

B树

@M了个J

<https://github.com/CoderMJLee>

<http://cnblogs.com/mjios>

码拉松



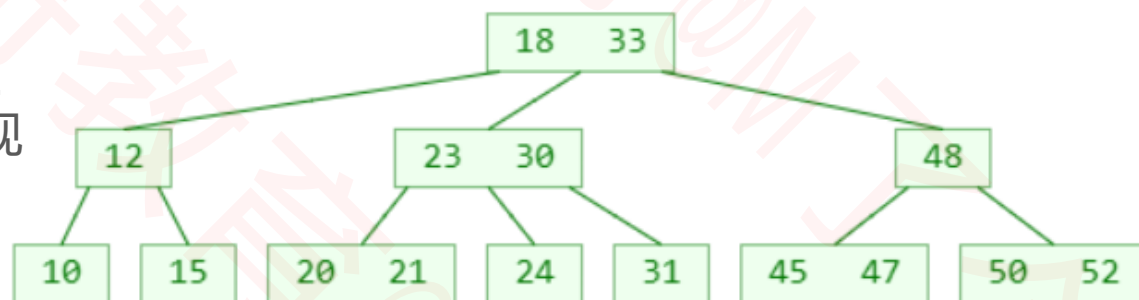
实力IT教育 www.520it.com

B树 (B-tree、B-树)

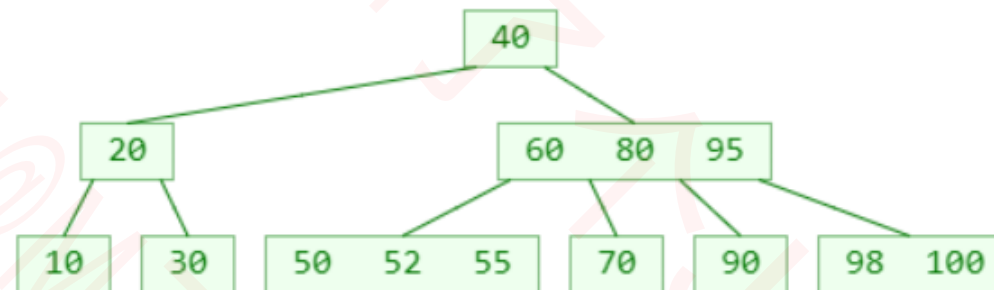
■ B树是一种平衡的**多路**搜索树，多用于文件系统、数据库的实现

■ 仔细观察B树，有什么眼前一亮的点？

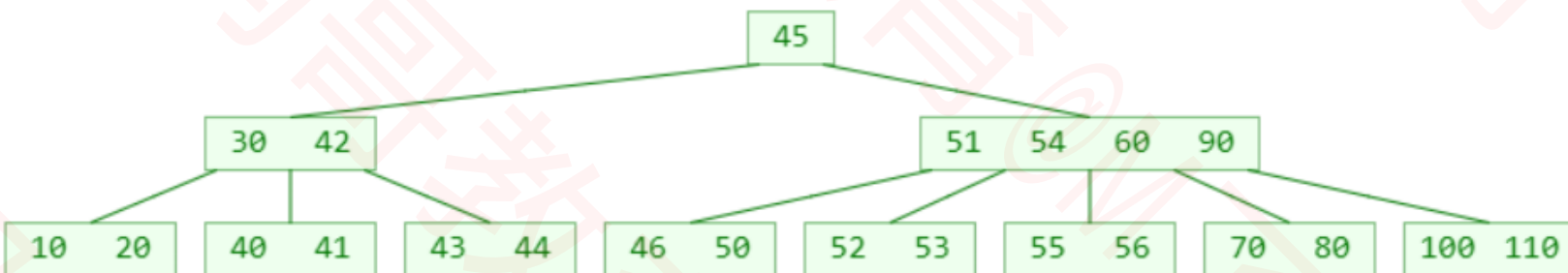
- 1 个节点可以存储超过 2 个元素、可以拥有超过 2 个子节点
- 拥有二叉搜索树的一些性质
- 平衡，每个节点的所有子树高度一致
- 比较矮



3阶B树



4阶B树



5阶B树

m阶B树的性质 ($m \geq 2$)

