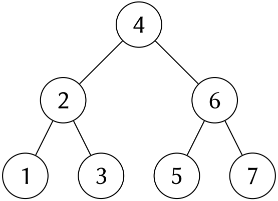
Algorithm Designs and Data Structure

AVL Trees

Binary Search Tree (BST)

- cost of BST operations depends on tree depth d
- binary tree with n nodes:
    - $\min d = \lfloor \log_2 n \rfloor$



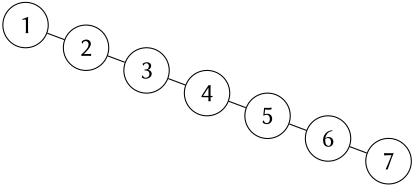
- best running time of BST operations is $\mathcal{O}(\log n)$

Unbalanced Binary Search Tree

- what happens when inserting sorted elements?
  i.e., 1,2,3,4,5,6,7

Unbalanced Binary Search Tree

- what happens when inserting sorted elements?
  i.e., 1,2,3,4,5,6,7



  - tree becomes unbalanced
  - worst running time of BST operations is $\mathcal{O}(n)$

Self-Balancing Binary Search Trees

- 2-3 trees
- **AVL trees**
- red-black trees
- splay trees
- etc.

AVL Trees

- named after 2 Russian mathematicians:
  - Georgii **A**delson-**V**elsky
  - Evgenii Mikhailovich **L**andis
- height-balanced BST
- balance factor of a node calculated as:

  | height (left subtree) - height (right subtree) |

Properties

Binary tree

all nodes must have between 0 and 2 children

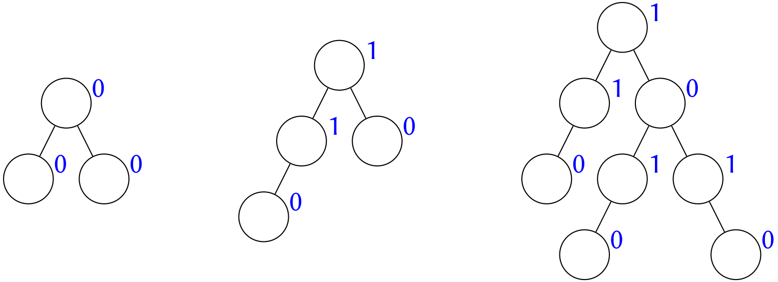
Binary search tree

key (left subtree) $\leq$ key (root) $\leq$ key (right subtree)

Height balanced

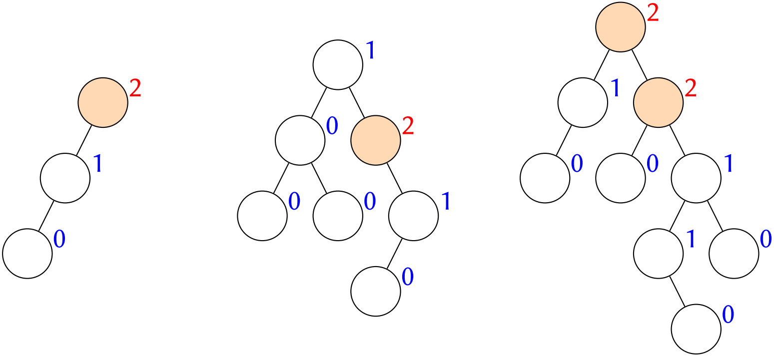
$|$ height (left subtree) - height (right subtree) $| \leq 1$

Examples



numbers denote balance factor at each node

Examples: not valid AVL trees



numbers denote balance factor at each node

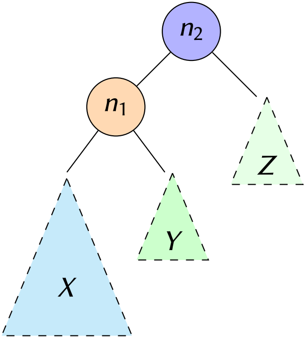
Operations

- search:
  - same as BST search
- insert:
  - similar to BST insert
  - also check balance factor and may need to restructure
- delete:
  - similar to BST delete
  - also check balance factor and may need to restructure

