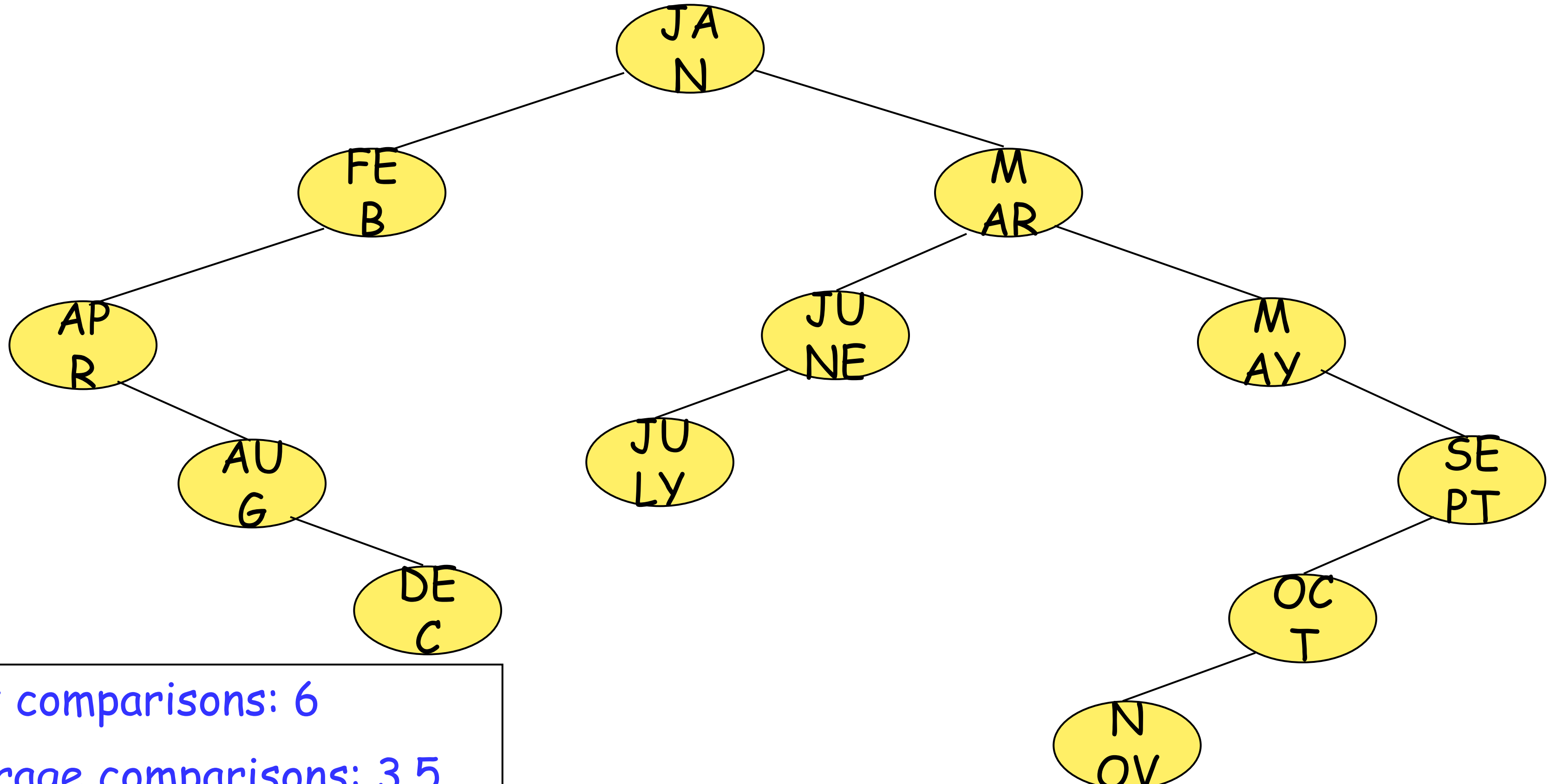
AVL Trees

- Dynamic tables may also be maintained as binary search trees.
- Depending on the order of the symbols putting into the table, the resulting binary search trees would be different. Thus the average comparisons for accessing a symbol is different.