■ 假设一个节点存储的元素个数为 x

□ 根节点: $1 \leq x \leq m - 1$

□ 非根节点: $\lceil m/2 \rceil - 1 \leq x \leq m - 1$

□ 如果有子节点, 子节点个数 $y = x + 1$

✓ 根节点: $2 \leq y \leq m$

✓ 非根节点: $\lceil m/2 \rceil \leq y \leq m$

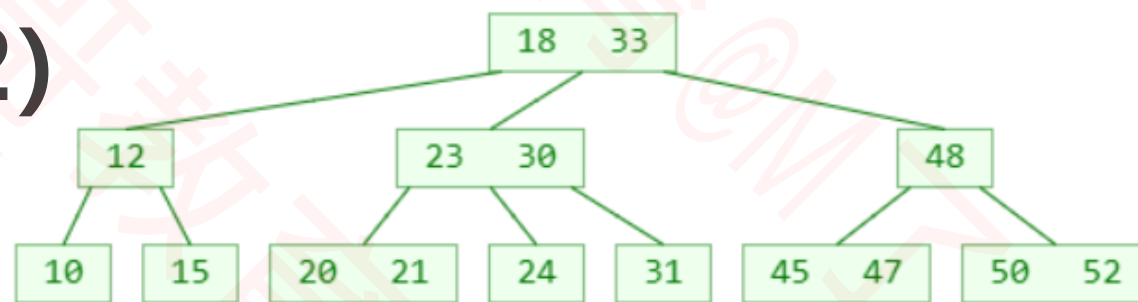
➤ 比如 $m = 3$, $2 \leq y \leq 3$, 因此可以称为 (2, 3) 树、2-3树

➤ 比如 $m = 4$, $2 \leq y \leq 4$, 因此可以称为 (2, 4) 树、2-3-4树

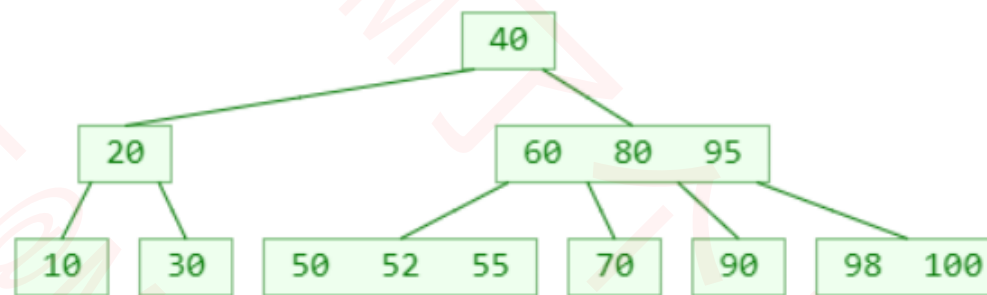
➤ 比如 $m = 5$, $3 \leq y \leq 5$, 因此可以称为 (3, 5) 树

➤ 比如 $m = 6$, $3 \leq y \leq 6$, 因此可以称为 (3, 6) 树

➤ 比如 $m = 7$, $4 \leq y \leq 7$, 因此可以称为 (4, 7) 树



3阶B树



4阶B树

■ 思考: 如果 $m = 2$, 那B树是什么样子?

■ 你猜数据库实现中一般用几阶B树?

