TREES

Trees

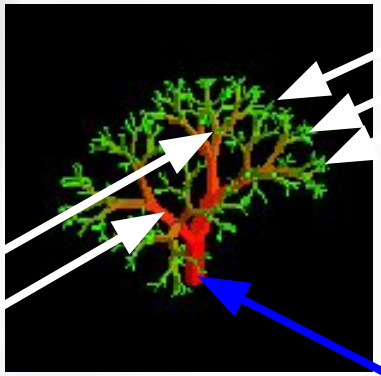




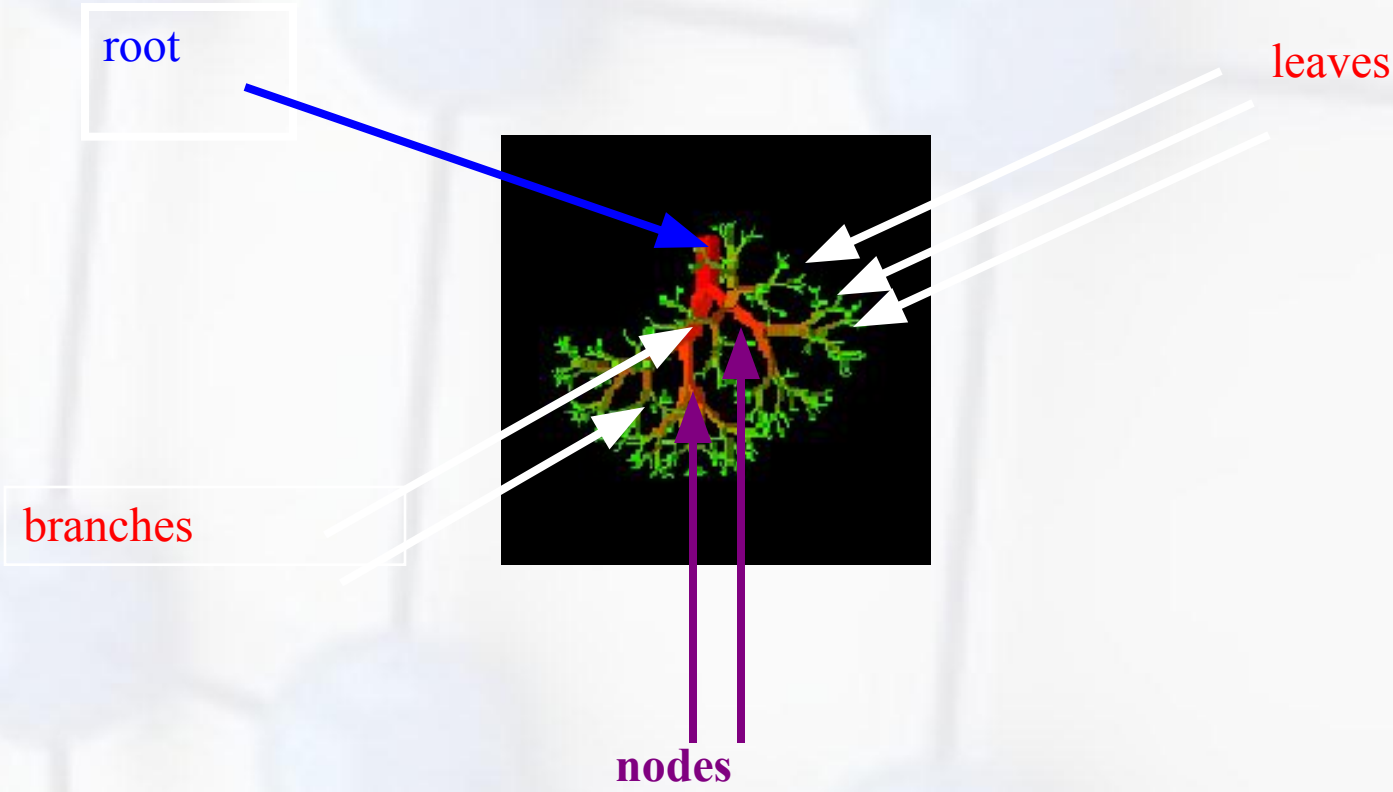





Nature Lover's View Of A Tree



Computer Scientist's View



Trees

- A tree is recursively defined as a set of one or more nodes where one node is designated as the root of the tree and all the remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root.
- Types of Trees
  - General Trees
  - Forests
  - Binary Trees
  - Expression Trees
  - Tournament Trees

Applications of Trees

- Trees are used to store simple as well as complex data. Here simple means an int value, char value and complex data (structure).
- Trees are often used for implementing other types of data structures like hash tables, sets, and maps.
- A self-balancing tree, Red-black tree is used in kernel scheduling to preempt massively multi-processor computer operating system use.
- Another variation of tree, B-trees are used to store tree structures on disc. They are used to index a large number of records.
- B-trees are also used for secondary indexes in databases, where the index facilitates a select operation to answer some range criteria.
- Trees are used for compiler construction.
- Trees are also used in database design.
- Trees are used in file system directories.
- Trees are also widely used for information storage and retrieval in symbol tables.

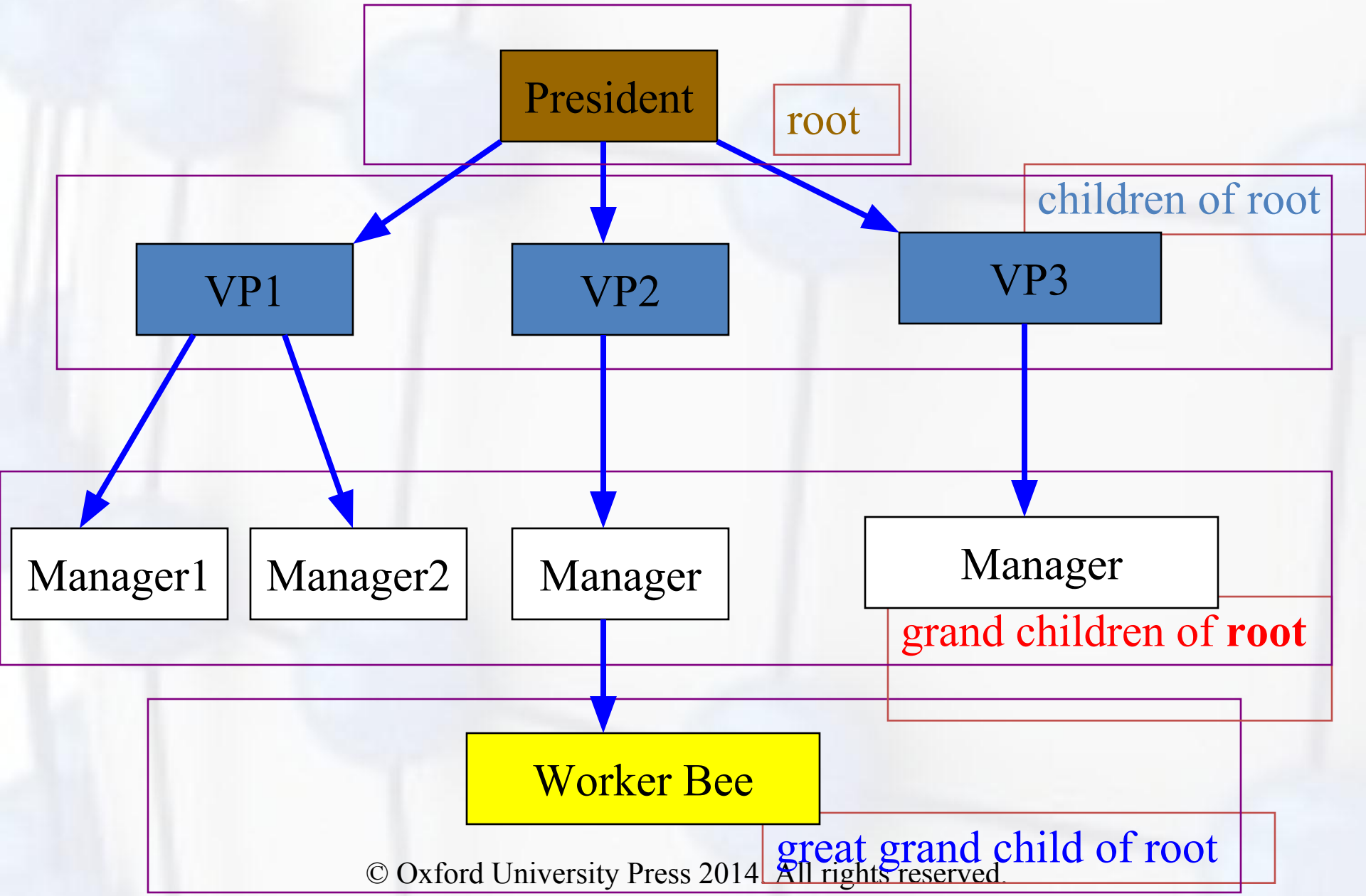
Linear Lists And Trees

- 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, and so on.

Hierarchical Data And Trees

- The element at the top of the hierarchy is the **root.**
- Elements next in the hierarchy are the **children** of the root.
- Elements next in the hierarchy are the **grandchildren** of the root, and so on.
- Elements that have no children are **leaves.**

