04-630
Data Structures and Algorithms for Engineers

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 14

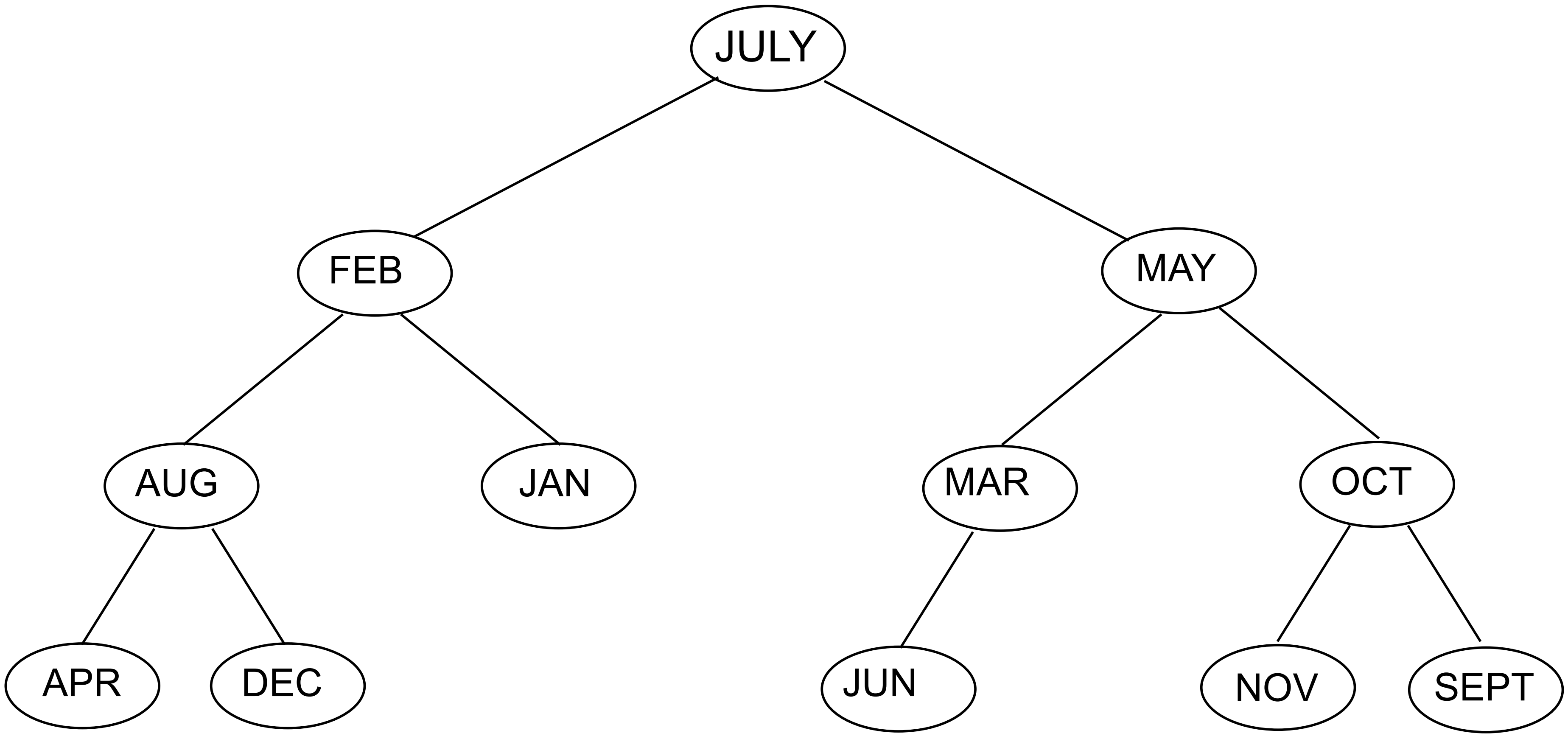
Trees

- Types of trees
- Binary Tree ADT
- Binary Search Tree
- Optimal Code Trees
- Height Balanced Trees
  - AVL Trees
  - Red-Black Trees
- Huffman's Algorithm

AVL Trees

- We know from our study of Binary Search Trees (BST) that the average search and insertion time is $O(\log n)$
  - If there are n nodes in the binary tree it will take, on average, $log_2 n$ comparisons/probes to find a particular node (or find out that it isn't there)
- However, this is only true if the tree is 'balanced'
  - Such as occurs when the elements are inserted in random order

AVL Trees

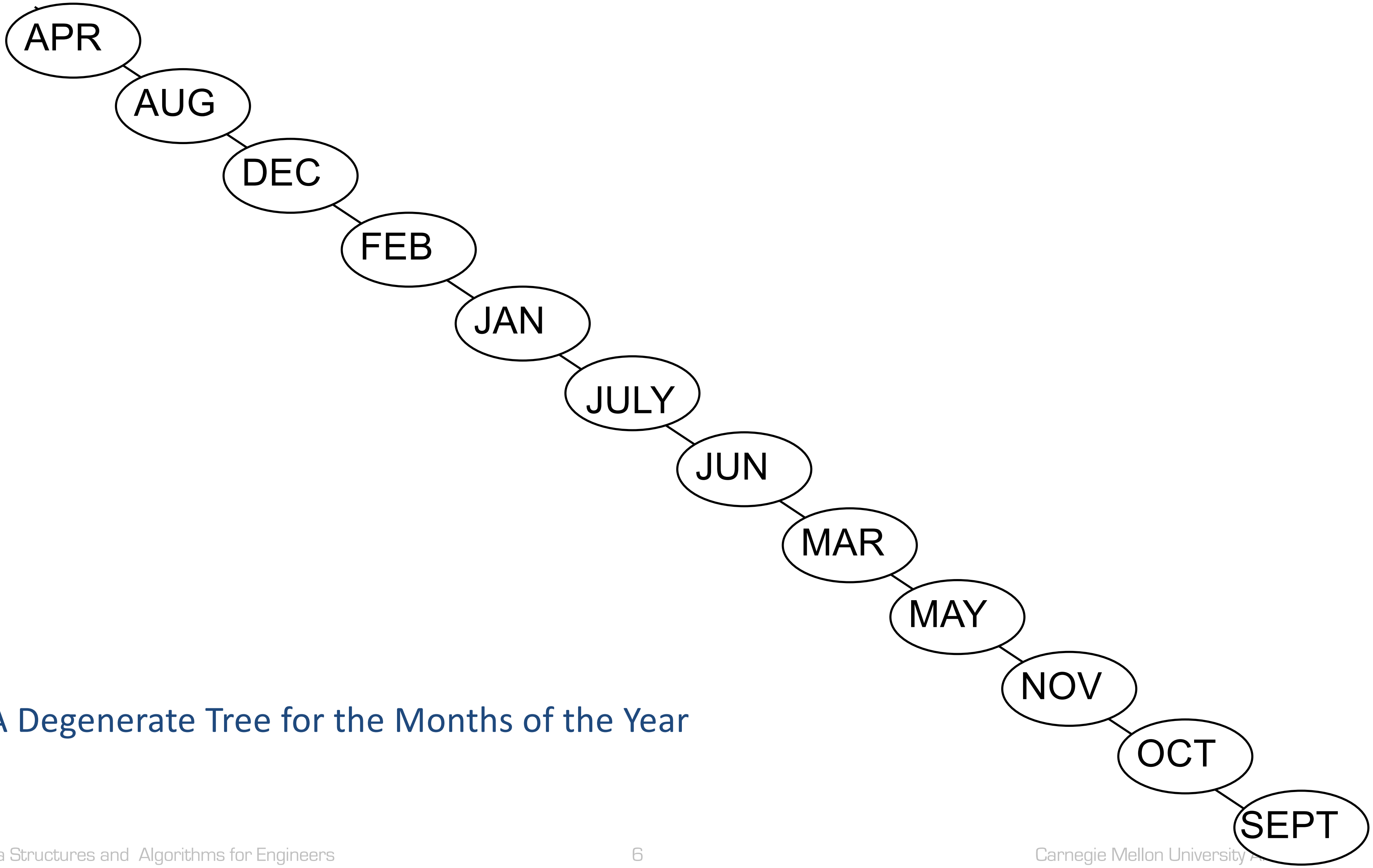


A Balanced Tree for the Months of the Year

AVL Trees

- However, if the elements are inserted in lexicographic order (i.e. in sorted order) then the tree degenerates into a skinny tree

AVL Trees



A Degenerate Tree for the Months of the Year

AVL Trees

- If we are dealing with a dynamic tree …
- nodes are being inserted and deleted over time
  - For example, directory of files
  - For example, index of university students
- we may need to restructure - balance - the tree so that we keep it
  - Fat
  - Full
  - Complete

AVL Trees

- Adelson-Velskii and Landis in 1962 introduced a binary tree structure that is balanced with respect to the heights of its subtrees

- Insertions (and deletions) are made such that the tree
  - starts off
  - and remains

- Height-Balanced

AVL Trees

- Definition of AVL Tree

- An empty tree is height-balanced

- If T is a non-empty binary tree with left and right sub-trees T_1 and T_2, then T is height-balanced iff

  – T_1 and T_2 are height-balanced, and

  – $|height(T_1) - height(T_2)| \leq 1$

AVL Trees

- So, every sub-tree in a height-balanced tree is also height-balanced

Recall: Binary Tree Terminology

- The height of T is defined recursively as

  0 if T is empty and

  $1 + max(height(T_1), height(T_2))$ otherwise,
  where T_1 and T_2 are the subtrees of the root

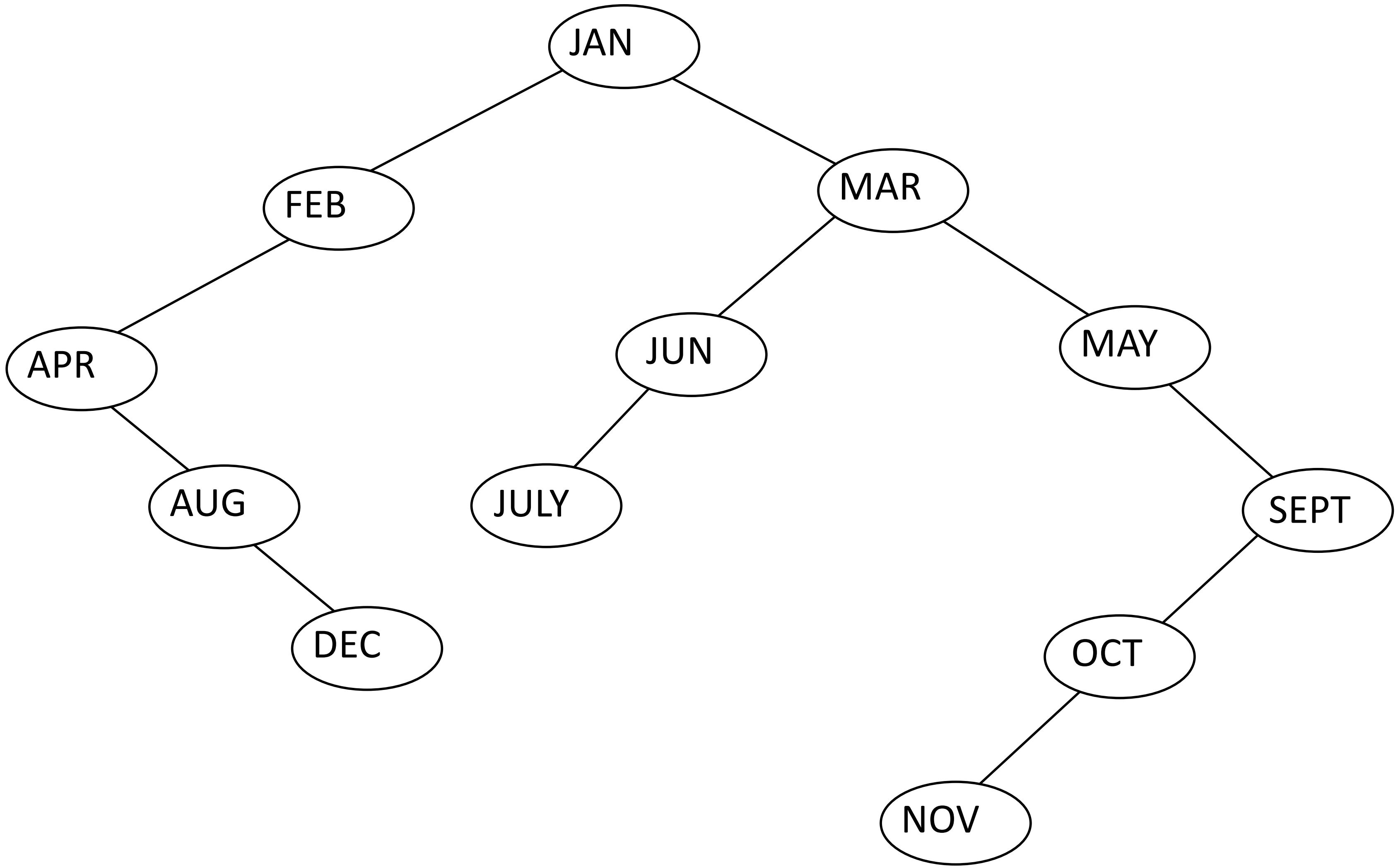
- The height of a tree is the length of a longest chain of descendents

Recall: Binary Tree Terminology

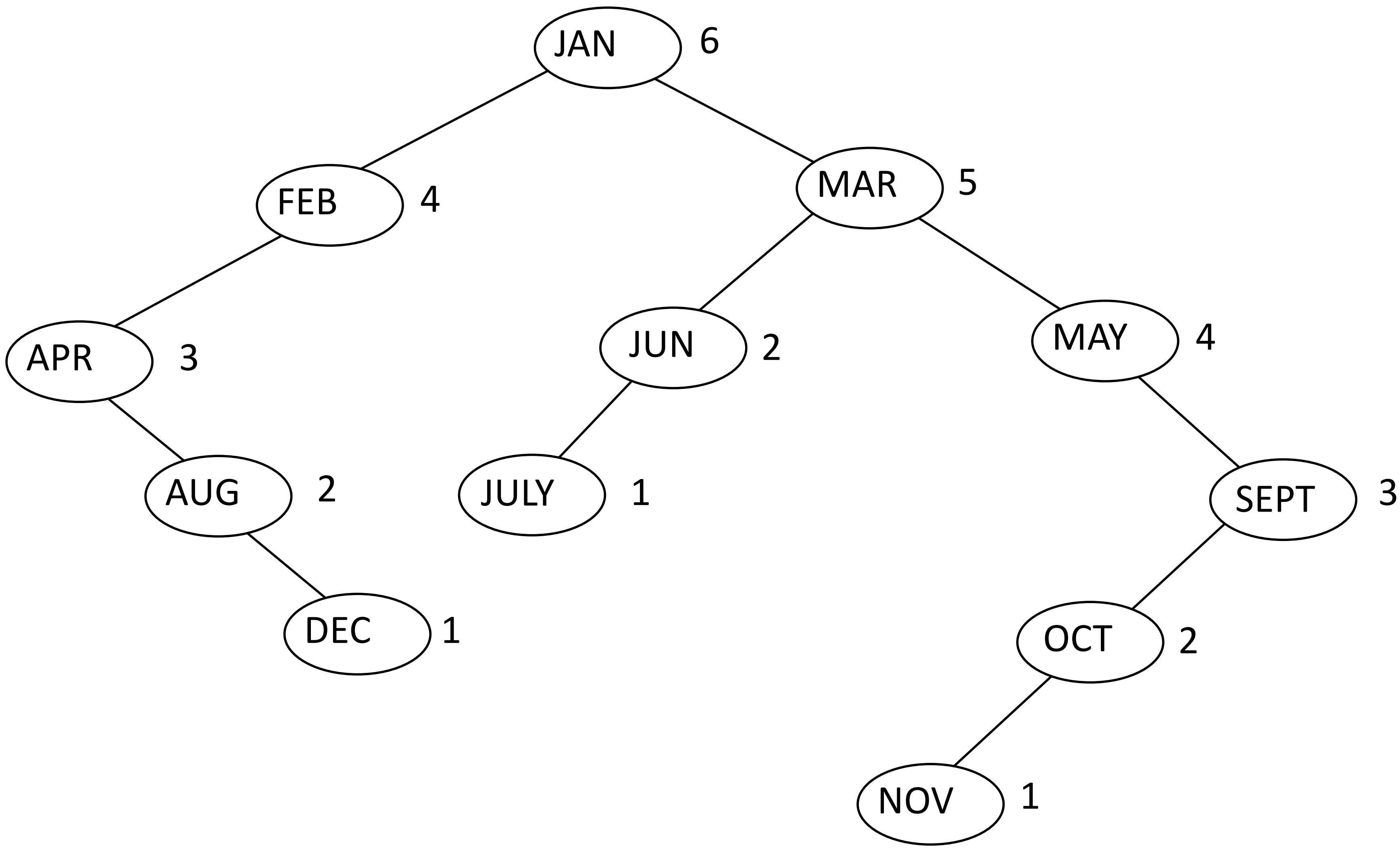
- Height Numbering
  - Number all external nodes 0
  - Number each internal node to be one more than the maximum of the numbers of its children
  - Then the number of the root is the height of T