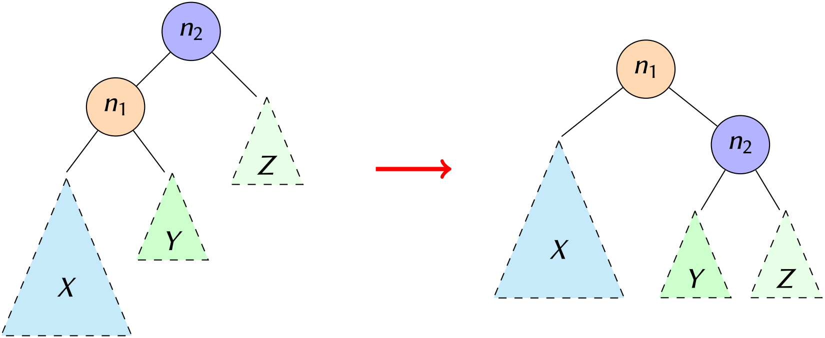
Restructuring

- single rotation
  - right rotation
  - left rotation
- double rotation
  - left-right rotation
  - right-left rotation

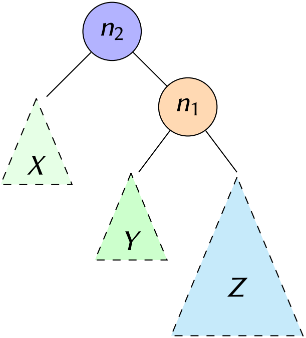
Right Rotation



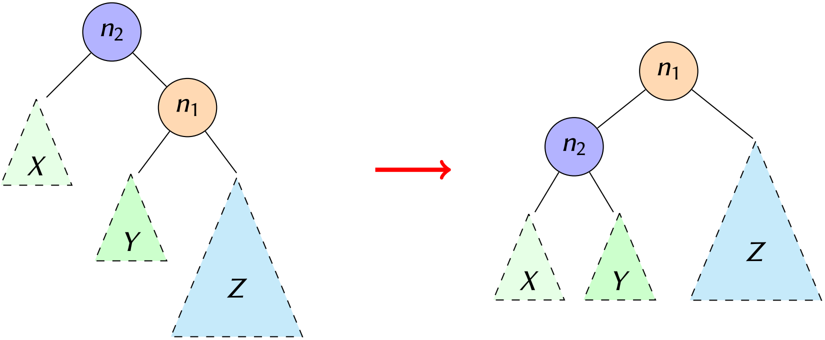
Right Rotation



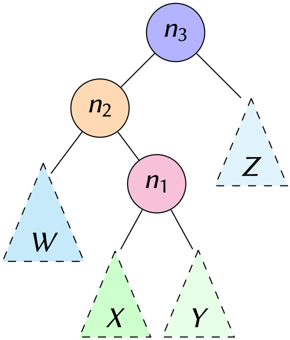
Left Rotation



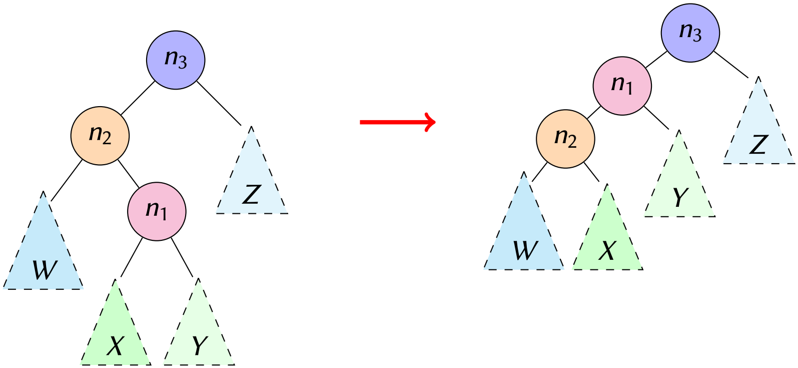
Left Rotation



