1. **聚簇索引和次级索引之间有什么区别?**

    1. **物理排序和存储方式:**

        - **聚簇索引:** 数据表的行按照聚簇索引的顺序物理上存储在磁盘上。这意味着具有相似值的行在存储上也是相邻的。聚簇索引通常是主键索引，因为主键必须是唯一的，而唯一的索引天然地将行进行了排序。
        - **次级索引:** 数据表的行在磁盘上的存储顺序与次级索引无关，而是根据表的聚簇索引进行的。次级索引只保存键值和对应的行指针，而不是实际的数据行。

    2. **唯一性:**

        - **聚簇索引:** 在一个表中，只能有一个聚簇索引，通常是主键索引。这是因为数据行的物理存储顺序是按照聚簇索引的顺序排列的。
        - **次级索引:** 一个表可以有多个次级索引，这些索引可以建立在表的非主键列上。次级索引可以是唯一的，也可以允许重复值。

    3. **查询性能:**

        - **聚簇索引:** 由于数据行的物理存储顺序与聚簇索引的顺序一致，对于范围查询和顺序访问的性能通常较好。
        - **次级索引:** 在使用次级索引进行查询时，数据库系统首先使用次级索引查找到相应的行指针，然后再根据行指针去找到实际的数据行。因此，在使用次级索引进行查询时可能需要两次查找，相比之下，可能会引入一些性能开销。

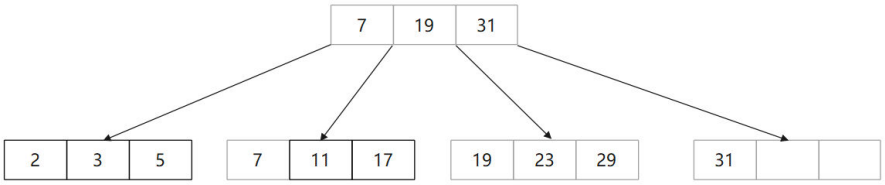
    4. **数据维护的开销:**

        - **聚簇索引:** 当插入新数据时，由于要保持物理顺序，可能需要对整个表进行重组，因此插入操作的开销可能较大。
        - **次级索引:** 插入新数据时，只需要更新相应的次级索引，因此插入操作的开销相对较小。

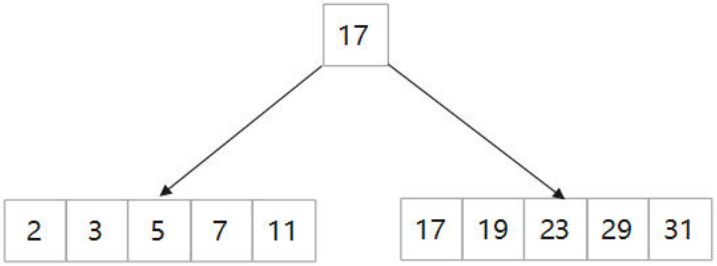
---

2. **构建一棵 B+-树，其中包含以下一组键值：(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)假设该树最初为空，并且值按升序添加。构建 B+-树，其中树的节点可以容纳的指针数量分别如下：**

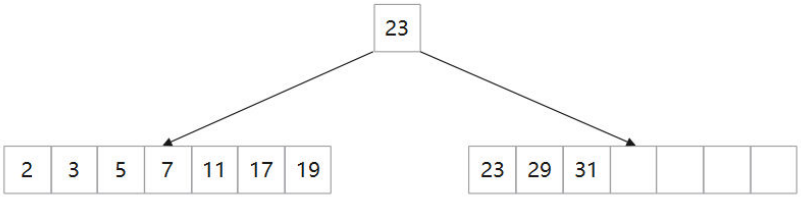
    1. 四个



    2. 六个



    3. 八个



---

3. 对于练习 2 中的每个B+-树，说明下面查询涉及的步骤：

    1. 查找搜索键值为11的记录。

        1. 四个 首先比较根节点第一个记录，11 > 7 因此转到第二个指针，发现 19 > 11 因此沿着指针向下，在叶子节点中找到了11
        2. 六个 首先比较根节点， 17 > 11 因此从做指针找记录，在叶子节点中找到了11

3. 八个 首先比较根节点， 23 > 11 因此从做指针找记录，在叶子节点中找到了11

2. 查找搜索键值在7到17之间（包括7和17）的记录。
    1. 四个 在根节点比较发现7 ~ 17 介于第一个记录和第二个记录之间，于是沿着第二个指针在叶子节点中找到7 ~ 17的记录
    2. 六个 经过比较后发现7 ~ 16可以在左边指针找到，17要到右边指针找到
    3. 八个 经过比较后发现全部都在左边指针所指向的叶子节点

---

4. **如果按排序顺序插入索引条目，B+-树的每个叶子节点的占用情况会是什么？请解释原因** B+-树的每个叶子节点逐渐填充，保持了树的平衡。只有在节点达到最大容量时，才会触发分裂操作。这样的设计使得 B+-树在范围查询和顺序访问时能够保持高效。

---

5. **假设你需要在大量名称上创建一个B+-树索引，其中一个名称的最大大小可能相当大（比如40个字符），而平均名称本身也很大（比如10个字符）。解释一下如何使用前缀压缩来最大化非叶子节点的平均扇出**

    1. **选择合适的前缀长度：** 需要确定一个合适的前缀长度，以便在非叶子节点上进行压缩。这个前缀长度应该足够长以减少节点大小，但也要注意不要太长，以避免牺牲索引的查询性能。

    2. **对键进行前缀压缩：** 对于每个键，将其前缀部分提取出来，并存储在非叶子节点中。这个前缀将用于在非叶子节点上进行比较，以确定向左还是向右分支。

    3. **存储非叶子节点：** 在非叶子节点中，存储压缩后的前缀以及指向下一级节点的指针。这样可以显著减小非叶子节点的大小，提高节点的扇出。

    4. **在叶子节点中存储完整的键值：** 叶子节点仍然存储完整的键值，以确保准确的搜索和范围查询。

    5. **注意前缀冲突：** 当多个键具有相同的前缀时，需要考虑如何处理前缀冲突。一种处理方式是存储所有可能的冲突前缀，或者使用其他

技术来解决这个问题，例如在非叶子节点上使用更多的信息来区分冲突。