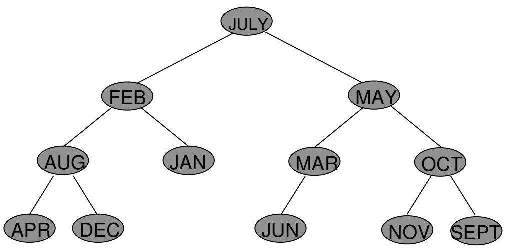
Height-Balanced Trees

AVL Trees

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, log2 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

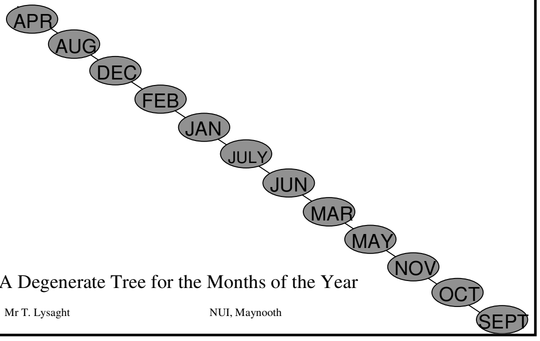


A Balanced Tree for the Months of the Year

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

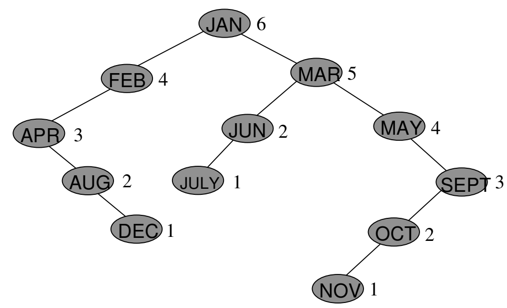
  $1 + max(height(T_1), height(T_2))$ otherwise, where T_1 and T_2 are the subtrees of the root.

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

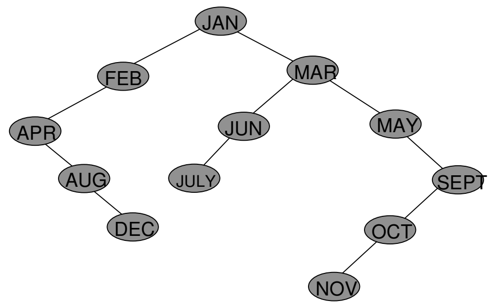
Recall: Binary Tree Terminology

- 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

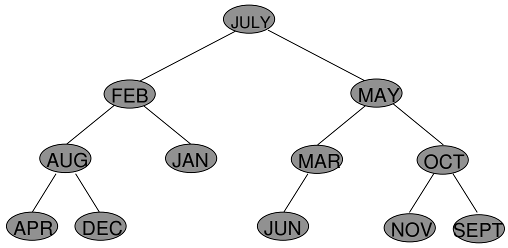
AVL Trees



AVL Trees

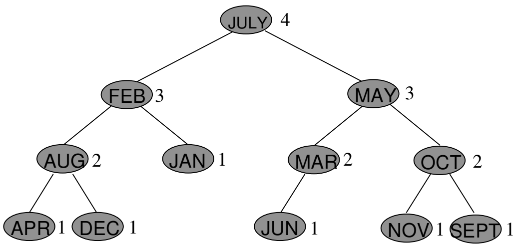


AVL Trees



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
-

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

New Identifier	After Insertion	After Rebalancing
MARCH		

New Identifier | After Insertion | After Rebalancing

MARCH


NO REBALANCING NEEDED

New Identifier | After Insertion | After Rebalancing

MARCH

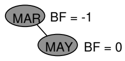

NO REBALANCING NEEDED

MAY


New Identifier | After Insertion | After Rebalancing

MARCH


NO REBALANCING NEEDED

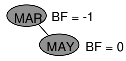
MAY


NO REBALANCING NEEDED

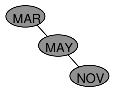
New Identifier | After Insertion | After Rebalancing

