左偏树的特点及其应用

广东省中山市第一中学　黄源河

左偏树的定义

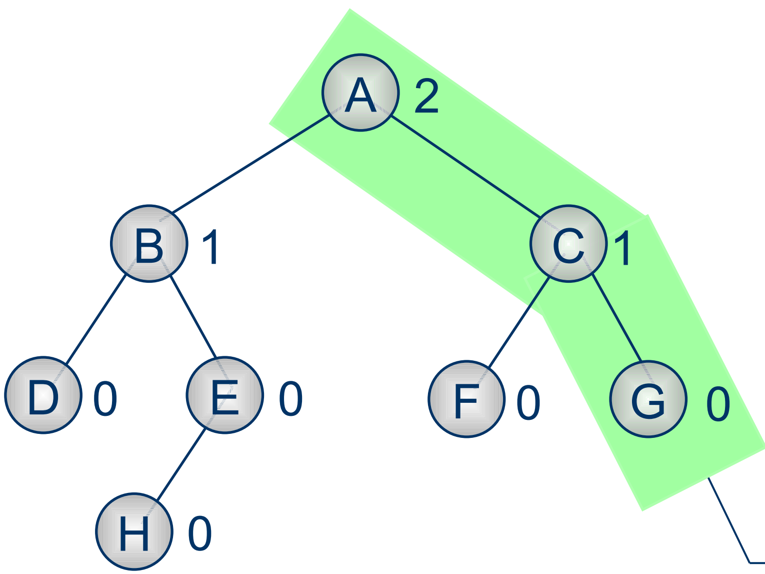
- **左偏树** (Leftist Tree) 是一种**可并堆** (Mergeable Heap) ，它除了支持优先队列的三个基本操作（插入，删除，取最小节点），还支持一个很特殊的操作——合并操作。
- 左偏树是一棵**堆有序** (Heap Ordered) 二叉树。
- 左偏树满足**左偏性质** (Leftist Property) 。

左偏树的定义 —— 左偏性质

- 定义一棵左偏树中的**外节点** (External Node) 为左子树或右子树为空的节点。
- 定义节点 i 的**距离** (dist(i)) 为节点 i 到它的后代中，最近的外节点所经过的边数。
- 任意节点的左子节点的距离不小于右子节点的距离（左偏性质）。
- 由左偏性质可知，一个节点的距离等于以该节点为根的子树最右路径的长度。