Binary Search Tree for The Months of The Year

Input Sequence: JAN, FEB, MAR, APR, MAY, JUNE, JULY, AUG, SEPT, OCT, NOV, DEC



Max comparisons: 6

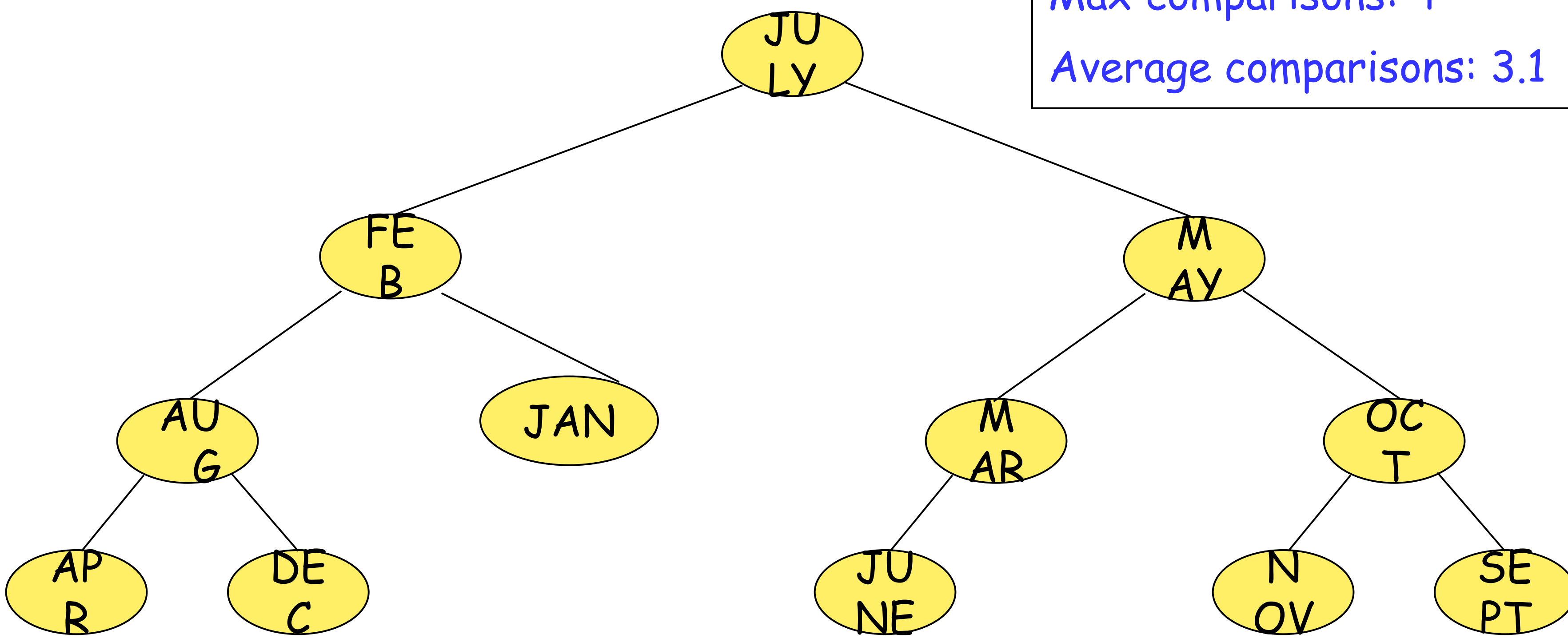
Average comparisons: 3.5

A Balanced Binary Search Tree For The Months of The Year

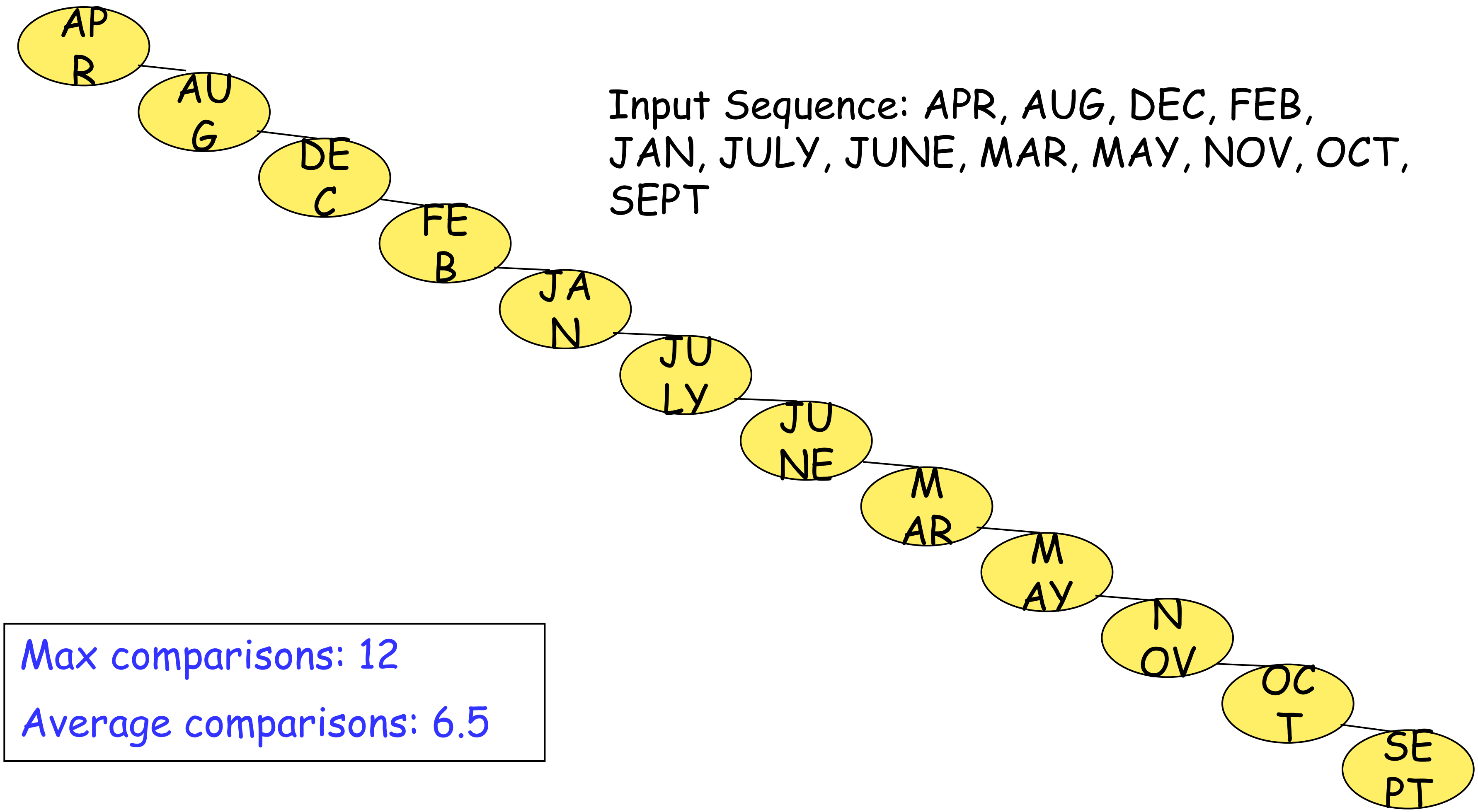
Input Sequence: JULY, FEB, MAY, AUG, DEC, MAR, OCT, APR, JAN, JUNE, SEPT, NOV

