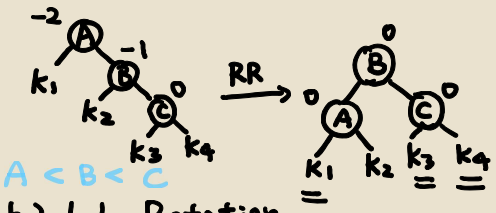
[AVL 树]

① Balance Factor：左右子树高度差($h_L - h_R$)

↳ AVL树 $\forall$ node, BF$\in\{0,\pm1\}$ 之二叉查找树.

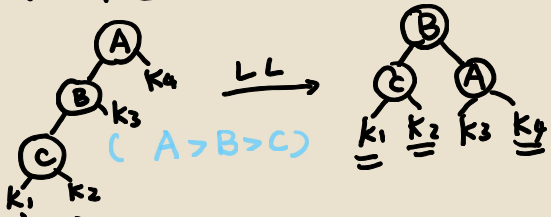
② AVL树的构建：[R/L][R/L]-Rotate

(a) RR Rotation



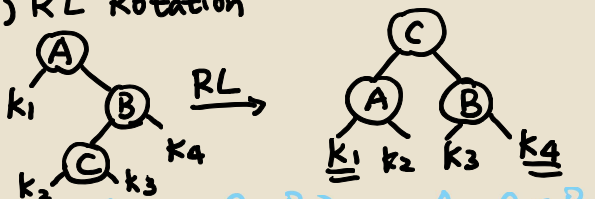
A < B < C

(b) LL Rotation



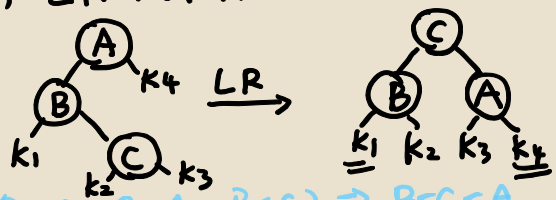
(A > B > C)

(c) RL Rotation



(A < B, A < C, C < B) ⇒ A < C < B

(d) LR Rotation



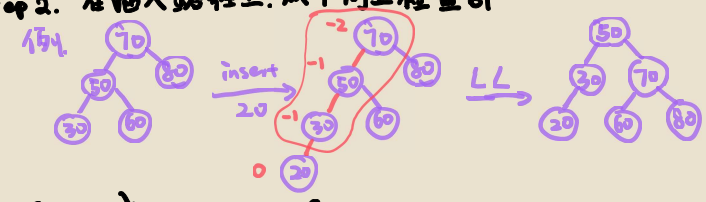
(B < A, C < A, B < C) ⇒ B < C < A

(i) insertion

Step 1. 二叉查找树插入

Step 2. 在插入路径上，从下向上检查BF

例.



Δ. 引入域 parent. bf.

③ 结论（摁背）

(a) Let n_h be min number of nodes in balanced (AVL) tree of height h.

$n_h = F_{h+2} - 1$, $F_i = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^i$ Fibonacci Sequence

(b) Amortized Analysis

(i) n节点树高 $O(\log n)$

(ii) 插入. 删除. 查找 $O(\log n)$

worst-case bound ≥ amortized bound ≥ average-case bound

关于 Amortized Analysis，摁背的结论.

① Initially empty stack : Push. Pop. PopAll (Multipop). $T = O(n)/n = 1$.

$\forall$ object. only 1 pop for 1 push

② Splay Tree . insertion . deletion & search. $O(\log n)$

Consecutive M operation. $T = O(M \log N)$

势能法. $\Phi(T) = \sum_{i\in T} R(i)$ $R(i) = \log$ (i节点后裔个数)

