Data Structures & Algorithms

TREES

Prepared By:-

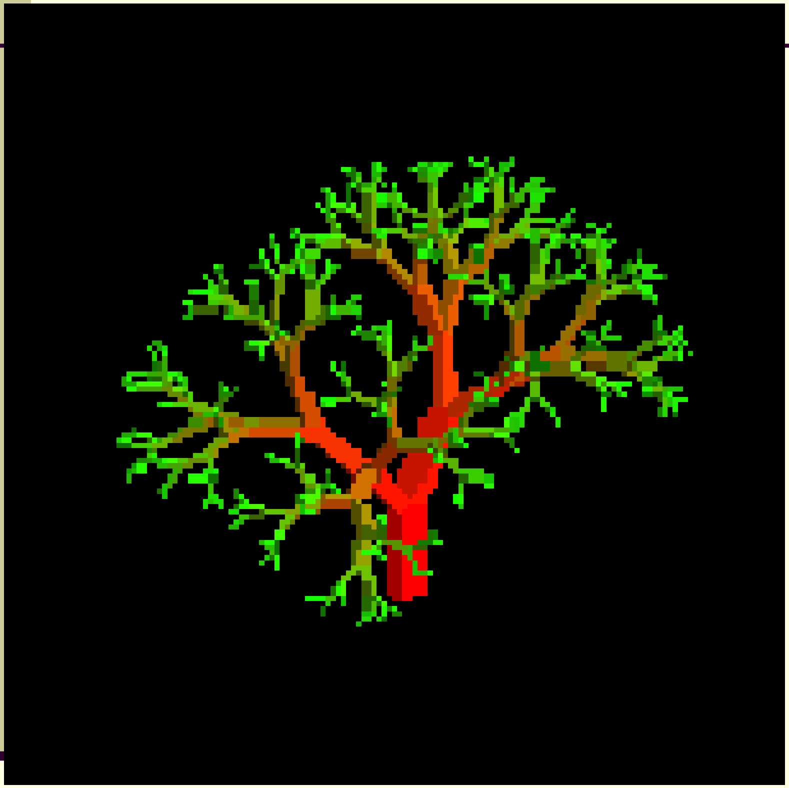
Dinesh Vashisht

Asstt. Professor, CSI Deptt.

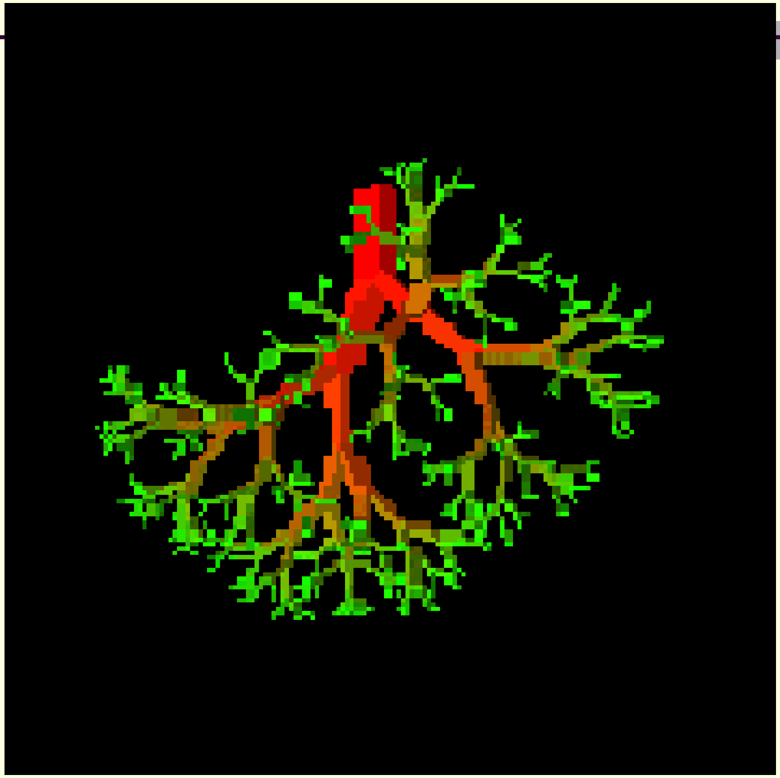
KIET Group of Institutions,

Ghaziabad

How We View a Tree



Nature Lovers View



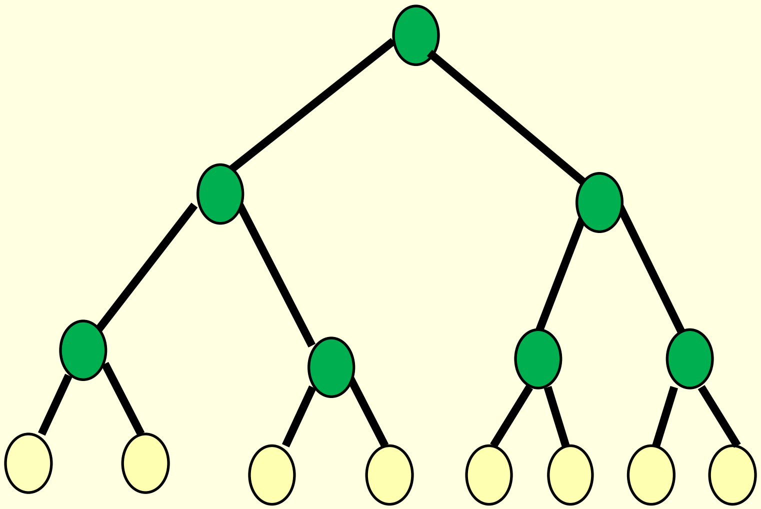
Computer Scientists View

Linear Lists And Trees

- Linear lists are useful for serially ordered data
  - $(e_0, e_1, e_2, \ldots, e_{n-1})$
  - Days of week, months in a year, students in this class
- Trees are useful for hierarchically ordered data
  - Employees of a corporation
    - President, vice presidents, managers, etc.

Tree : Example

- A is Root Node
- B is parent of D & E
- A is ancestor of D & E
- C is sibling to B
- D & E are children of B
- H,I,J,K,L,M,N and O are leaves



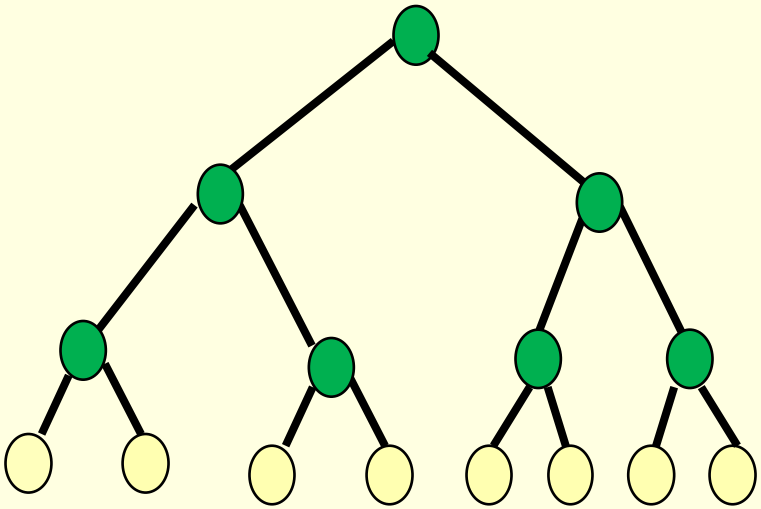
Tree : Example

- Level of a Node

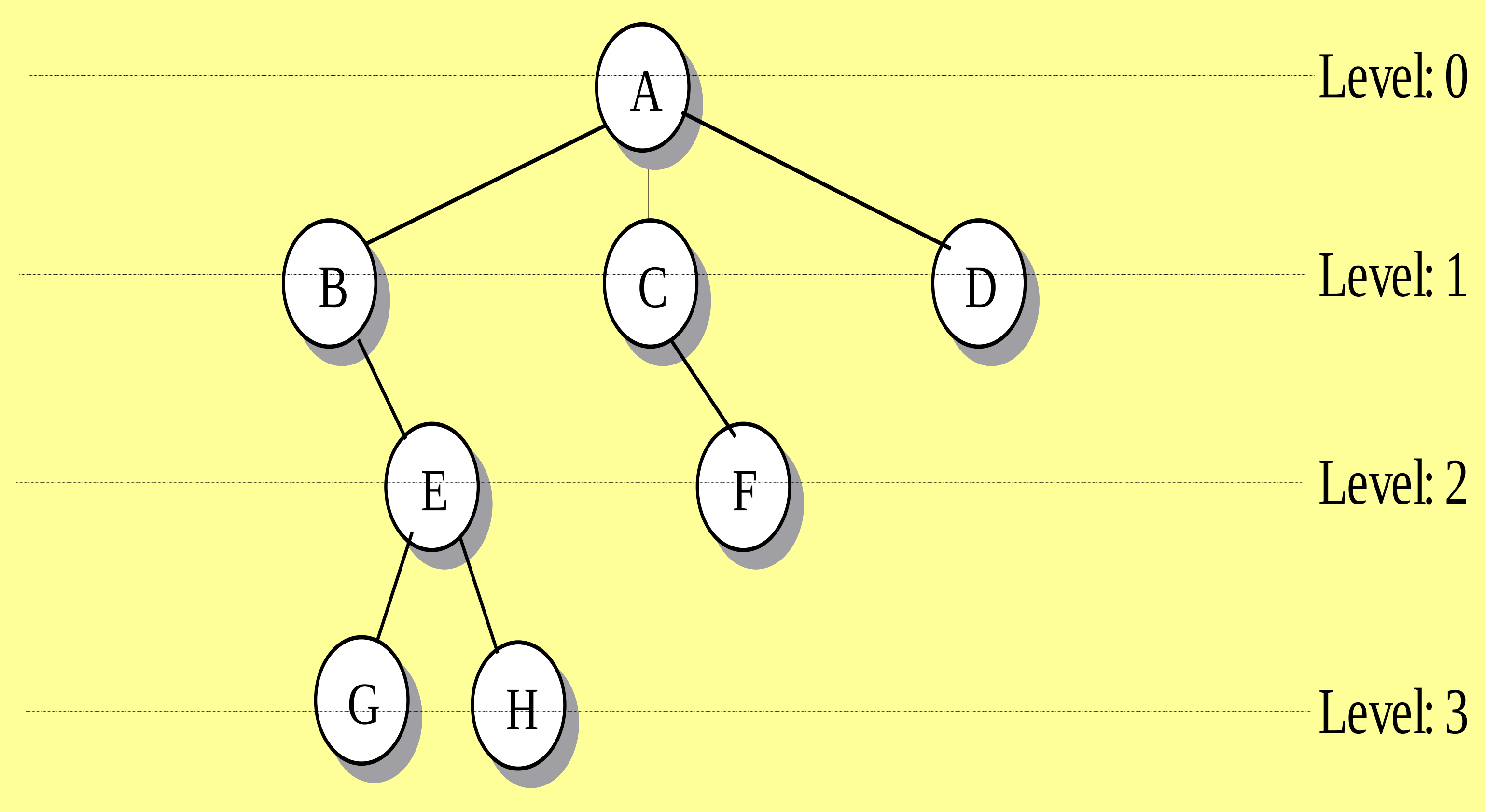
  Level of the root of a tree is 0, and the level of any other node in the tree is one more than the level of its parent

