堆

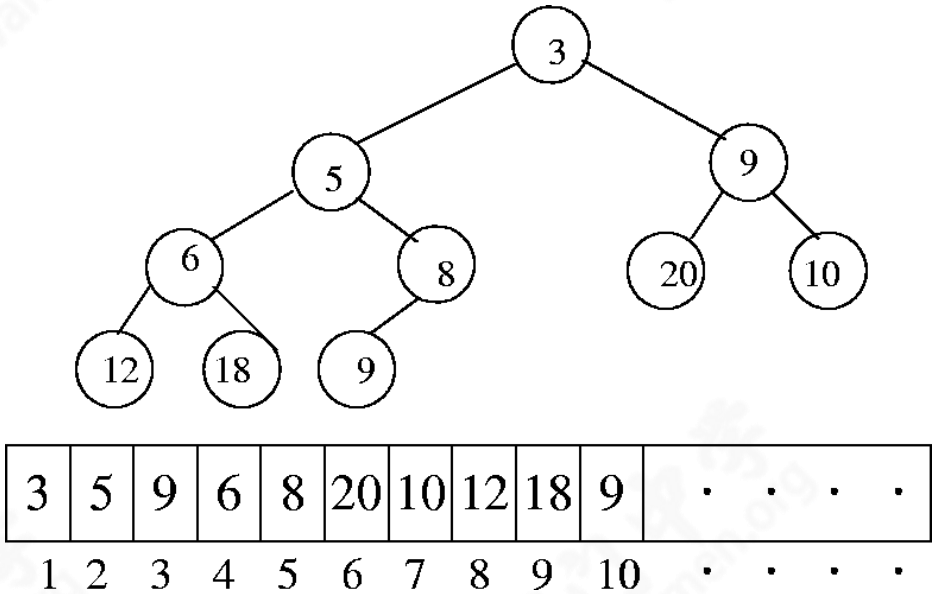
主讲人：邓哲也

大纲

- 堆的定义
- 堆的性质
- 堆的基本操作
- 完全二叉树的实现

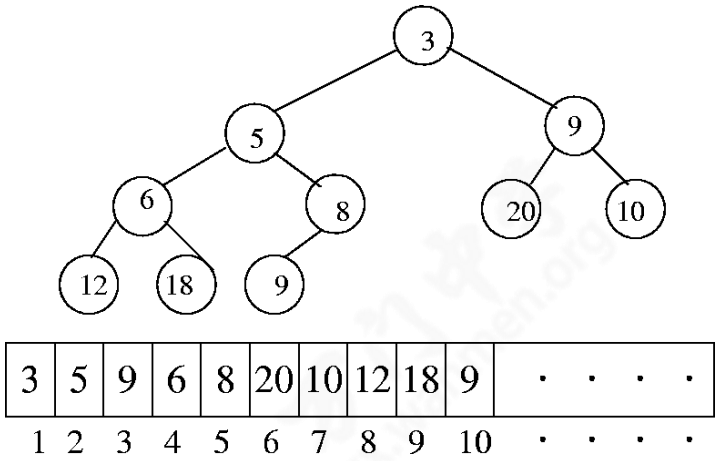
堆的定义

- 最小堆：最小堆是一个关键码序列{K1, K2, …, Kn}，它具有如下特性：
  - $K_i \leq K_{2i}$
  - $K_i \leq K_{2i+1}$
- 类似可以定义最大堆。