MARCH


NO REBALANCING NEEDED

MAY


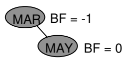
NO REBALANCING NEEDED

NOVEMBER


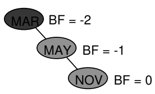
New Identifier | After Insertion | After Rebalancing

MARCH


NO REBALANCING NEEDED

MAY


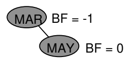
NO REBALANCING NEEDED

NOVEMBER


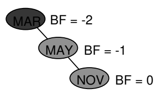
New Identifier | After Insertion | After Rebalancing

MARCH


NO REBALANCING NEEDED

MAY


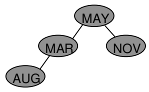
NO REBALANCING NEEDED

NOVEMBER


RR rebalancing

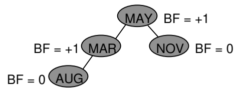
New Identifier | After Insertion | After Rebalancing

AUGUST



New Identifier | After Insertion | After Rebalancing

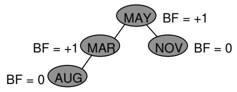
AUGUST



NO REBALANCING NEEDED

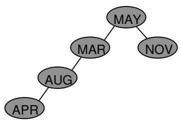
New Identifier | After Insertion | After Rebalancing

AUGUST



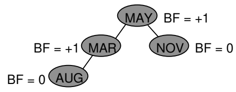
NO REBALANCING NEEDED

APRIL



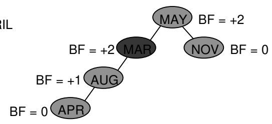
New Identifier | After Insertion | After Rebalancing

AUGUST



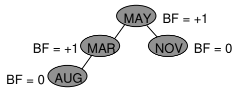
NO REBALANCING NEEDED

APRIL



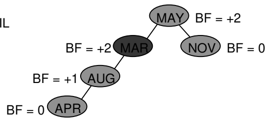
New Identifier | After Insertion | After Rebalancing

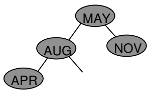
AUGUST



NO REBALANCING NEEDED

APRIL

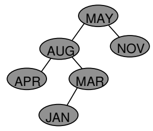


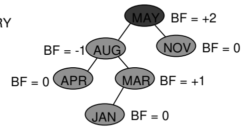


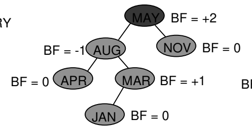
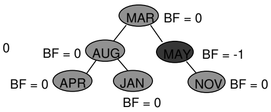
LL rebalancing

New Identifier | After Insertion | After Rebalancing

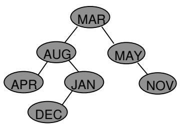
JANUARY

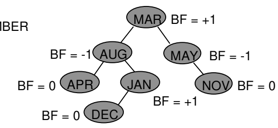


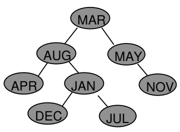
New Identifier	After Insertion	After Rebalancing
JANUARY	

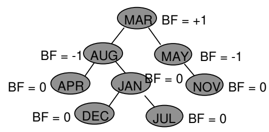
New Identifier	After Insertion	After Rebalancing
JANUARY		

LR rebalancing

New Identifier	After Insertion	After Rebalancing
DECEMBER	

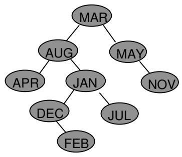
New Identifier	After Insertion	After Rebalancing
DECEMBER		NO REBALANCING NEEDED

New Identifier	After Insertion	After Rebalancing
JULY	

New Identifier	After Insertion	After Rebalancing
JULY		NO REBALANCING NEEDED

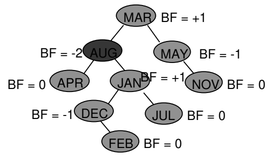
New Identifier	After Insertion	After Rebalancing

FEBRUARY



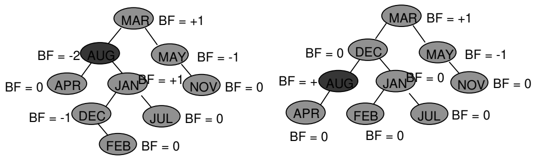
New Identifier	After Insertion	After Rebalancing