- The height of a node u in T is the height of the subtree rooted at u

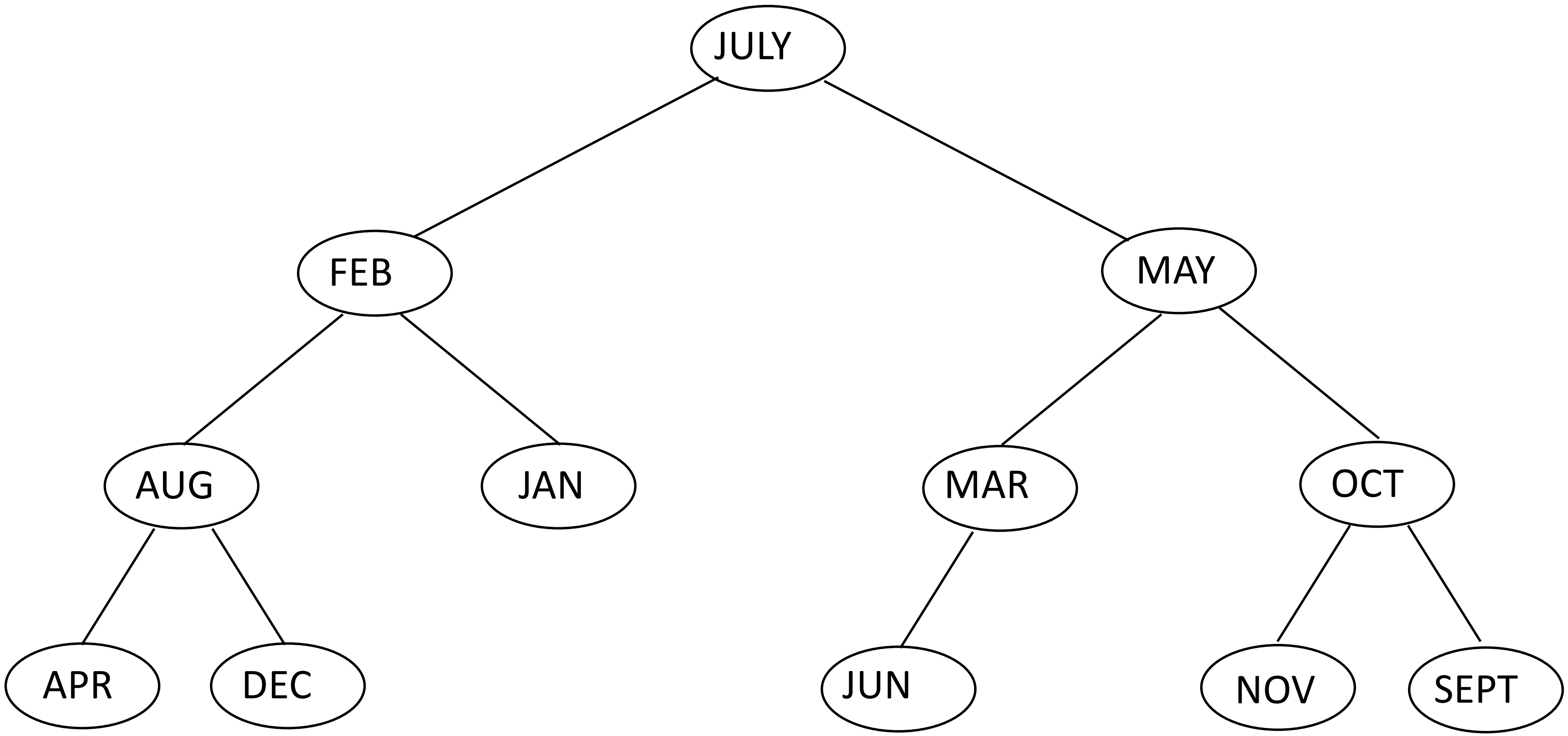
AVL Trees



AVL Trees

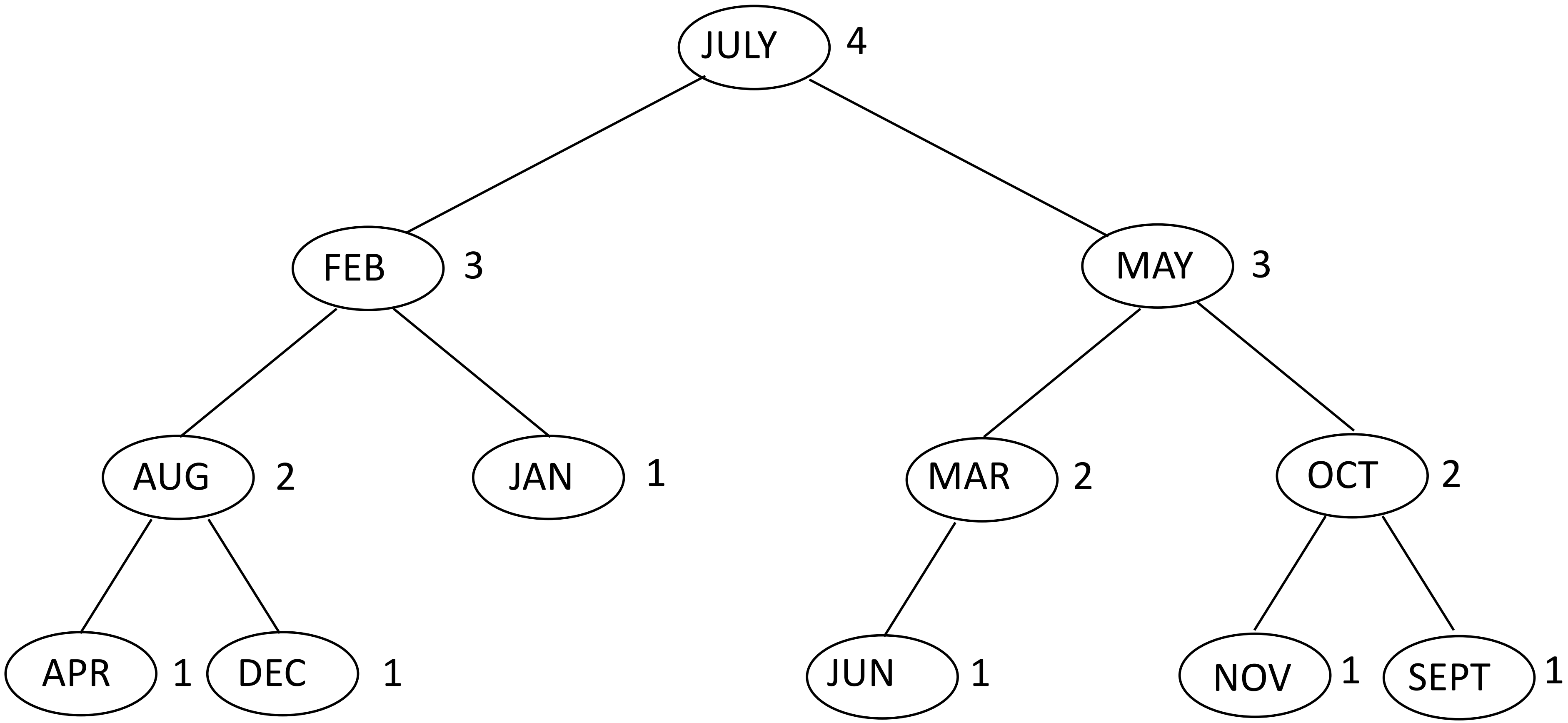

JAN 6
FEB 4
MAR 5
APR 3
JUN 2
MAY 4
AUG 2
JULY 1
SEPT 3
DEC 1
OCT 2
NOV 1

AVL Trees



A Balanced Tree for the Months of the Year

AVL Trees



A Balanced Tree for the Months of the Year

AVL Trees

- Let's construct a height-balanced tree
- Order of insertions:

  March, May, November, August, April, January, December, July, February, June, October, September

- Before we do, we need a definition of a balance factor

AVL Trees

- Balance Factor $BF(T)$ of a node T in a binary tree is defined to be

  $height(T_1)$ - $height(T_2)$

  where T_1 and T_2 are the left and right subtrees of T

- For any node T in an AVL tree
  $BF(T) = -1, 0, +1$

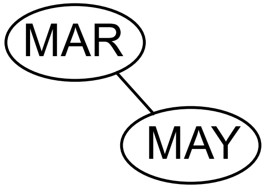
AVL Trees

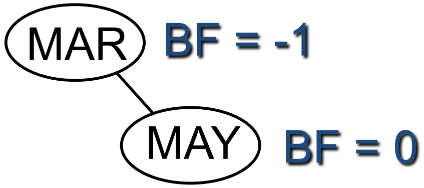
- All re-balancing operations are carried out with respect to the closest ancestor of the new node having balance factor +2 or -2

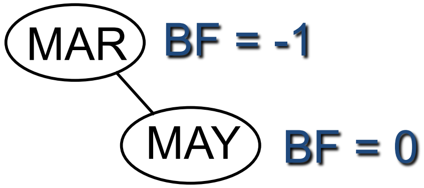
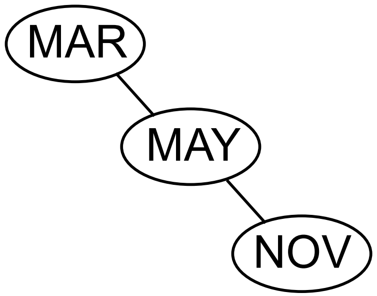
- There are 4 types of re-balancing operations (called rotations)
  - RR
  - LL (symmetric with RR)
  - RL
  - LR (symmetric with RL)

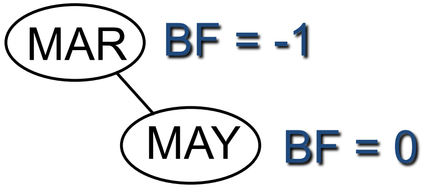
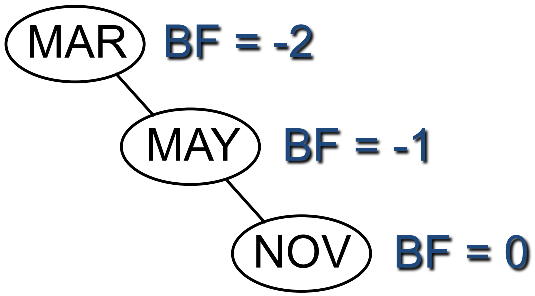
New Identifier	After Insertion	After Rebalancing
MARCH		

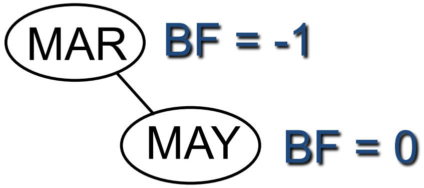
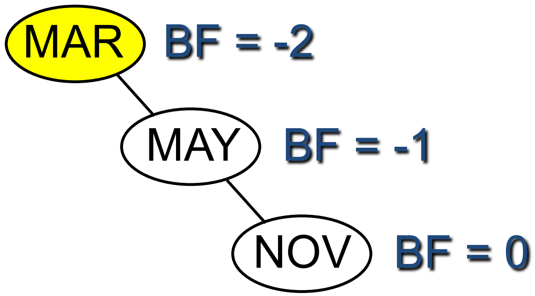
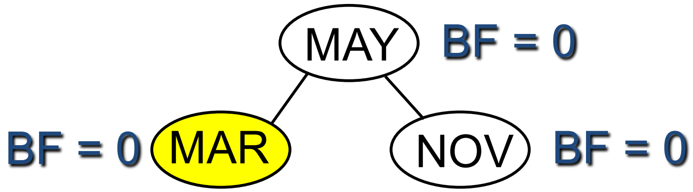
New Identifier	After Insertion	After Rebalancing
MARCH	MAR BF = 0	NO REBALANCING NEEDED

New Identifier	After Insertion	After Rebalancing
MARCH	 **BF = 0**	NO REBALANCING NEEDED
MAY		

New Identifier	After Insertion	After Rebalancing
MARCH		NO REBALANCING NEEDED
MAY		NO REBALANCING NEEDED

New Identifier	After Insertion	After Rebalancing
MARCH		NO REBALANCING NEEDED
MAY		NO REBALANCING NEEDED
NOVEMBER		

New Identifier	After Insertion	After Rebalancing
MARCH		NO REBALANCING NEEDED
MAY		NO REBALANCING NEEDED
NOVEMBER		

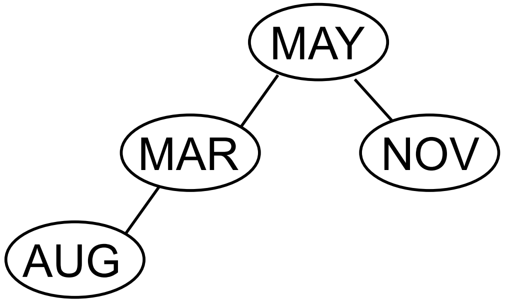
New Identifier	After Insertion	After Rebalancing
MARCH		NO REBALANCING NEEDED
MAY		NO REBALANCING NEEDED
NOVEMBER		 RR rebalancing

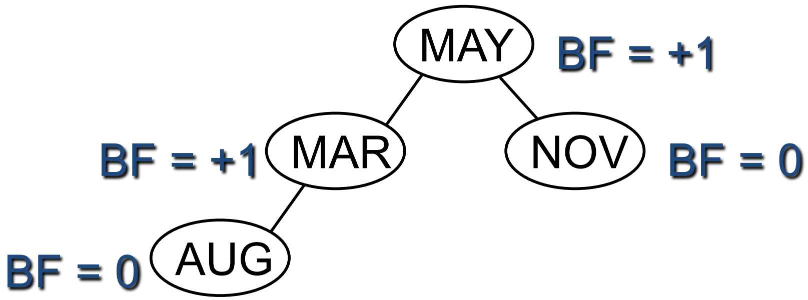
New Identifier

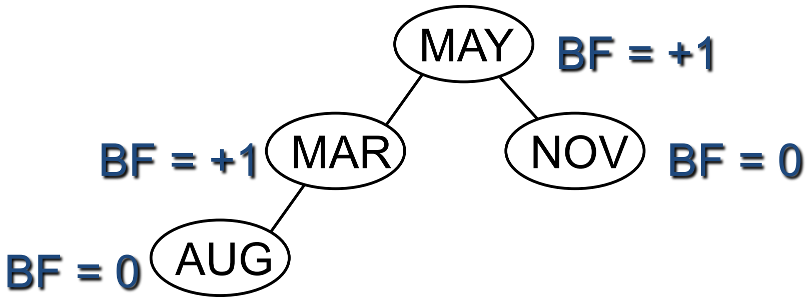
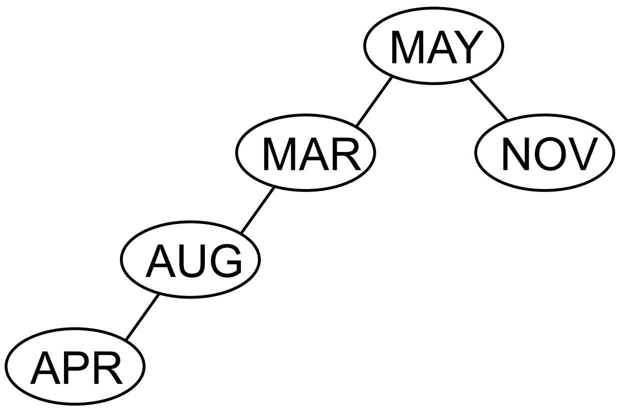
After Insertion