左偏树的性质

- 定理：若一棵左偏树有 N 个节点，则该左偏树的距离不超过 ô log(N+1)⌋ -1 。



最右路径 : A — C — G
最右路径节点数 = 3
距离 = 2

8 个节点的左偏树的最大距离：ô log(8+1)⌋ -1 = 2

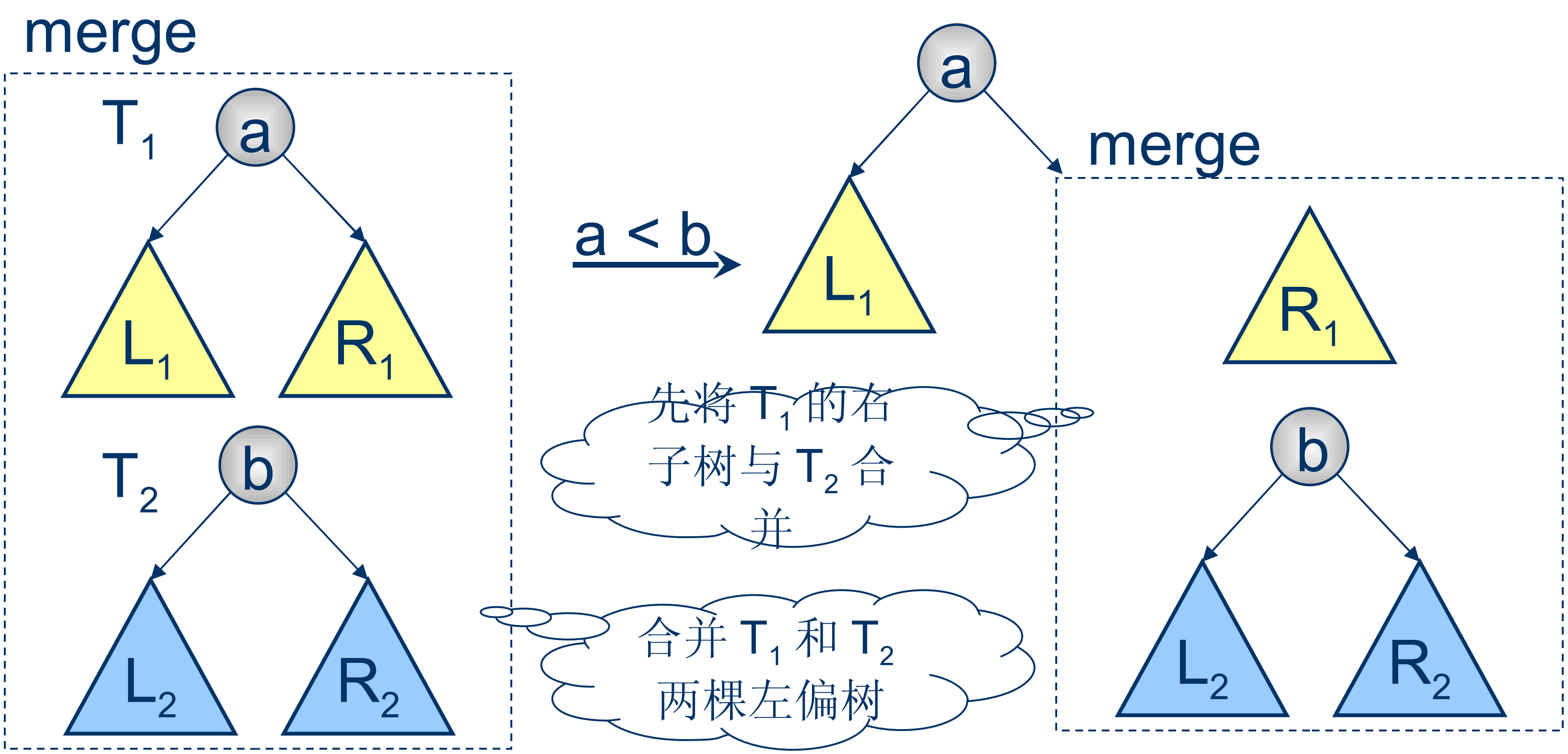
最右路径长度即为左偏树的距离

左偏树的操作

- 左偏树支持下面这些操作：
  - MakeNull —— 初始化一棵空的左偏树
  - Merge —— 合并两棵左偏树
  - Insert —— 插入一个新节点
  - Min —— 取得最小节点
  - DeleteMin —— 删除最小节点
  - Delete —— 删除任意已知节点
  - Decrease —— 减小一个节点的键值

