Balanced Trees

Binary Search trees

Tree

- A tree is a finite set of elements that is either empty or partitioned into ***three or more disjoint sets***
- The first subset contains one element and is called ***root of the tree***
- The other subsets are themselves ***tree*** and are called the ***subtrees*** of original tree
- Each element of the tree is called a ***node***
- ***This node*** is different from the node of a singly linked list in the sense that it always contains ***more than one address field.***

Binary Tree

- A binary tree is a finite set of elements that is either empty or partitioned into **three disjoint sets**
- The first subset contains single element and is called **root of the tree**
- The other subsets are themselves **binary tree** and are called ***left subtree and right subtree of original tree***
- Each element of the binary tree is called a node having two address fields.

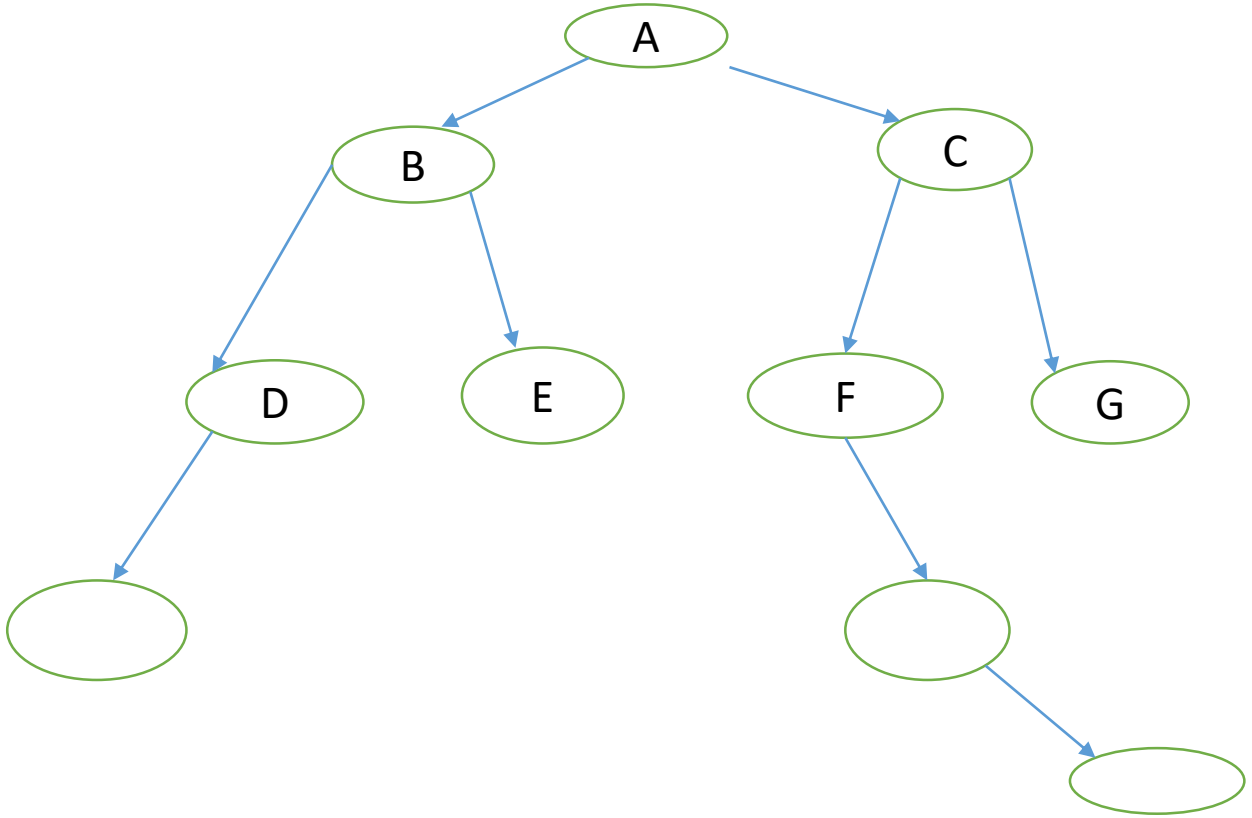
Why balanced tree??

