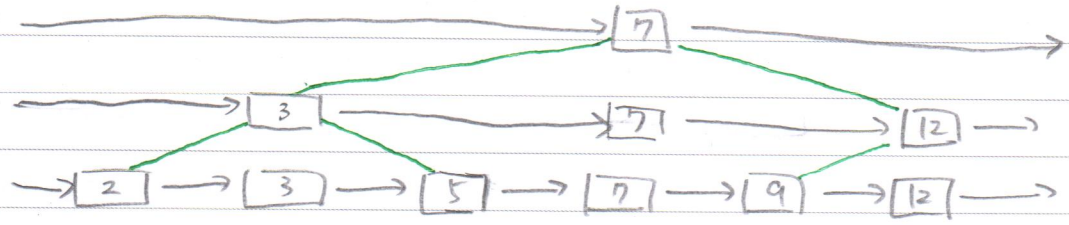
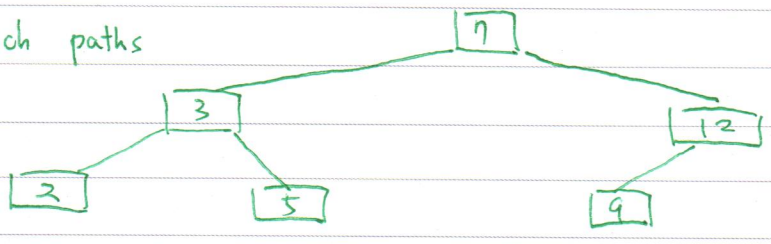
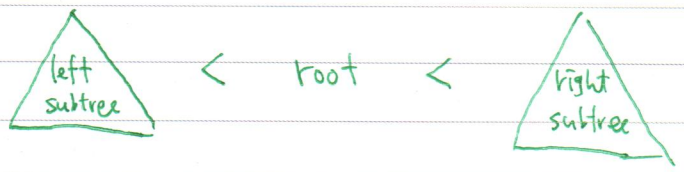
* skip list to BST



the search paths





for every subtree : called binary search tree

* search for key on BST

```
BST-Search (k, T) {
    mid = root of T
    if (k < mid)
        return BST-Search (k, T.left);
    if (k > mid)
        return BST-Search (k, T.right);
    else
        return mid.value;
}
```

worst case : O(h) w/ h being height

* insert : similar to search

* remove :

leaf : simple

one child : simple

two children : find right-most decendent of left-subtree O(h)

Double A

* binary search trees

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

* AVL tree (1962)
Adelson Velskii Landis

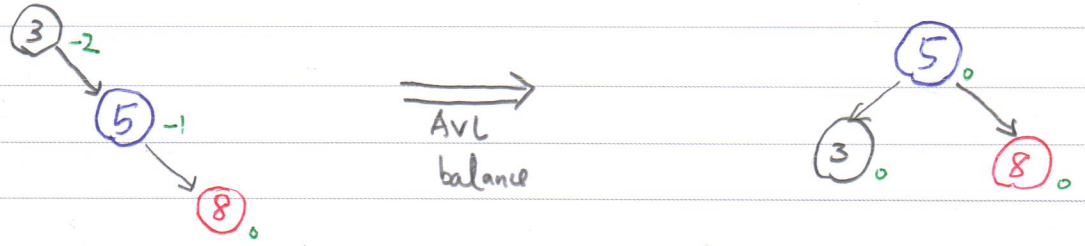
a BST such that

$$\left| \underbrace{\text{height}(T_L)}_{h_L} - \underbrace{\text{height}(T_R)}_{h_R} \right| \leq 1 \quad \text{for every subtree}$$