FEBRUARY



New Identifier	After Insertion	After Rebalancing

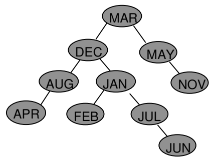
FEBRUARY



RL rebalancing

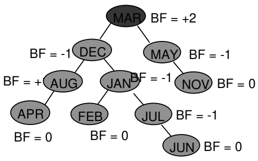
New Identifier	After Insertion	After Rebalancing

JUNE



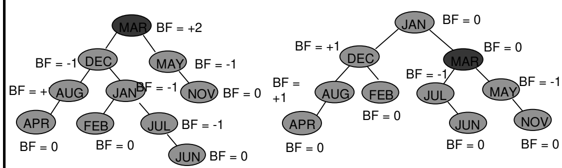
New Identifier	After Insertion	After Rebalancing

JUNE



New Identifier	After Insertion	After Rebalancing

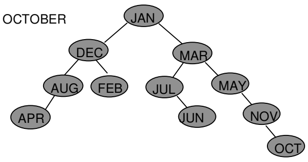
JUNE



LR rebalancing

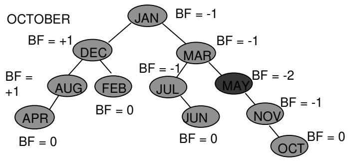
New Identifier | After Insertion | After Rebalancing

OCTOBER



New Identifier | After Insertion | After Rebalancing

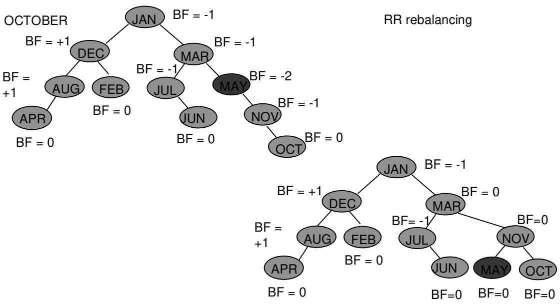
OCTOBER



New Identifier | After Insertion | After Rebalancing

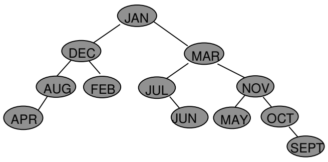
OCTOBER

RR rebalancing



New Identifier | After Insertion | After Rebalancing

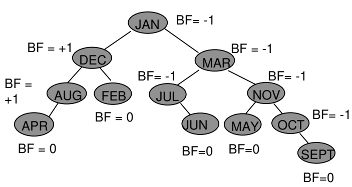
SEPTEMBER



New Identifier | After Insertion | After Rebalancing

SEPTEMBER

NO REBALANCING NEEDED



AVL Trees

- 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)

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

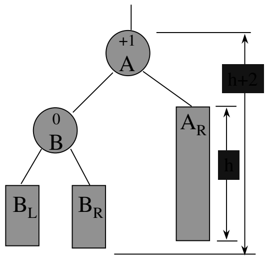
- LL: Y is inserted in the
- LR: Y is inserted in the

