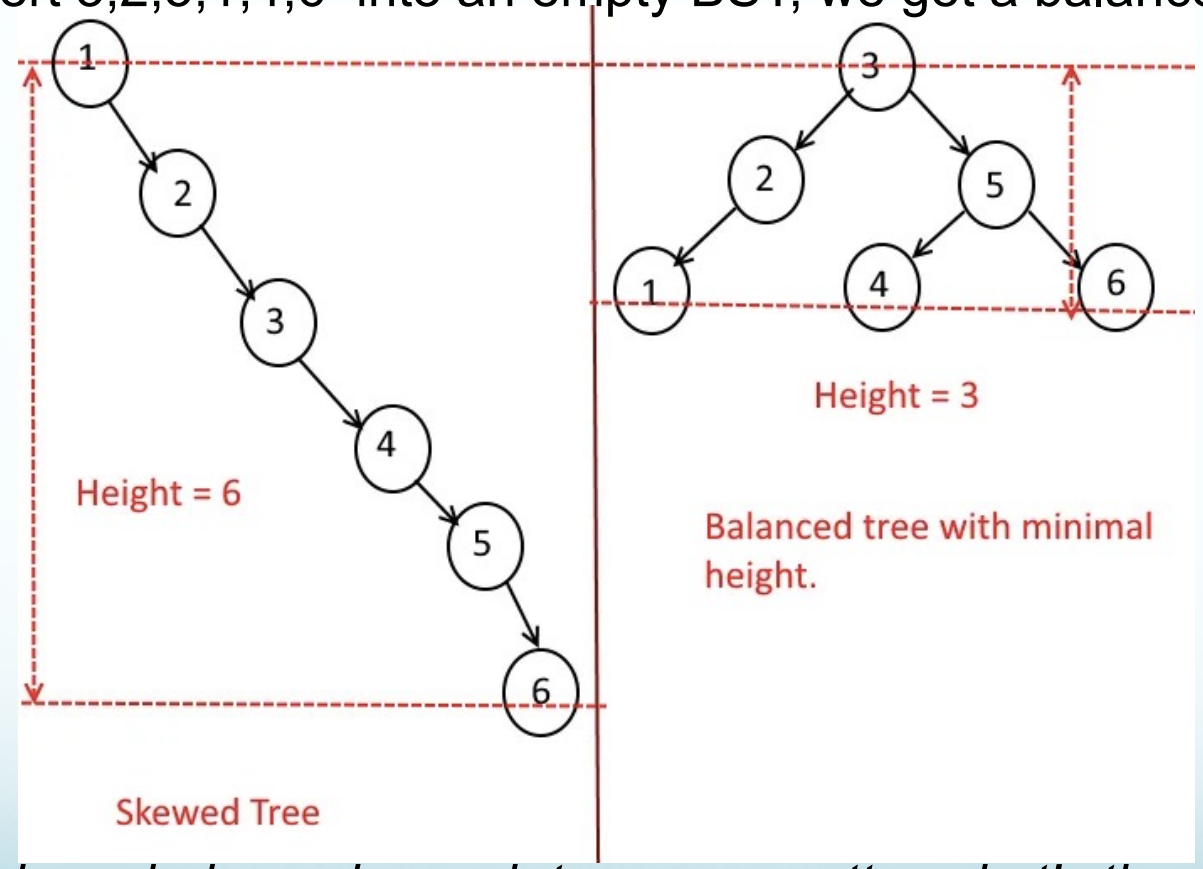
COMP20007 Workshop Week 9

	Preparation: - *have draft papers and pen ready* - open `wokshop9.pdf` (from `LMS`), and
1	Why BST, AVL, 2-3 Tree?
2	BST: Rotation, Balance factor: Problems 1, 2
3	AVL Tree: Concepts, Insertion (Problem 3)
4	2-3 Tree: Concepts, Insertion (Problem 4)
5	B-tree?
LAB	BST: insertion & level traversal (optional)
	2-3-4 Tree: insertion & level traversal (optional)
	Questions for previous week materials

BST: skewed, unbalanced, balanced. Why AVL/2-3 Trees?

The efficiency of searching in a BST depends on how balance the tree is.

- If we insert 1,2,3,4,5,6 into an empty BST, we get a skewed tree.
- If we insert 3,2,5,1,4,6 into an empty BST, we get a balanced tree.



We want to have balanced search trees, no matter what's the data input order

AVL Tree, 2-3 Tree

- AVL=? – just a balanced BST
- How: re-balance BST when it becomes unbalanced

- As opposed to 2-3 Trees:
  - 2-3 Tree is a search tree, but not a binary tree
  - 2-3 Tree is self-balancing tree [no re-balancing needed]

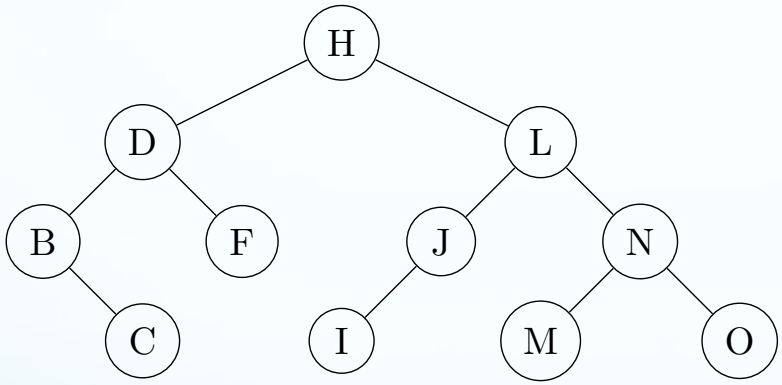
Using Rotations to rebalance AVL

- *At the start:* an empty BST is an AVL
- *Problem:* After a insertion/deletion, the resulted tree might become unbalanced
- *Approach*: use Rotations to rebalance.
- It's important first to determine:

  Which node to be rotated? How to Rotate?