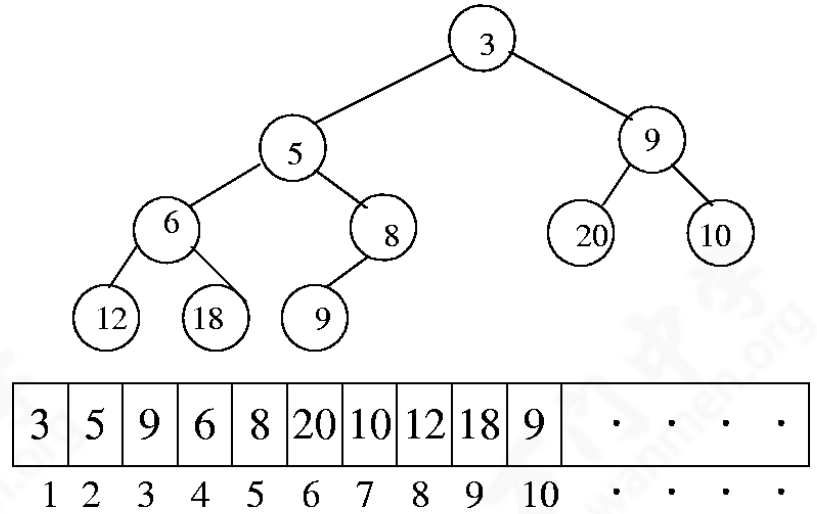
堆的性质

- 完全二叉树的层次序列，可以用数组表示。
- 堆中储存的数是局部有序的，堆不唯一。
  - 节点的值与其孩子的值之间存在限制。
  - 任何一个节点与其兄弟之间都没有直接的限制。
- 从逻辑角度看，堆实际上是一种树形结构。



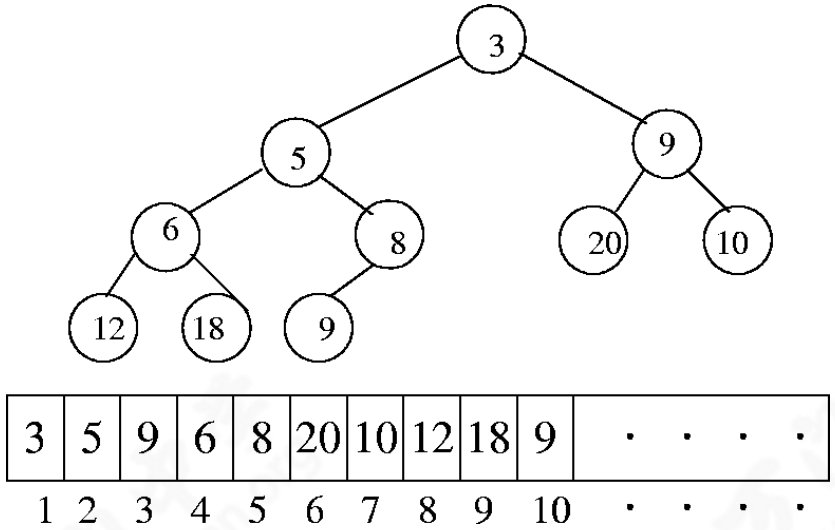
维护堆的性质

- **shift_up**
  - 每次将该元素与其父元素比较，若该元素小于其父元素，则将其父元素与其交换位置，直到达到根节点或者该元素大于其父元素。
- **shift_down**
  - 每次将该元素与其左右子元素比较，若该元素大于任意一个左右子元素，则将其与左右子元素中的最小的那个交换位置，直到达到叶结点或者该元素小于其两个左右子元素。