Max comparisons: 4

Average comparisons: 3.1



Degenerate Binary Search Tree



Input Sequence: APR, AUG, DEC, FEB, JAN, JULY, JUNE, MAR, MAY, NOV, OCT, SEPT

Max comparisons: 12

Average comparisons: 6.5

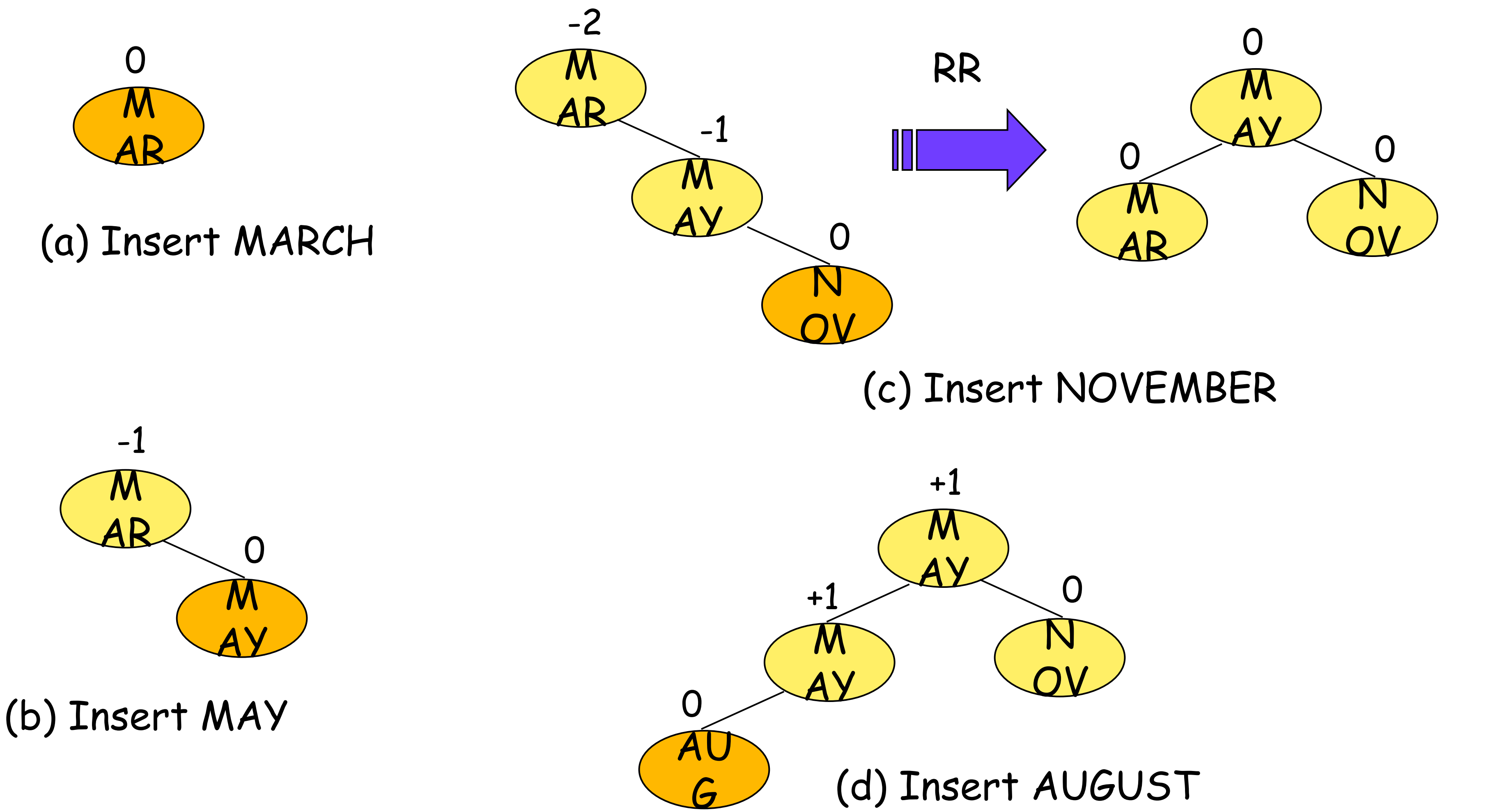
Minimize The Search Time of Binary Search Tree In Dynamic Situation

- From the above three examples, we know that the average and maximum search time will be minimized if the binary search tree is maintained as a complete binary search tree at all times.
- However, to achieve this in a dynamic situation, we have to pay a high price to restructure the tree to be a complete binary tree all the time.
- In 1962, Adelson-Velskii and Landis introduced a binary tree structure that is balanced with respect to the heights of subtrees. As a result of the balanced nature of this type of tree, dynamic retrievals can be performed in O(log n) time if the tree has n nodes. The resulting tree remains height-balanced. This is called an AVL tree.