AVL: How to know if a BST is balanced? use *balance factor*

Problem 2: A node's '*balance factor*' is defined as the height of its right subtree minus the height of its left subtree. Calculate the balance factor of each node in the following binary search tree.



Note on balance factor (BF) definition

- Popular (as in lectures):
  BF= height(T.left) - height(T.right)
- Option 2:
  height(T.right) - height(T.left)
- Option 3:
  absolute value of the height difference

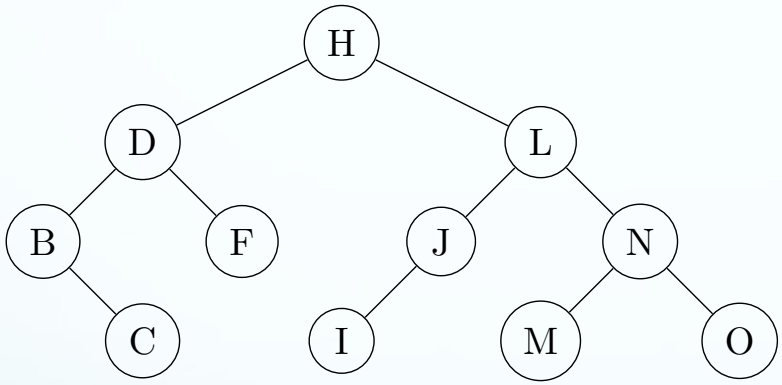
Any of them is OK, but needs **consistency**!

Balanced tree = when the balance factor of each node is 0, -1, or +1

= for each node, the difference of subtree heights is at most 1

AVL: How to re-balance an un-balanced BST? use *rotation*

Problem 2: A node's '*balance factor*' is defined as the height of its right subtree minus the height of its left subtree. Calculate the balance factor of each node in the following binary search tree.



Note on balance factor (BF) definition

- Popular (as in lectures):
  BF= height(T.left) - height(T.right)
- Option 2:
  height(T.right) - height(T.left)
- Option 3:
  absolute value of the height difference

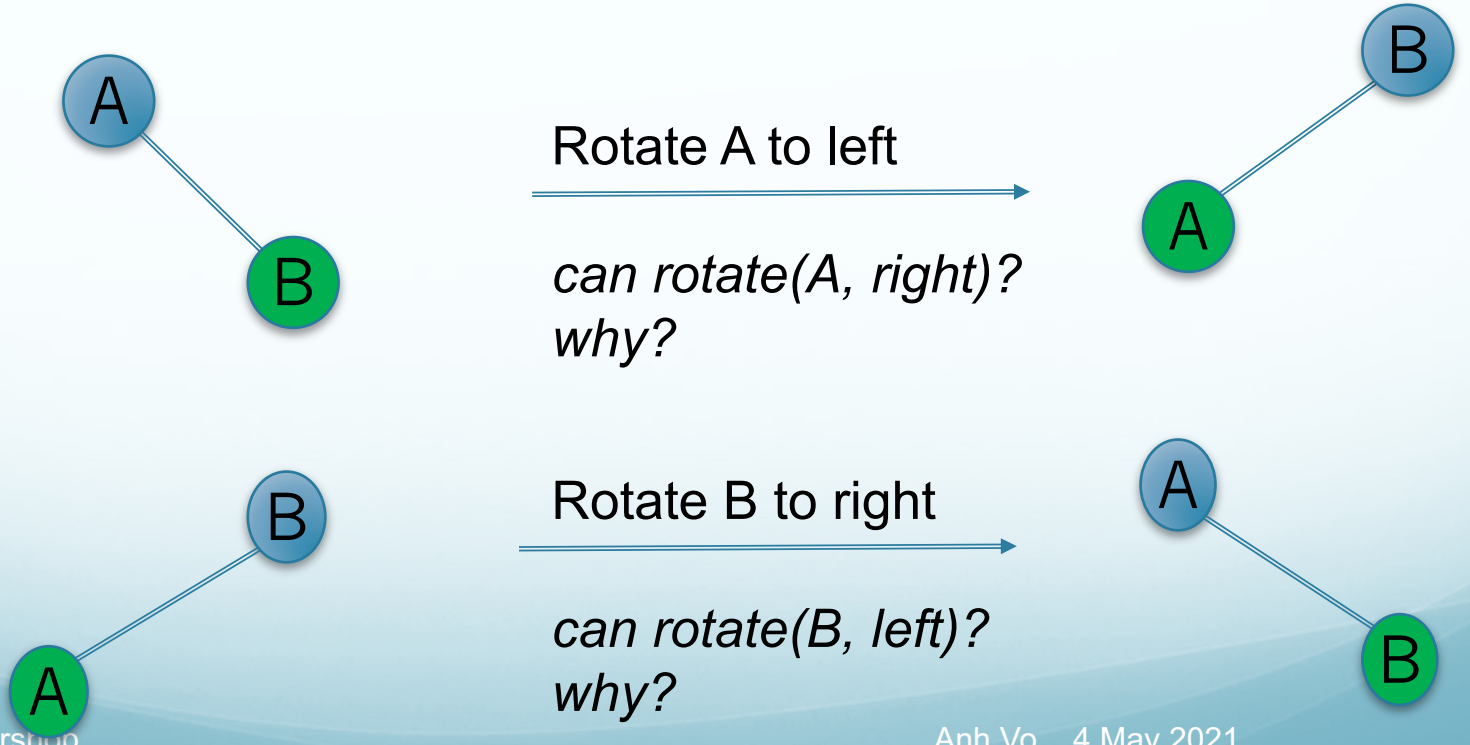
Any of them is OK, but needs **consistency**!