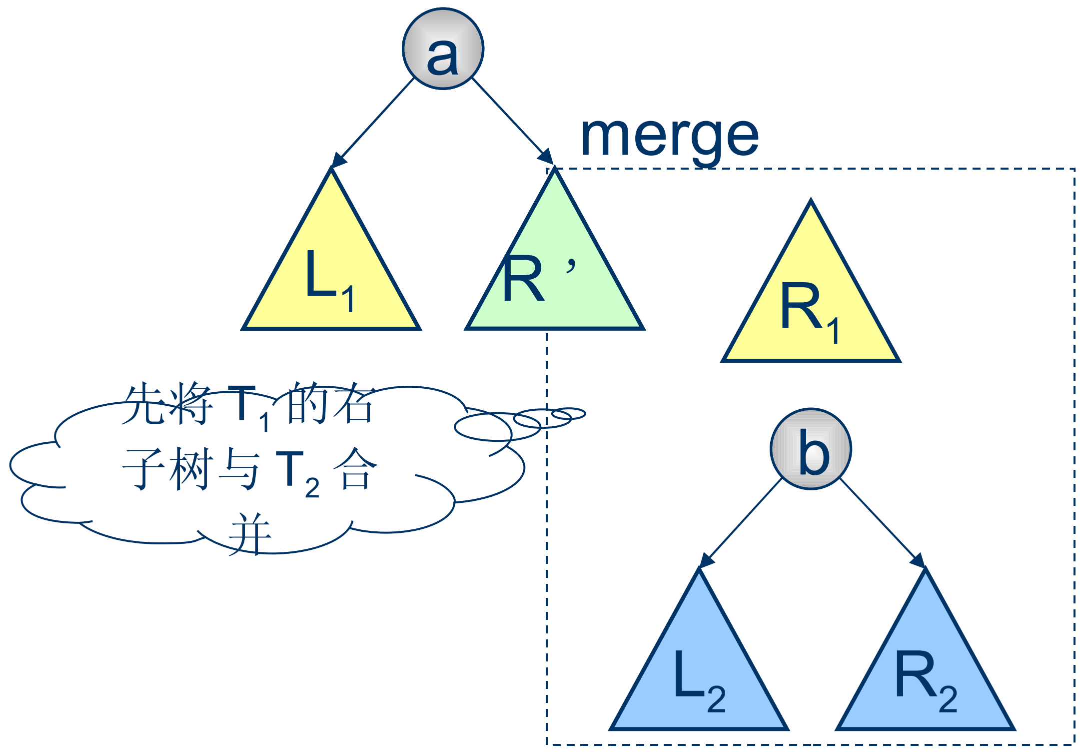
左偏树的操作 —— 合并

- 合并操作是递归进行的



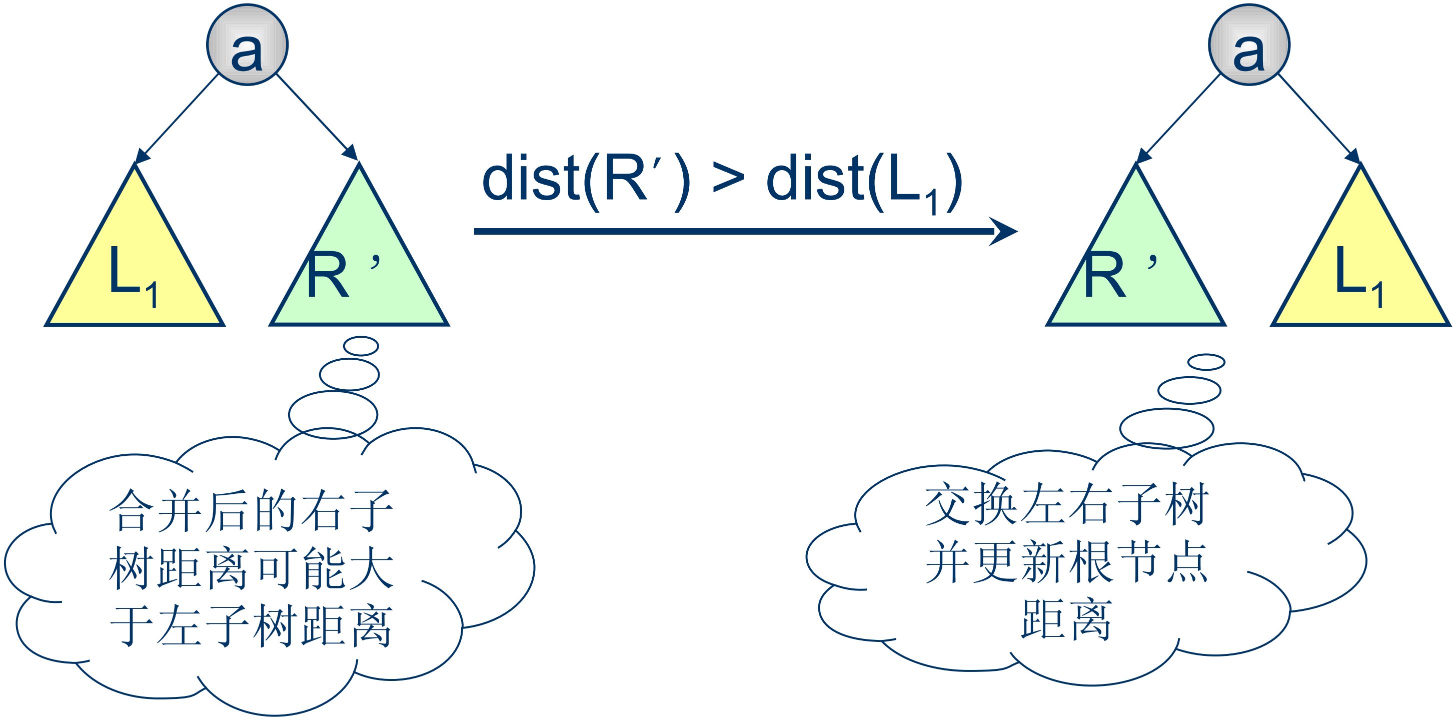
左偏树的操作 —— 合并

- 合并操作是递归进行的



左偏树的操作 —— 合并

- 合并操作是递归进行的



左偏树的操作 —— 合并

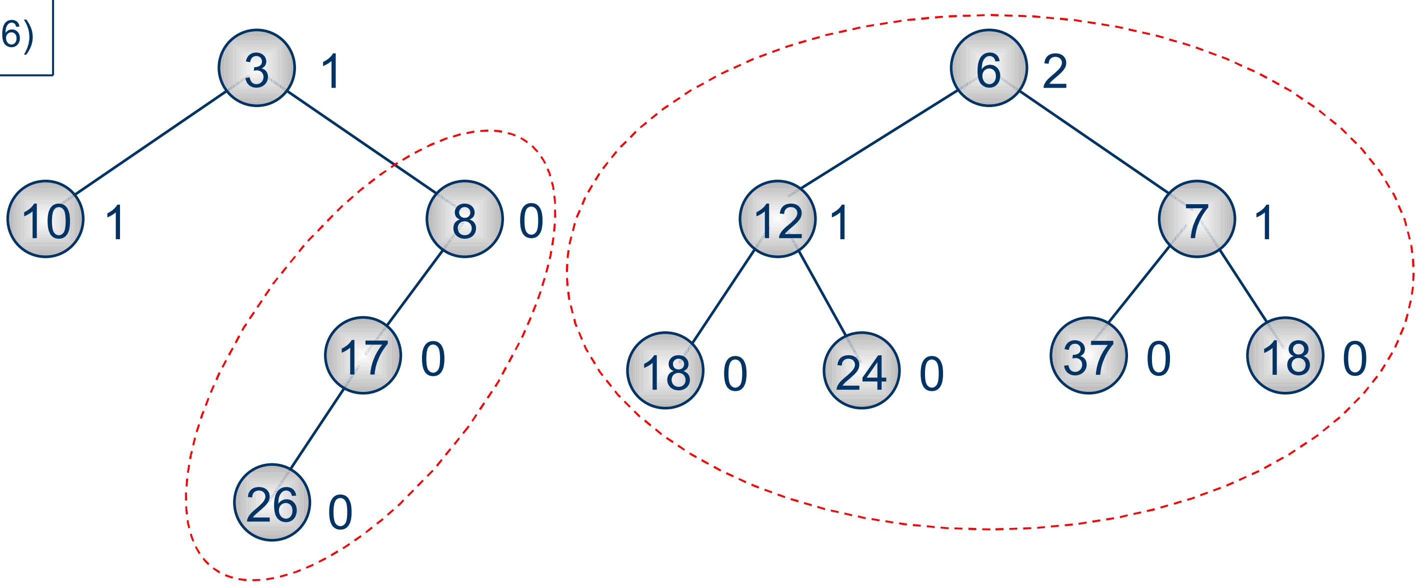
- 合并操作的代码如下：

```
Function Merge(A, B)
    If A = NULL Then return B
    If B = NULL Then return A