$\hat{C}_i = C_i + \Phi(D_i) - \Phi(D_{i-1})$

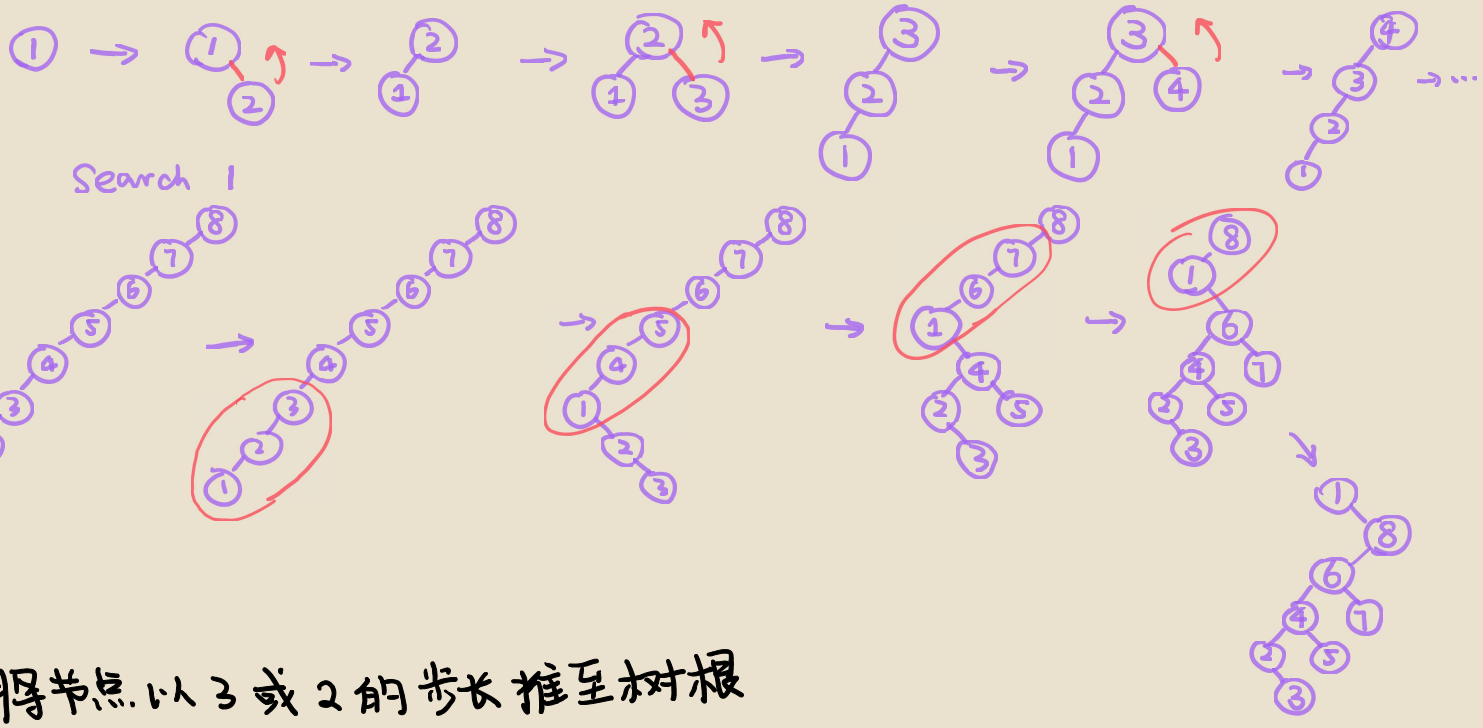
③ 二进制计数法器 INCREMENT $O(n)$

[Splay Tree]

