数据库第五次作业

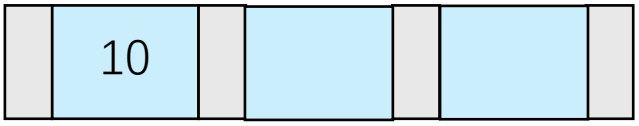
2352018 刘彦

Q1: Construct a B⁺-tree from an empty tree. Each node can hold four pointers

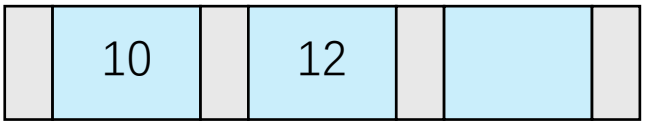
- The sequential values to be inserted are: 10, 12, 27, 5, 9, 15, 30, 7, 17, 26, 19
- Then delete 10 and 15, respectively
- Please give the B⁺ trees after each insertion and each deletion

B⁺树的每个节点有 4 个指针（$n=4$），每个叶节点最多可有$n-1$个值。最少应包含$\lceil\frac{n-1}{2}\rceil$个值。在这个 B⁺树中，每个叶节点必须包含最少 2 个并且最多 3 个值。

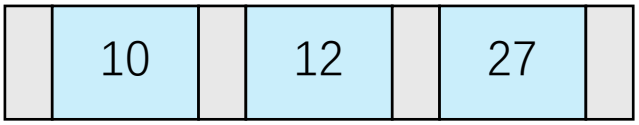
插入 10：



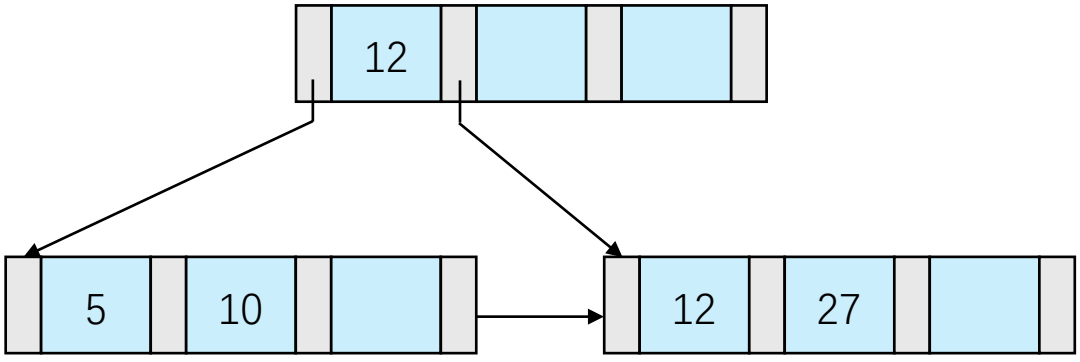
插入 12，可以放下，12 大于 10，插在右侧：



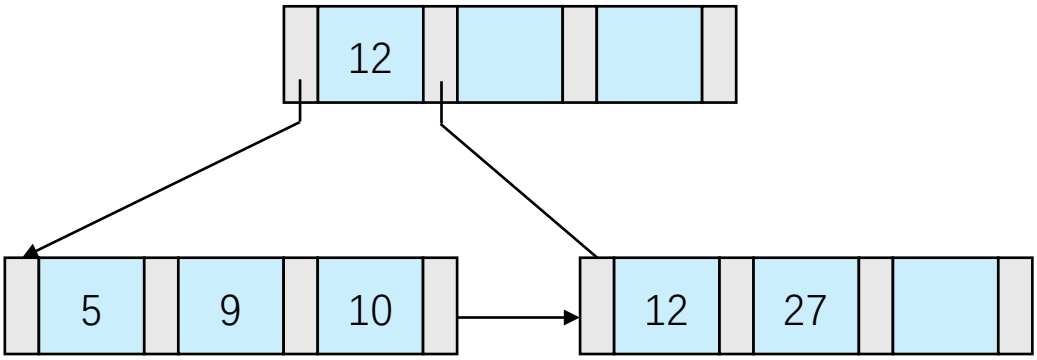
插入 27，可以放下，27 大于 12，插在右侧：



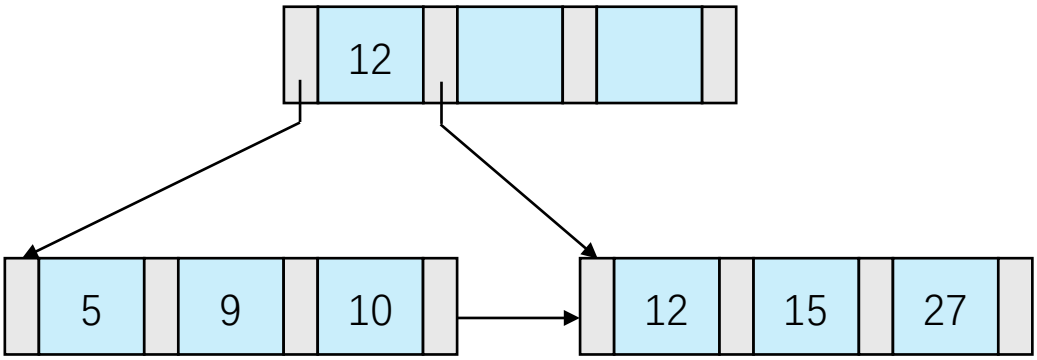