    If key(B) < key(A) Then swap(A, B)
    right(A) ← Merge(right(A), B)
    If dist(right(A)) > dist(left(A)) Then
            swap(left(A), right(A))
    If right(A) = NULL Then dist(A) ← 0
    Else dist(A) ← dist(right(A)) + 1
    return A
End Function
```

左偏树的操作 —— 合并

- 下面是一个合并的例子：

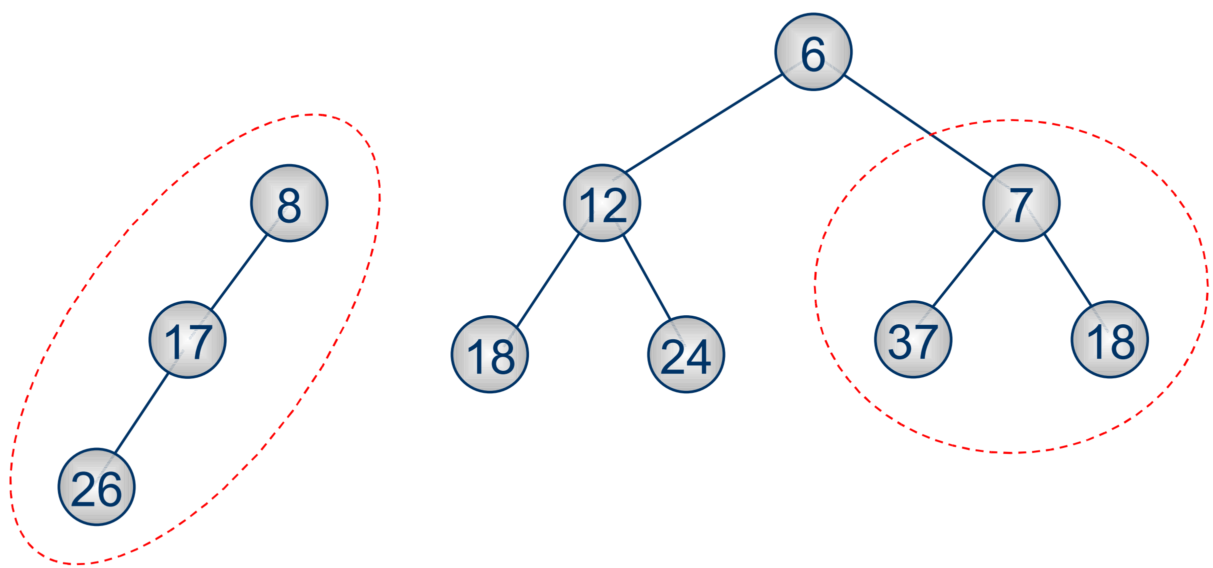
Merge (3, 6)