Balanced tree = when the balance factor of each node is 0, -1, or +1

= for each node, the difference of subtree heights is at most 1

BST: what's rotation

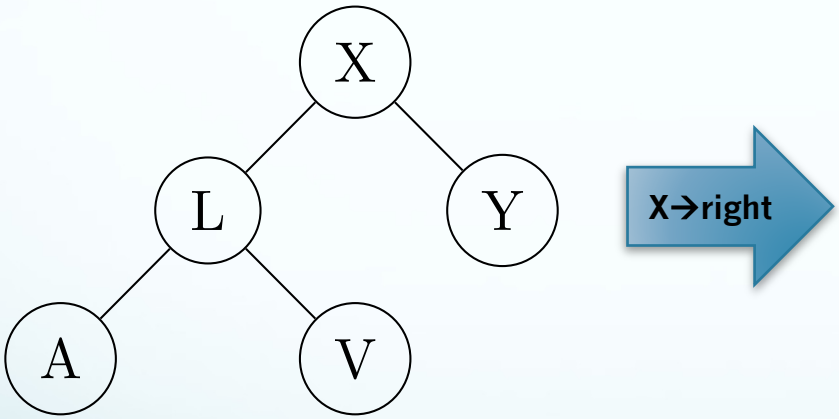
- For each simple tree A rotation reverses the parent-child relationship of 2 connected nodes
- left rotation: rotate parent to the left (to become the left child)
- right rotation: rotate parent to the right (to become the right child)



Problem 1 [class]: Rotation

In the following binary trees, rotate the ‘X’ node to the right (that is, rotate it and its left child). Do these rotations improve the overall balance of the tree?

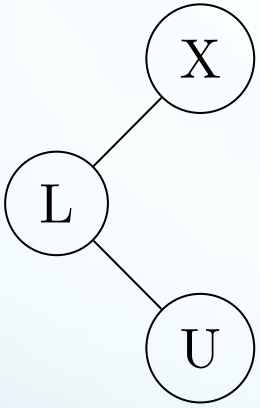
(a)



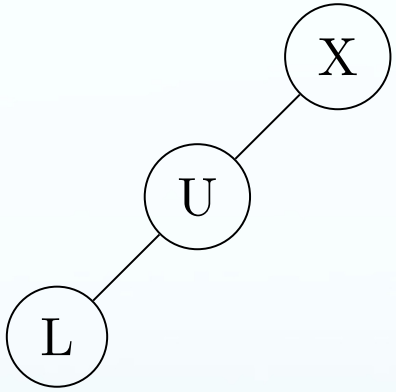
Problem 1 [class]: Rotation

In the following binary trees, rotate the ‘X’ node to the right (that is, rotate it and its left child). Do these rotations improve the overall balance of the tree?

(b)



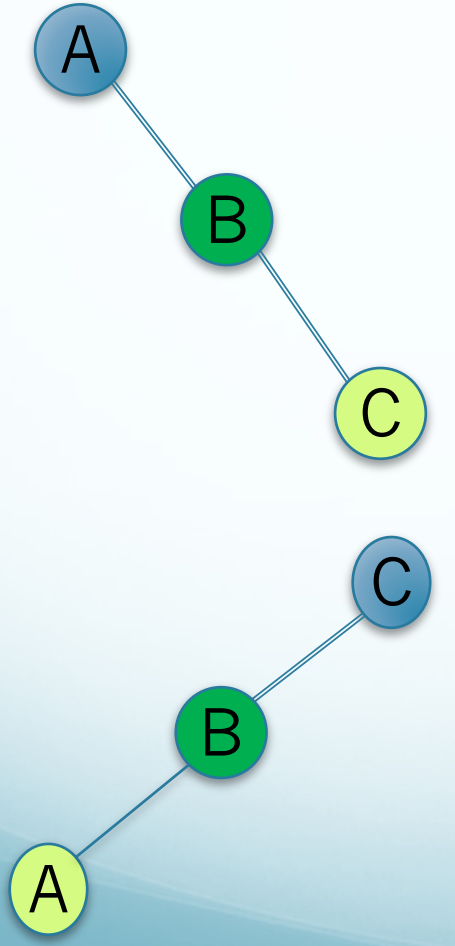
(c)



Remember: 2 types of rotations: *Right Rotation* (a node and its left child), and *Left Rotation* (a node and its right child)

