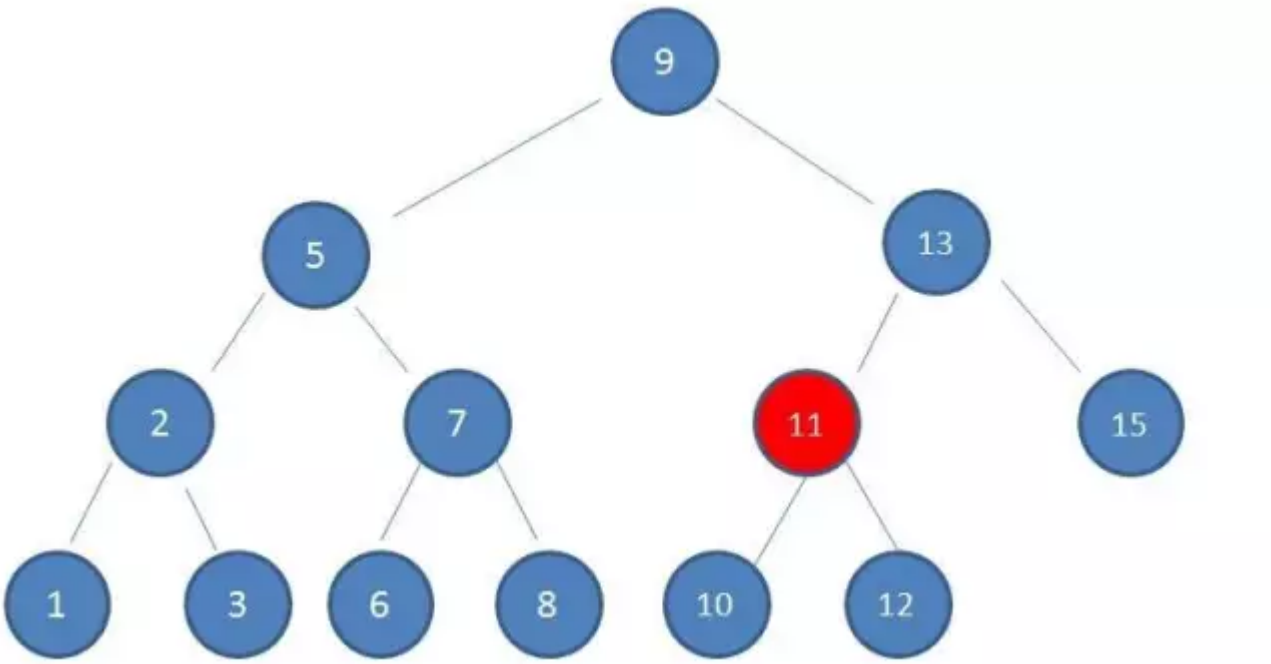
第1次磁盘IO：



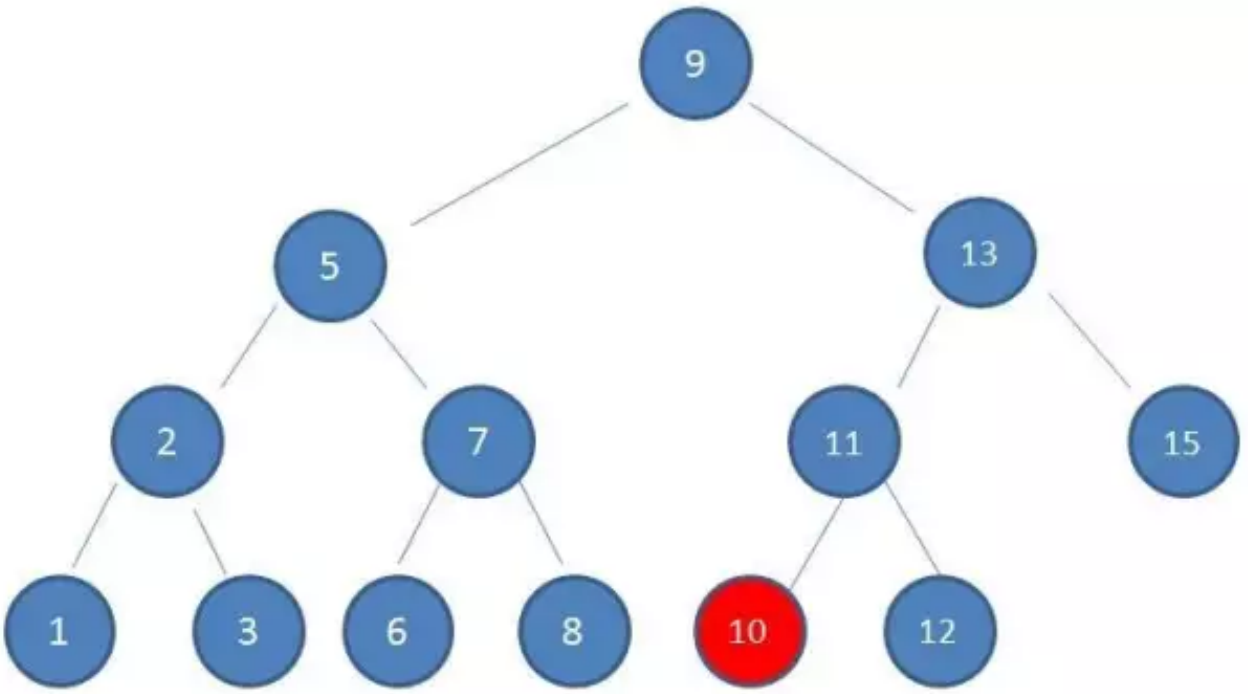
第2次磁盘IO：



第3次磁盘IO：



第4次磁盘IO：



小灰，有没有从中看出，磁盘 IO 的次数是由什么决定？

磁盘 IO 的次数是 4 次，索引树的高度也是 4。所以最坏情况下，磁盘 IO 次数等于索引树的高度。

没错，既然如此，为了减少磁盘 IO 次数，我们就需要把原本“瘦高”的树结构变得“矮胖”。这就是 B- 树的特征之一。