AVL Trees

- RR: Y is inserted in the
- RL: Y is inserted in the

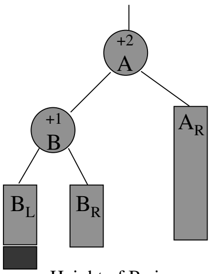
AVL Trees

Balanced Subtree



AVL Trees

Unbalanced following insertion

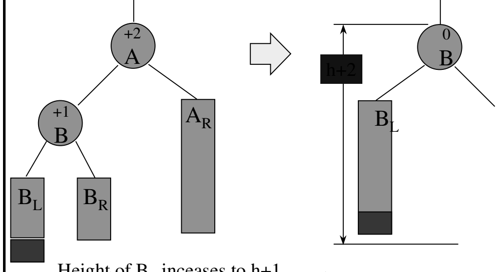


Height of B_L inceases to h+1

AVL Trees - LL (single) rotation

Unbalanced following insertion

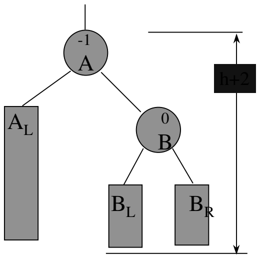
Rebalanced subtree



Height of B_L inceases to h+1

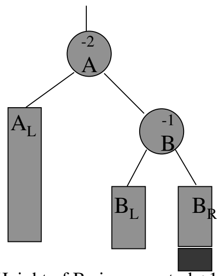
AVL Trees

Balanced Subtree



AVL Trees

Unbalanced following insertion

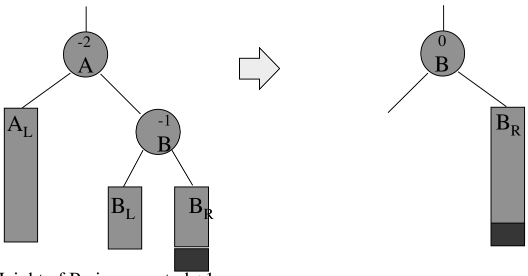


Height of B_R inceases to h+1

AVL Trees - RR Rotation

Unbalanced following insertion

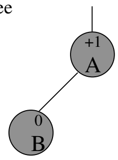
Rebalanced subtree



Height of B_R inceases to h+1

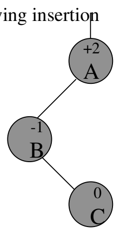
AVL Trees

Balanced Subtree

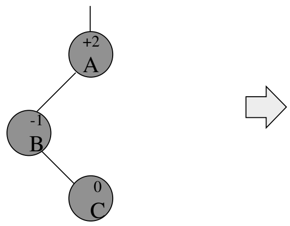


AVL Trees

Unbalanced following insertion

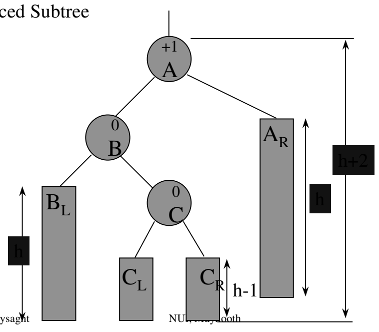


AVL Trees - LR rotation (a)



AVL Trees

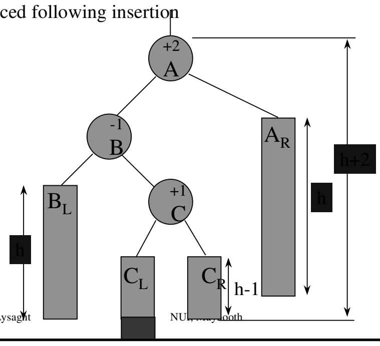
Balanced Subtree



Mr T. Lysaght NUI, Maynooth

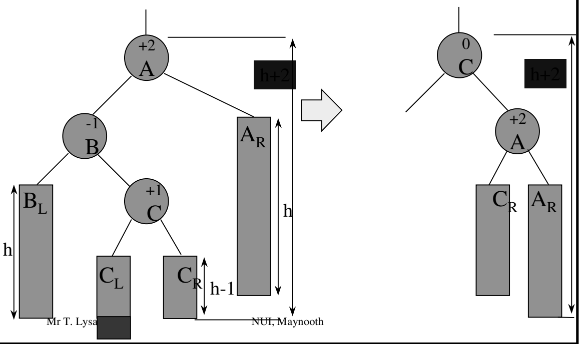
AVL Trees

Unbalanced following insertion



Mr T. Lysaght NUI, Maynooth

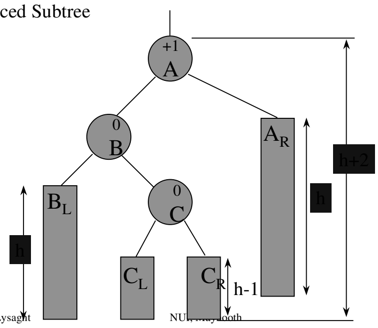
AVL Trees - LR rotation (b)



