ADS Week 1

相比完全BST，不是那么平衡的BST： BBT (Balanced Binary Tree) / AVL Tree

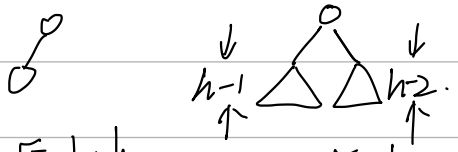
Balance Factor $= |h_L - h_R| \Rightarrow \forall u \in BBT.\ BF_u \le 1$

[Lemma] A BBT with n nodes has $O(\log n)$ height

$\Leftrightarrow$ Any BBT with height h has at least c^h nodes

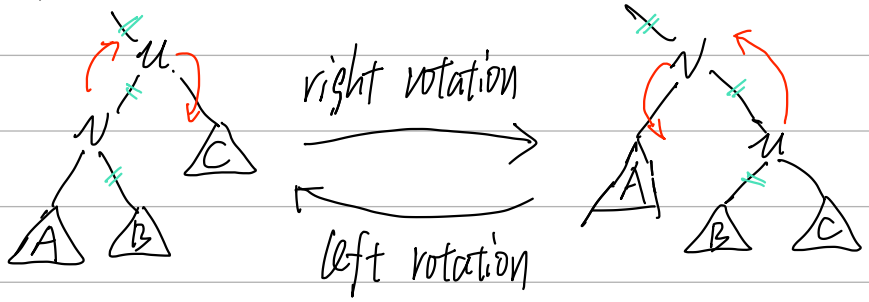
proof: 定义 $n(h)$ 为高度为 h 的 BBT 最少的结点数

则: $n(1) = 1 \quad n(2) = 2 \quad n(h) = n(h-1) + n(h-2) + 1 \quad (h \ge 3)$



$\Rightarrow n(h) \ge \left(\frac{\sqrt{5}+1}{2}\right)^h \Leftrightarrow c = \frac{\sqrt{5}+1}{2}$

恢复平衡方法：Rotation



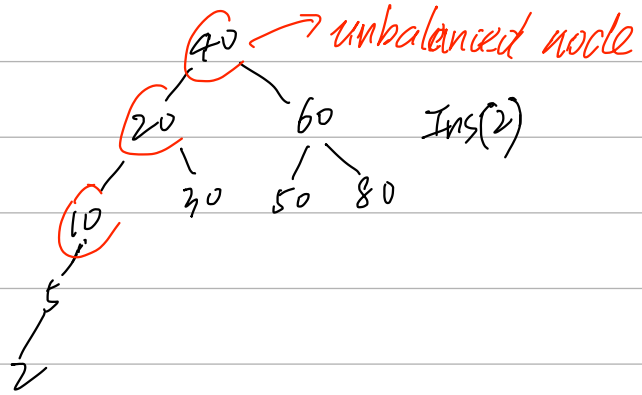
// : 被更改的指针

preserve BST property

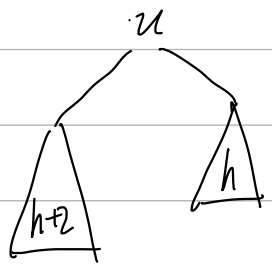
$O(1)$ time complexity

Insertion

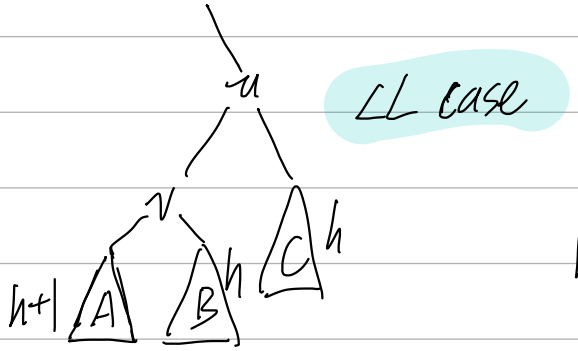
1. insert as in BST
2. restore the balance with rotation ($O(\log(n))$ nodes become unbalanced)



L case

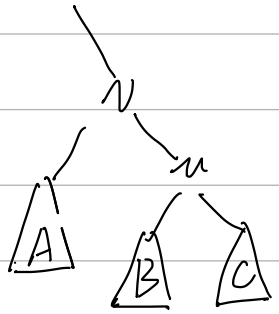


⇓ 展开右子树.