△. 每次查找/插入节点时，将其推至树根

插入. 删除. 查找. $O(\log n)$

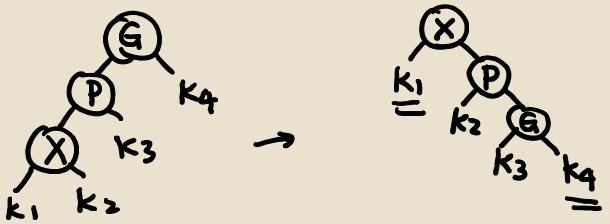
例： insert 1, 2, 3, ···, 8



将节点以3或2的步长推至树根

Case 1. 节点距树根大于等于2

Zig-Zig

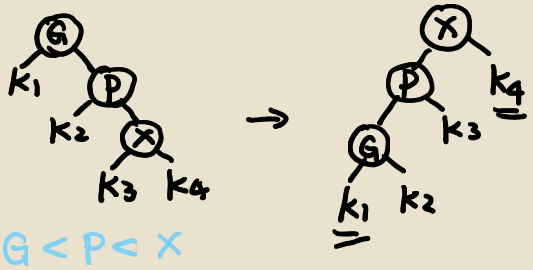


$X < P < G$

$P < k_3 < G$

$X < k_2 < P < G$

直观上. k_i左右顺序不变

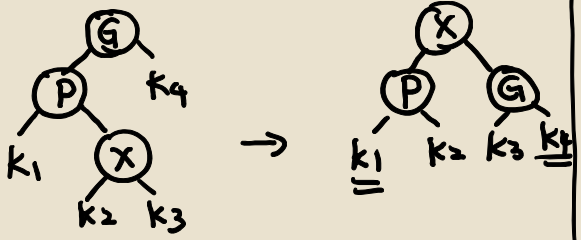


$G < P < X$

$P < k_2 < G$

$P < k_3 < X$

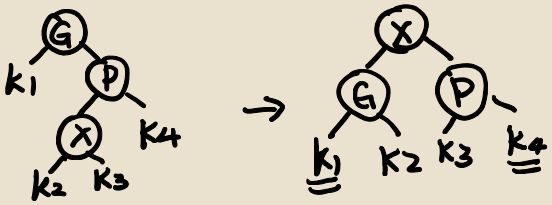
Zig-Zag



$P < X < G$

$P < k_2 < X$

$X < k_3 < G$

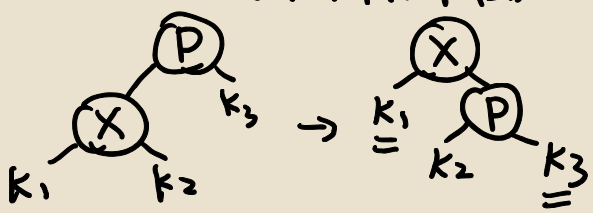


$G < X < P$

$G < k_2 < X$

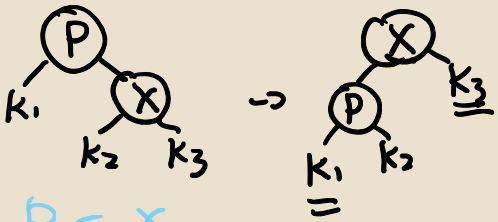
$X < k_3 < P$

Case 2. X与树根距离1.



$X < P$

$X < k_2 < P$

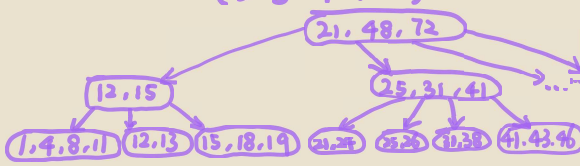


$P < X$

$P < k_2 < X$

[B+Tree]

example: A B+Tree of order 4 (2-3-4 Tree)



① Definition. A B+Tree of order M is a tree with the following structural props:

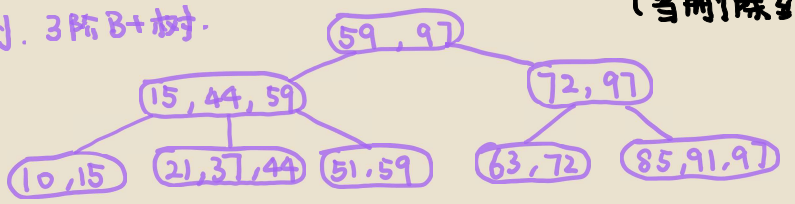
(1) The root is either a leaf or has [2, M] children
(2) All non-leaf nodes (except the root) have [ceil(M/2), M] children
(3) All leaves are at the same depth.

② Insertion & Deletion

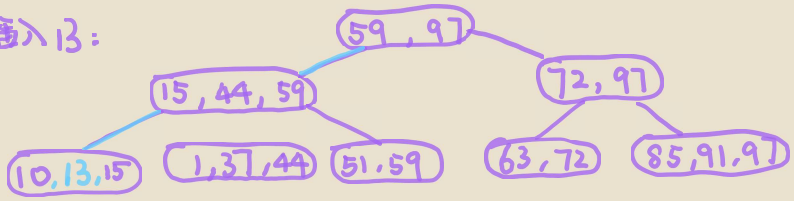
插入操作在叶节点进行. 并需适时分裂

删除操作在叶节点进行. 并需适时改变父节点中的索引, 并进行结点合并使其关键字个数补齐 M/2 个 (当删除到节点中最大值) 优先级: 1. 从兄弟节点中取1个 2. 与兄弟节点合并

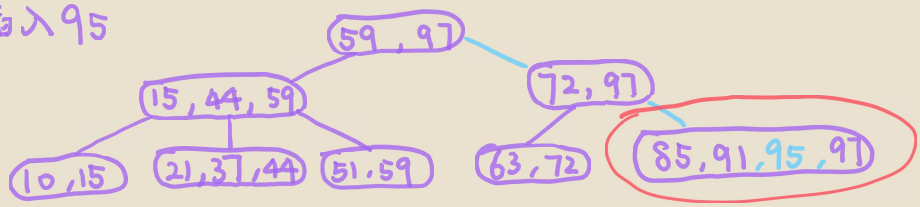
例. 3阶B+树:

