MySQL数据库索引数据结构：
主要为hash表和B+树，部分非关系型数据库为B-树

hash索引：

```
hash索引优势：hash索引就是采用一定的哈希算法，将键值换算成新的哈希值，
检索时不需要类似B+树那样从根节点到叶子结点逐级查找，只需要一次哈希算法
即可立刻定位到相应位置，速度非常快。

不足：
1.hash索引仅仅能满足"="和"<=>"等职查询，不能使用范围查询。
如果是等值查询，那么哈希索引有绝对优势，因为只需要经过一次算法即可找到
相应的键值；但前提是，键值都是唯一的，若键值不唯一，就需要先找到该键所
在
位置，然后再根据链表往后扫描，直到找到相应的数据。由于Hash索引比较的是
进行Hash运算之后的Hash值，所以它只能用于等值的过滤，不能用于基于范围
的
过滤，因为经过相应的Hash算法处理之后的Hash值的大小关系，并不能保证和
Hash运算钱一样。
2.Hash索引无法被用来进行数据的排序操作。
由于Hash索引中存放的是经过Hash计算之后的 Hash 值，而且Hash值的大小关
系
并不一定和 Hash 运算前的键值完全一样，所以数据库无法利用索引的数据来
避免任何排序运算。
3.Hash 索引不支持多列联合索引的最左匹配规则
对于组合索引，Hash 索引在计算 Hash 值的时候是组合索引键合并后再一起
计算 Hash 值，而不是单独计算 Hash 值，所以通过组合索引的前面一个或
几个索引键进行查询的时候，Hash 索引也无法被利用。
4.Hash 索引在任何时候都不能避免表扫描
Hash 索引是将索引键通过 Hash 运算之后，将 Hash运算结果的 Hash 值和
所对应的行指针信息存放于一个 Hash 表中，由于不同索引键存在相同 Hash
值，所以即使取满足某个 Hash 键值的数据的记录条数，也无法从 Hash 索引
```

中直接完成查询，还是要通过访问表中的实际数据进行相应的比较，并得到相应的结果。

5.B+树索引的关键字检索效率比较平均，不像B树那样波动幅度大，在有大量重复键值情况下，哈希索引的效率也是极低的，因为存在所谓的哈希碰撞问题。

B树和B+树：

1.数据库索引为什么要使用树结构存储？

原因是树的查询效率高，且可以保持有序。

2.那为什么索引没有使用二叉查找树来实现呢？

二叉查找树时间复杂度O(logN)。

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。但需要考虑磁盘IO。数据库索引是存储在磁盘上的，当数据量比较大时，索引的大小可能有几个G甚至更多。当我们利用索引查询时，不可能将整个索引全部加载到内存中，能做的只有逐一加载每一个磁盘页，这里的磁盘页对应索引树的节点。