- Depth of a Tree

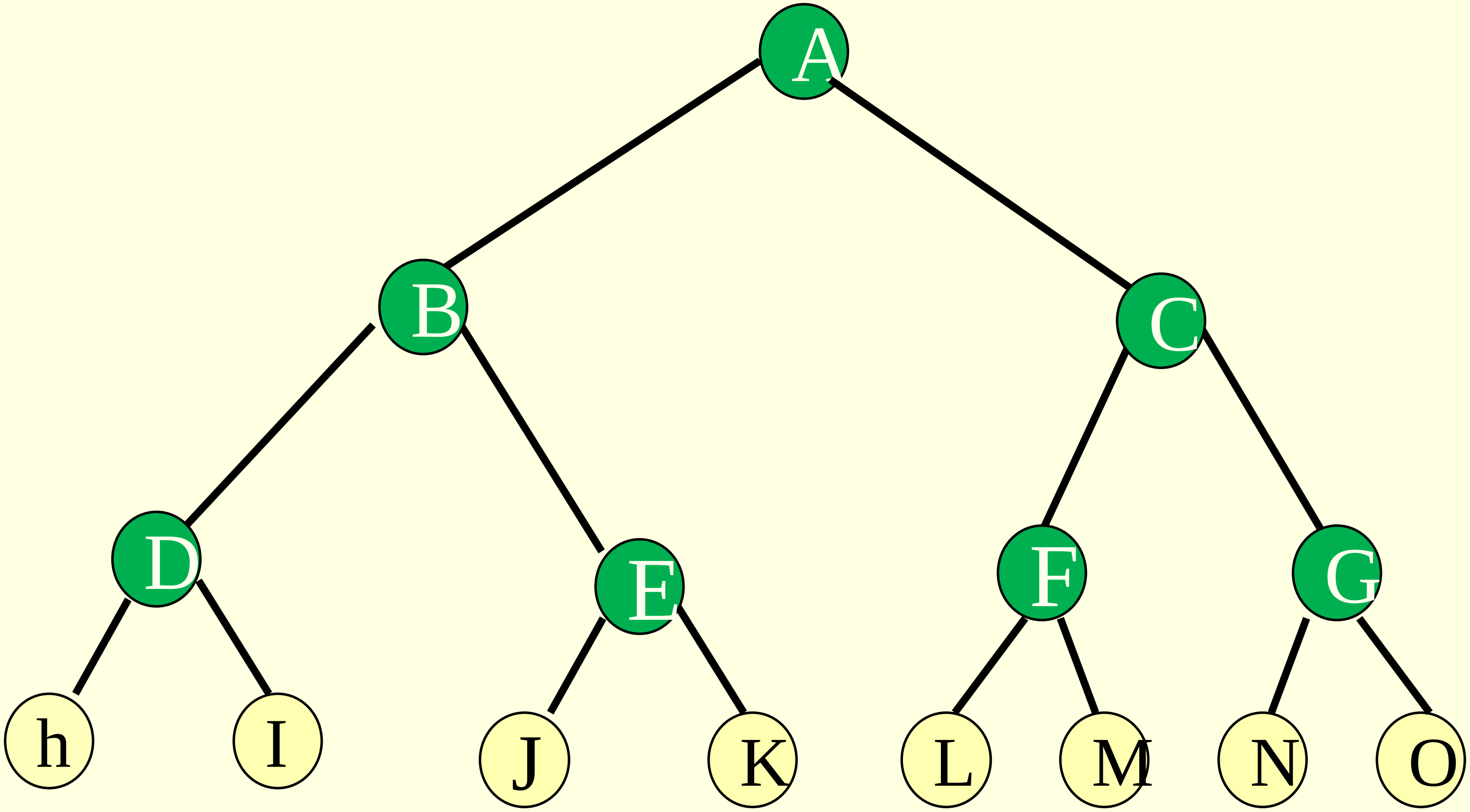
  The depth of a tree is the maximum level of any leaf in the tree (also called the height of the tree)



Levels in A Tree



Tree : Example



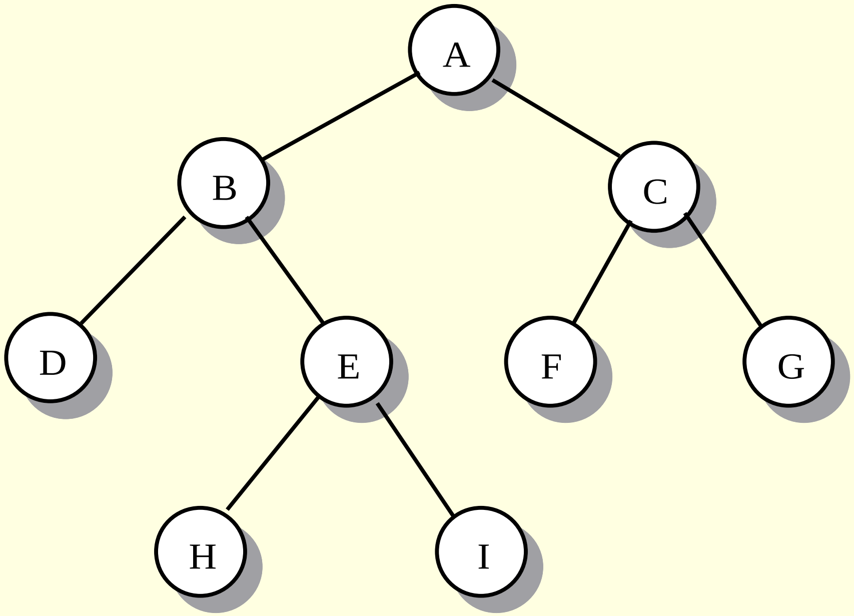
Tree : Definitions

- A tree `t` is a finite nonempty set of elements
- One of these elements is called the root
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of `t`.

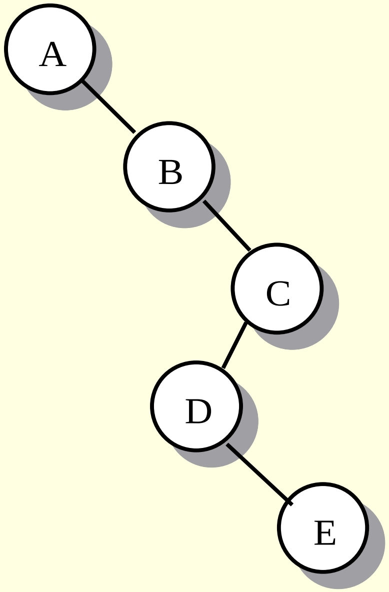
Binary Tree

- A binary tree T is a finite set of nodes with one of the following properties:
  - (a) T is a tree if the set of nodes is empty. (An empty tree is a tree.)
  - (b) The set consists of a root, R, and exactly two distinct binary trees, the left subtree T_L and the right subtreeT_R. The nodes in T consist of node R and all the nodes in T_L and T_R.
- **In a binary tree, the maximum degree of any node is two**

Binary tree Examples



Tree A
Size 9 Depth 3



Tree B
Size 5 Depth 4

Tree Vs Binary Tree

- No node in a binary tree may have a degree more than 2, whereas there is no limit on the degree of a node in a tree

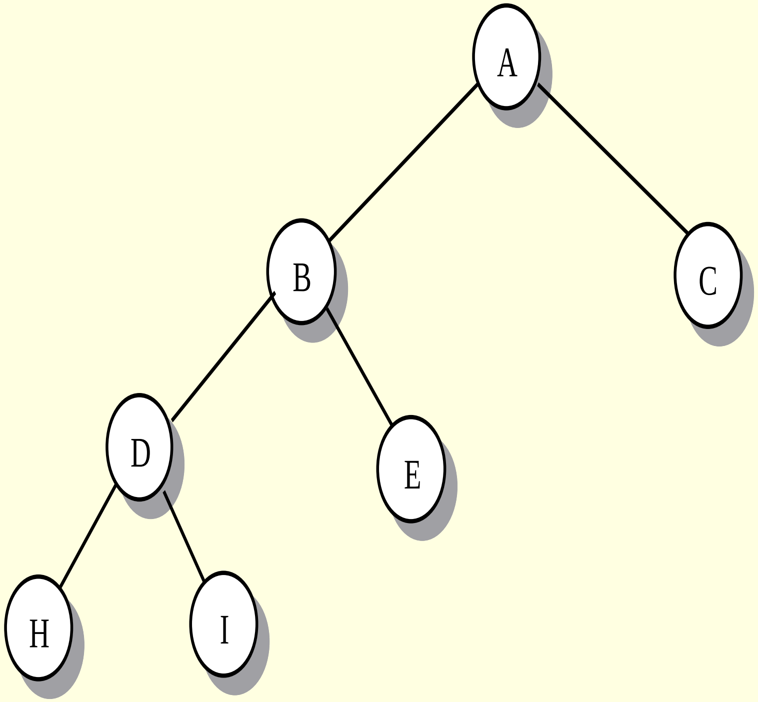
Tree Vs Binary Tree

- The sub trees of a binary tree are ordered; those of a tree are not ordered.

- Are different when viewed as binary trees
- Are the same when viewed as trees

Strictly Binary Tree

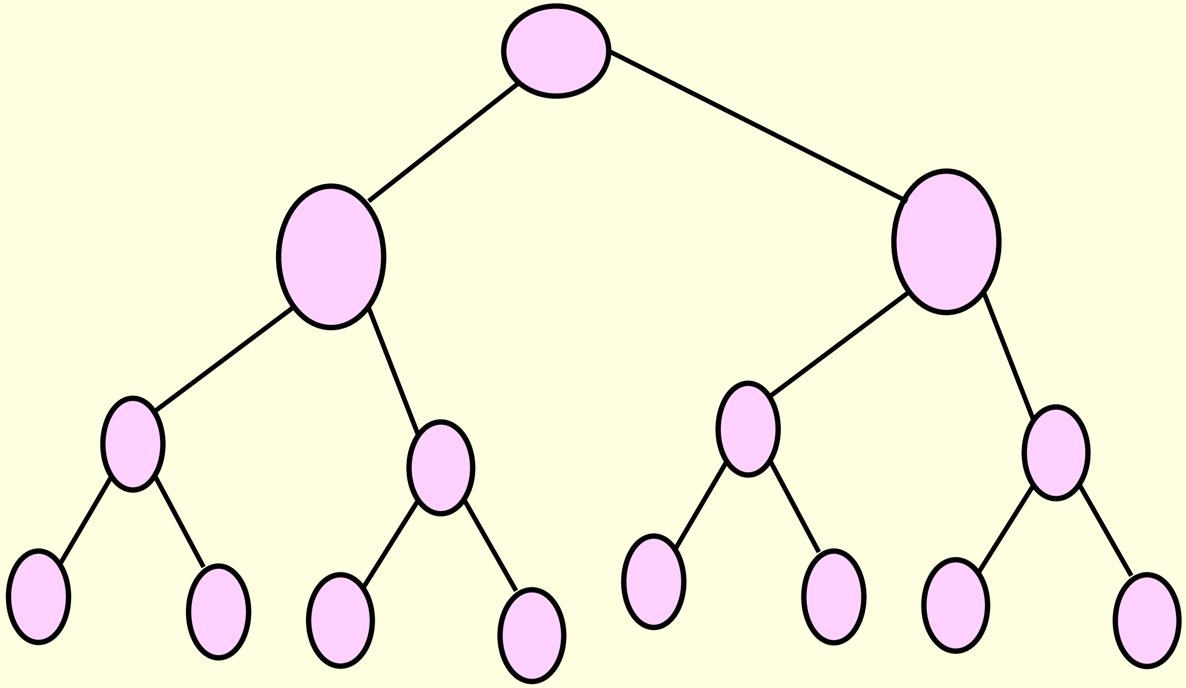
- A binary tree is called *strictly binary tree* if every nonleaf node in the tree has nonempty left and right subtrees
  - i.e., every nonleaf node has two children.