□ 200 ~ 300

B树 VS 二叉搜索树

■ B树 和 二叉搜索树，在逻辑上是等价的

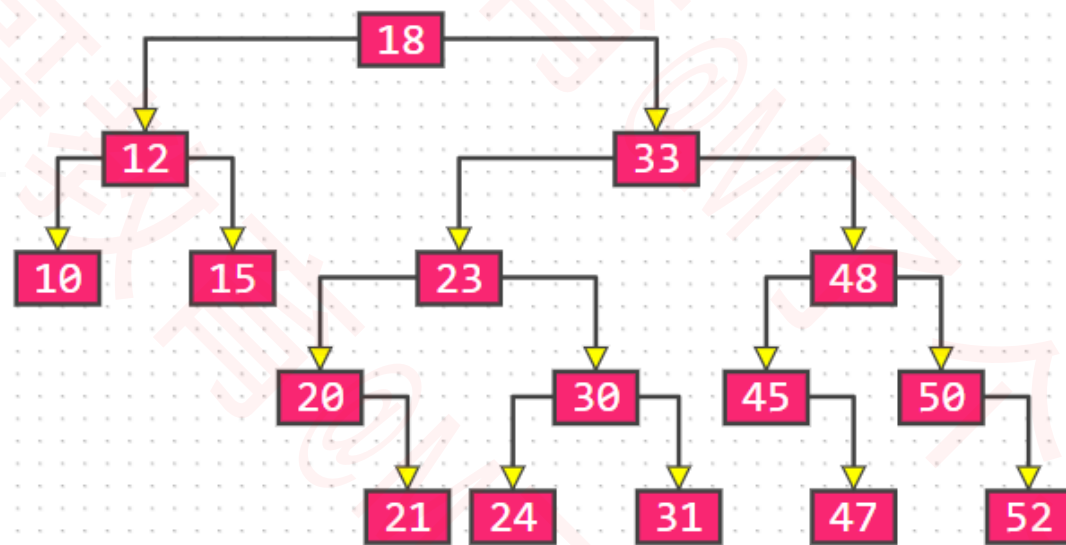
■ 多代节点合并，可以获得一个超级节点

□ 2代合并的超级节点，最多拥有 4 个子节点（至少是 4阶B树）

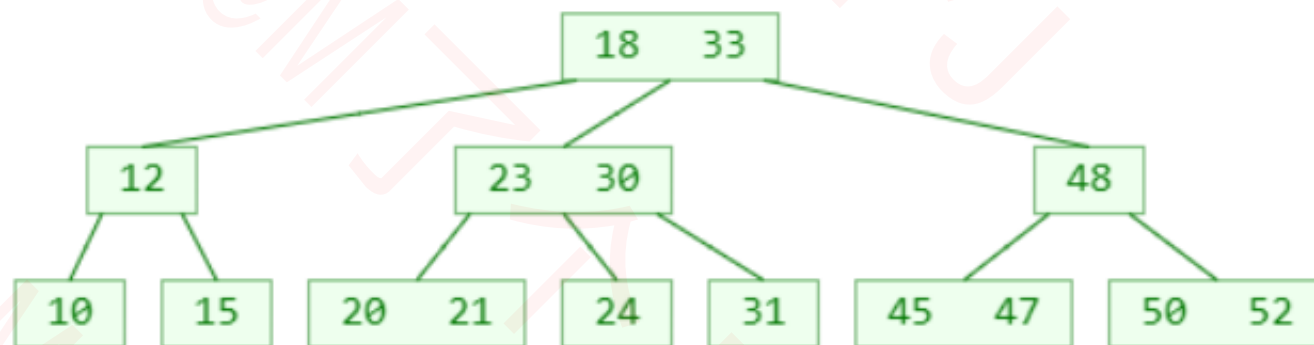
□ 3代合并的超级节点，最多拥有 8 个子节点（至少是 8阶B树）

□ n代合并的超级节点，最多拥有 2^n 个子节点（至少是 2^n 阶B树）

■ m阶B树，最多需要 $\log_2 m$ 代合并

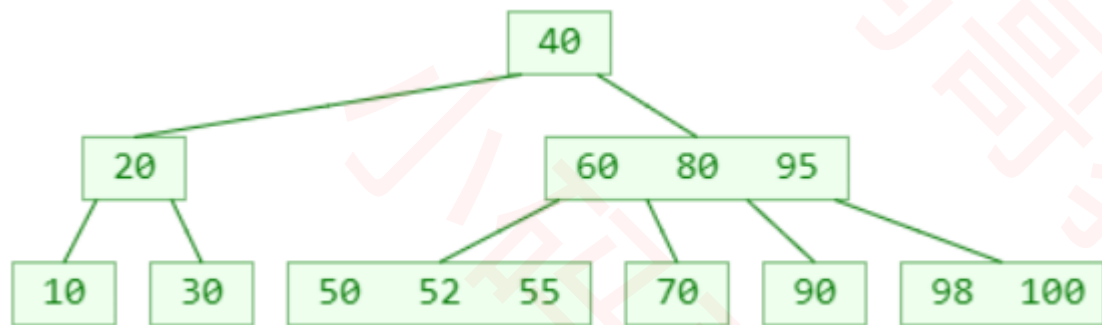


二叉搜索树



3阶B树

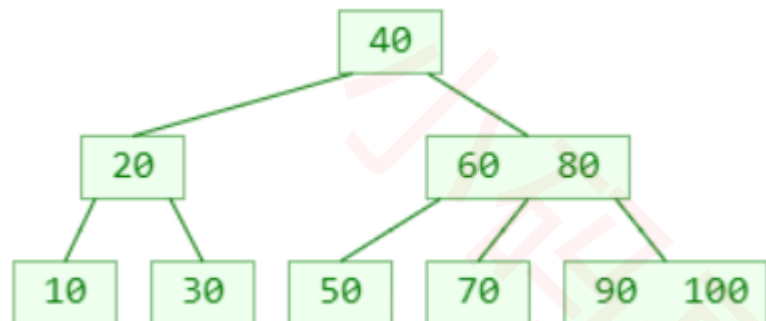
■ 跟二叉搜索树的搜索类似



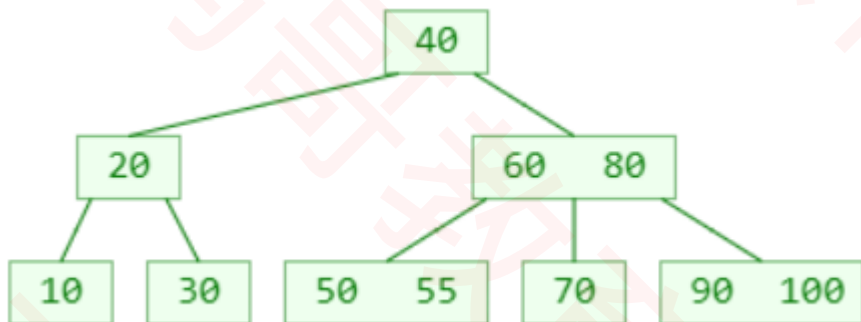