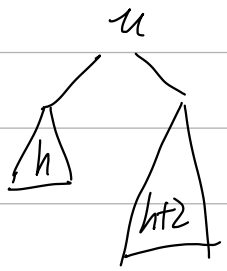


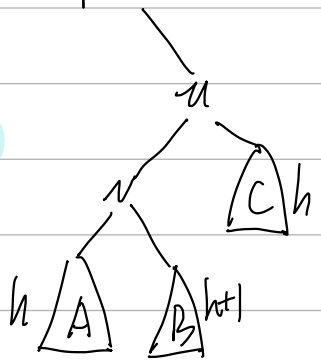
LL case

⇓ 对u右旋.



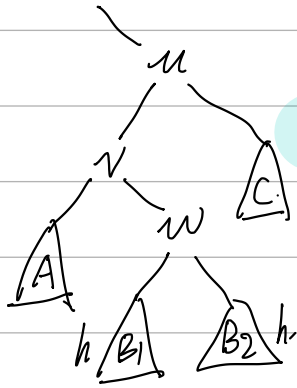
R case





LR case

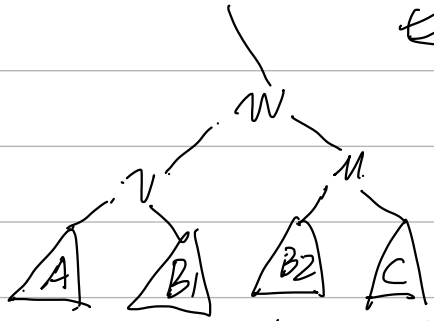
⇓ 展开B



对v左旋

对u右旋.

此处可复用之前的代码



只要恢复了最初下的节点，BST就会平衡了（将u的高度降低了）

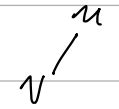
因此恢复平衡的 time complexity = O(1)

Deletion

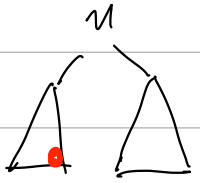
1. delete as in BST
2. restore balance.

Every deletion in BST is essentially to remove a leaf

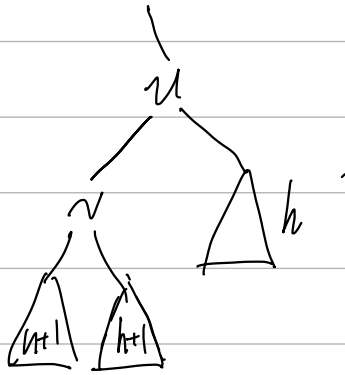
- 1. if u is a leaf
- 2. if u has only one child



- 3. if u has two child



exchange u with that red node ⇒ case 2



LR case

其他就按照插入的旋转即可.

(Q 1) how many nodes become imbalance immediately after deletion?

only 1. 因为删去一个结点不影响自己所在树的高度

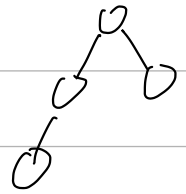
(Q. 2) how many rotations do we need to restore balance?

$\log n$ 旋转完后, 整棵subtree高度会减1, 可能导致 u 的父节点不平衡. the worst case 是不平衡一直向上传递

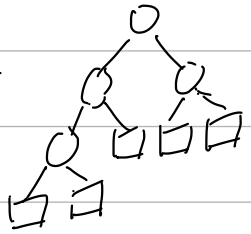
发现删除时仍需较长时间, 考虑进一步放宽"平衡"条件

原条件 $|h_L - h_R| \le 1$ 放宽为 $\frac{1}{2} \le h_L / h_R \le 2$ 从差变为倍率

红黑树 Red Black Tree

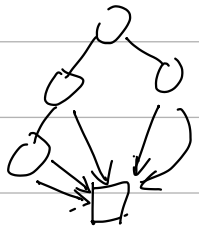


补上空节点□



□: NIL external node

○: internal node



A RBT is a BST whose extended version satisfies that

(1) every node is ● or ▨

(2) root node is ●

(3) every leaf (NIL) is ▨

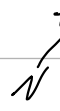
} easy to understand

(4) the children of a ▨ must be ● / ▨ (no red next to red)