Strictly Binary Tree

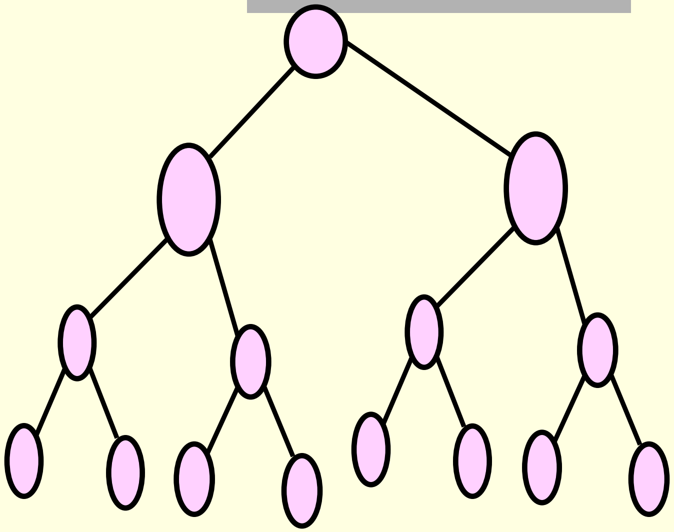
Complete Binary Tree

- A *complete binary tree* of depth *d* is a strictly binary tree with all leaf nodes at level *d*.



Complete binary Tree

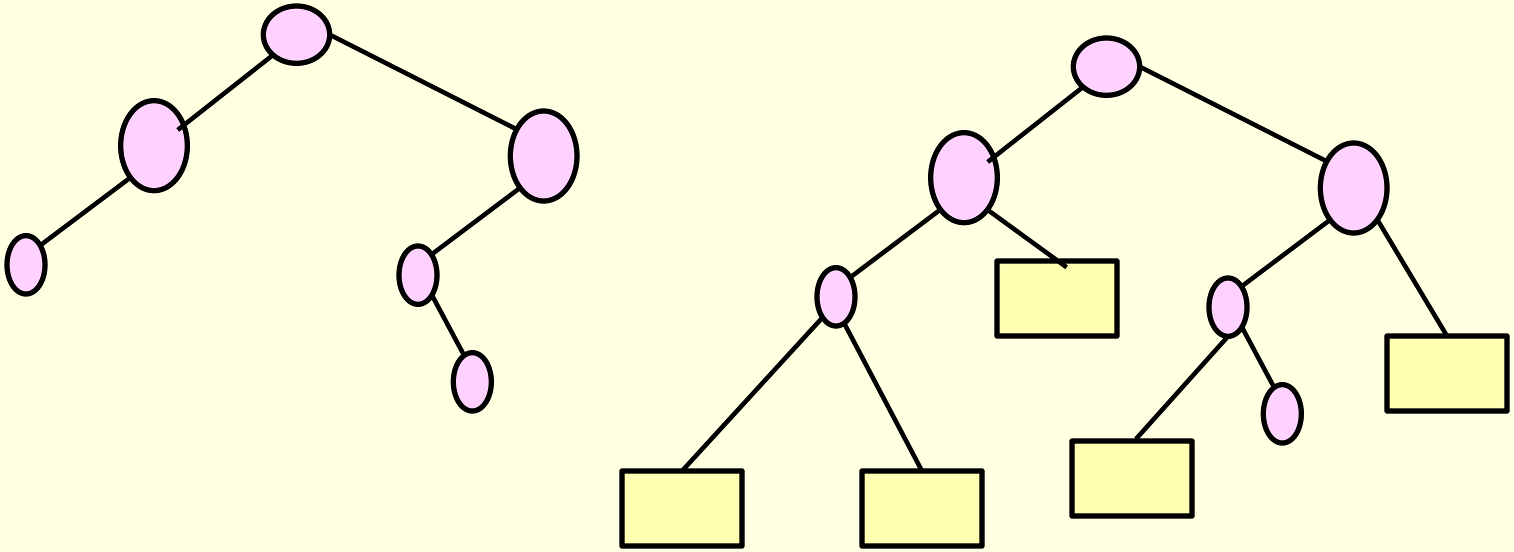
- Level i has 2^i nodes
- In a tree of height h
  - Leaves are at level h
  - No of leaves is 2^h
  - No of internal nodes $=1 + 2 + 2^2 ... 2^{h-1} = 2^h - 1$
  - No of internal nodes = No of leaves – 1
  - Total no of nodes $= 2^{h+1}-1 = N$
- In a tree of n nodes
  - No of leaves is $(n+1)/2$



Extended Binary Tree

- A binary tree T is said to be a 2-tree or extended binary tree if each node N has either 0 or 2 children.
- In such a case node with 2 children are called internal nodes, and nodes with 0 children are called external nodes. sometimes nodes are distinguished in diagram by using circles foe internal nodes and rectangles for external nodes.

Extended Binary Tree



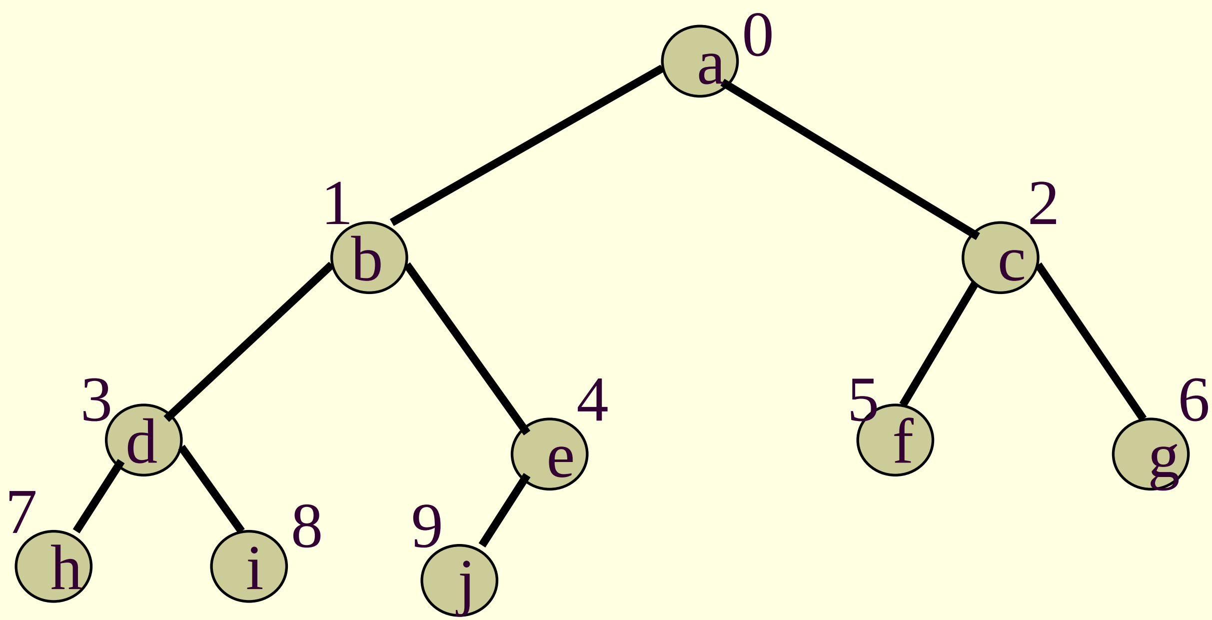
Binary Tree Representation

- Again, there are two ways to implement a tree data structure:
  - Array representation
  - Linked representation

Array Representation of Tree

- Father(n) is at floor(n-1)/2 if n not equal to 0. if n equal to 0 then it is the root and has no father.
- Lchild(n) is at (2n+1)
- Rchild(n) is at (2n+2)
- Siblings: if the left child at index n is given then its right sibling is at (n+1)