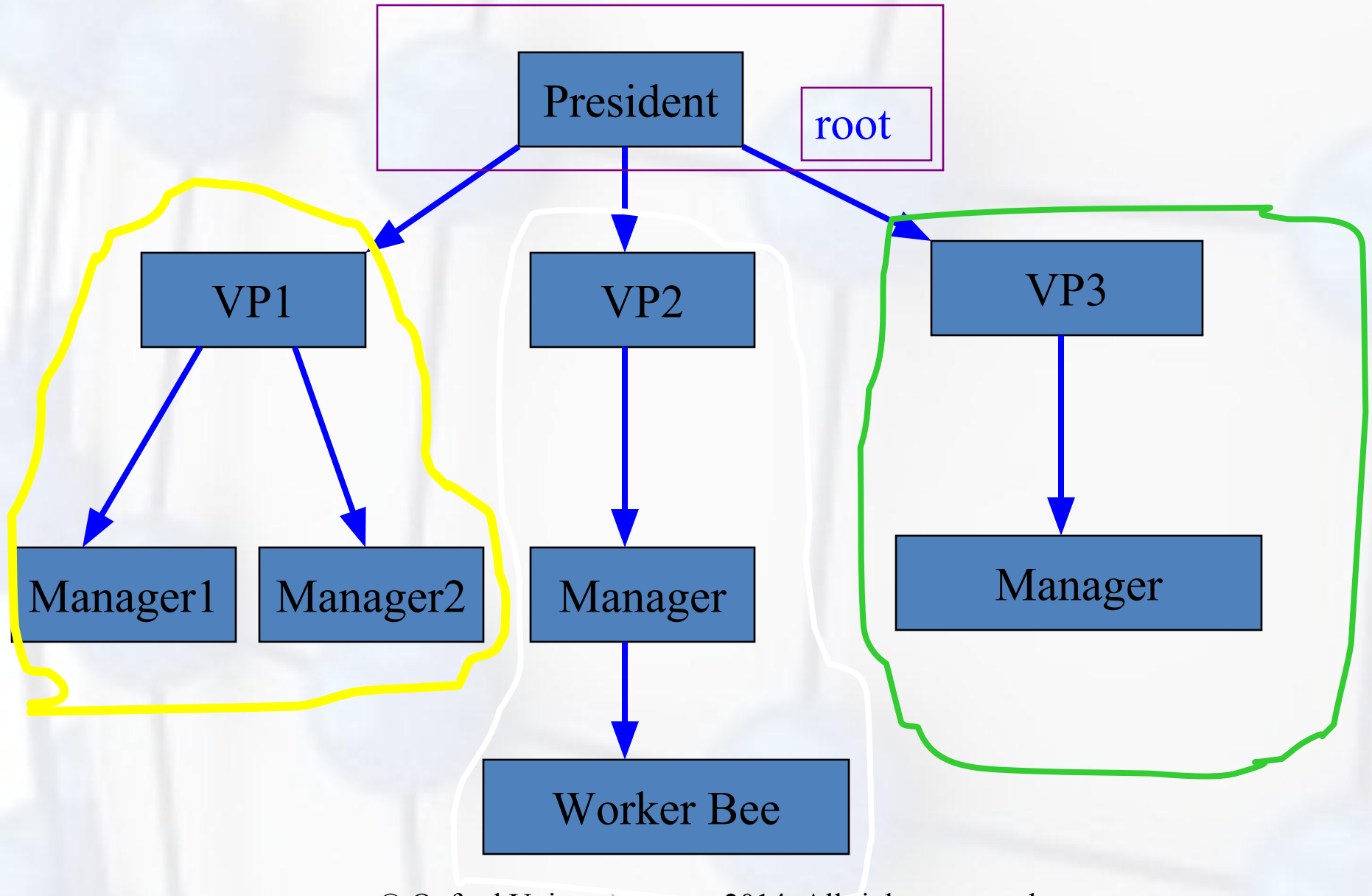
Example Tree



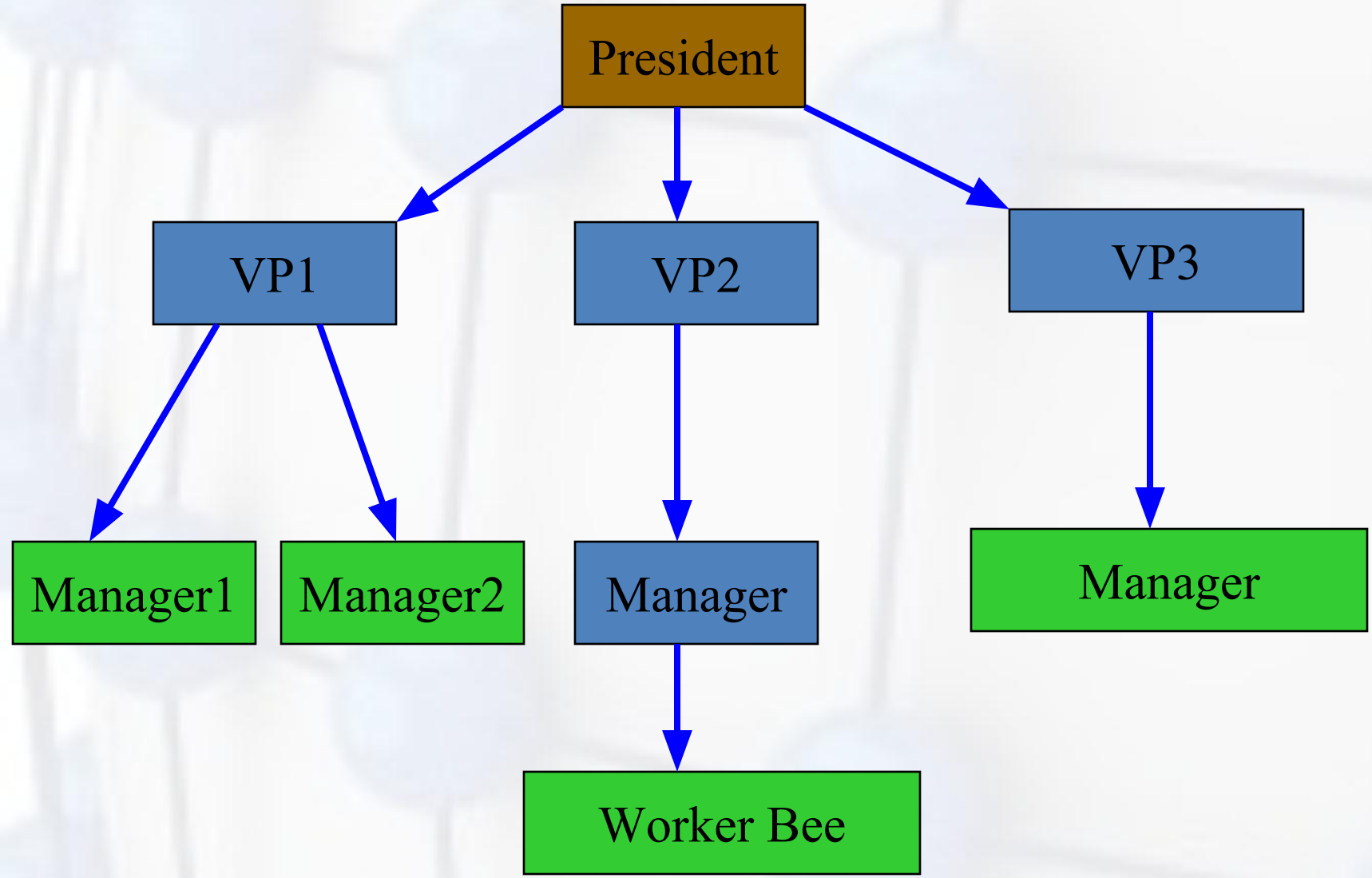
Definition

- 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.