注意：若使用二叉搜索树，查询时间复杂度不稳定，最好一次就可以找到，最坏与树的高度成正比。当二叉搜索树深度很深时，查询效率降低。

因此针对二叉搜索树的深度问题，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“。--B树的特征之一

B-树：

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。

一个m阶的B-树的特征：

a.根节点至少有两个子女。

b.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

c.每一个叶子结点都包含k-1个元素，其中m/2<=k<=m

d.所有的叶子结点都位于同一层

e.每个节点中的元素从小到大排序，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布。

25

26 查询：

27 B-树如下图所示，通过整个流程可以看到，B-树在查询中的比较次数其实不必二叉

28 查找树少，尤其当单一节点中的元素数量很多时。但是相比磁盘IO的速度，内存中

29 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

30 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

31 要不超过磁盘页的大小即可。--B-树的优势之一

32

33 插入：

34 插入操作是插入(key,value)的键值对，如果B树中已存在需要插入的键值对，则更新

35 value值。若B树中不存在这个key，则一定是在叶子结点中进行插入操作。

36 1)根据要插入的key值，找到叶子结点并插入；

37 2)判断当前节点key的个数是否小于等于m-1，若满足则结束，否则进行第三步

38 3)以节点中间的key为中心分裂为左右两部分，然后将这个中间的key值插入到父节点

39 中，这个key的做字数指向分裂后的左半部分，右子树指向分裂后的右半部分，然后将

40 当前节点指向父节点，继续进行第三步。

41 插入图解如下：

42 注意：虽然B树插入元素可能会引起多个节点变动，但也正是如此，才使得

43 B-树能够始终维持多路平衡。--B-树的优势之一：自平衡

44

45 删除：

46 删除操作是根据key删除记录，若B树中没有该记录，则删除失败。

47 1)如果当前需要删除的key位于非叶子结点，则用后继key覆盖要删除的key，然后

48 在后继key所在的子支中删除该后继key。此时后继key一定位于叶子结点，这个过程

49 和二叉搜索树删除节点的方式类似。删除这个记录后执行第2步；

50 2)该节点key个数大于等于Math.ceil(m/2-1),则父节点中的key下移到该节点，

51 兄弟节点中的一个key上移，删除操作结束。否则，将父节点中的key下移与当前

52 节点及它的兄弟节点中的key合并，形成一个新的节点。原父节点中的key的两个孩子

53 指针就变成了一个孩子指针，指向这个新节点。然后当前节点的指针指向父节点，

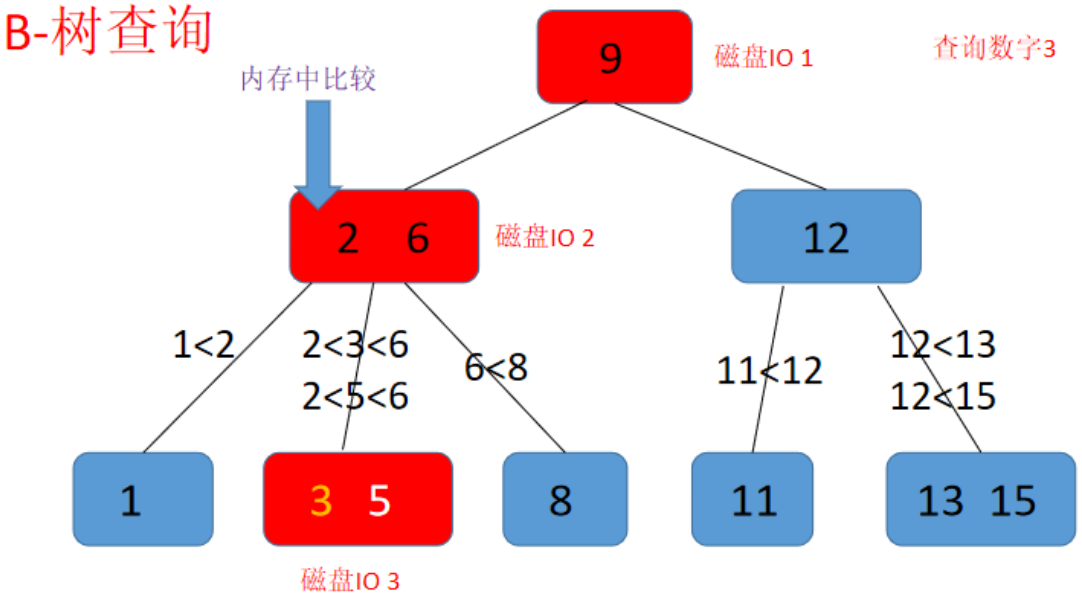
54 重复上第2步。

55 删除图解如下：

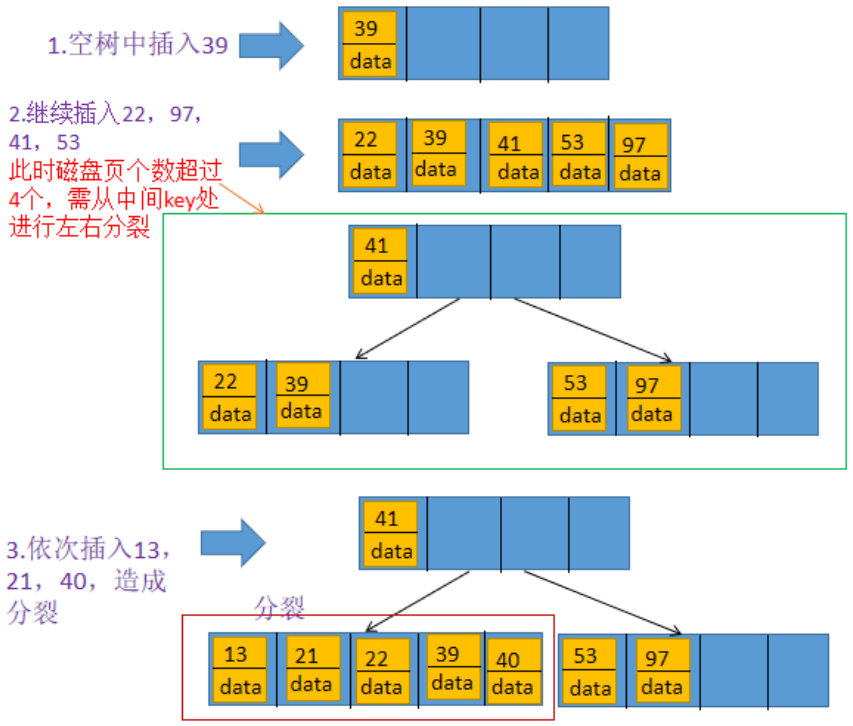
56

57 B-树的实际应用：

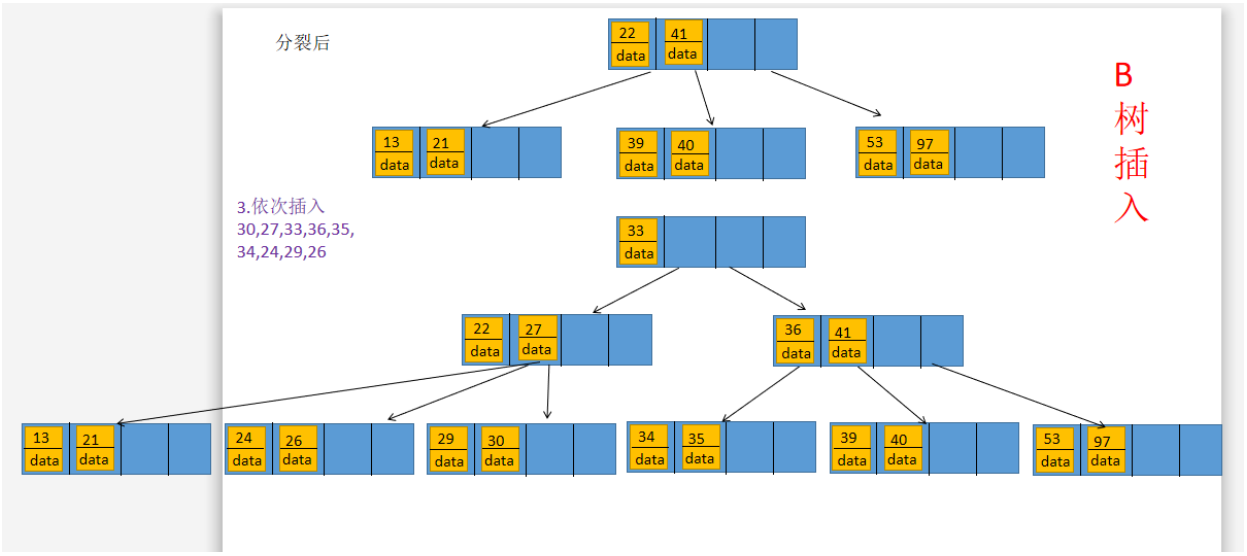
58 B-树主要应用于文件系统以及部分数据库索引，比如注名的非关系型苏杭句酷MongoDB



