* hash table of b entries after inserting n keys

if $\begin{cases} \frac{n}{b} \text{ large} & \Rightarrow \text{hash won't work well} \\ \text{hash non-uniform} & \Rightarrow \frac{n}{b_{\text{eff}}} \text{ large} \end{cases}$ ⇑⇑

* idea: increase b when $\frac{n}{b}$ large
(dynamic / extendable hashing)

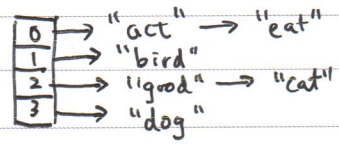
* naive approach :
① set $b^{new} = 2b$
② change $h(key)$ to $\{0, 1, \cdots, 2b-1\}$ (range)
③ rebuild new hash table : $O(n)$ if insert is $O(1)$

- cannot do often (say, whenever $\frac{n}{b} > \theta$?)
- can still cause long waits

* lazy approach :
① set $b^{new} = 2b$
② change $h(key)$
③ rebuild only the overflow entry : $O(b) + O(\frac{n}{b})$

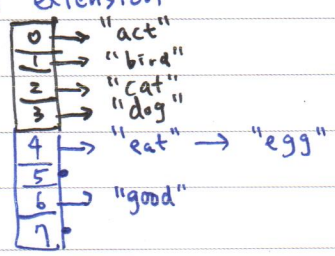
e.g. hashing w/ chaining of length 2

$h(key) = (key[0] - 'a') \% b$

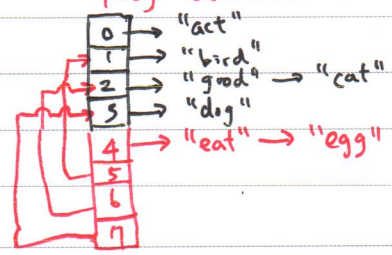


insert "egg"

naive extension



(directory)
lazy extension :



hash in bin form

000
001
010
011
100
101
110
111

change b ≡ use one more bit

* usage of directory-based dynamic hashing
  - "chain" is a file of fixed size
  - "get" needs only 1 read disk access
  - "insert" needs at most 2 write disk access (and 1 read)
  - "delete" needs only 1 write disk access

* binary search trees

restriction	loose				strict
worst search time	$O(n)$	$>$	$O(h)$	$>$	$O(\log n)$
maintanence time after insertion	$O(1)$	$<$	$O(1)$	$<$	$O(n)$
worst height	$O(n)$	$>$	$O(\log n)$	$>$	$O(\log n)$
	common BST		RB	AVL	complete BST

* AVL Tree (1962)
Adelson Velskii Landis

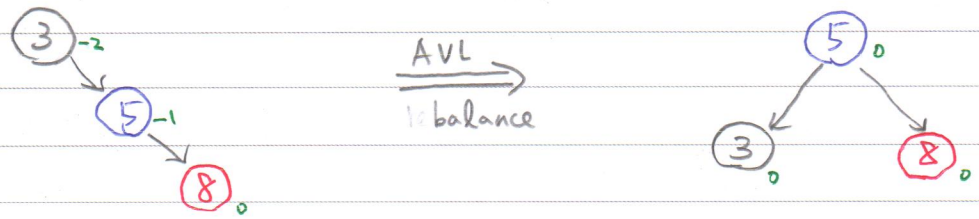
a binary search tree such that

$$|height(T_L) - height(T_R)| \leq 1$$

(h_L, h_R)