AVL: Two Basic Rotations: 1) Single Rotation

Applied when an AVL (subtree) is a ”stick”:



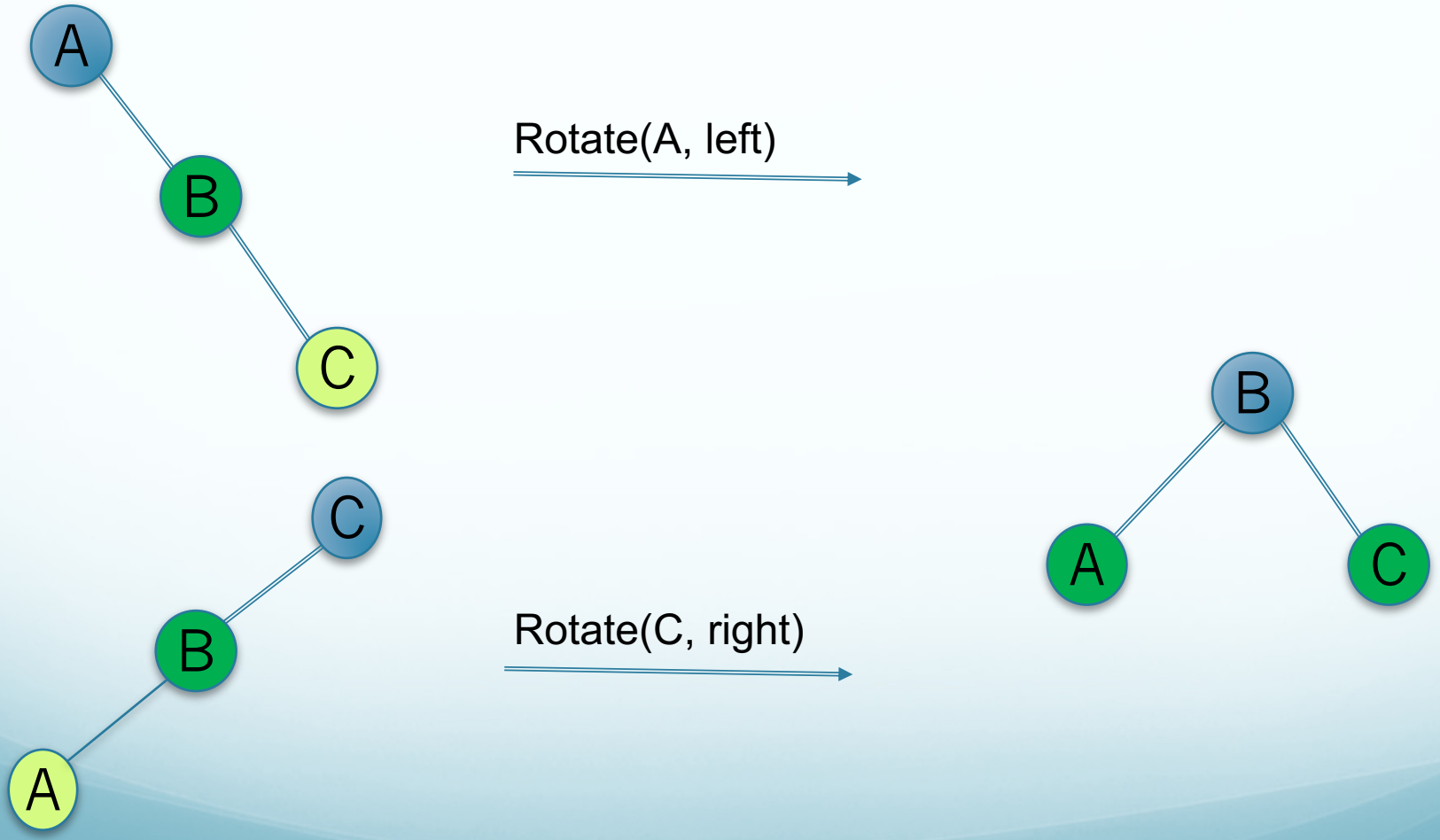
Questions we should ask ourselves:

- Which node (or subtree) is unbalanced?
- Which rotation can be done?

HERE:
→ Rotate the root and hence balance the stick

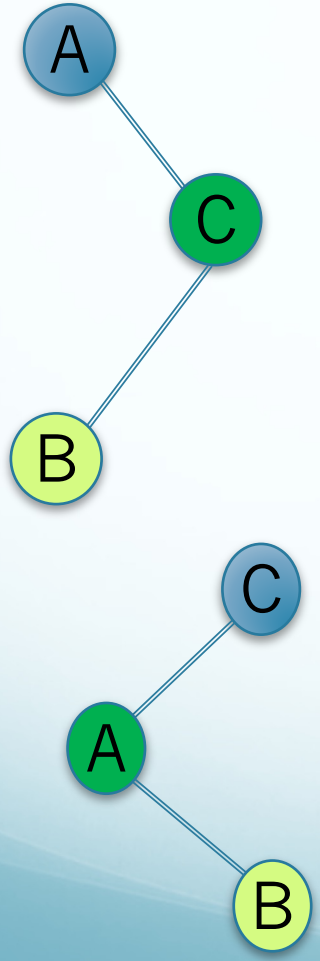
AVL: Two Basic Rotations: 1) Single Rotation