Array Representation of Tree

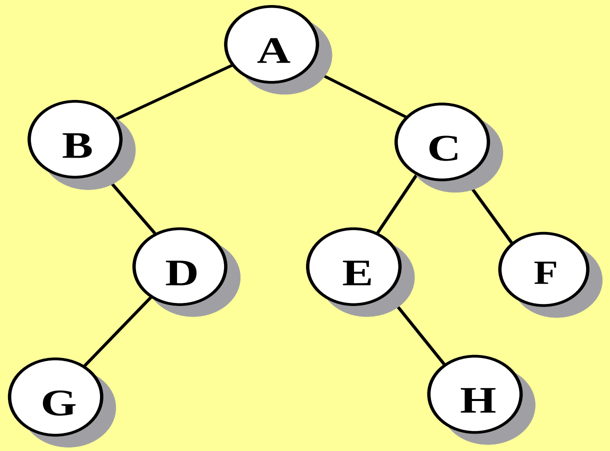


a	b	c	d	e	f	g	h	i	j	
0	1	2	3	4	5	6	7	8	9	

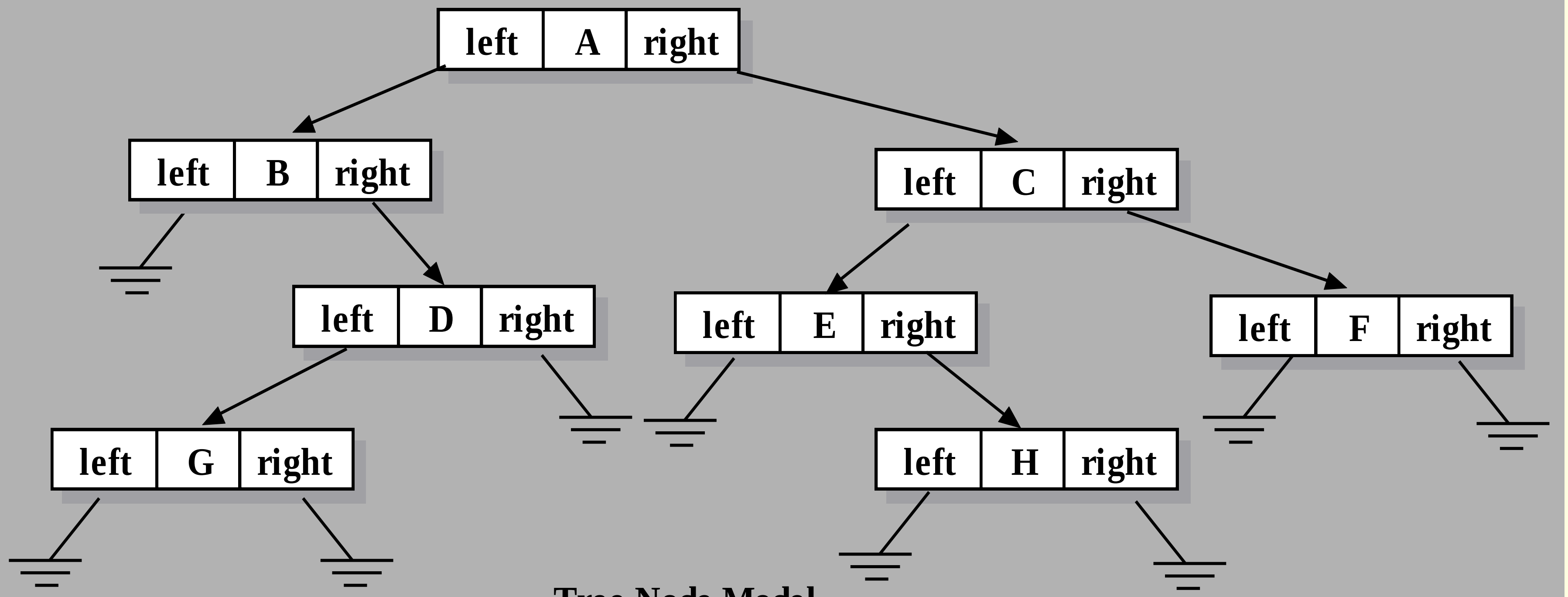
Linked representation of Tree

```
struct node
{
    char data;
    struct node *rchild;
    struct node *lchild;
} ;
typedef struct node NODE;
NODE *ptr;
ptr=(NODE *)malloc(sizeof(NODE));
```

Binary Tree Representation



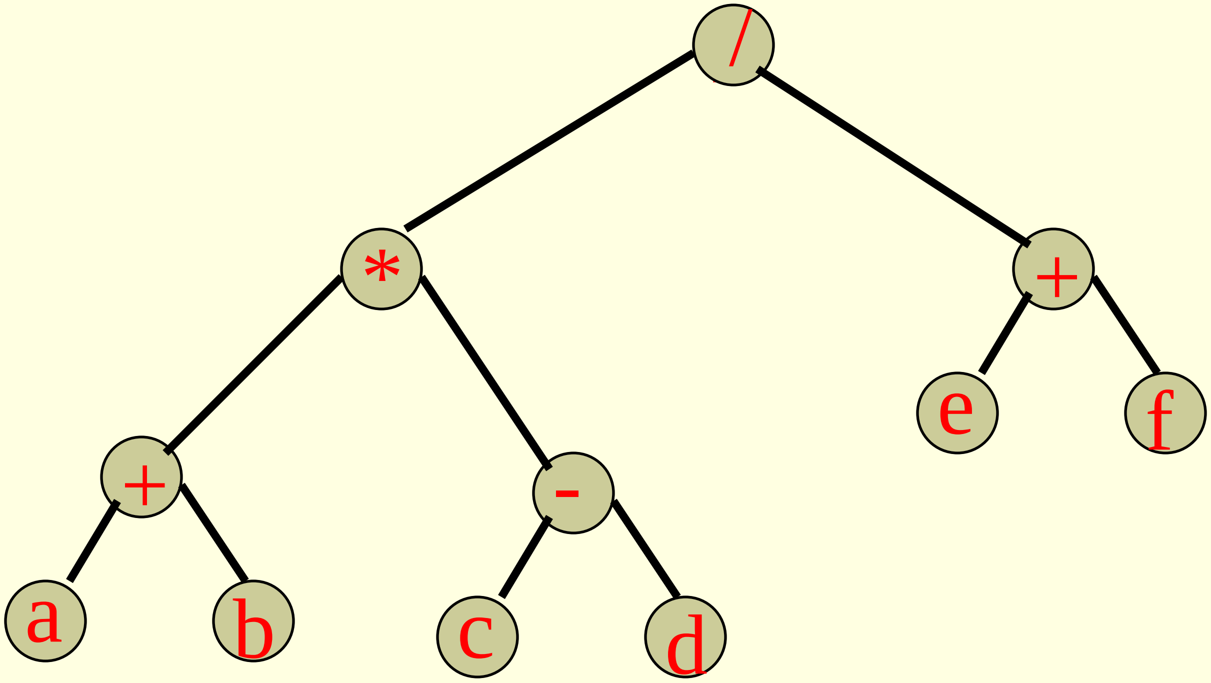
Abstract Tree Model



Tree Node Model

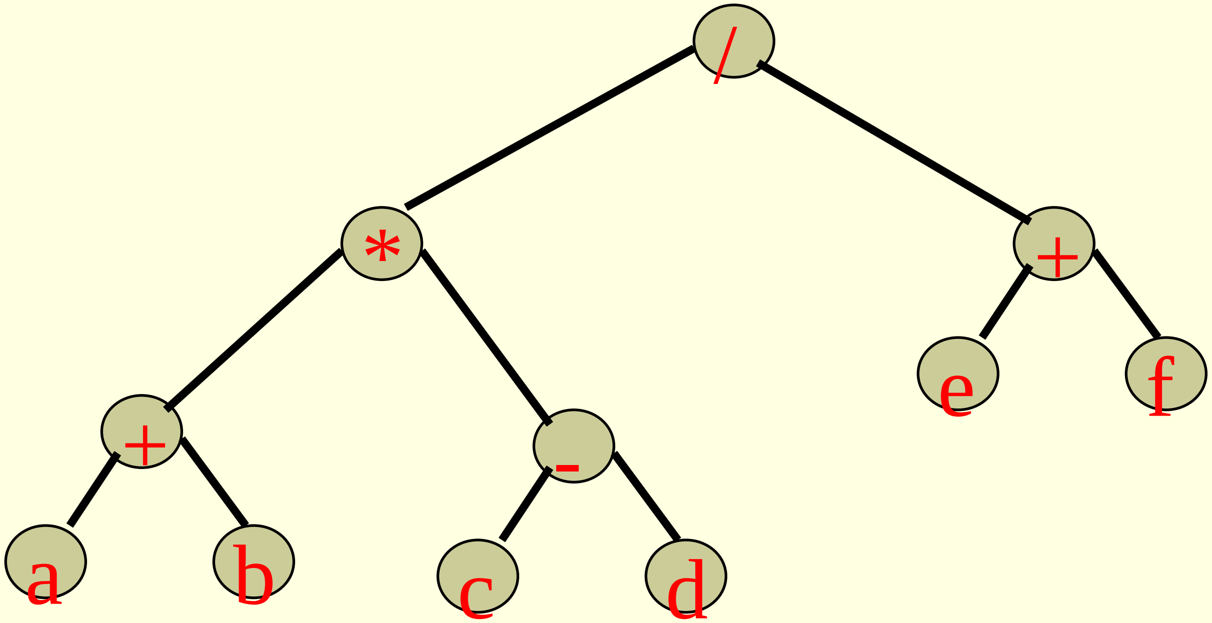
Binary Tree Form

- (a + b) * (c – d) / (e + f)



Merits Of Binary Tree Form

- Left and right operands are easy to visualize
- Code optimization algorithms work with the binary tree form of an expression
- Simple recursive evaluation of expression



Binary Tree Traversal

- Many binary tree operations are done by performing a traversal of the binary tree
- In a traversal, each element of the binary tree is visited exactly once

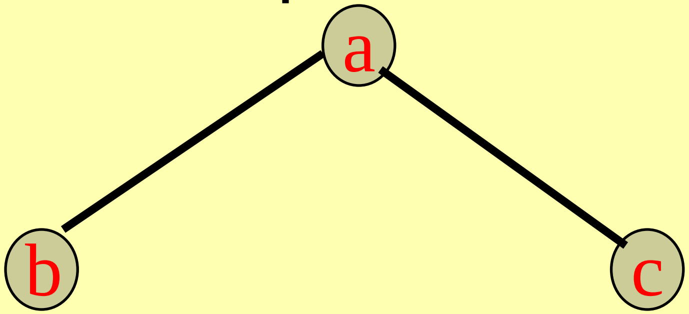
1. **Preorder (Root, Left, Right)**
2. **Inorder (left, Root, Right)**
3. **Postorder (Left, Right, Root)**
4. **Level order**

Preorder Traversal (visit = print)

(Root, Left, Right)

1. Visit the root node
2. Traverse the left subtree in preorder(L)
3. Traverse the right subtree in preorder(R)

- Example

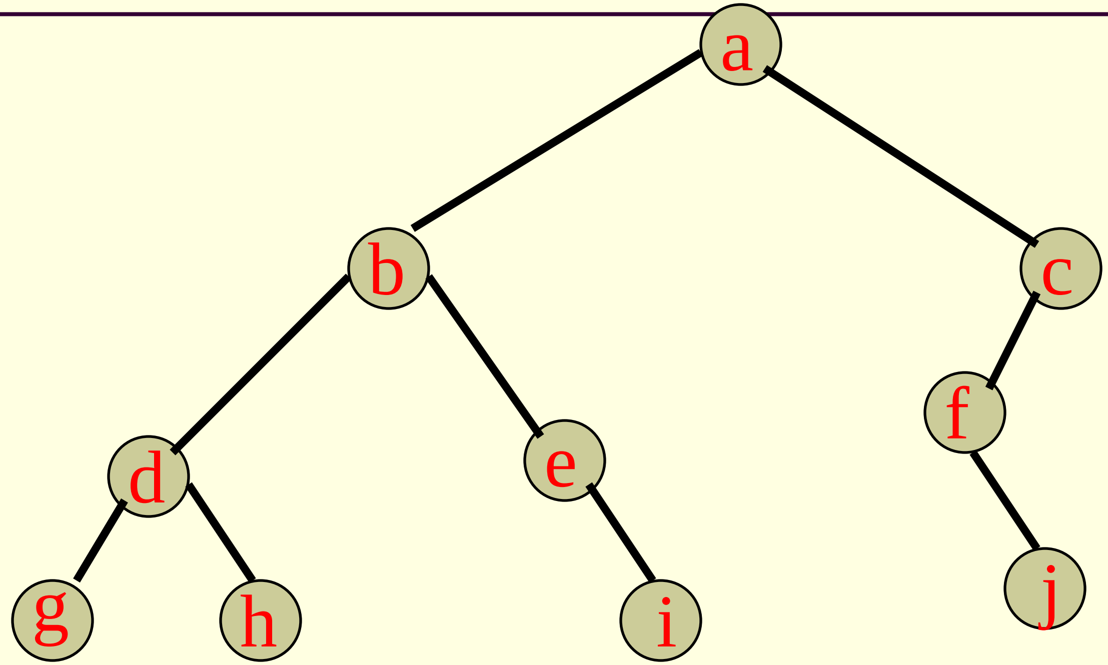


a b c

Preorder Traversal

```
Void Preorder(node *p)
{
    If(p != NULL)
    {
            printf(“%c”, p - > info);
            Preorder(p - >left);
            Preorder(p - >right);
    }
 }
```

Preorder Example (visit = print)