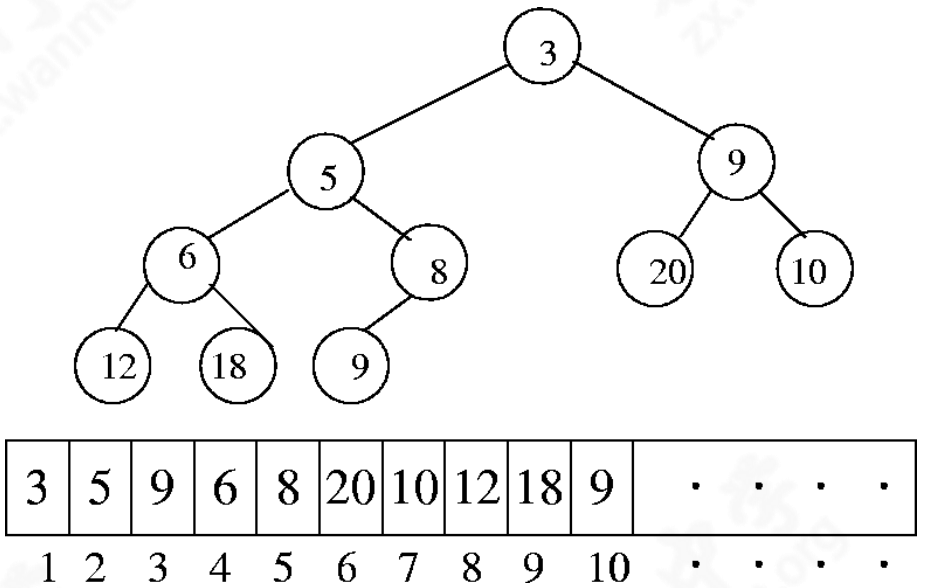
最小堆与二叉搜索树的区别

- 对于二叉搜索树的任意一个节点：
  - 左子树的值都小于这个节点的值
  - 右子树的值都大于这个节点的值
  - 两个子树都是二叉搜索树
- 对于最小堆的任意一个节点：
  - 所有的子节点值都大于这个节点的值
  - 两个子树都是最小堆