左偏树的操作 —— 合并

- 下面是一个合并的例子：

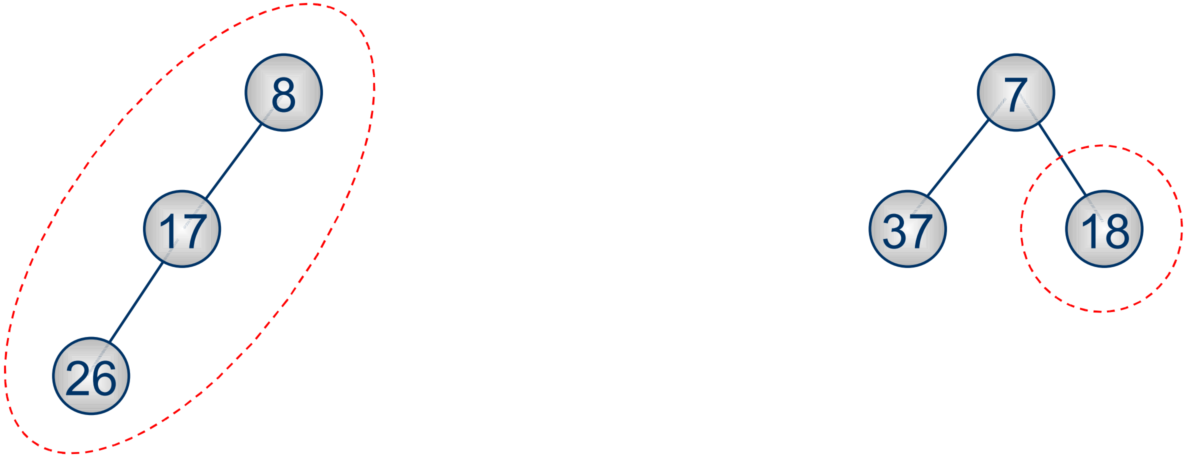
Merge (8, 6)
Merge (3, 6)



左偏树的操作 —— 合并

- 下面是一个合并的例子：

Merge (8, 7)
Merge (8, 6)
Merge (3, 6)



左偏树的操作 —— 合并

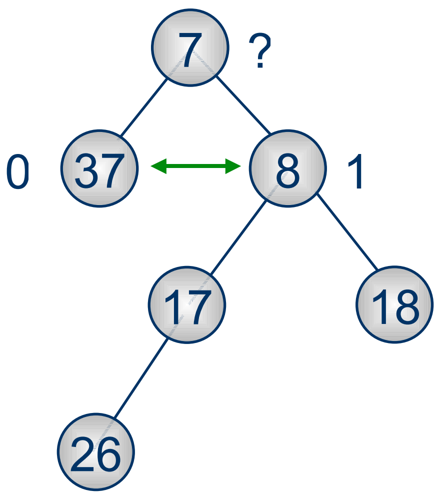
- 下面是一个合并的例子：

Merge (8,18)
Merge (8, 7)
Merge (8, 6)
Merge (3, 6)

左偏树的操作 —— 合并

- 下面是一个合并的例子：

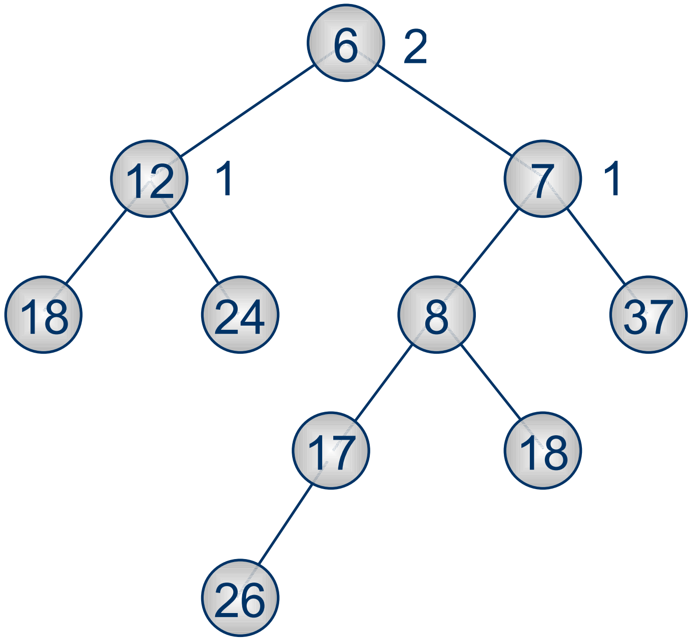
Merge (8, 7)
Merge (8, 6)
Merge (3, 6)