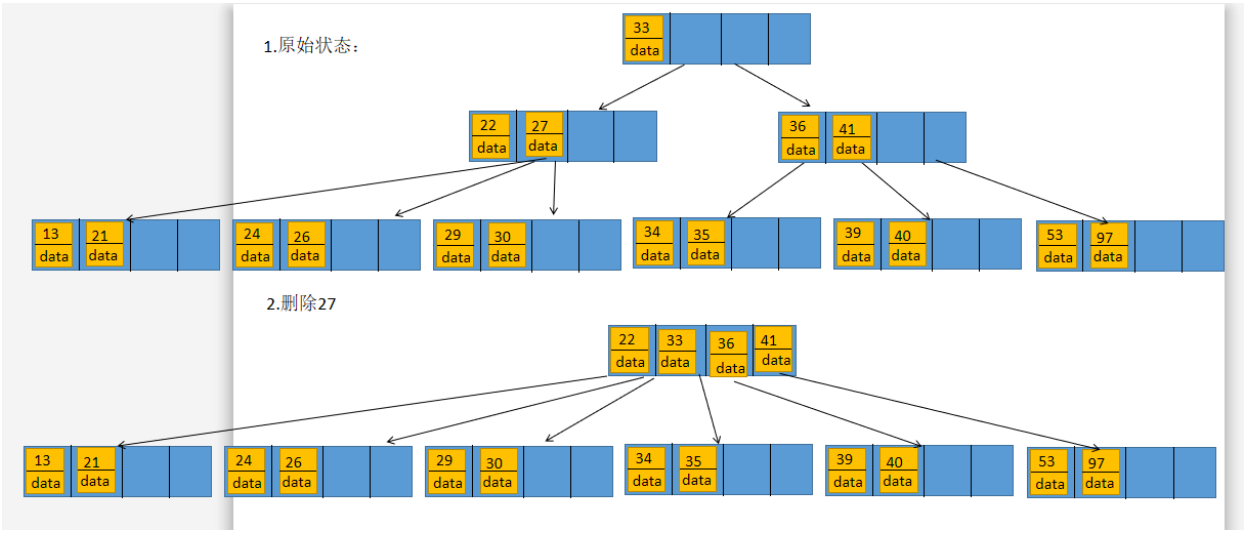
以5阶B树为例，进行插入操作，任一节点最多为4个，最少为2个



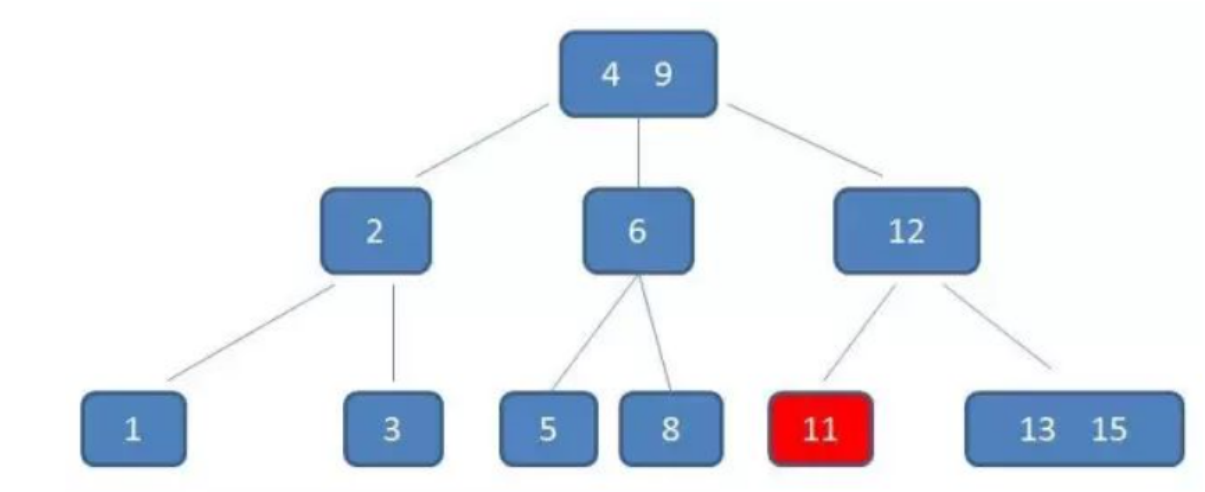
B树插入



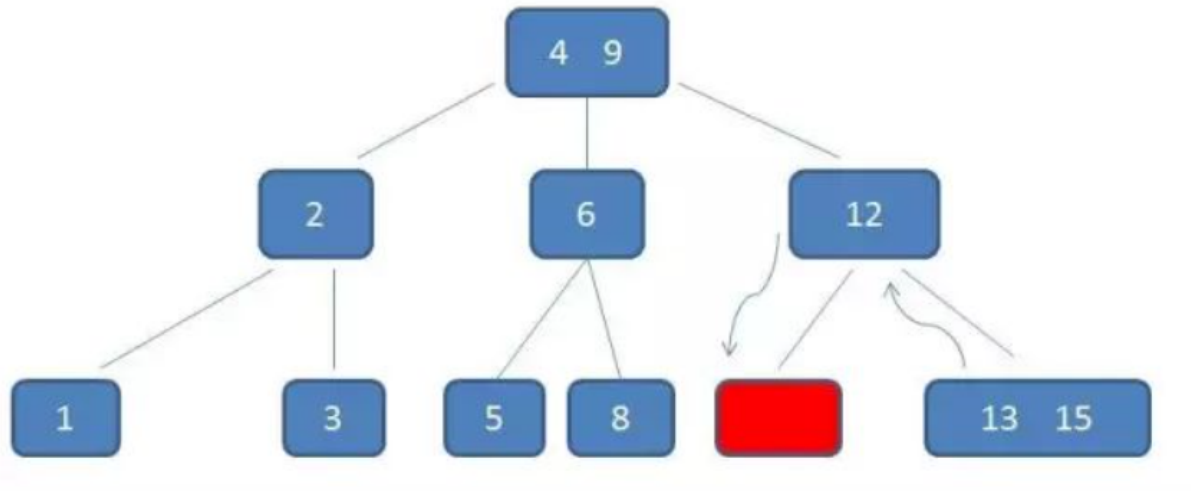
B树插入

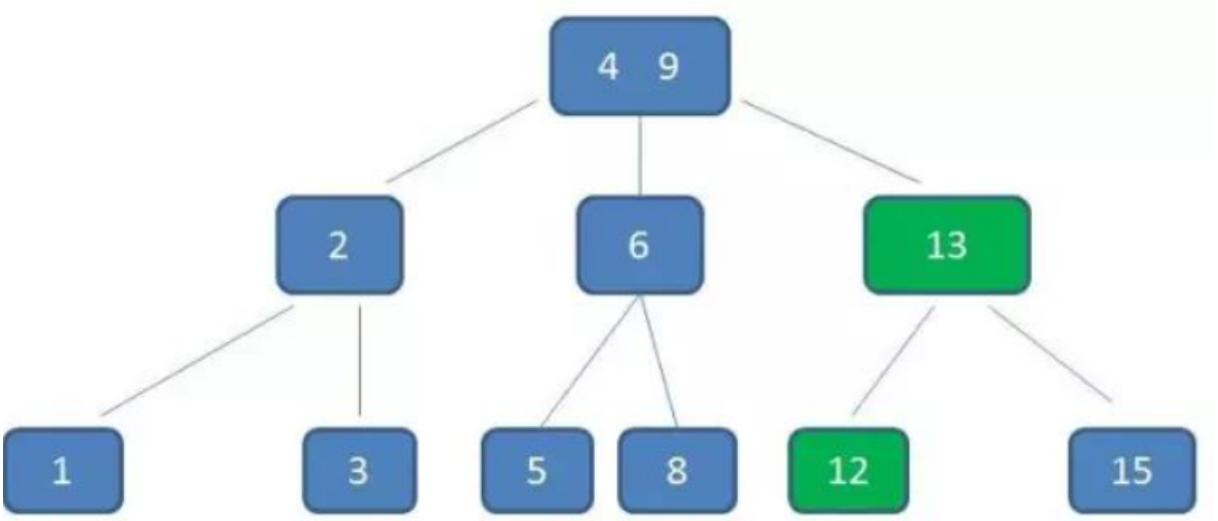


删除11：



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**）





B+树：---基于B-树的一种变体，比B-树查询性能更快

大部分关系型数据库，比如Mysql，使用B+树作为索引

一个m阶的B+树特征：

1.有k个子树的中间节点包含有k个元素(B树中是k-1个元素)，每个元素不保存数据，

只用来索引，所有数据都保存在叶子结点。

2.所有的叶子节点中包含了全部元素的信息，及指向含这些元素记录的指针，且叶子

节点本身依关键字的大小自小而大顺序链接。

3.所有的中间节点元素都同时存在于子节点，在子节点元素中是最大(或最小)元素。

B+树特点：

1.每个父节点的元素都出现在子节点中，是子节点的最大(或最小)元素

2.卫星数据的位置(在索引之外)。所谓卫星数据，指的是索引元素指向的数据记录，