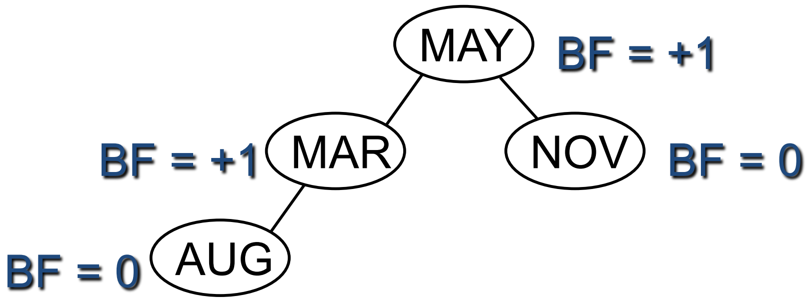
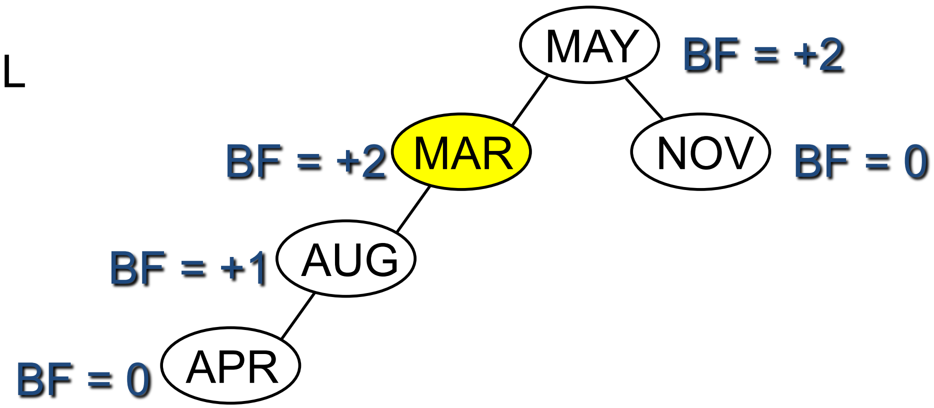
After Rebalancing

AUGUST



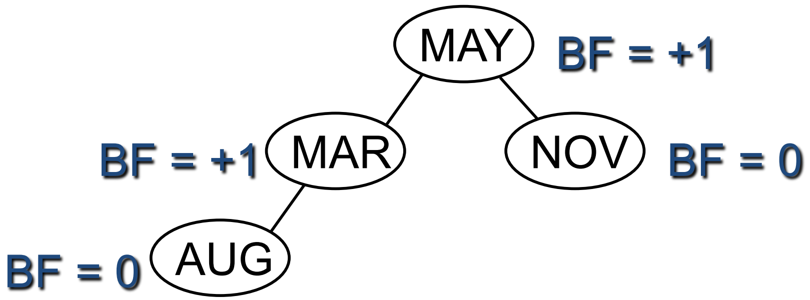
New Identifier	After Insertion	After Rebalancing
AUGUST		NO REBALANCING NEEDED

New Identifier	After Insertion	After Rebalancing
AUGUST		NO REBALANCING NEEDED
APRIL		

New Identifier	After Insertion	After Rebalancing
AUGUST		NO REBALANCING NEEDED
APRIL		

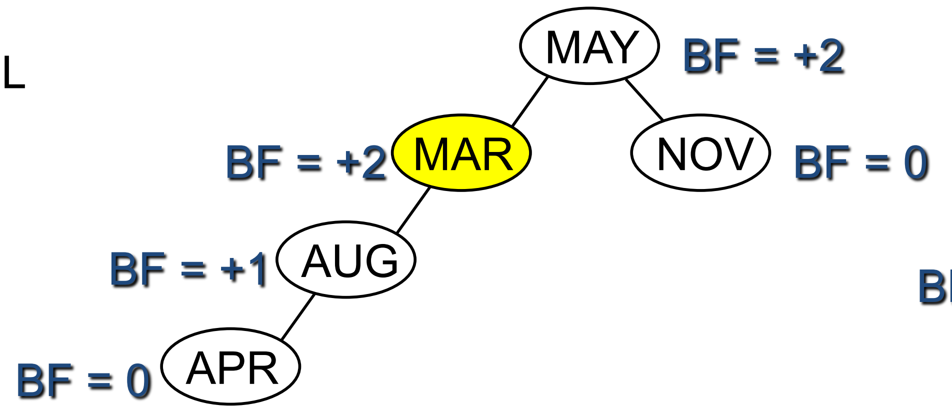
New Identifier	After Insertion	After Rebalancing

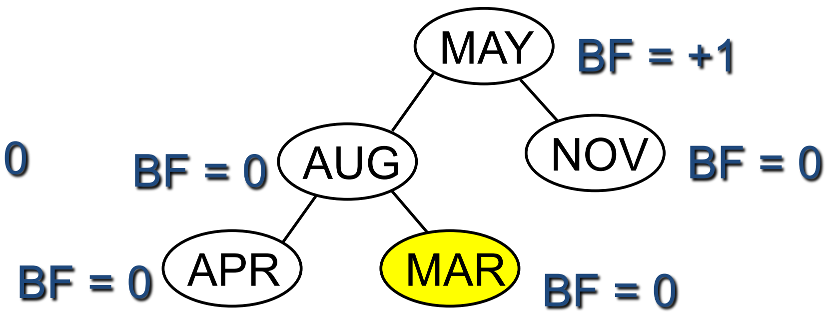
AUGUST



NO REBALANCING NEEDED

APRIL



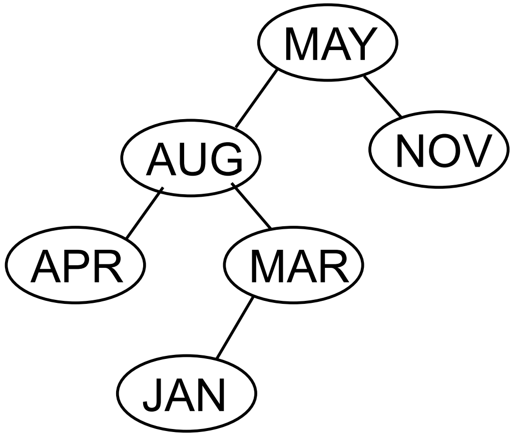


LL rebalancing

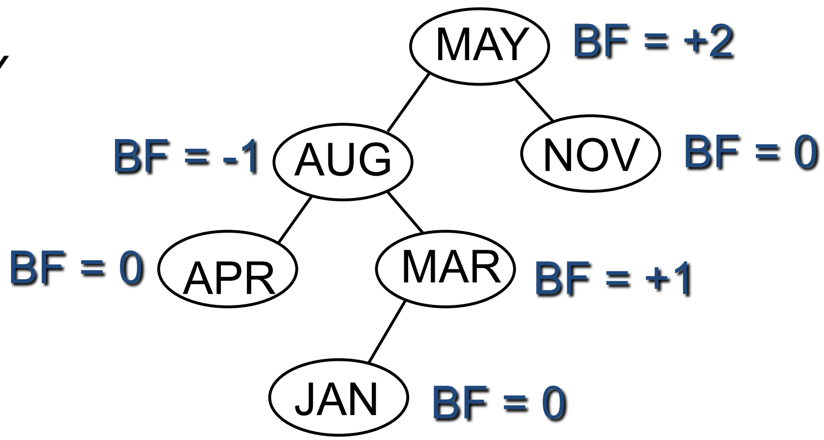
New Identifier

JANUARY

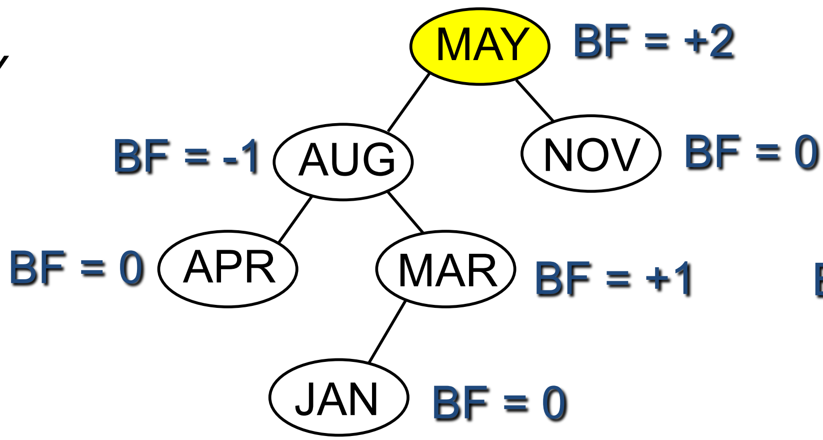
After Insertion

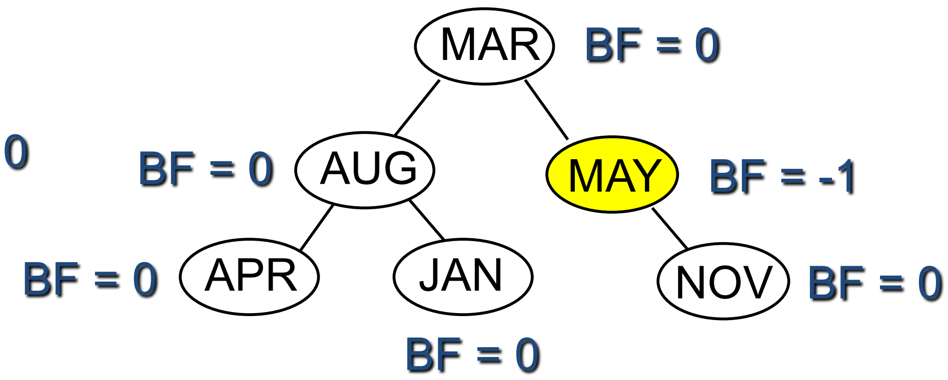


After Rebalancing

New Identifier	After Insertion	After Rebalancing
JANUARY		

New Identifier	After Insertion	After Rebalancing
JANUARY		

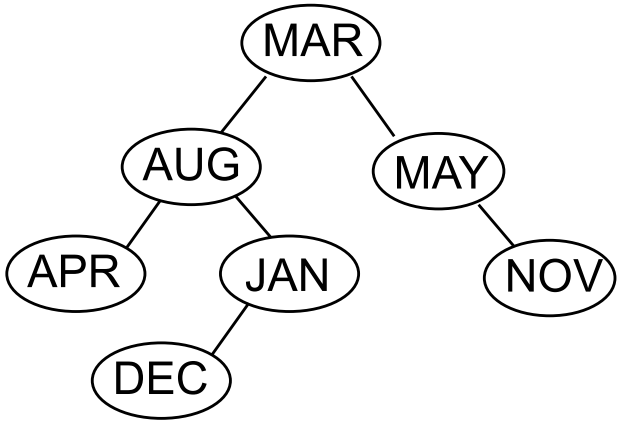




LR rebalancing

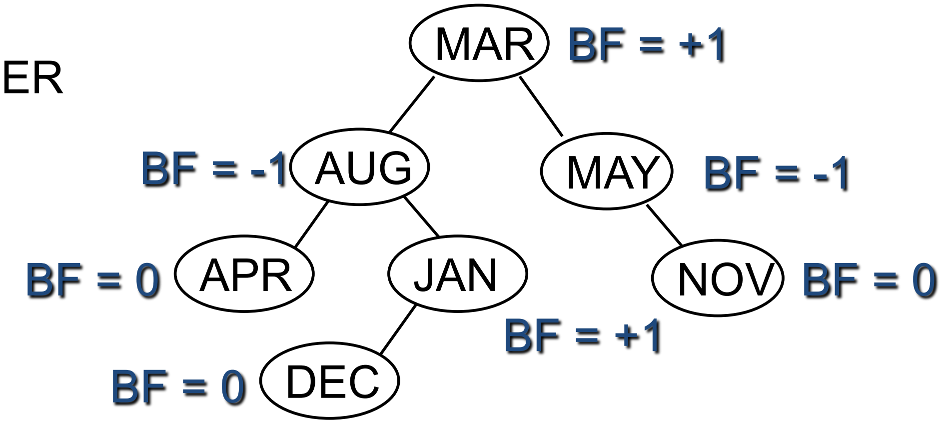
New Identifier | After Insertion | After Rebalancing