B 树是一种多路平衡查找树，它的每一个节点最多包含 k 个孩子，k 被称为 B 树的阶。k 的大小取决于磁盘页的大小。

下面来具体介绍一下B-树（Balance Tree），一个m阶的B树具有如下几个特征：

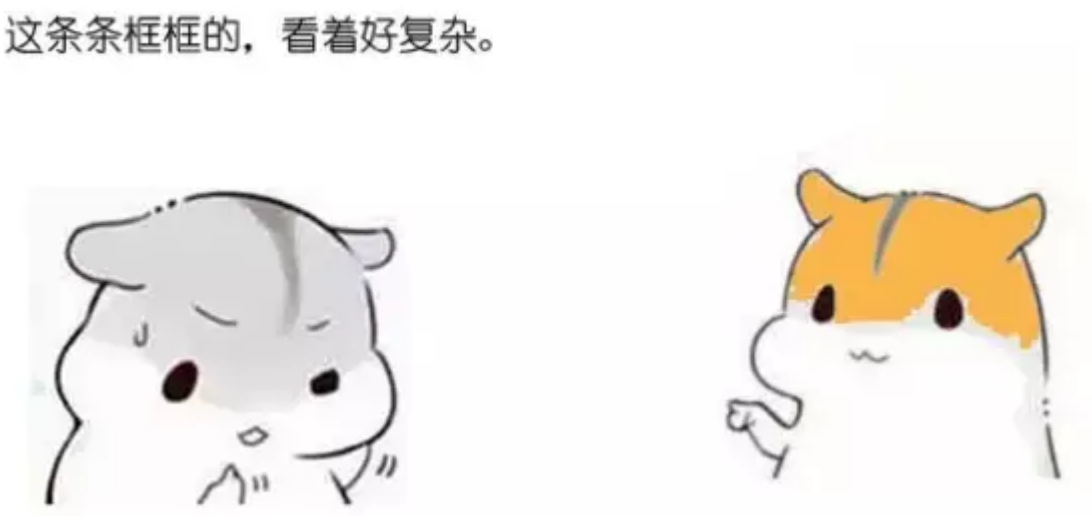
1.根结点至少有两个子女。

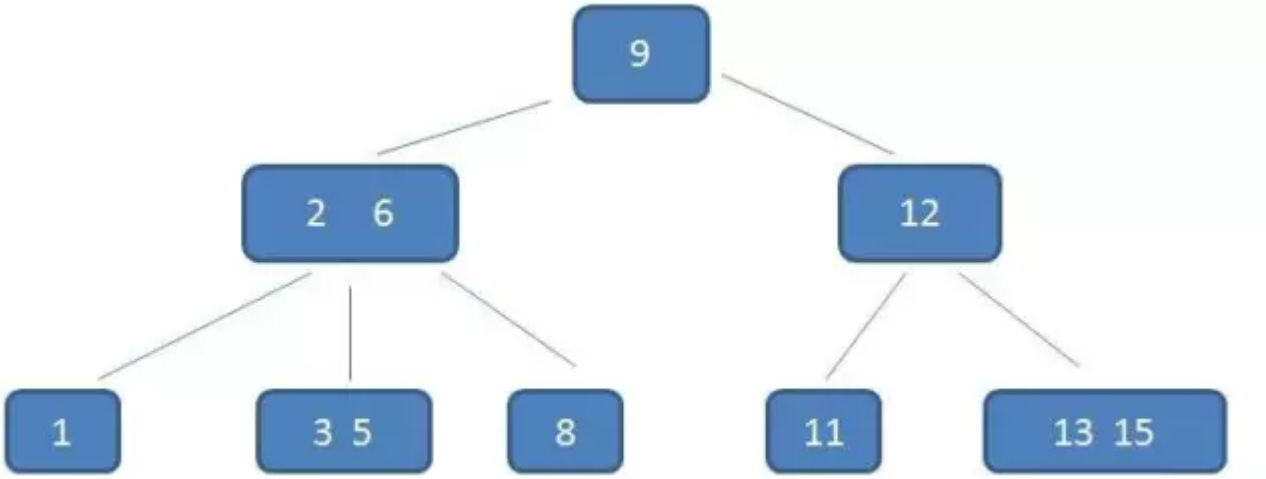
2.每个中间节点都包含k-1个元素和k个孩子，其中 m/2 <= k <= m

3.每一个叶子节点都包含k-1个元素，其中 m/2 <= k <= m

