B+树索引

> 由朱悦铭于2024年设计
> 参考：
> https://www.geeksforgeeks.org/deletion-in-b-tree/
> https://blog.csdn.net/wyll19980812/article/details/106069433
> 彭智勇 彭煜玮. 《PostgreSQL 数据库内核分析》, 机械工业出版社 2012.

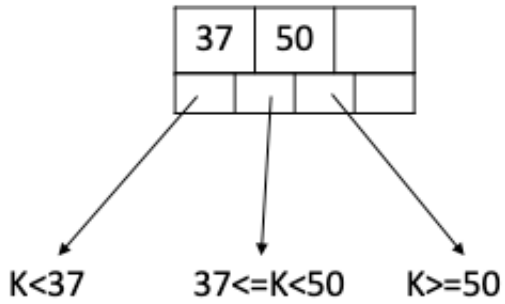
简介

B+树的叶节点存储**键及其对应的地址**，上层节点（根节点和内部节点）用作索引。对于阶数为M的B+树，各节点的键数量定义如下：

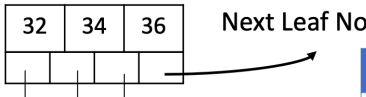
- **根节点**：1 至 M-1
- **内部节点**：⌈M/2⌉-1 至 M-1
- **叶节点**：⌈M/2⌉-1 至 M-1

以阶数M=4为例：

- **根节点或内部节点**结构：

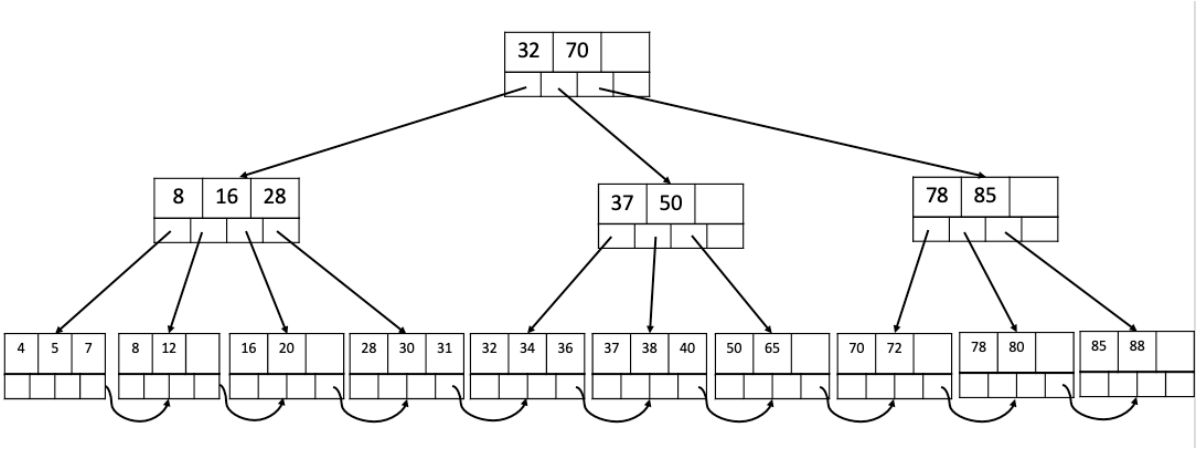


- **叶节点**结构（示例为**聚簇索引**）：



Id	Start_Station	End_Station	price	date
30	Shabo	Qianwan Park	11	2023-08-05
31	Luohu North	Nanshan	6	2024-01-21
32	Mawan	Mumianwan	8	2023-02-23
33	Qianwan	Science Museum	6	2023-06-24
34	Henggang	Nanshan Book Mall	8	2023-10-26
35	Fuhai West	Liyumen	6	2023-02-03
36	Keyuan	Bantian North	7	2024-01-22

- **完整的B+树索引**示例：



插入操作

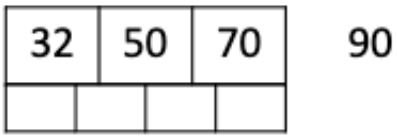
通用流程

插入操作需先定位目标叶节点，根据键数量分为三种情况：

1. **叶节点键数量 < M-1**：直接插入。
2. **叶节点键数量 = M-1**：
   - 分裂为两个叶节点：
     - 左叶节点包含索引：1 ~ ⌊M/2⌋
     - 右叶节点包含索引：⌊M/2⌋+1 ~ M
   - 将右叶节点的首个键插入父节点。
3. **父节点键数量 = M**：
   - 分裂父节点为两个内部节点：
     - 左内部节点包含索引：1 ~ ⌊M/2⌋
     - 右内部节点包含索引：⌊M/2⌋+2 ~ M
   - 将中间键（⌊M/2⌋+1）插入更上层节点。
   - 若根节点分裂，树的高度加1。

