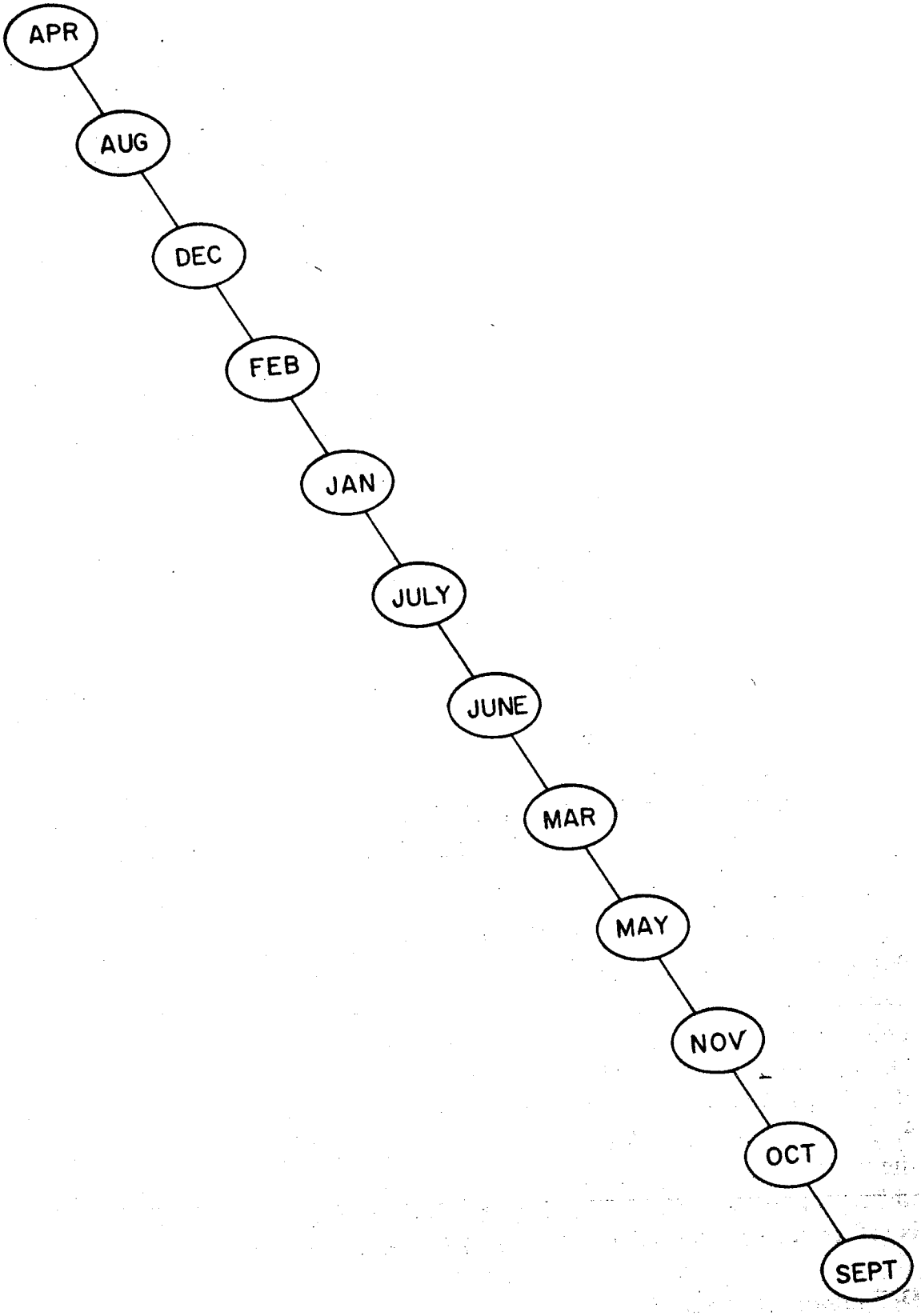
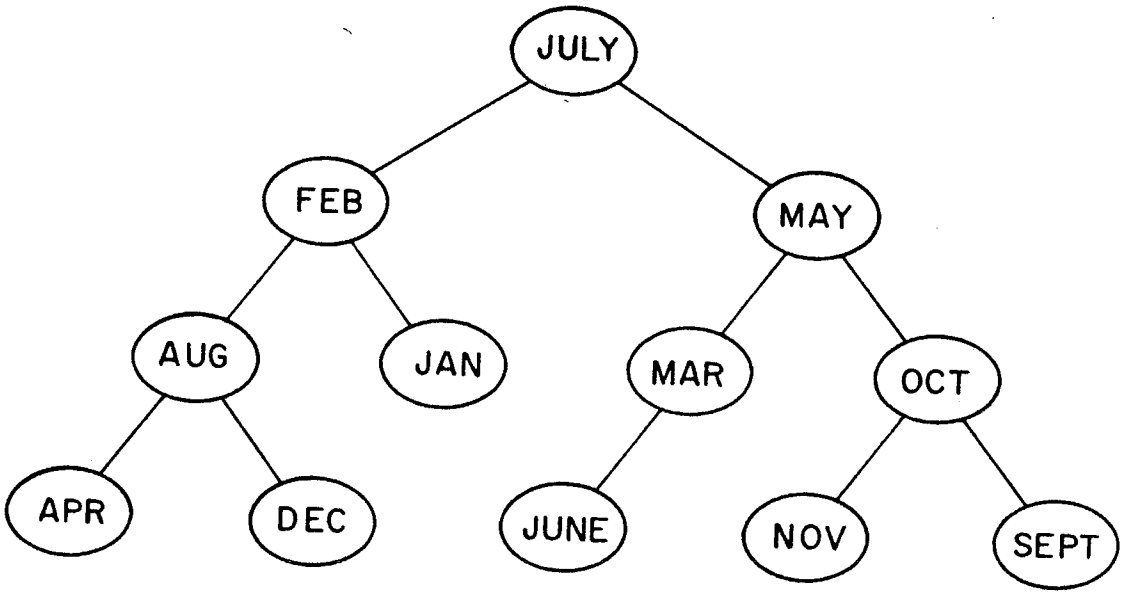
Balanced Trees