- The time required to search a binary tree varies between O(n) and O($\log_2$n)
- The structure of tree depends on the order in which the records are inserted.
- If the records are in some sorted order, then the resulting tree will have either left link NULL or right link NULL
- But, if the records are inserted such that half the records are in left subtree and half of them are in right subtree --- a kind of balanced tree is created
- It can be observed that the degree of balance is dependent on the order in which keys are inserted
- The insertion algorithm studied by us does not ensure a balanced tree

A balanced binary search tree (AVL tree)

- The balance of a tree looks related to height of its left subtree and height of its right.
- Height of a null tree is -1, and 0 if it contains single node.
- ***Balance of a node = height of its left subtree - height of its right subtree***
- A balanced binary tree is a binary tree in which balance of every node is never more than 1 or always less than equal to one
- This means balance of every node is either 0 or -1 or 1 in case of a balanced tree
- ***A balanced binary tree also called AVL Tree is a binary tree in which the heights of two subtrees never differ by more than 1***
- AVL tree is a self balancing binary tree invented by G.M. Adelson-Velsky and E.M. Landis

Calculating the balance of each node

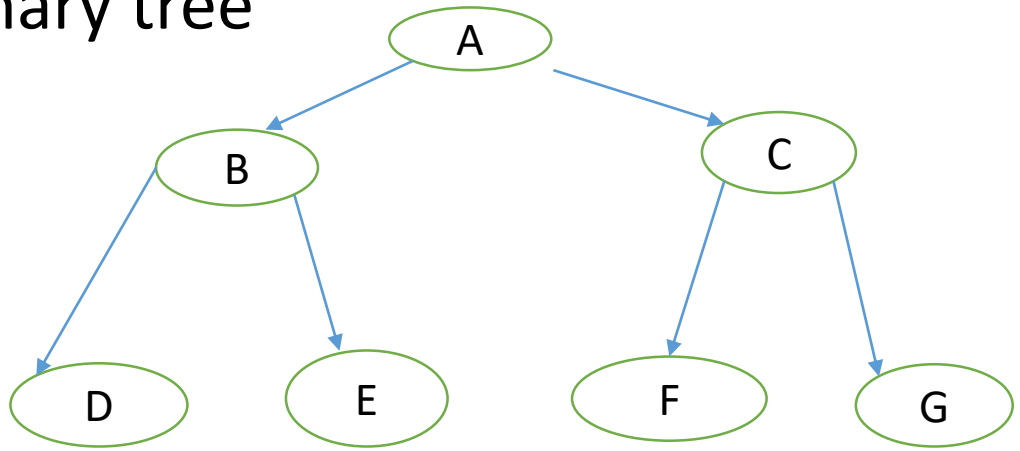


Building an AVL Tree

- AVL tree is same as BST but with a little difference that in AVL tree balance of every node is also stored.
- So, while building an AVL tree, it has to be checked that the balance of any node at any time is not more than 1
- If the balance becomes more than 1 than right rotations and left rotations are performed so that the tree becomes balanced
- What is rotation??
- Let us examine a balanced tree:

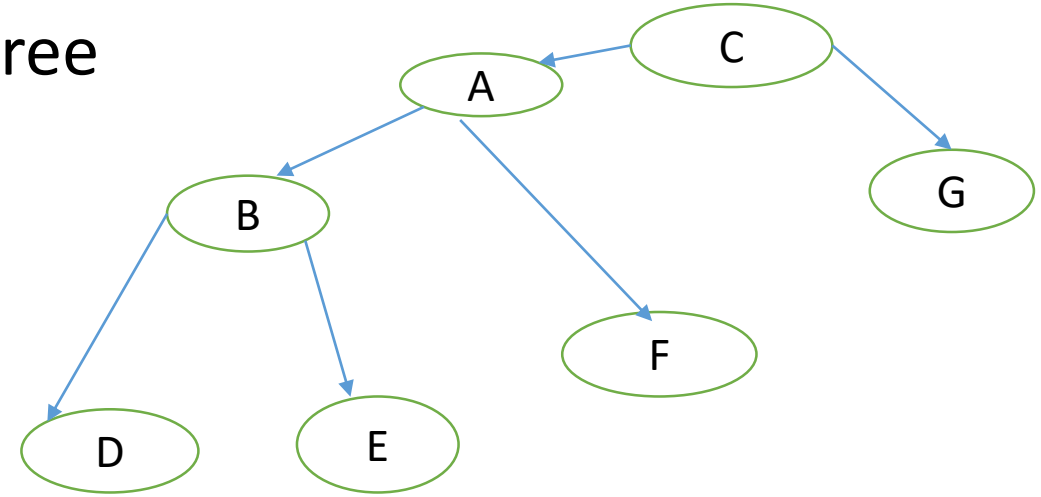
Rotation Concept

Complete binary tree



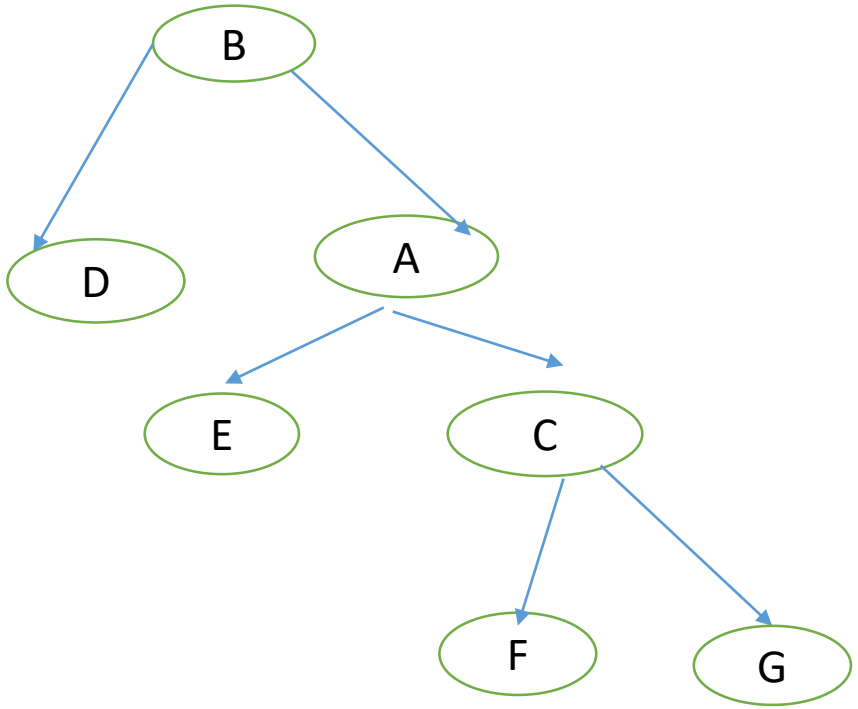
Left Rotation

Unbalanced tree



Right Rotation of complete tree

Unbalanced tree



Building an AVL tree

While building an AVL tree, it is necessary to perform rotations to keep it balanced but the rotations shall be such that :

1. The inorder traversal of the transformed tree is same as of original tree
2. The transformed tree is balanced i.e. balance of every node is not more than one.

Rotation algorithms

- **An algorithm to implement a left rotation at p is as follows**:

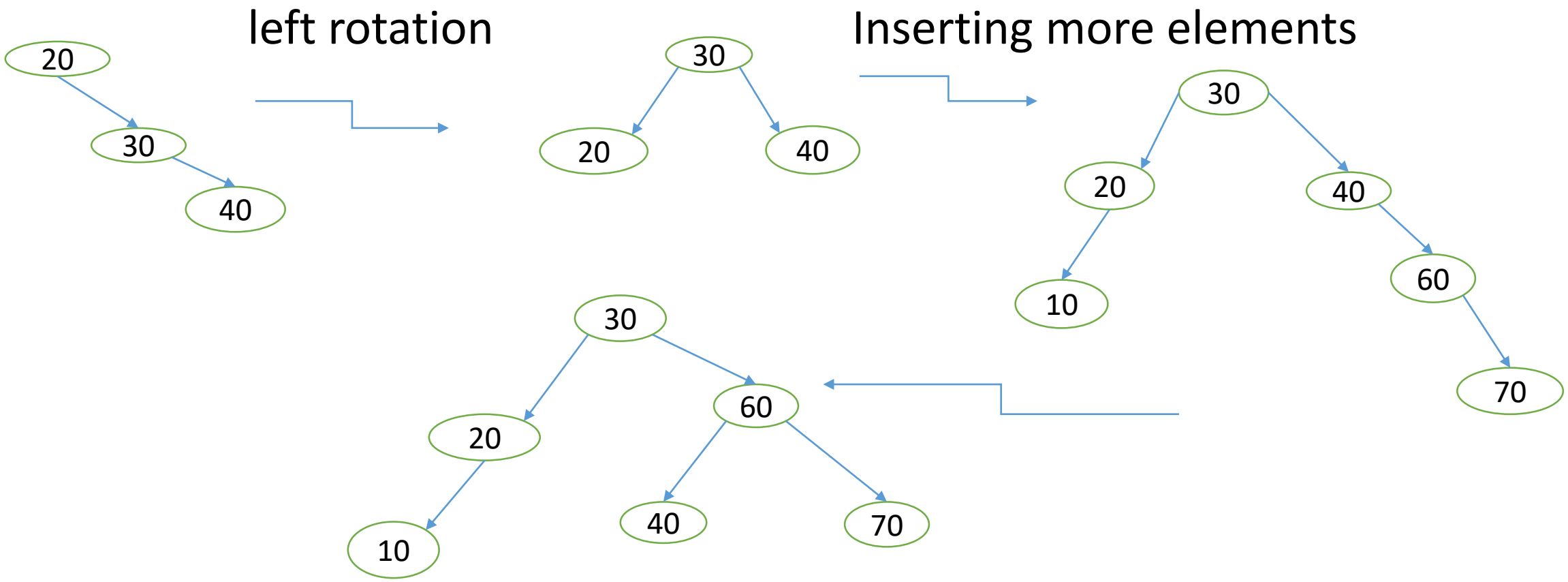
```
q = p->right;
temp = q->left;
q->left = p;
P->right = temp;
```