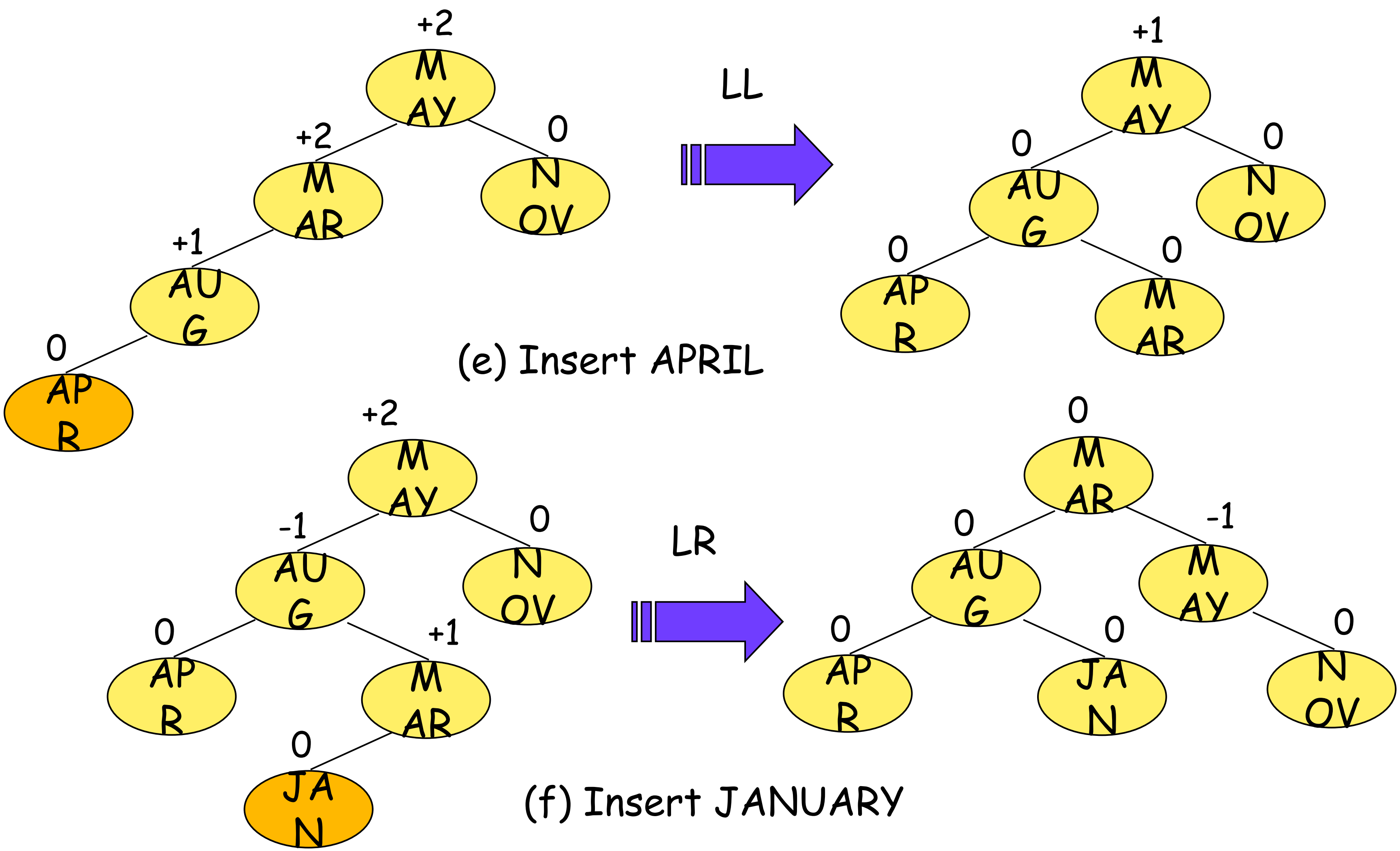
AVL Tree

- Definition: An empty tree is height-balanced. If T is a nonempty binary tree with T_L and T_R as its left and right subtrees respectively, then T is height-balanced iff
  (1) T_L and T_R are height-balanced, and
  (2) $|h_L - h_R| \leq 1$ where h_L and h_R are the heights of T_L and T_R, respectively.
- Definition: The Balance factor, $BF(T)$, of a node T is a binary tree is defined to be $h_L - h_R$, where h_L and h_R, respectively, are the heights of left and right subtrees of T. For any node T in an AVL tree, $BF(T)$ = -1, 0, or 1.

Balanced Trees Obtained for The Months of The Year



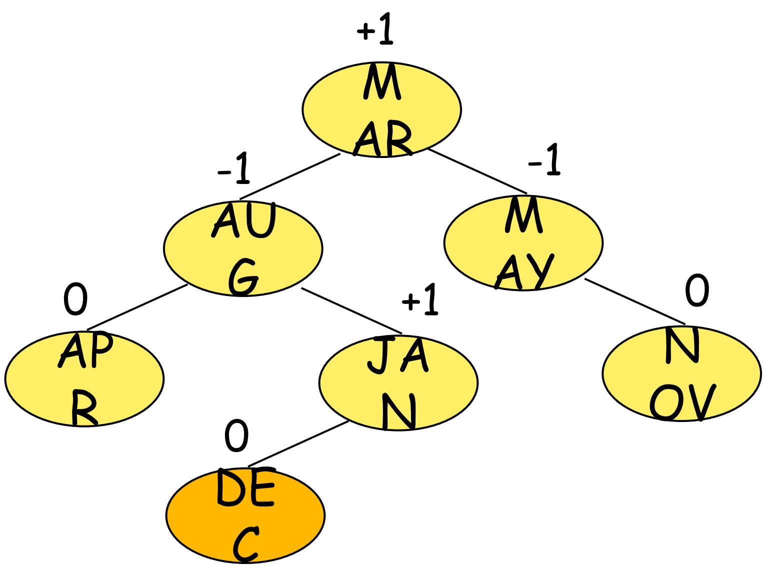
Balanced Trees Obtained for The Months of The Year (Cont.)



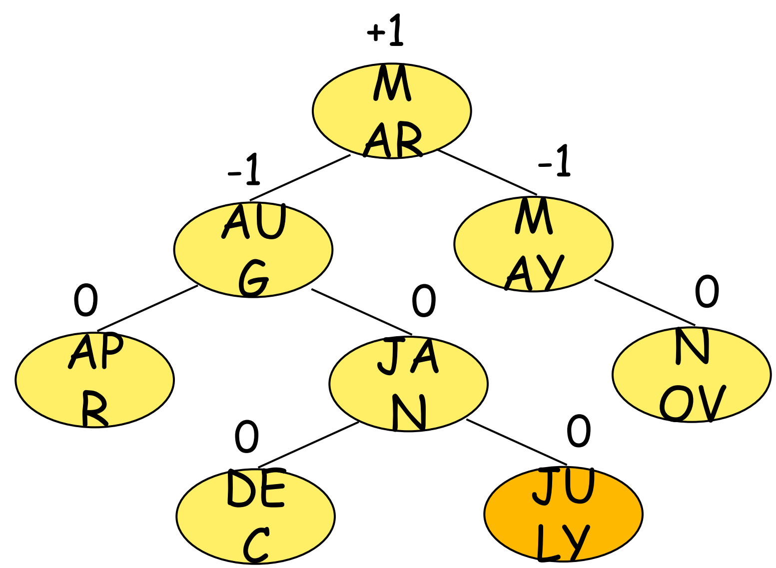
(e) Insert APRIL

(f) Insert JANUARY

Balanced Trees Obtained for The Months of The Year (Cont.)