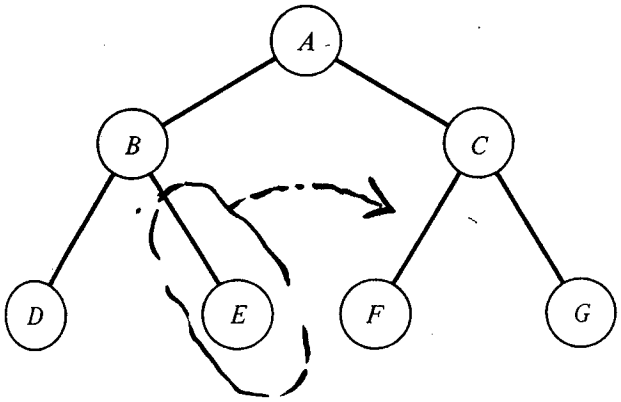
- Why balanced tree?
  - If all keys equally likely to be searched, a balanced binary search tree is most efficient
- Generating balanced tree (AVL)

Example

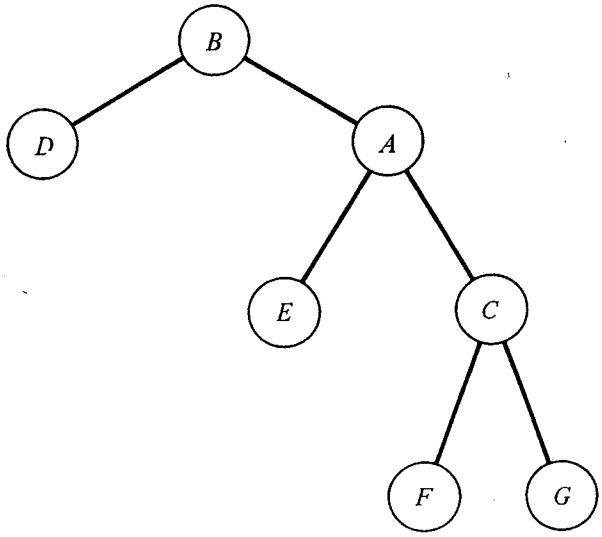


Example

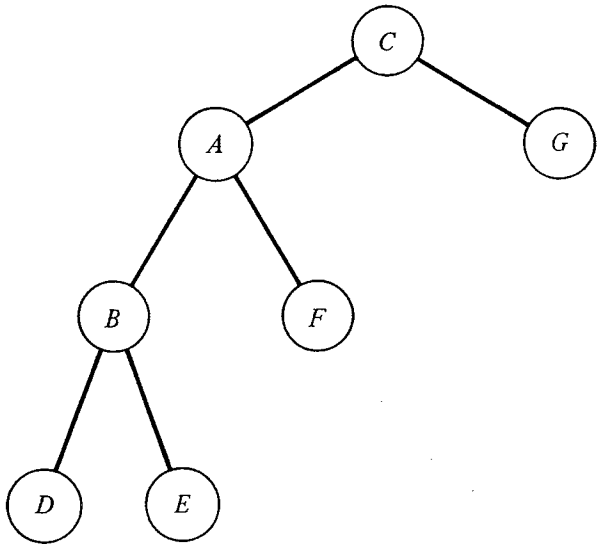




(a) Original tree.



(b) Right rotation.

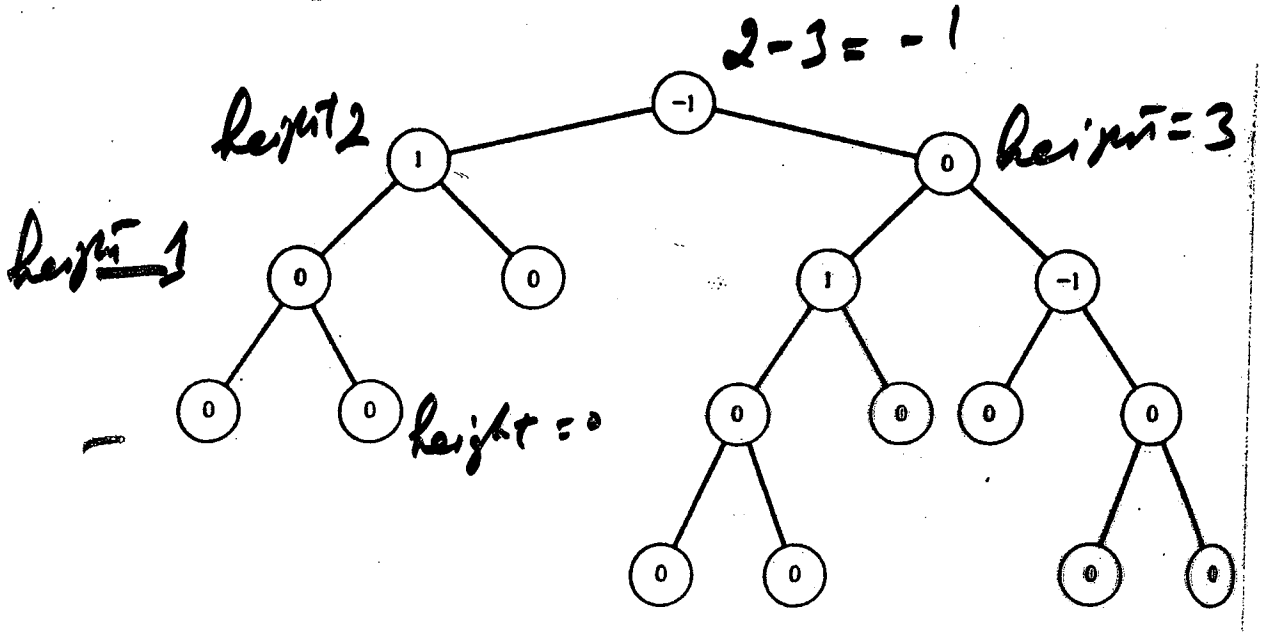


(c) Left rotation.

Inorder traversal is same for all cases.