Applied when an AVL (subtree) is a ”stick”:



AVL: Two Basic Rotations: 2) Double Rotation

Applied when an AVL (subtree) is not a “stick”:



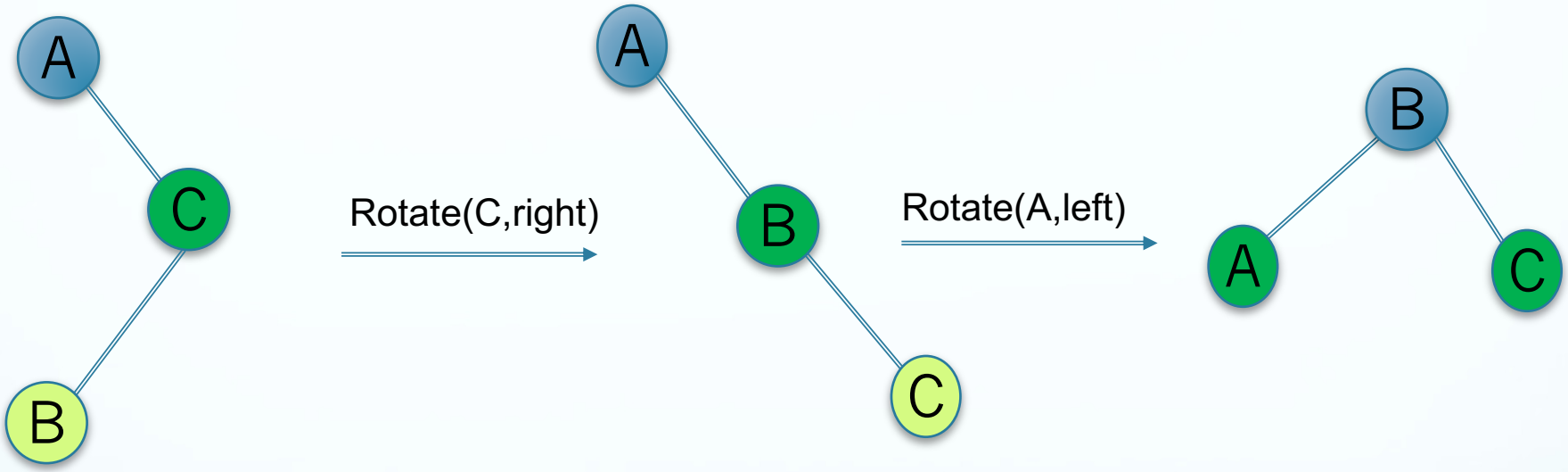
Rotation1:

- Rotate the child of the unbalanced root and turn the tree to a stick

Rotation2:

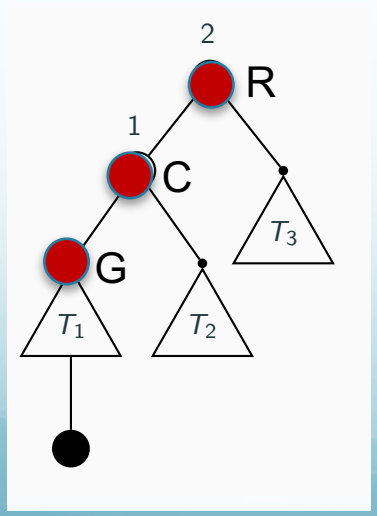
- Rotate the root of the stick.

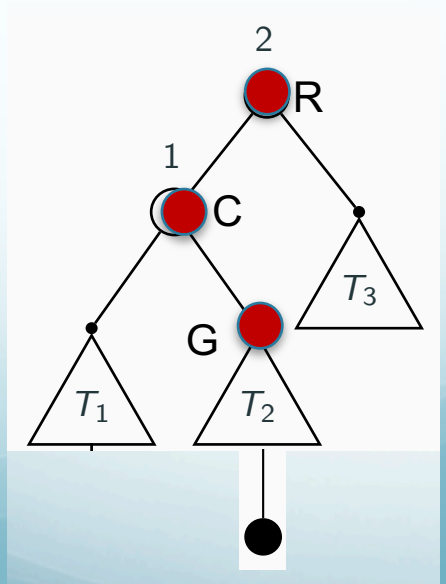
2) Double Rotation Example: RL rotation



AVL: Using Rotations to rebalance AVL

- Problem: When inserting/deleting node, AVL might become unbalanced
- Approach: Rotations (Rotate WHAT?, and HOW?)
- Rotate WHAT?
  - Walk up, find the *lowest* subtree R which is unbalanced
- HOW
  - Consider *the first 3 nodes* R→C→G in the path from root R to the just-inserted node
  - Apply a single rotations if that path is a stick, double rotation otherwise





AVL Tree Insertion

Insert the following keys into an initially-empty AVL Tree.