插入 5，5 小于 10，插在左侧，该节点一共 4 个值，放不下，需要分裂：



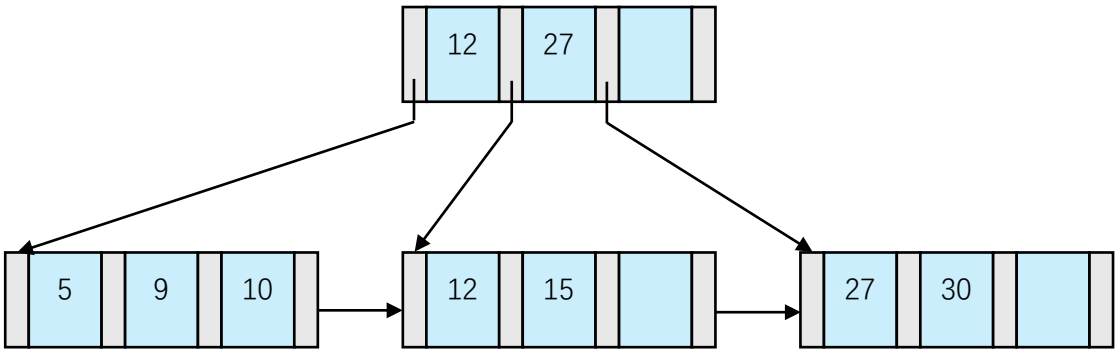
插入 9，可以放下，9 大于 5 小于 10，插在二者之间：



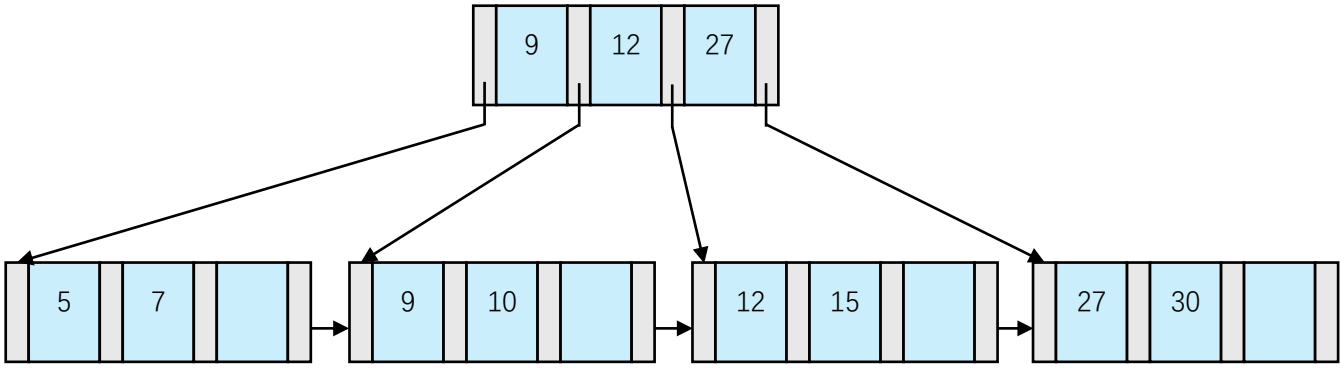
插入 15，可以放下，15 大于 12 小于 27，插在二者之间：



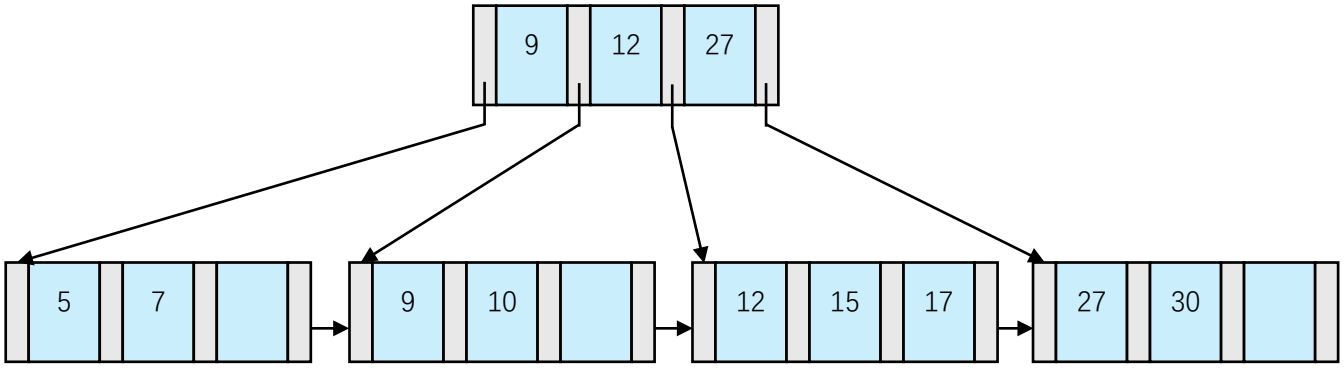
插入 30，30 大于 27，插在右侧，该节点一共 4 个值，放不下，需要分裂：



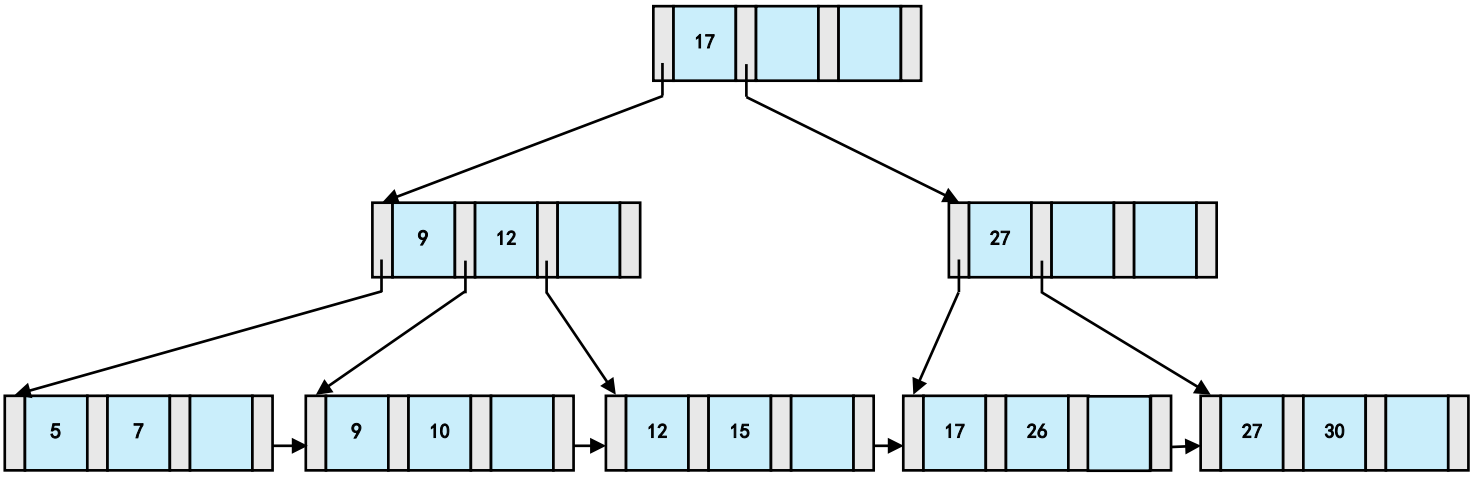
插入 7，7 大于 5 小于 10，插在二者之间，该节点一共 4 个值，放不下，需要分裂：