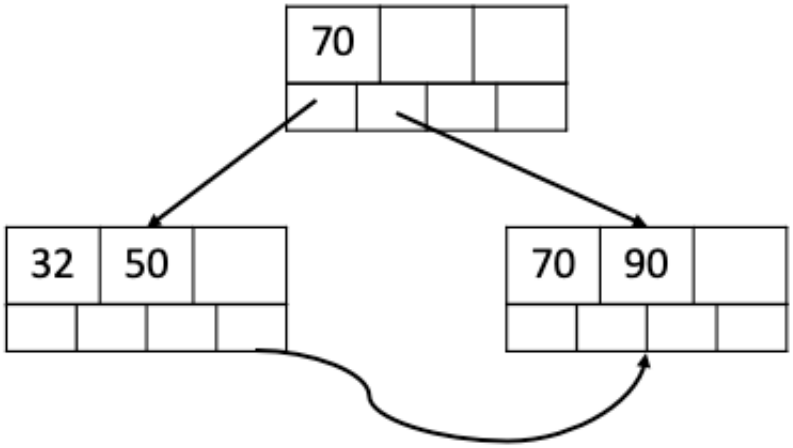
示例（阶数M=4）

- **初始插入32, 50, 70, 90**：

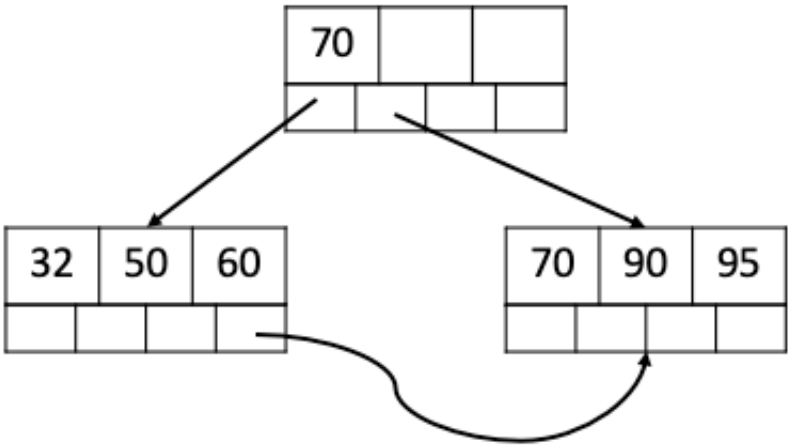


90

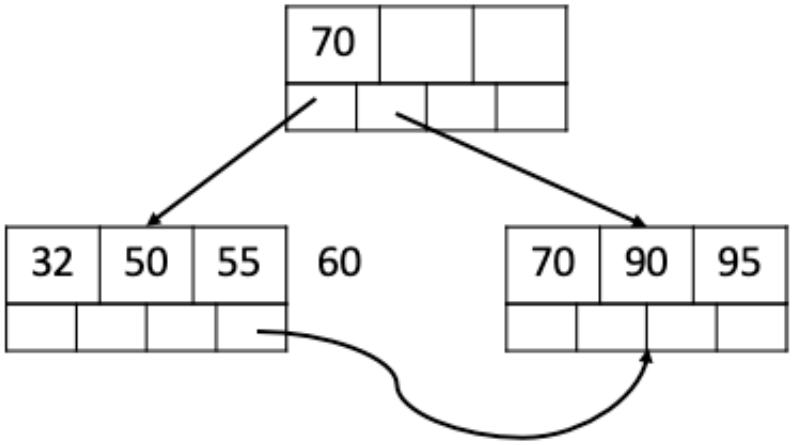
叶节点满后分裂：

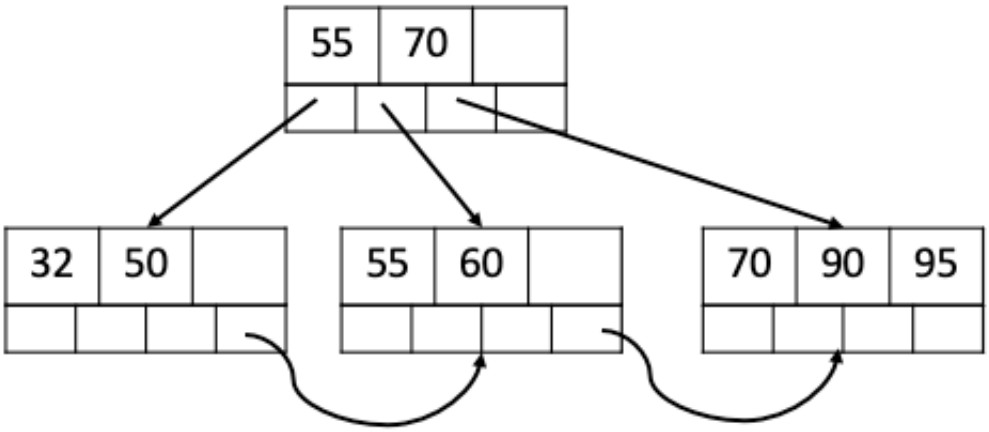


- **插入60, 95**：

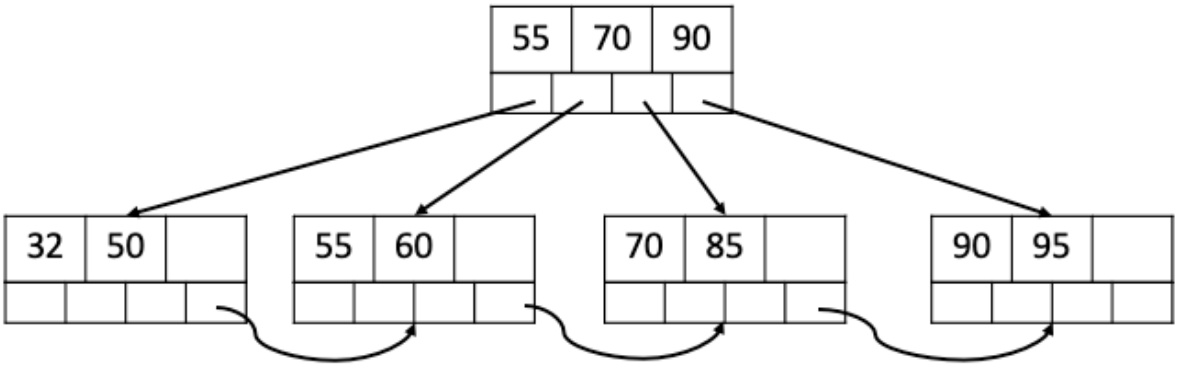


- **插入55**：