- **An algorithm to implement a right rotation at p is as follows:**

```
q = p->left;
temp = q->right;
q->right = p;
P->left = temp;
```

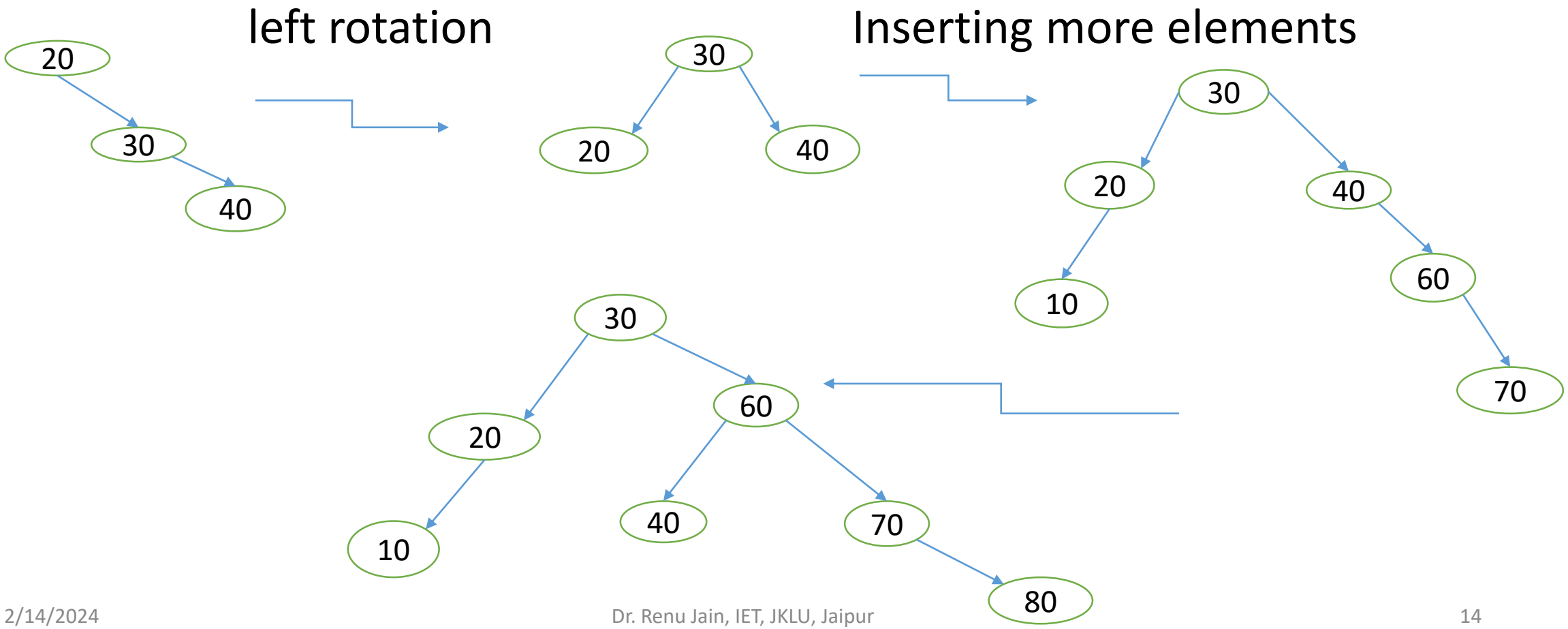
Creation of an AVL tree