左偏树的操作 —— 合并

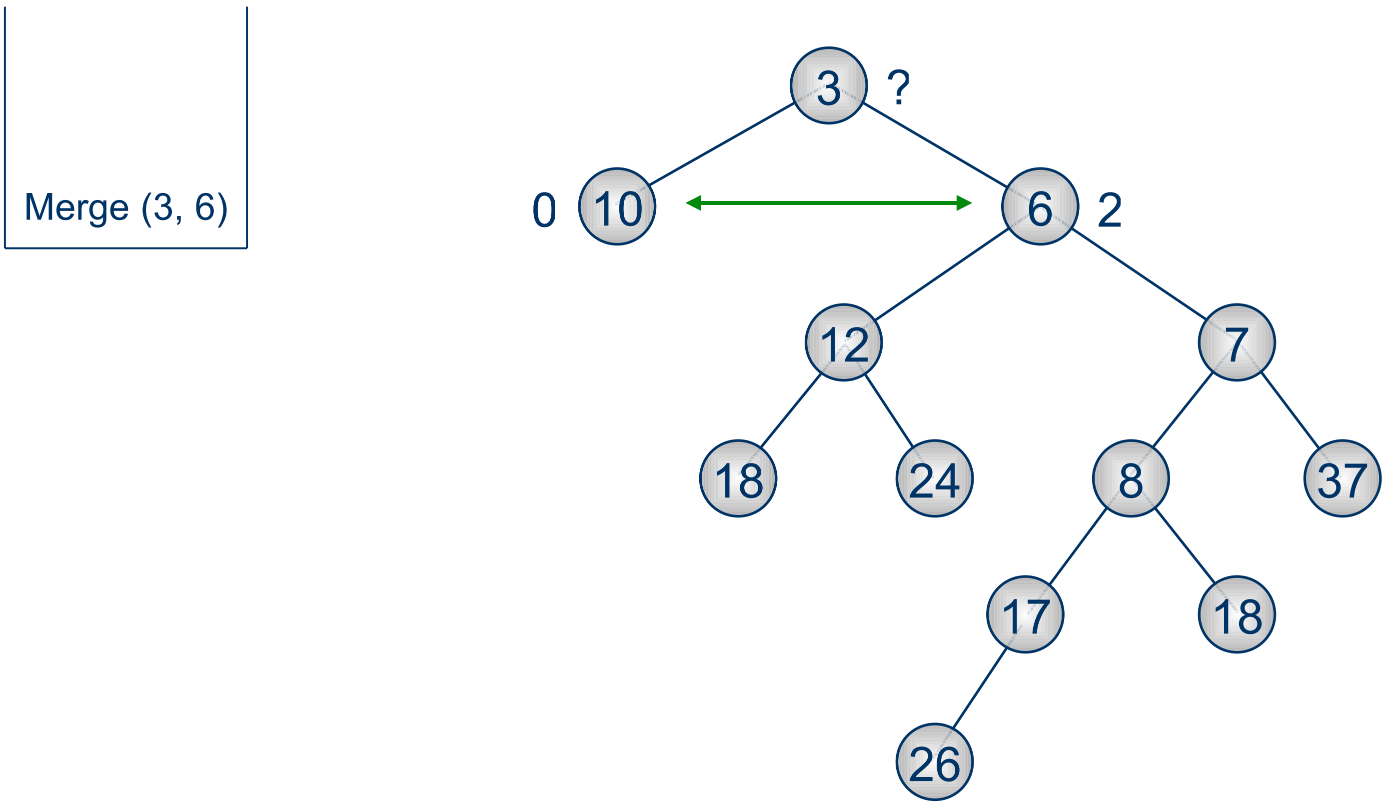
- 下面是一个合并的例子：

Merge (8, 6)
Merge (3, 6)