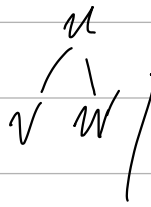
(5) for each node u, all descending paths from u to leaves have the same ● nodes

对于节点 u，定义 $bh(u)$ = 包括u在内的，路径中黑节点的数值

PPT中不包含 ⚡

(5)$\Leftrightarrow$ for every u:  $bh(v) = bh(w)$

(4)$\Rightarrow$ $bh(u) \geq \frac{1}{2}h(u)$



$$\begin{cases} bh(v) = bh(w) \\ h(v) \geq bh(v) \geq \frac{1}{2}h(v) \\ h(w) \geq bh(w) \geq \frac{1}{2}h(w) \end{cases} \Rightarrow \frac{1}{2} \leq \frac{h(v)}{h(w)} \leq 2$$

通过红黑机制实现（在之后平衡性恢复时有用）

[Lemma] A RBT (extended version) with n internal nodes has height at most $2\log(n+1)+2$

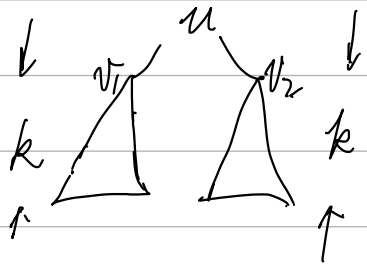
Proof: for any $u \in T$ T_u 为以 u 为根的子树

$size(T_u)$ 为 T_u 的 internal nodes 数量.

证明: $size(T_u) \geq 2^{bh(u)}-1$ (当 u 为根时, $n \geq 2^{bh(root)-1}-1 \geq 2^{\frac{h}{2}-1}-1$)

归纳法: 对 $h(T_u)=1$ $size(T_u)=0$ $bh(T_u)=1$

assume that $size(T_u) \geq 2^{bh(u)-1}-1$ $(h(T_u) \leq k)$

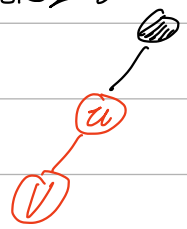


$$\begin{aligned} size(T_u) &= 1 + size(T_{v_1}) + size(T_{v_2}) \\ &\geq 1 + 2^{bh(v_1)-1} - 1 + 2^{bh(v_2)-1} - 1 \\ &= 2^{bh(v_1)-1} + 2^{bh(v_2)-1} - 1 \\ &\geq 2^{bh(u)-2} + 2^{bh(u)-2} - 1 \\ &= 2^{bh(u)-1} - 1 \end{aligned}$$

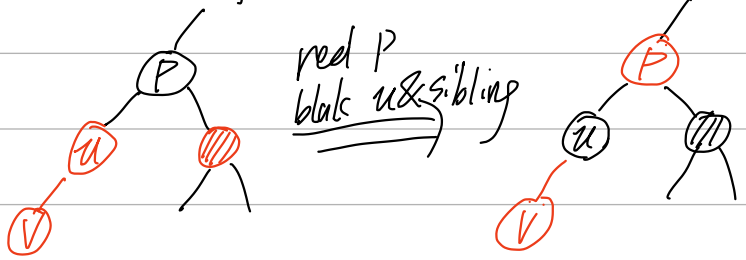
Insertion

"别忘了, 红黑树的维护就是为了"