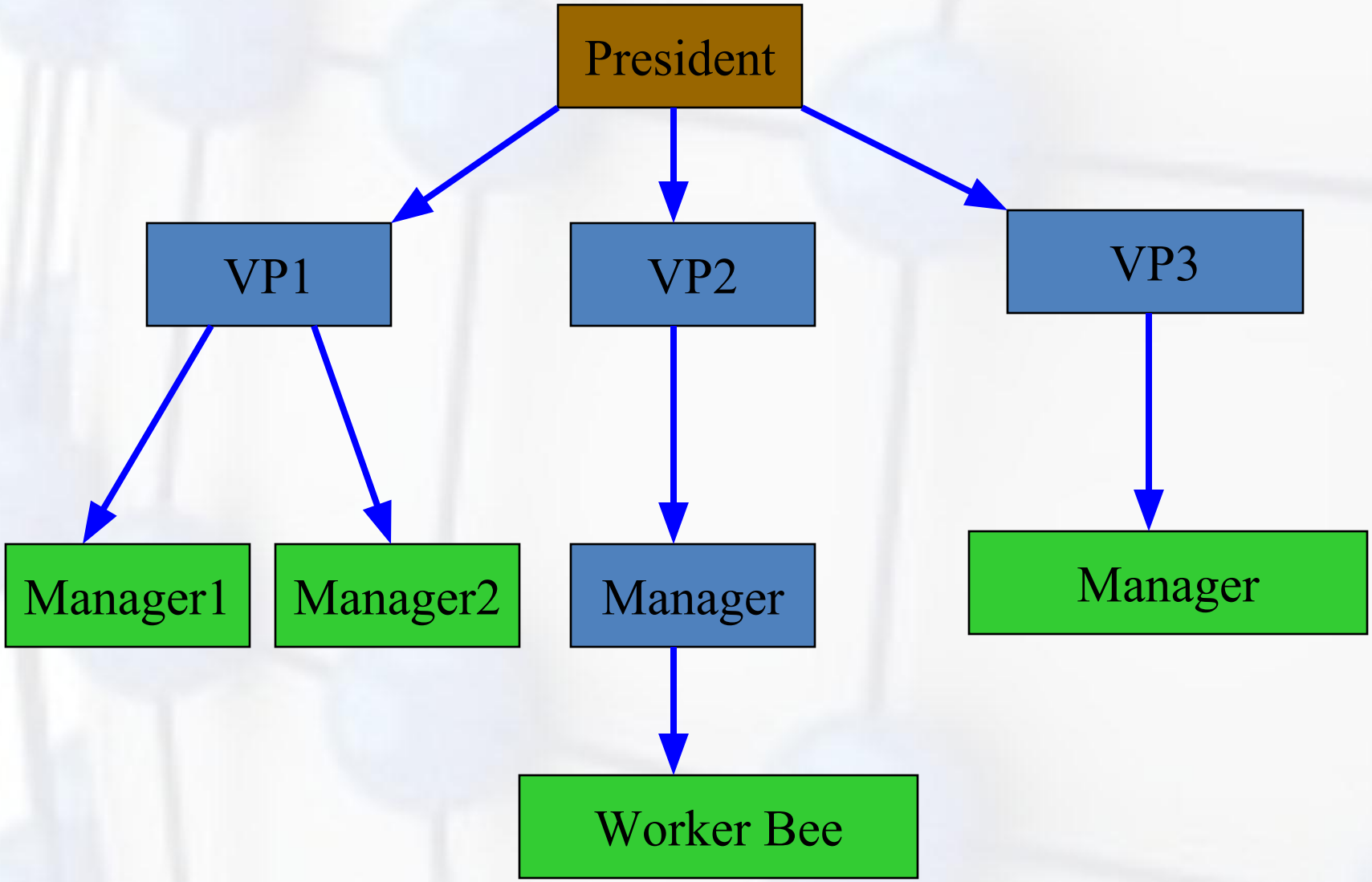
Subtrees



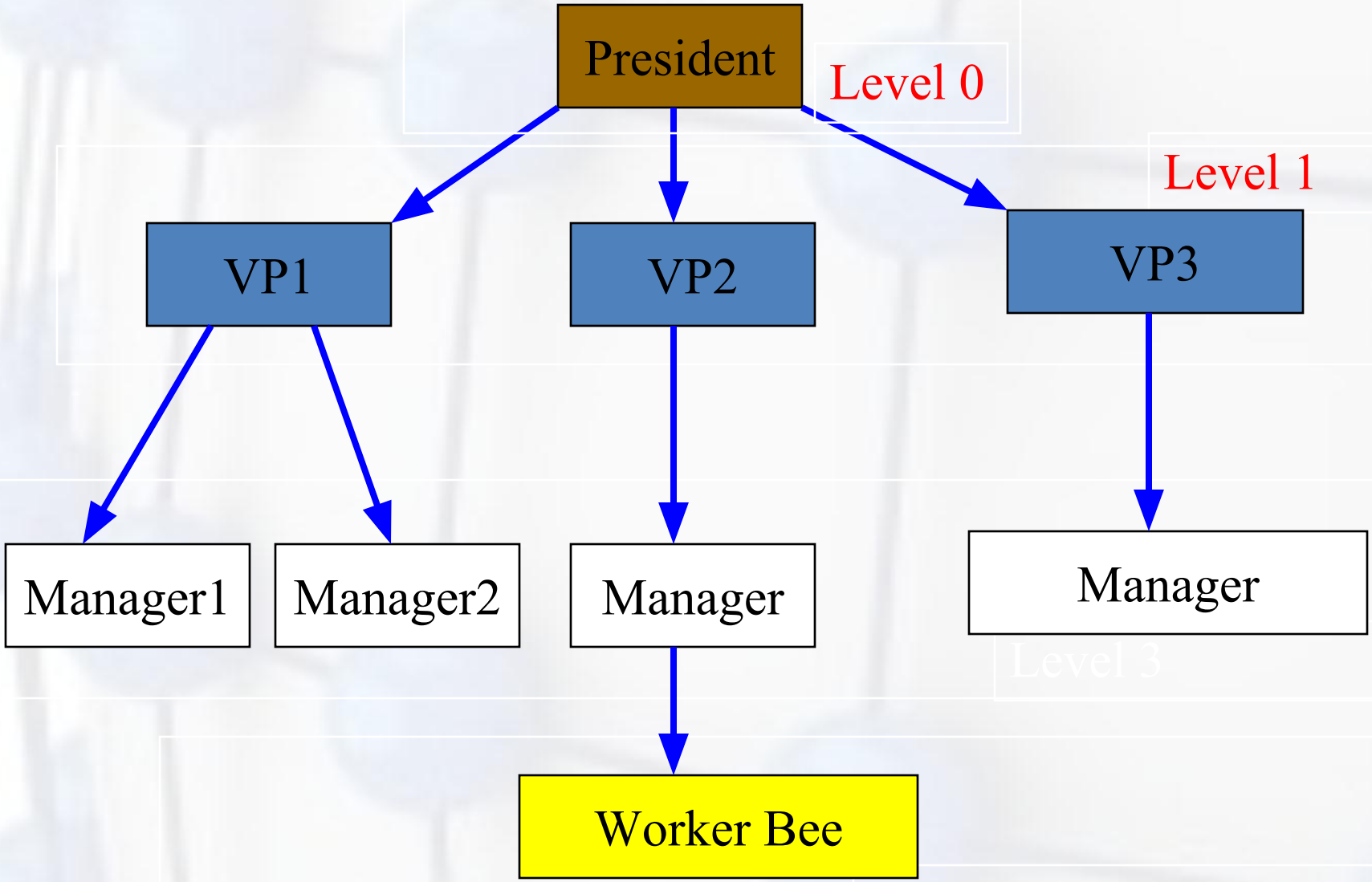
Leaves



Parent, Grandparent, Siblings, Ancestors, Descendants



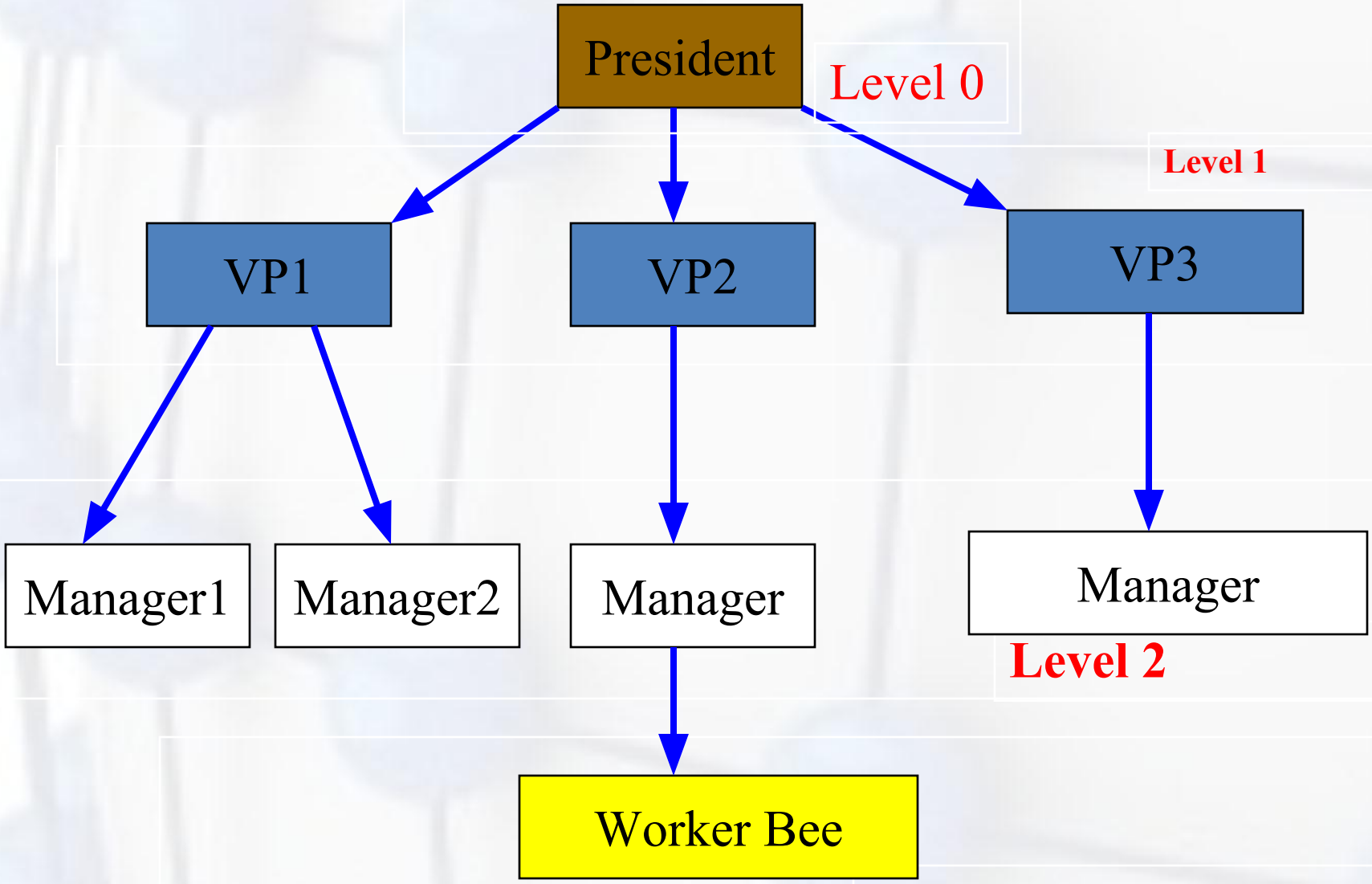
Levels



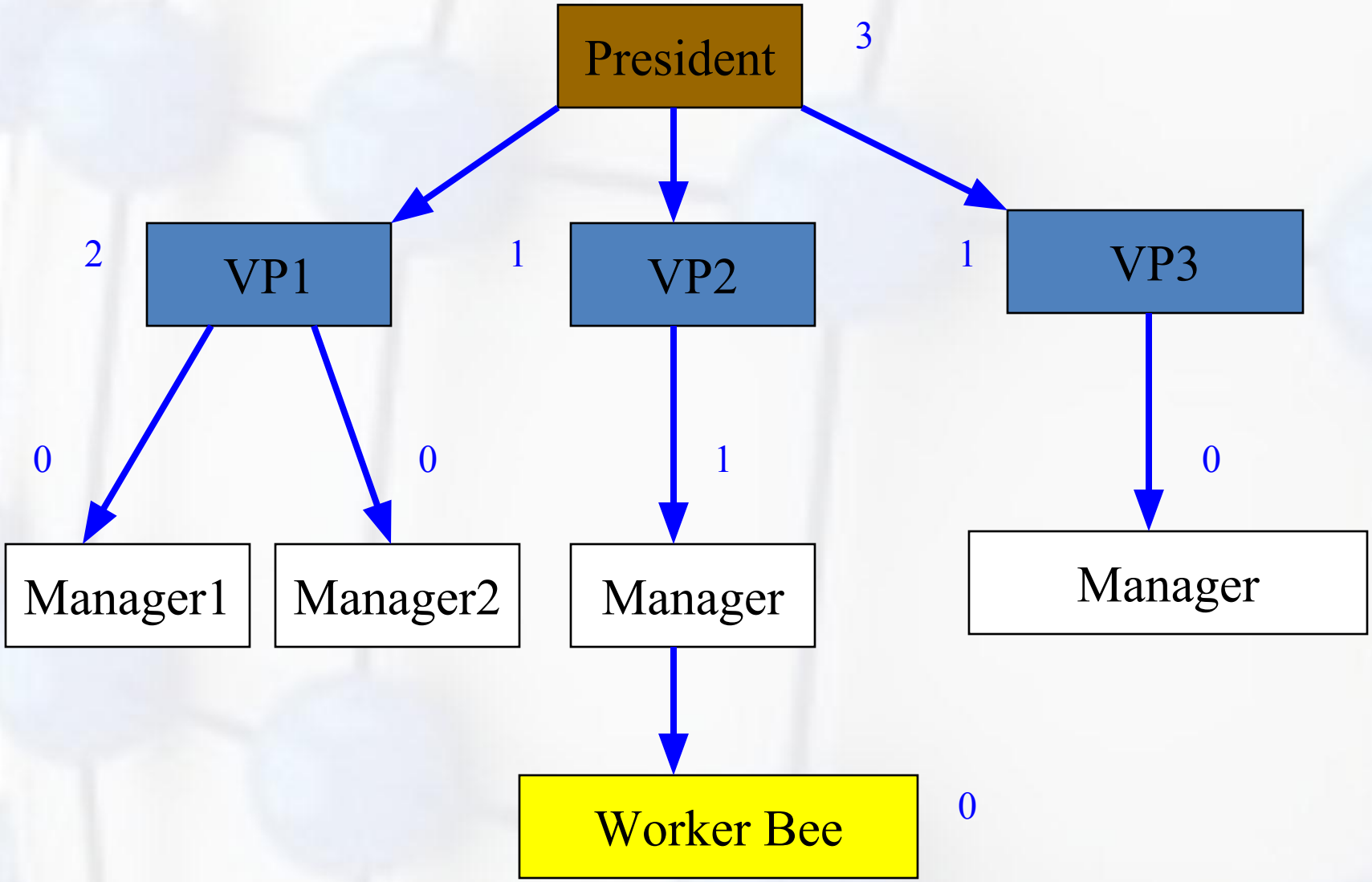
Caution

- Some texts start level numbers at 1 rather than at 0.
- Root is at level 0.
- Its children are at level 1.
- The grand children of the root are at level 2.
- And so on.
- We shall number levels with the root at level 0.

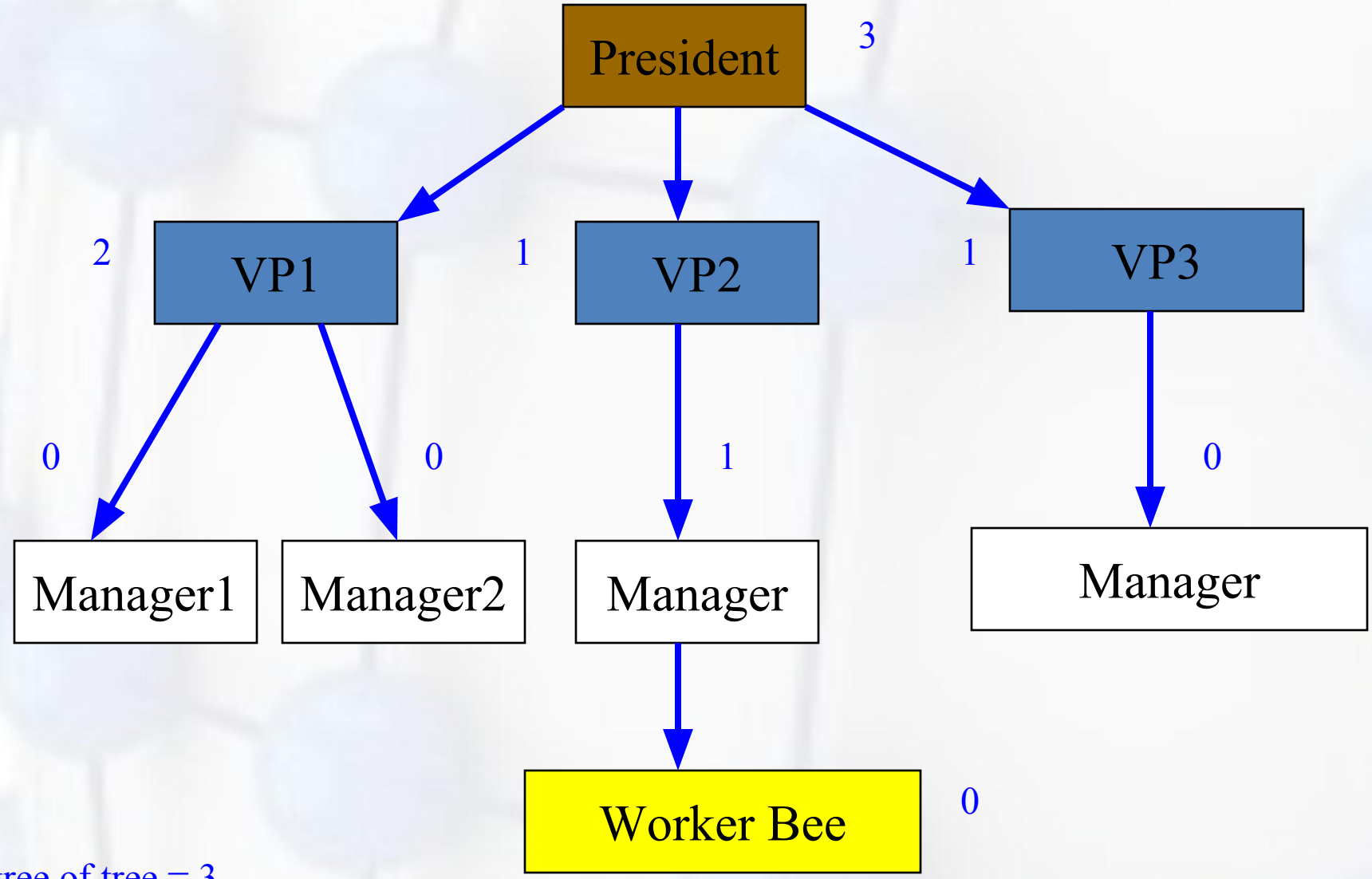
height = depth = number of levels



Node Degree = Number Of Children



Tree Degree = Max Node Degree



Degree of tree = 3.