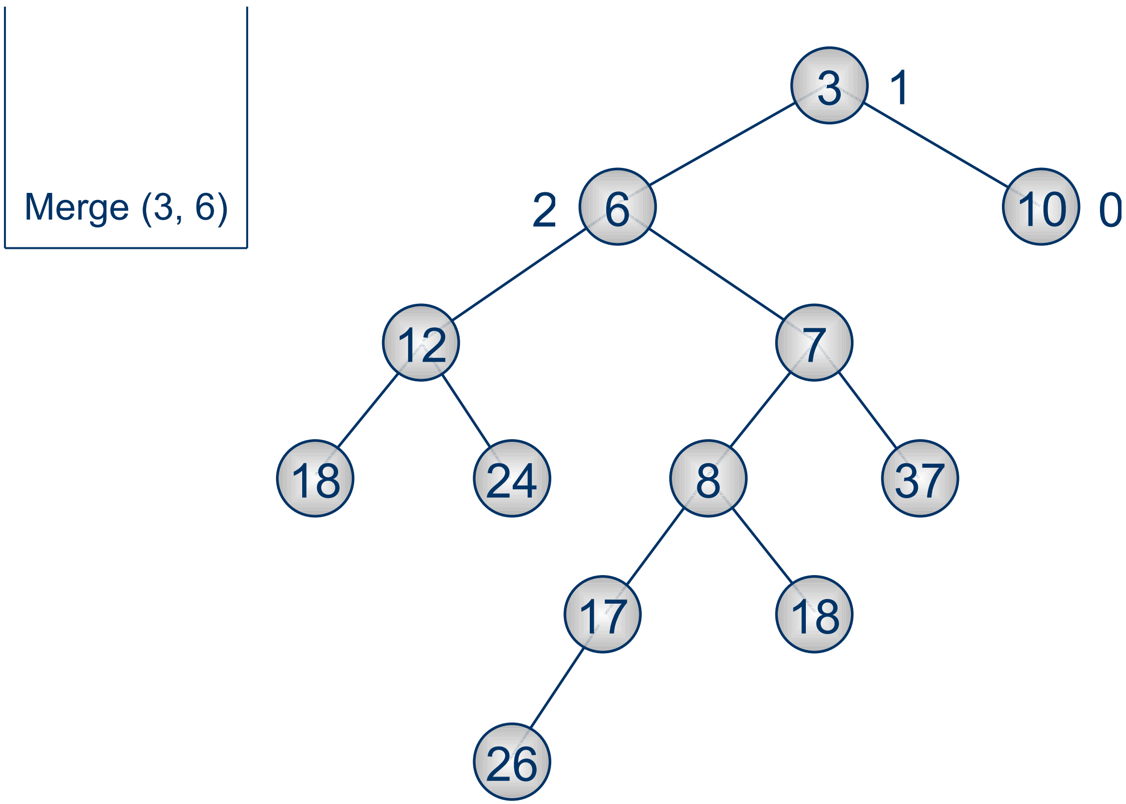
左偏树的操作 —— 合并

- 下面是一个合并的例子：



左偏树的操作 —— 合并

- 下面是一个合并的例子：



左偏树的操作 —— 合并

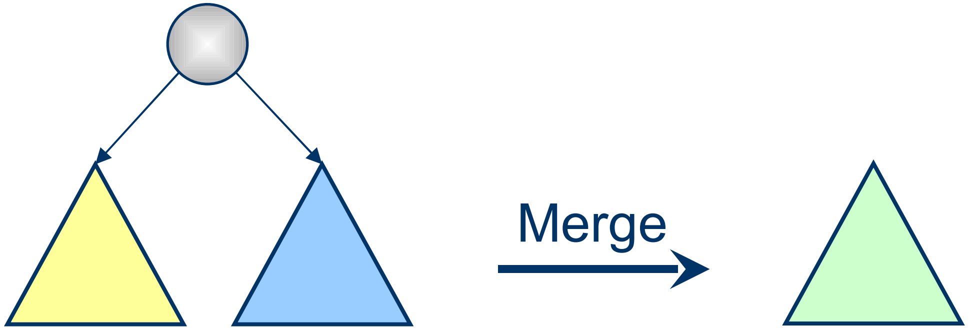
- 合并操作都是一直沿着两棵左偏树的最右路径进行的。
- 一棵 N 个节点的左偏树，最右路径上最多有 ô log(N+1)⌋ 个节点。
- 因此，合并操作的时间复杂度为：$O(\log N_1 + \log N_2) = O(\log N)$

左偏树的操作 —— 插入

- 插入一个新节点
  - 把待插入节点作为一棵单节点左偏树
  - 合并两棵左偏树
  - 时间复杂度： O(log N)

左偏树的操作 —— 删除

- 删除最小节点
  - 删除根节点
  - 合并左右子树
  - 时间复杂度： O(log N)