1. insert as in BST and mark as 🔴 (may break (4))
2. rotation



Case 1: sibling of u is 🔴

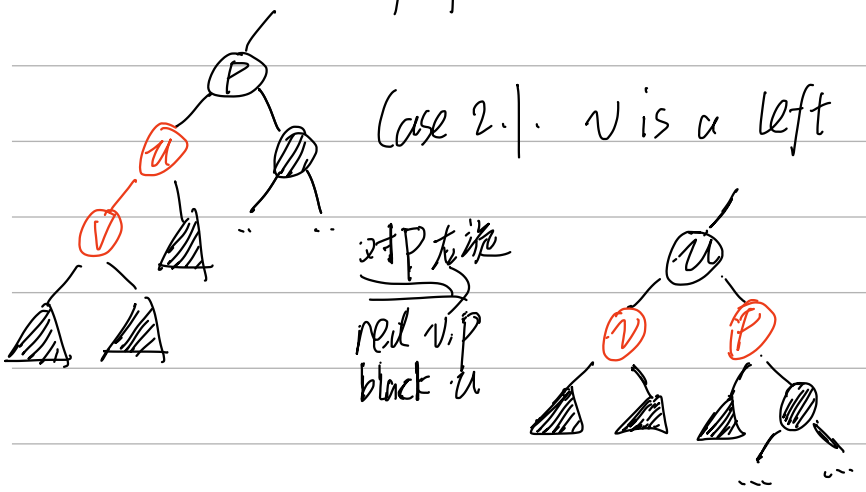


1-1: if P's parent is ⚫, than well done

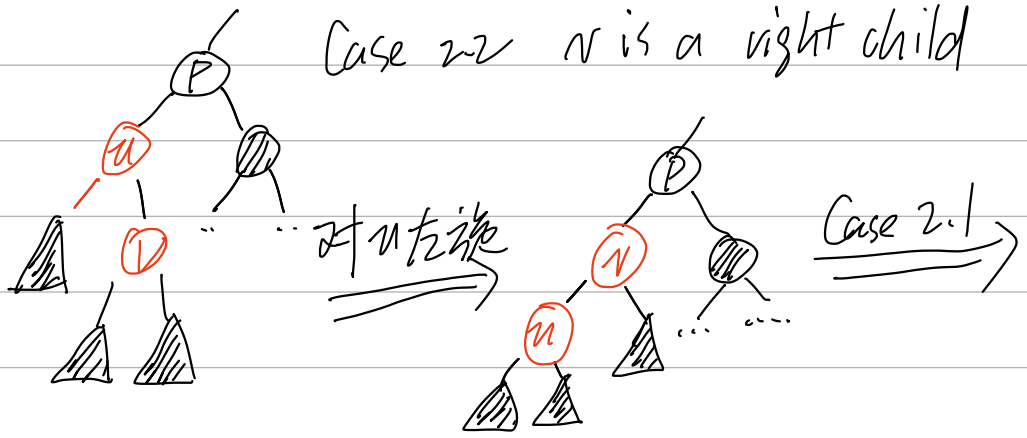
1-2: if P is root, black P

1-3: if t's parent is ● , than ……

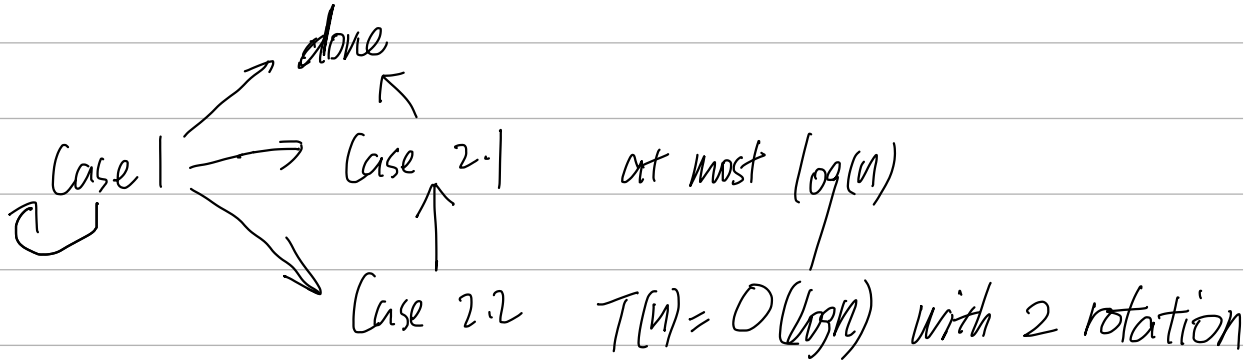
Case 2: sibling of u is ●



Case 2.1: v is a left child



Case 2.2 v is a right child

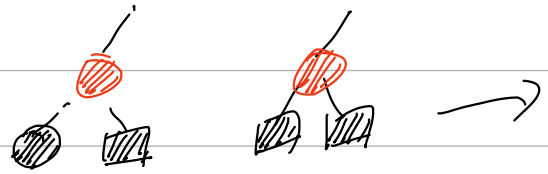


Deletion

1. Delete as in ~~BST~~

Observation: 被删去的节点最多只有一个 child (不包括 NIL)