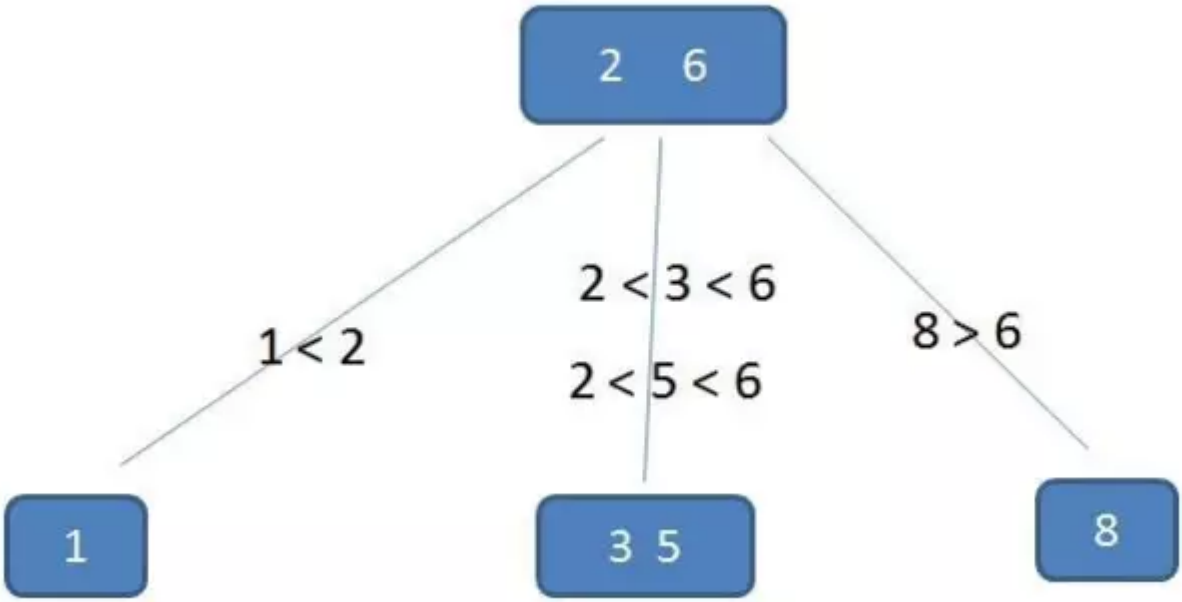
4.所有的叶子结点都位于同一层。

5.每个节点中的元素从小到大排列，节点当中k-1个元素正好是k个孩子包含的元素的值域分划。



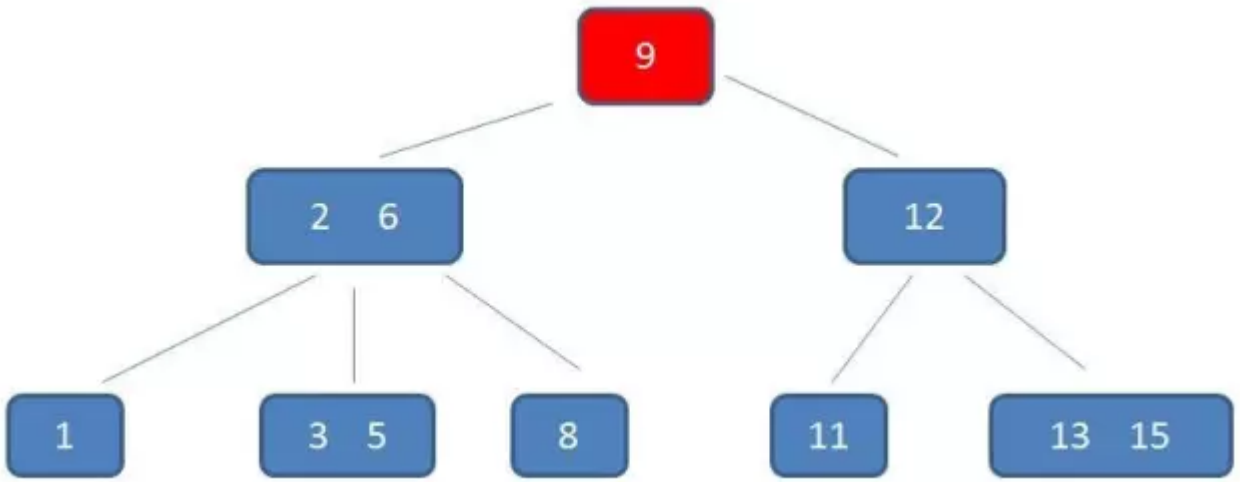


这棵树中，咱们重点来看 (2, 6) 节点。该节点有两个元素 2 和 6，又有三个孩子 1，(3, 5)，8。其中 1 小于元素 2，(3，5) 在元素 2, 6 之间，8 大于 (3, 5)，正好符合刚刚所列的几条特征。