DECEMBER



New Identifier | After Insertion | After Rebalancing

DECEMBER



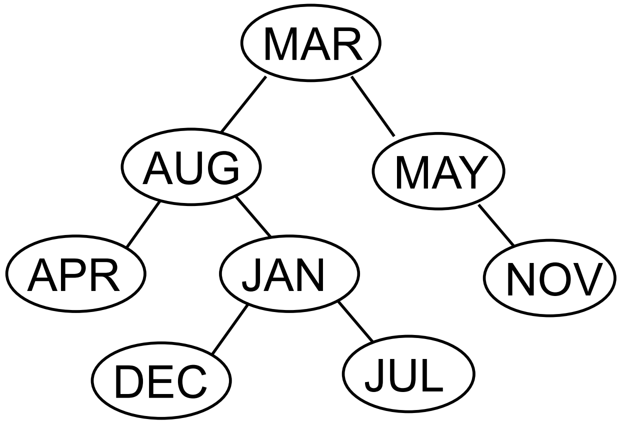
NO REBALANCING NEEDED

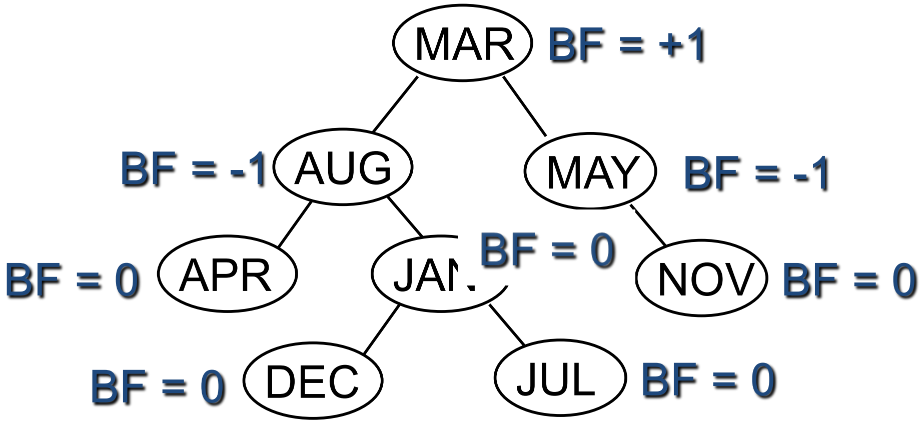
New Identifier

After Insertion

After Rebalancing

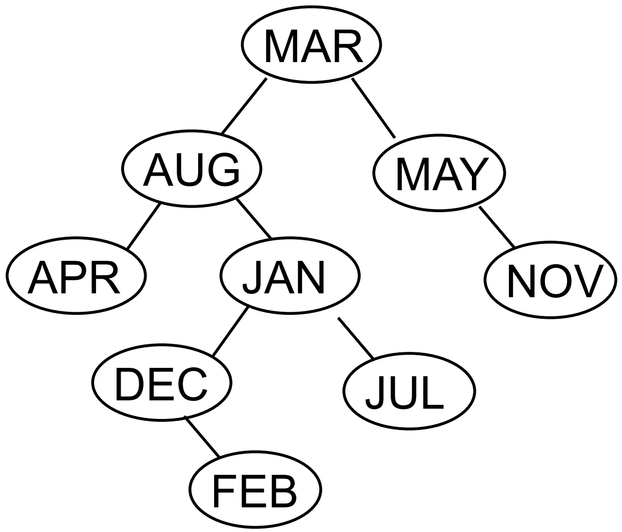
JULY



New Identifier	After Insertion	After Rebalancing
JULY		NO REBALANCING NEEDED

New Identifier | After Insertion | After Rebalancing

FEBRUARY

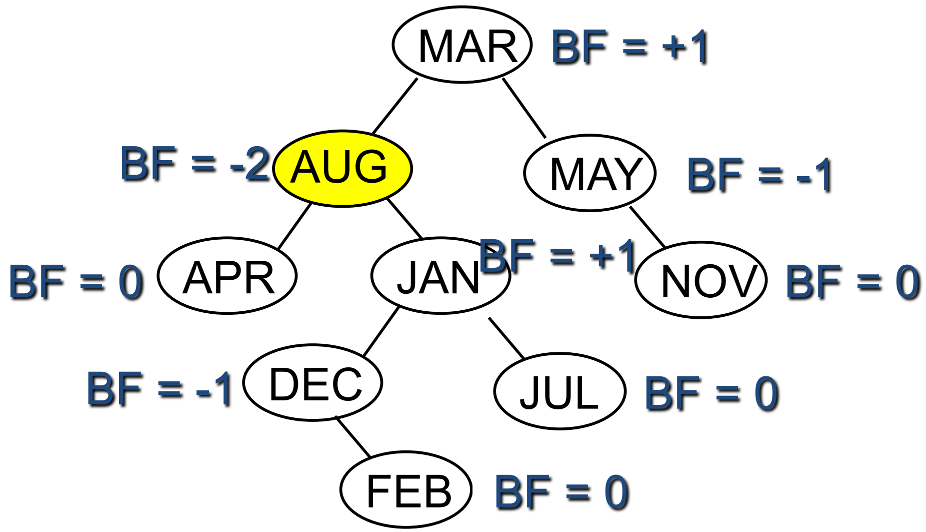


New Identifier

After Insertion

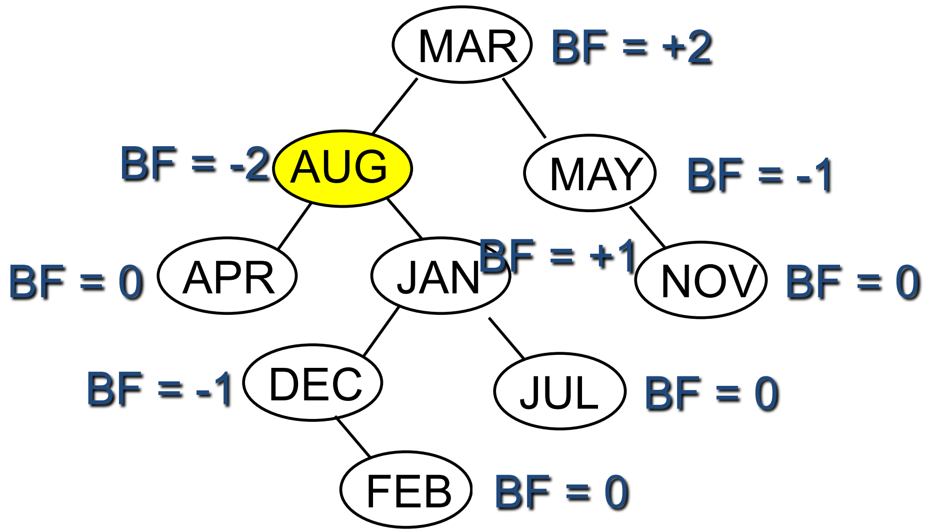
After Rebalancing

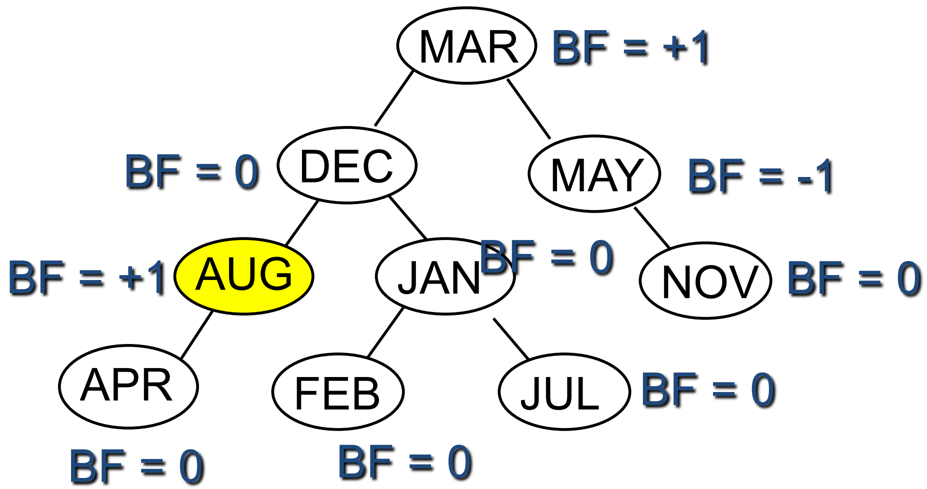
FEBRUARY



New Identifier | After Insertion | After Rebalancing

FEBRUARY

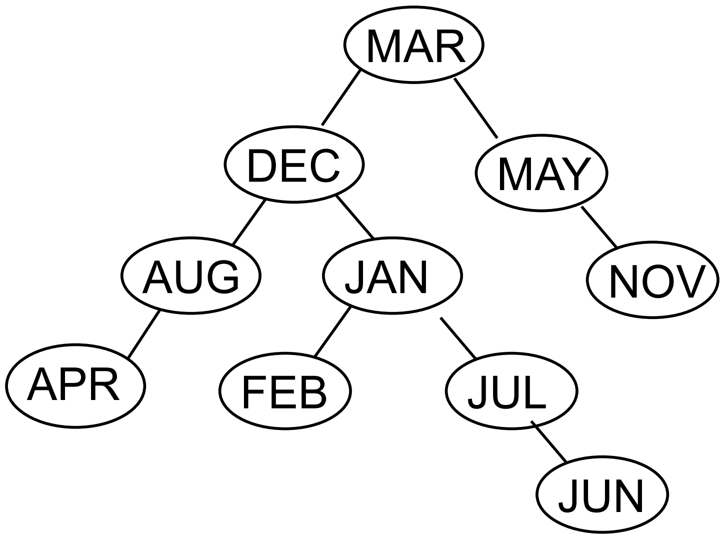




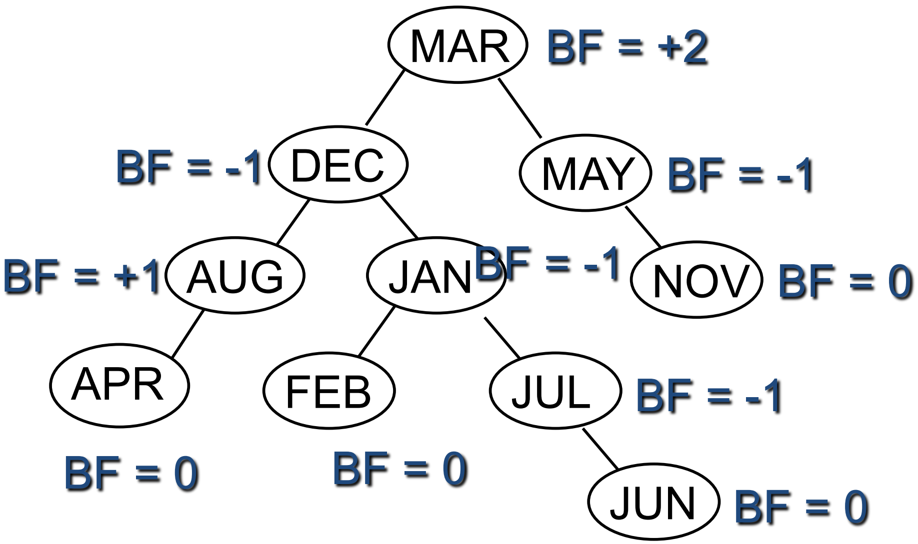
RL rebalancing

New Identifier | After Insertion | After Rebalancing

JUNE



New Identifier	After Insertion	After Rebalancing
JUNE		

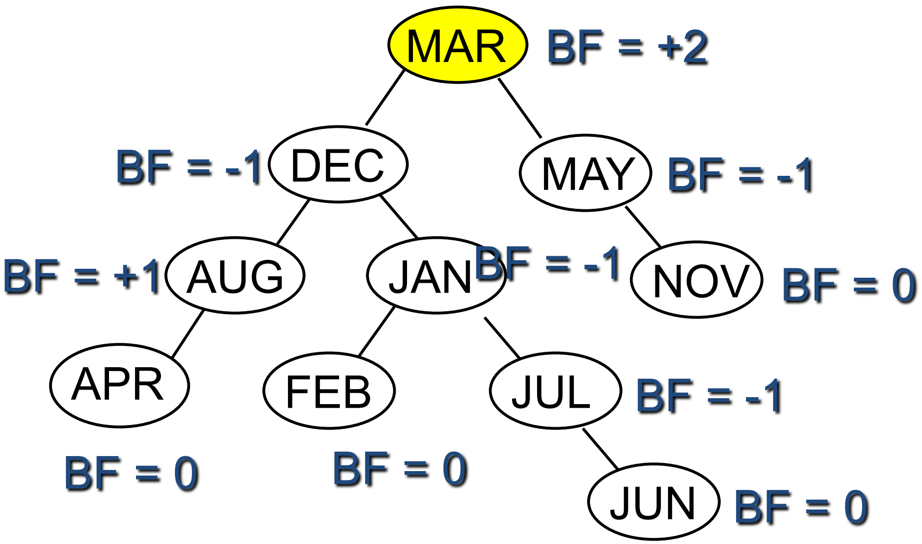


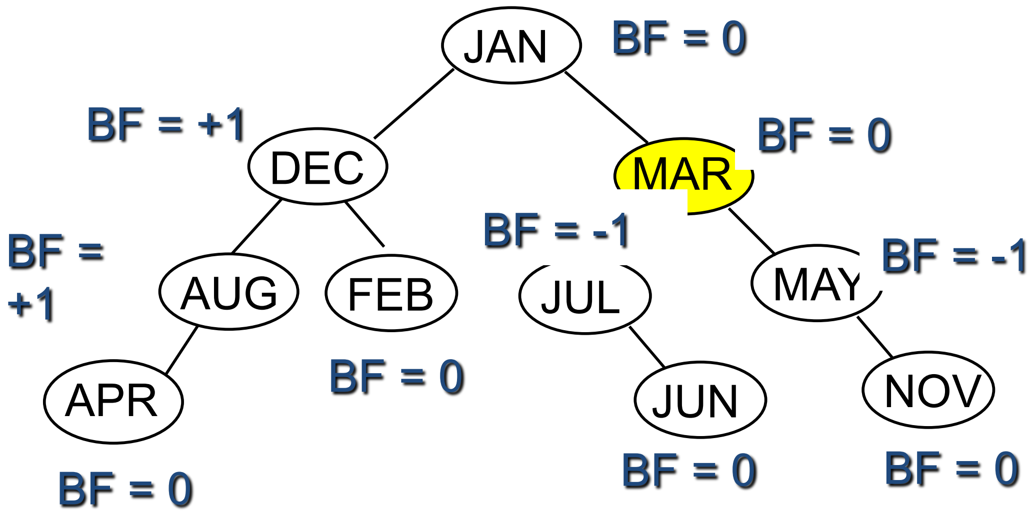
New Identifier

After Insertion

After Rebalancing

JUNE





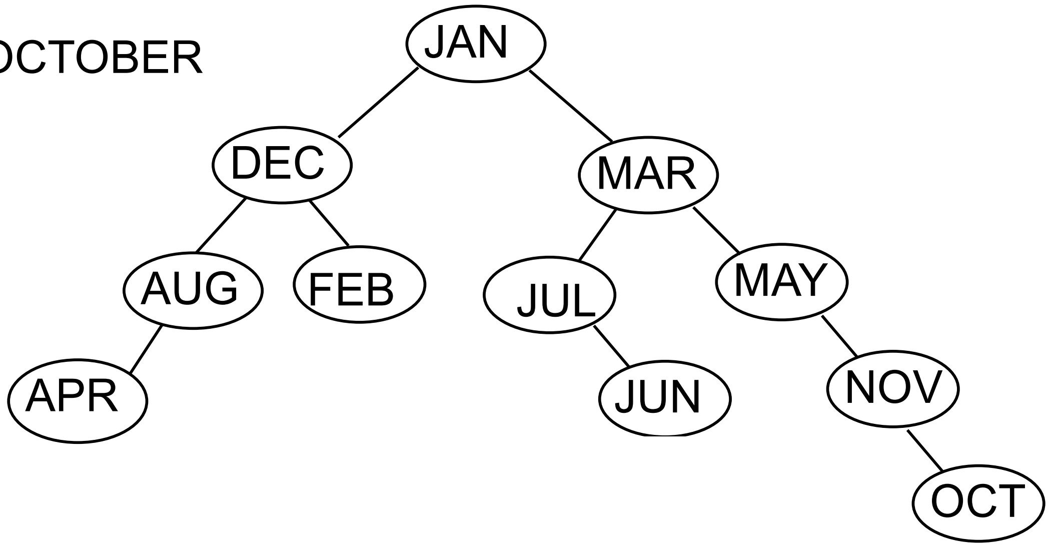
LR rebalancing

New Identifier

After Insertion

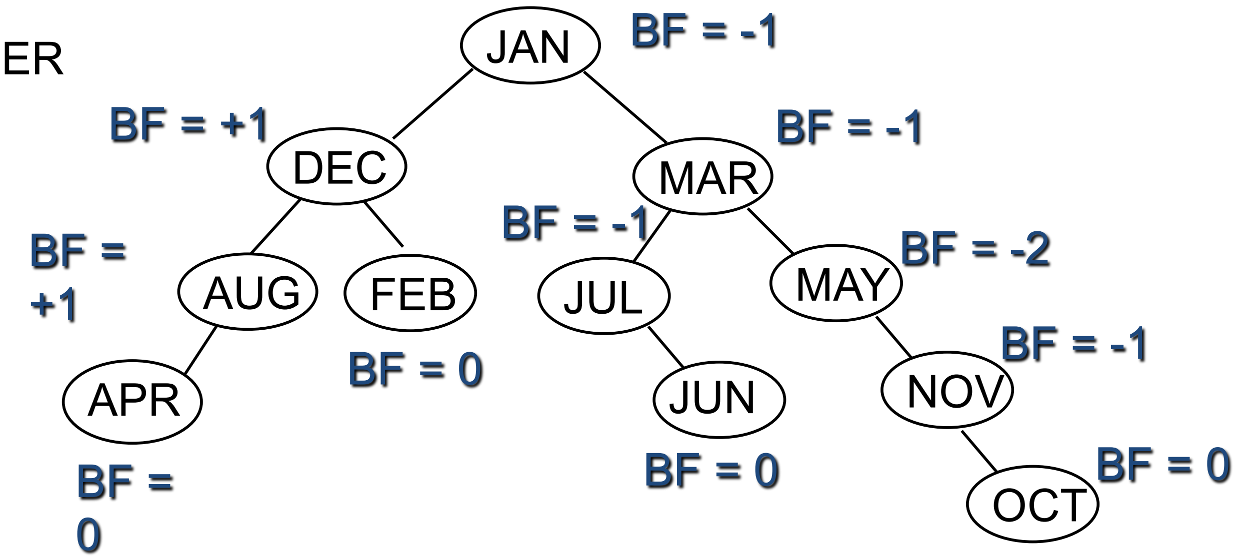
After Rebalancing

OCTOBER

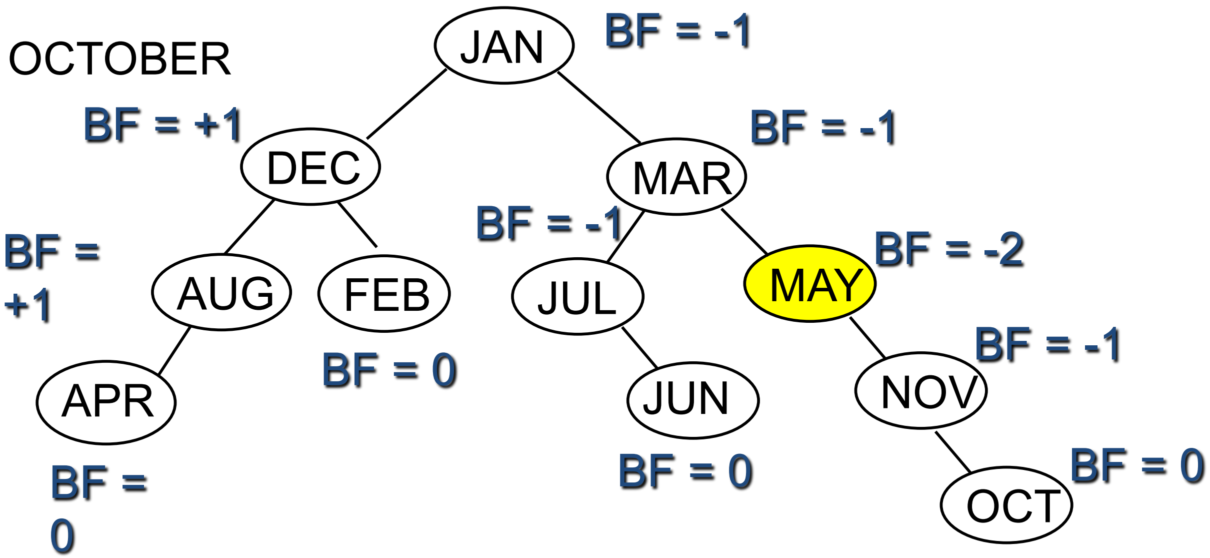


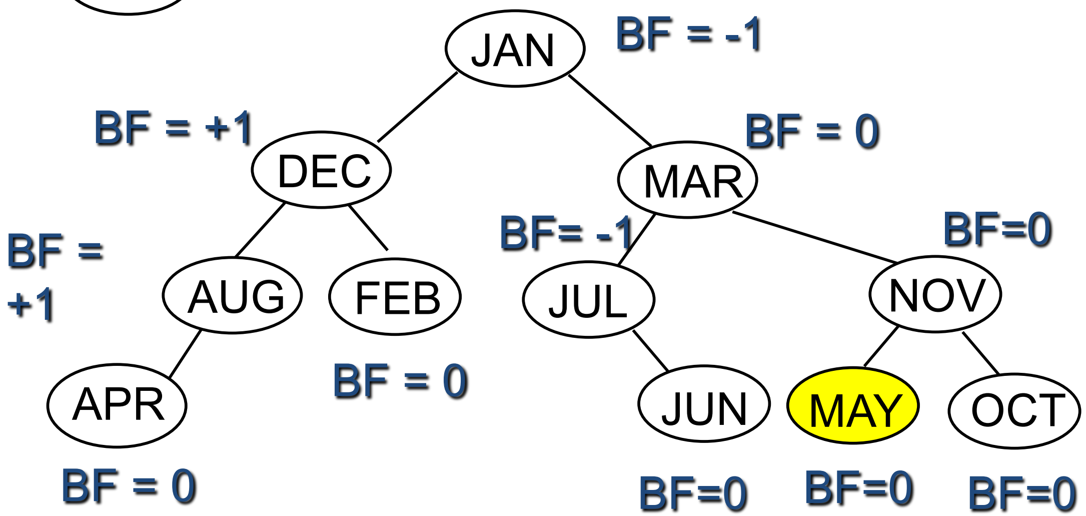
New Identifier | After Insertion | After Rebalancing