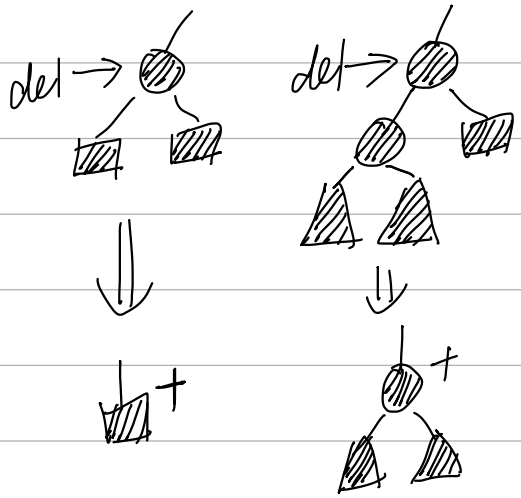
① If deleted node is 🔴 : 直接 Done



② If it is ⚫

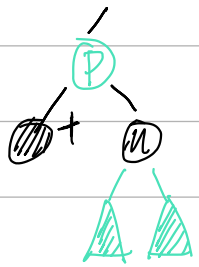
↳ if its child is 🔴, red the child, Done

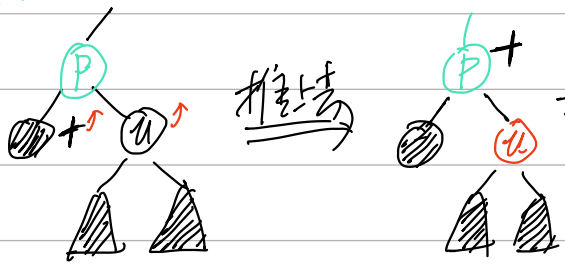
↳ if its child is ⚫.



两个黑，之后再修复处理

下面修复处理 "⚫+"

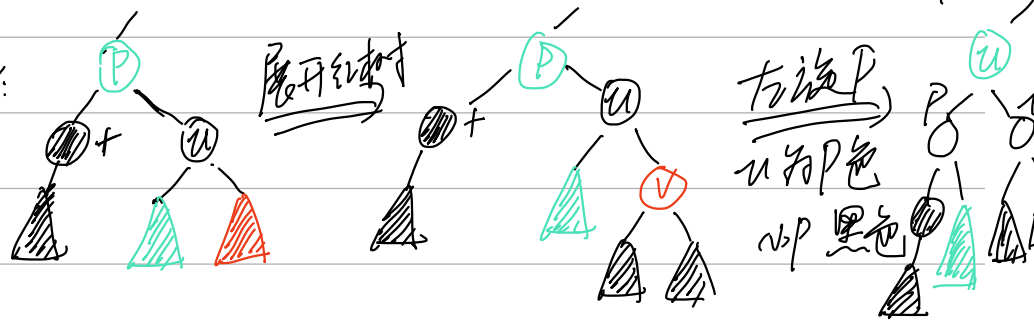
Case 1:  (u is not NIL)

Case 1.1: 