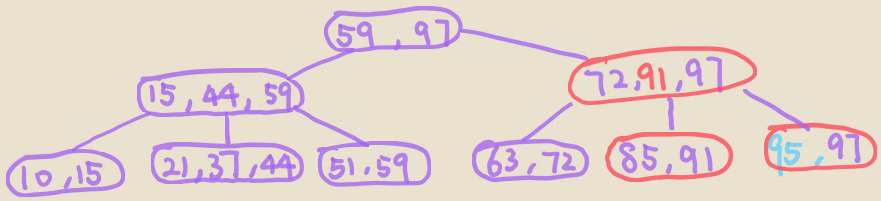


插入13:

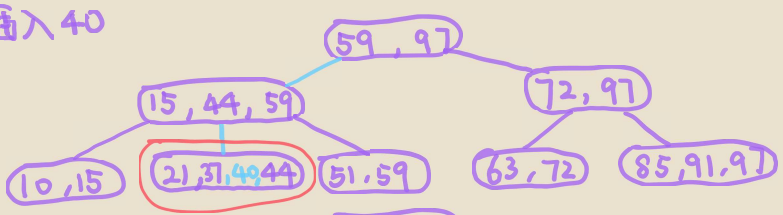


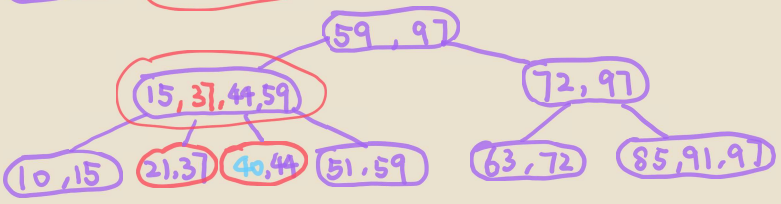
插入95

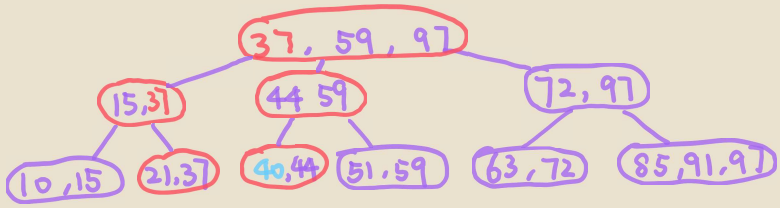




插入40

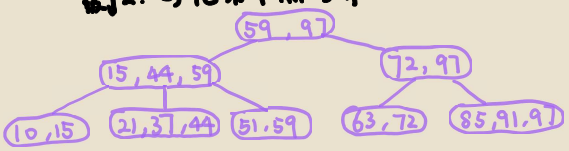




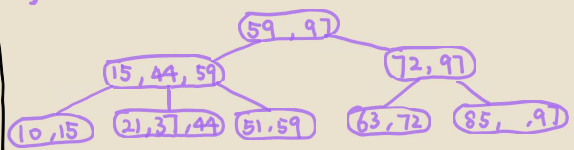


$O(\log n)$

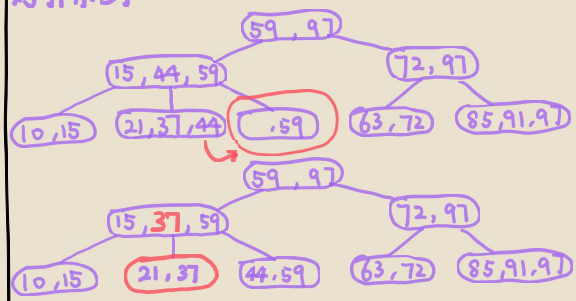
height.



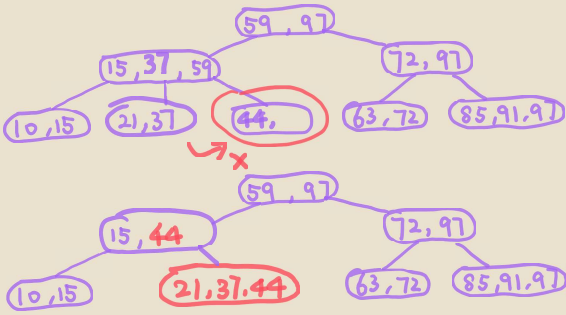
删除91



删除51



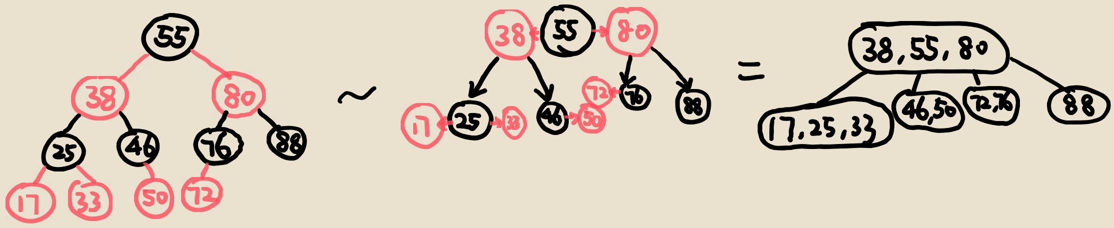
删除59(在上图中)