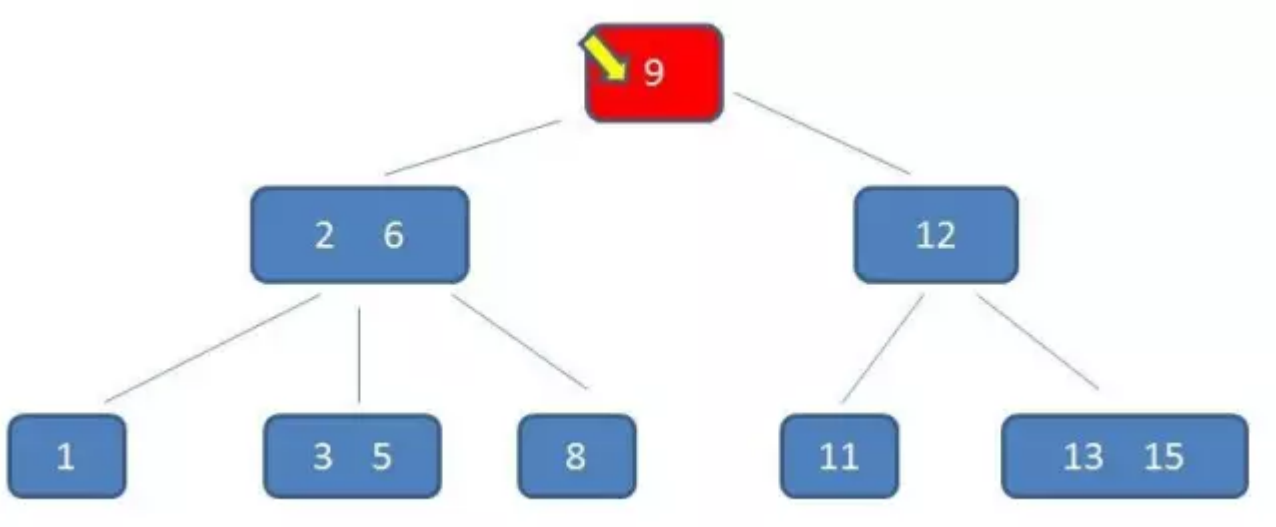


这棵树长得可真奇怪，它真的能实现高效的查询吗？

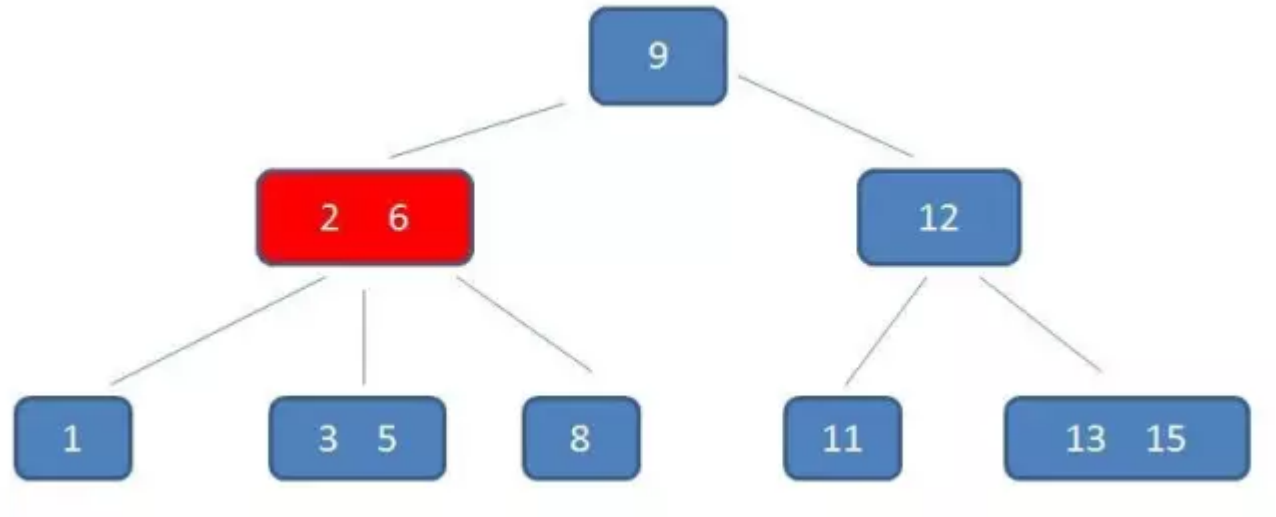
第1次磁盘IO：



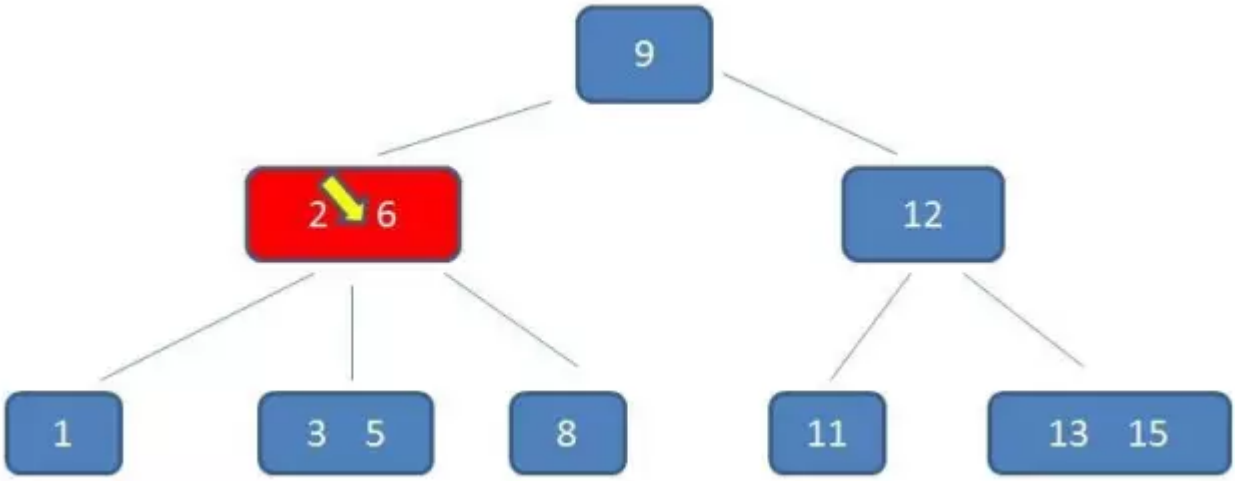