1. 先在节点内部从小到大开始搜索元素
2. 如果命中，搜索结束
3. 如果未命中，再去对应的子节点中搜索元素，重复步骤 1

添加

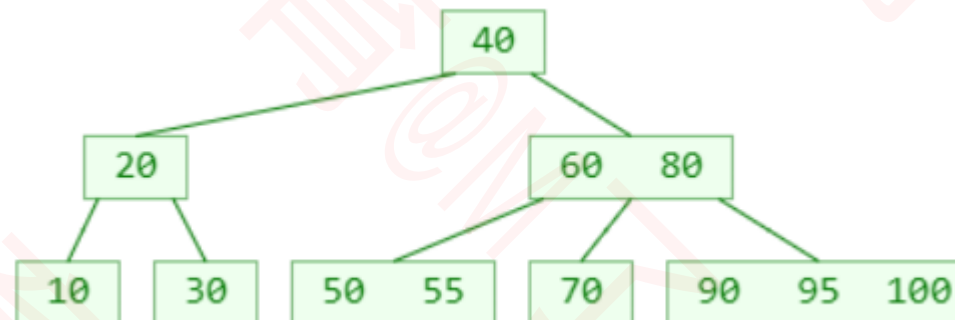
- 新添加的元素必定是添加到叶子节点



- 插入55



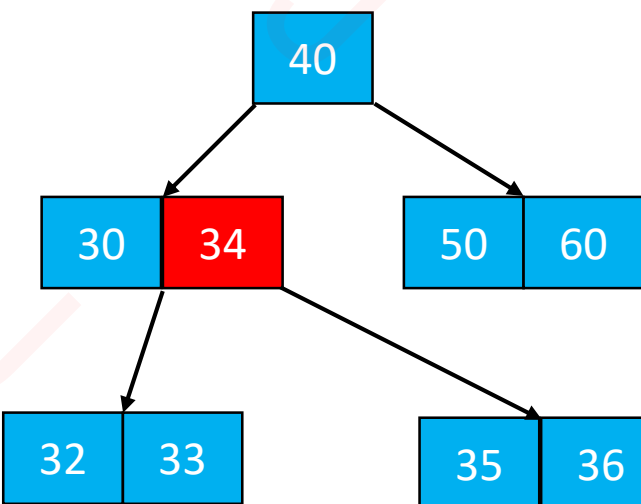
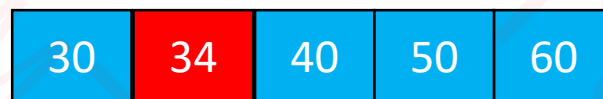
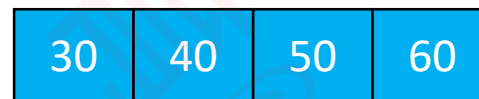
- 插入95



- 再插入 98 呢？（假设这是一棵 4阶B树）
- 最右下角的叶子节点的元素个数将超过限制
- 这种现象可以称之为：**上溢 (overflow)**

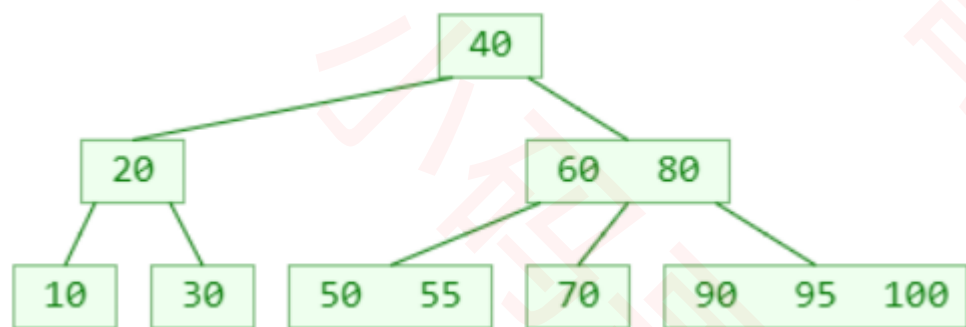
添加 - 上溢的解决(假设5阶)