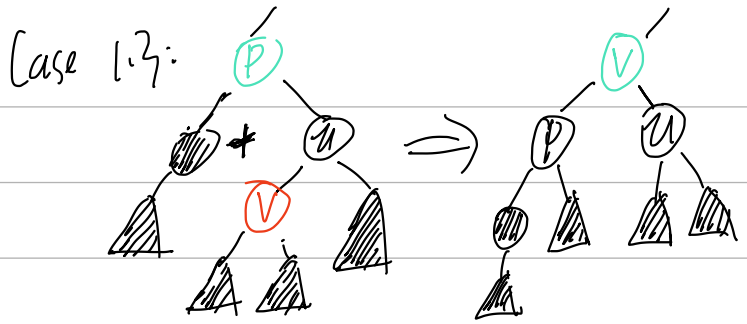
if P is 🔴, black it

if P is not, Done

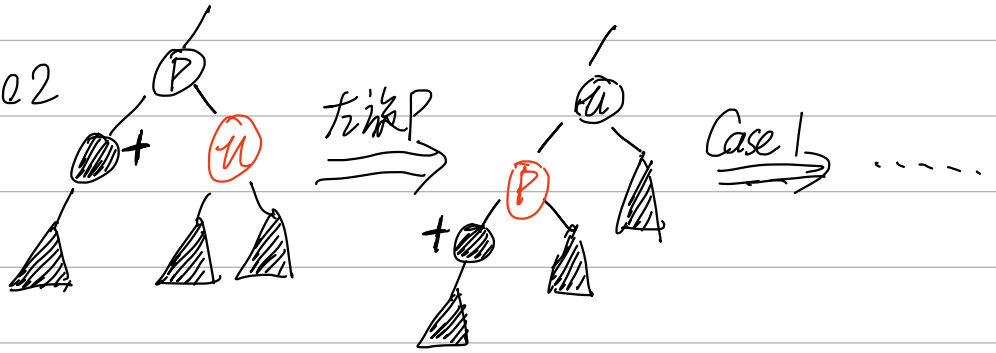
if P is ⚫, 递归

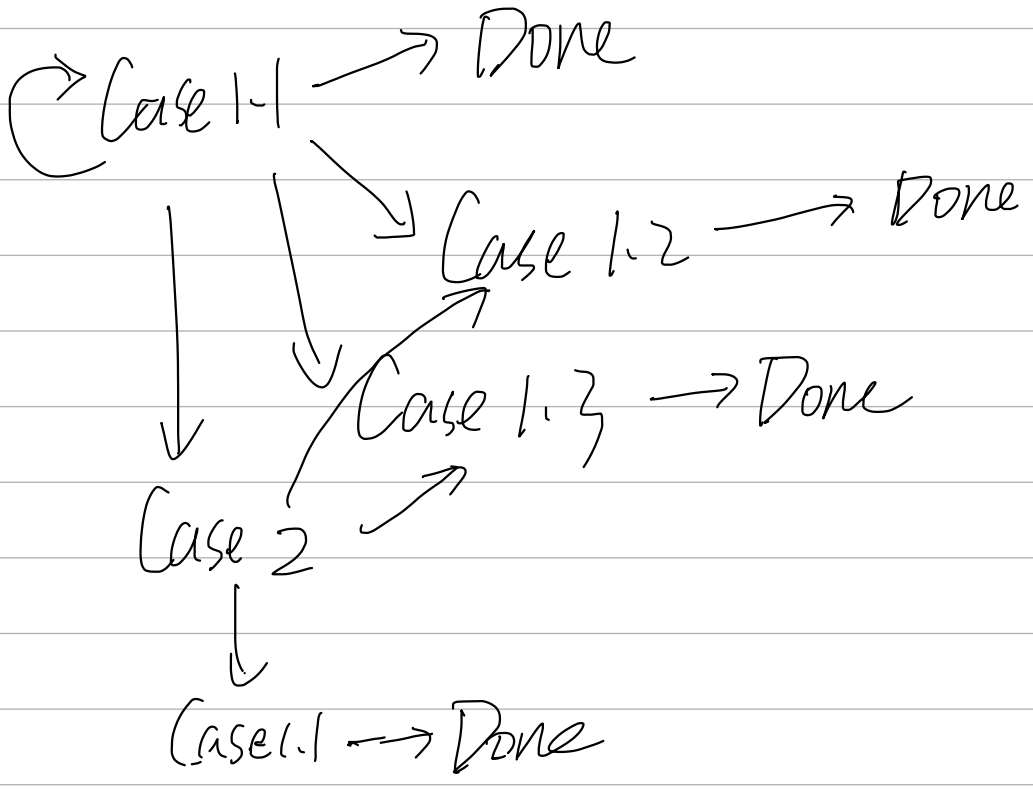
Case 1.2: 

Case 1.3:



Case 2





$T(N) = O(\log n)$ with $O(1)$ rotations

	AVL	RBT
query	$O(\log n)$	$O(\log n)$
update	$O(\log n)$	$O(\log n)$

树高红黑树高
但是红黑树删除时
只需要 $O(1)$ 次 rotation
⇒ 红黑树更快一些