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

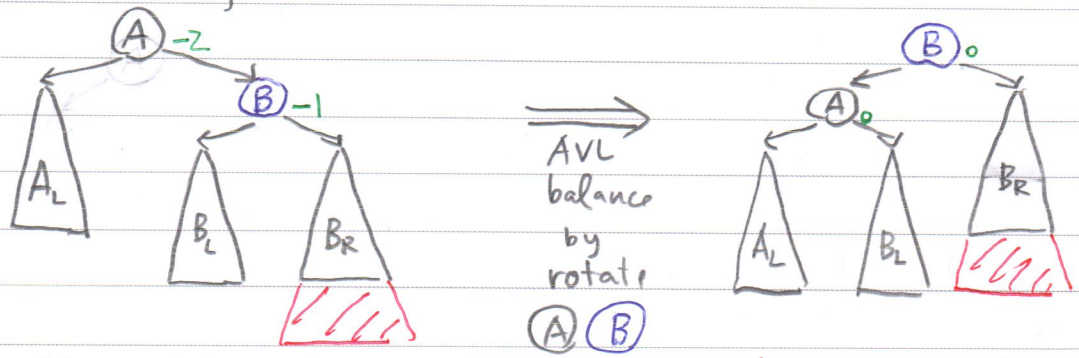
balance factor

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



operation : rotate (3) (5)

* case RR during insertion



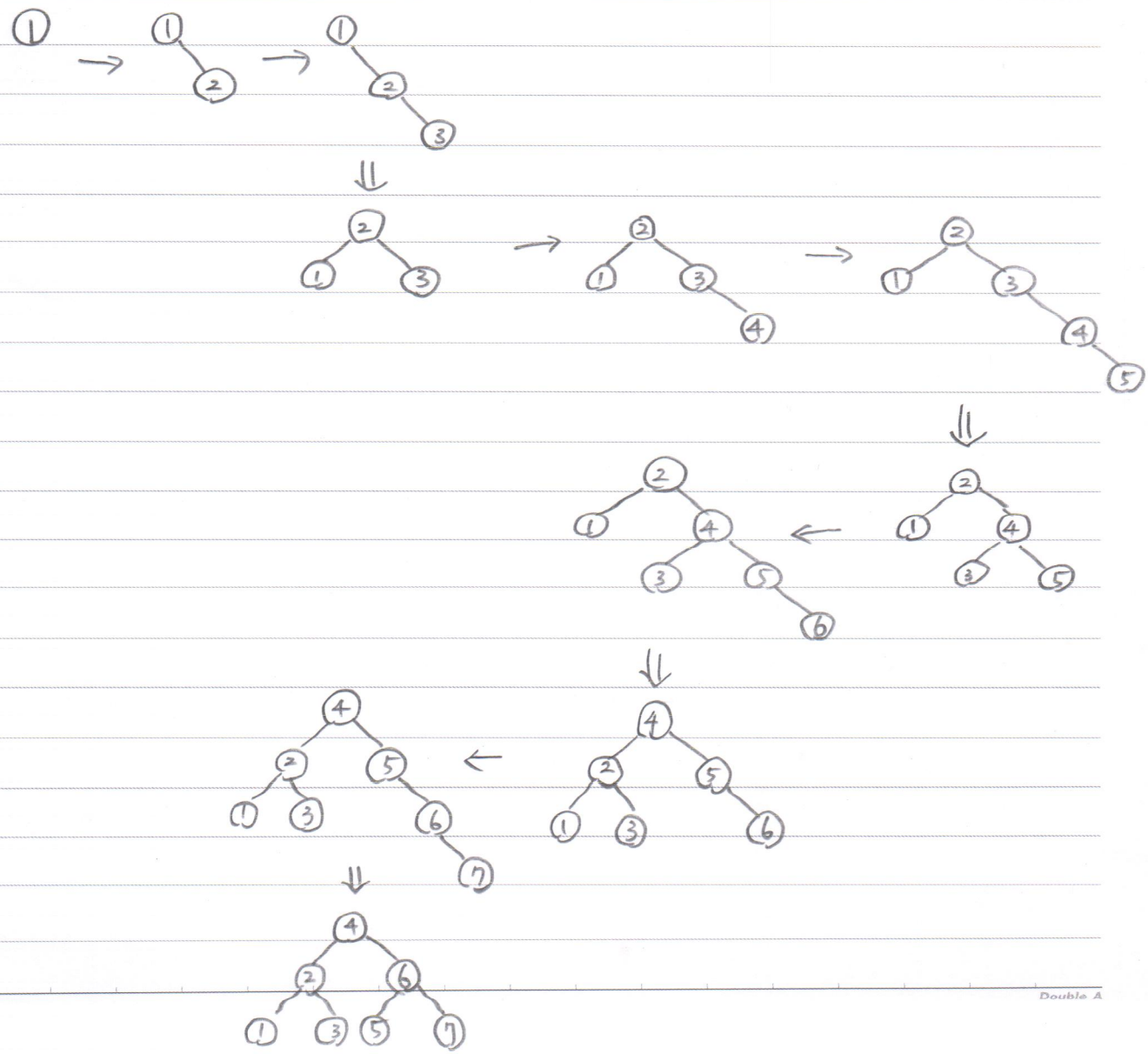
(height ↑)