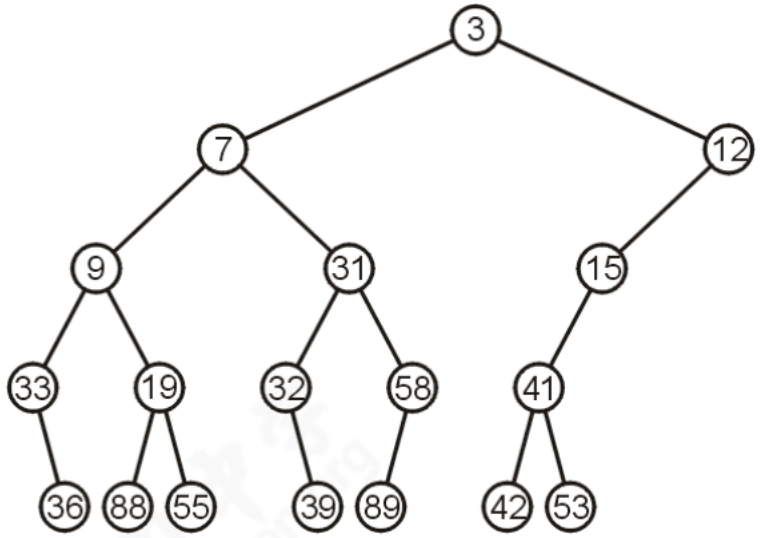
堆的基本操作

- top
  - 返回当前的最小值
- pop
  - 弹出当前的最小值
- push
  - 插入一个数



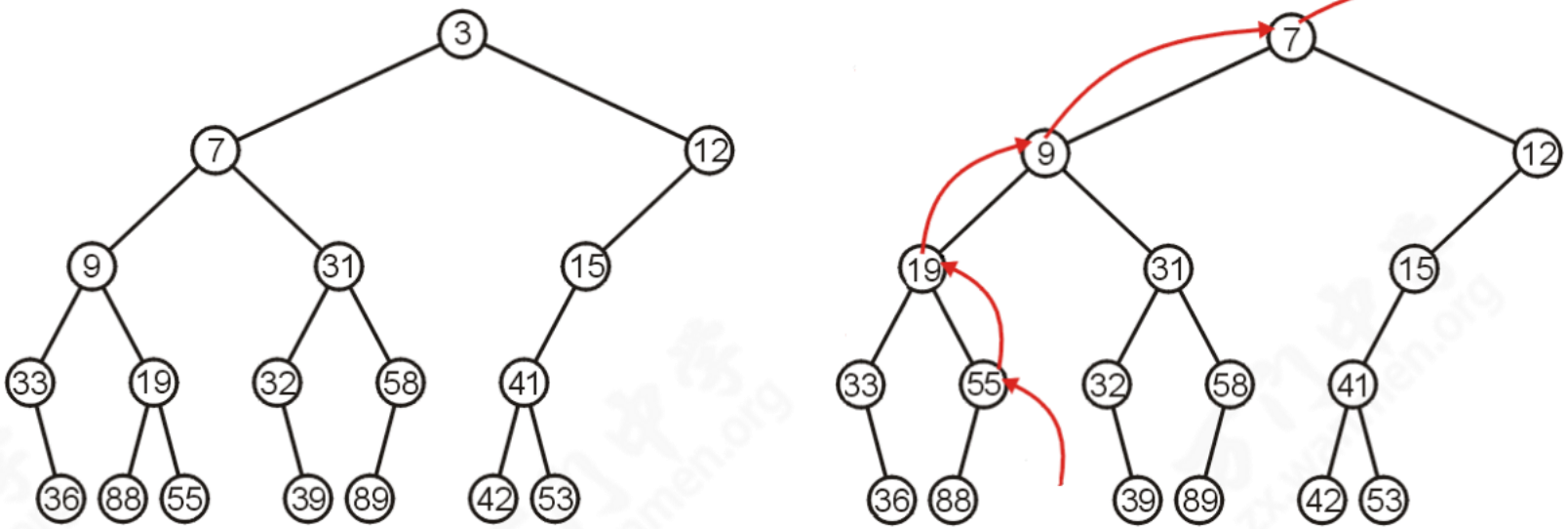
堆的基本操作

- top
  - 直接返回根结点
  - 时间复杂度O(1)