- 上溢节点的元素个数必然等于 m
- 假设上溢节点最中间元素的位置为 k
- 将 k 位置的元素向上与父节点合并
- 将 $[0, k-1]$ 和 $[k+1, m-1]$ 位置的元素分裂成 2 个子节点
- ✓ 这 2 个子节点的元素个数, 必然都不会低于最低限制 ($\lceil m/2 \rceil - 1$)
- 一次分裂完毕后, 有可能导致父节点上溢, 依然按照上述方法解决
- 最极端的情况, 有可能一直分裂到根节点

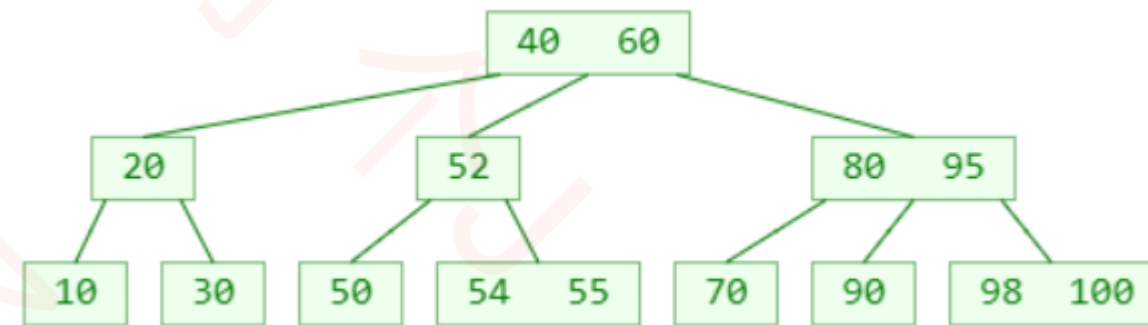
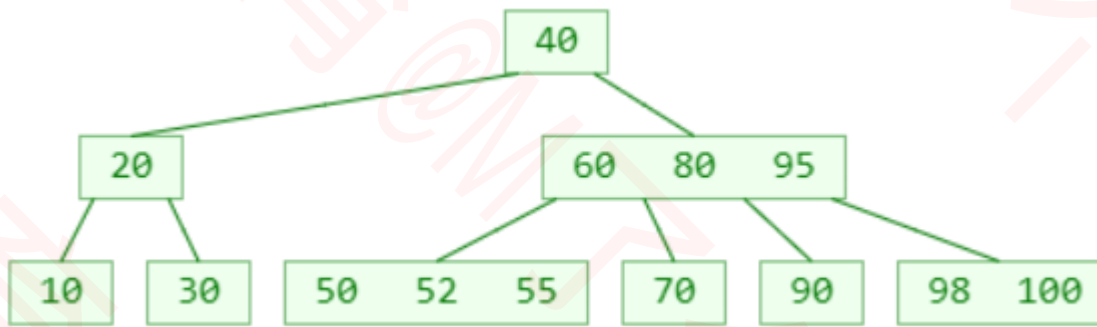
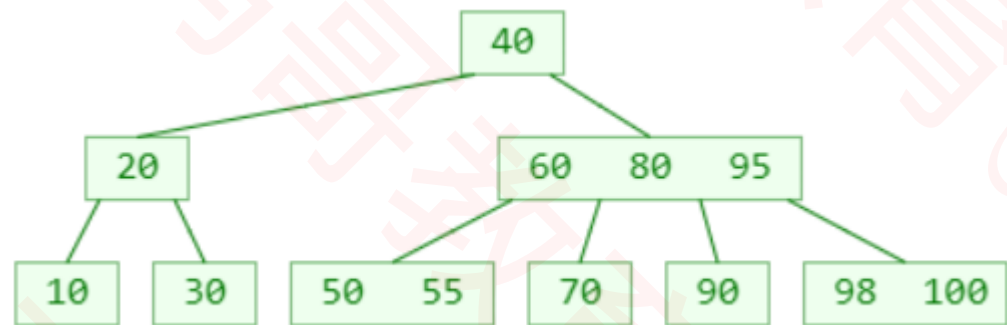