比如数据库中的某一行。在B-树中，无论中间节点还是叶子结点都带有卫星数据。

而在B+树中，只有叶子节点带有卫星数据，其余中间节点仅仅是索引，没有任何数据

关联。

注意：

1.根节点的最大元素，也就等同于整个B+树的最大元素。以后无论插入删除多少元素，始终要保持最大元素在根节点当中。

2.由于父节点的元素都出现在子节点，因此所有叶子节点包含了全量元素信息。并且每一个叶子结点都带有指向下一个节点的指针，形成了一个有序链表。

3.在数据库的聚集索引中，叶子结点直接包含卫星数据；在非聚集索引中，叶子结点带有指向卫星数据的指针。

B+树优点：主要体现在查询性能上。

单行查询：B+树会自顶向下逐层查找结点，最终找到匹配的叶子结点。

单行查询B+与B-树区别：

1.B+树的中间节点没有卫星数据，所以同样大小的磁盘页可以容纳更多的节点元素。

这就意味着，数据量相同的情况下，B+树的结构比B-树更加”矮胖“，因此查询时IO次数也更少。

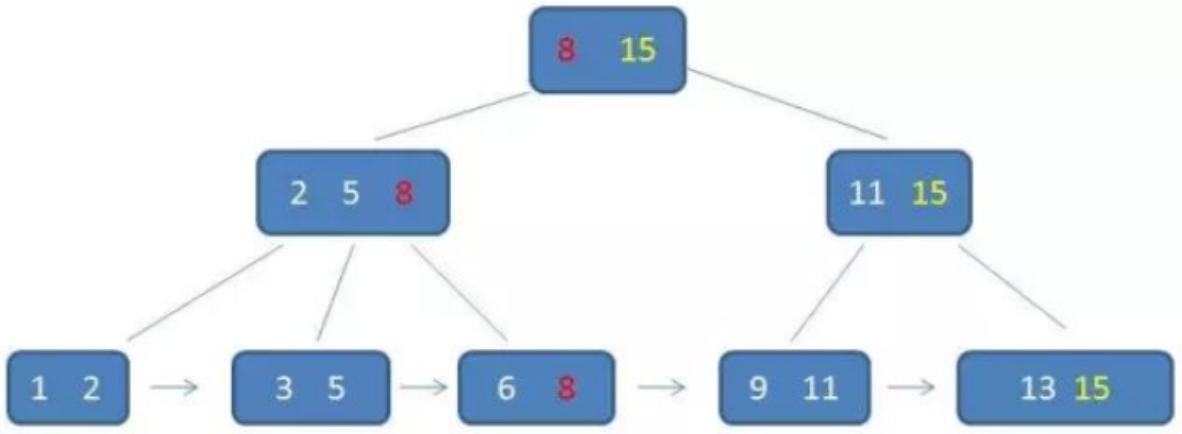
2.B+树的查询必须最终查找到叶子结点，而B-树只要找到匹配元素即可，无论匹配元素处于中间节点还是叶子结点。因此B-树的查找性能并不稳定(最好情况是只查根节点，最坏情况是查到叶子结点)，而B+树的每一次查找都是稳定的。