Binary Tree

- Finite (possibly empty) collection of elements.
- A nonempty binary tree has a root element.
- The remaining elements (if any) are partitioned into two binary trees.
- These are called the left and right subtrees of the binary tree.

Differences Between A Tree & A 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.
- A binary tree may be empty; a tree cannot be empty.

Differences Between A Tree & A Binary Tree

- The subtrees 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.

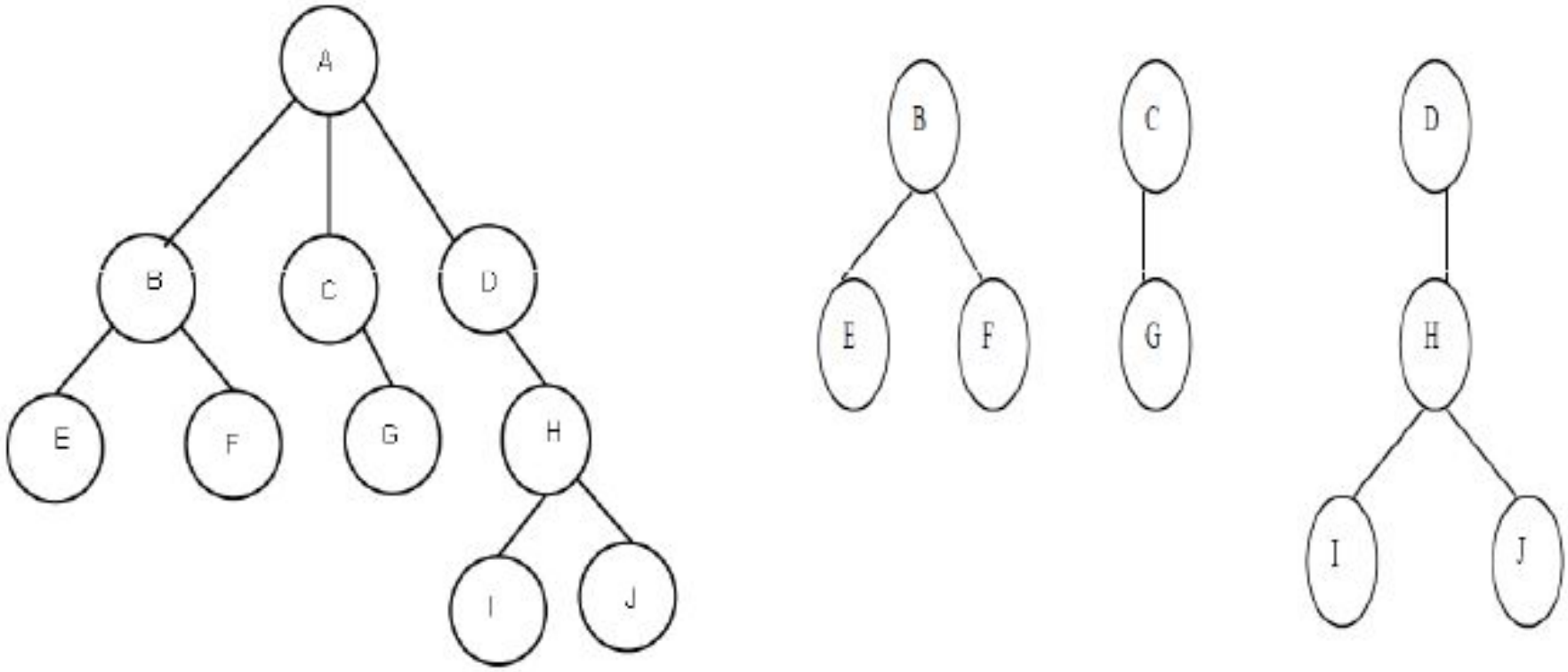
General Trees

- General trees are data structures that store elements hierarchically.
- The top node of a tree is the root node and each node, except the root, has a parent.
- A node in a general tree (except the leaf nodes) may have zero or more sub-trees.
- General trees which have 3 sub-trees per node are called ternary trees.
- However, the number of sub-trees for any node may be variable. For example, a node can have 1 sub-tree, whereas some other node can have 3 sub-trees.

Forests

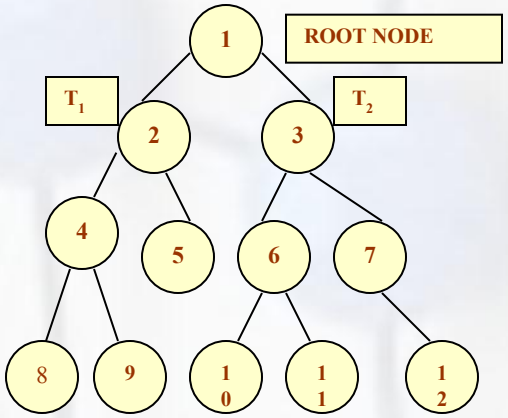
- A forest is a disjoint union of trees. A set of disjoint trees (or forest) is obtained by deleting the root and the edges connecting the root node to nodes at level 1.
- Every node of a tree is the root of some sub-tree. Therefore, all the sub-trees immediately below a node form a forest.
- A forest can also be defined as an ordered set of zero or more general trees.
- While a general tree must have a root, a forest on the other hand may be empty because by definition it is a set, and sets can be empty.
- We can convert a forest into a tree by adding a single node as the root node of the tree.

Forest



Binary Trees

- A binary tree is a data structure which is defined as a collection of elements called nodes.
- In a binary tree, the topmost element is called the root node, and each node has 0, 1, or at the most 2 children.
- Every node contains a data element, a "left" pointer which points to the left child, and a "right" pointer which points to the right child.
- The root element is pointed by a "root" pointer.
- If root = NULL, then it means the tree is empty.



R – Root node (node 1)

T_1- left sub-tree (nodes 2, 4, 5, 8, 9)

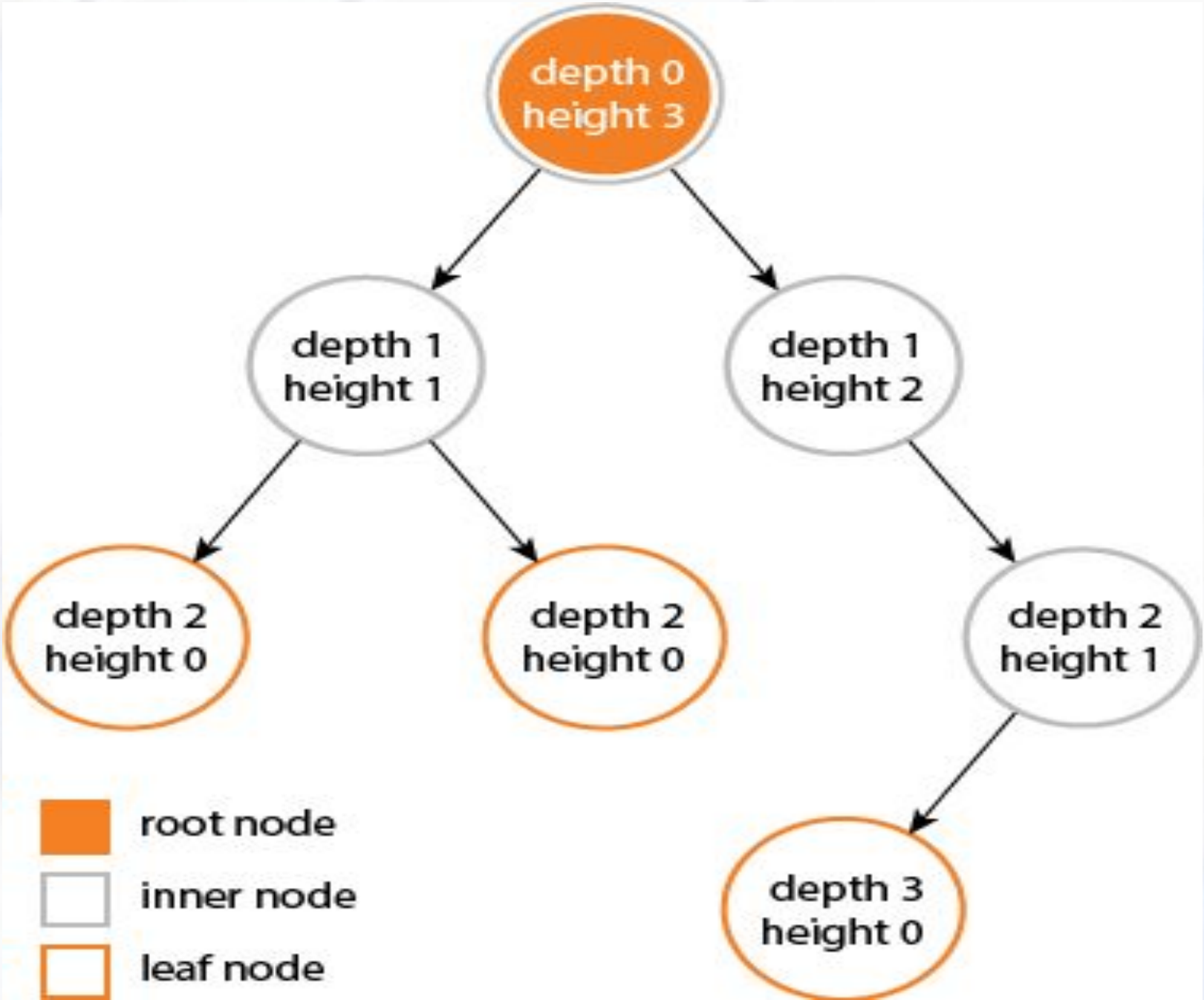
T_2- right sub-tree (nodes 3, 6, 7, 10, 11, 12)

Binary Trees - Key Terms

- *Parent:* If N is any node in T that has *left successor* S1 and *right successor* S2, then N is called the *parent* of S1 and S2. Correspondingly, S1 and S2 are called the left child and the right child of N. Every node other than the root node has a parent.
- *Sibling*: S1 and S2 are said to be *siblings*. In other words, all nodes that are at the same level and share the same parent are called *siblings* (brothers).
- ***Level number:*** Every node in the binary tree is assigned a *level number*. The root node is defined to be at level 0. The left and right child of the root node have a level number 1. Similarly, every node is at one level higher than its parents.
- ***Leaf node:*** A node that has no children.
- ***Degree:*** Degree of a node is equal to the number of children that a node has.