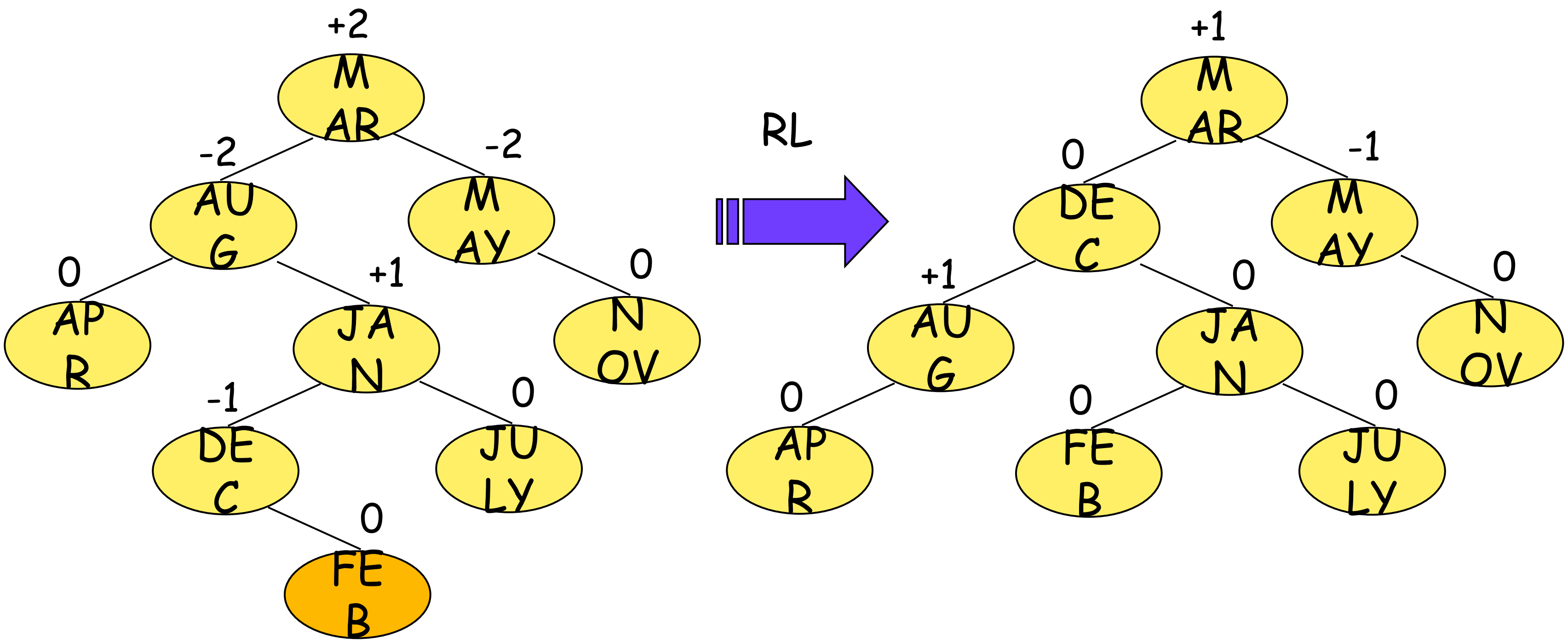


(g) Insert DECEMBER



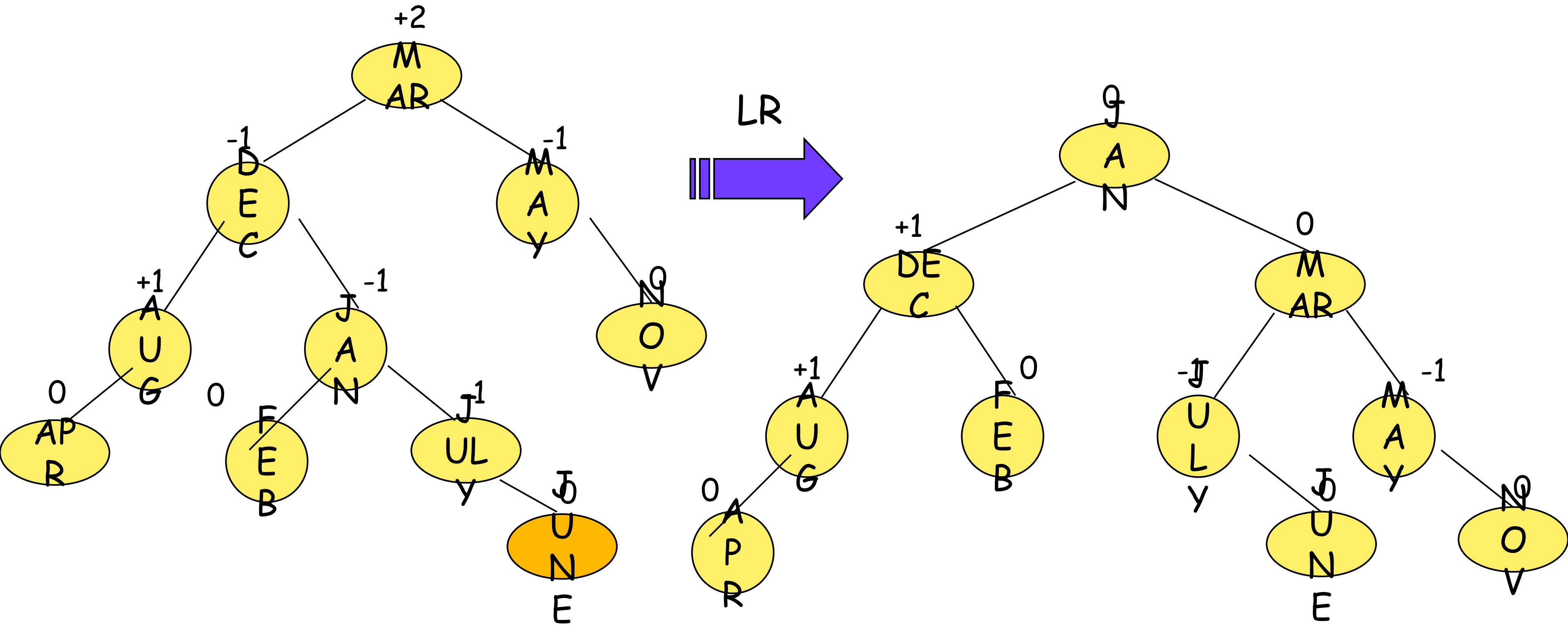
(h) Insert JULY

Balanced Trees Obtained for The Months of The Year (Cont.)