Balance

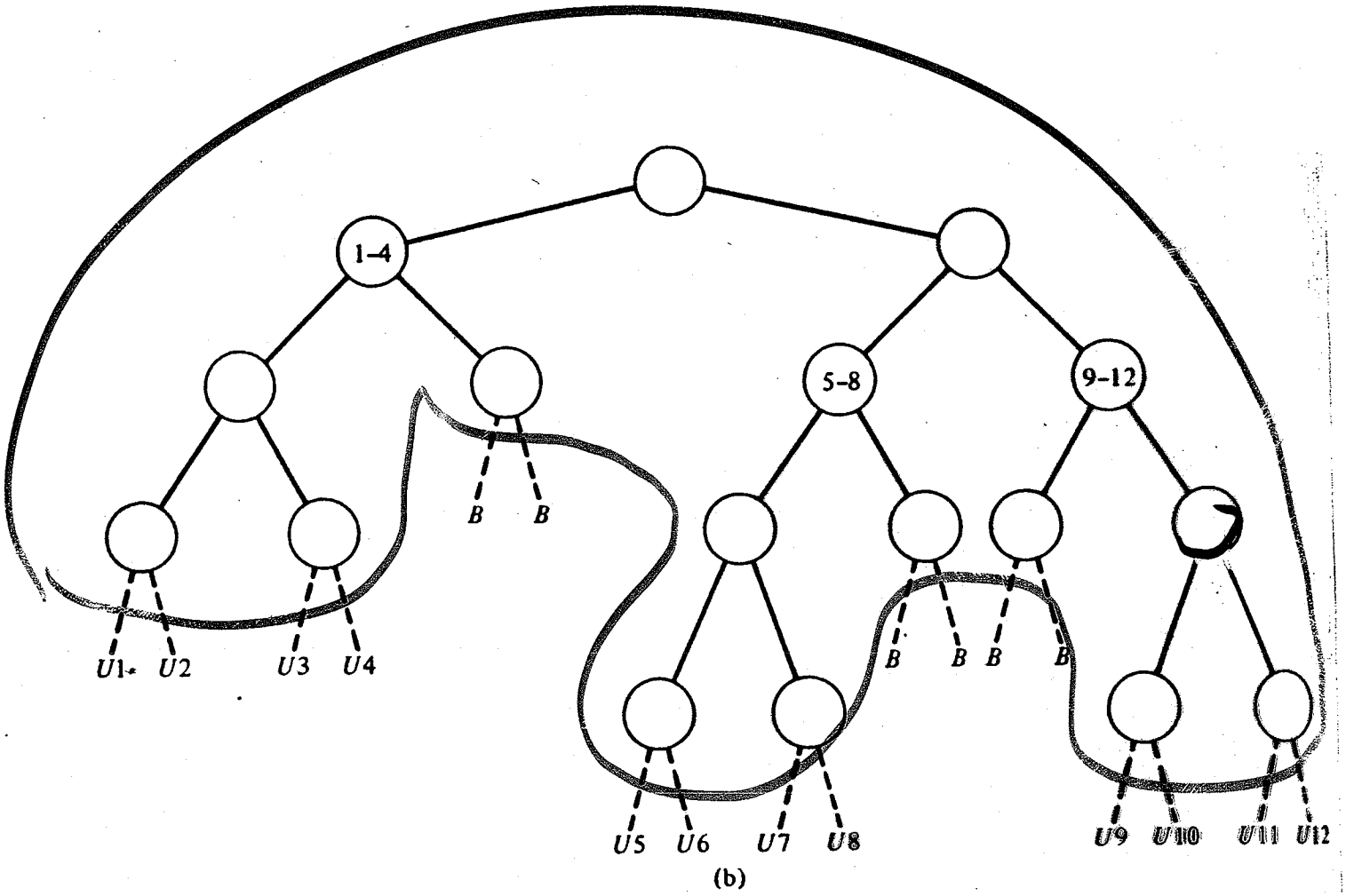
The balance of a node is defined as the height of its left subtree minus the height of its right subtree



An AVL node can only have a balance of –1, 0, or 1.

Note: height of a node is the max level of its leaves + 1 (A null tree has height of –1).

Example



(b)

Operations on AVL

Insertion: Can we insert a node in the AVL tree and maintain the balance requirements?

Look at the previous Example

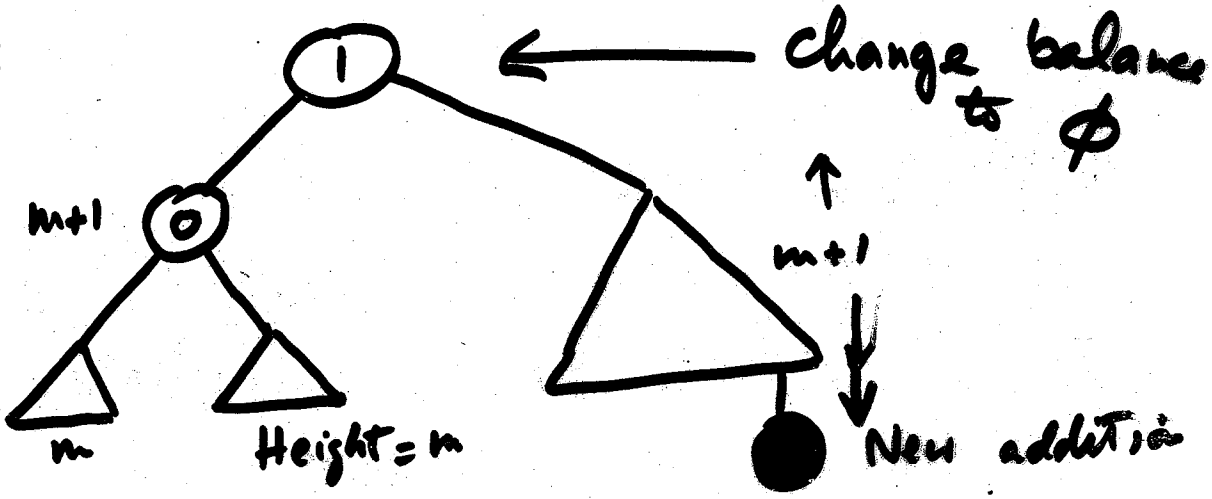
B – maintains balance
U – doesn't

A tree becomes unbalanced when the inserted node is the left descendent of a node whose balance was 1 or the right descendent of a node whose balance was –1