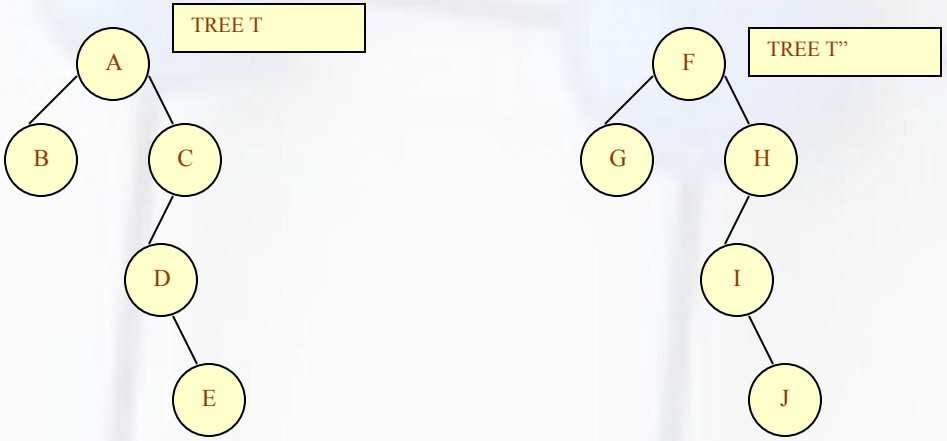
Binary Trees - Key Terms

- *In-degree* of a node is the number of edges arriving at that node.
- *Out-degree* of a node is the number of edges leaving that node.
- *Edge:* It is the line connecting a node N to any of its successors
- *Path:* A sequence of consecutive edges is called a *path.*
- *Depth:* The *depth* of a node N is given as the length of the path from the root to the node N. The depth of the root node is zero.
- *Height:* It is the total number of nodes on the path from the root node to the deepest node in the tree. A tree with only a root node has a height of 1.
- A binary tree of height h has at least h nodes and at most $2^{h+1} - 1$ nodes. This is because every level will have at least one node and can have at most 2 nodes.
- The height of a binary tree with n nodes is at least $\log_2(n+1)$ and at most n.

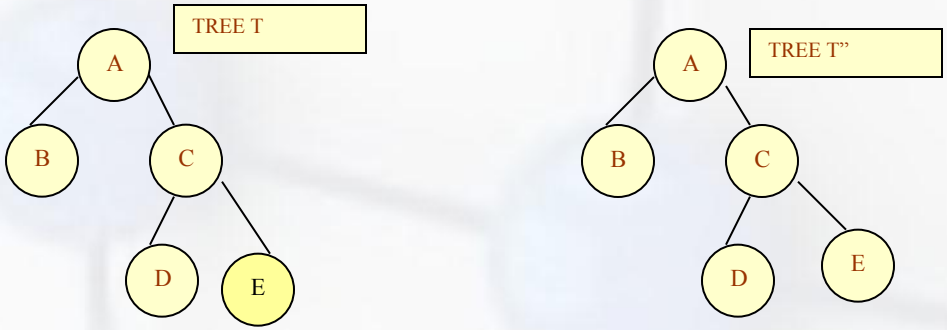

depth 0
height 3
depth 1
height 1
depth 1
height 2
depth 2
height 0
depth 2
height 0
depth 2
height 1
depth 3
height 0
root node
inner node
leaf node

Binary Trees - Key Terms

- *Similar binary trees:* Given two binary trees T and T' are said to be similar if both these trees have the same structure.



- *Copies of binary trees:* Two binary trees T and T' are said to be *copies* if they have similar structure and same content at the corresponding nodes.



Properties of Binary Tree

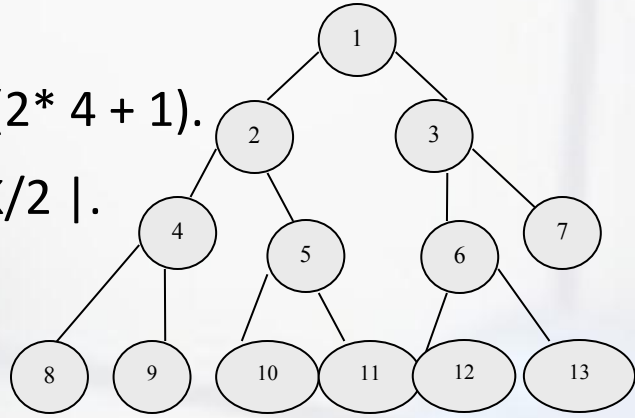
- ***The maximum number of nodes at level 'l' of a binary tree is 2^l.***
- ***Maximum number of nodes in a binary tree of height 'h' is $2^{h+1} - 1$.***

Complete Binary Trees

- A *complete binary tree* is a binary tree which satisfies two properties.
- First, in a complete binary tree every level, except possibly the last, is completely filled.
- Second, all nodes appear as far left as possible
- In a complete binary tree T_n, there are exactly n nodes and level r of T can have at most 2^r nodes.
- The formula to find the parent, left child and right child can be given as:
- If K is a parent node, then its left child can be calculated as

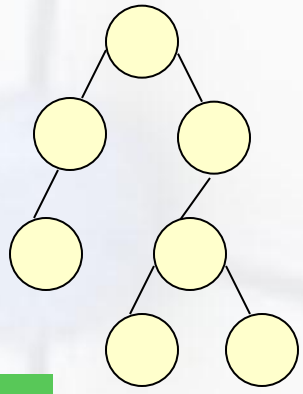
  2 * K and its right child can be calculated as 2 * K + 1.

  For example, the children of node 4 are 8 (2*4) and 9 (2* 4 + 1).
- Similarly, the parent of node K can be calculated as | K/2 |.

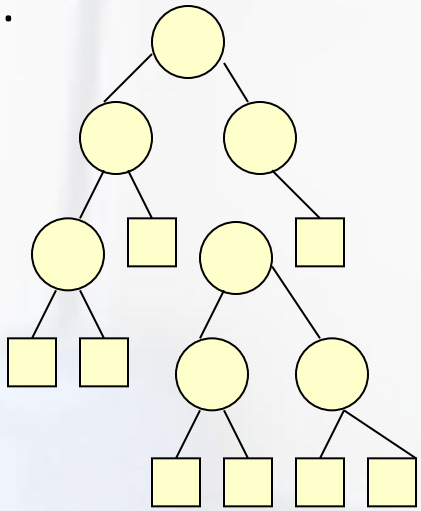


Extended Binary Trees

- A binary tree T is said to be an extended binary tree (or a 2-tree) if each node in the tree has either no child or exactly two children.
- In an extended binary tree nodes that have two children are called internal nodes and nodes that have no child or zero children are called external nodes. In the figure internal nodes are represented using a circle and **external nodes are represented using squares.**
- To convert a binary tree into an extended tree, every empty sub-tree is replaced by a new node. The original nodes in the tree are the internal nodes and the new nodes added are called the **external nodes**.



Binary tree



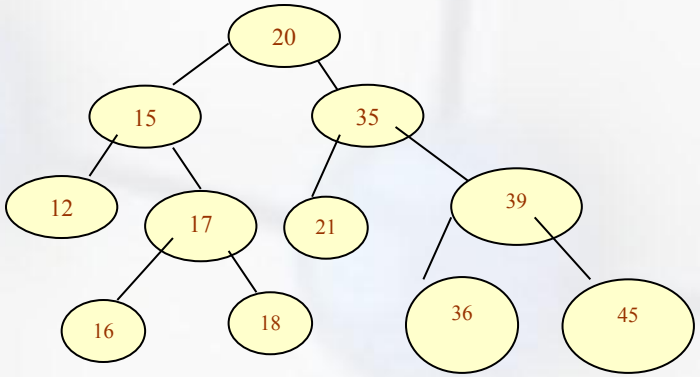
Extended binary tree

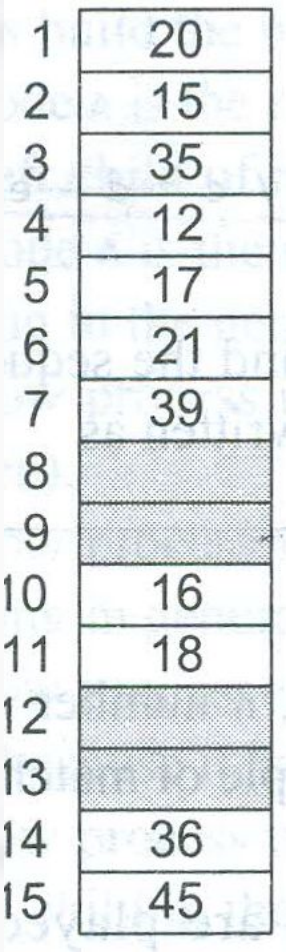
Binary Tree Representation

- Array representation.
- Linked representation.

Array Representation of Binary Trees

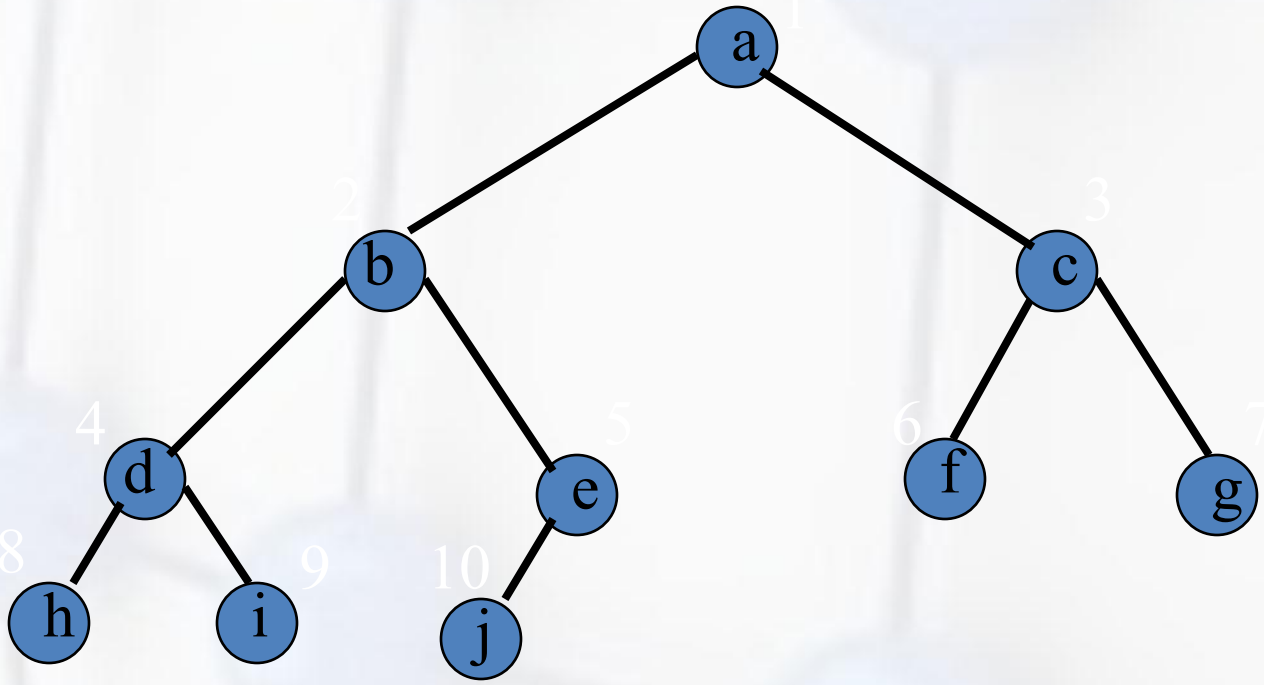
- Sequential representation of trees is done using a single **or one dimensional array**. Though, it is the simplest technique for memory representation, **it is very inefficient as it requires a lot of memory space.**
- A sequential binary tree follows the rules given below:
- One dimensional array called TREE is used.
- The root of the tree will be stored in the first location. That is, TREE[1] will store the data of the root element.
- The children of a node K will be stored in location (2*K) and (2*K+1).
- The maximum size of the array TREE is given as $(2^{h+1}-1)$, where h is the height of the tree.
- An empty tree or sub-tree is specified using NULL. If TREE[1] = NULL, then the tree is empty.