(height same)

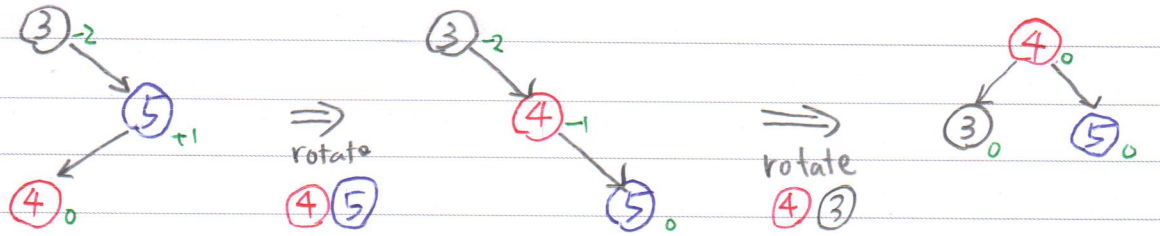
no more rotations

similar for LL

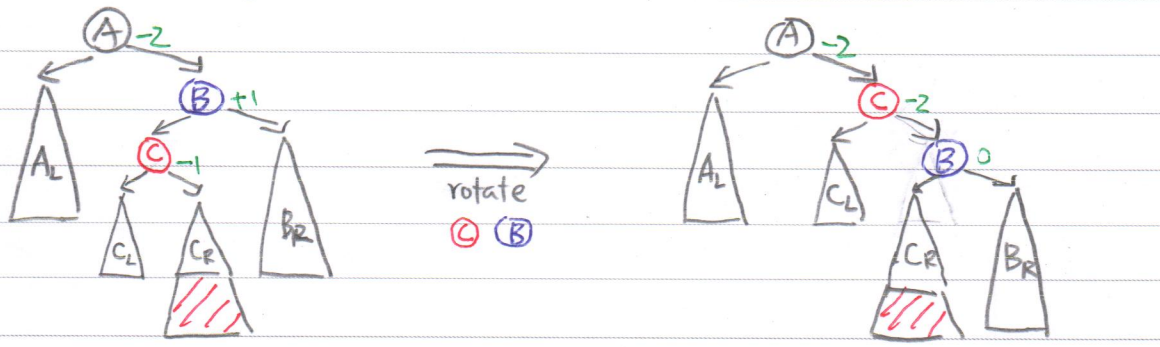
* insert 1, 2, 3, 4, 5, 6, 7

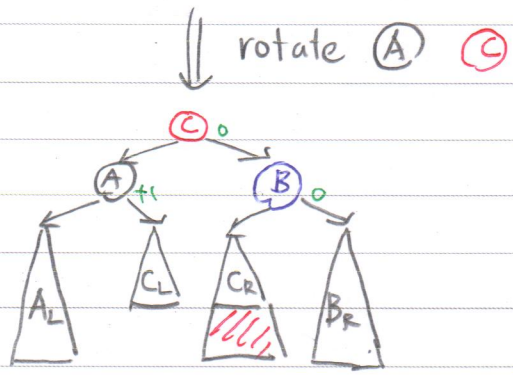


* insert (3) (5) (4) to AVL

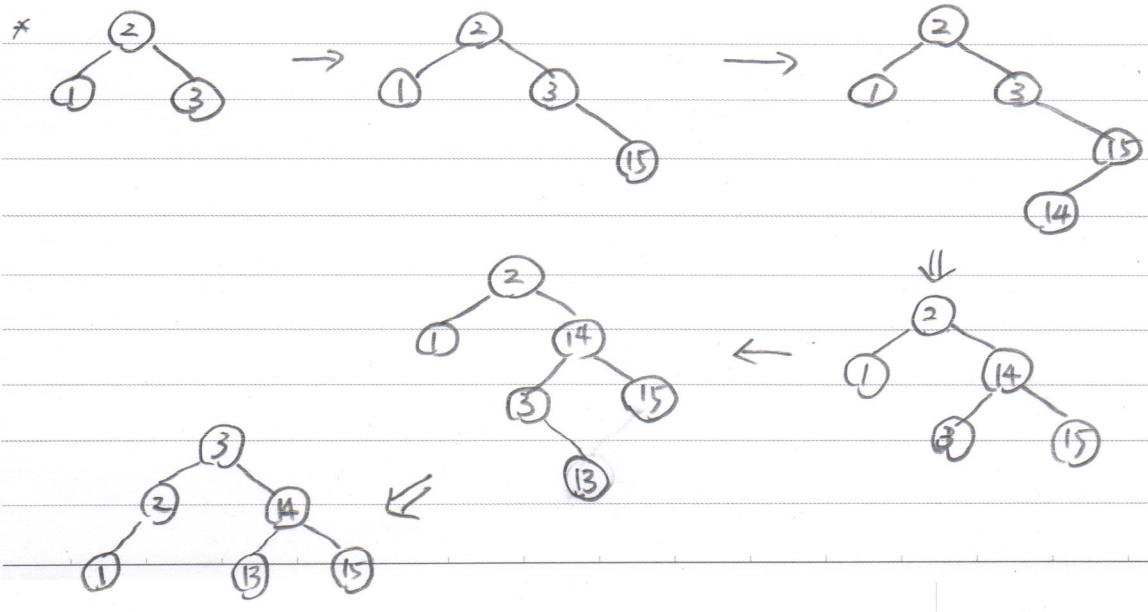


* case RL during insertion





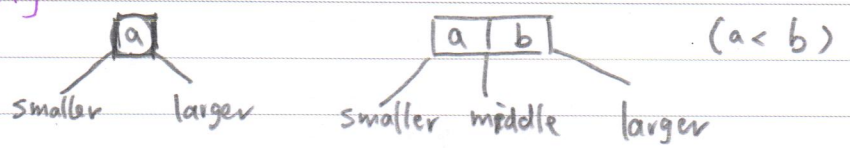
similar for LR



* remove in AVL tree
  rotate (similarly), and go up

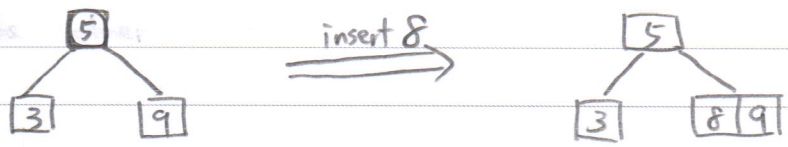
* AVL : strict & rebalance "almost immediately"
  after adding a new node