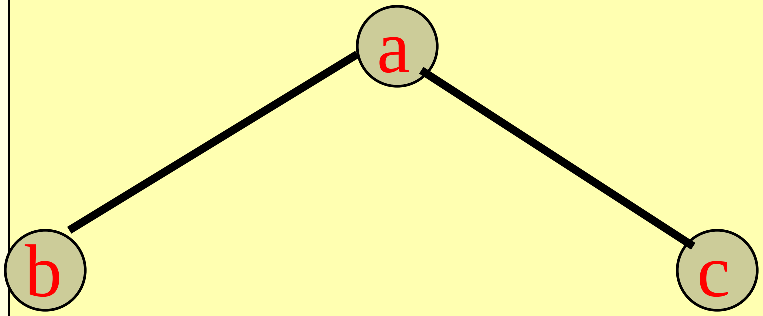
a b d g h e i c f j

Inorder Example (visit = print)

(Left, Root, Right)

1. Traverse the left subtree in inorder(L)
2. Visit the root node
3. Traverse the right subtree in inorder(R)

- Example

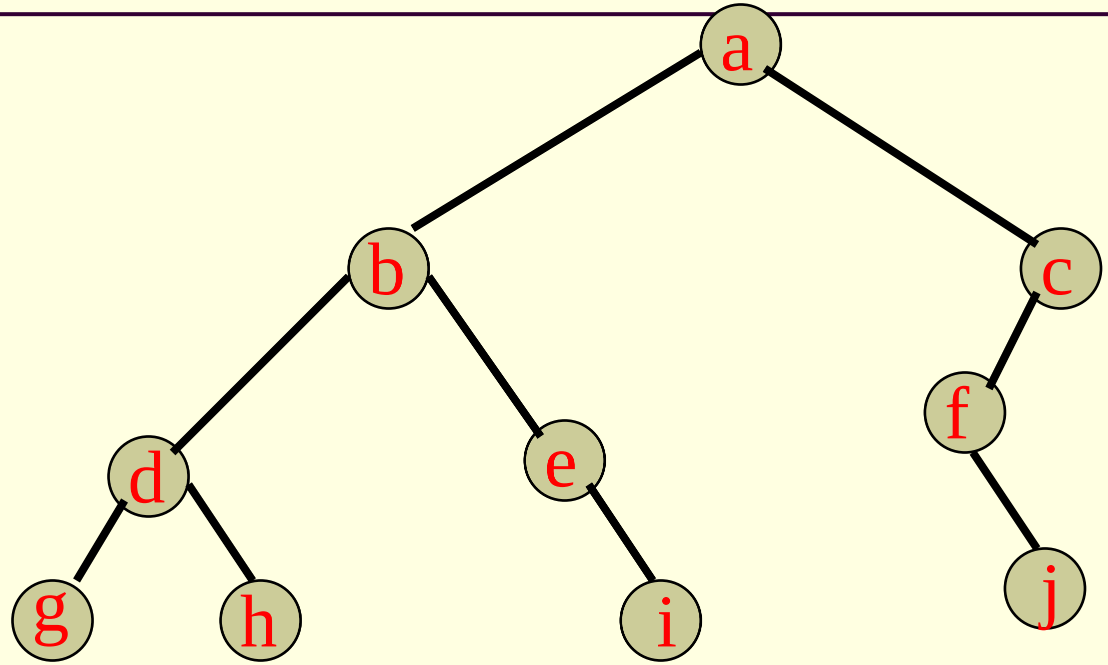


b a c

Inorder Traversal(LRR)

```
Void Inorder(node *p)
{
    If(p != NULL)
    {
            inorder(p - >left);
            printf("%c", p - > info);
            inorder(p - >right);
    }
 }
```

Inorder Example (visit = print)

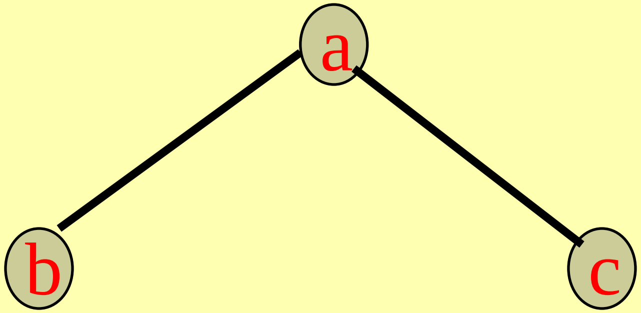


g d h b e i a f j c

Postorder Example (visit = print)

(Left, Right, Root)

1. Traverse the left subtree in postorder(L)
2. Traverse the right subtree in inorder(R)
3. Visit the root node

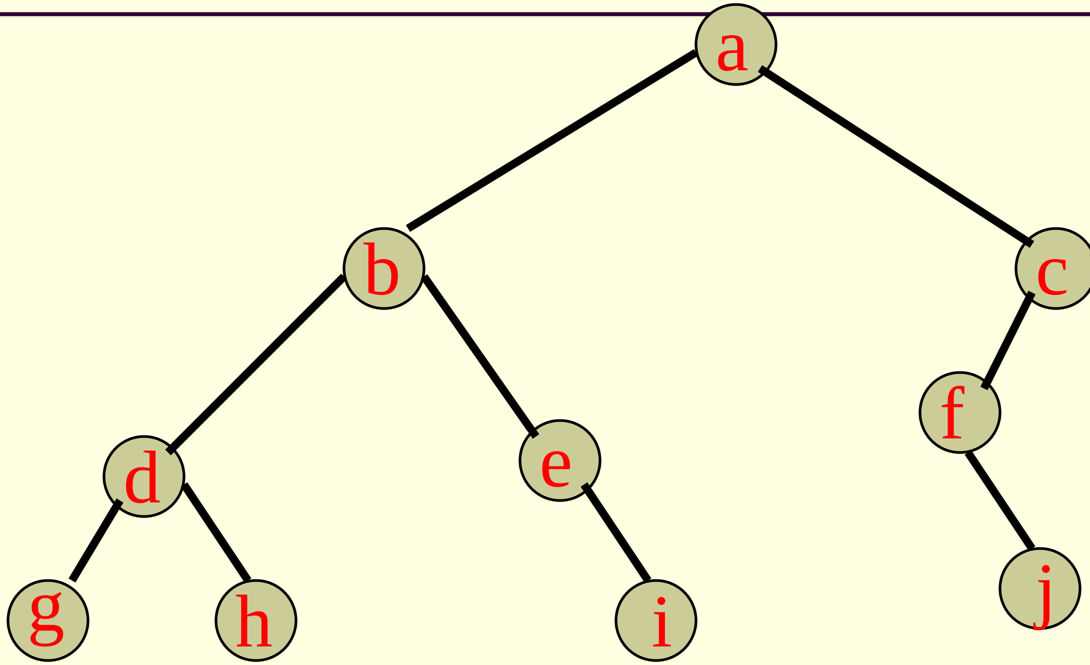


b c a

Postorder Traversal

```
Void postorder(node *p)
  {
      If(p != NULL)
      {
               postorder(p - >left);
               postorder(p - >right);
              printf(“%c”, p - > info);
      }
   }
```

Postorder Example (visit = print)



g h d i e b j f c a

Creating a tree from pre order and inorder traversal

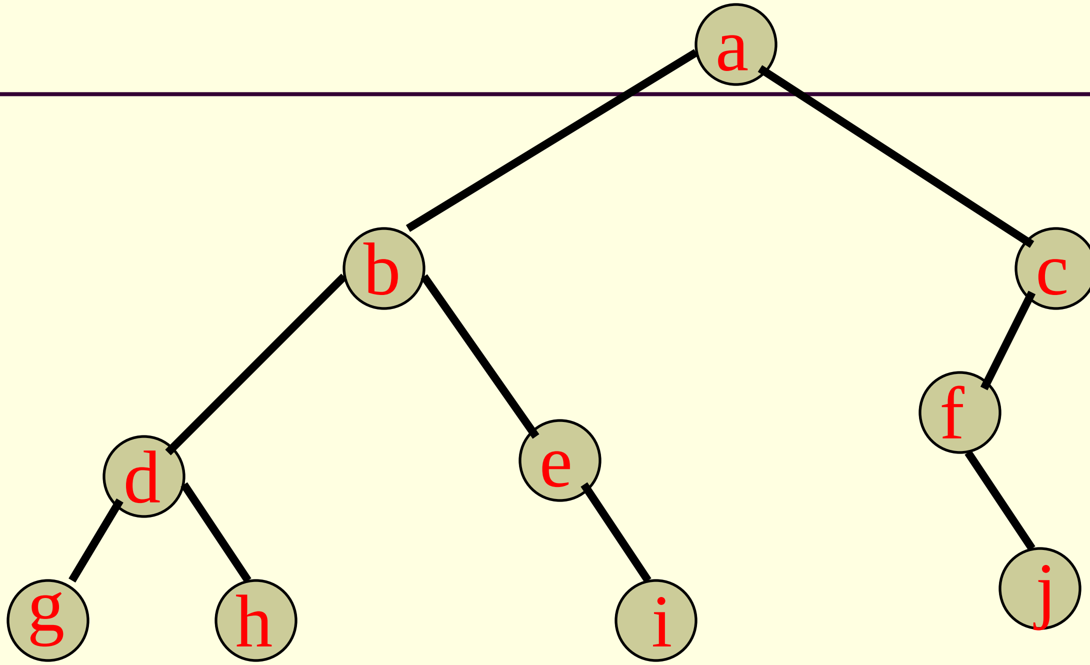
Preorder : a b c d f g e

Inorder : c b f d g a e

Creating a tree from post order and inorder traversal

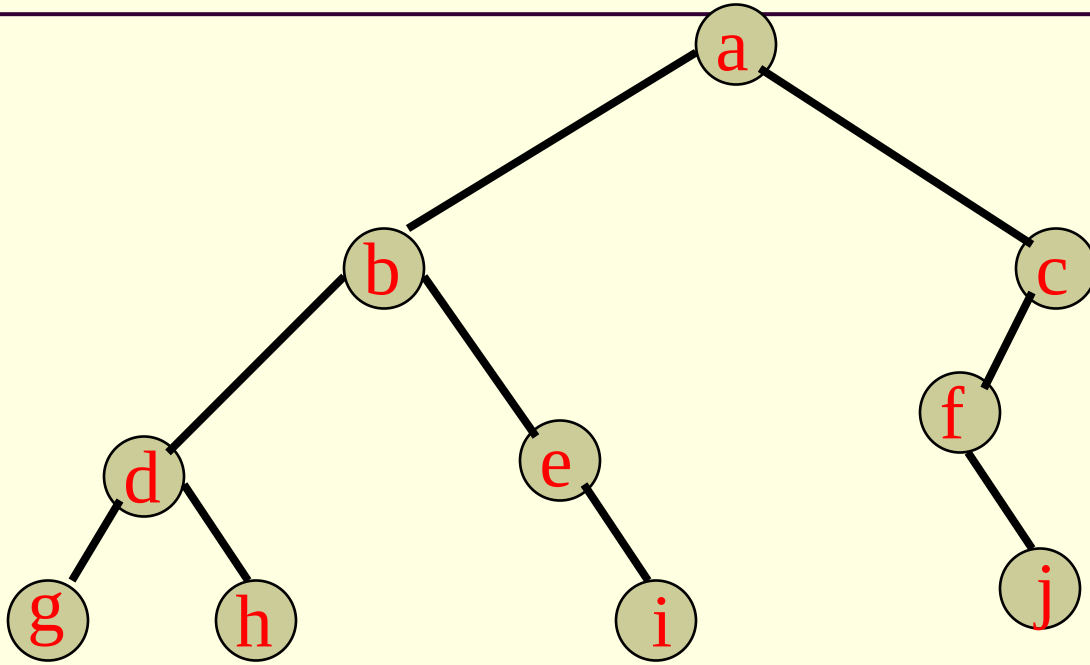
Home assignment

Traversal Applications



- Determine height
- Determine number of nodes

Level-Order Example (visit = print)