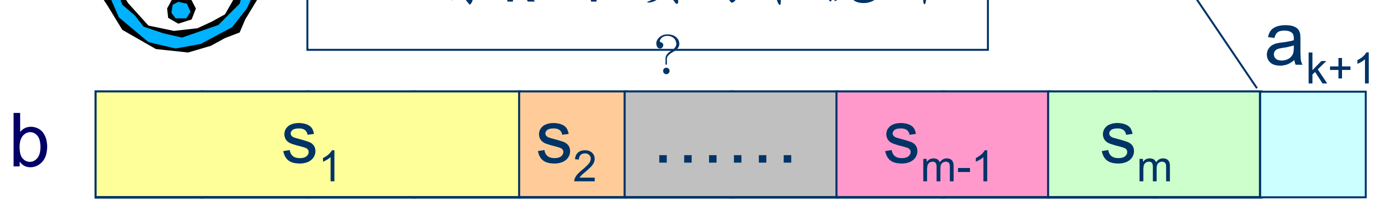
例题：数字序列

- 给定一个整数序列 $a_1, a_2, \ldots, a_n$，求一个不下降序列 $b_1 \le b_2 \le \ldots \le b_n$，使得数列 $\{a_i\}$ 和 $\{b_i\}$ 的各项之差的绝对值之和 $|a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$ 最小。
- 数据规模： $1 \le n \le 10^6$, $0 \le a_i \le 2*10^9$

例题：数字序列 —— 算法分析

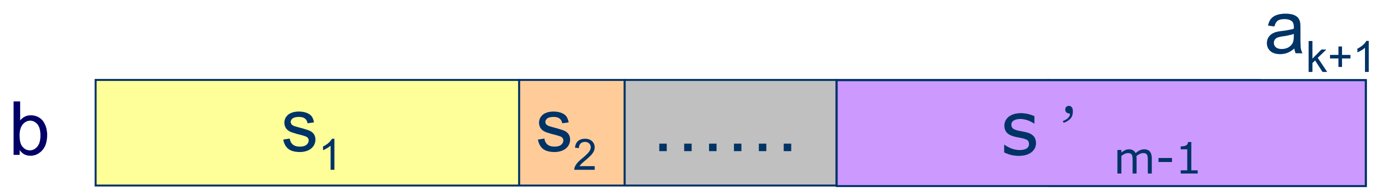
- 假设数列 $a_1, a_2, \ldots, a_k$ 的最优解为 $b_1, b_2, \ldots, b_k$
- 合并 $\{b_i\}$ 中相同的项，得到 m 个区间和数列 $s_1, s_2, \ldots, s_m$
- 显然，s_i 为数列 a 中，下标在第 i 个区间内的各项的

加入 a_{k+1} 后，怎样得到前 k+1 项的最优解？



例题：数字序列 —— 算法分析

- 若 $a_{k+1}>s_m$，直接令 $s_{m+1}=a_{k+1}$，得到前 k+1 项的最优解；
- 否则，将 a_{k+1} 并入第 m 个区间，并更新 s_m
- 不断 检查最后两个区间的解 s_{m-1} 和 s_m，若 $s_{m-1}\geq s_m$，合并最后两个区间，并令新区间的解为该区间内的中位数。



例题：数字序列 —— 算法分析

- 下面考虑数据结构的选取
- 我们需要维护若干个有序集，并能够高效完成下面两个操作：
  - 合并两个有序集
  - 查询某个有序集的中位数
- 进一步分析，加入一个元素后，发生一连串合并操作，合并后有序集的中位数不会比原来大
- 因此，每个有序集内只保存较小的一半元素，查询中位数操作转化为取最大元素操作。

例题：数字序列 —— 算法分析

- 现在，我们需要合并、取最大元素和删除三种操作，而这些都是可并堆的基本操作。
- 下表列出了几种可并堆相应操作的时间复杂度

操作	二叉堆	左偏树	二项堆	Fibonacci堆
取最小节点	O(1)	O(1)	O(1)	O(1)
插入	O(log N)	O(log N)	O(1)	O(1)
删除最小节点	O(log N)	O(log N)	O(log N)	O(log N)
合并	O(N)	O(log N)	O(log N)	O(1)

例题：数字序列 —— 算法分析

- 在本题中，合并操作和取最大元素操作少于 n 次，删除操作不超过 n/2 次
- 由于合并次数比较多，二叉堆的合并操作太慢了，总时间复杂度也无法令人满意。
- 二项堆和 Fibonacci 堆某些操作比左偏树快，但对于本题，三者的总时间复杂度均为 O(nlogn)
- 二项堆和 Fibonacci 堆的空间需求比较大，编程实现也远没有左偏树简单。
- 相比之下，本题用左偏树实现，时空复杂度都可以接受，编程实现也非常简单，是十分理想的选择。

总结

- 左偏树的特点：
  - 时空效率高
  - 编程复杂度低

性价比高

- 左偏树的应用：
  - 可并堆
  - 优先队列

补充二叉堆的不足

谢谢大家