  what if we can temporarily
  add a new element without increasing height

* (2,3) tree
  ① each node can contain
  flexibility 1 or 2 elements



(a < b)

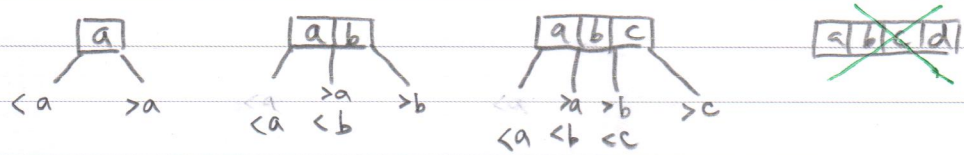
i.e. 2 or 3 children

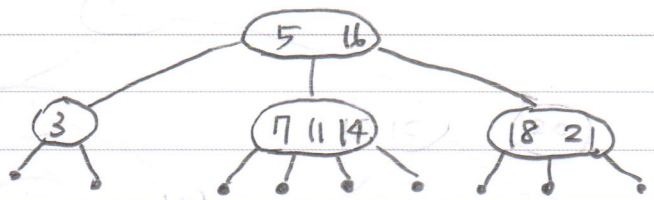


no need to increase height immediately

② all leaf nodes are of the same height
(balanced)

* (2,3,4) tree ((2,4) tree)





- insert 4 : (3,4)
- insert 19 : (18,19,21) or insert 25 (18,21,25)
- insert 15 : (7,11,14,15) overflow

split ⇓ ← insert to upper level (5,14,16)



Double A