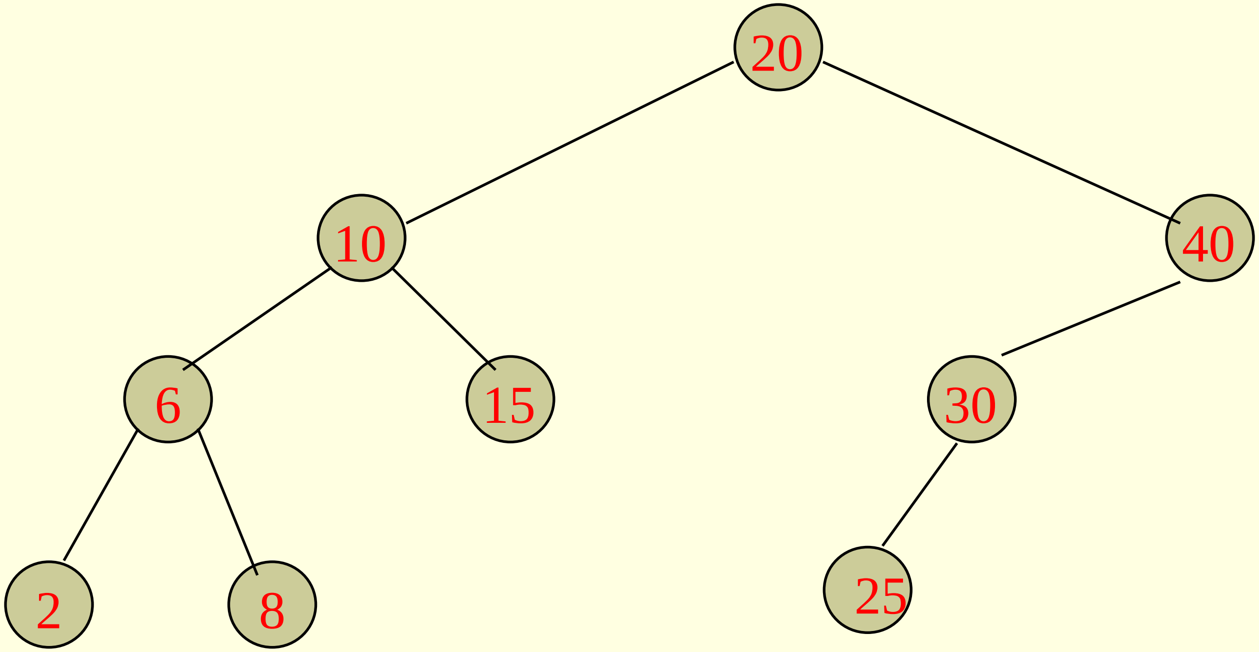
a b c d e f g h i j

Binary Search Tree

- A BST is a binary tree which is either empty or satisfies the following conditions

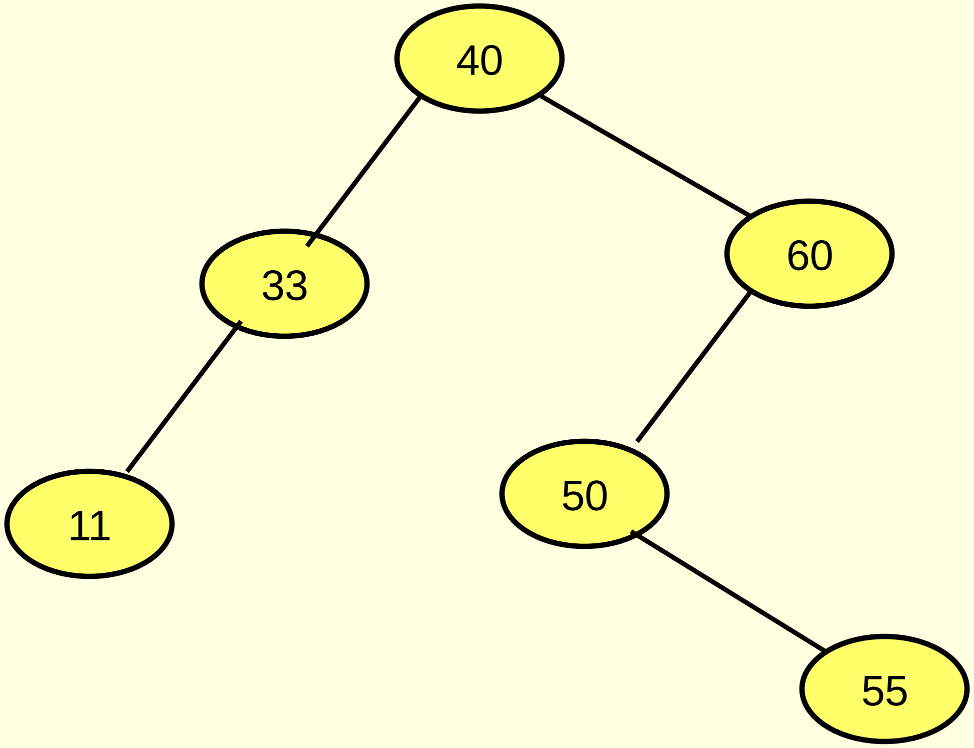
1. The value of the key in the left chlid or left subtree is less than the value of the root.
2. The value of the key in the right child or right subtree is more than the value of the root.

Binary Search Tree



Insertion/creation of a BST

Given nodes: 40,60,50,33,55,11



Insertion in a BST

Algo: struct node * insert(struct node *p,int digit)

Step 1: [check if tree is empty]

```
if (p==NULL) then
        {p=node * malloc(sizeof(node))
        p->left=p->right=NULL
        P->info=digit
        return(p);}
```

Step 2: if(digit<p->info) then p->left=insert(p->left,digit)

Step 3: if(digit> p->info) then p->right=insert(p->right,digit)

Step 4: if(digit==p->info) then print(“duplicate nodes)

Step 5: [Exit]

Searching a node in BST

Algo: Void search(struct node *p, int digit)

Step 1: [check if tree is empty]

```
if (p==NULL) then
        {Print("node does not exist")
return();}
```

Step 2: if(digit = p->info) then printf(digit)

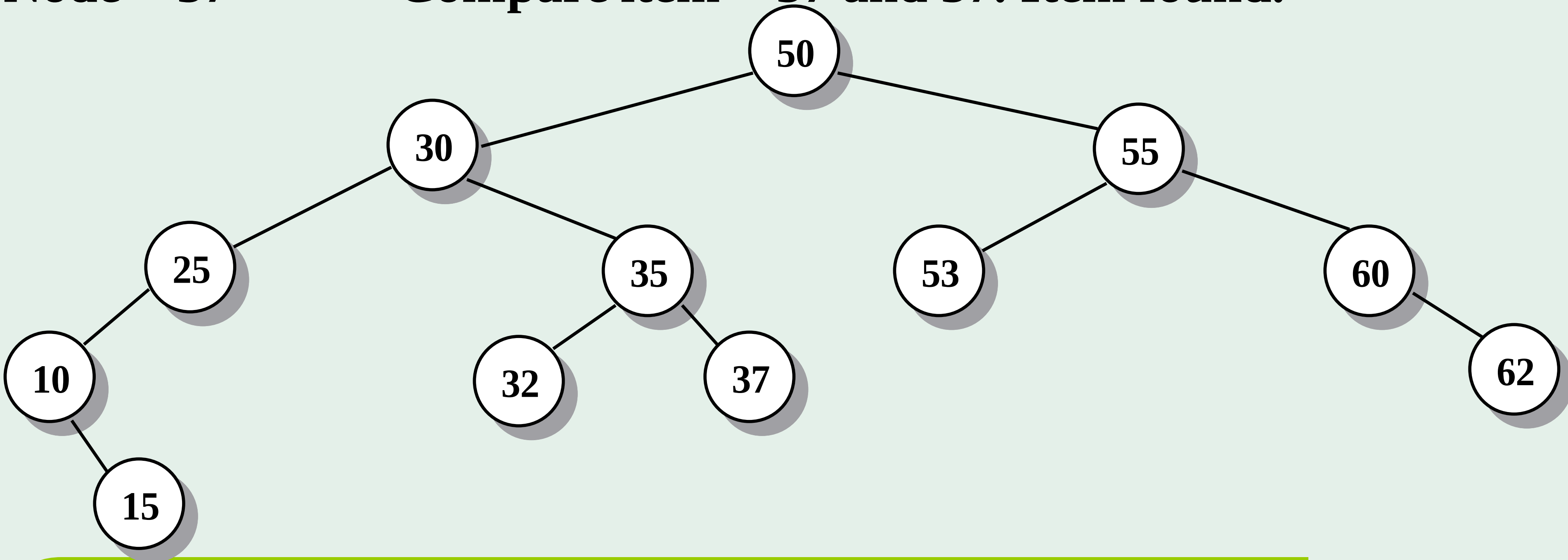
Step 3: if(digit< p->info)

```
then    search(p->left,digit)
else    search(p->right, digit)
```

Step 5: [Exit]

-LOCATING DATA IN A TREE-

Current Node	Action
Root = 50	Compare item = 37 and 50
	37 < 50, move to the left subtree
Node = 30	Compare item = 37 and 30
	37 > 30, move to the right subtree
Node = 35	Compare item = 37 and 35
	37 > 35, move to the right subtree
Node = 37	Compare item = 37 and 37. Item found.