Mr T. Lysaght NUI, Maynooth

AVL Trees

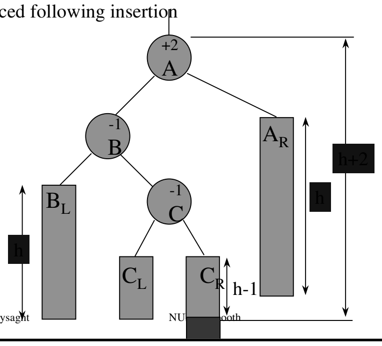
Balanced Subtree



Mr T. Lysaght NUI, Maynooth

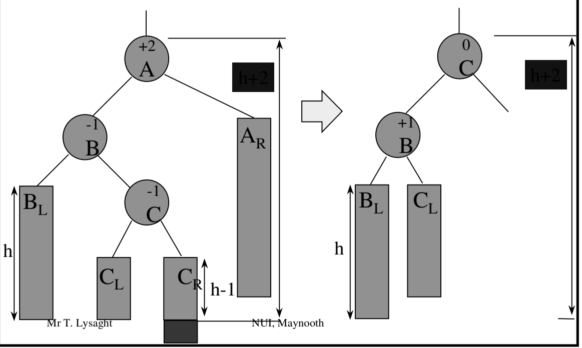
AVL Trees

Unbalanced following insertion



Mr T. Lysaght NUI, Maynooth

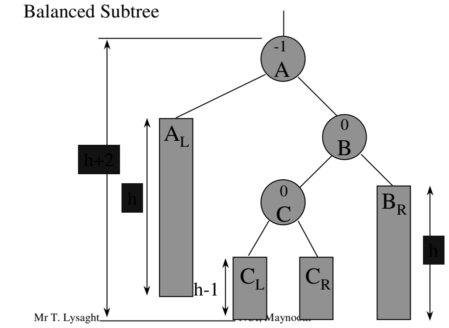
AVL Trees - LR rotation (c)



Mr T. Lysaght NUI, Maynooth

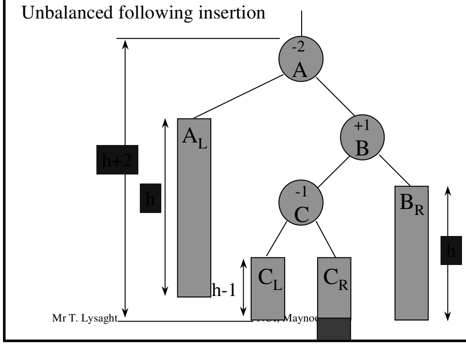
AVL Trees

Balanced Subtree

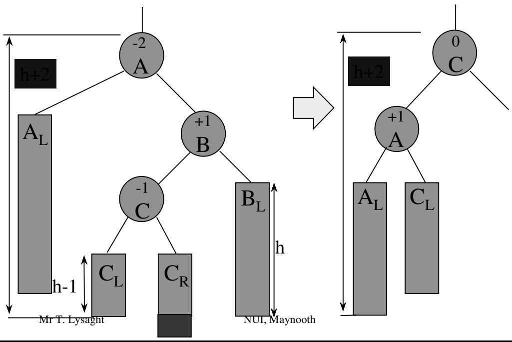


AVL Trees

Unbalanced following insertion



AVL Trees - RL rotation



AVL Trees

- To carry out this rebalancing we need to locate A, i.e. to window A
- We also need to locate F, the parent of A
  - This is where our complex window variable is of use to us ... it gives us this automatically

AVL Trees

- Note in passing that, since A is the nearest ancestor to Y whose balance factor was +1 or -1 before insertion, the balance factor of all other nodes on the path from A to Y must be 0
- When we re-balance the tree, the balance factors change (see diagrams above)
  - But changes only occur in subtree which is being rebalanced

AVL Trees

- The balance factors also change following an insertion which requires no rebalancing
- BF(A) is +1 or -1 before insertion
- Insertion causes height of one of A's subtrees to increase by 1
- Thus, BF(A) must be 0 after insertion (since, in this case, it's not +2 or -2)