OCTOBER



New Identifier	After Insertion	After Rebalancing
OCTOBER		RR rebalancing



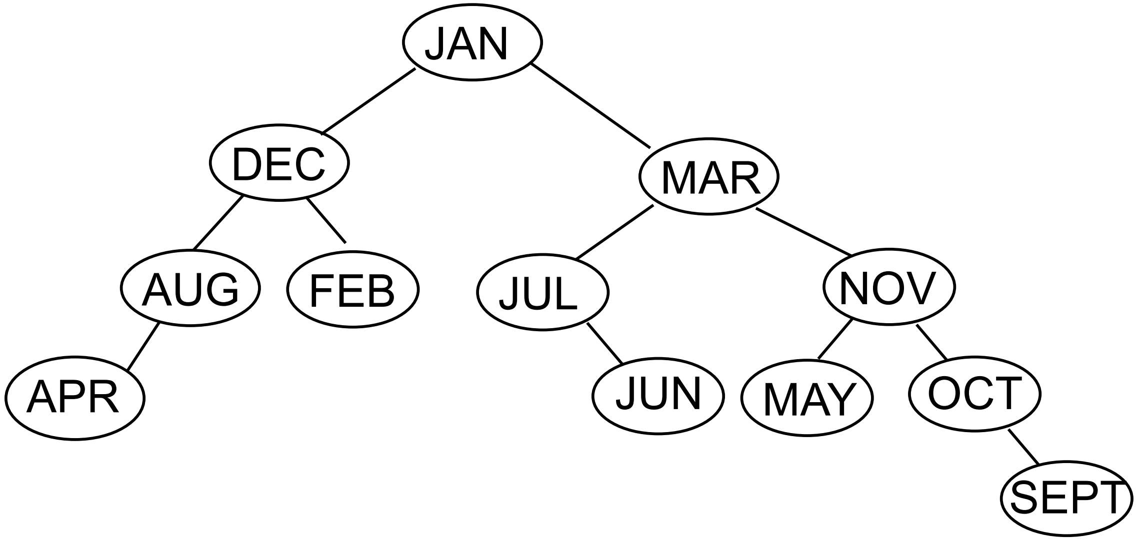


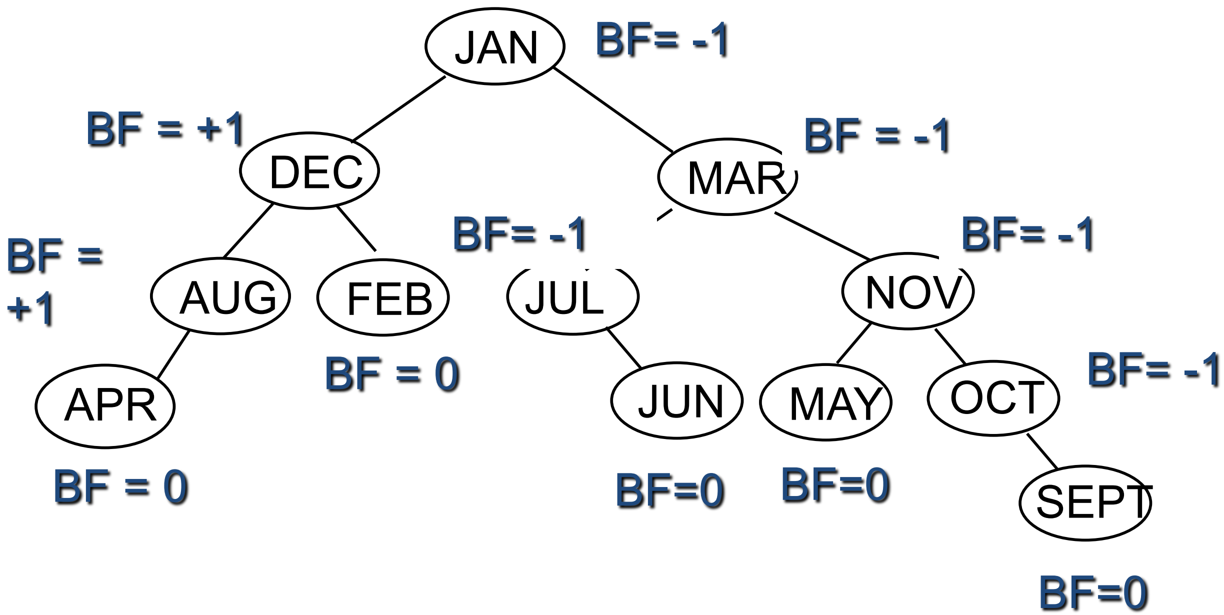
New Identifier

After Insertion

After Rebalancing

SEPTEMBER



New Identifier	After Insertion	After Rebalancing
SEPTEMBER		NO REBALANCING NEEDED

AVL Trees

- Let's refer to the node inserted as Y

- Let's refer to the nearest ancestor having balance factor +2 or -2 as A

AVL Trees

- LL: Y is inserted in the
  Left subtree of the Left subtree of A

  - LL: the path from A to Y
  - Left subtree then Left subtree

- LR: Y is inserted in the
  Right subtree of the Left subtree of A

  - LR: the path from A to Y
  - Left subtree then Right subtree

AVL Trees

- RR: Y is inserted in the
  Right subtree of the Right subtree of A

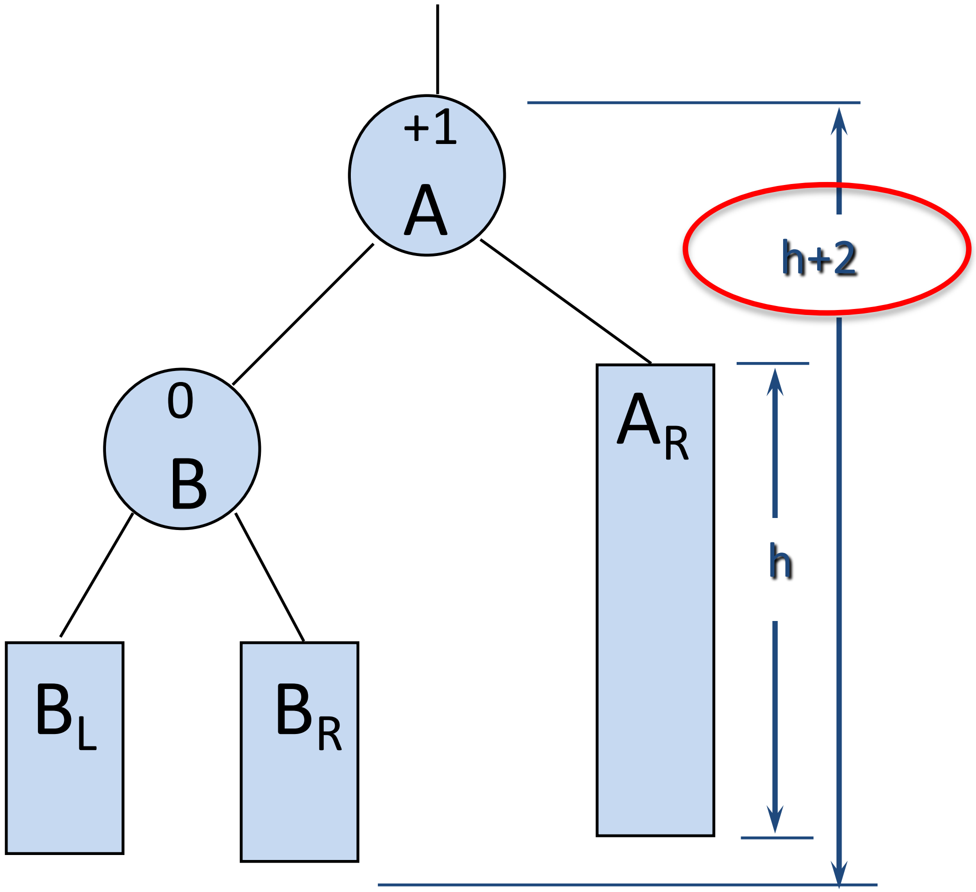
  - RR: the path from A to Y
  - Right subtree then Right subtree

- RL: Y is inserted in the
  Left subtree of the Right subtree of A

  - RL: the path from A to Y
  - Right subtree then Left subtree

AVL Trees

Balanced Subtree

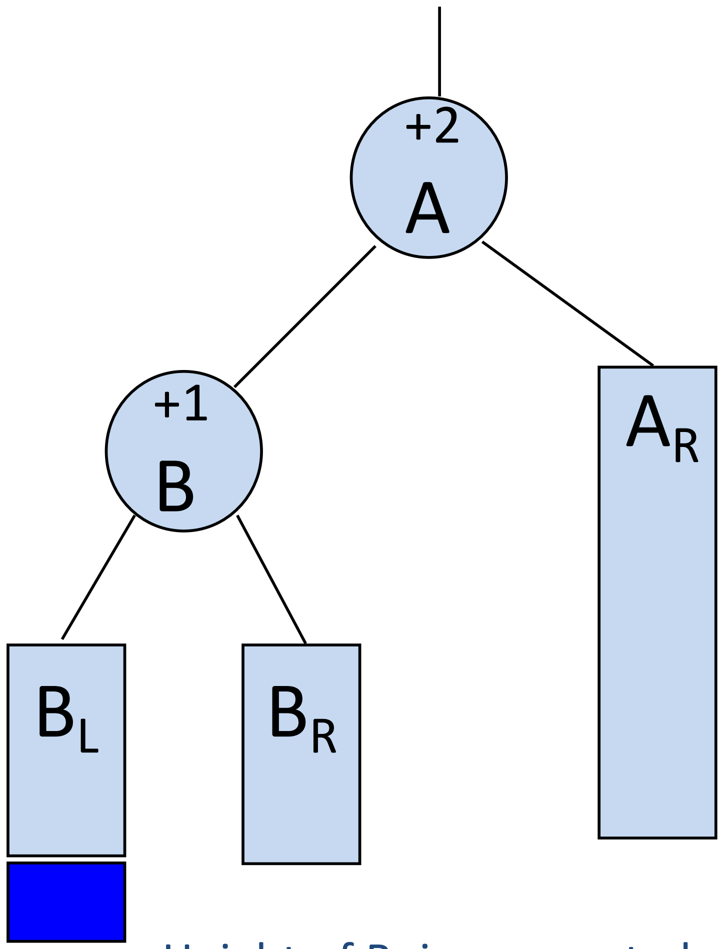


Why?