  分裂叶节点（32, 50, 55, 60）为[32, 50]和[55, 60]，并将55插入父节点：

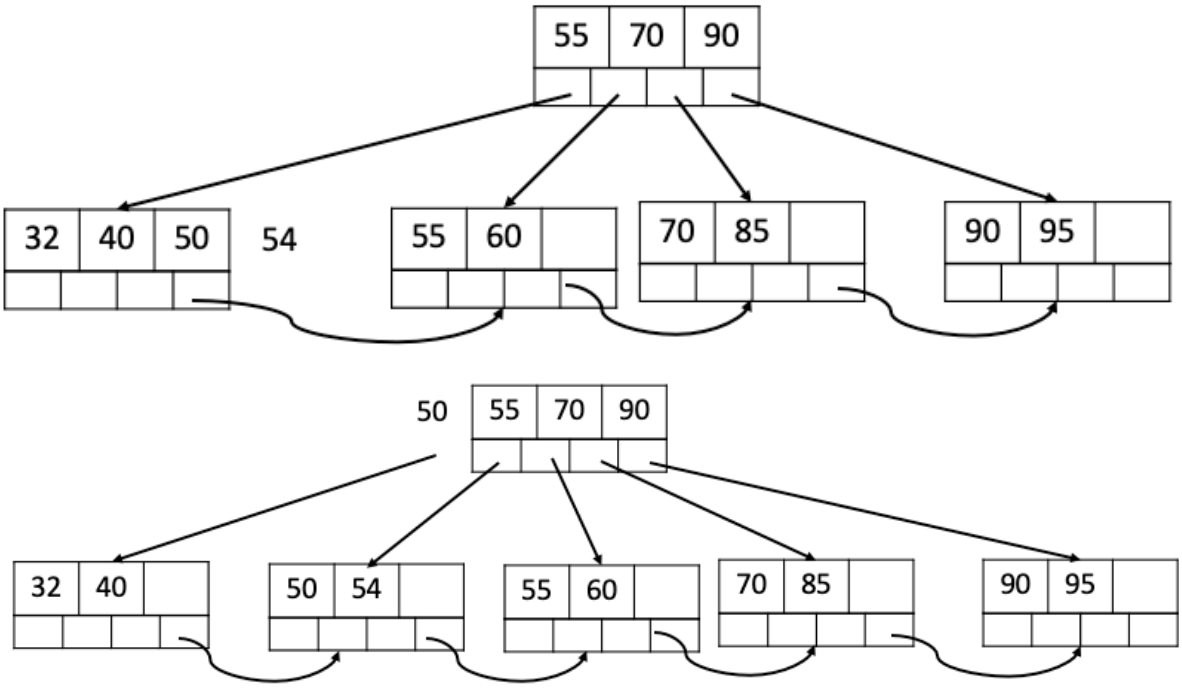




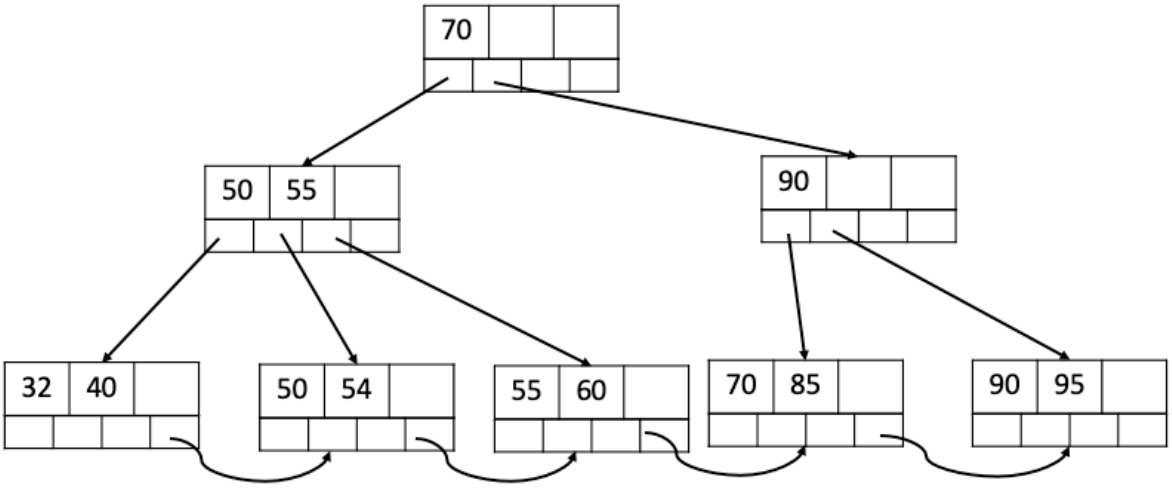
- **插入85**：



- **插入40, 54**：
  分裂叶节点（32, 40, 50, 54）为[32, 40]和[50, 54]，并将50插入父节点。父节点满后继续分裂：



  树高度增加：



删除操作

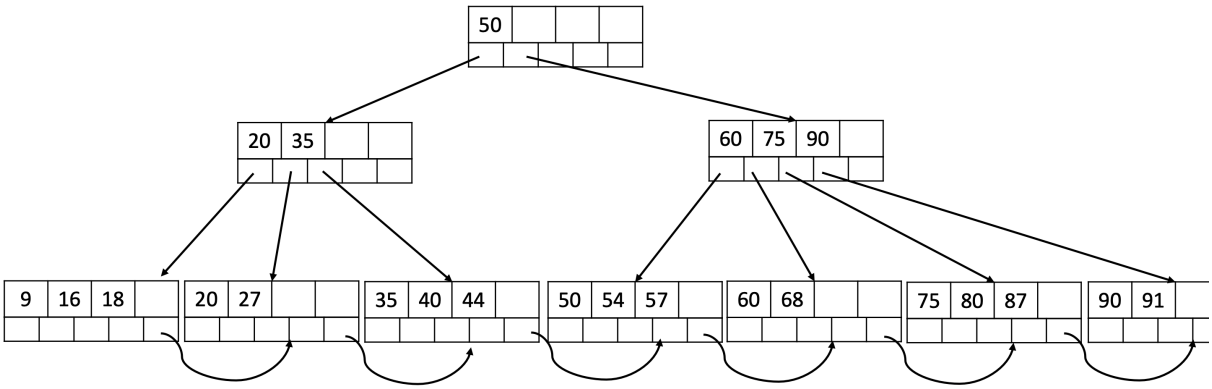
通用流程

删除操作需定位目标叶节点，根据键数量和兄弟节点状态分为五种情况：