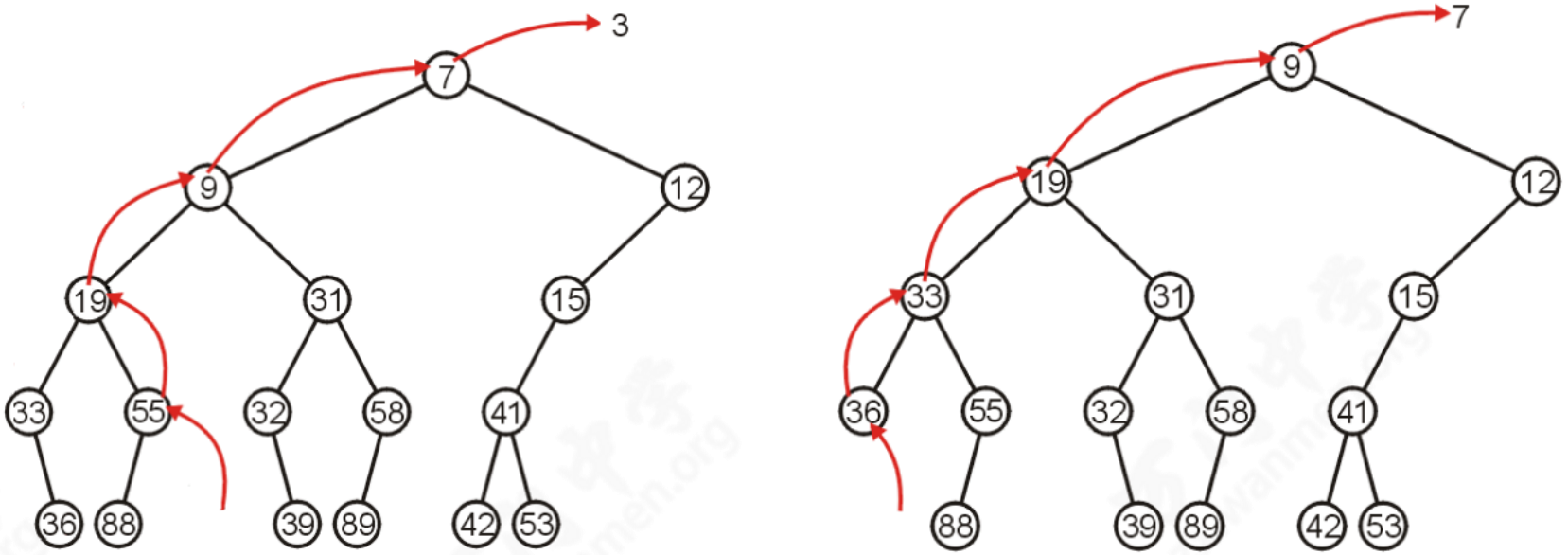
堆的基本操作

- **pop**
  - 删掉根节点
  - 然后递归把子节点补上来



堆的基本操作

- **pop**
  - 删掉根节点
  - 然后递归把子节点补上来

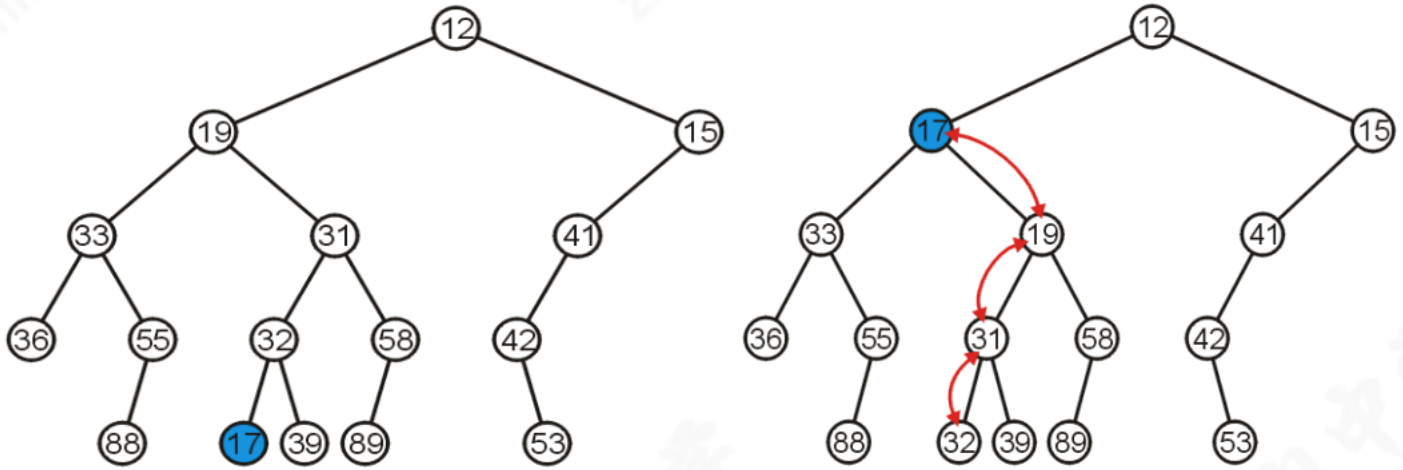


堆的基本操作

- push的时候我们有两种思路：
  - 插入根
  - 插入一个空节点
- 第一种情况，我们可以和当前节点比较，保留小的，大的递归下去插入子树。
- 第二种情况，如果当前节点的值小于父节点的值，就和父节点交换，直到大于父节点的值或成为根节点。