Hint:
balance factor
of A is +1

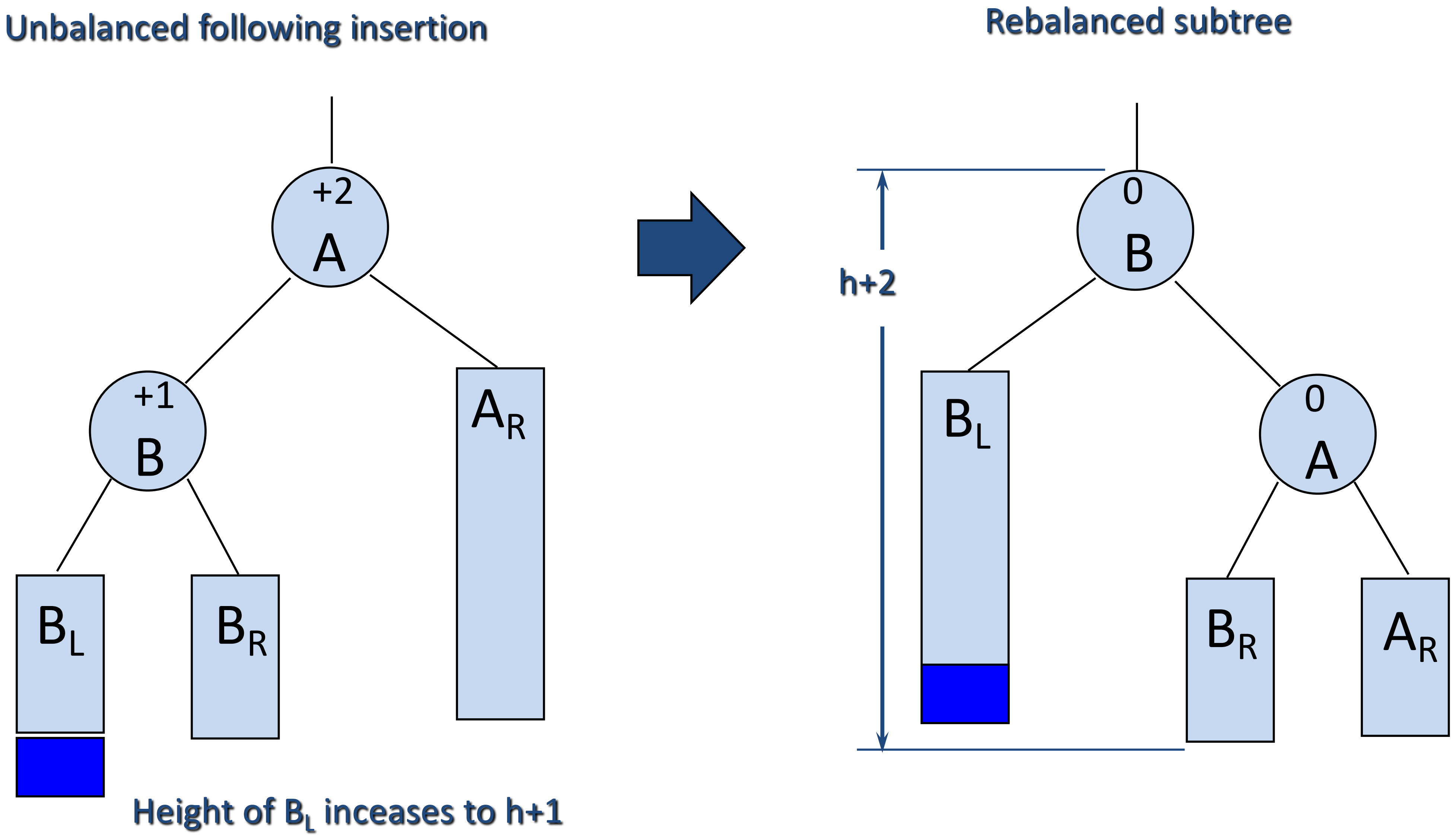
AVL Trees

Unbalanced following insertion



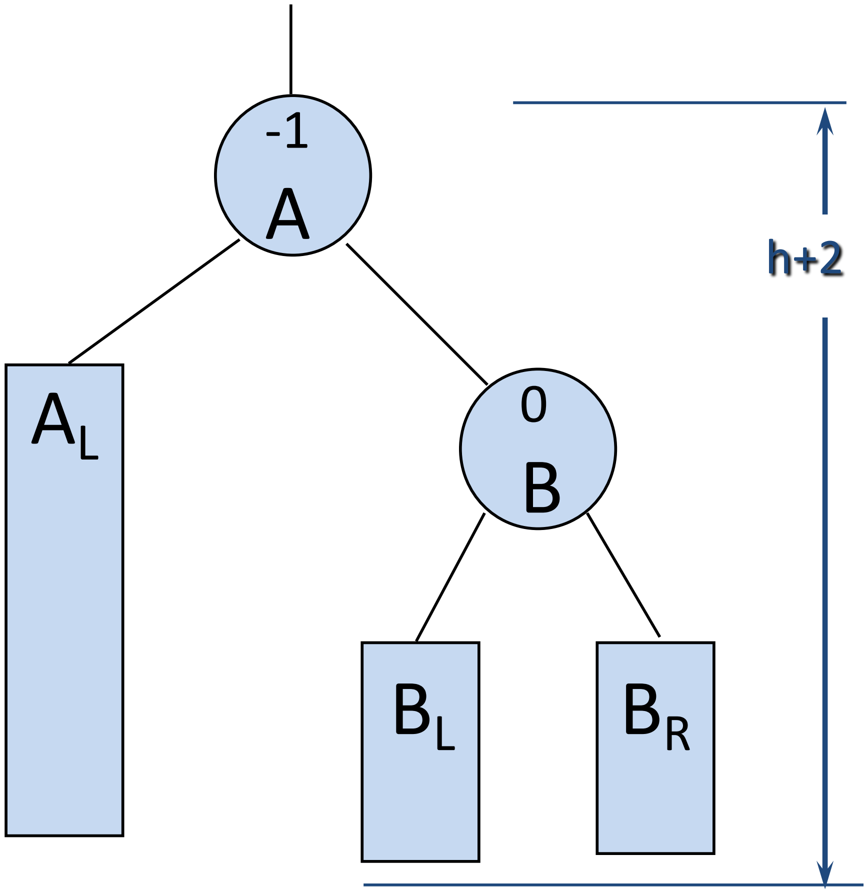
Height of B_L increases to h+1

AVL Trees - LL rotation



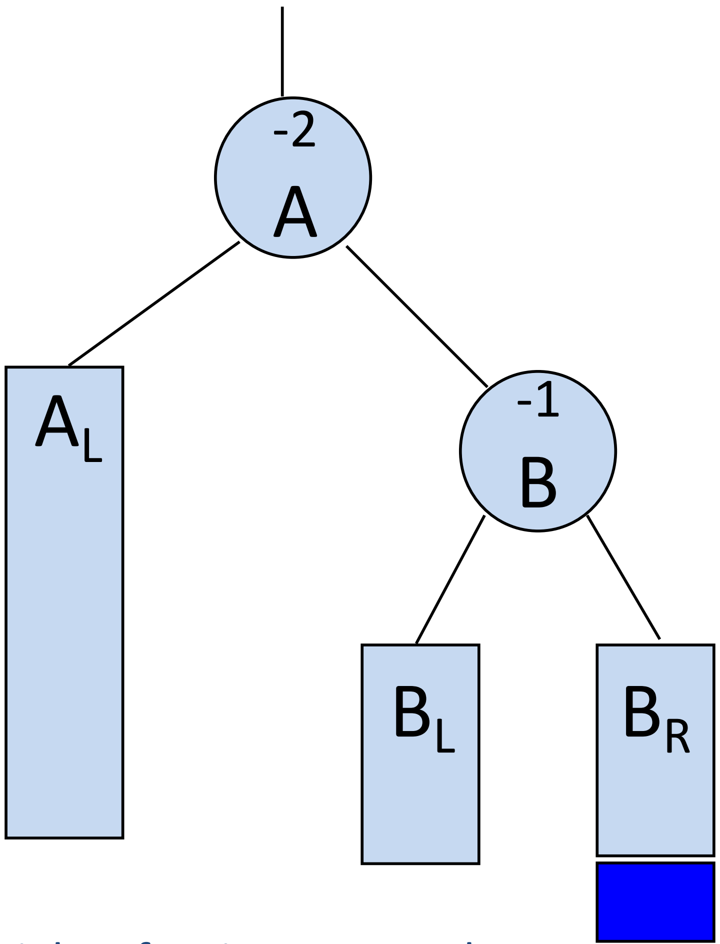
AVL Trees

Balanced Subtree



AVL Trees

Unbalanced following insertion

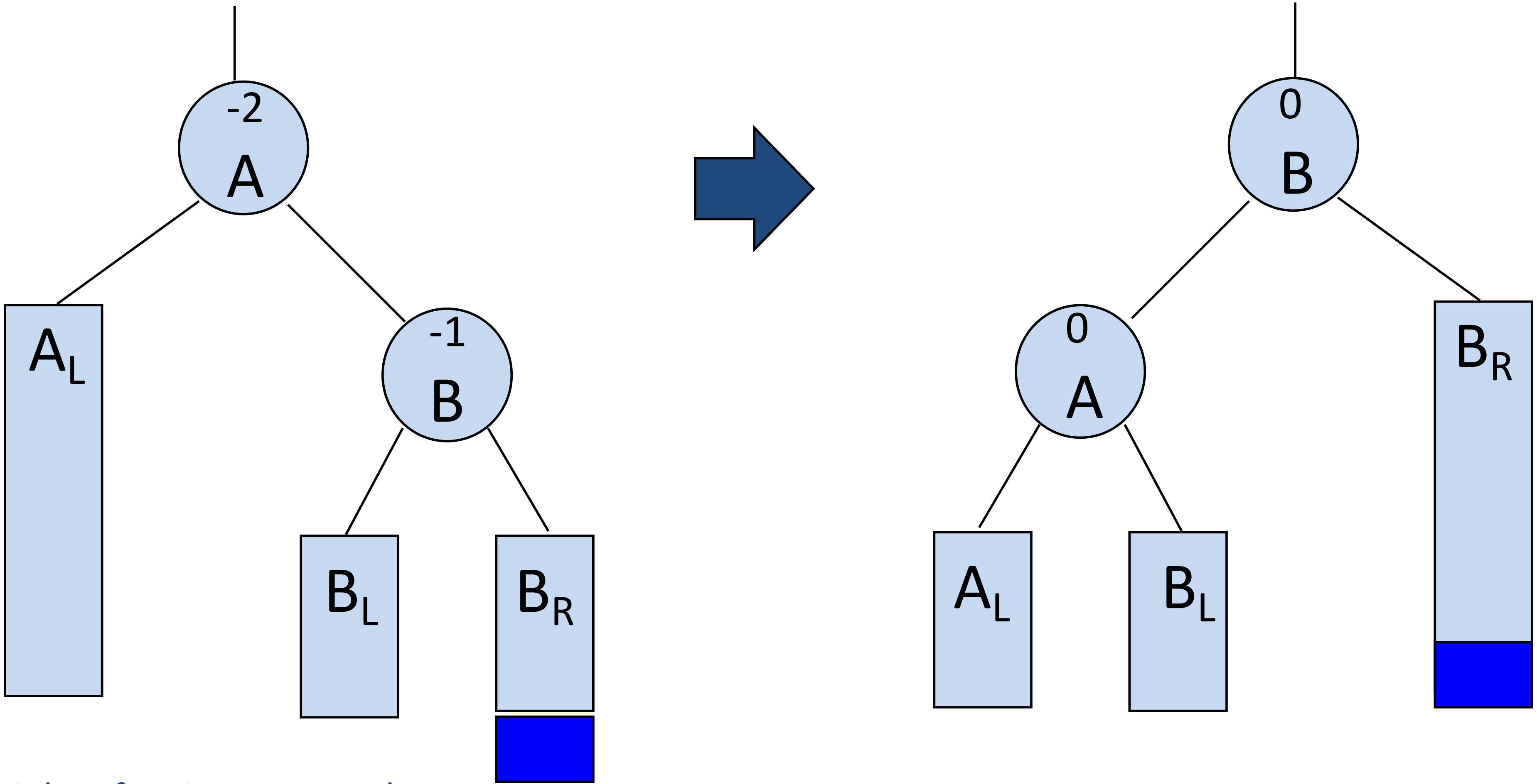


Height of B_R inceases to h+1

AVL Trees - RR Rotation

Unbalanced following insertion

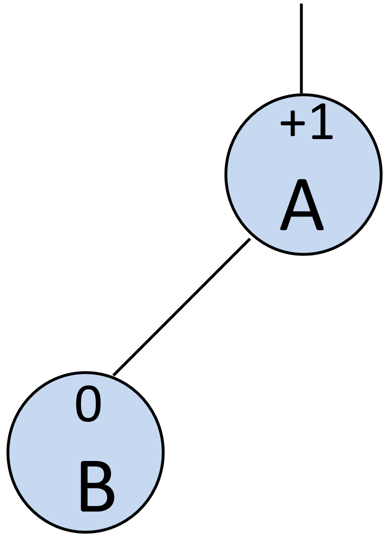
Rebalanced subtree



Height of B_R inceases to h+1

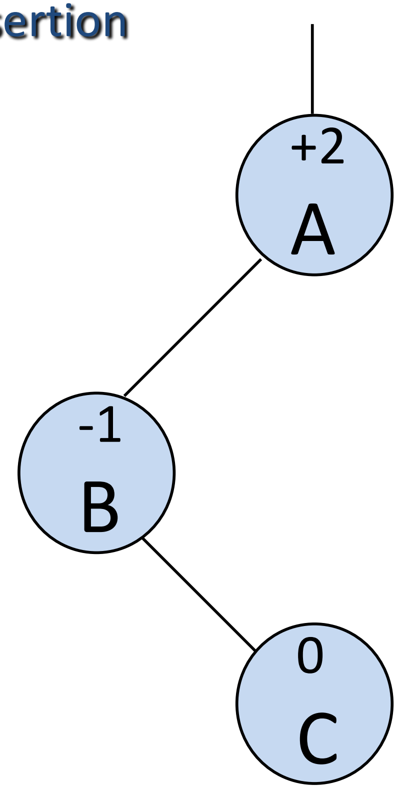
AVL Trees

Balanced Subtree

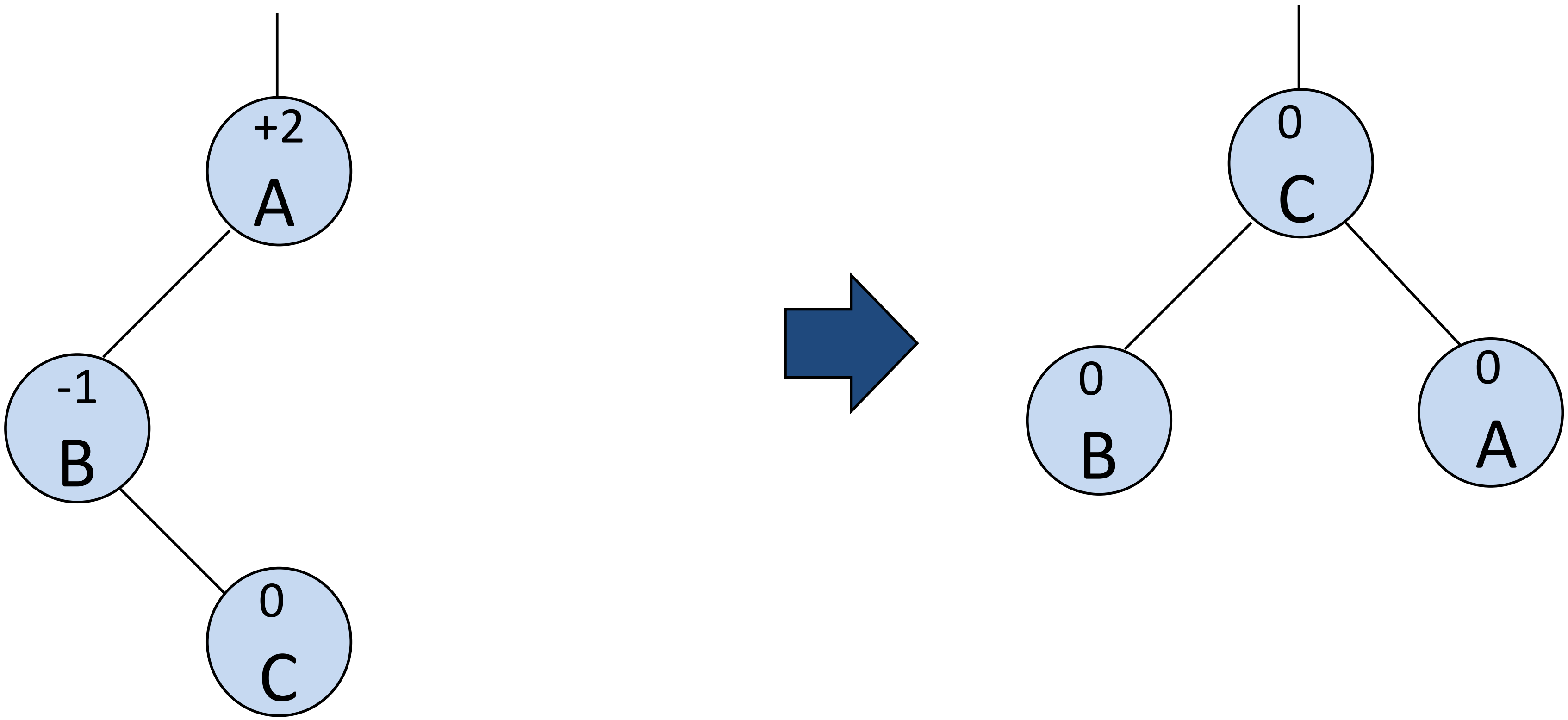


AVL Trees

Unbalanced following insertion

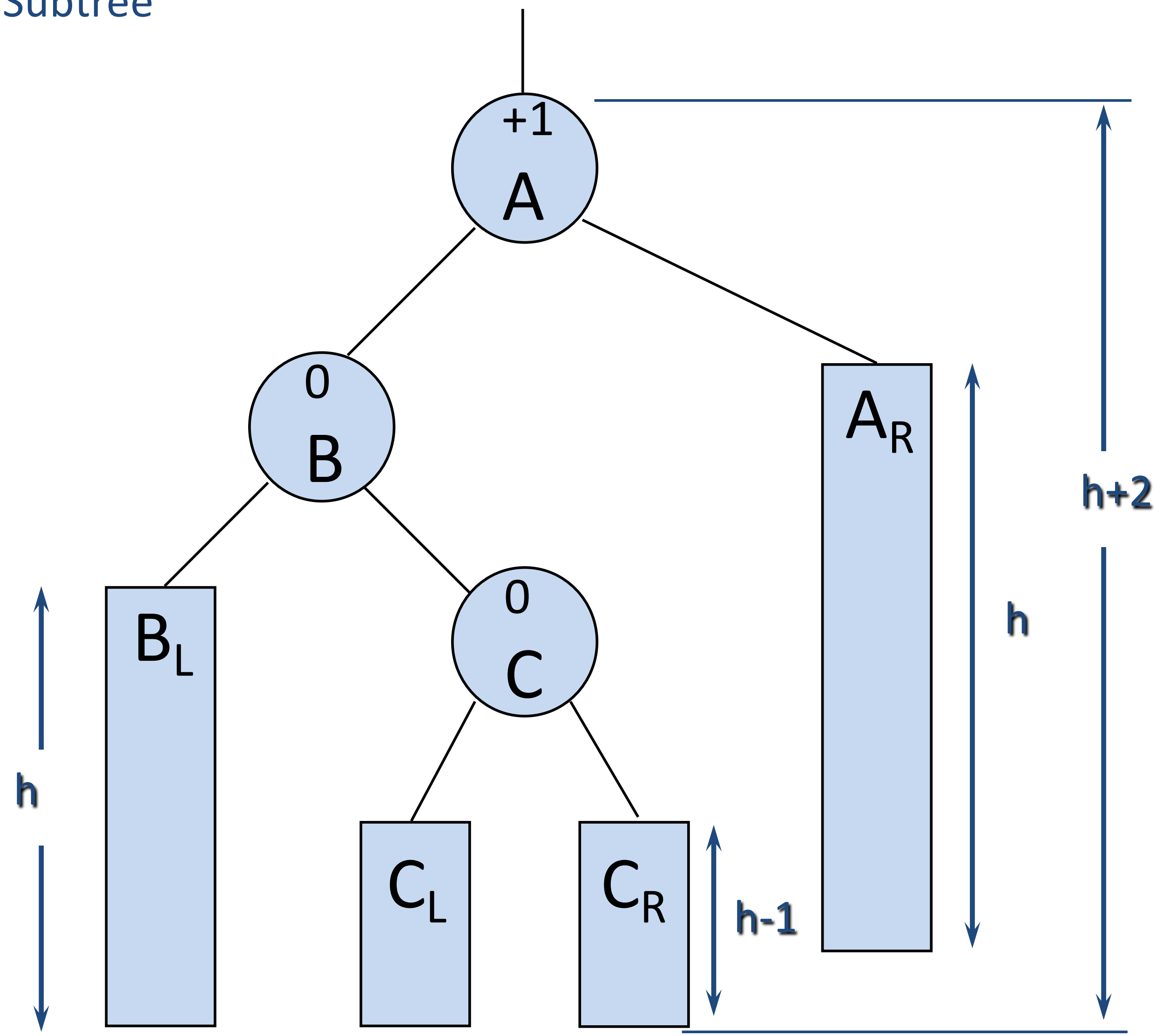


AVL Trees - LR rotation (a)