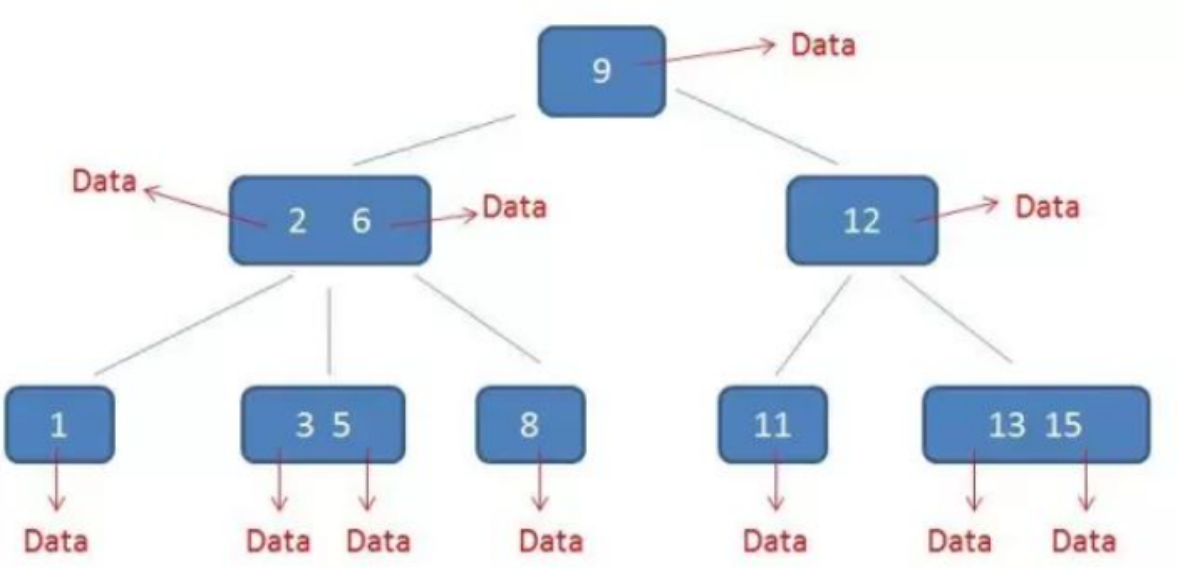
范围查询：

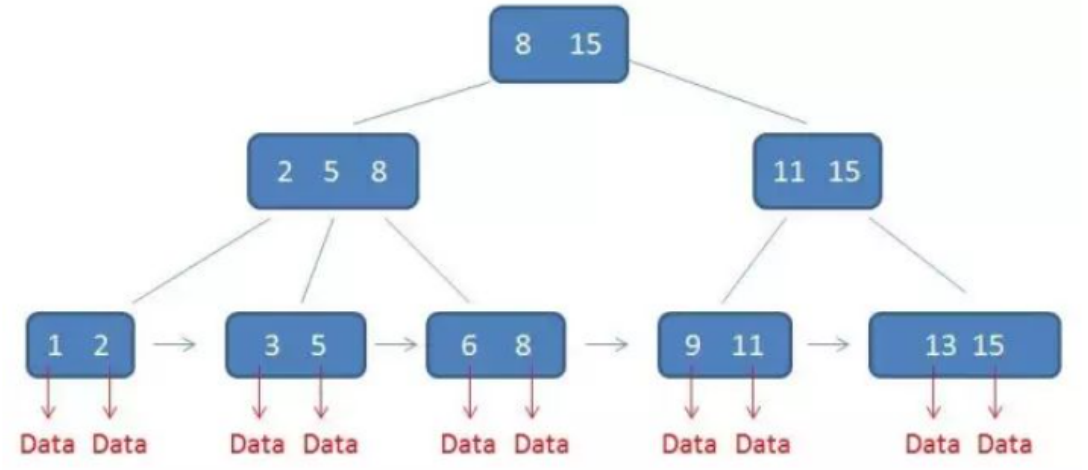
范围查询B+与B-树区别：

B-树依靠中序遍历。如下图查找范围3到11的元素：

B+树只需在链表上遍历即可。如下图：

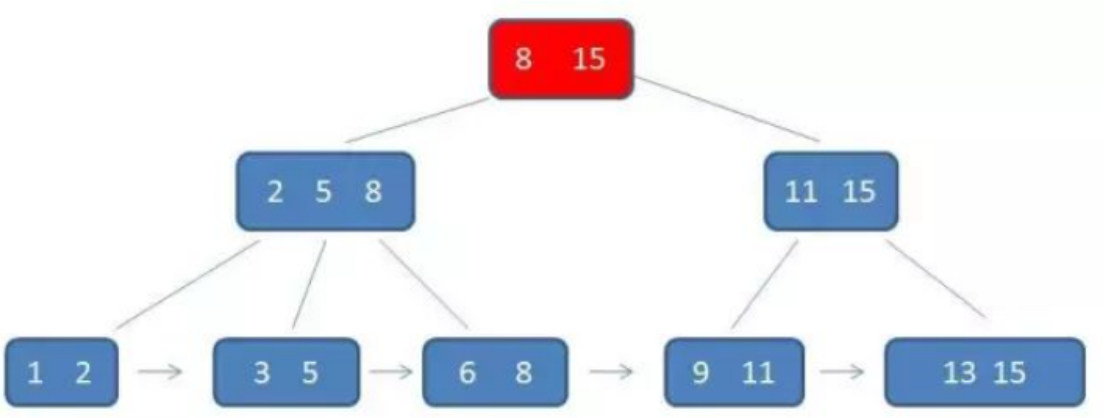



9
Data
Data
2 6
Data
12
Data
1
3 5
8
11
13 15
Data
Data
Data
Data
Data
Data
Data

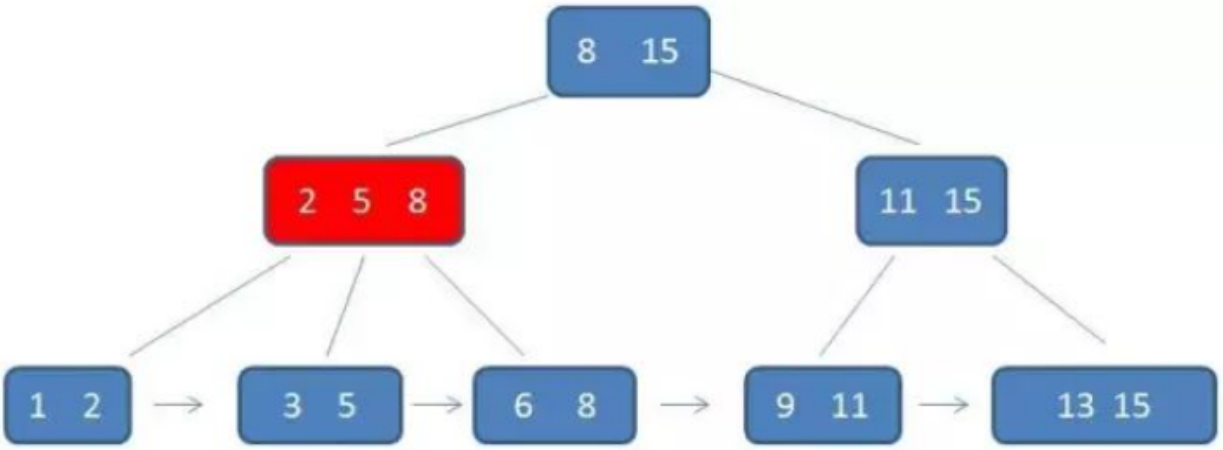

8 15
2 5 8
11 15
1 2
3 5
6 8
9 11
13 15
Data Data
Data Data
Data Data
Data Data
Data Data