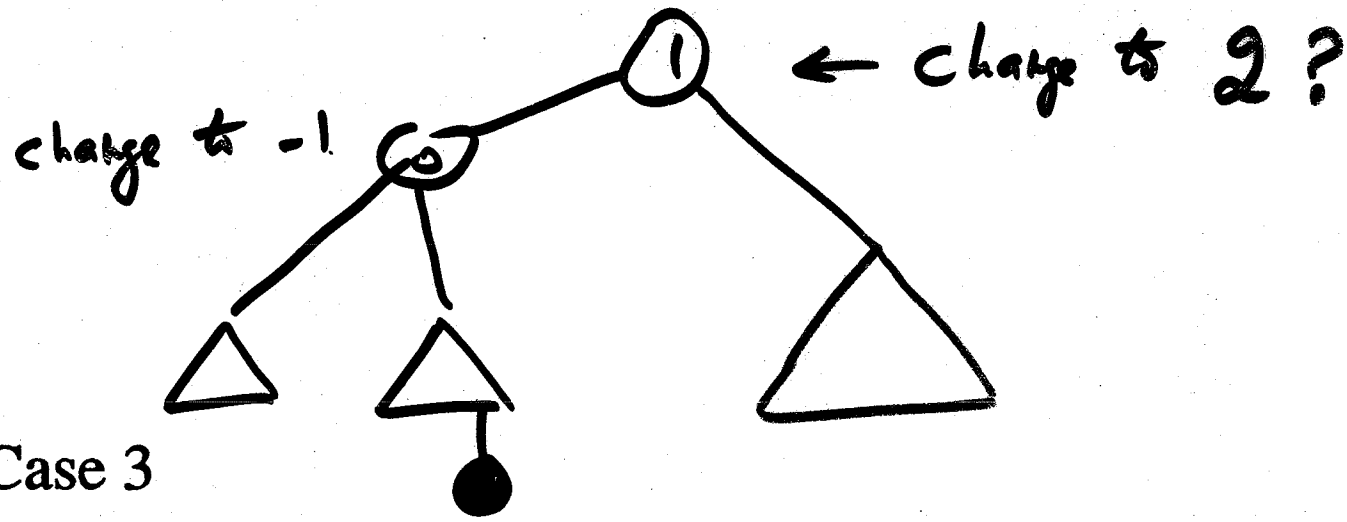
Insertion in AVL

Suppose we have a balanced tree:

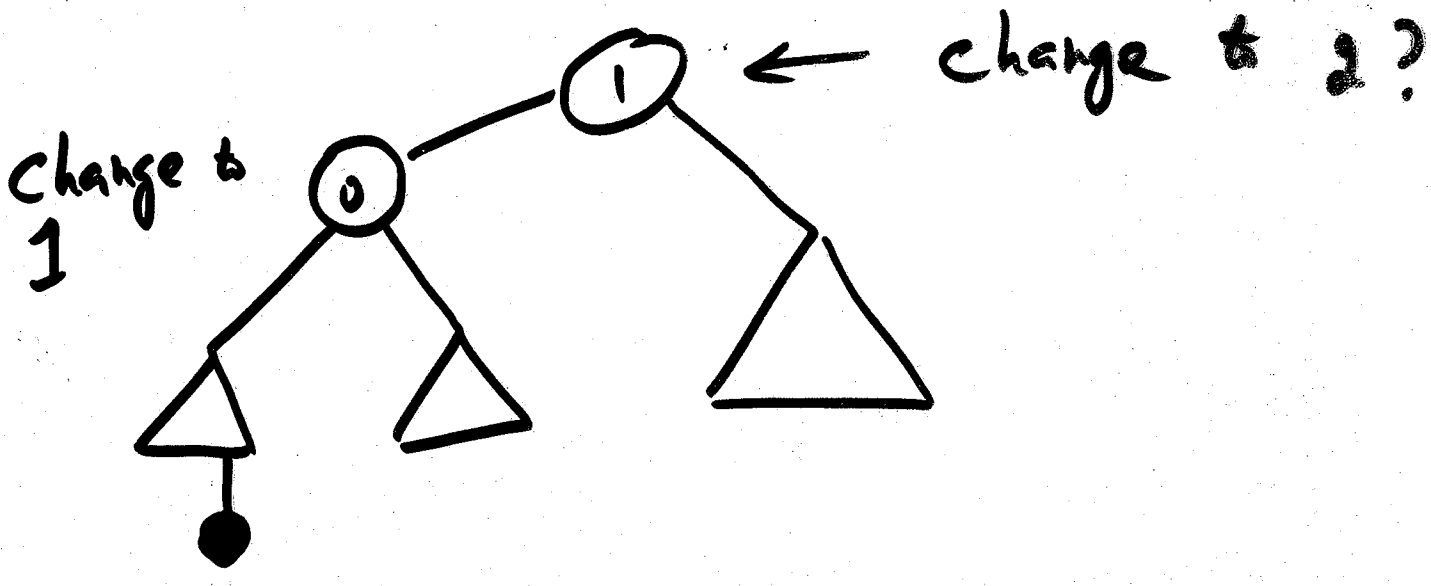
Case 1:



Case 2:



Case 3



Insertion in AVL

Case 1 is ok on balance requirement but Case 2 and 3 require adjusting the tree in some manner to make it balanced {–1,0,1}, while maintaining the same inorder listing!

To maintain a balance tree, we
must find some transformation that
we can apply to the tree such that

1. The transformed tree meets **balance requirement.**

2. The **inorder transversal** is the same for the original trees.

– We will use rotation to achieve these 2 goals

Observations

– Rotations are always carried out with respect to the closest parent of the new node having a balance of ± 2 .

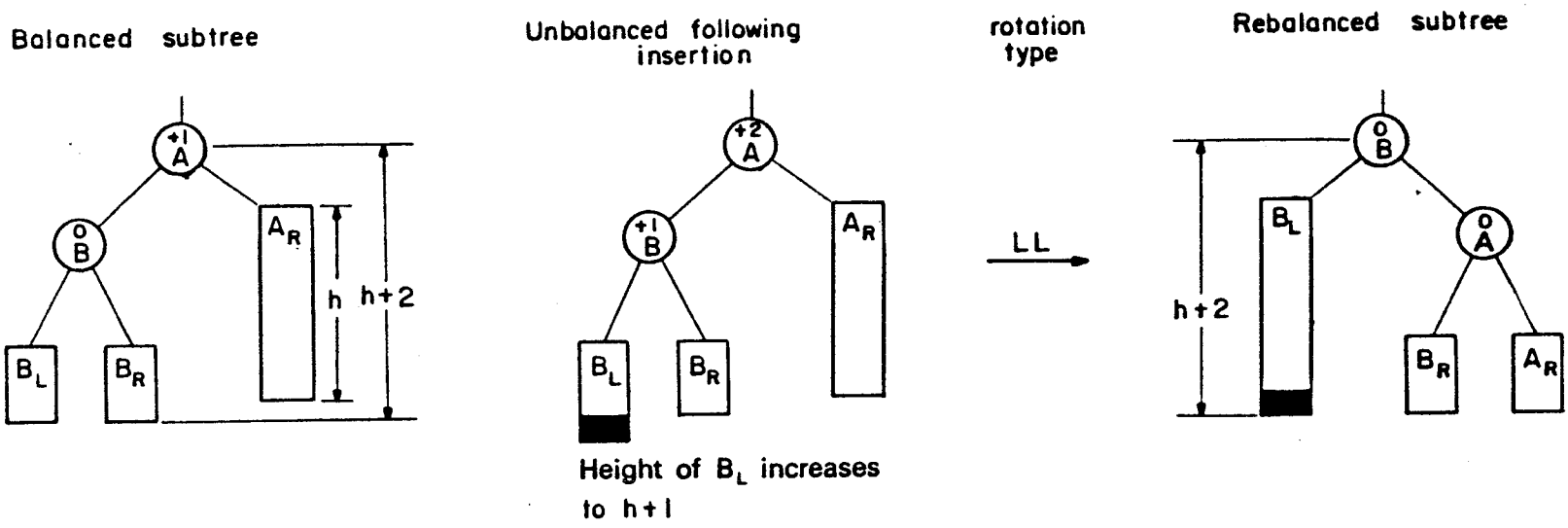
– Four different kinds of rotations (characterized by the nearest ancestor A of the new node Y, whose balance factor becomes ± 2).