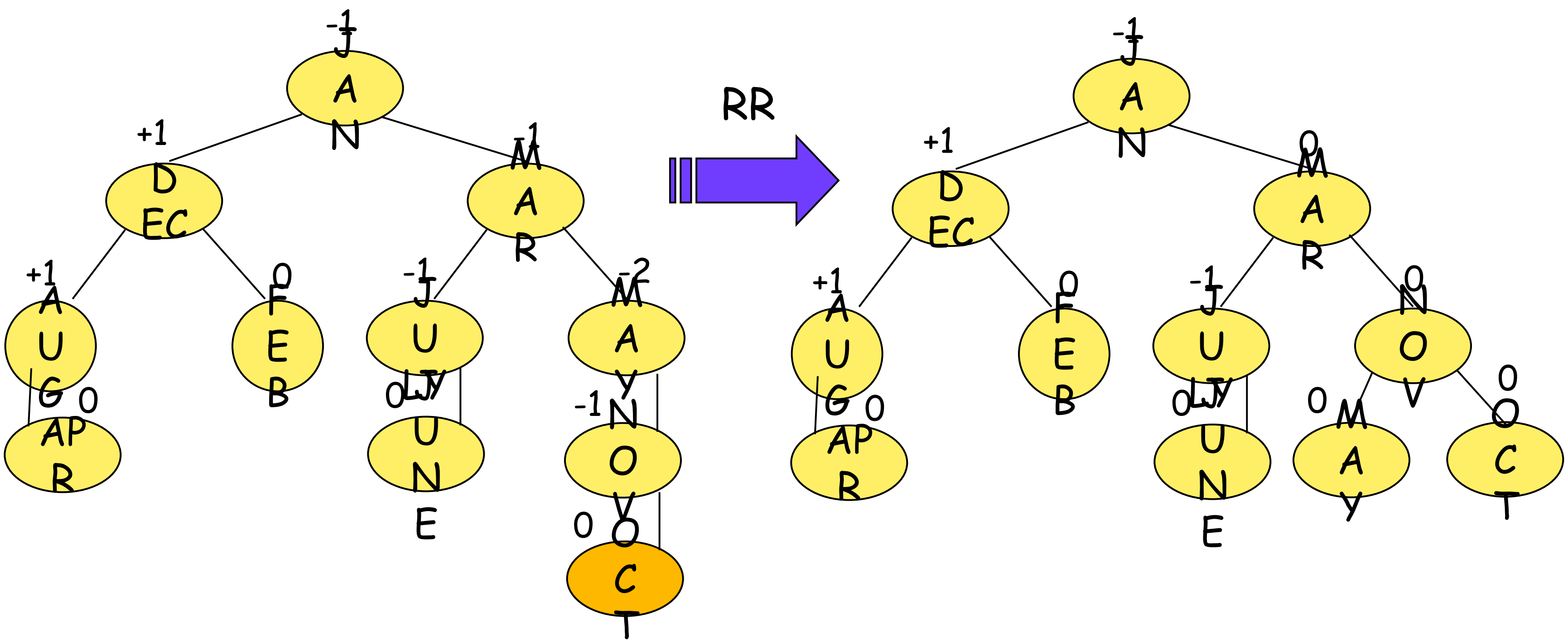
(i) Insert FEBRUARY

Balanced Trees Obtained for The Months of The Year (Cont.)



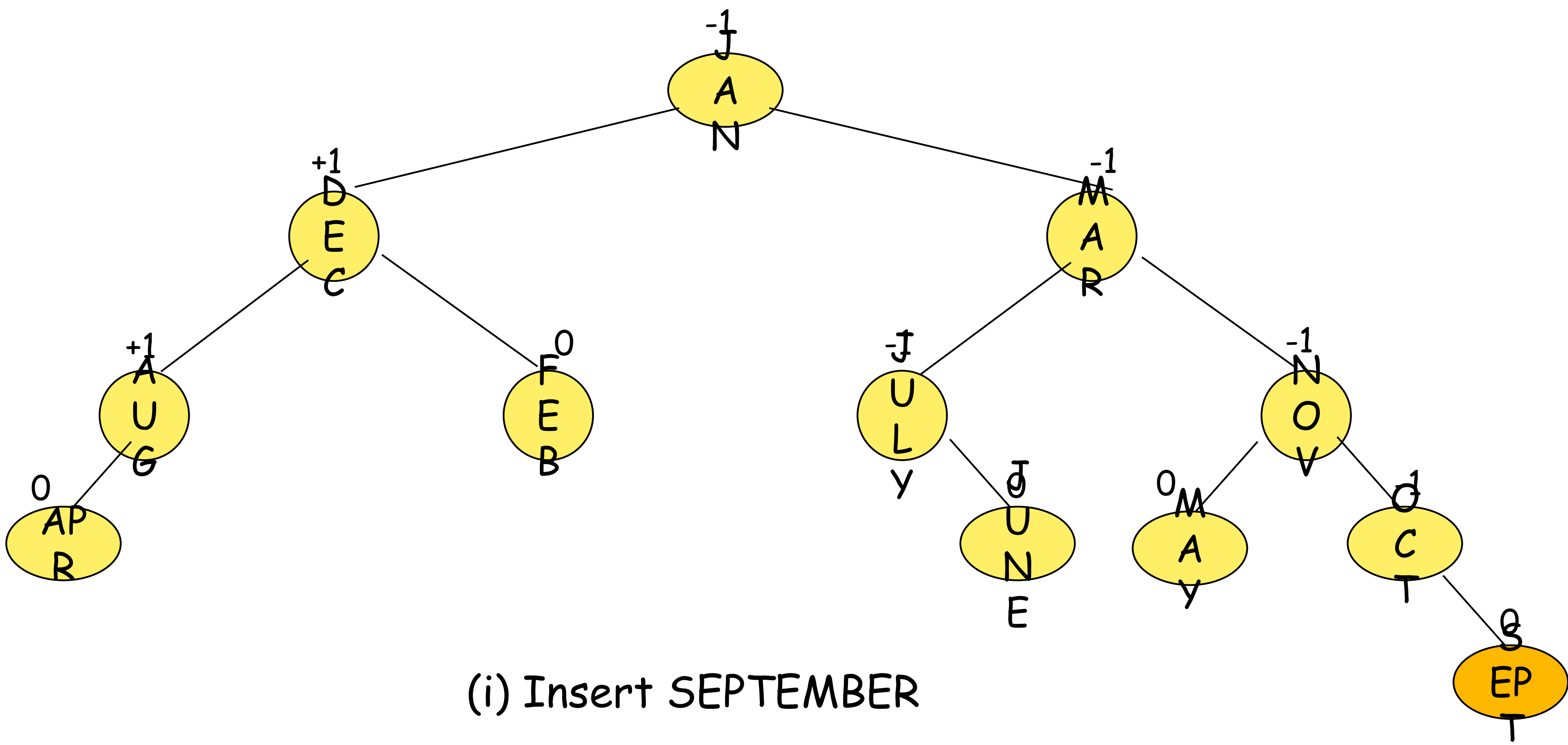
(j) Insert JUNE

Balanced Trees Obtained for The Months of The Year (Cont.)



(k) Insert OCTOBER

Balanced Trees Obtained for The Months of The Year (Cont.)