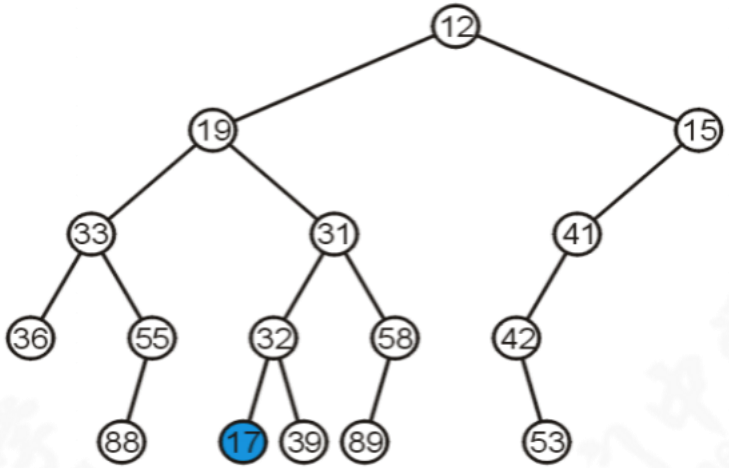
堆的基本操作

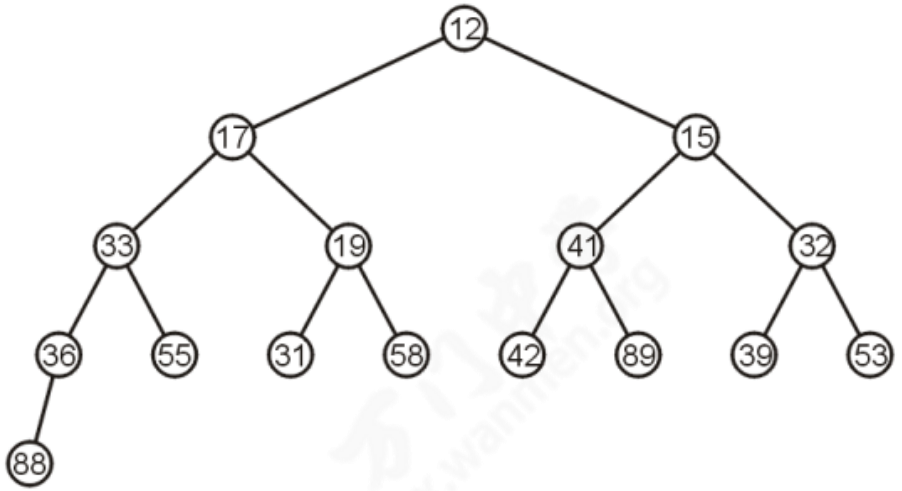
- **push**


12
19
15
33
31
41
36
55
32
58
42
88
17
39
89
53
12
17
15
33
19
41
36
55
31
58
42
88
32
39
89
53

完全二叉树的实现

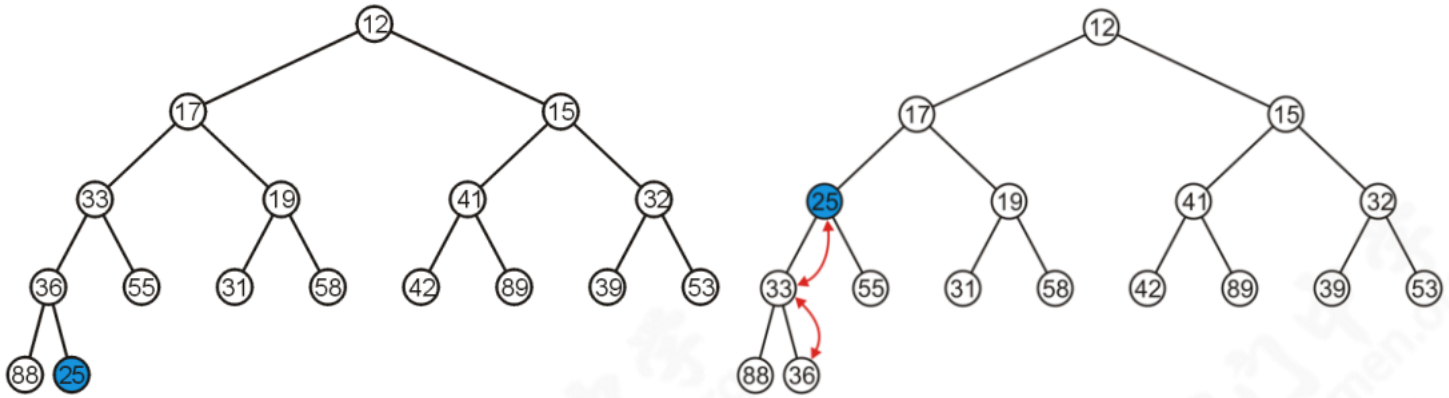
- 有很多种数据结构来实现最小堆，包括：完全二叉树，左偏树，斜堆，二项堆，斐波那契堆等等。完全二叉树是这里面内存利用效率最高的。