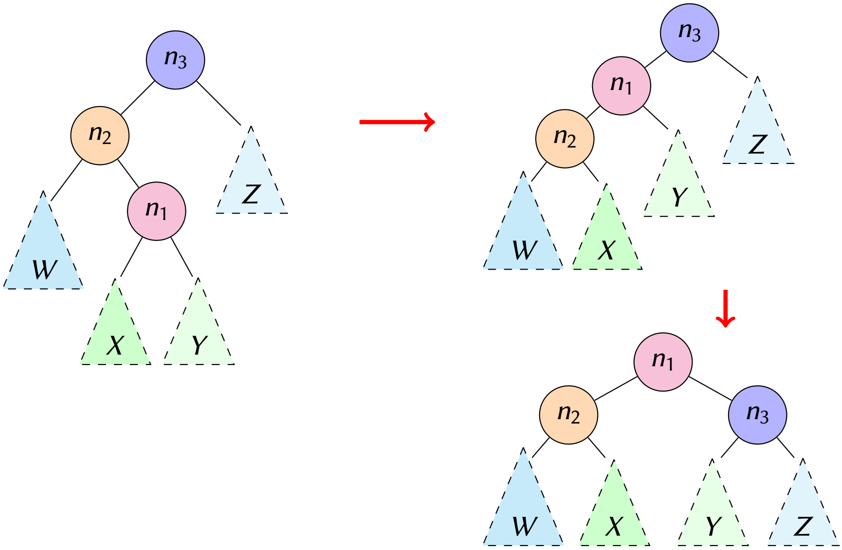
Left-Right Rotation



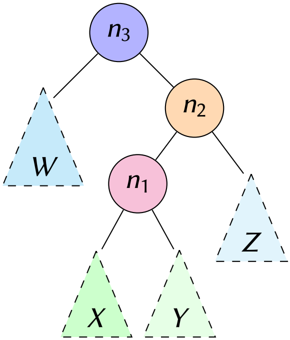
Left-Right Rotation



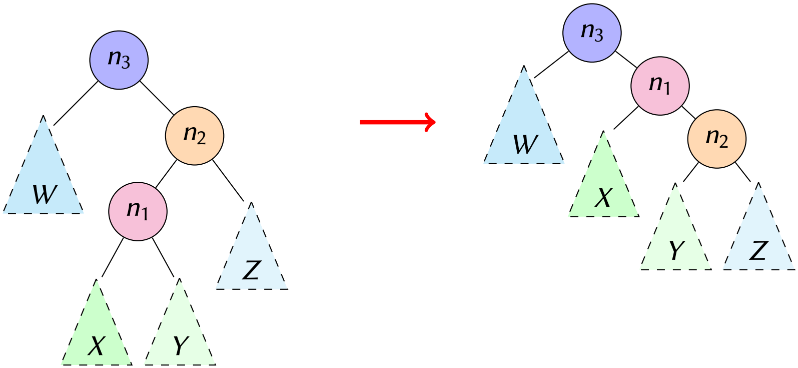
Left-Right Rotation



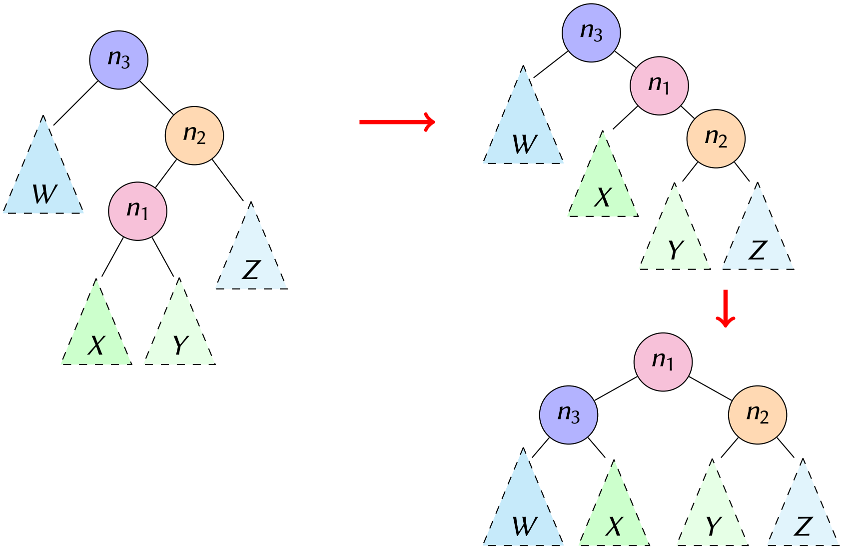
Right-Left Rotation



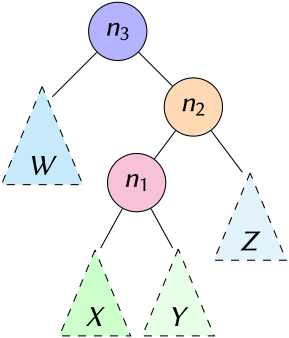
Right-Left Rotation



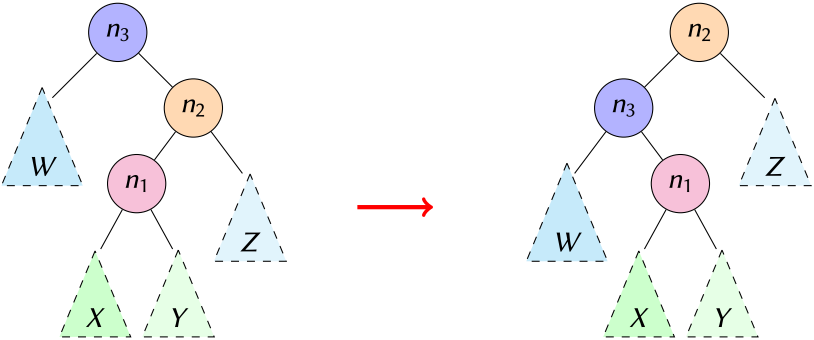
Right-Left Rotation



Why Double Rotation?