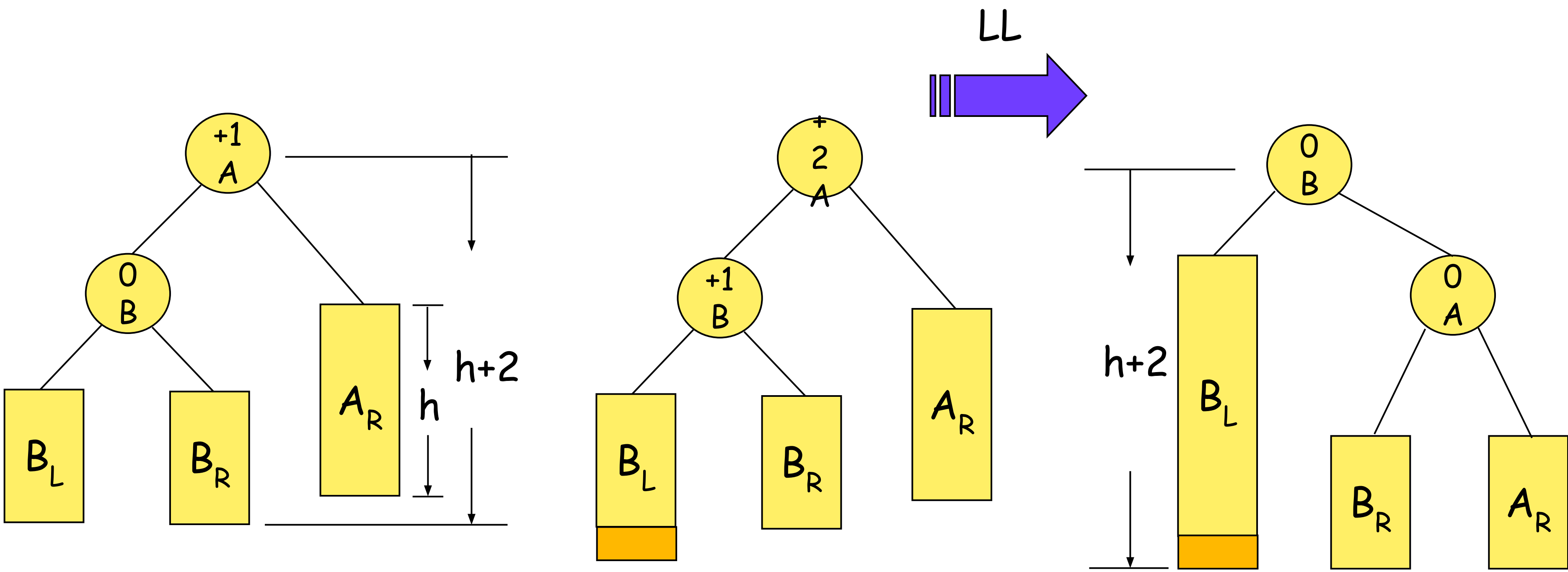


(i) Insert SEPTEMBER

Rebalancing Rotation of Binary Search Tree

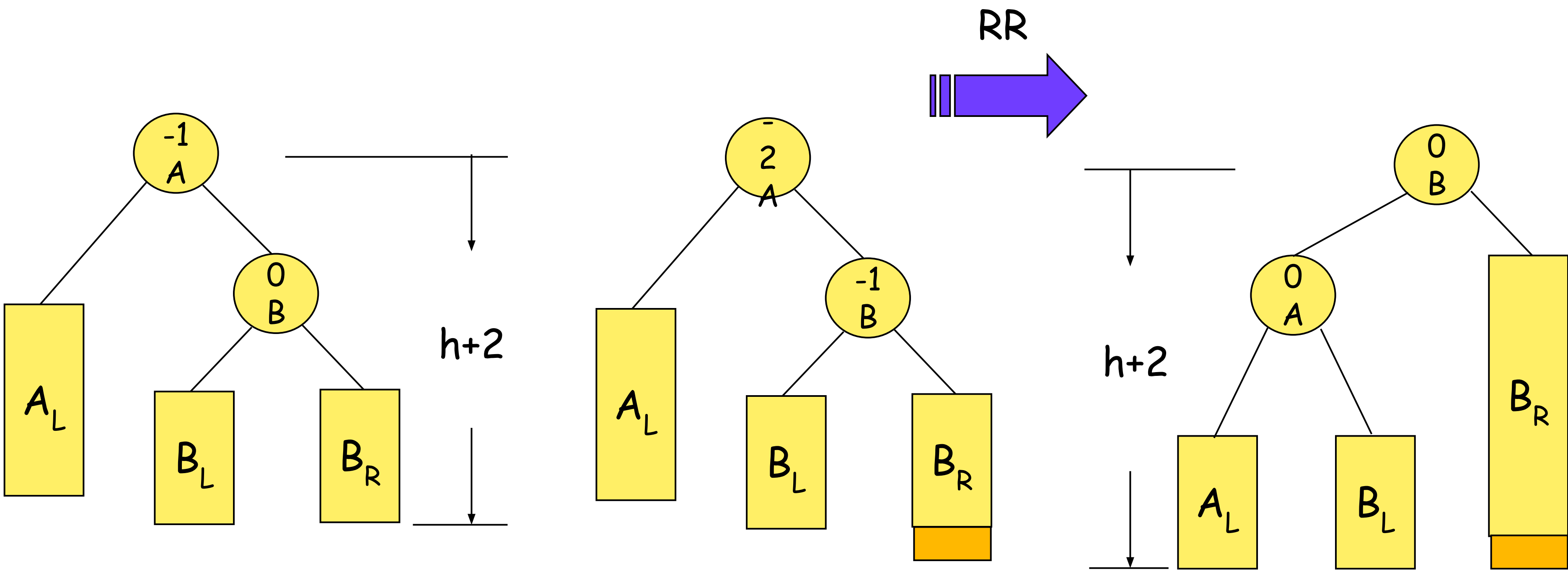
- LL: new node Y is inserted in the left subtree of the left subtree of A
- LR: Y is inserted in the right subtree of the left subtree of A
- RR: Y is inserted in the right subtree of the right subtree of A
- RL: Y is inserted in the left subtree of the right subtree of A.
- If a height–balanced binary tree becomes unbalanced as a result of an insertion, then these are the only four cases possible for rebalancing.