添加

■ 插入 52

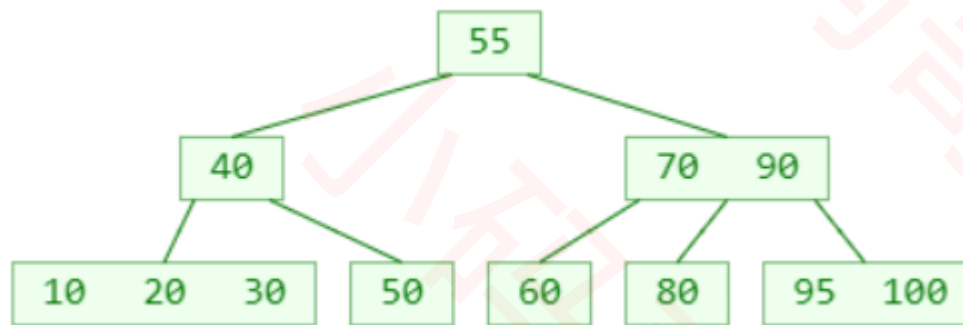


■ 插入 54

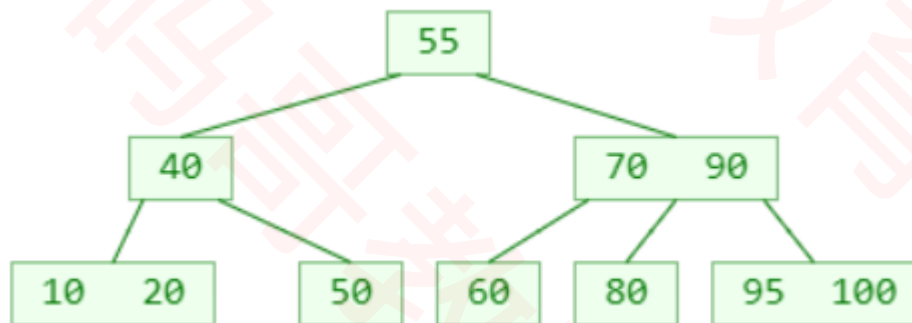


删除 - 叶子节点

- 假如需要删除的元素在叶子节点中，那么直接删除即可



- 删除 30

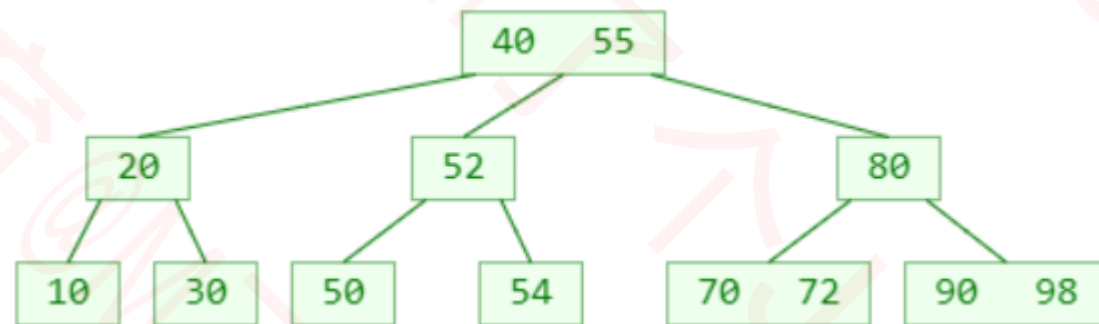
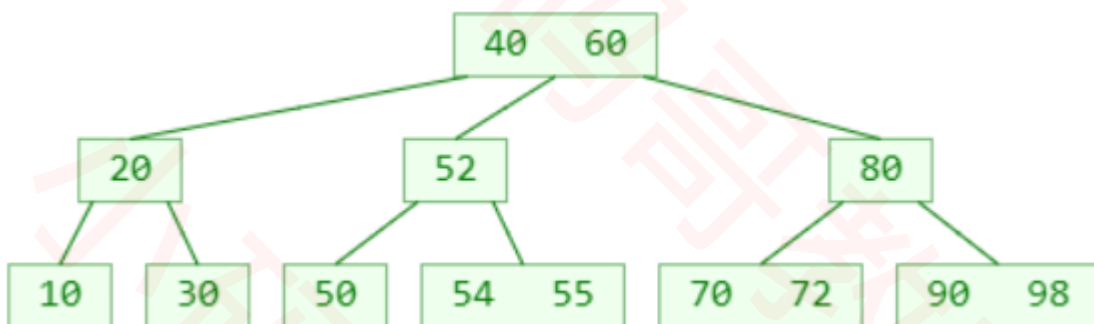