- 20, 30, 40, 10, 60, 70

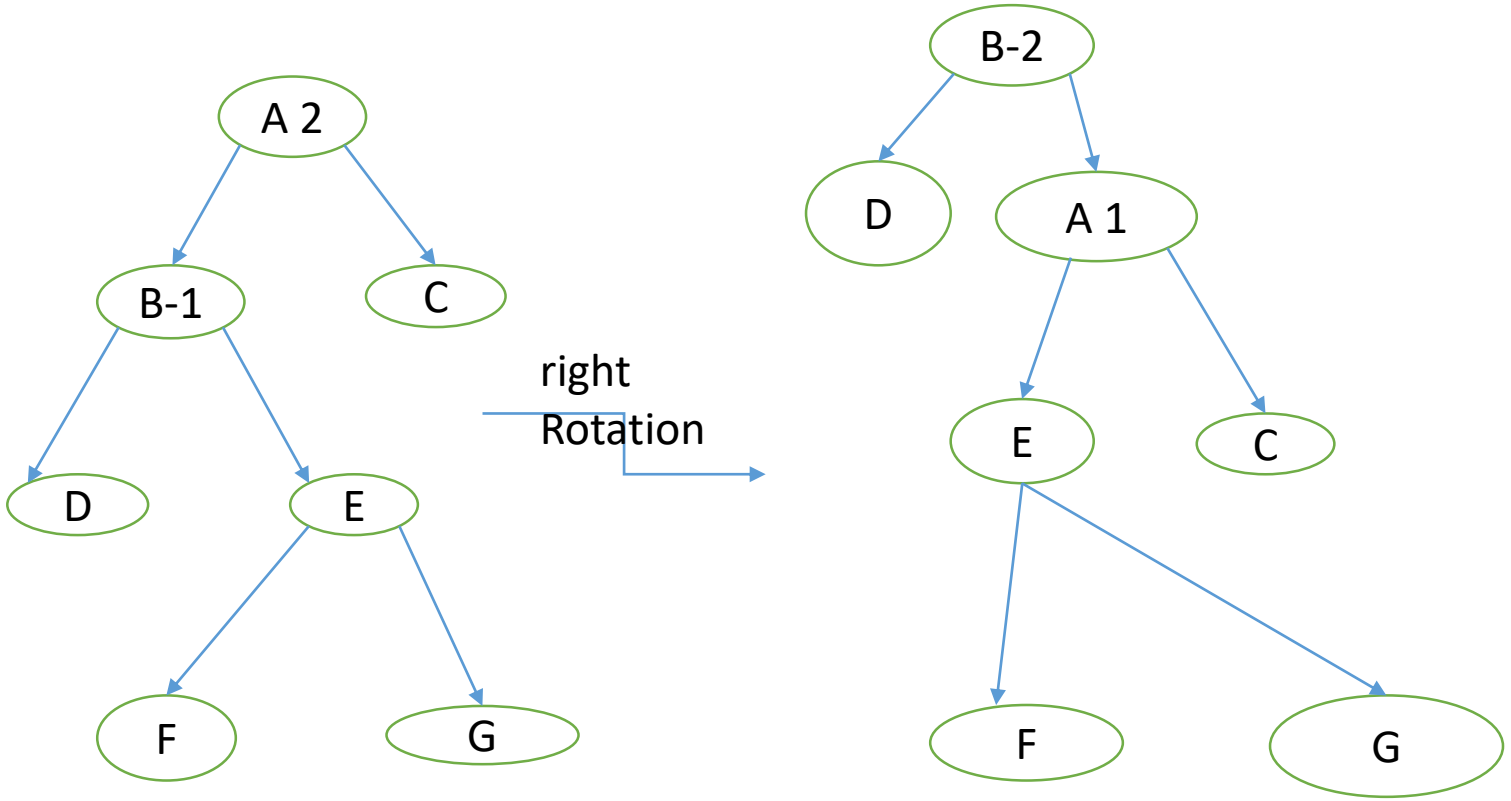


Creation of an AVL tree

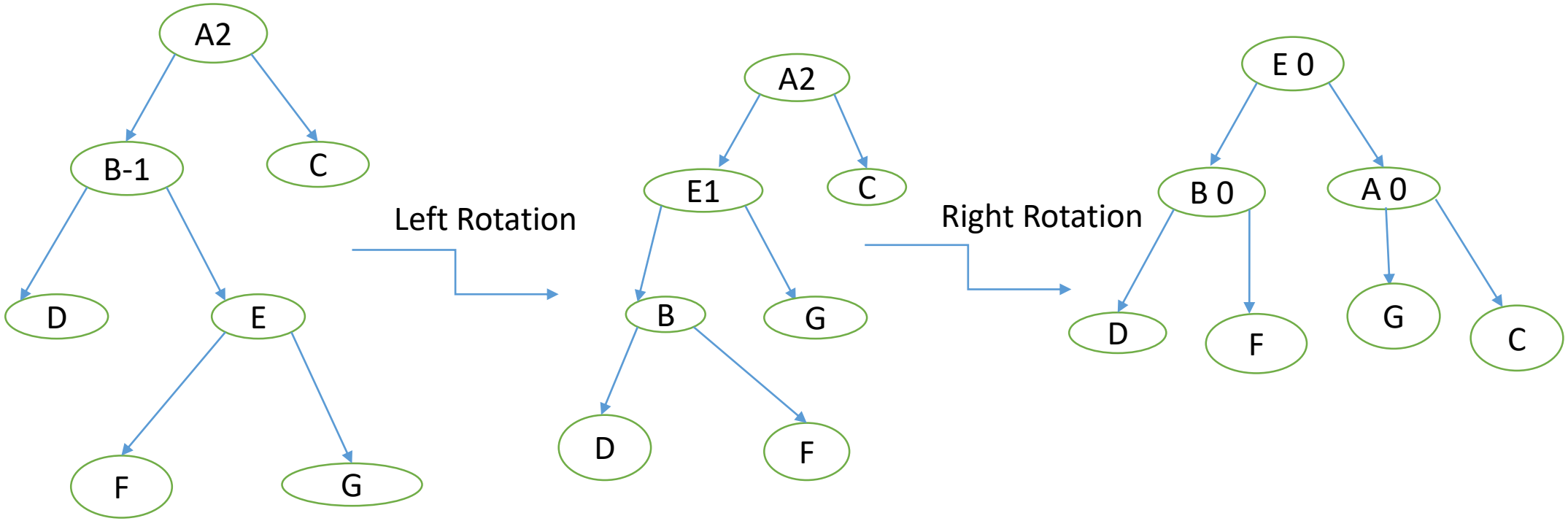
- 20, 30, 40, 10, 60, 70, 80



Double rotation(why)