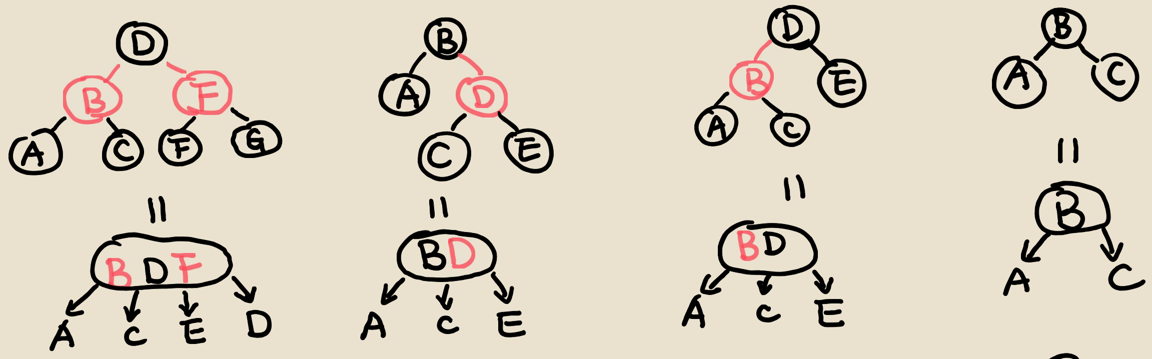


删除63 (在删除91后)

[Red - Black Tree] — BST

① RBT 与 4阶 B-Tree (3阶 ∨) 的转换





左 < 根 < 右 ① (BST)

⇒ [Prop.]
(i) 节点为红色 or 黑色
(ii) 根节点为黑色
(iii) 为每个叶节点补充2个 NIL子节点,叶节点为黑色 } 根叶黑 ②
(iv). 红色节点 子节点为黑色
- 红色节点 父节点为黑色
- 从根节点到任意叶节点,不存在2个连续红色节点 — 不红红 ④

(v) 任意节点的黑高相同 — 黑路同 ③

⇒ 左根右,根叶黑,黑路同,不红红

② Insertion. Deletion & Search is $O(\log n)$

③ 结论.

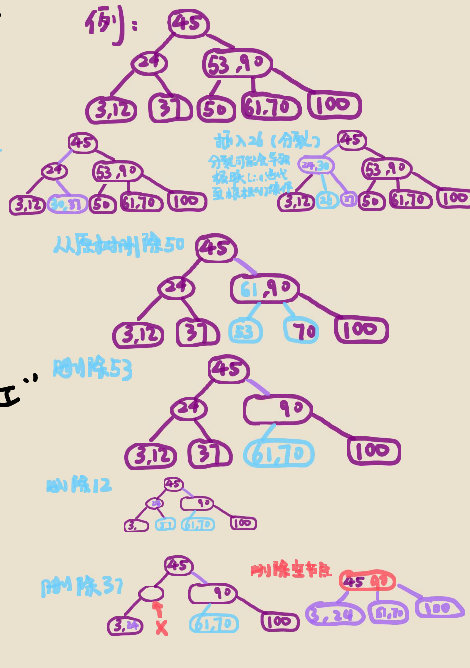
(i) 从根到任意叶节点,路径长,最长路不大于最短路之2倍

(ii) n个内部节点的红黑树高 $h \le 2\log_2(n+1)$

(iii) 根节点黑高(至叶节点,不含根节点,路径上黑节点总数) 为h的RBT,内部节点数至少有 $2^{2h}-1$ 个.