单行查询：

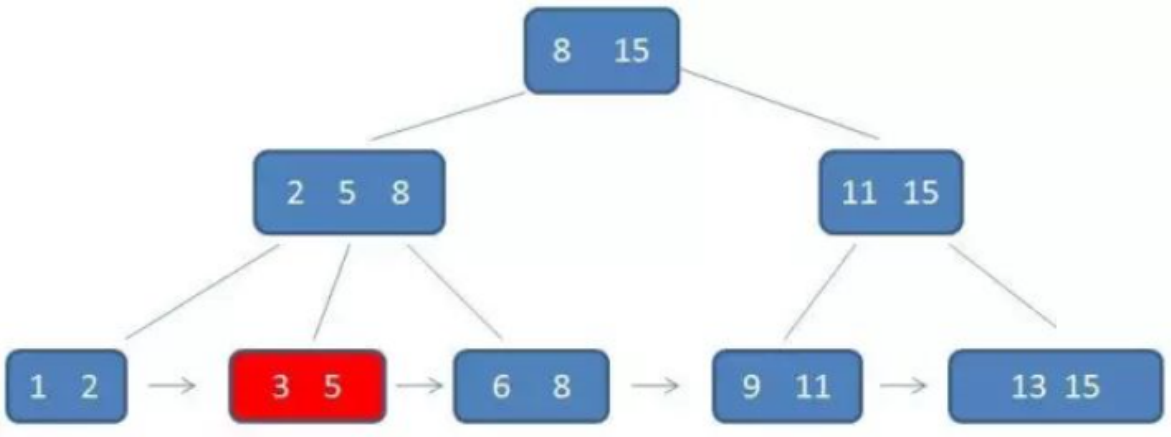
第一次磁盘IO：



第二次磁盘IO：

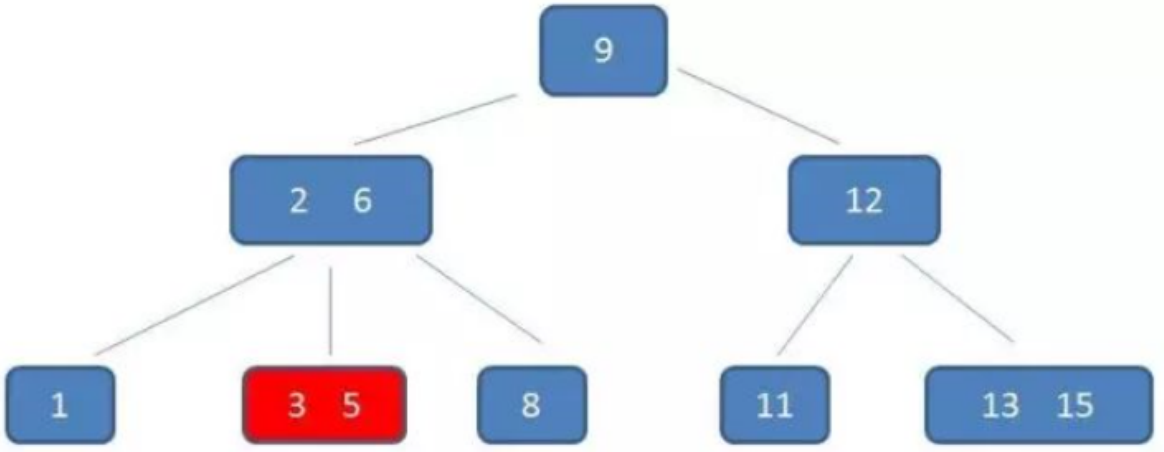


第三次磁盘IO：

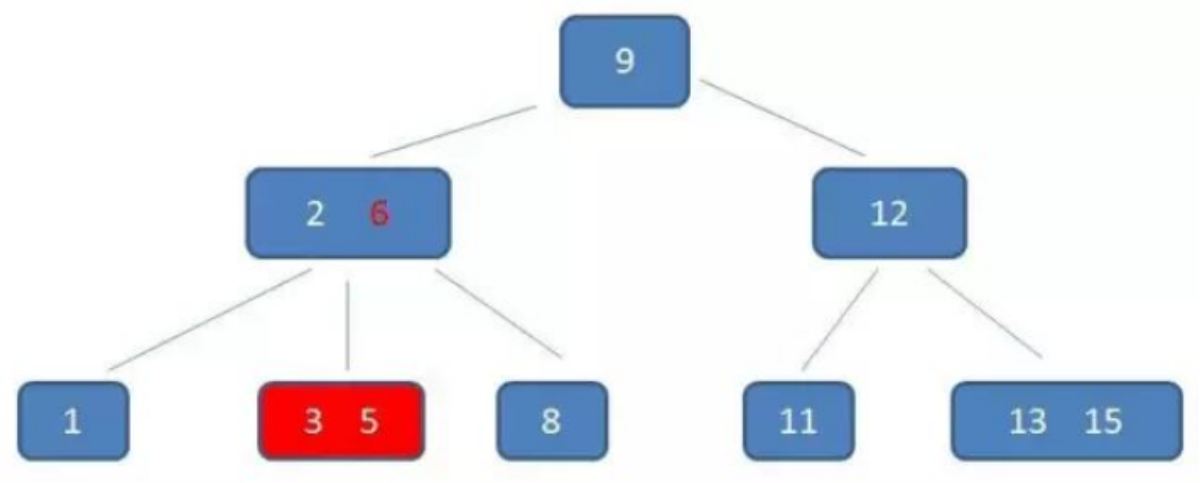


B-树范围查询：中序遍历

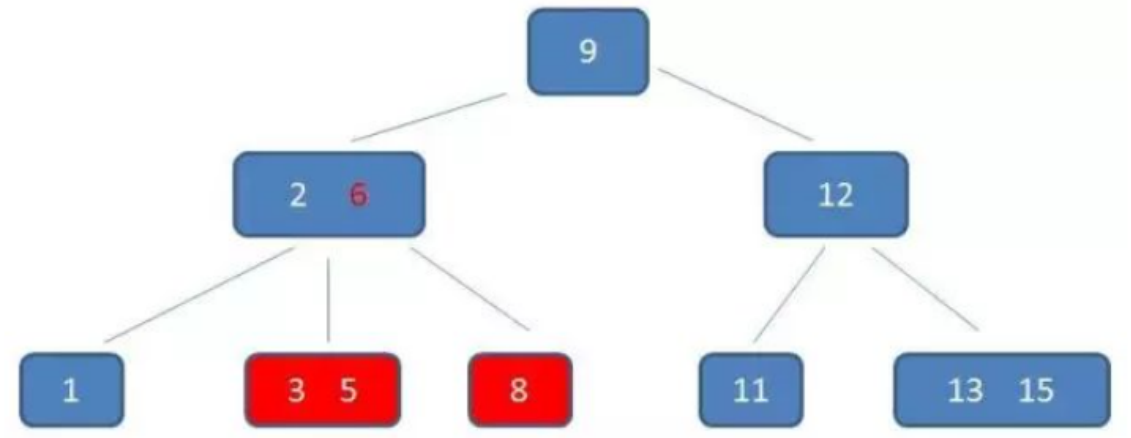