在内存中定位（和9比较）：



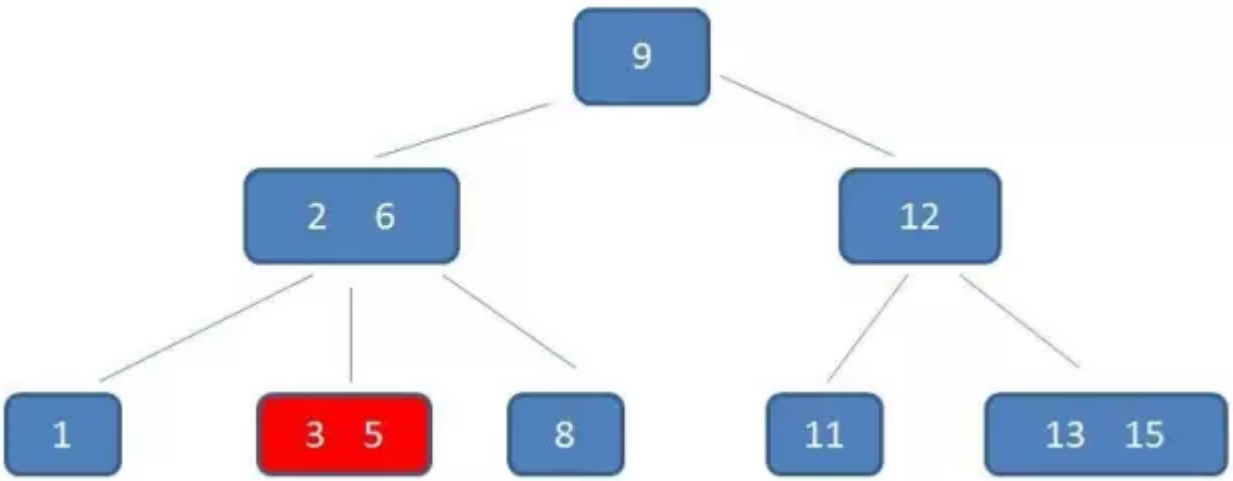
第2次磁盘IO：



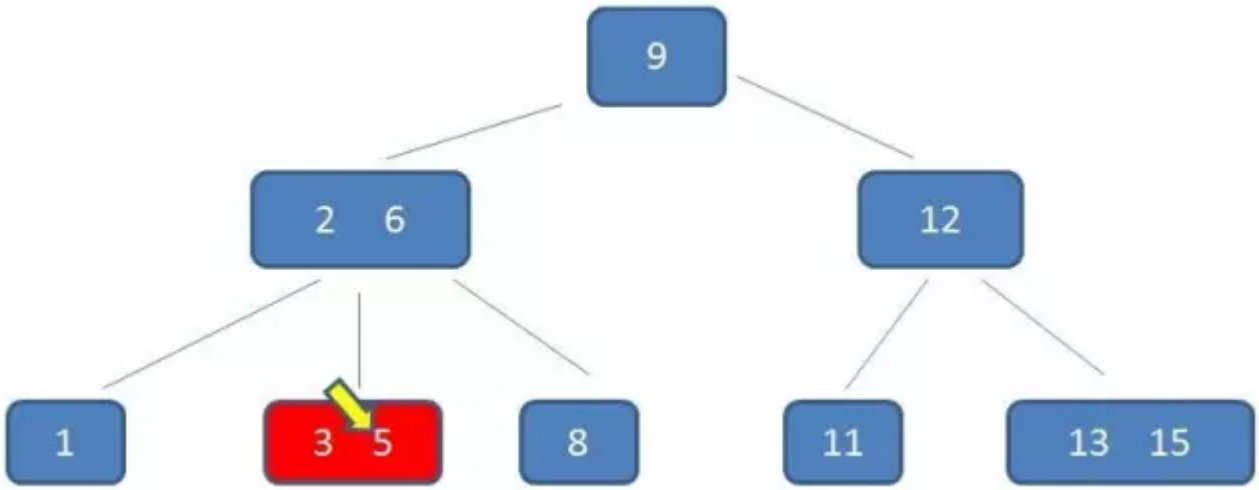
在内存中定位（和2，6比较）：



第3次磁盘IO：



在内存中定位（和3，5比较）：