△同一组数,根据插入顺序不同可能产生不同RBT

⑤ B-树的插入、删除



④ 插入

Step 1. 插入BST

Step 2. 新节点 { 为根节点, → 黑色; 不为根节点, → 红色 } — 只需考虑"不红红"

Step 3. 检查此时树是否满足RBT

若否,进行调整: case 1 黑叔叔 — 根据叔叔节点 (AVL) { ·LL 染色; ·LR 染色; ·RL 染色; ·RR 染色 }

Case 2. 红叔叔

叔.父.爷染色.

deden节点,变为逻辑上(祖父)的"新"节点,递归step3

[Leftist Heap]

合并两个堆为O(logn)，与之相比，完全二叉堆为O(n)

① null path length (NPL(X)), of any node X is the length of the shortest path from X to a node without 2 children. Define NPL(NULL) = -1

Recursively: $NPL(X) = \min\{NPL(C)\} + 1$, for all C as children of X

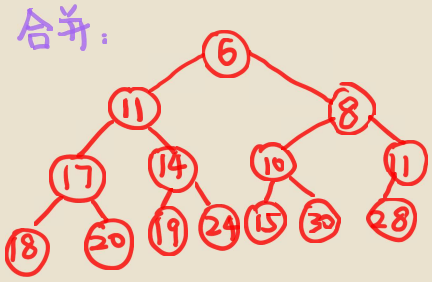
- Definition. The leftist heap is for all node X in the heap. npl(LeftChild) ≥ NPL(RightChild)

② 左式堆的构建　插入、删除(最小值) → 合并.

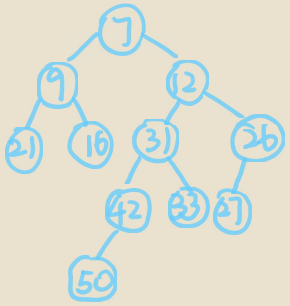
(i) Merge(H1, H2)　(递归过程) @ PPT

Step 1. Merge(H1→Right, H2) (to H2)
Step 2. Attach(H2, H1→Right) (to H1)
Step 3. Swap(H1→Right, H1→Left) if necessary

(ii) 例：合并：

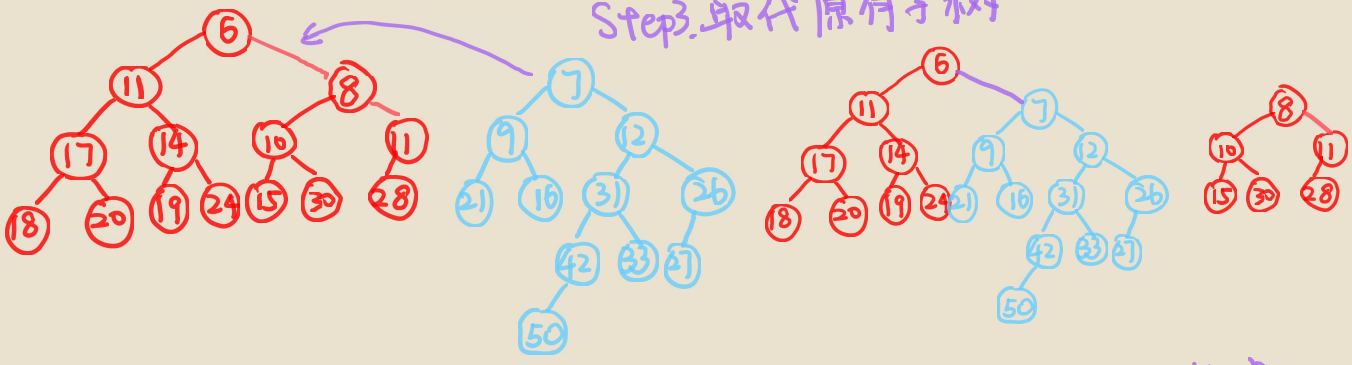


(H1)

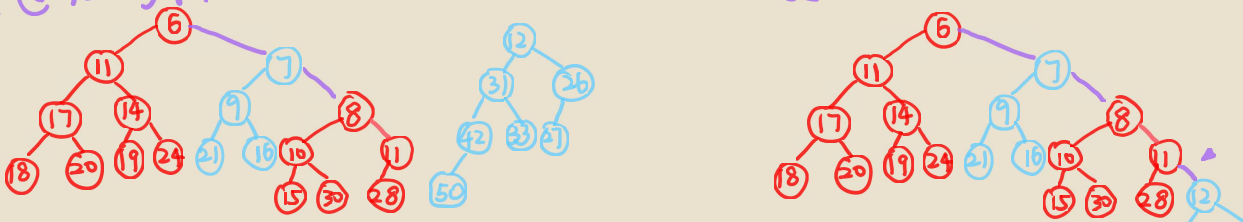


(H2)