自顶向下，查找到范围的下限（3）：



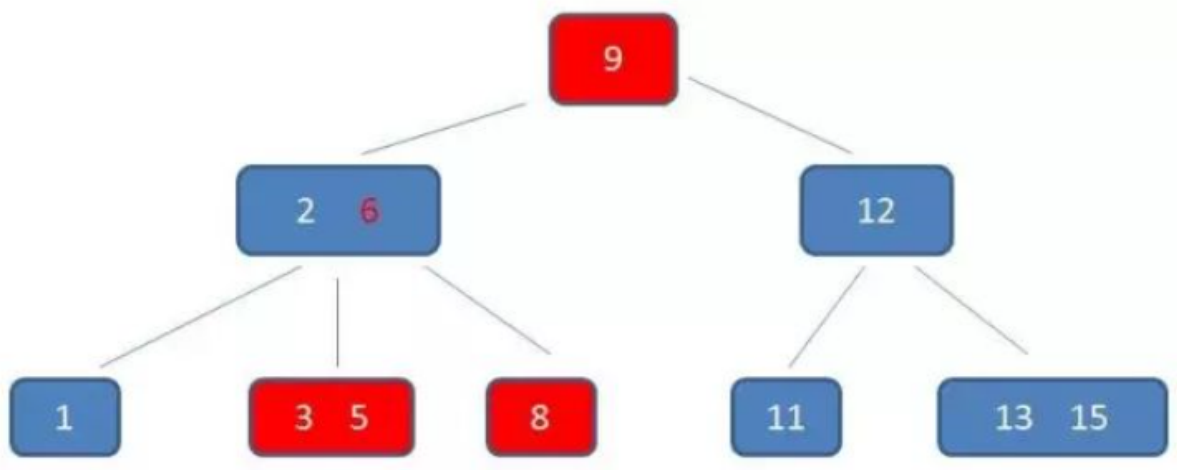
中序遍历到元素6：



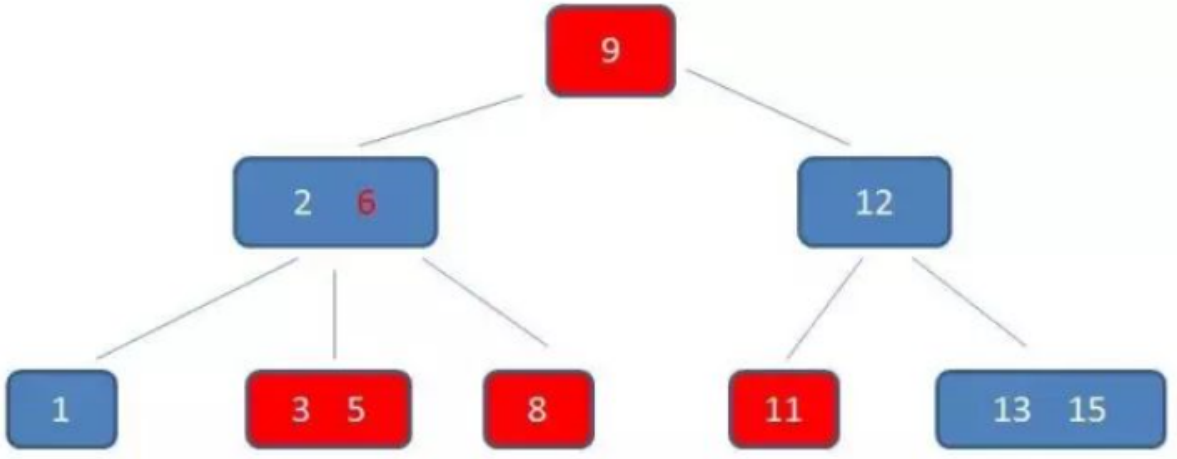
中序遍历到元素8：



中序遍历到元素9：

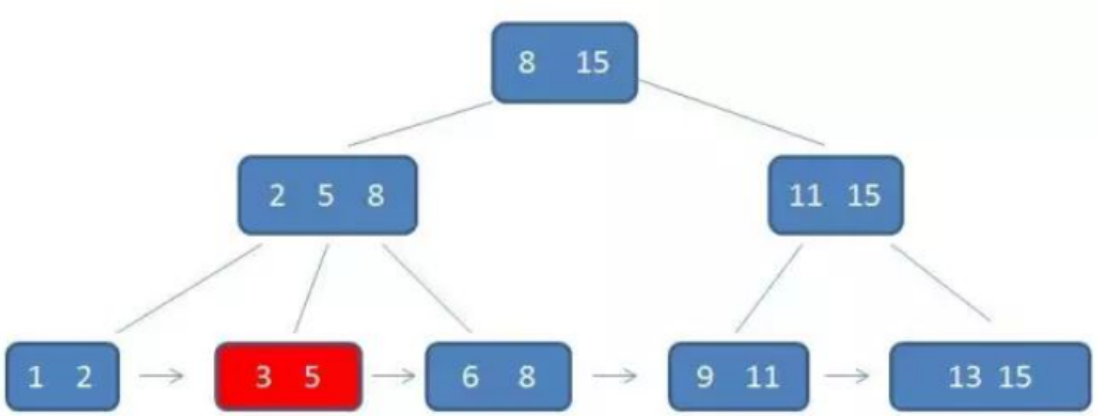


中序遍历到元素11，遍历结束：

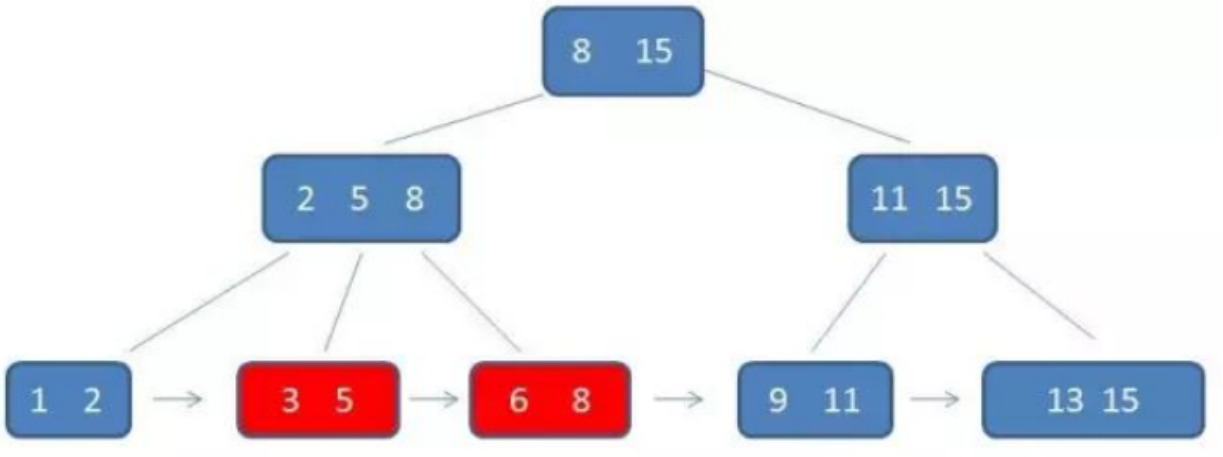