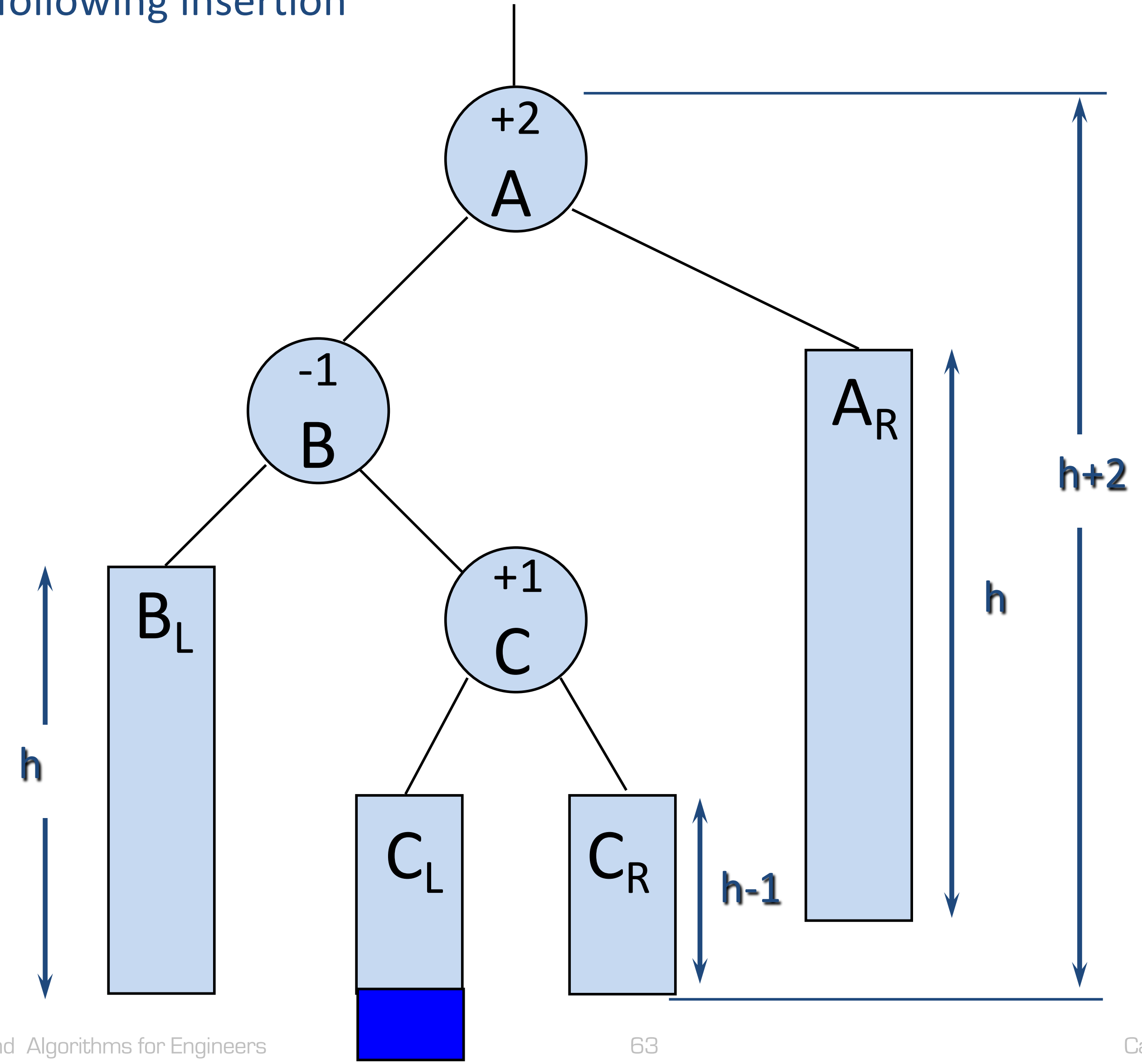
AVL Trees

Balanced Subtree

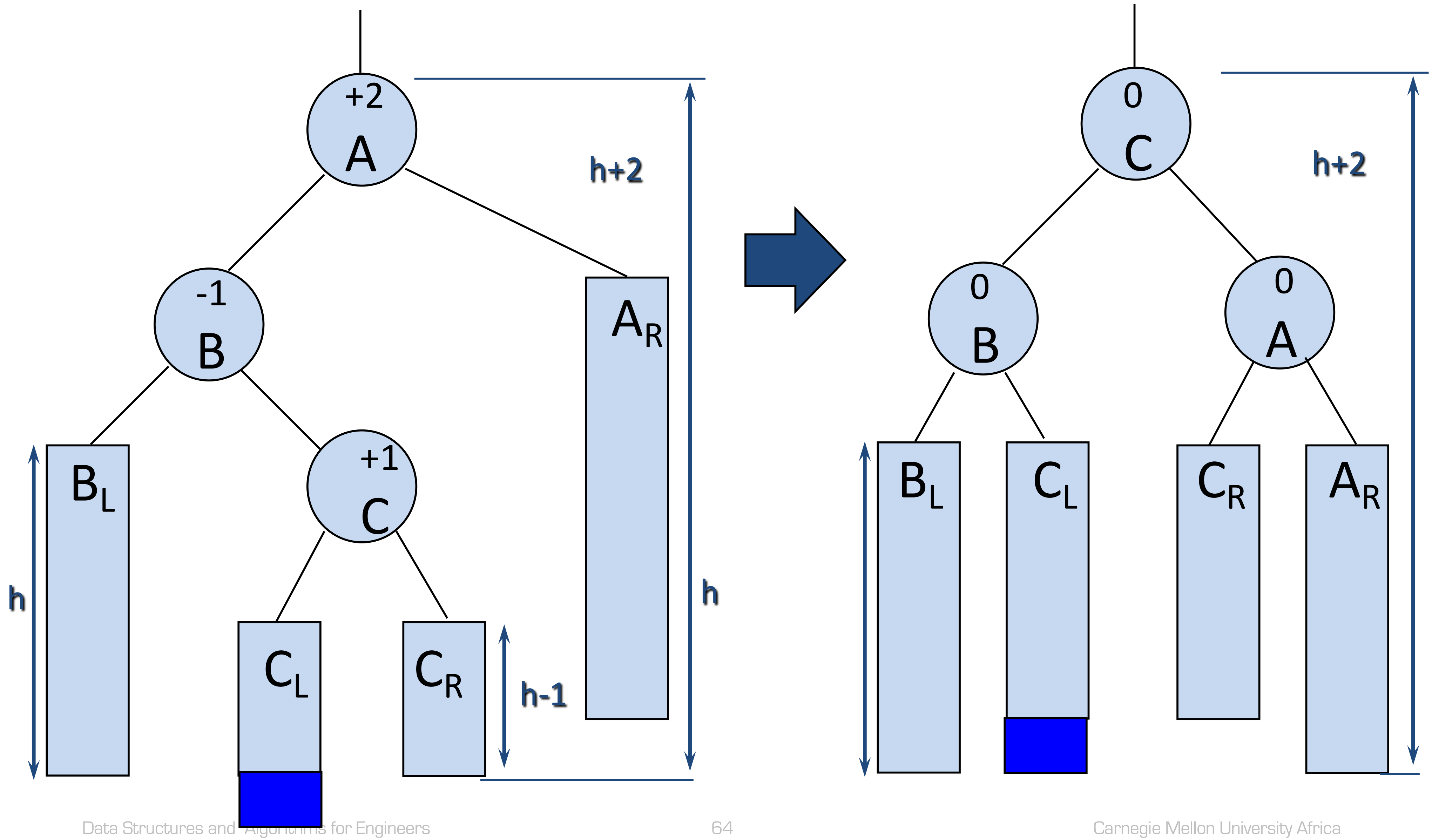


AVL Trees

Unbalanced following insertion

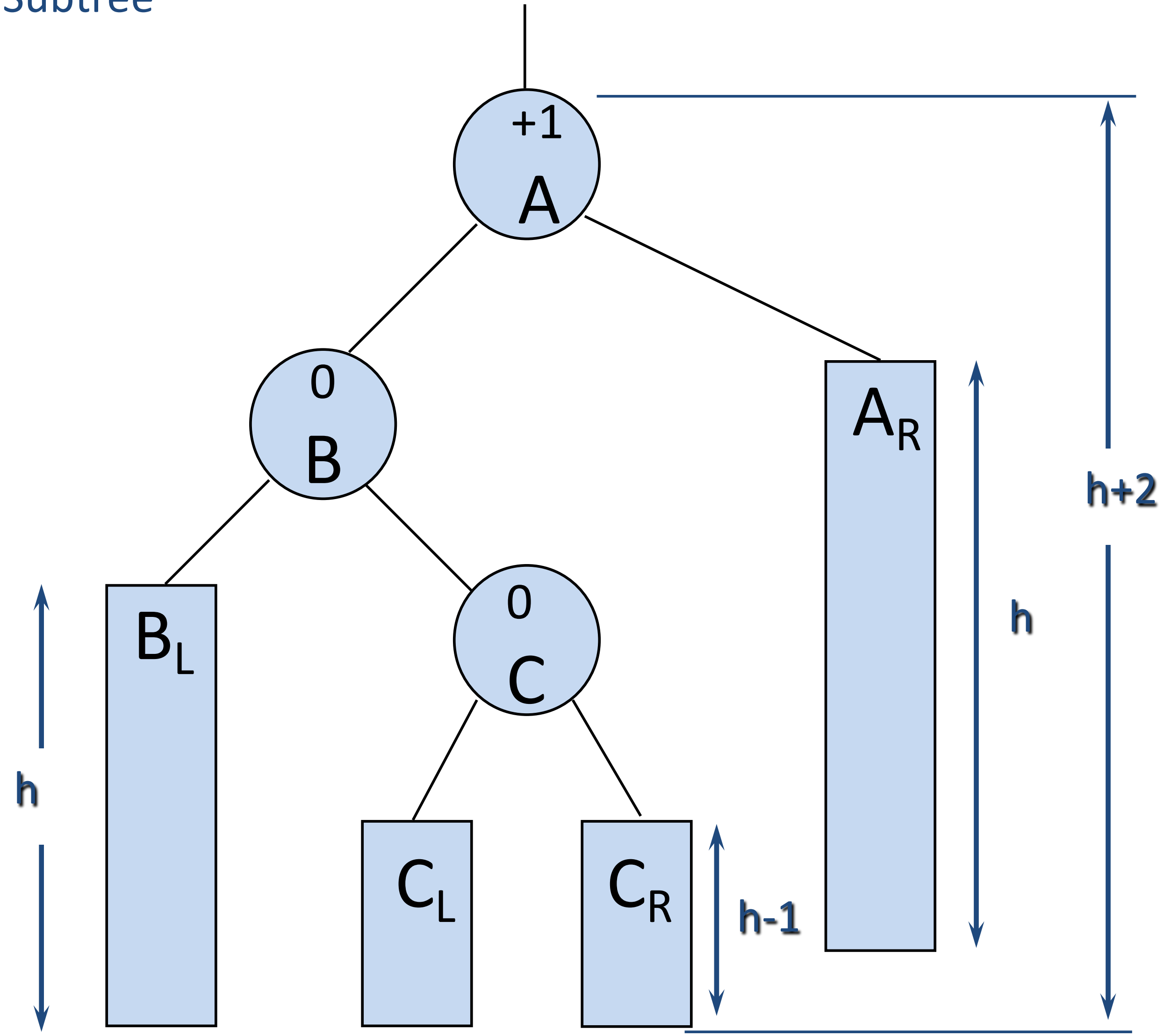


AVL Trees - LR rotation (b)