Class example:

20 10 5 15 30 17 8 2 12 4

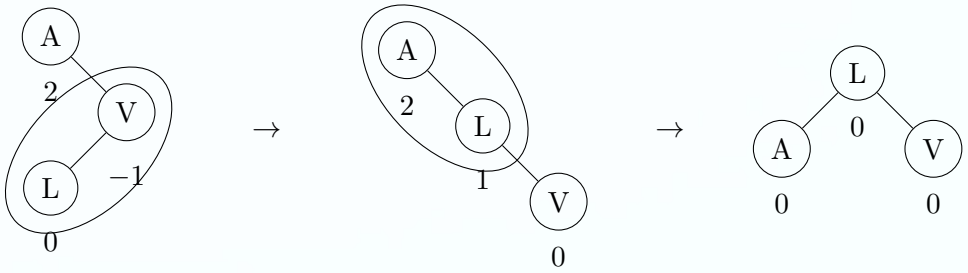
Problem 3 [*group/individual*]:

A V L T R E X M P

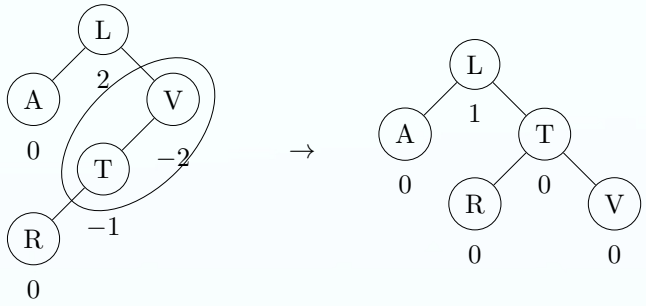
Problem 3: Check your solution

Insert the following letters into an initially-empty AVL Tree: A V L T R E X M P

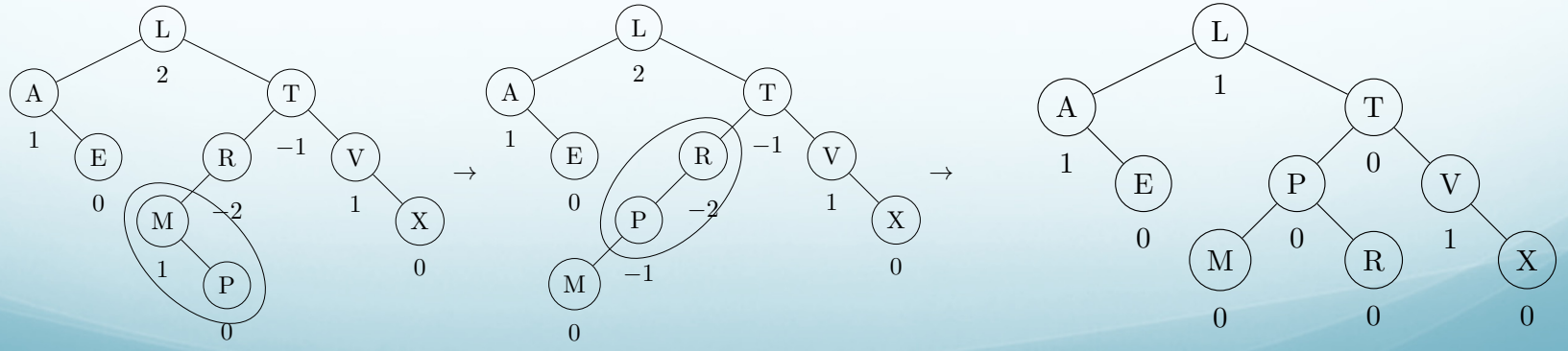
after A V L:



after T R:

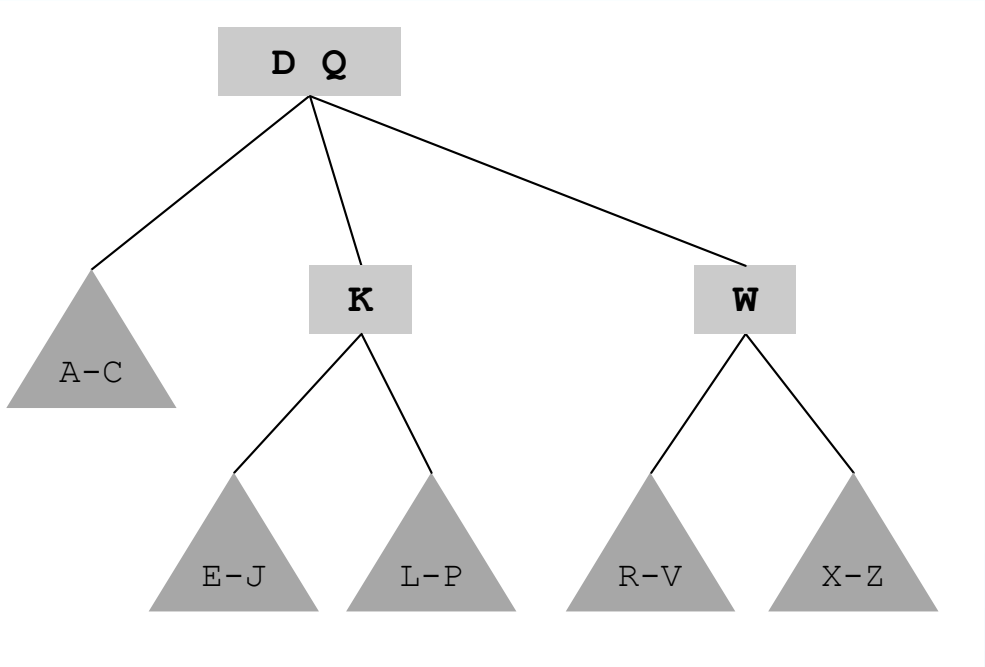


after R E X M P



2-3 Trees: What? What Special?