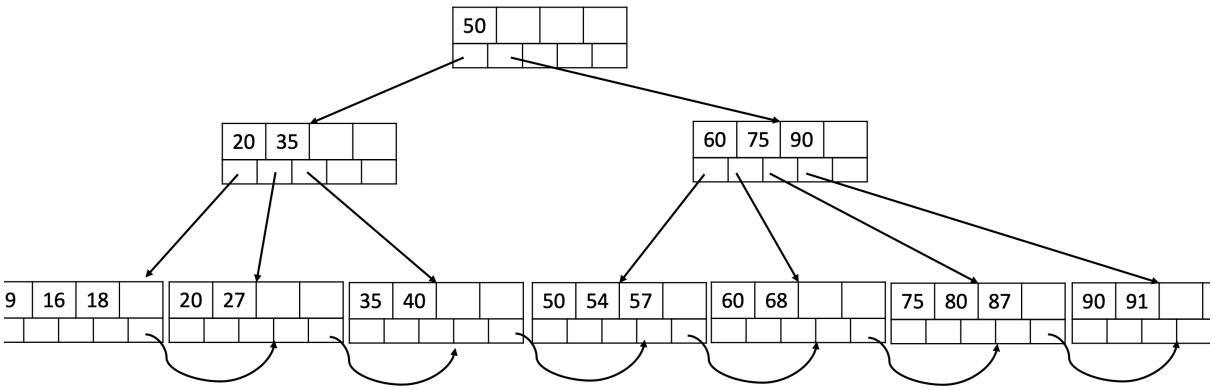
1. **叶节点键数量 > ⌈M/2⌉-1**：直接删除。若删除的键是当前节点首键，需更新父节点对应键。
2. **叶节点键数量 = ⌈M/2⌉-1，且兄弟节点有富余键**：从兄弟节点借键，并更新父节点键。
3. **叶节点键数量 = ⌈M/2⌉-1，且兄弟节点无富余键**：合并两个叶节点，父节点键数量减1。
4. **父节点键数量 < ⌈M/2⌉-1，且兄弟节点有富余键**：
    - 将父节点键下移，兄弟节点键上移。
5. **父节点键数量 < ⌈M/2⌉-1，且兄弟节点无富余键**：合并父节点与其兄弟节点。若根节点键数量为0，树高度减1。