Double rotation



Exercise

- Build an AVL tree for the given data:
- H, I, J, B, A, E, C
- H, I, J, B, A, E, C, F, D
- H, I, J, B, A, E, C, F, D, G, K, L

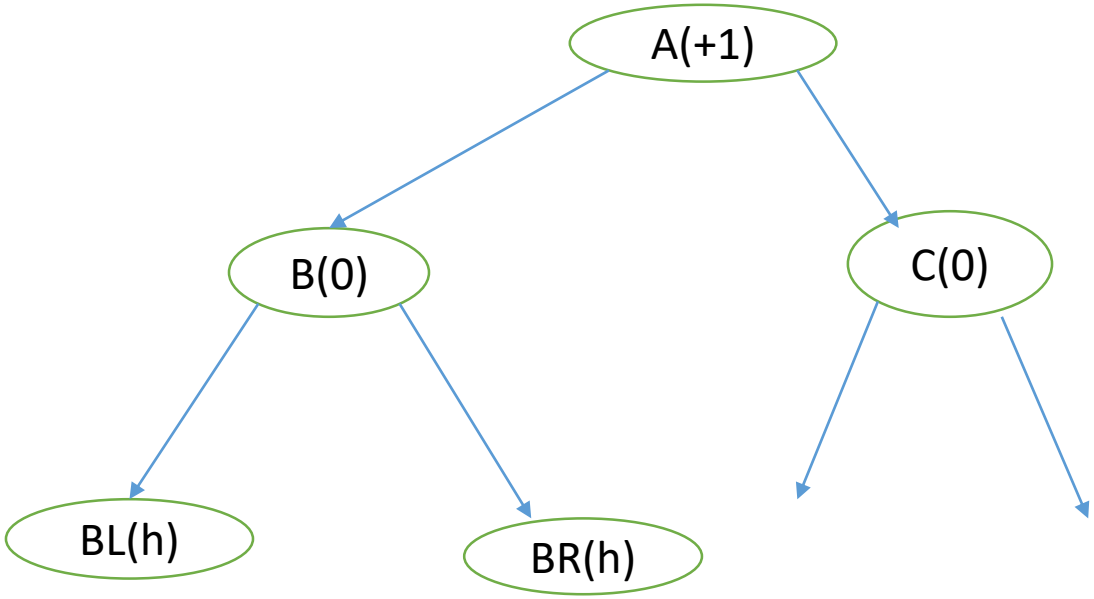
Type of Rotations

- There are four types of rotations:

1. LL rotation : inserted node is in the left subtree of left subtree of node
2. RR rotation : inserted node is in the right subtree of right subtree of node
3. LR rotation : inserted node is in the right subtree of left subtree of node
4. RL rotation : inserted node is in the left subtree of right subtree of node