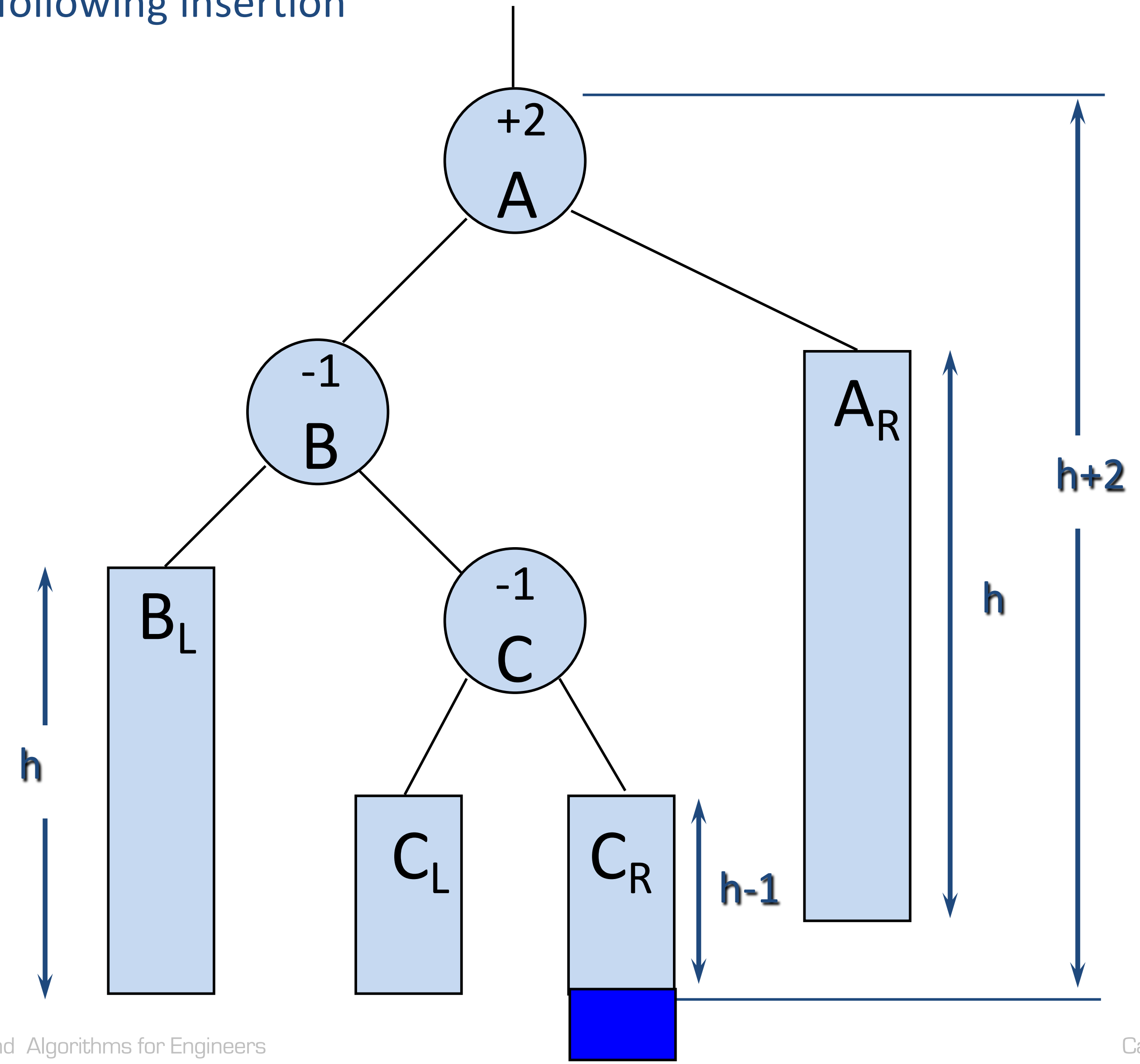
AVL Trees

Balanced Subtree

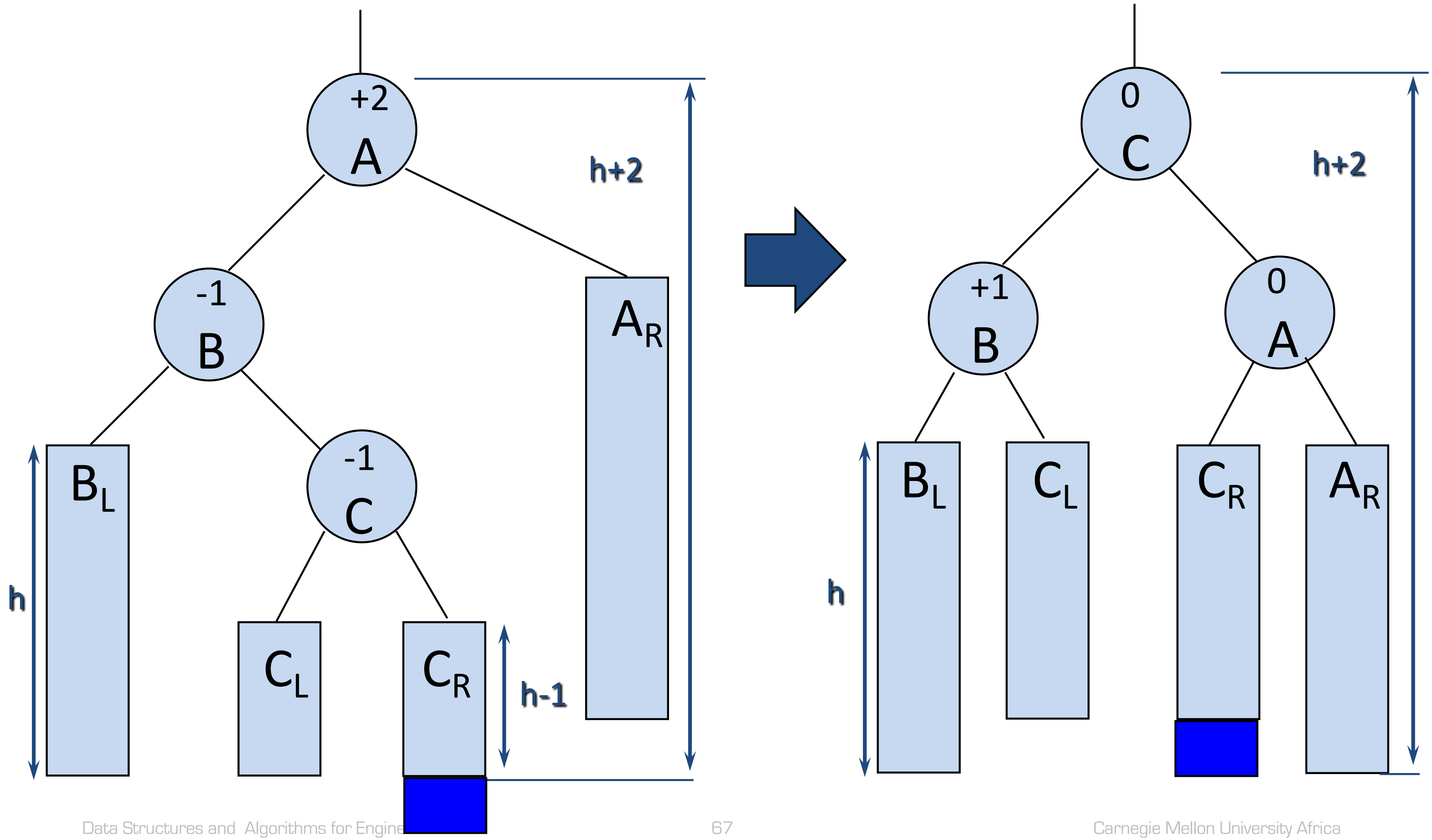


AVL Trees

Unbalanced following insertion

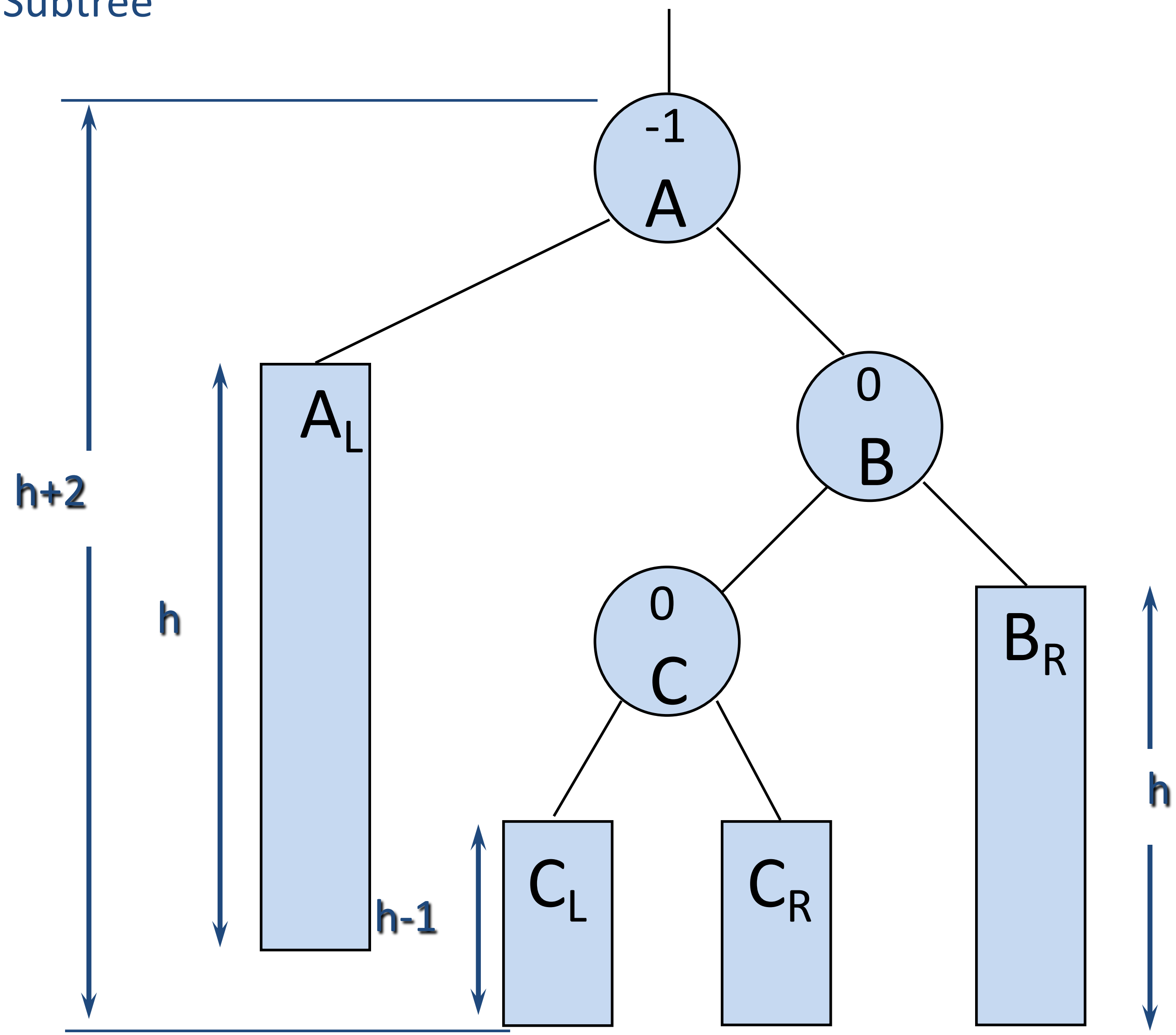


AVL Trees - LR rotation (c)



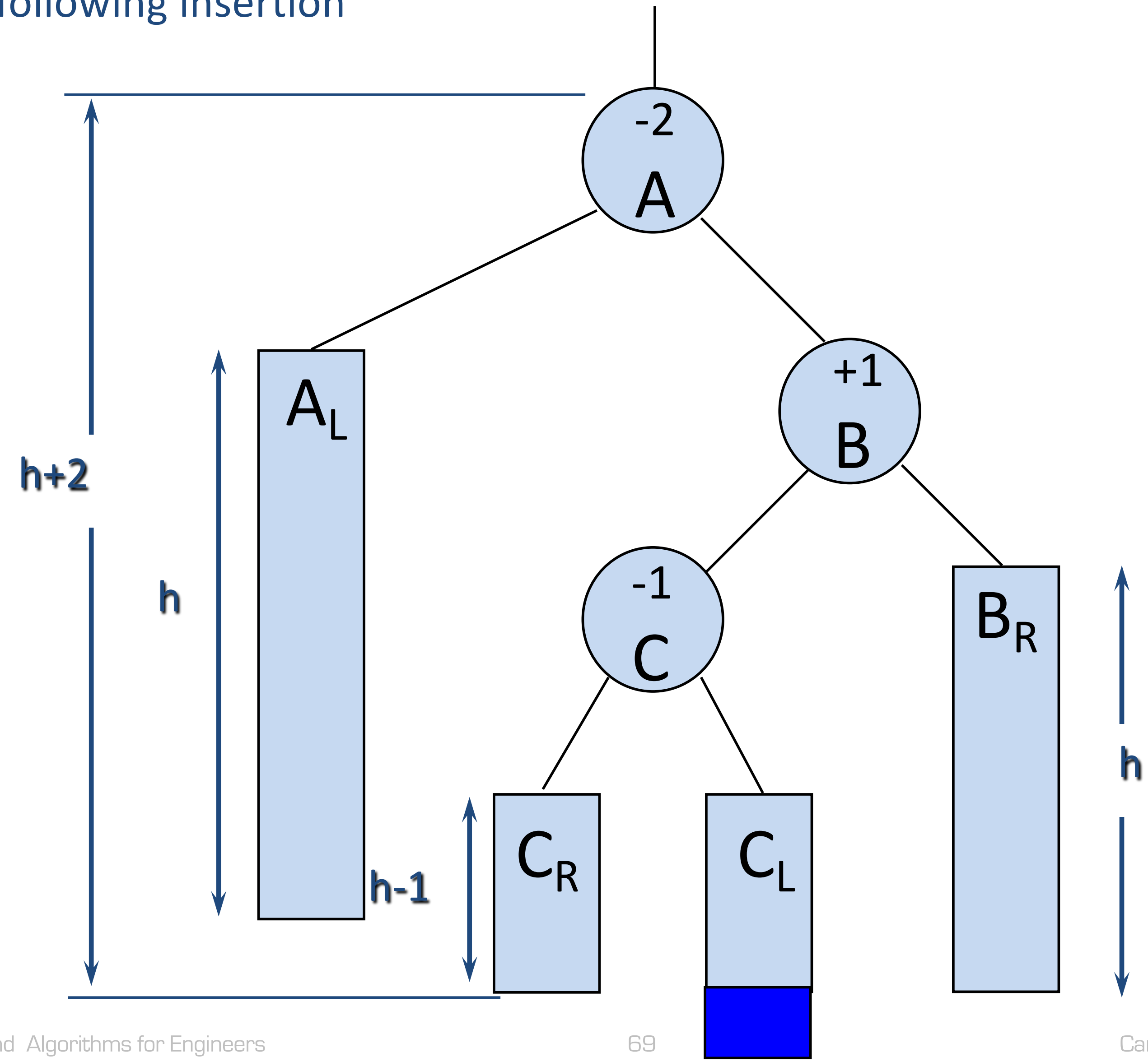
AVL Trees

Balanced Subtree

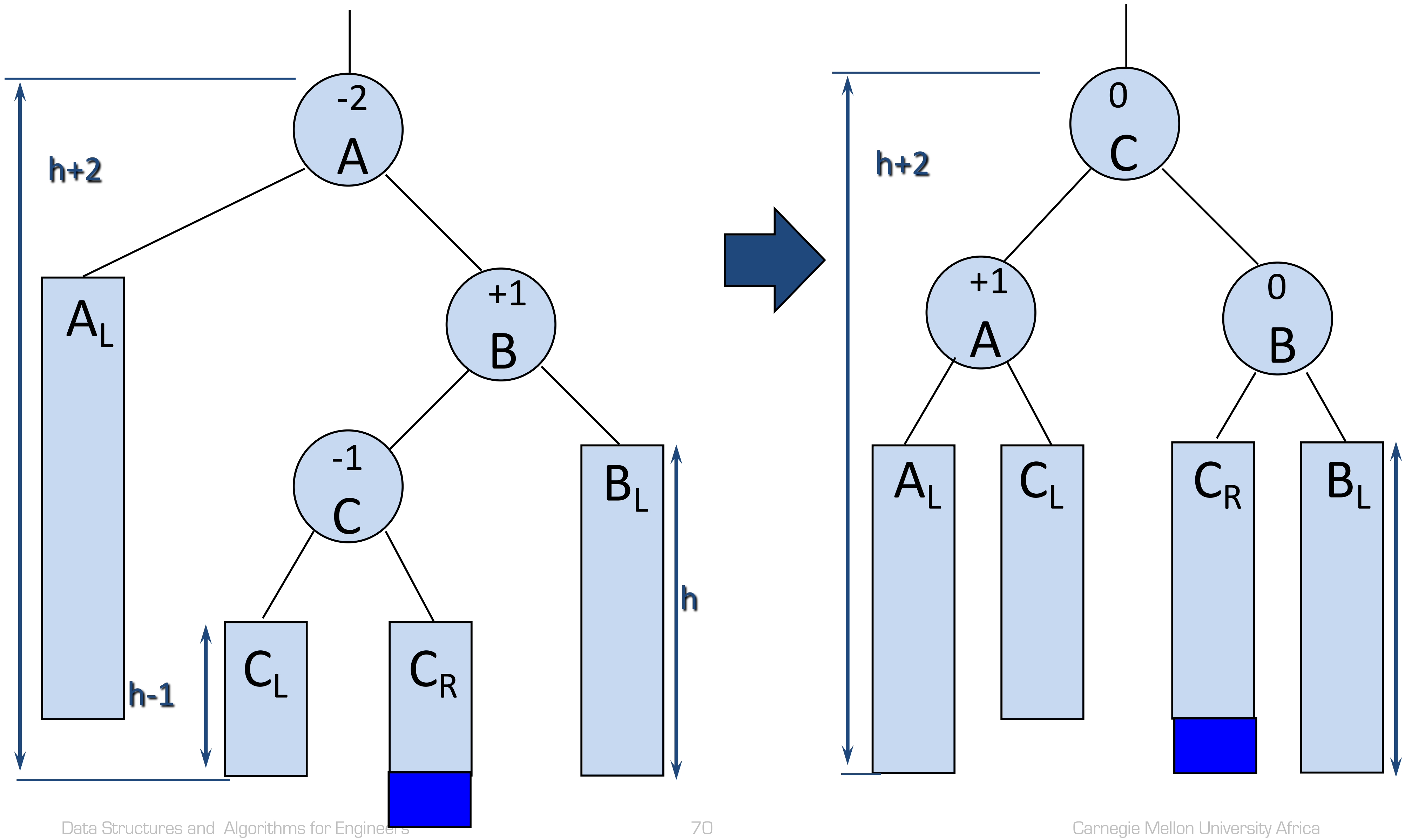


AVL Trees

Unbalanced following insertion



AVL Trees - RL rotation



AVL Trees

- To carry out this rebalancing we need to locate A, i.e. to window A
  - A is the nearest ancestor to Y whose balance factor becomes +2 or -2 following insertion
  - Equally, A is the nearest ancestor to Y whose balance factor was +1 or -1 before insertion
- We also need to locate F, the parent of A … (why?)