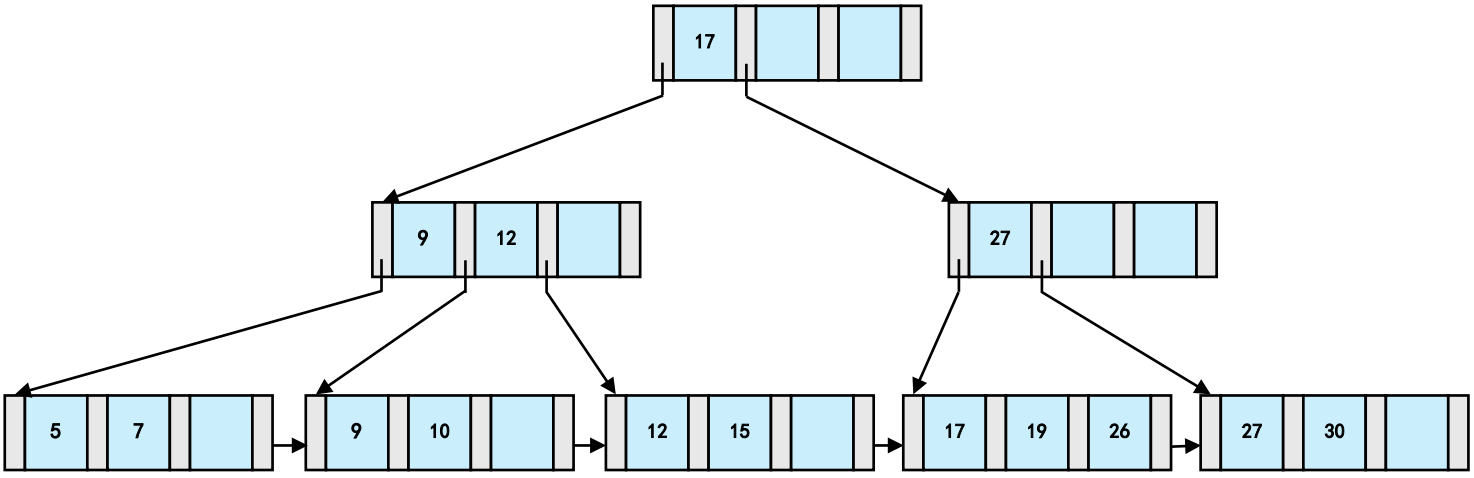
插入 17，可以放下，17 大于 15，插在右侧：



插入 26，26 大于 17，插在右侧，该节点一共 4 个值，放不下，需要分裂，分裂后根节点也有 4 个值，放不下，继续分裂：

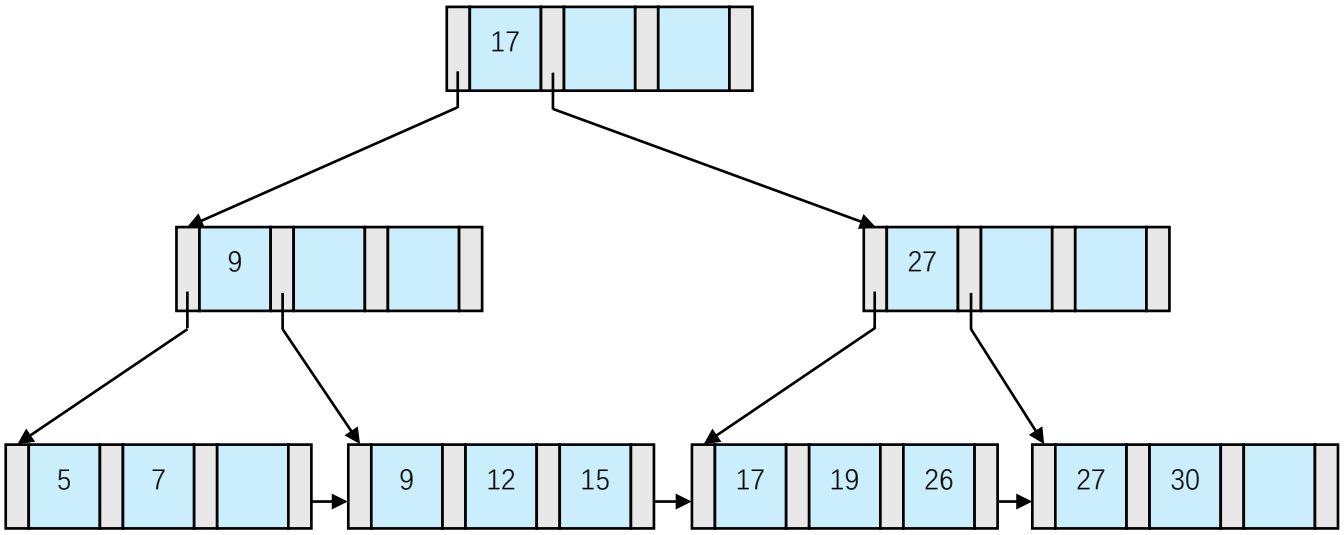


插入 19，可以放下，19 大于 17 小于 26，插在二者之间：

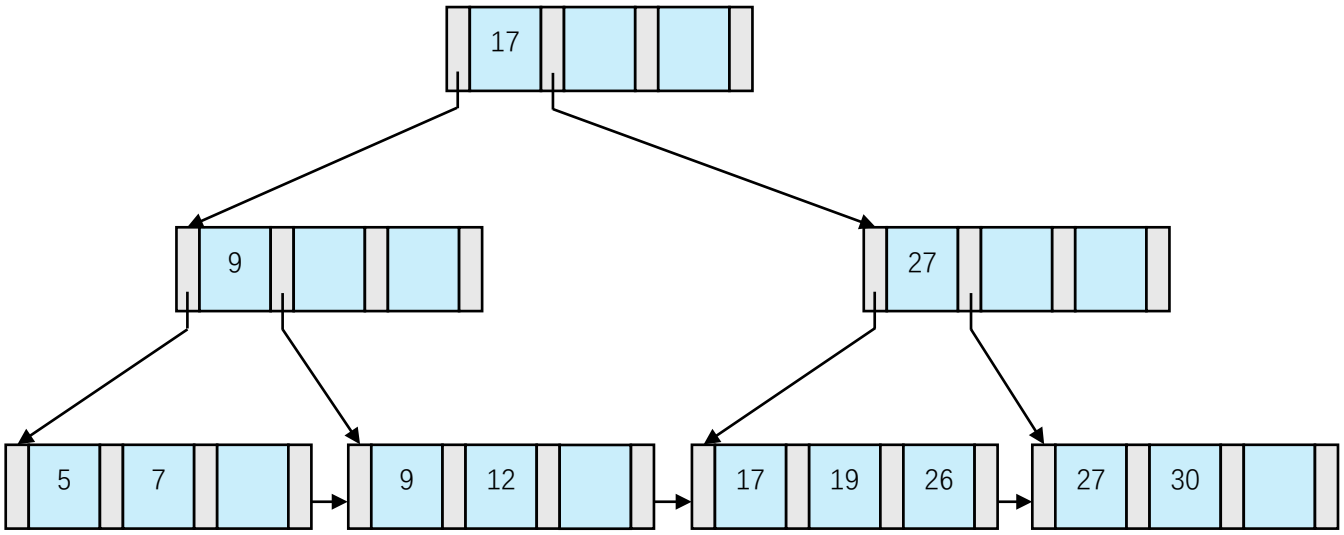


B^+树插入节点完毕，下面进行删除节点。

删除 10，其所在叶子节点只剩 1 个值，左右兄弟都不够借，只能合并，合并后删除 1 个相关索引。



删除 15，其所在叶子节点剩 2 个值。



最终得到的 B^+树如上所示。

Q2: Compare B^+-tree and B-tree and describe their difference

1. 数据存储

B^+树只有叶子节点存放数据，内部节点只存放索引（键值）；B 树所有节点（包括内部节点和叶子节点）都可以存放数据。