示例1（阶数M=5）

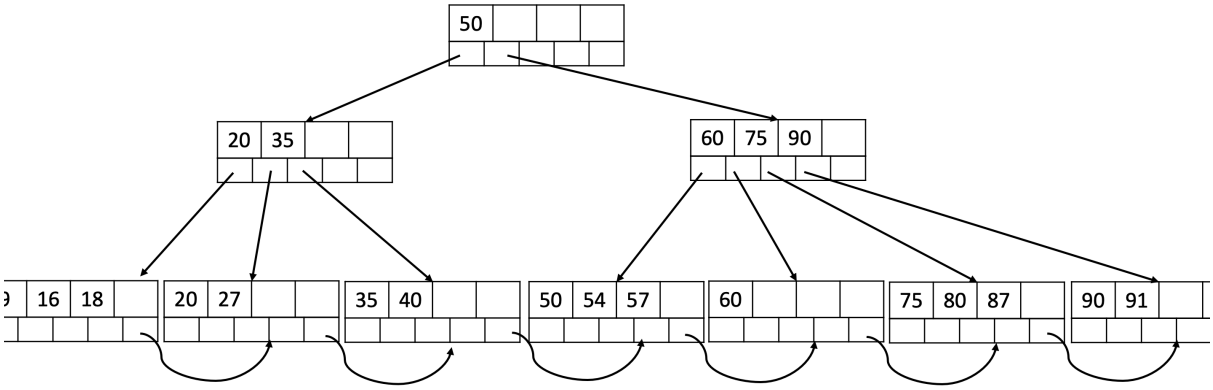
- **原始B+树**：

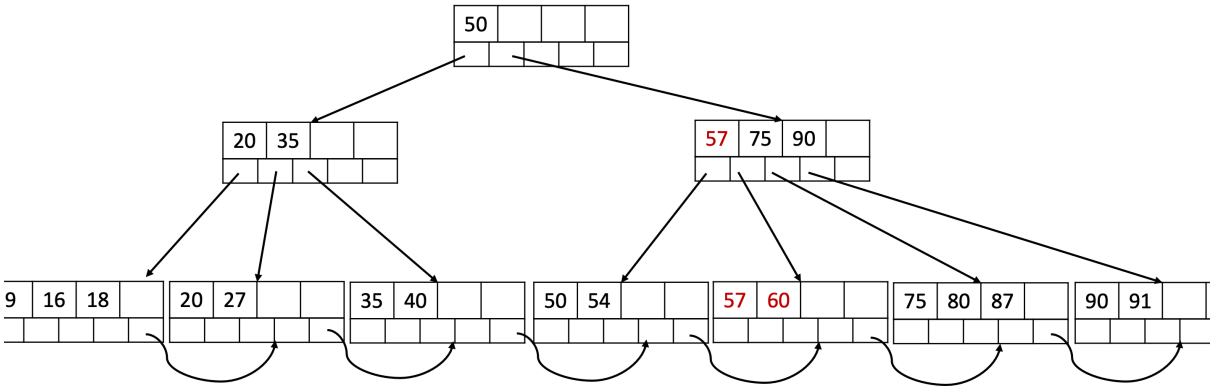


- **删除44**：直接删除：

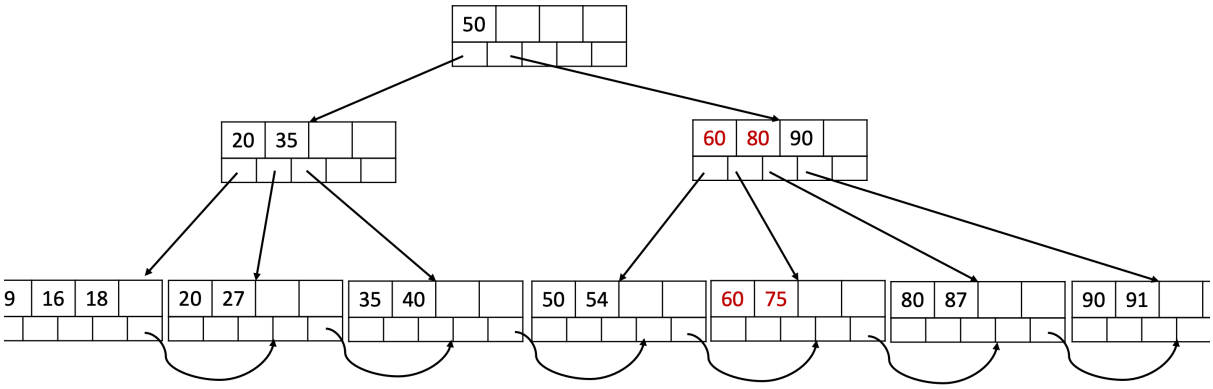


- **删除68**：叶节点键不足，从兄弟节点借57并更新父节点：

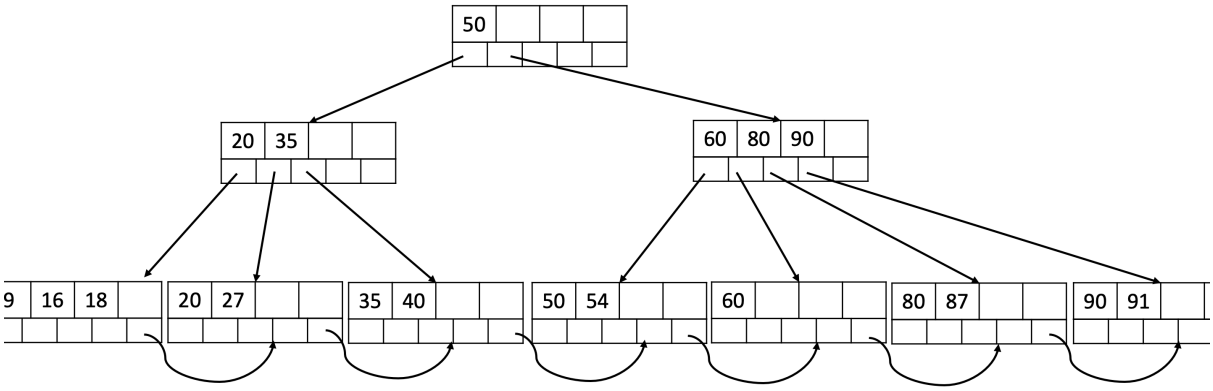


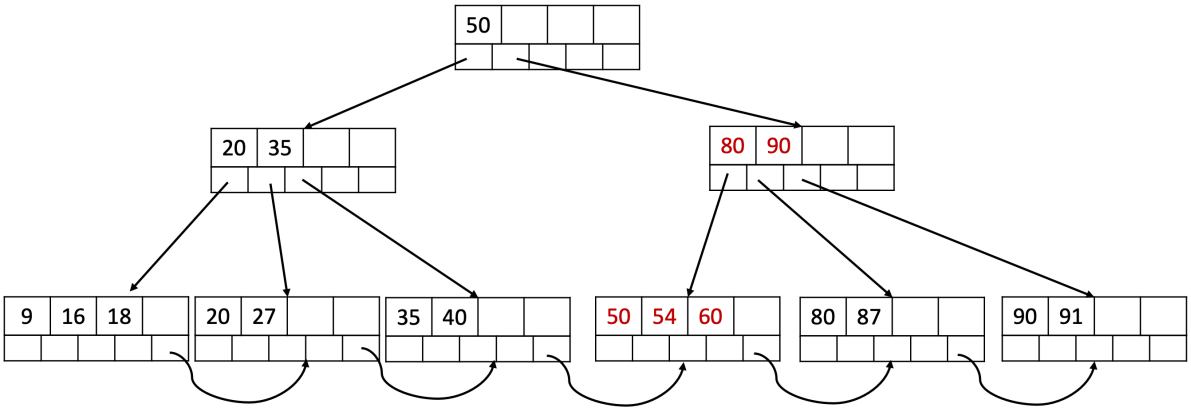


- **删除57**：

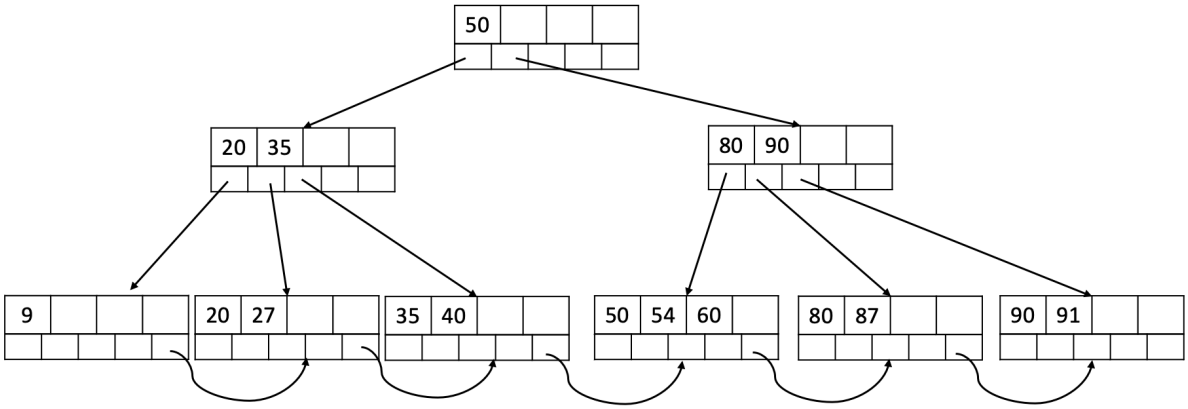


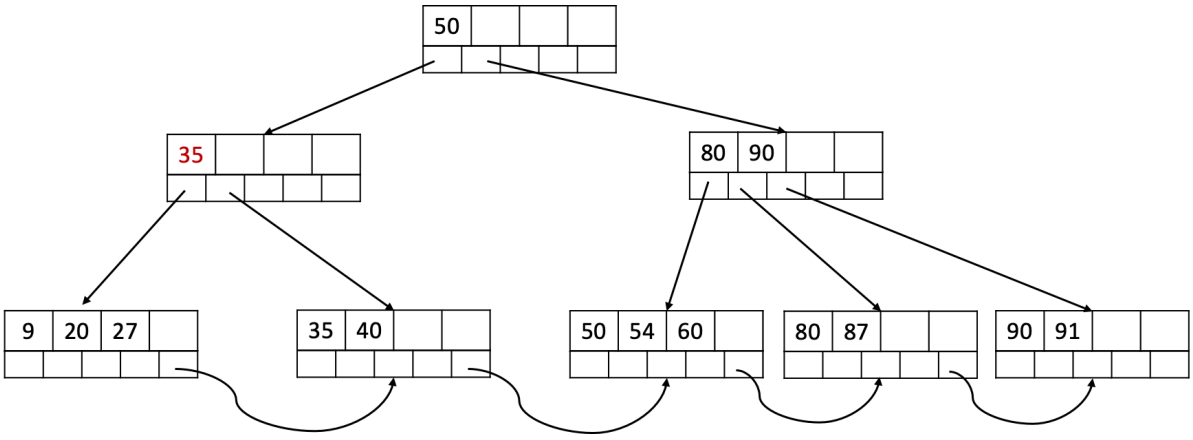
- **删除75**：合并叶节点并更新父节点：

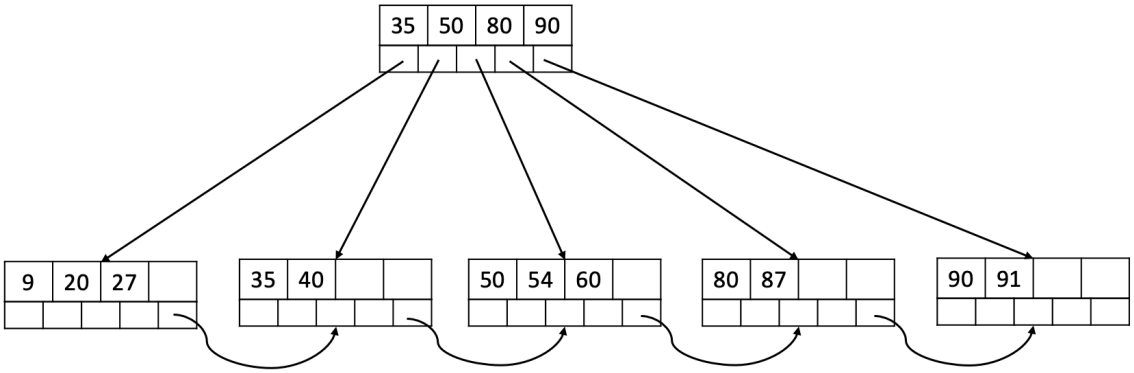




- **删除18, 16**：合并叶节点后父节点键不足，最终合并内部节点并降低树高度：

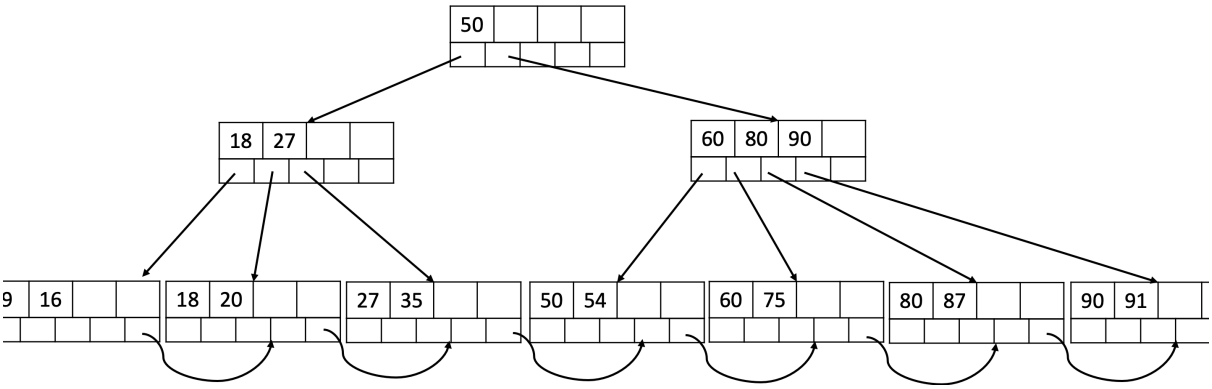


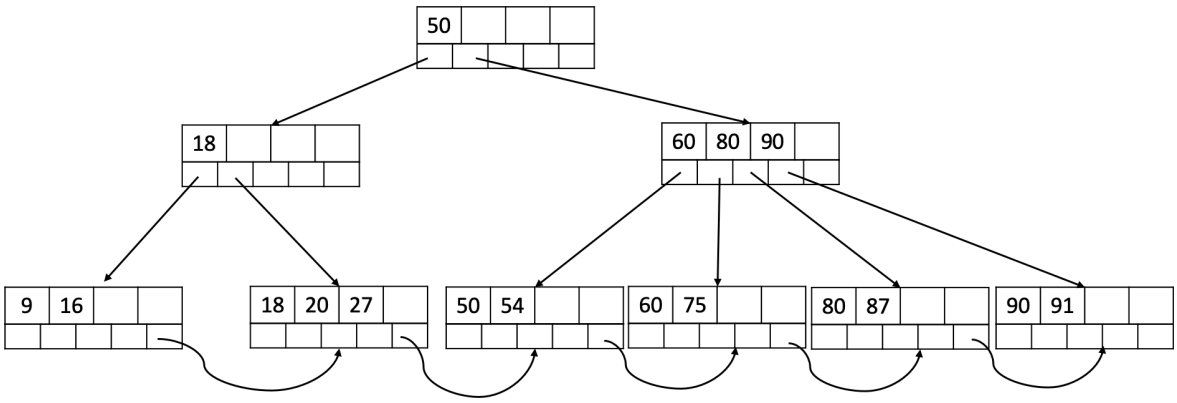


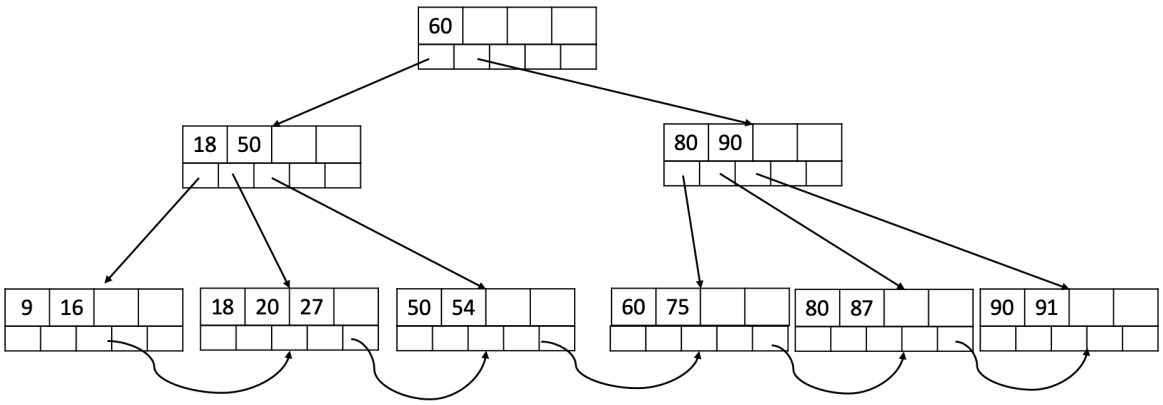


示例2

- **删除35**：合并叶节点后父节点键不足，借键并调整结构：







代码设计