通过整个流程我们可以看出，B- 树在查询中的比较次数其实不比二叉查找树少，尤其当单一节点中的元素数量很多时。

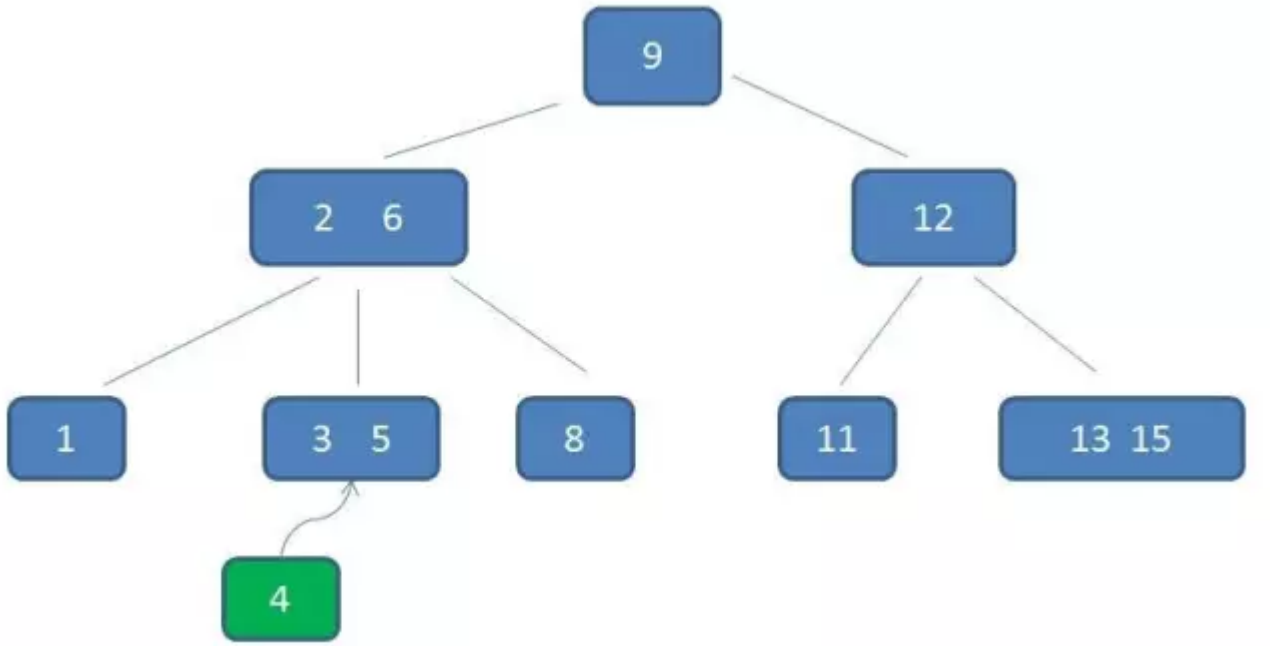
可是相比磁盘 IO 的速度，内存中的比较耗时几乎可以忽略。所以只要树的高度足够低，IO 次数足够少，就可以提升查找性能。