B^+树内部节点仅存储键值，内部节点只存索引，能存储更多分支，提高了树的扇出度（branching factor），从而减少树的高度，加快查找；B 树内部节点存储数据，导致每个节点能存储的键值较少，节点存储效率略低。

2. 节点结构

B^+树叶子节点通过指针形成有序链表,非叶子节点仅包含键值和指向子节点的指针;B 树

叶子节点之间无链接，非叶子节点包含键值和指向子节点的指针。

3.查询效率

单条查询：B^+树必须访问到叶子节点才能获取数据，可能增加 I/O 次数，但查找路径更统一；B 树可能在非叶子节点直接命中数据，减少 I/O 次数。

范围查询：B^+树叶子节点通过指针相连，效率高，通过叶子节点链表顺序遍历；B 树需要进行中序遍历，需多次回溯到父节点，效率较低。

磁盘 I/O：相同情况下，B^+树的高度更低，I/O 次数更少；B 树的高度可能略高，I/O 次数较多。

4.插入与删除

B^+树插入删除操作更稳定，叶子节点的链表结构简化了范围调整；B 树插入删除操作可能导致节点分裂或合并，影响性能。

5.典型应用场景

B^+树适用于大规模数据的高效范围查询，更适合数据库、文件系统等磁盘存储场景，因为减少了 I/O 次数；B 树适用于小规模数据的快速查找，适合内存中的数据索引。