Java实现

```
class Node {
    List<Integer> keys;
    List<Node> values;
    boolean leaf;
    Node next;

    public Node(boolean leaf) {
        this.keys = new ArrayList<>();
        this.values = new ArrayList<>();
        this.leaf = leaf;
        this.next = null;
    }
}

public class BPlusTree {
    private Node root;
    private int degree;

    public BPlusTree(int degree) {
        this.root = new Node(true);
        this.degree = degree;
    }
}
```

Java 语言完整代码参考：

https://github.com/jiaguofang/b-plus-tree

Python实现

```
class Node:
    def __init__(self, leaf=False):
        self.keys = []
        self.values = []
        self.leaf = leaf
        self.next = None

class BPlusTree:
    def __init__(self, degree):
        self.root = Node(leaf=True)
        self.degree = degree
```

Python语言完整代码参考：

https://github.com/raymondlee2020/b-plus-tree

C++实现

```
class Node {
public:
    std::vector<int> keys;
    std::vector<Node*> values;
    bool leaf;
    Node* next;

    Node(bool isLeaf) : leaf(isLeaf), next(nullptr) {}
};

class BPlusTree {
private:
    Node* root;
    int degree;
};
```

C++语言完整代码参考：

https://github.com/solangii/b-plus-tree

练习：

练习1: 基于B+树索引的I/O练习题目

描述

本作业基于B+树的插入操作测试。所有插入规则遵循《Tutorial14 B+树》文档中的描述。B+树的索引类型为整型，仅测试索引键，不测试键对应的值数据。

可使用Java、Python或C++完成本次练习。

输入描述

- 第一行包含一个整数M（3 ≤ M ≤ 8），表示B+树的阶数。
- 第二行包含多个整数，表示待插入的键。整数数量大于1且小于1000，以空格分隔。