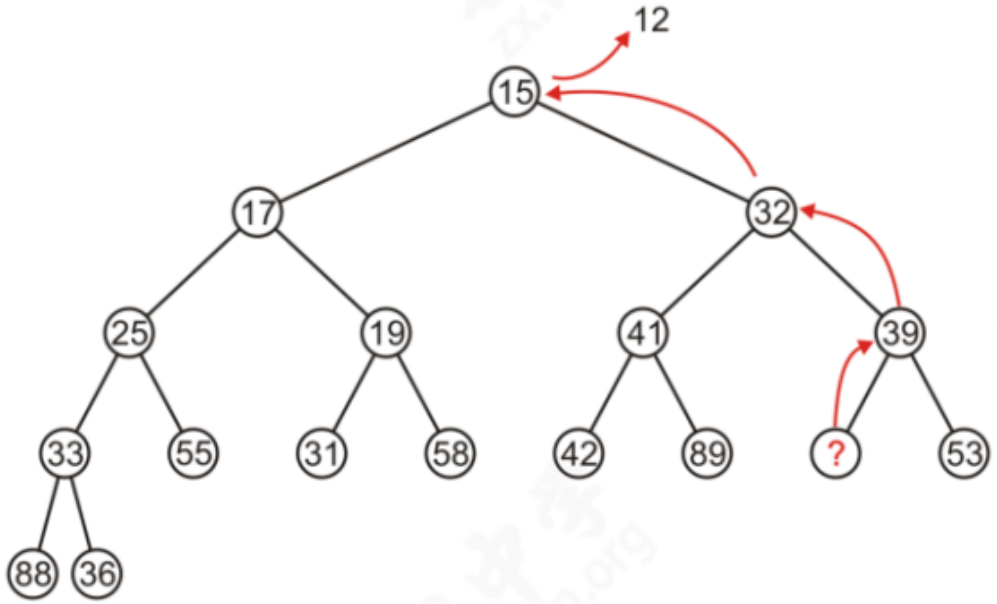
完全二叉树的实现

- 这个时候，选择插入新节点的位置变得更加简单：只有一个位置可以选择



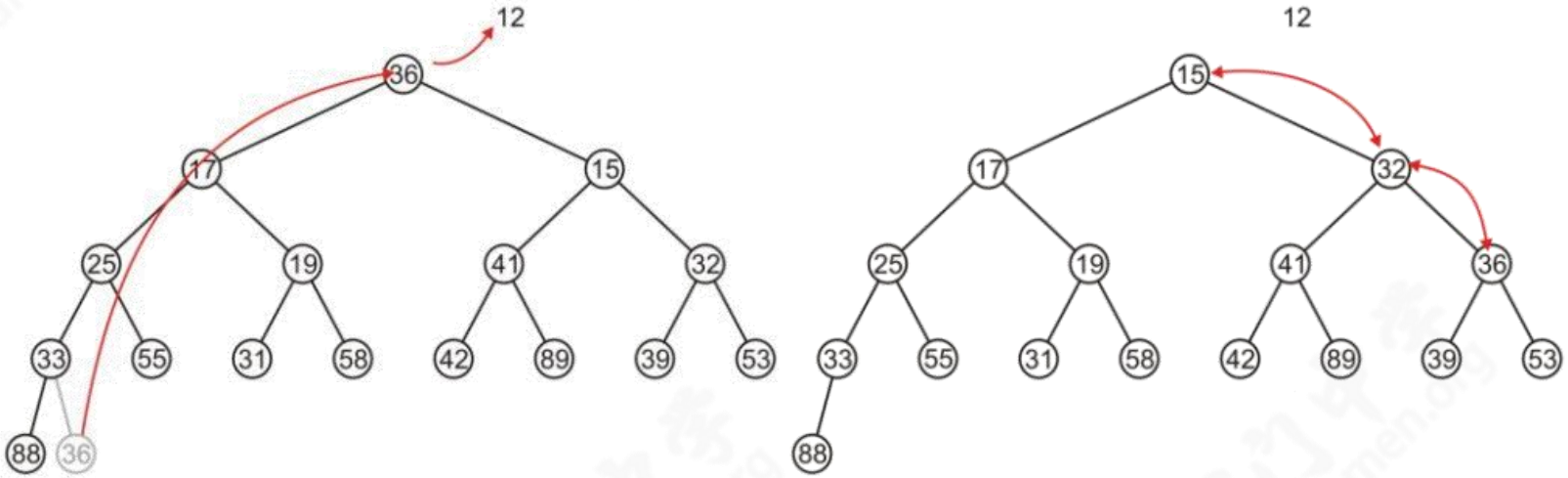
完全二叉树的实现

- 但是pop操作用之前的策略就会破坏完全二叉树的性质。



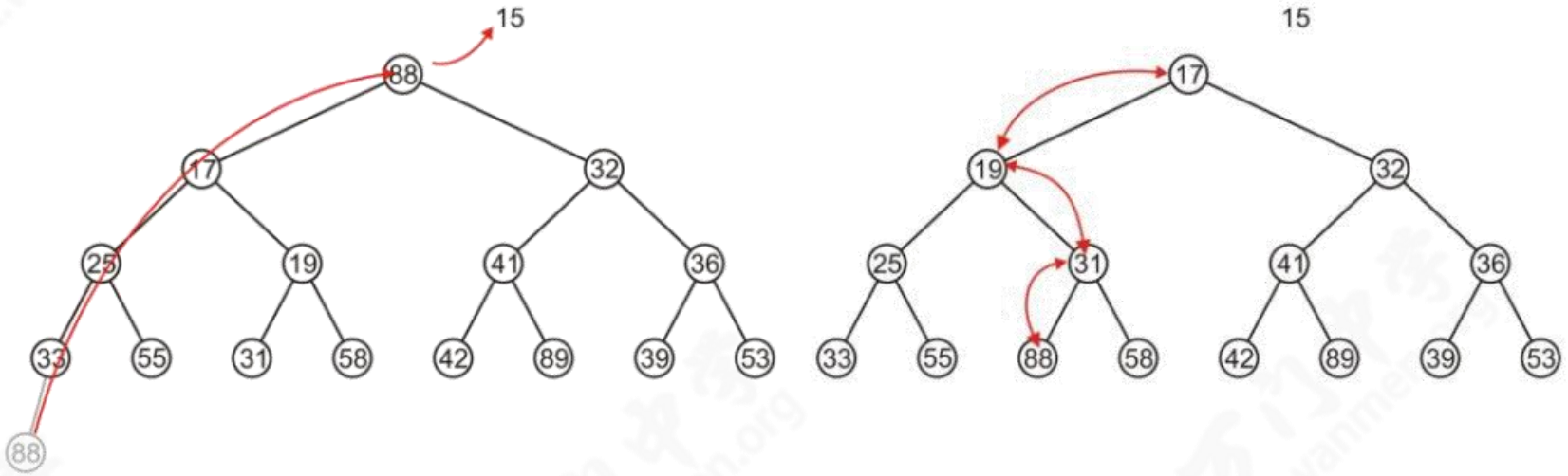
完全二叉树的实现

- 改变策略，先把根节点删除，把最后一个节点移到根，然后从上往下更新。



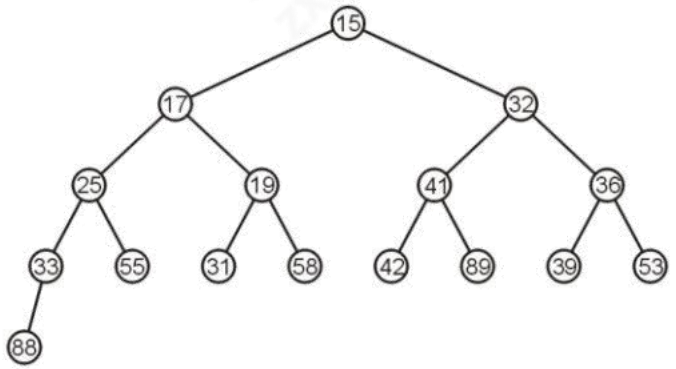
完全二叉树的实现

- 改变策略，先把根节点删除，把最后一个节点移到根，然后从上往下更新。



完全二叉树的实现