B+树的范围查找过程：

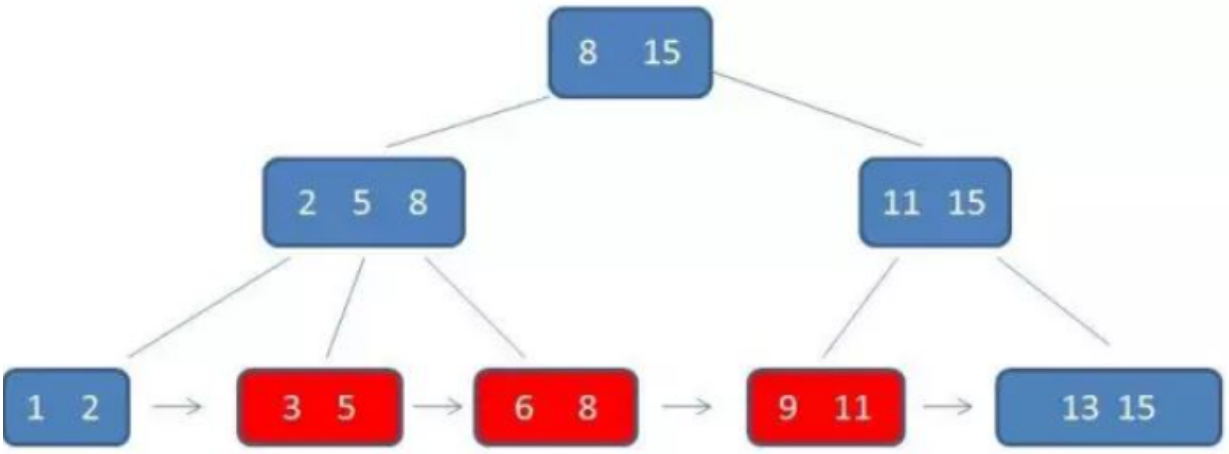
自顶向下，查找到范围的下限（3）：



通过链表指针，遍历到元素6, 8：



通过链表指针，遍历到元素9, 11，遍历结束：



B+树相比B-树的优势有三个：
1.IO次数更少；2.查询性能稳定；3.范围查询简便

B+树的删除与插入与B-树类似。

B+树的优势：
1.单一节点存储更多的元素，是的查询的IP次数更少；