删除 - 非叶子节点

■ 假如需要删除的元素在非叶子节点中

1. 先找到前驱或后继元素，覆盖所需删除元素的值
2. 再把前驱或后继元素删除

■ 删除 60

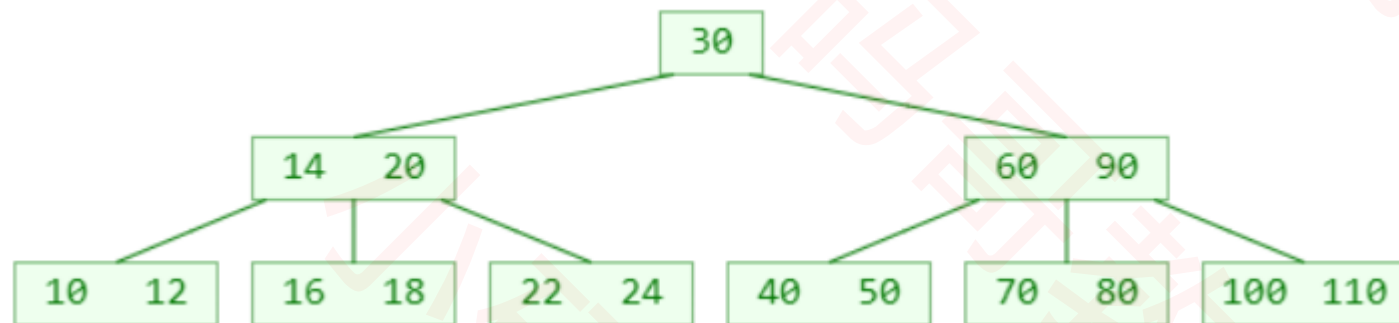


■ 非叶子节点的前驱或后继元素，必定在叶子节点中

□ 所以这里的删除前驱或后继元素，就是最开始提到的情况：删除的元素在叶子节点中

□ 真正的删除元素都是发生在叶子节点中

删除 - 下溢



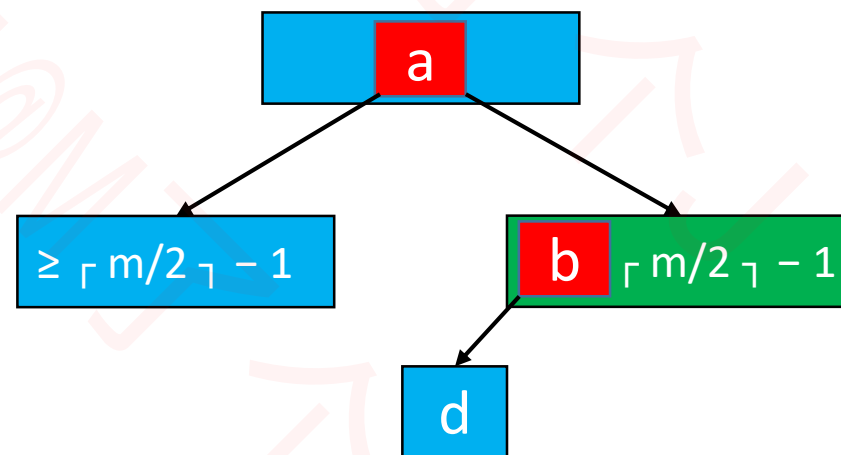
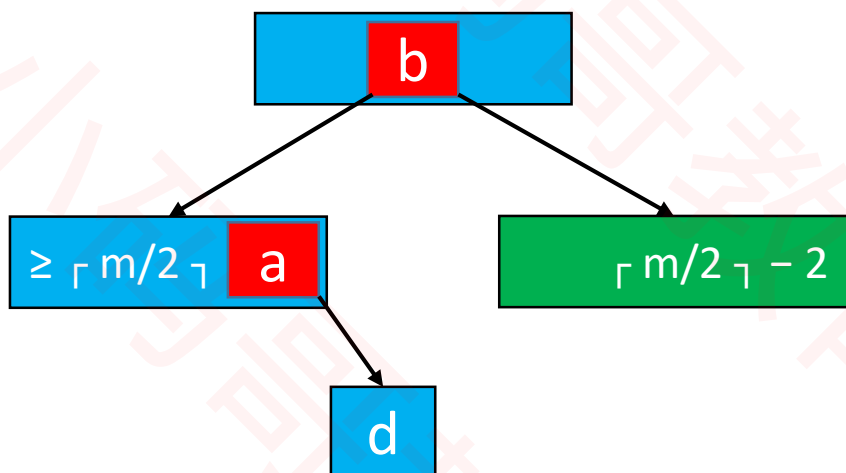
■ 删除 22 ? (假设这是一棵 5 阶 B 树)

□ 叶子节点被删掉一个元素后, 元素个数可能会低于最低限制 ($\geq \lceil m/2 \rceil - 1$)