- LL
- RR
- LR
- RL

LL and RR are symmetric, as are LR and RL

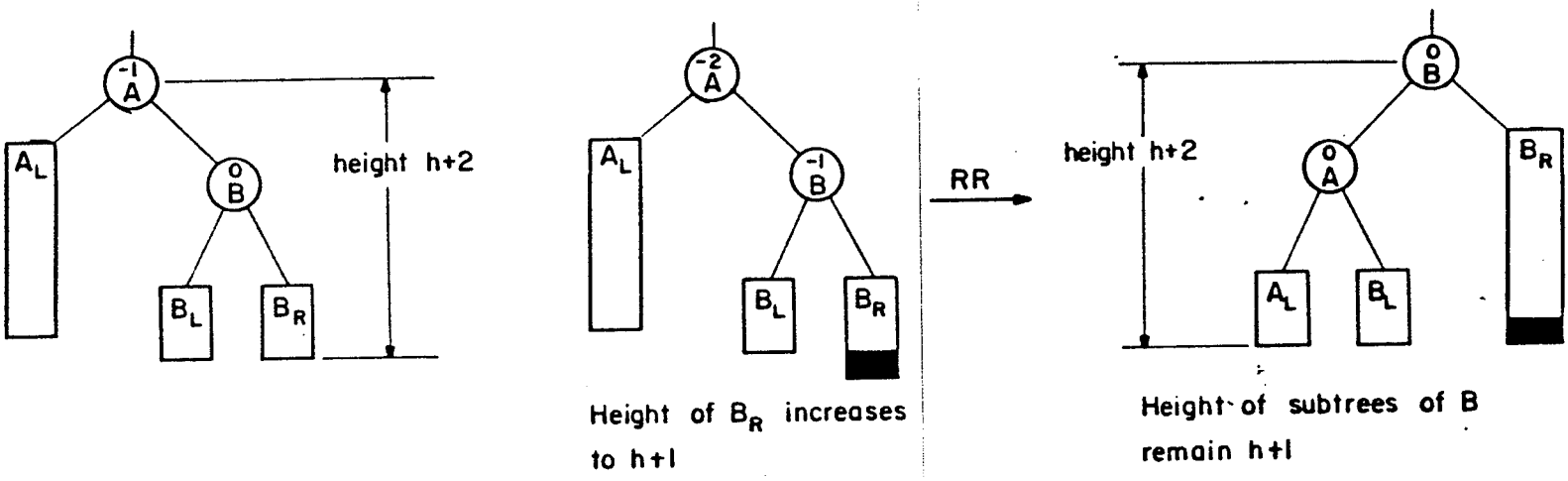
Summary of Insertions and Transformation

LL : new node Y is inserted in the left subtree of the left subtree of A.



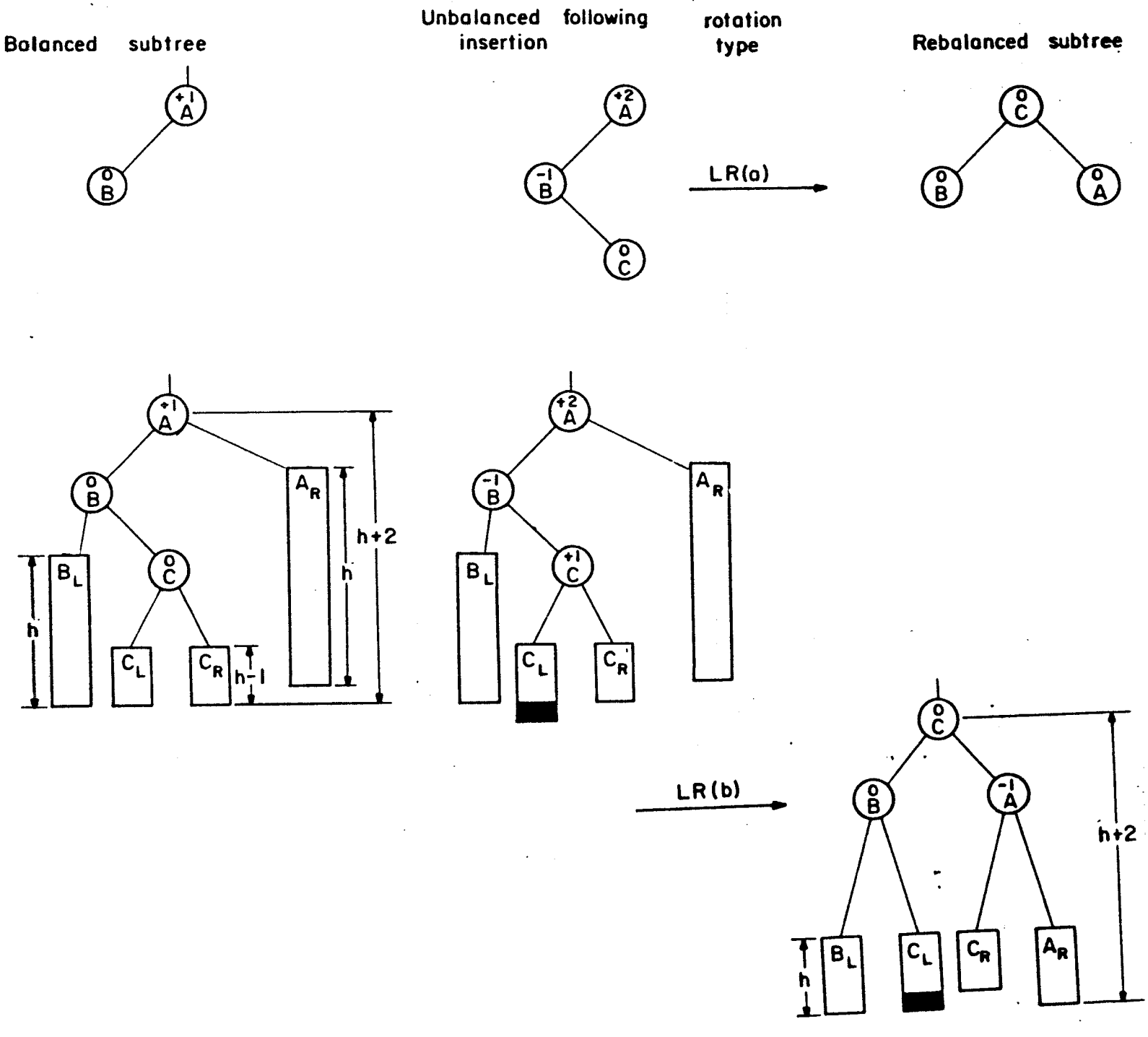
Summary of Insertions and Transformation