相比之下节点内部元素多一些也没有关系，仅仅是多了几次内存交互，只要不超过磁盘页的大小即可。这就是 B- 树的优势之一。

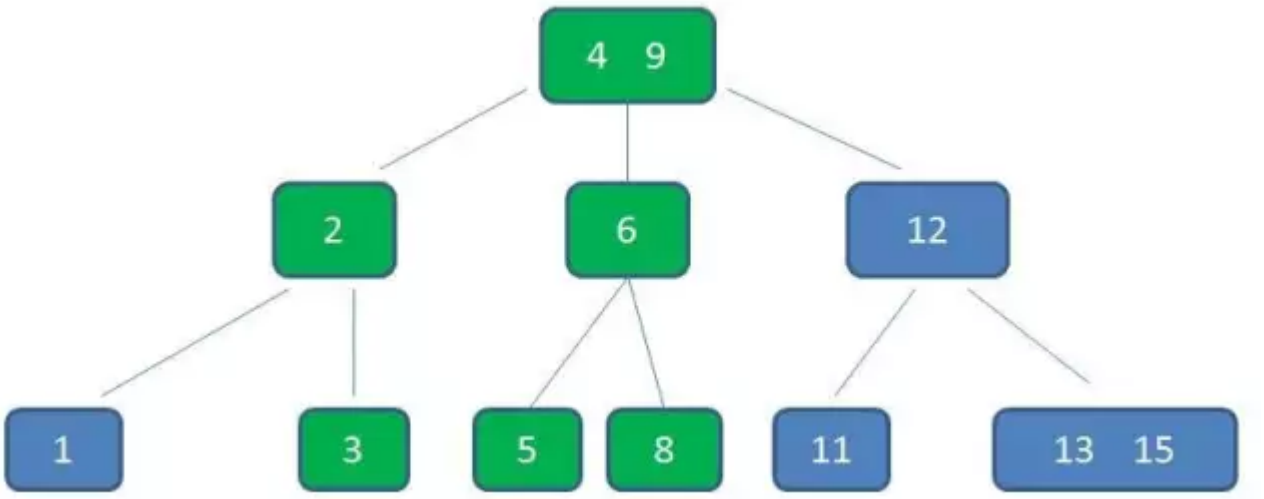
原来如此，查询的过程明白了。那么 B- 树是如何做插入和删除的呢？

B- 树插入新节点的过程比较复杂，而且分成很多种情况。由于时间关系，我们只举一个最典型的例子，加入我们要插入的值是 4

自顶向下查找4的节点位置，发现4应当插入到节点元素3，5之间。



节点3，5已经是两元素节点，无法再增加。父亲节点 2， 6 也是两元素节点，也无法再增加。根节点9是单元素节点，可以升级为两元素节点。于是**拆分**节点3，5与节点2，6，让根节点9升级为两元素节点4，9。节点6独立为根节点的第二个孩子。