LL Rotation

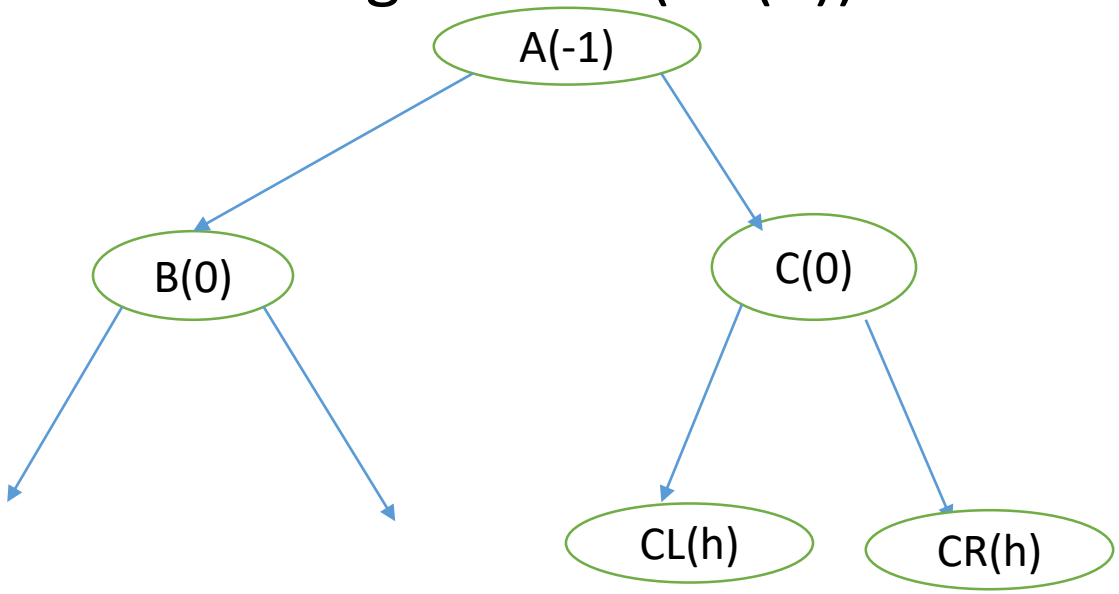
- If a node is added to the left of BL



- In this situation, to make it balanced right rotation is performed with respect to node A

RR Rotation

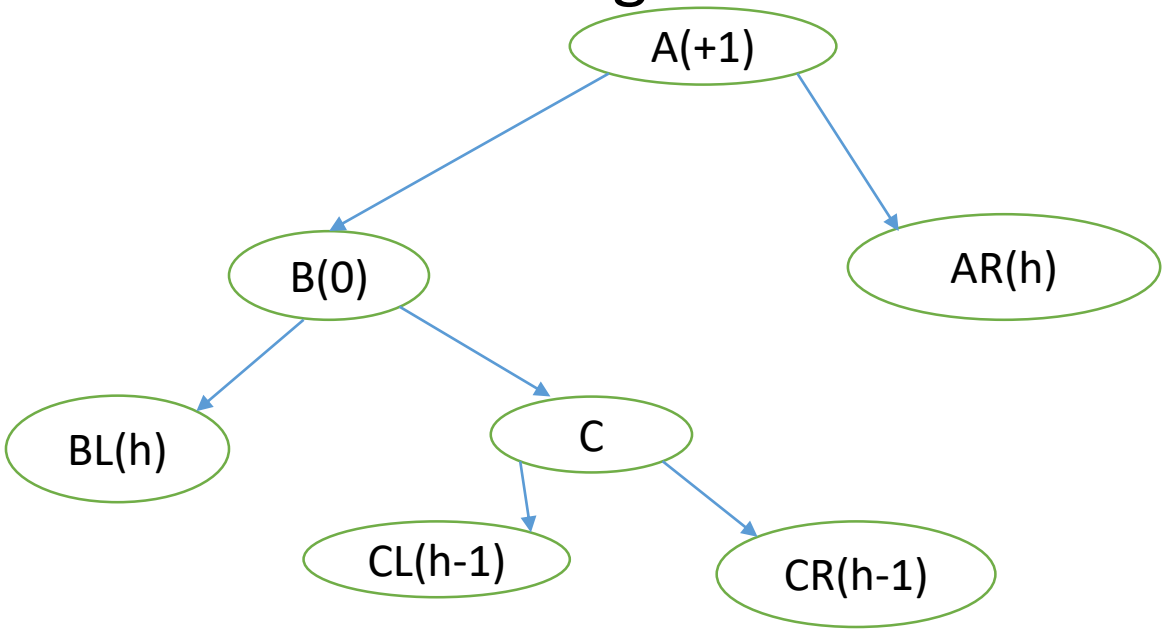
- If a node is added to right of CL (CR(h))



- In this situation, to make it balanced left rotation is performed with respect to node A

LR Rotation

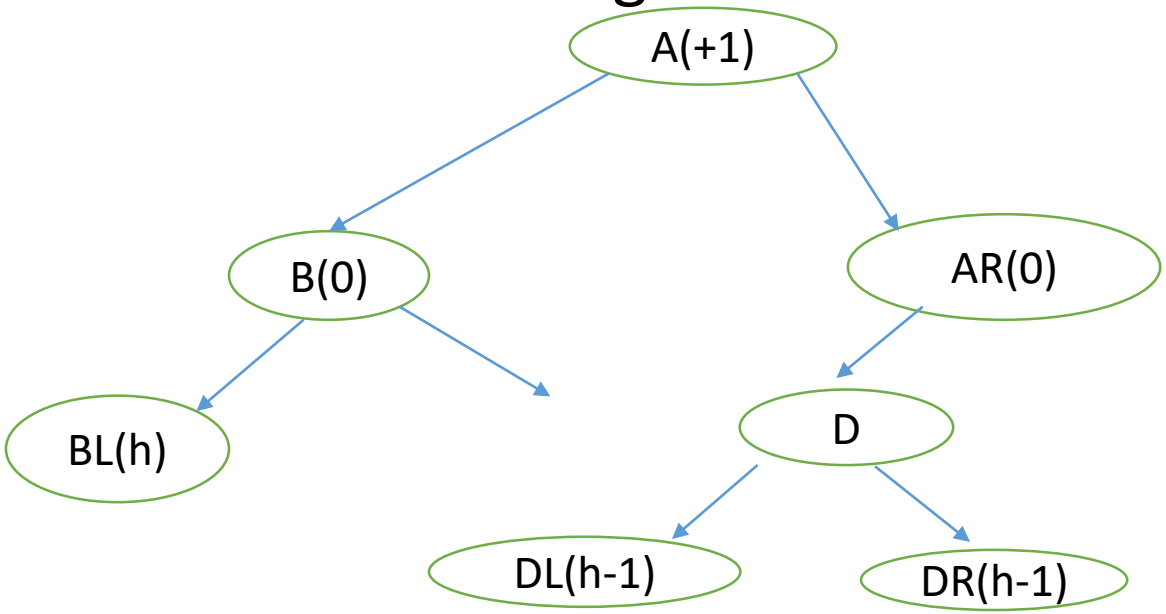
- If a node is added in the right or left subtree of C