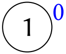
Why Double Rotation?



still not AVL tree

Example: inserting numbers from 1 to 7

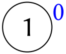
insert 1



no restructuring needed

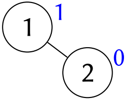
Example: inserting numbers from 1 to 7

insert 1



no restructuring needed

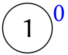
insert 2



no restructuring needed

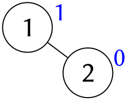
Example: inserting numbers from 1 to 7

insert 1



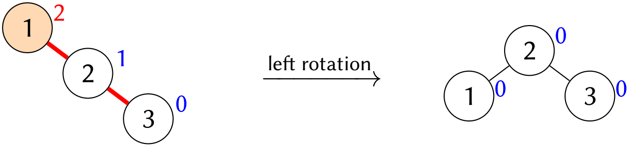
no restructuring needed

insert 2



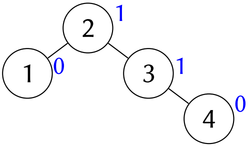
no restructuring needed

insert 3



Example: inserting numbers from 1 to 7

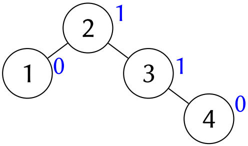
insert 4



no restructuring needed

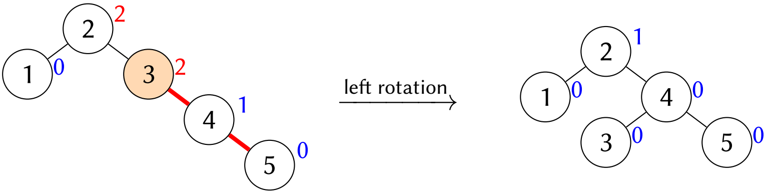
Example: inserting numbers from 1 to 7

insert 4



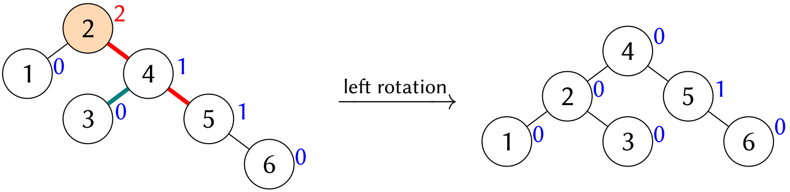
no restructuring needed

insert 5



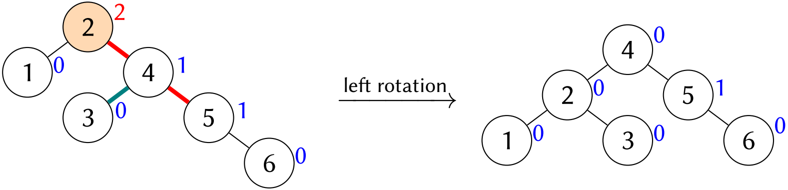
Example: inserting numbers from 1 to 7

insert 6

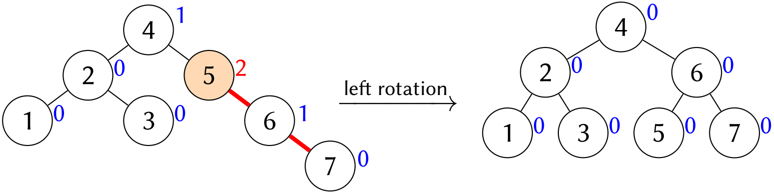


Example: inserting numbers from 1 to 7

insert 6



insert 7



Performance

AVL tree with n nodes

- data structure uses $\mathcal{O}(n)$ space
- restructuring takes $\mathcal{O}(1)$ time (using linked structure)
- searching takes $\mathcal{O}(\log n)$ time, no restructuring needed
- insertion takes $\mathcal{O}(\log n)$ time, may involve restructuring
- deletion takes $\mathcal{O}(\log n)$ time, may involve restructuring

Comparison

Data structure		Hash table	Self-balancing BST	Unbalanced BST
Search	average	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
	worst	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
Insert	average	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
	worst	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
Delete	average	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
	worst	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
Ordered		No	Yes	Yes