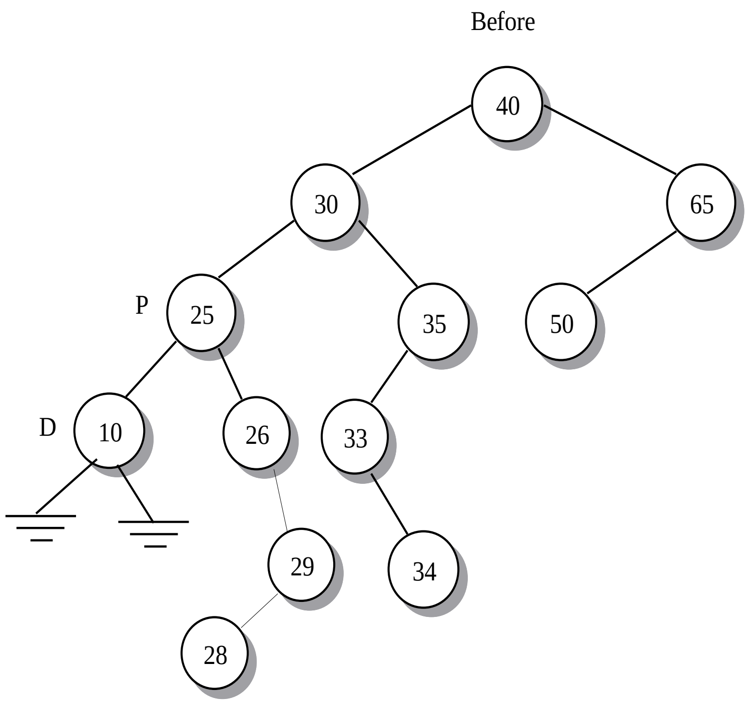


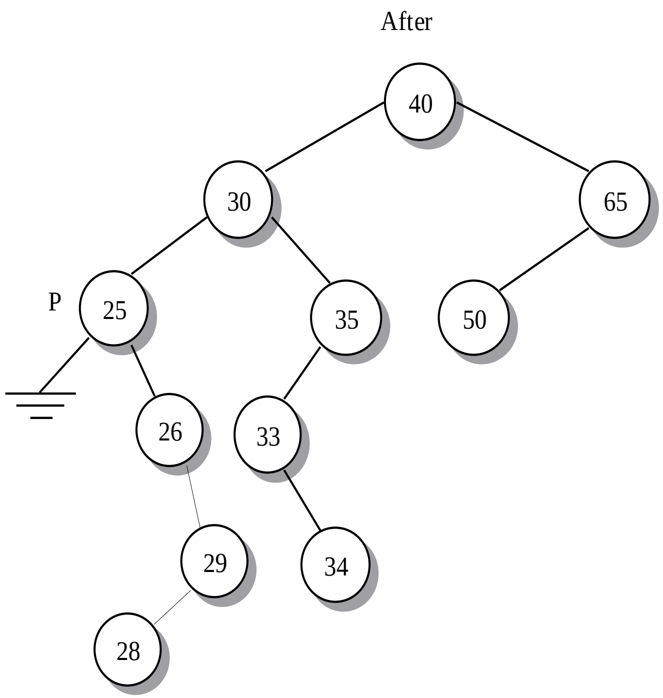
Deleting a node from BST

There can be following cases:

1. Delete a node with no child
2. Delete a node with a single child(either left or right but not both)
3. Delete a node with both children

Removing an Item From a Binary Tree

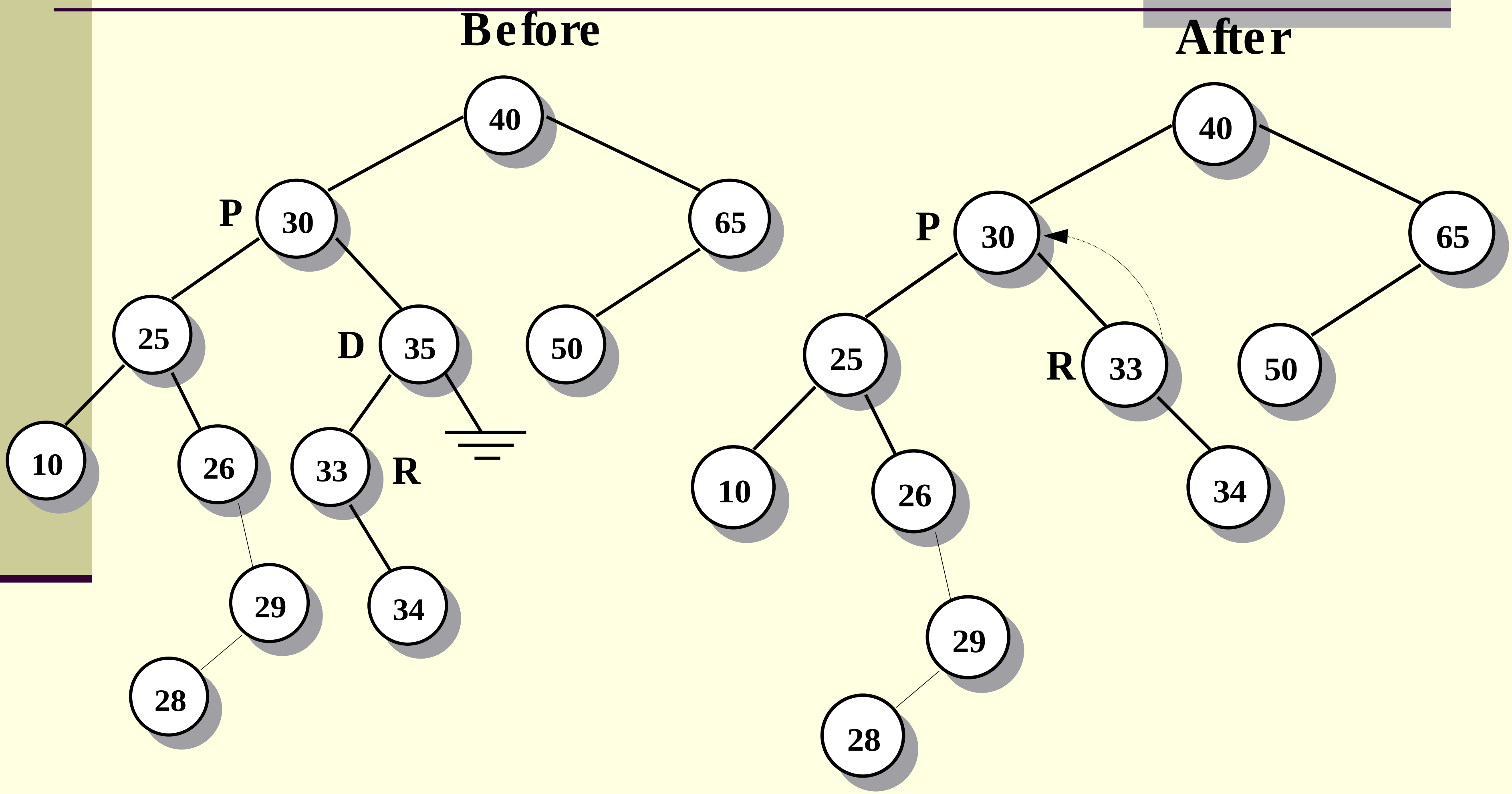




Delete leaf node 10.
pNodePtr->left is dNode

No replacement is necessary.
pNodePtr->left is NULL

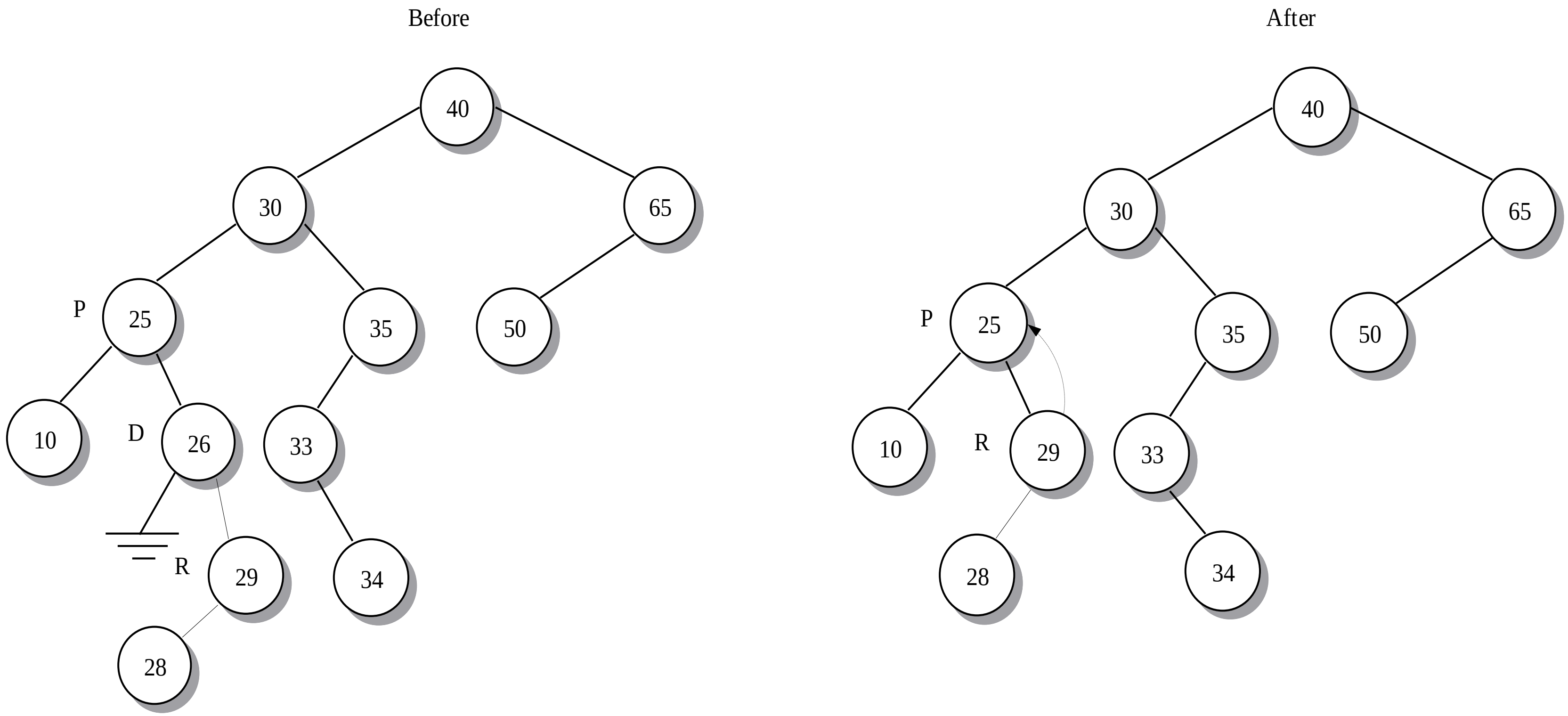
Removing an Item From a Binary Tree



Delete node 35 with only a left child: Node R is the left child.

Attach node R to the parent.