Rebalancing Rotation LL



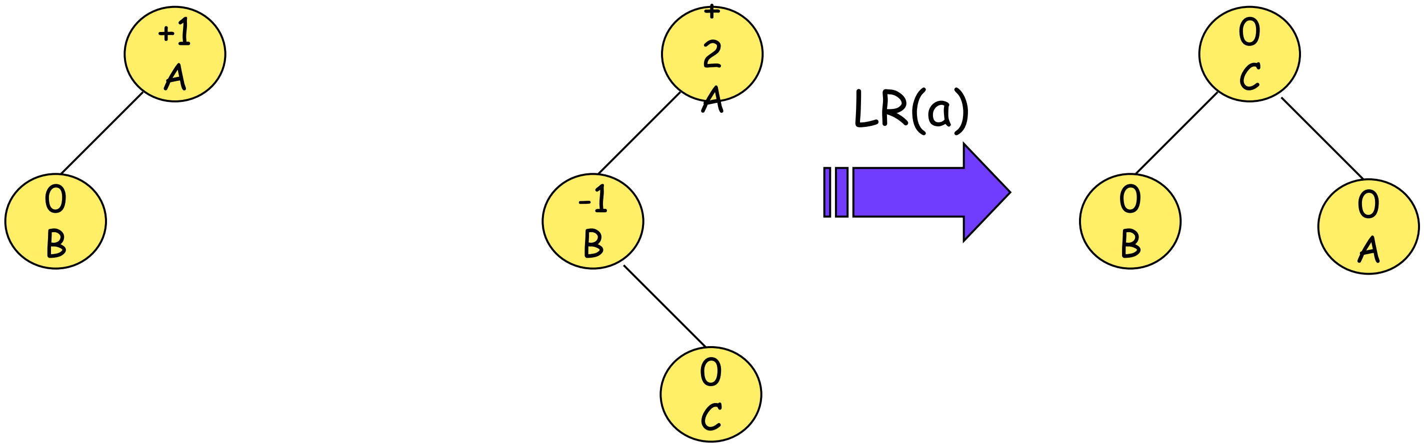
height of B_L increases to h+1

Rebalancing Rotation RR

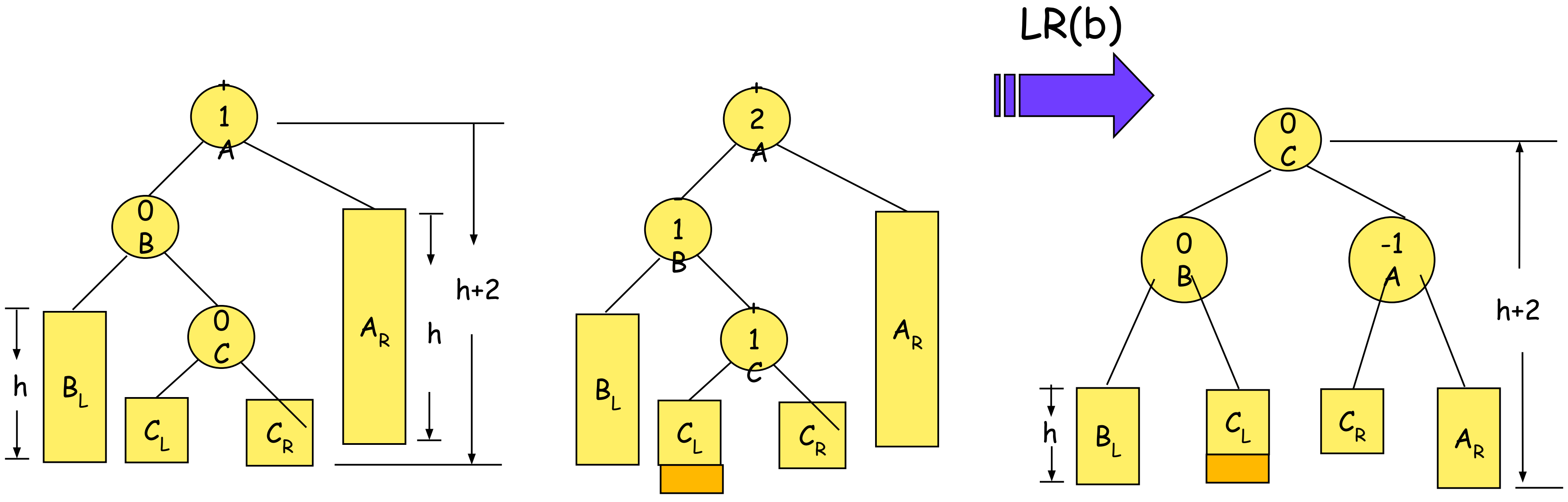


height of B_R increases to h+1

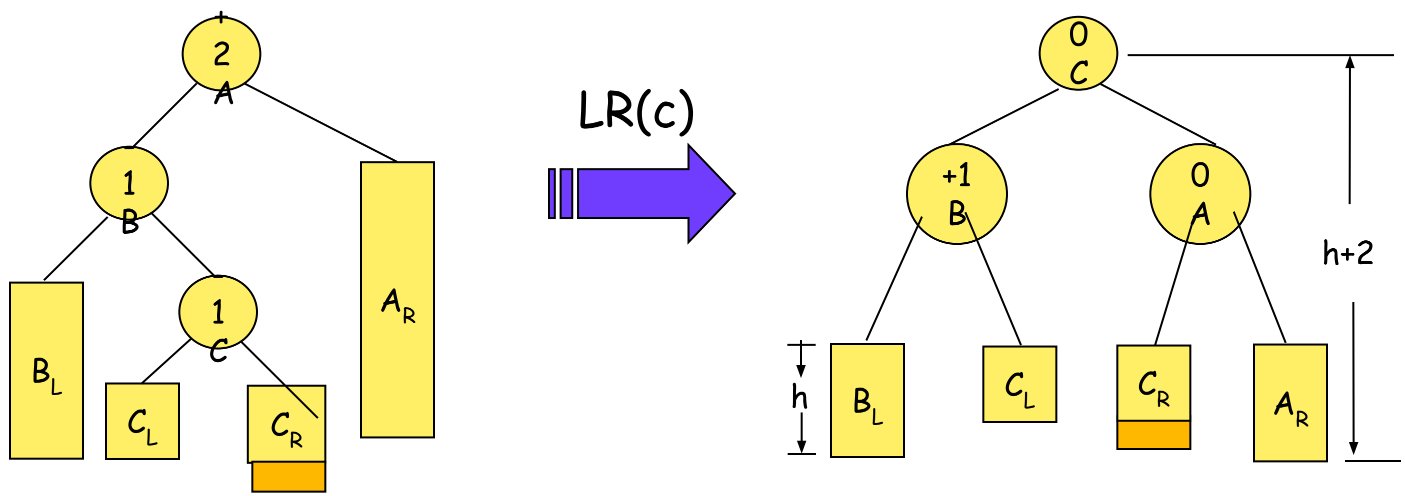
Rebalancing Rotation LR(a)



Rebalancing Rotation LR(b)



Rebalancing Rotation LR(c)



AVL Trees (Cont.)

- Once rebalancing has been carried out on the subtree in question, examining the remaining tree is unnecessary.
- To perform insertion, binary search tree with n nodes could have O(n) in worst case. But for AVL, the insertion time is O(log n).