Step 1. 将根节点键值小的作为被插入树. (H1)
Step 2. 沿着被插入树右路径寻找插入点.
Step 3. 取代原有子树



(递归1)将被取代的子树尝试插入　(递归2)将被取代的子树尝试插入



Step 4 (递归返回). 自底向上. 在需要时交换左右子树

③ 结论

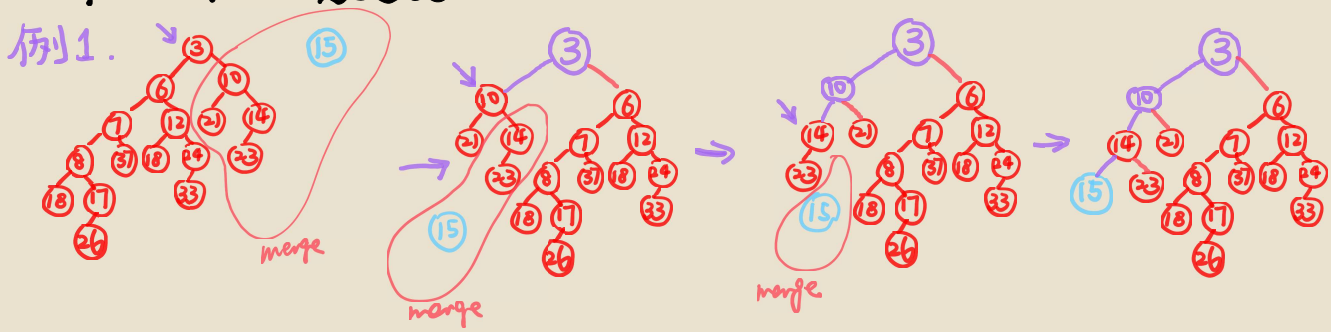
(i) A leftist heap with r nodes on the right path must have at least $2^r - 1$ nodes.

[Skew Heap] 合并的Tamortize(N) = O(log n)