RR : new node Y is inserted in the right subtree of the right subtree of A.



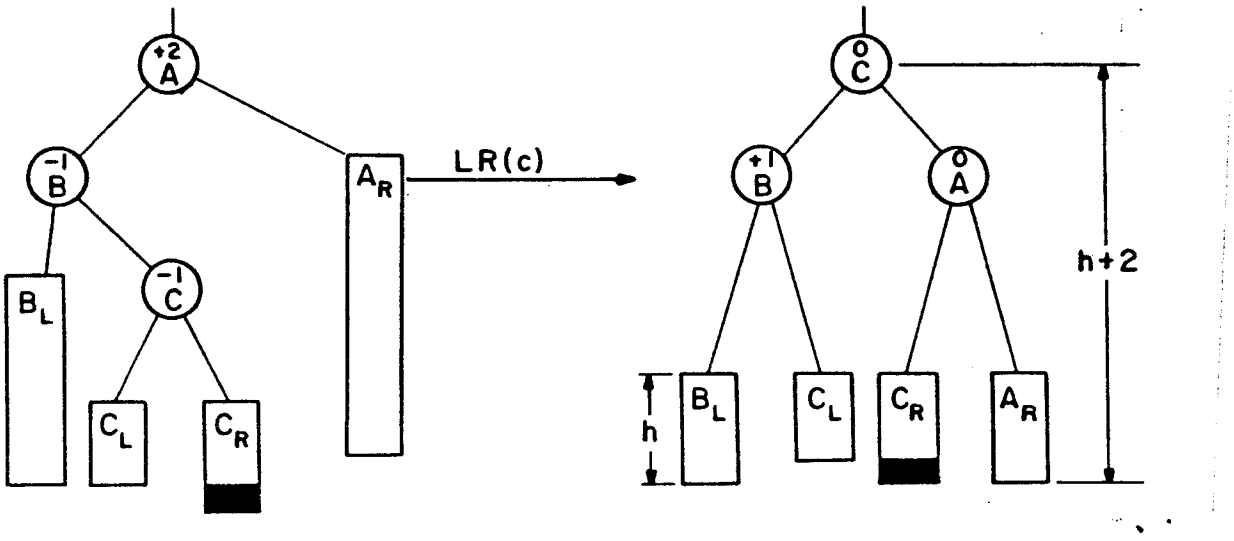
Summary of Insertions and Transformation

LR : new node Y is inserted in the right subtree of the left subtree of A.

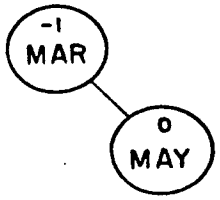
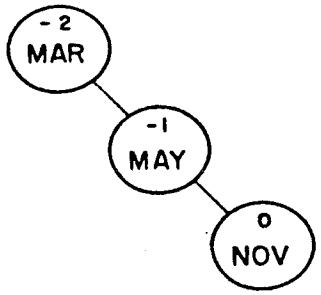
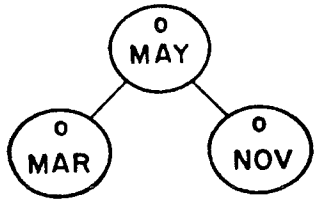


Summary of Insertions and Transformation

LR : new node Y is inserted in the right subtree of the left subtree of A.



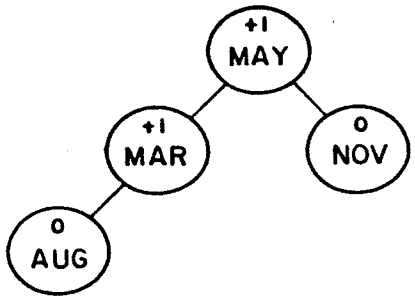
Example

New Identifier	After Insertion	After Rebalancing
(i) MARCH		No rebalancing needed
(ii) MAY		No rebalancing needed
(iii) NOVEMBER		