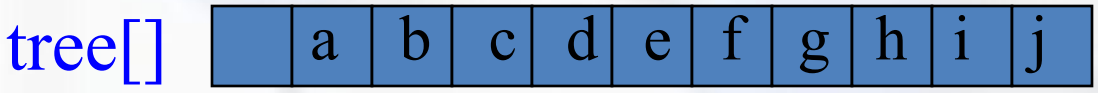




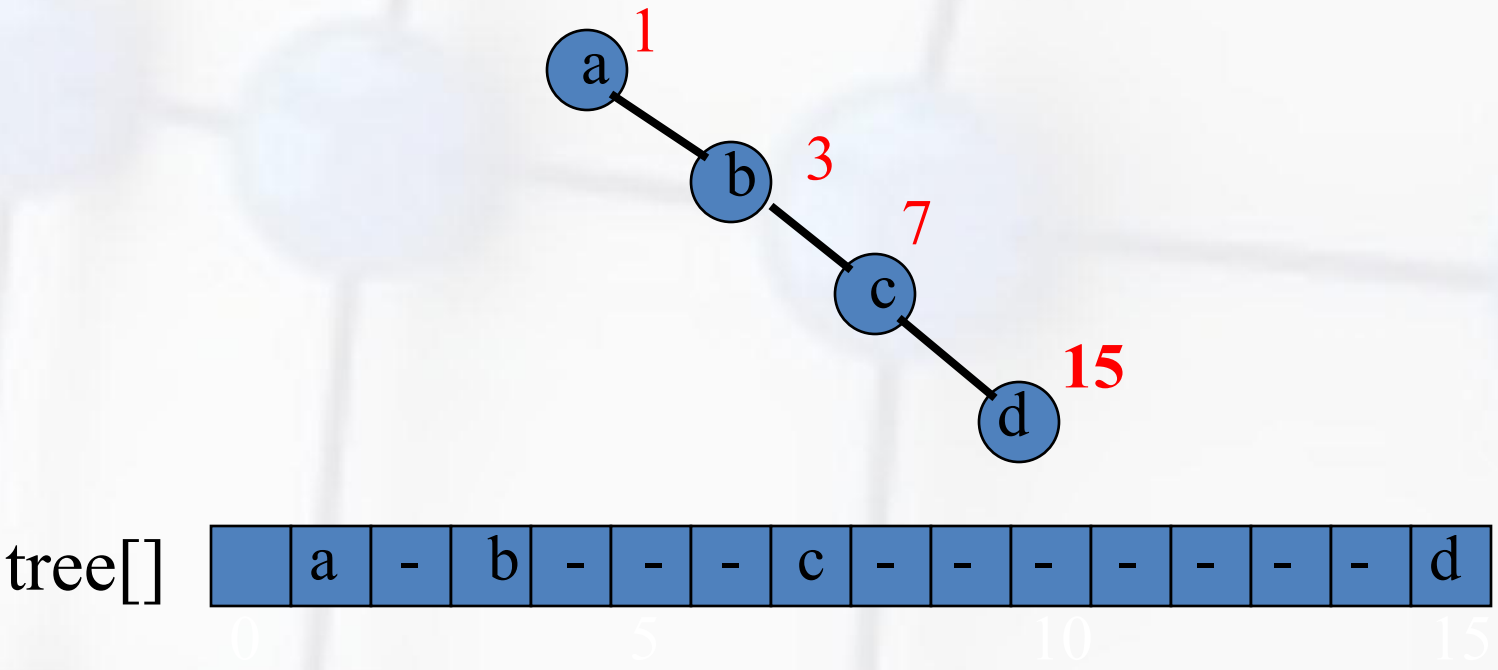
Array Representation

- Number the nodes using the numbering scheme for a full binary tree. The node that is numbered i is stored in tree[i].





Right-Skewed Binary Tree

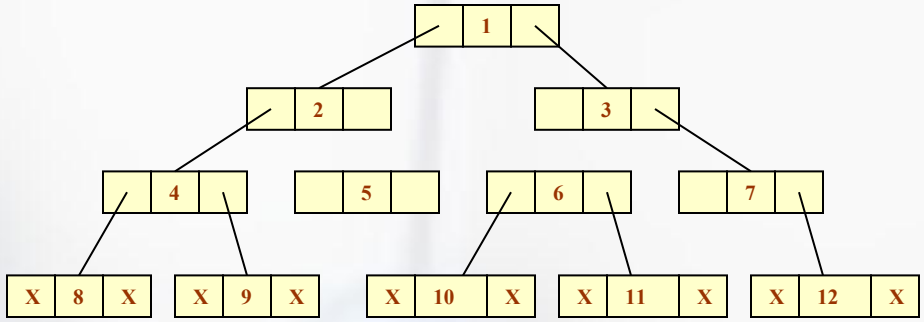


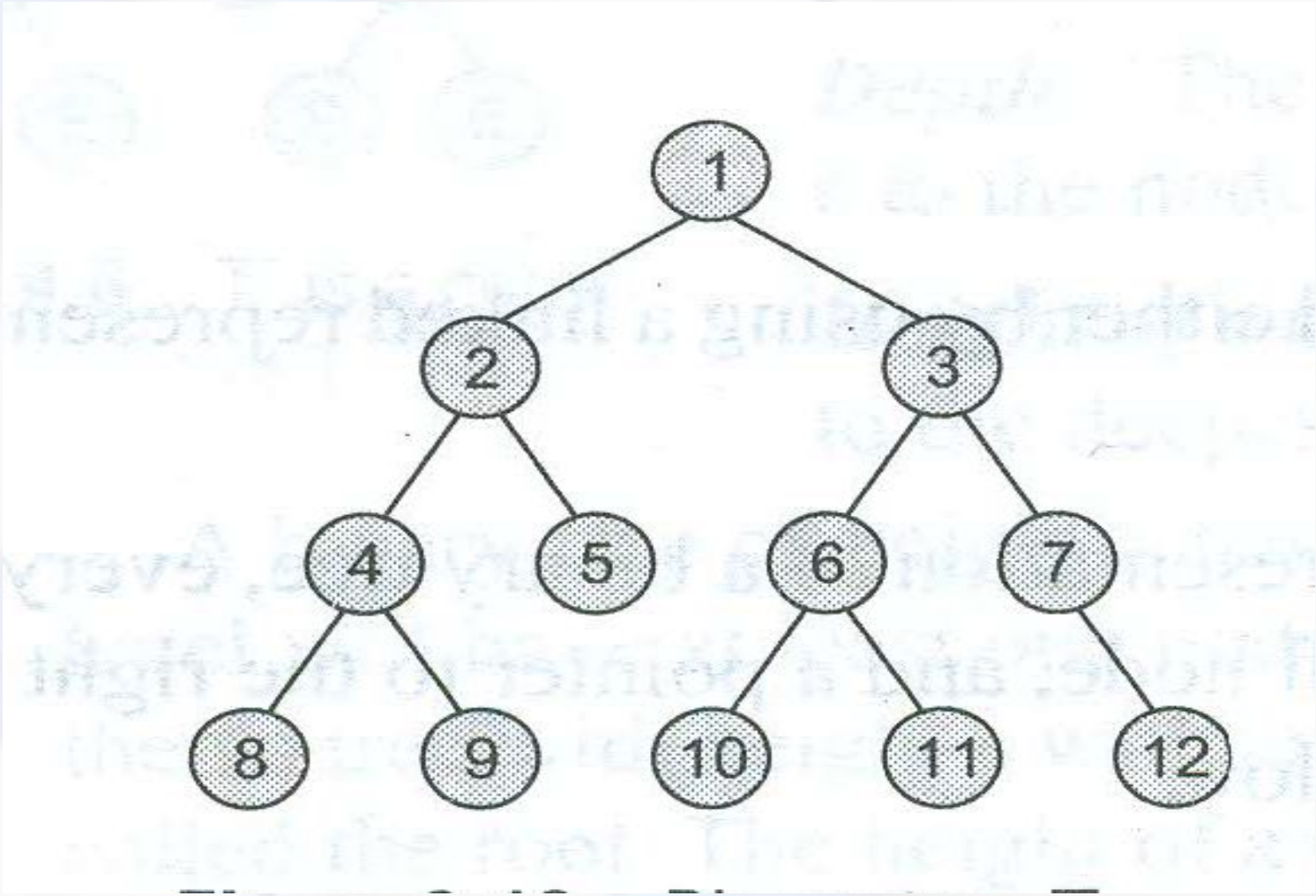
- An n node binary tree needs an array whose length is between n+1 and 2^n.

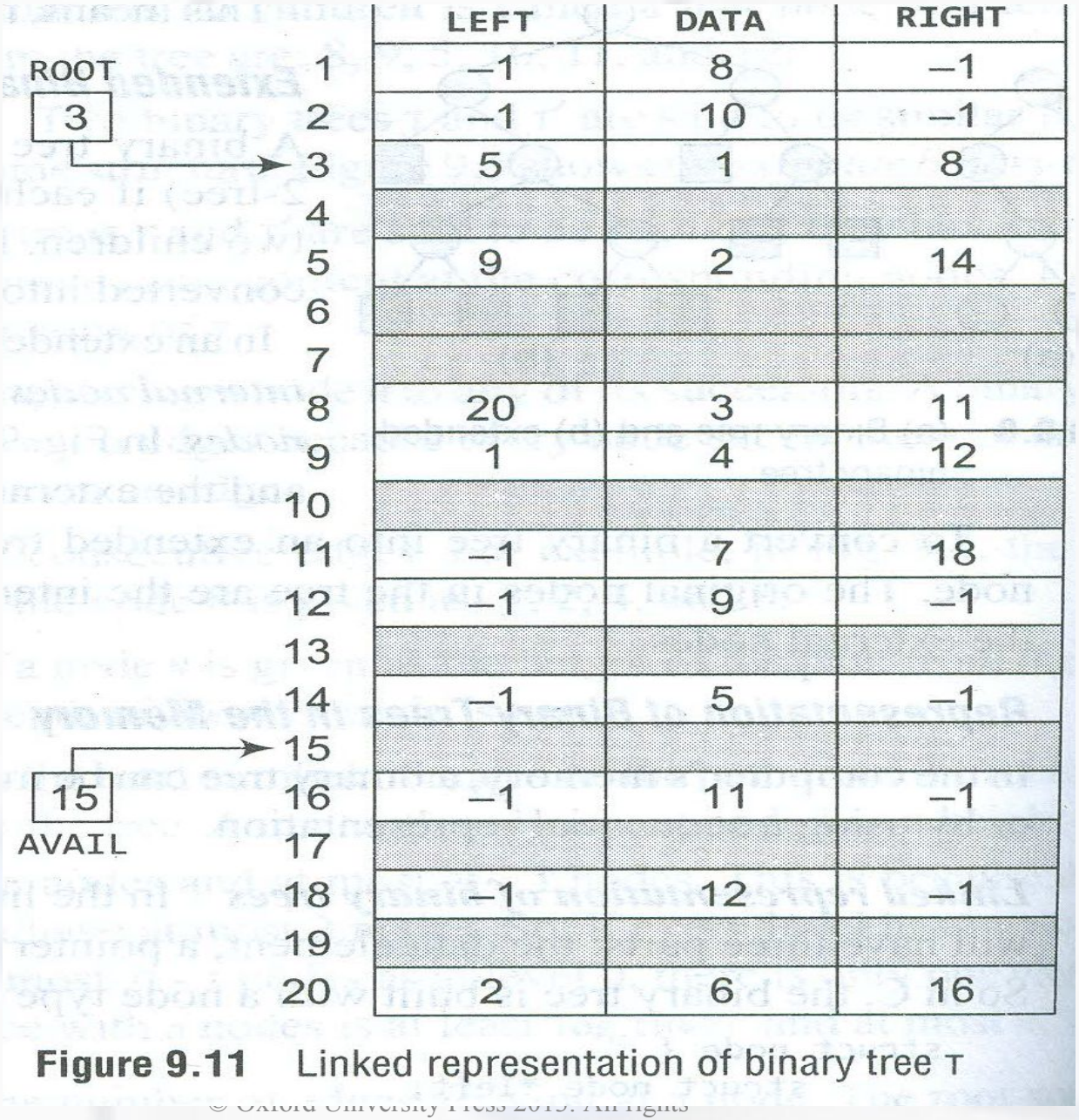
Linked Representation of Binary Trees

- **In computer's memory, a binary tree can be maintained either using a linked representation or using sequential representation.**
- **In linked representation of binary tree, every node will have three parts: the data element,**
- **a pointer to the left node and**
- **a pointer to the right node.**
- **So in C, the binary tree is built with a node type given as below.**

```
struct node
{
    struct node* left;
        int data;
    struct node* right;
};
```




1
2
3
4
5
6
7
8
9
10
11
12



	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16

Figure 9.11 Linked representation of binary tree T

Linked Representation

- Each binary tree node is represented as an object whose data type is TreeNode.
- The space required by an n node binary tree is

  n * (space required by one node).