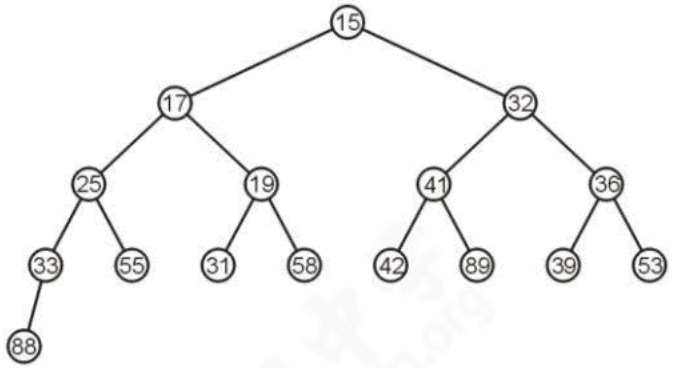
- 用数组存堆。下标为k的节点的子节点为2k和2k+1，父节点为k/2。



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	15	17	32	25	19	41	36	33	55	31	58	42	89	39	53	88

完全二叉树的实现

- 时间复杂度分析：
  - **pop和push操作都和树的高度正相关。**
  - 完全二叉树树高为log n
  - 故**pop和push**的时间复杂度均为O(log n)



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	15	17	32	25	19	41	36	33	55	31	58	42	89	39	53	88

下节课再见