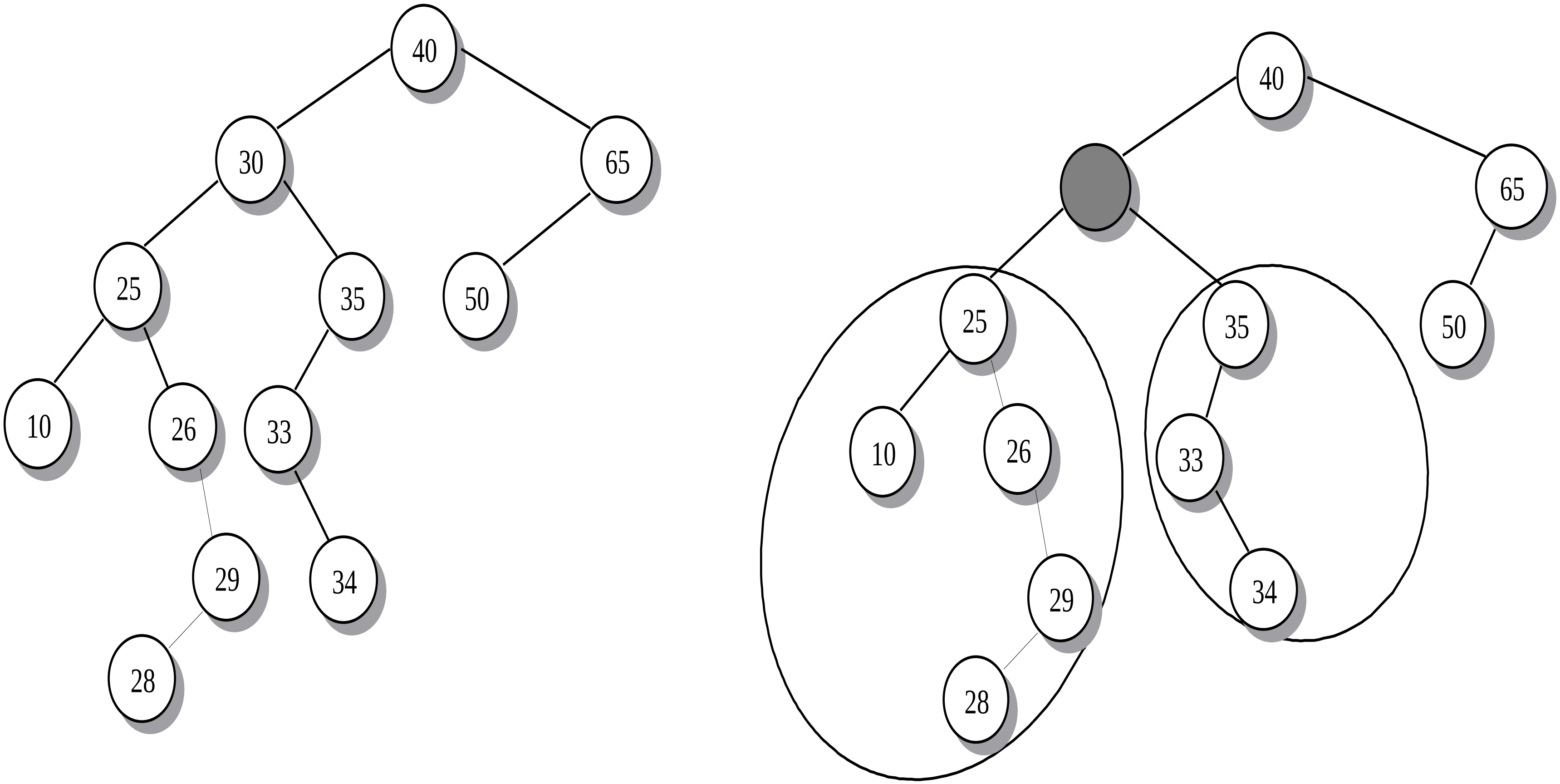
Removing an Item From a Binary Tree



Delete node 26 with only a right child: Node R is the right child.

Attach node R to the parent.

Delete a node with two child



Delete node 30 with two children.

Orphaned subtrees.

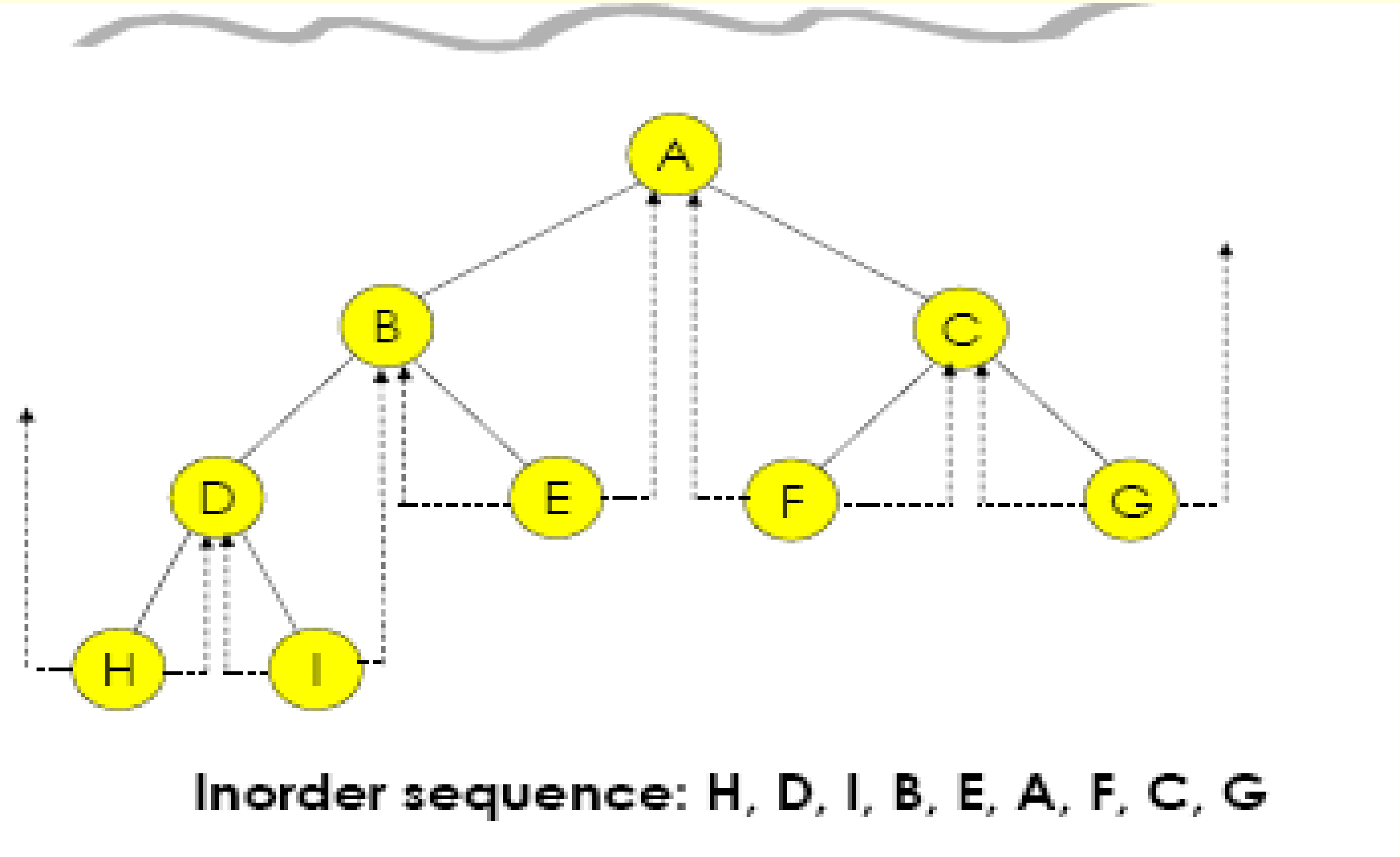
Program to implement Binary Search Tree

Threaded Binary Tree

Problem: **There are more null links than actual points**

- How to make use of these null links?
  - Threads !
- Threading rules
  - Ifptr>left_child=NULL
    - ptr>left_child = inorder predecessor of ptr
  - If ptr>right_child = NULL
    - ptr>right_child = inorder successor of ptr

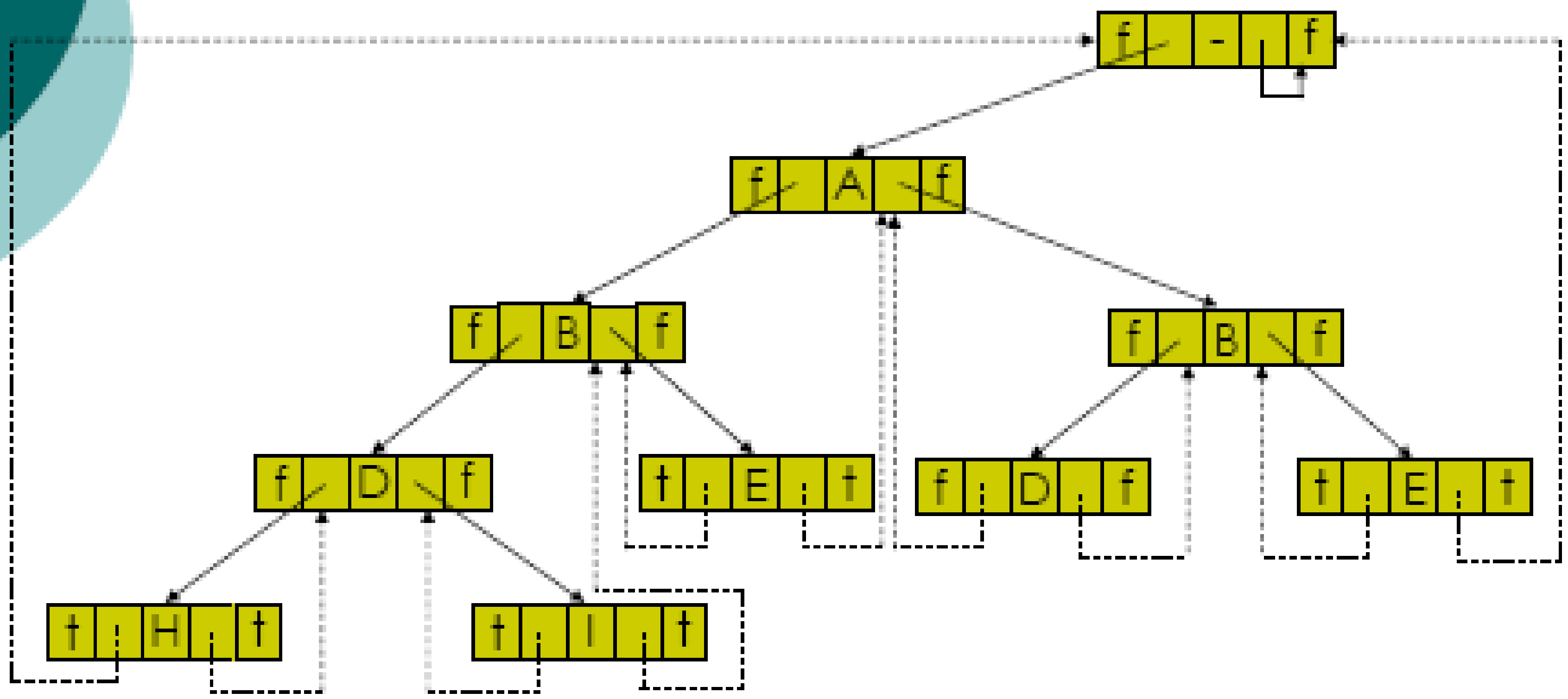
Threaded Binary Tree



Threaded Binary Tree

- To avoid dangling threads, a head node is used in representing a binary tree
- The original tree becomes the left subtree of the head node
- Empty Binary Tree

Threaded Tree :Memory representation



Summary

Tree:----

- hierarchical structures that place elements in nodes along branches that originate from a root.
- Nodes in a tree are subdivided into levels in which the topmost level holds the root node.
- Any node in a tree may have multiple successors at the next level. Hence a tree is a non-linear structure.
- Tree terminology with which you should be familiar:

  parent | child | descendents | leaf node | interior node | subtree.

Queries

Thanks!!!