Some Binary Tree Operations

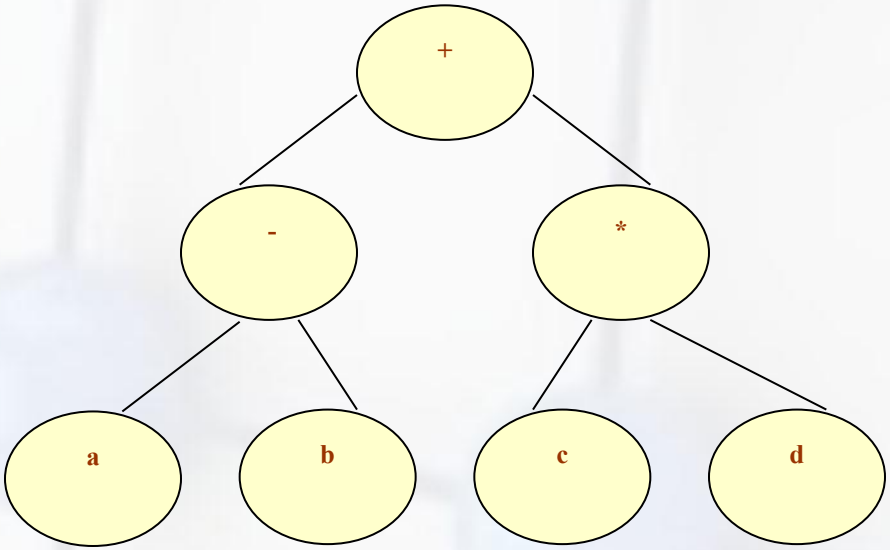
- **Determine the height.**
- **Determine the number of nodes.**
- **Make a clone.**
- **Determine if two binary trees are clones.**
- **Display the binary tree.**
- **Evaluate the arithmetic expression represented by a binary tree.**
- **Obtain the infix form of an expression.**
- **Obtain the prefix form of an expression.**
- **Obtain the postfix form of an expression.**

Expression Trees

- Binary trees are widely used to store algebraic expressions. For example, consider the algebraic expression Exp given as:

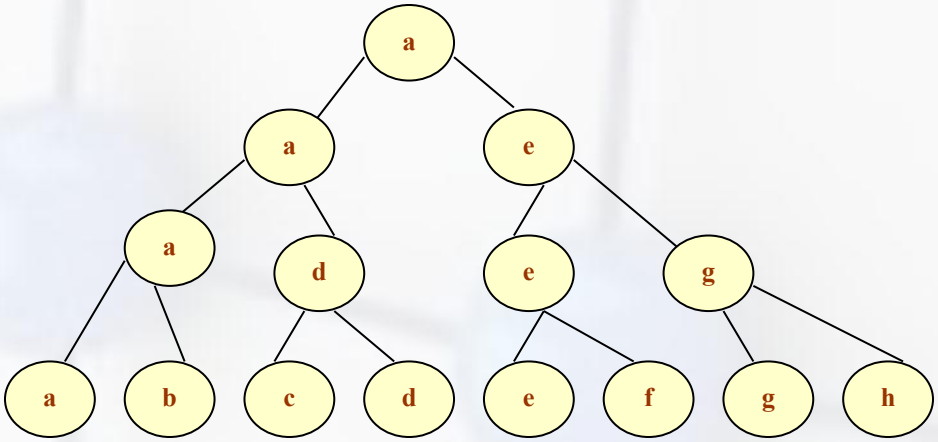
  Exp = (a – b) + (c * d)

- This expression can be represented using a binary tree as shown in figure



Tournament Trees

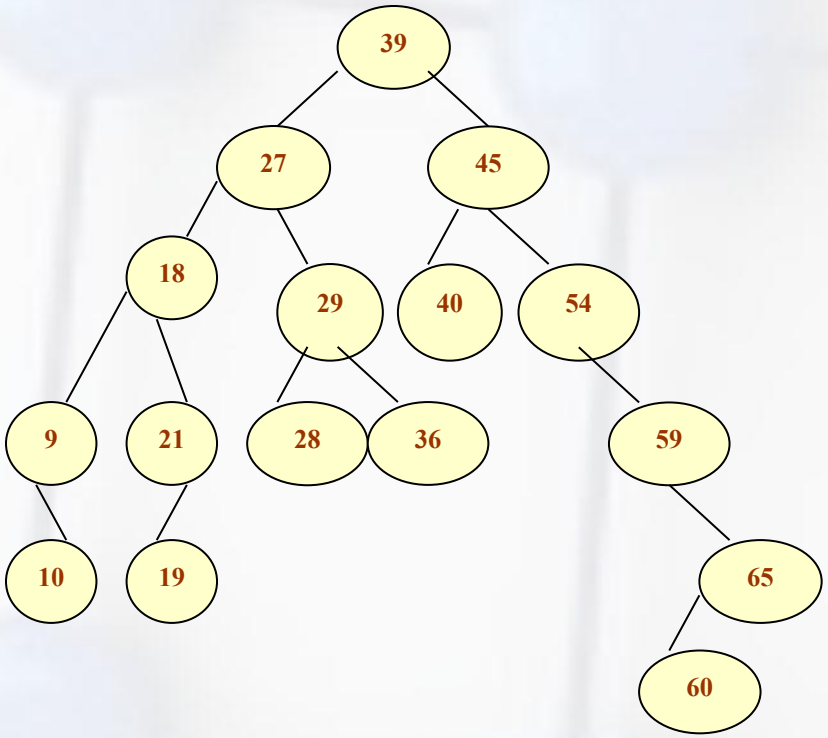
- In a tournament tree (also called a selection tree), each external node represents a player and each internal node represents the winner of the match played between the players represented by its children nodes.
- These tournament trees are also called winner trees because they are being used to record the winner at each level.
- We can also have a loser tree that records the loser at each level.



Binary Search Trees

- A binary search tree (BST), also known as an ordered binary tree, is a variant of binary tree in which the nodes are arranged in order.
- In a BST, all nodes in the left sub-tree have a value less than that of the root node.
- Correspondingly, all nodes in the right sub-tree have a value either equal to or greater than the root node.
- The same rule is applicable to every sub-tree in the tree.
- Due to its efficiency in searching elements, BSTs are widely used in dictionary problems where the code always inserts and searches the elements that are indexed by some key value.

Binary Search Tree

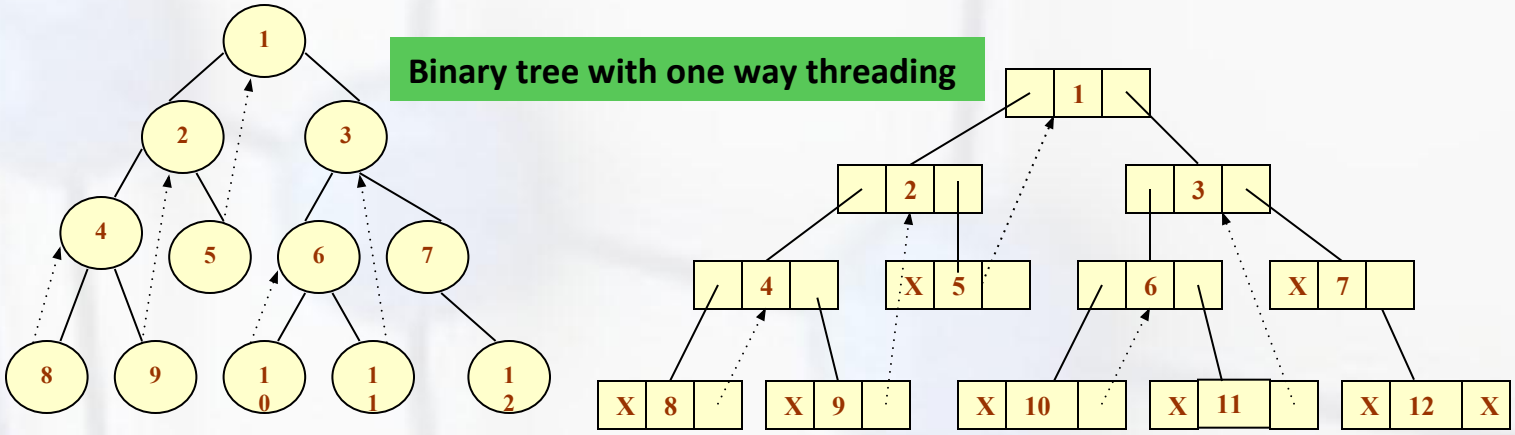


Threaded Binary Trees

- A threaded binary tree is same as that of a binary tree but with a difference in storing NULL pointers.
- In the linked representation of a BST, a number of nodes contain a NULL pointer either in their left or right fields or in both. This space that is wasted in storing a NULL pointer can be efficiently used to store some other useful piece of information.
- For example, the NULL entries can be replaced to store a pointer to the in-order predecessor, or the in-order successor of the node. These special pointers are called ***threads*** and binary trees containing threads are called ***threaded trees***. In the linked representation of a threaded binary tree, threads will be denoted using dotted lines.