□ 这种现象称为: 下溢 (underflow)

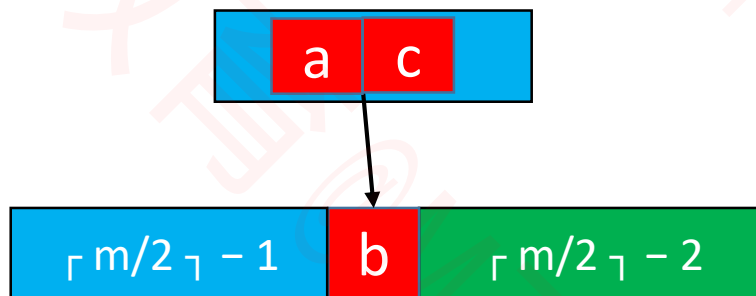
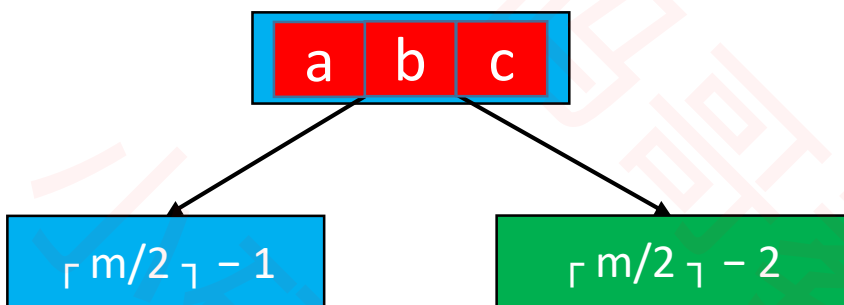
删除 - 下溢的解决

- 下溢节点的元素数量必然等于 $\lceil m/2 \rceil - 2$
- 如果下溢节点临近的兄弟节点，有至少 $\lceil m/2 \rceil$ 个元素，可以向其借一个元素
- 将父节点的元素 b 插入到下溢节点的 0 位置（最小位置）
- 用兄弟节点的元素 a （最大的元素）替代父节点的元素 b
- 这种操作其实就是：旋转

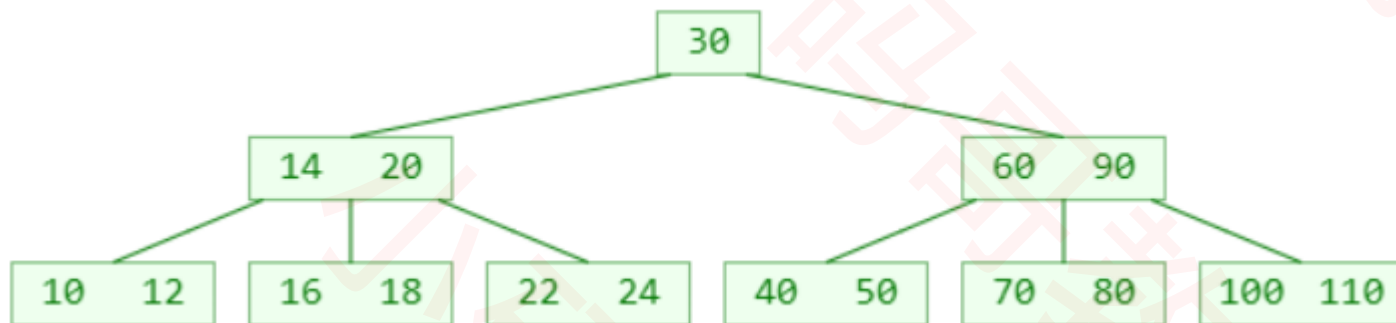


删除 - 下溢的解决

- 如果下溢节点临近的兄弟节点，只有 $\lceil m/2 \rceil - 1$ 个元素
- 将父节点的元素 b 挪下来跟左右子节点进行合并
- 合并后的节点元素个数等于 $\lceil m/2 \rceil + \lceil m/2 \rceil - 2$ ，不超过 $m - 1$
- 这个操作可能会导致父节点下溢，依然按照上述方法解决，下溢现象可能会一直往上传播



删除



■ 删除 22 ? (假设这是一棵 5阶B树)



4阶B树

■ 如果先学习4阶B树（2-3-4树），将能更好地学习理解红黑树

4阶B树的性质

□ 所有节点能存储的元素个数 x : $1 \leq x \leq 3$

□ 所有非叶子节点的子节点个数 y : $2 \leq y \leq 4$

■ 添加

□ 从 1 添加到 22

■ 删除

□ 从 1 删除到 22