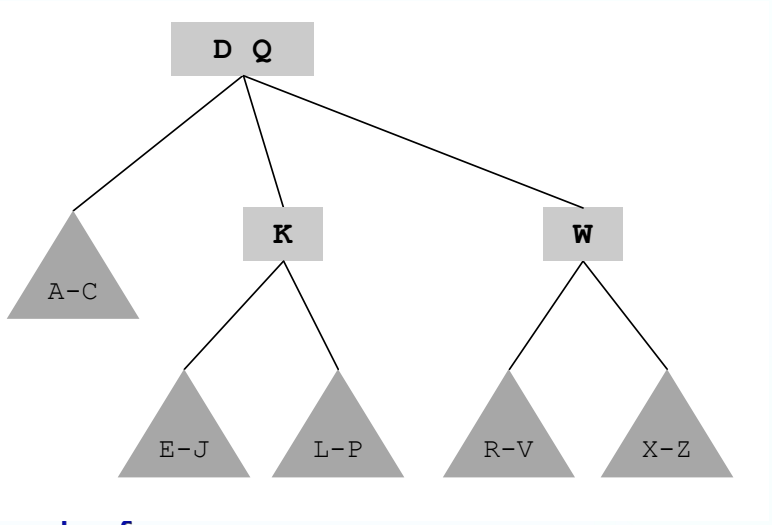
- *What?* It's a search tree, but not a binary! Each node might have 1 or 2 data, and hence 2 or 3 children.



- Always insert to a leaf node!

2-3 Trees

- *What?* It's a search tree, but not a binary! Each node might have 1 or 2 data, and hence 2 or 3 children.



- How to insert:
  - start from root, go down and insert to a leaf
  - if the new leaf has <=2 data, it's ok
  - if the new leaf has 3 data: promote the median data to the parent (the promoting might continue upward)

2-3 Tree Insertion

Insert the following keys into an initially-empty 2-3 Tree.

Class example:

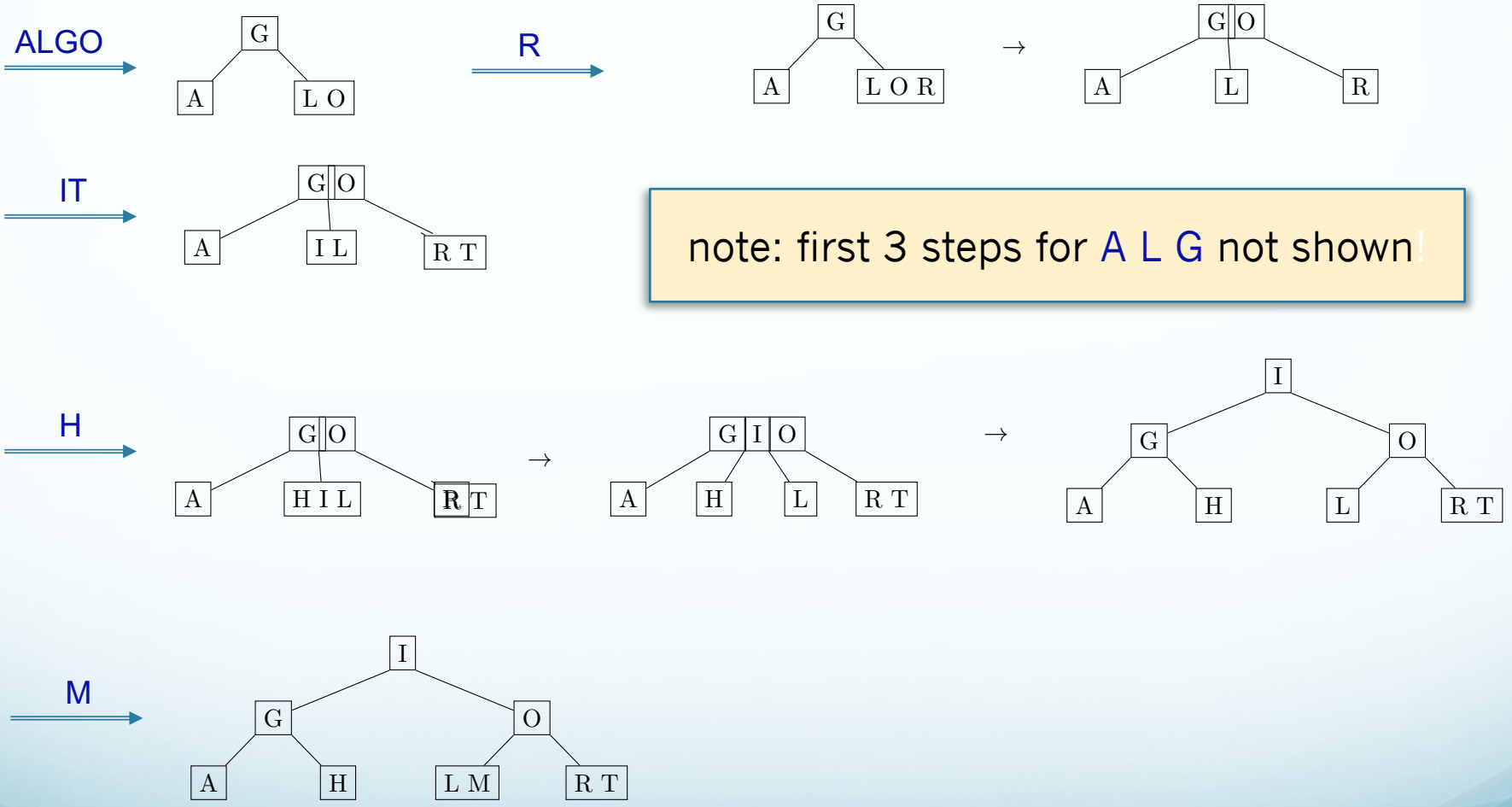
20 10 5 15 30 17 8 2 12 4

Problem 4:

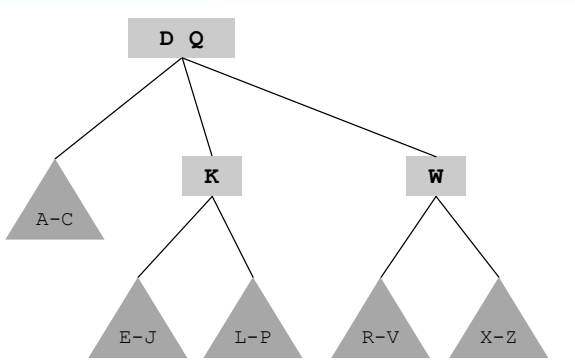
A L G O R I T H M

Problem 4: Check your solution

Insert the following keys into an initially-empty 2-3 Tree: A L G O R I T H M

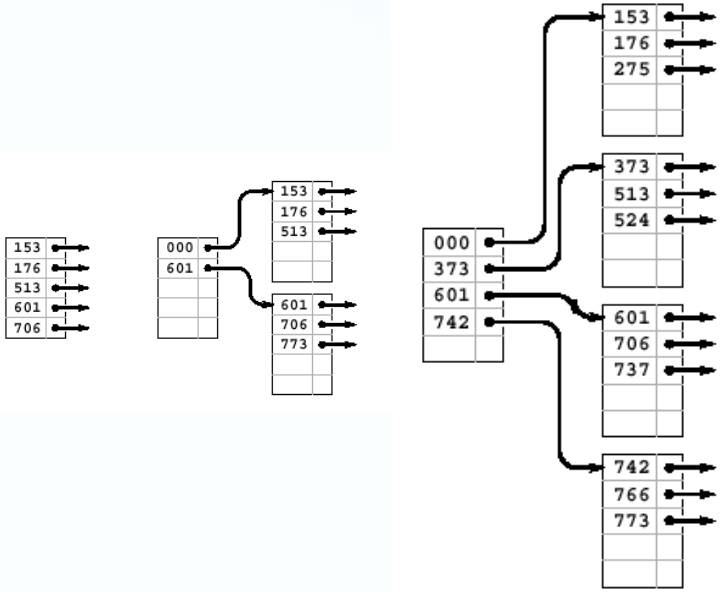


2-3 Trees, 2-3-4 Trees, B-Tree

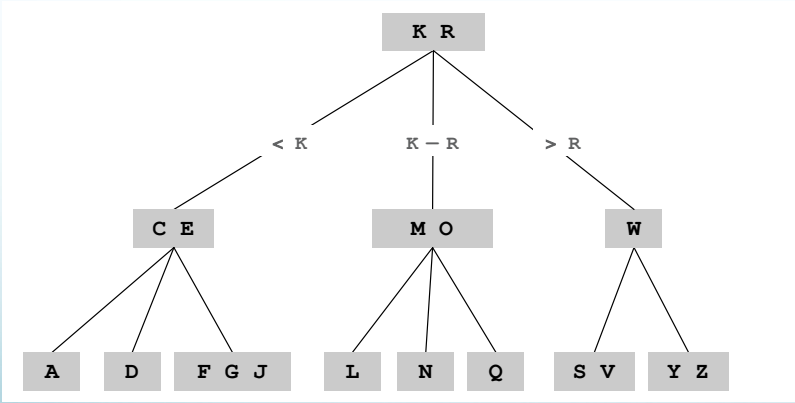


2-3 trees= B-trees of order 3

(order= max number of children)



a B-tree of order 6



2-3-4 trees= B-trees of order 4

B-tree principles

- Always insert at leaves
- When a node full: promote the median data to the node's parent [and walk up further if needed]

Labasy

- Q&A on previous week materials, and/or
- Implement BST: insert, build tree from data, printing the tree. Note:
  - Build your program from scratch
  - But you can use last week list and queue modules
- And/or: Implement 2-3-4 Tree with the above operations
- For easy printing of trees, build a complete tree first, using data:

```
50 30 80 20 40 60 90 15 25 35 45 55 65
```

Simple defs for binary trees

```
typedef struct treenode *tree_t;
struct treenode {
  int key;
  tree_t left, right;
} ;
```

```
tree_t insert(int key, tree_t t);
//OR
void insert(int key, tree_t *t);
```

Example

```
tree_t t= NULL;

t= malloc(sizeof(*t));
assert(t);
t->left= t->right= NULL;
t->data= 50;
t= insert(t, 10); //OR
insert(&t, 10);
// depending on insert header
```