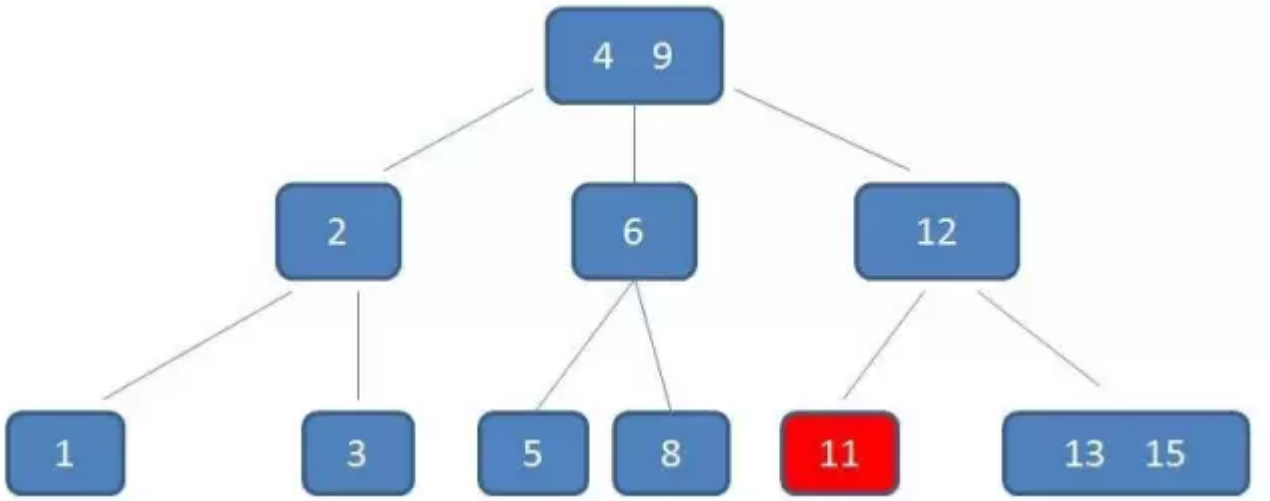


哎妈呀，就为了插入一个元素，让整个 B 树的那么多节点都发生了连锁改变。

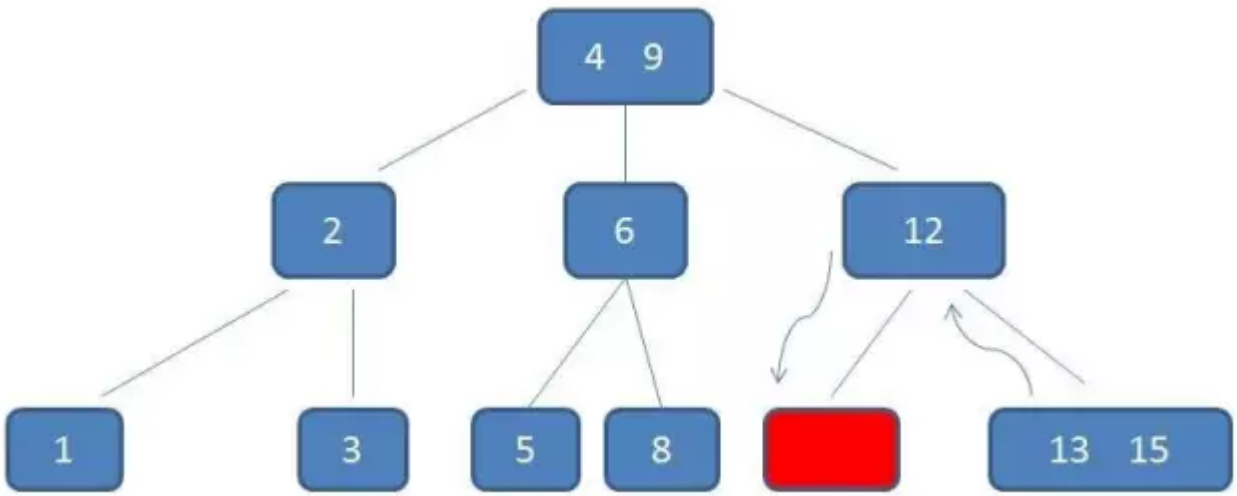
确实有些麻烦，但也正因为如此，让 B- 树能够始终维持多路平衡。这也是 B- 树的一大优势：自平衡。

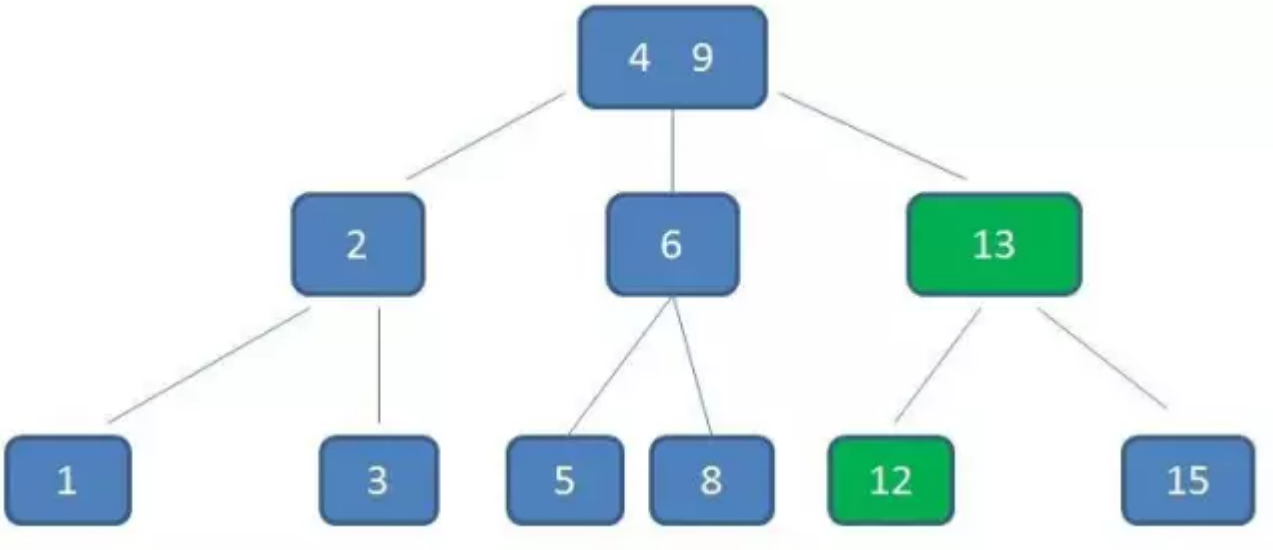
下面我们再来说一说 B- 树删除的过程。同样只举一个典型例子，删除元素 11

自顶向下查找元素11的节点位置。



删除11后，节点12只有一个孩子，不符合B树规范。因此找出12,13,15三个节点的中位数13，取代节点12，而节点12自身下移成为第一个孩子。（这个过程称为**左旋**）





以上就是 B- 树的插入和删除，也是学习 B- 树最绕的部分。没太看明白的小伙伴也不必心急，最关键的还是理解 B- 树的核心思想。

那么，B- 树都有哪些实际应用呢？

B- 树主要应用于文件系统以及部分数据库索引，比如著名的非关系型数据库 MongoDB。

而大部分关系型数据库，比如 Mysql，则使用 B+ 树作为索引。有关 B+ 树的知识，下一次我再做具体介绍吧。

漫画算法系列

- 漫画算法：最小栈的实现
- 漫画算法：判断 2 的乘方
- 漫画算法：找出缺失的整数
- 漫画算法：辗转相除法是什么鬼?
- 漫画算法：什么是动态规划？（整合版）
- 漫画算法：什么是跳跃表?

觉得本文有帮助？请分享给更多人

关注「算法爱好者」，修炼编程内功

算法爱好者

专注算法相关内容

微信号：AlgorithmFans

长按识别二维码关注

伯乐在线 旗下微信公众号

商务合作QQ：2302462408