- In this situation, to make it balanced we have to perform two rotations– first left rotation and then right rotation

RL Rotation

- If a node is added in the right or left subtree of D



- In this situation, to make it balanced we have to perform two rotations– first right rotation and then left rotation

Height of an AVL Tree

- The best height of BST is : O(log n)
- AVL tree is also a variation of BST so its best case is also also O(log n)
- Worst case of BST is : O(n)
- What is the worst case for AVL : O(log n)

Proof:

Proof..

1. Let us assume the height of the AVL tree to be *h* and the minimum number of nodes in the tree will be N_h.
2. Height of one of its immediate child's subtree would have a height of *h-1* and have N_{h-1} nodes.
3. Assume it is the right subtree.
4. In the worst case, the balance factor of the root will be -1, 1.
5. So the subtree of left child of the root has a height of *h-2* and has N_{h-2} nodes.
6. We now have a recurrence relation of form ($N_h = N_{h-1} + N_{h-2} + 1$).
7. Base cases: single node and two nodes
8. Let us also assume that the height of the single node (root) is 1 and height of two nodes is 2 ($N_1 = 1$;$N_2=2$)

- Now that we have a recurrence relation and two base cases, we can reduce them as :

  $N_h = N_{h-1} + N_{h-2} + 1$

  $N_h = (N_{h-2} + N_{h-3} + 1) + N_{h-2} + 1$ (as ($N_{h-1} = N_{h-2} + N_{h-3} + 1$) from the recurrence relation)

  $N_h > 2\ N_{h-2}$

  $N_h > 2^{h/2}$

  $\log (N_h) > \log (2^{h/2})$

  $2 \log (N_h) > h$

  $h = O (\log (N_h))$

Exercise

Q1. 40, 30, 50, 20, 5, 70, 65, 45

Make AVL tree for this data

Q2. If preorder and inorder traversals are given, can you make the binary tree preorder: a,b,d,c inorder: c,d,b,a

Q3. If postorder and inorder traversals are given, can you make the binary tree postorder: c, b,d,a inorder: c,b,a,d

Some exercises

- The postorder traversal of a binary tree is 8, 9, 6, 7, 4, 5, 2, 3, 1. The inorder traversal of the same tree is 8, 6, 9, 4, 7, 2, 5, 1, 3. The height of a tree is the length of the longest path from the root to any leaf. The height of the binary tree above is
- _4_.
- **Explanation:** Given, post-order – 8, 9, 6, 7, 4, 5, 2, 3, 1
  and in-order – 8, 6, 9, 4, 7, 2, 5, 1, 3
- Construct a binary tree from postorder and inorder traversal

- For the workshops organized in college, students can participate more than one workshop saying that all are organized in different timings. 150 students are enrolled in total for one or more than one workshop. Registration details are as follows :

99 students are enrolled for Cyber Security. 56 students are enrolled for Solar Energy, 89 students are enrolled for Artificial Intelligence. 25 students for both Cyber Security and Solar Energy. 65 students for both Cyber Security and Artificial Intelligence and 18 students for both Solar Energy and Artificial Intelligence. How many students are participating in all three workshops ?
(A) 12
(B) 16
(C) 10
(D) 14

Answer: (D)

Explanation: $n(A \cup B \cup C) = n(A) + n(B) + n(C) – [n(A \cap B) + n(A \cap C) + n(B \cap C)] + n(A \cap B \cap C)$
$150 = 99+56+89 – [25+65+18] + x$

- Which one of the following is the tightest upper bound that represents the time complexity of inserting an object into a binary search tree of n nodes?
  (A) O(1)
  (B) O(Logn)
  (C) O(n)
  (D) O(nLogn)

  Answer: (C)

  Explanation: To insert an element, we need to search for its place first. The search operation may take O(n) for a skewed tree