RR

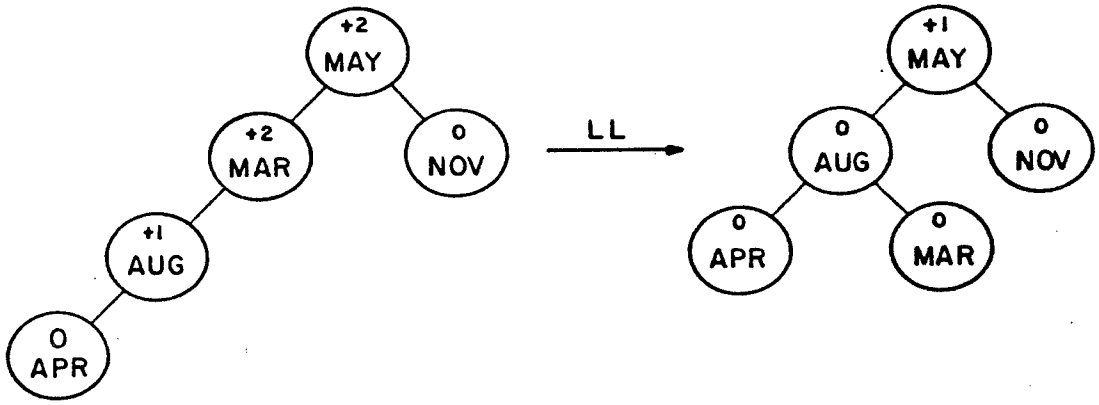
Example

(iv) AUGUST



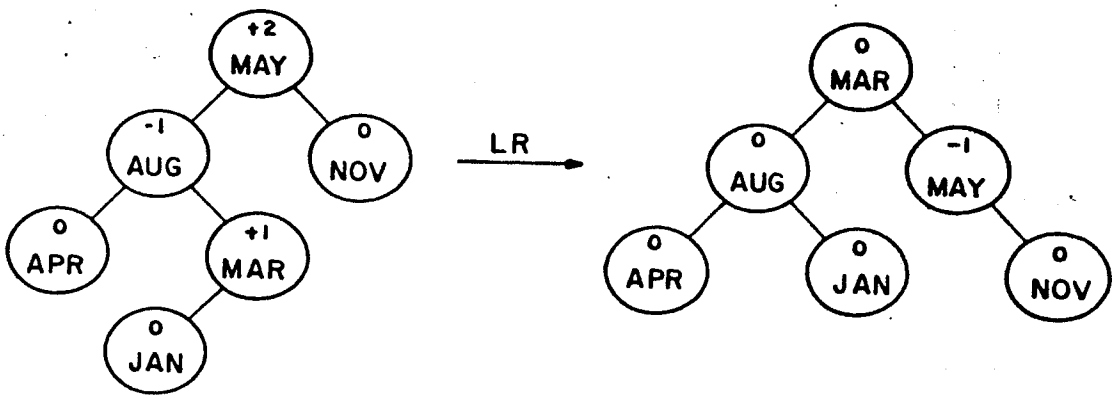
No rebalancing needed

(v) APRIL

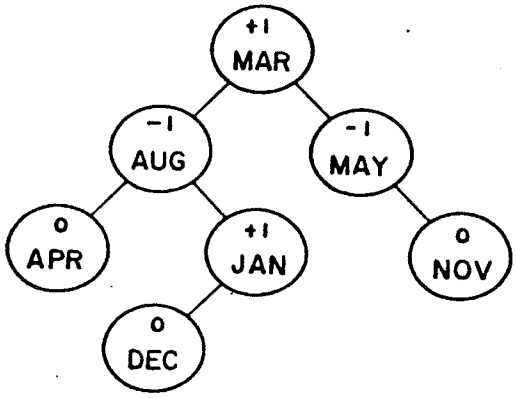


Example

(vi) JANUARY

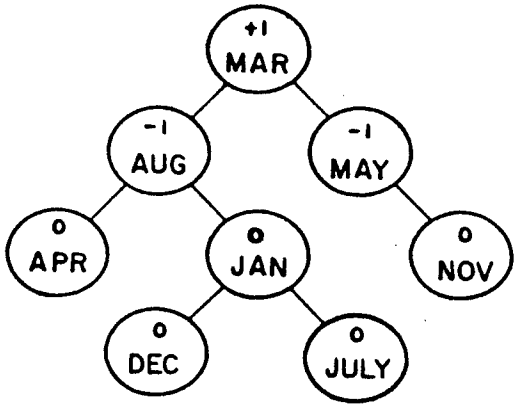


(vii) DECEMBER



No rebalancing needed

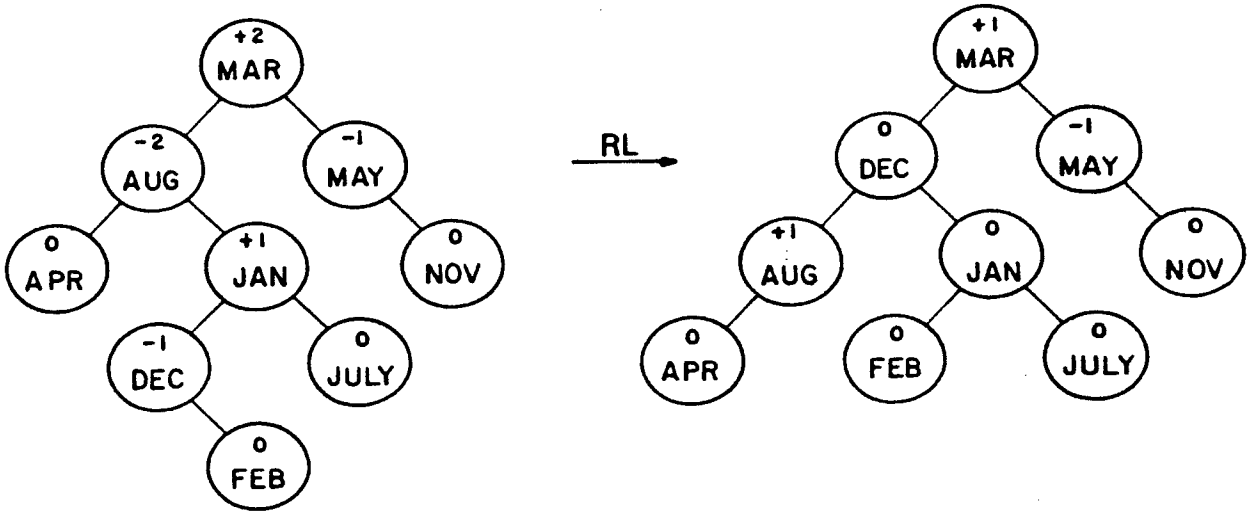
(viii) JULY



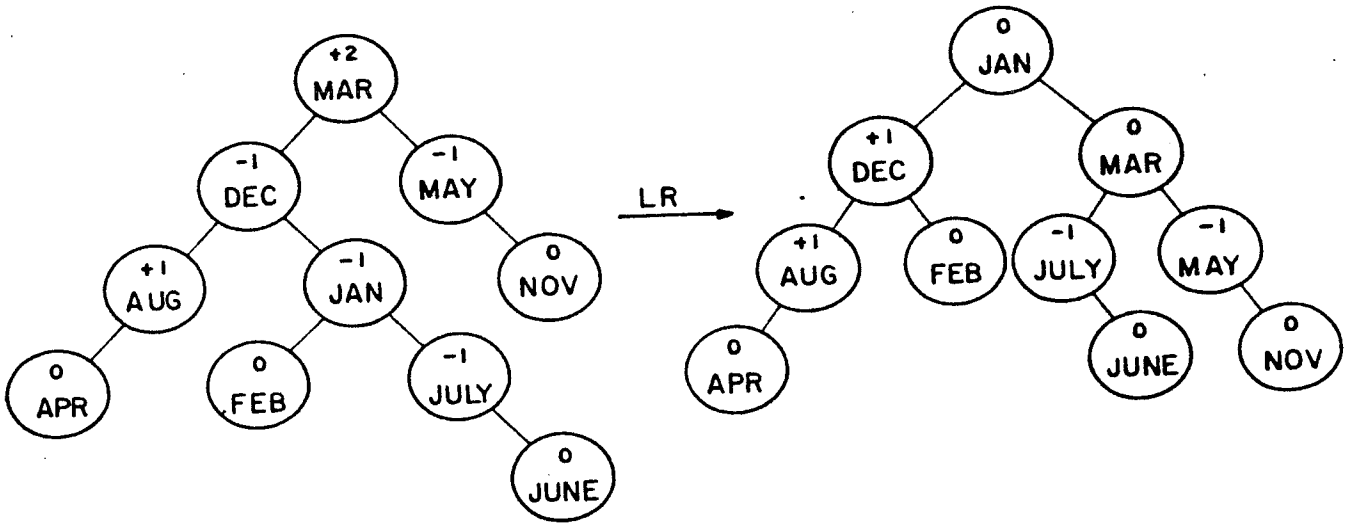
No rebalancing needed

Example

(ix) FEBRUARY

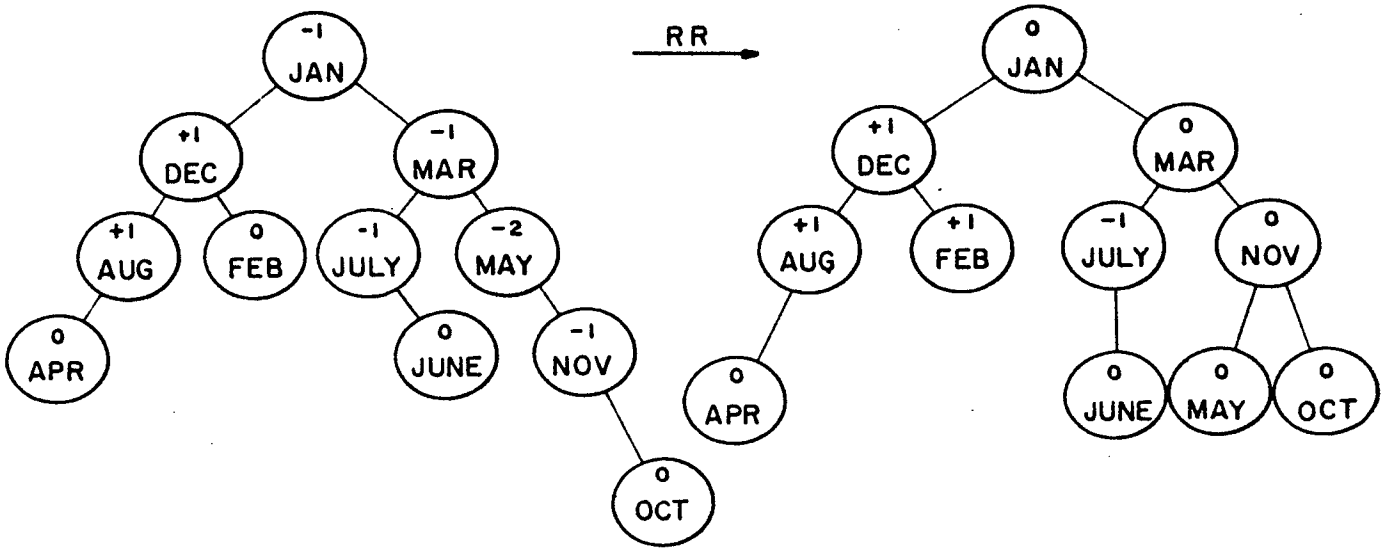


(x) JUNE

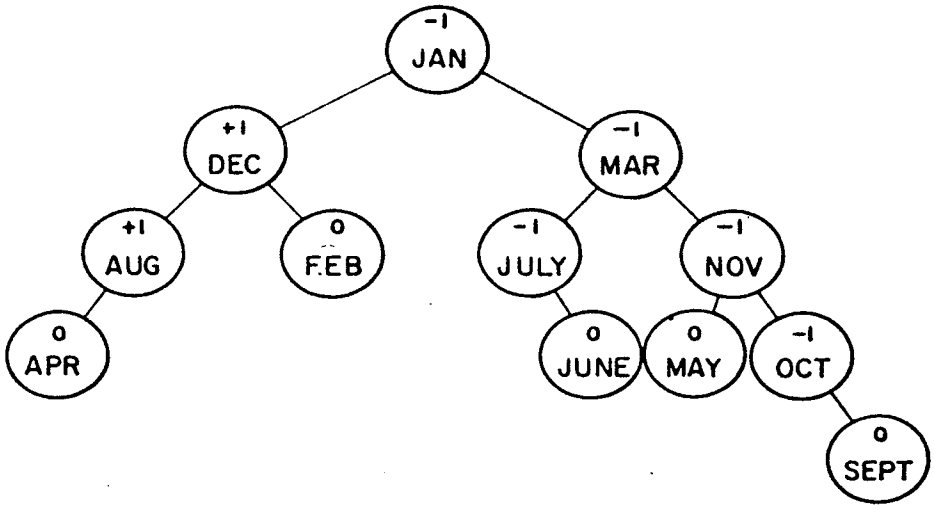


Example

(xi) OCTOBER



(xii) SEPTEMBER



No rebalancing needed

Complexity of AVL

– What is the time to insert a node in a balanced binary tree?

– What is the time needed to rebalance?

– The maximum height of any balanced search tree is 1.44 log n. In actual practice, they behave much better — log n + .25 search time for large n.

– On the average a rotation is required in something less than half the insertions — 46.5% of the insertions.

– The algorithm to delete is much harder — A deletion requires more than 1–2 rotation at each level of the tree — 0 (log n) rotations. Average found is .214 (single or double rotations) per deletion.