- 第三行包含一个整数-1，表示输入结束。

输出描述

插入完成后，按层输出B+树的所有节点，每层用一行分隔。具体要求如下：

- 第一行输出根节点。
- 第二行输出第二层节点。
- 最后一行输出叶节点层。

每个节点的输出格式为 `[key1, key2, ..., keyn]`，同一层的节点用 `,` 分隔。例如：`[50, 55] [90]`。

示例输入1

```
4
32 50 70 90 60 95 55 85 40 54
-1
```

示例输出1

```
[70]
[50, 55] [90]
[32, 40] [50, 54] [55, 60] [70, 85] [90, 95]
```

示例输入2

```
5
90 3 59 68 12 11 15 16 88 67 65 44 34 89 85 25 64 5 83 42 79 60
-1
```

示例输出2

```
[65]
[15, 34, 44] [68, 85, 89]
[3, 5, 11, 12] [15, 16, 25] [34, 42] [44, 59, 60, 64] [65, 67] [68, 79, 83]
[85, 88] [89, 90]
```

练习2: 基于B+树索引应用于火山模型结构中

根据项目的结构，创建IndexScan，这里为了简化难度，可以选择创建一个基于内存的B+树索引作为练习，并将其应用在火山模型结构中。 并设计查询优化器，设计在什么情况下，使用IndexScan, 在什么情况下， 使用SeqScan.