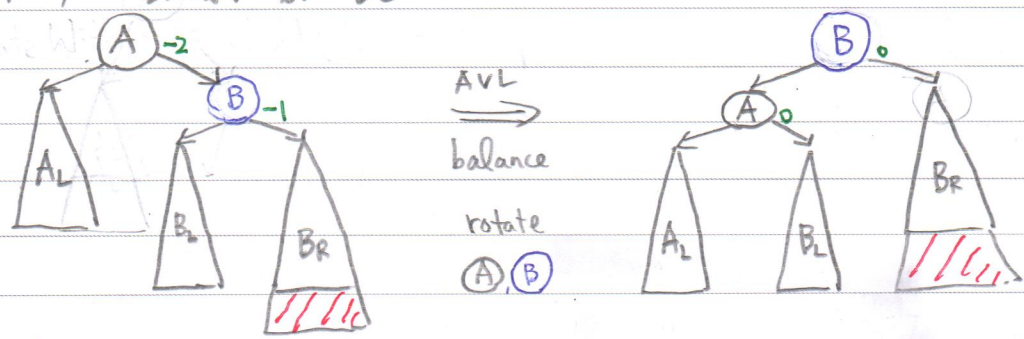
$$BF(T): \quad h_L - h_R = \begin{cases} 1 \\ 0 \\ -1 \end{cases}$$

* insert (3) (5) (8) to AVL



operation : rotate (3), (5)

* case RR, similar for LL



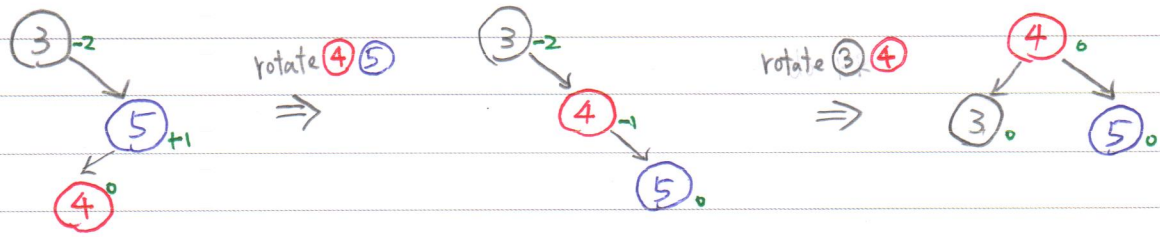
(height ↑)

(height same)
no more rotation needed

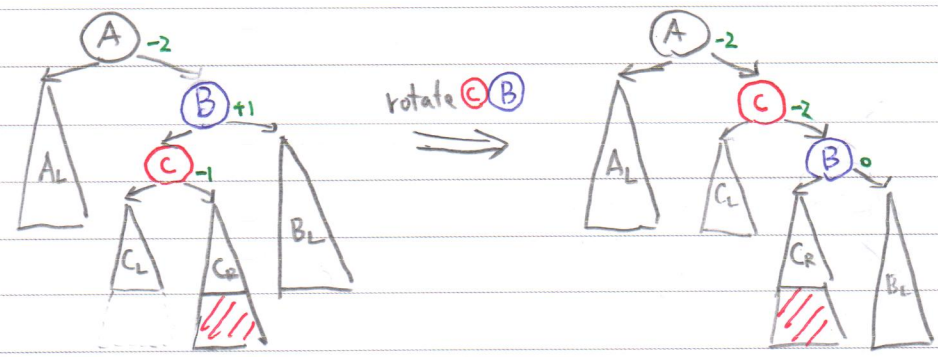
* insert 1, 2, 3, 4, 5, 6, 7

* void rotateWithRightChild (Node* parent) {
    Node* child = parent → right;
    parent → right = child → left;
    child → left = parent;
  }

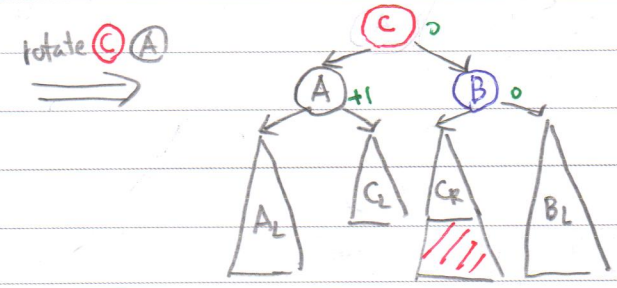
* insert (3) (5) (4) to AVL



* case RL, similar for LR



(height ↑)



(height same)

* insert 15, 14, 13, 12, 11, 10, 9, 8