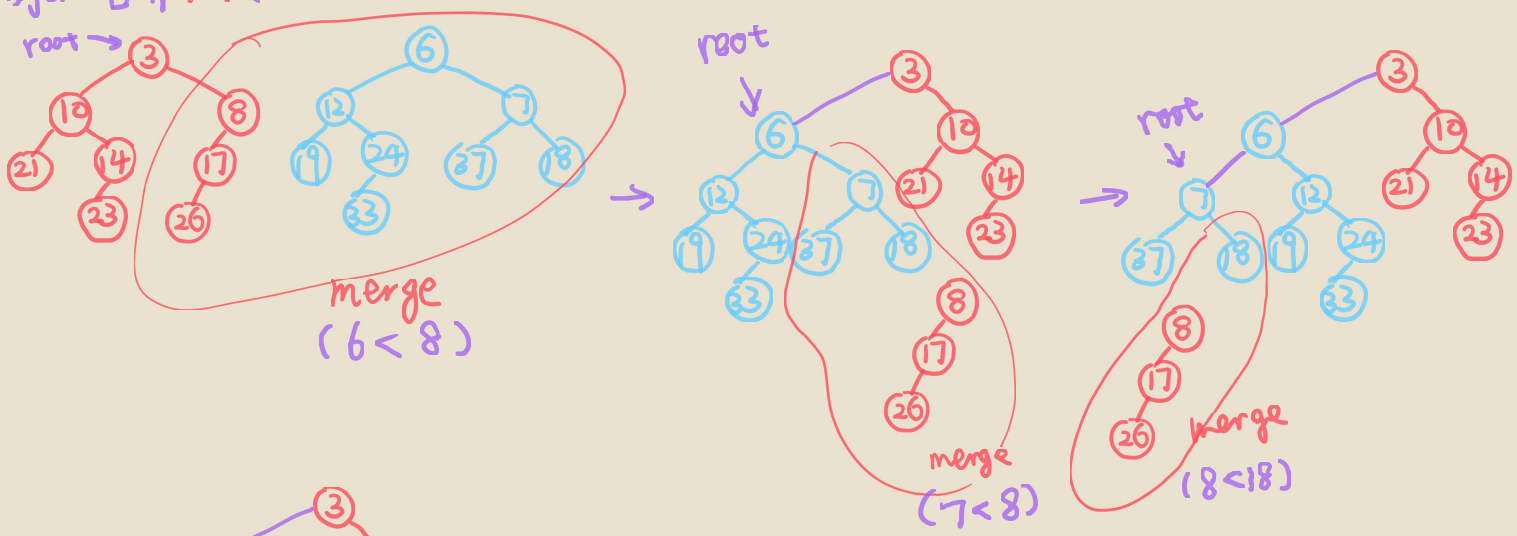
① 合并操作

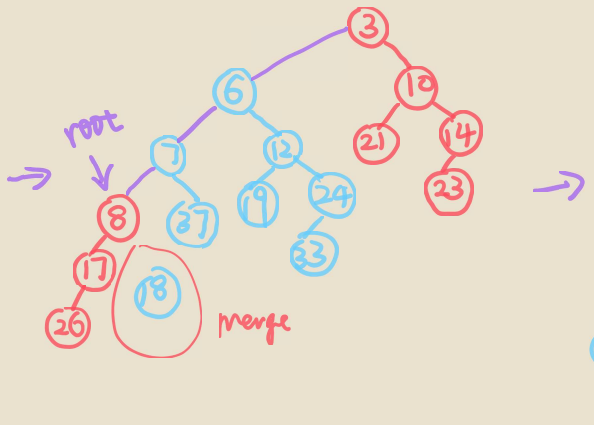
△：每次合并，交换当前根节点的左右子树

例1.



例2. 合并 H_1、H_2





Finish

△. 对于每一步.

(i) snap(H_1, root→L, H_1, root→R)

(ii) merge(原(H_1, root→R), H_2)

(iii) root = merge后形成的根(插入H_1的)

一个并入堆中

一个成为新的H_2

递归终止：merge后，不再存在两个堆

[Binomial Heap / Queue]

一个由二叉堆组成的森林.

$B_0 B_1 \cdots B_k$. $\forall B_i$. 其节点数为 2^i, 它是由两个 B_{k-1} 树成的

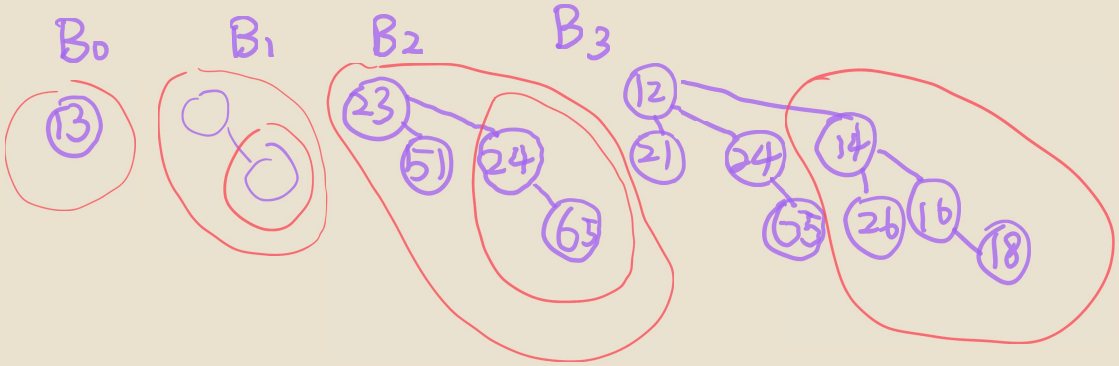
△就像二也则数一级

它的根节点有k个子节点

深度为d的节点有 C_k^d

① 用Binomial Heap表示节点数为n的堆.

例 $n = 13 = [1101]_2$



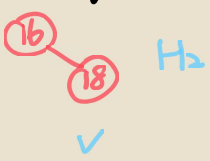
② Find Min.

遍历所有的根. $O(\log N)$

△若使用额外的域以标记 minkey. 则为 $\Theta(1)$

③ Merge, just like add 2 binary number $\Theta(\log n)$

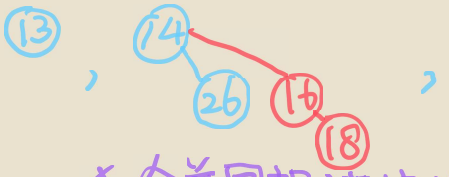
例. H_1

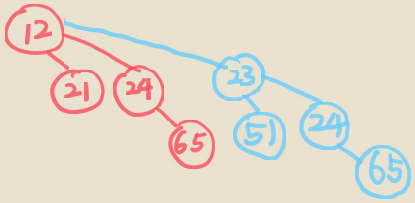


H_2

$$\begin{array}{r} 1\ 1\ 0 \\ +\ 1\ 1\ 1 \\ \hline 1\ 1\ 0\ 1 \end{array}$$

Sum →





△合并同规模的堆时,

以 root 较小的为根(主支)

△. 发生 1+1+carry 时.

可先 1+1. 不管 carry

④ insert Taverage = O(1)

⑤ Delete Min $O(\log n)$

Step 1. FindMin (设出现在 B_i 中) $O(\log n)$

Step 2. Remove B_i from H. $O(1)$ → H'

Step 3. Remove root from B_i $O(\log n)$ → H''

Step 4. Merge (H', H'')