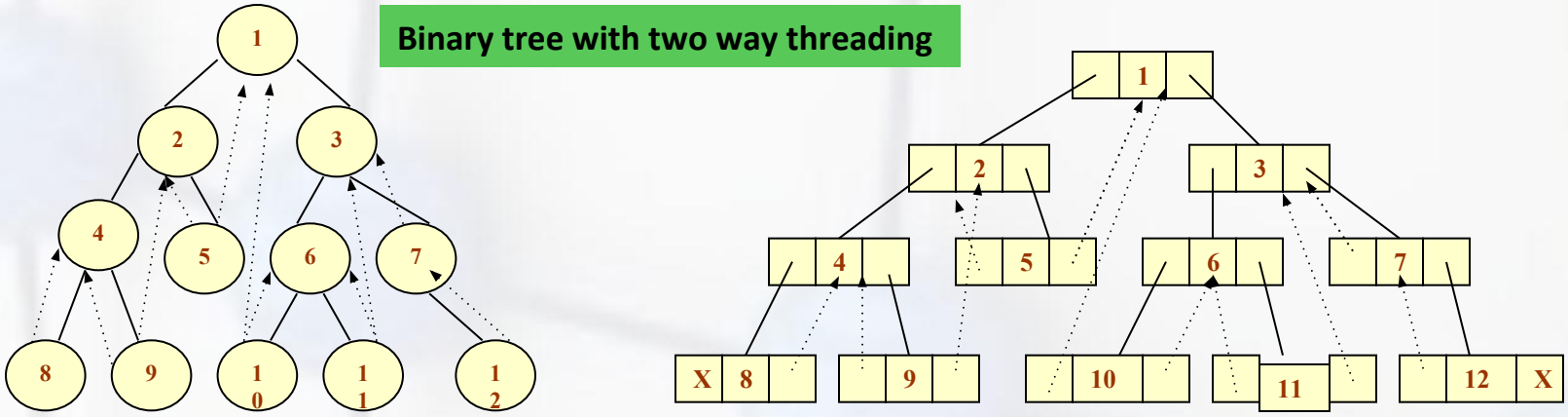
Threaded Binary Trees

- In one way threading, a thread will appear either in the right field or the left field of the node.
- If the thread appears in the left field, then it points to the in-order predecessor of the node. Such a one way threaded tree is called a left threaded binary tree.
- If the thread appears in the right field, then it will point to the in-order successor of the node. Such a one way threaded tree is called a right threaded binary tree.



Threaded Binary Trees

- In a two way threaded tree, also called a doubled threaded tree, threads will appear in both the left and right fields of the node.
- While the left field will point to the in-order predecessor of the node, the right field will point to its successor.
- A two way threaded binary tree is also called a fully threaded binary tree.



Traversing a Binary Tree

- Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way.
- There are three different algorithms for tree traversals, which differ in the order in which the nodes are visited.
- These algorithms are:

✔ Pre-order algorithm

✔ In-order algorithm

✔ Post-order algorithm

Pre-order Algorithm

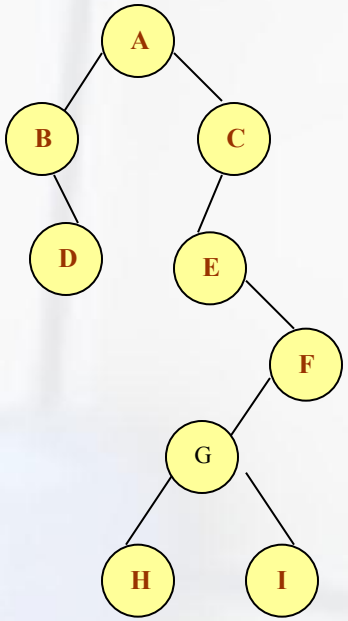
- To traverse a non-empty binary tree in preorder, the following operations are performed recursively at each node.
- The algorithm starts with the root node of the tree and continues by:

✔ Visiting the root node

✔ Traversing the left subtree

✔ Traversing the right subtree

A, B, D, C, E, F, G, H and I



PREORDER

- Step 1 Repeat steps 2 to 4 while Tree! = NULL

  STEP 2 Write TREE-□ DATA

  STEP 3 PREORDER (TREE□ LEFT)

  STEP 4 PREORDER (TREE□ RIGHT)

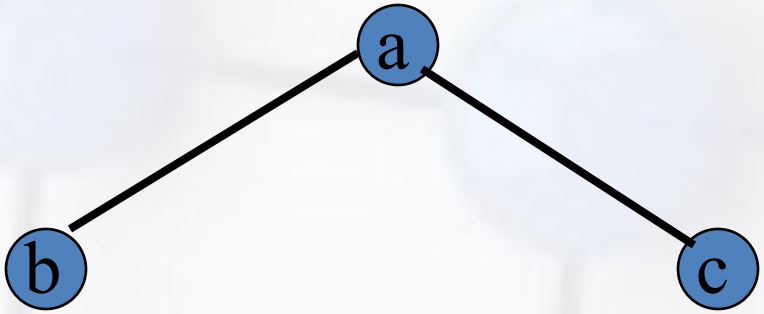
  End of Loop

  STEP 5 END

Preorder Traversal

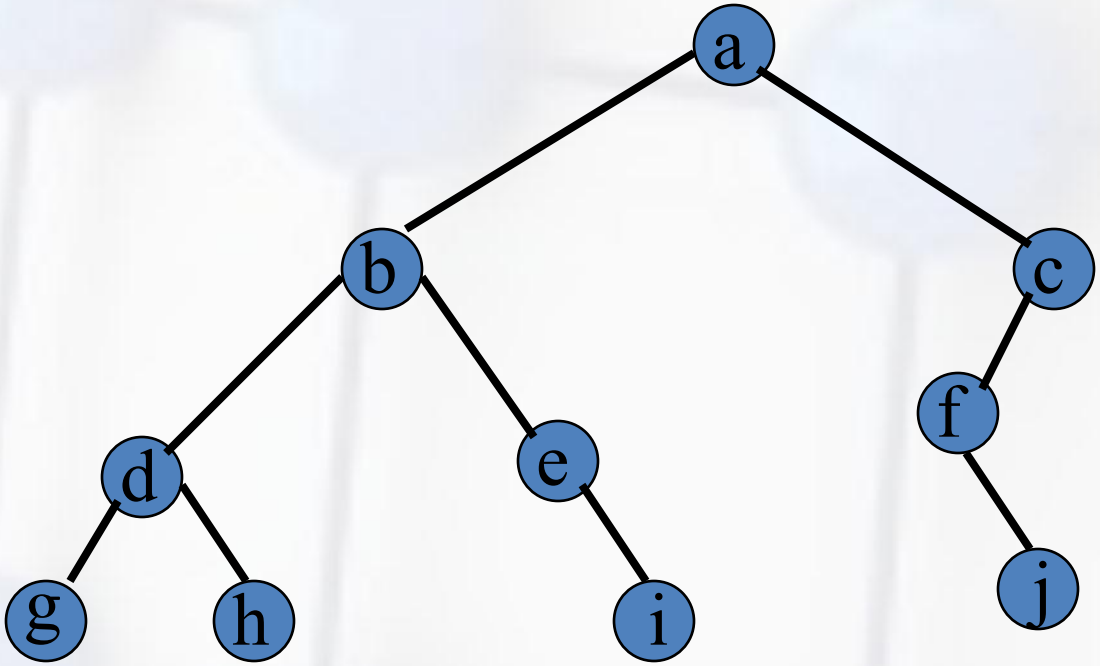
```
void preOrder(treePointer ptr)
{
    if (ptr != NULL)
    {
        visit(t);
        preOrder(ptr->leftChild);
        preOrder(ptr->rightChild);
    }
}
```

Preorder Example (Visit = print)



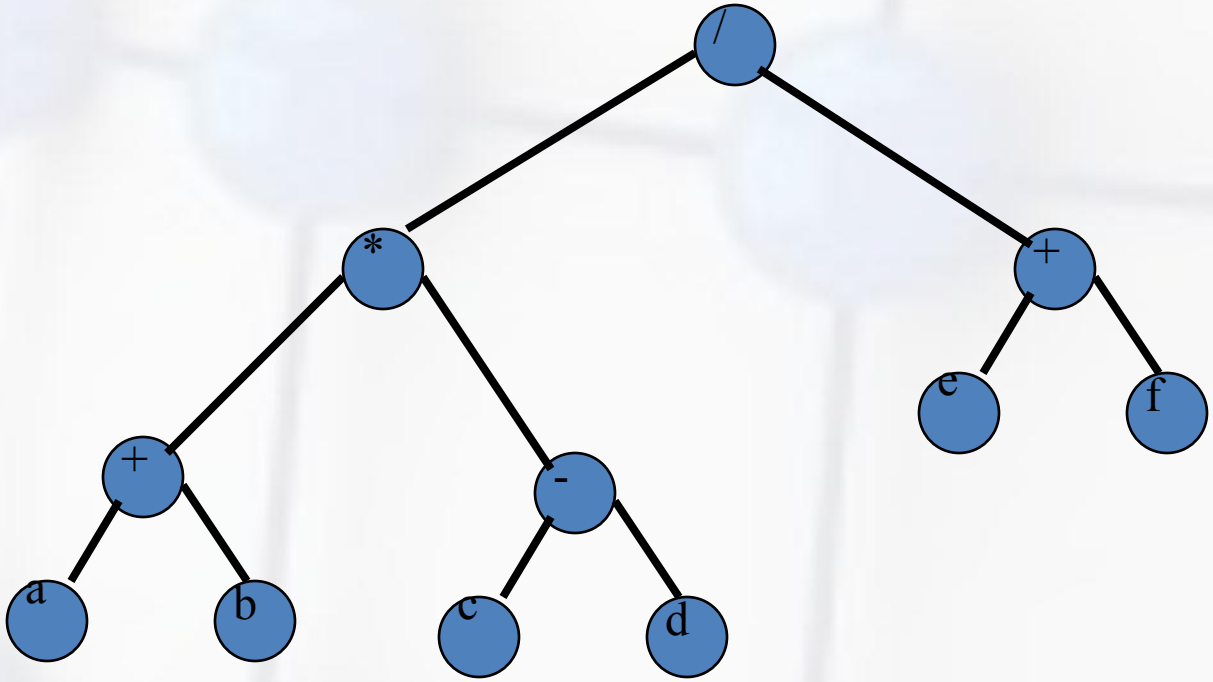
a b c

Preorder Example (Visit = print)



a b d g h e i c f j

Preorder Of Expression Tree



/ * + a b - c d + e f

Gives prefix form of expression!

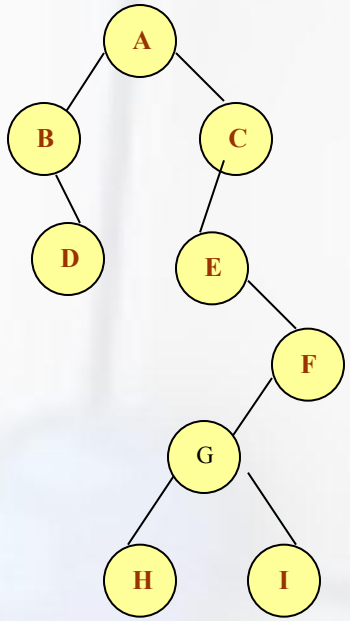
In-order Algorithm

- To traverse a non-empty binary tree in **in-order**, the following operations are performed recursively at each node.
- The algorithm starts with the root node of the tree and continues by,

✔ Traversing the left subtree

✔ Visiting the root node

✔ Traversing the right subtree



B, D, A, E, H, G, I, F and C

INORDER

- Step 1 Repeat steps 2 to 4 while Tree! = NULL

STEP 2 INORDER (TREE LEFT)

STEP 3 Write TREE- DATA

STEP 4 INORDER (TREE RIGHT)

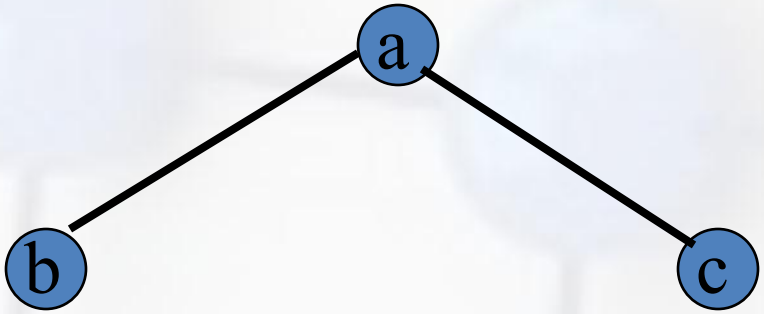
End of Loop

STEP 5 END

Inorder Traversal

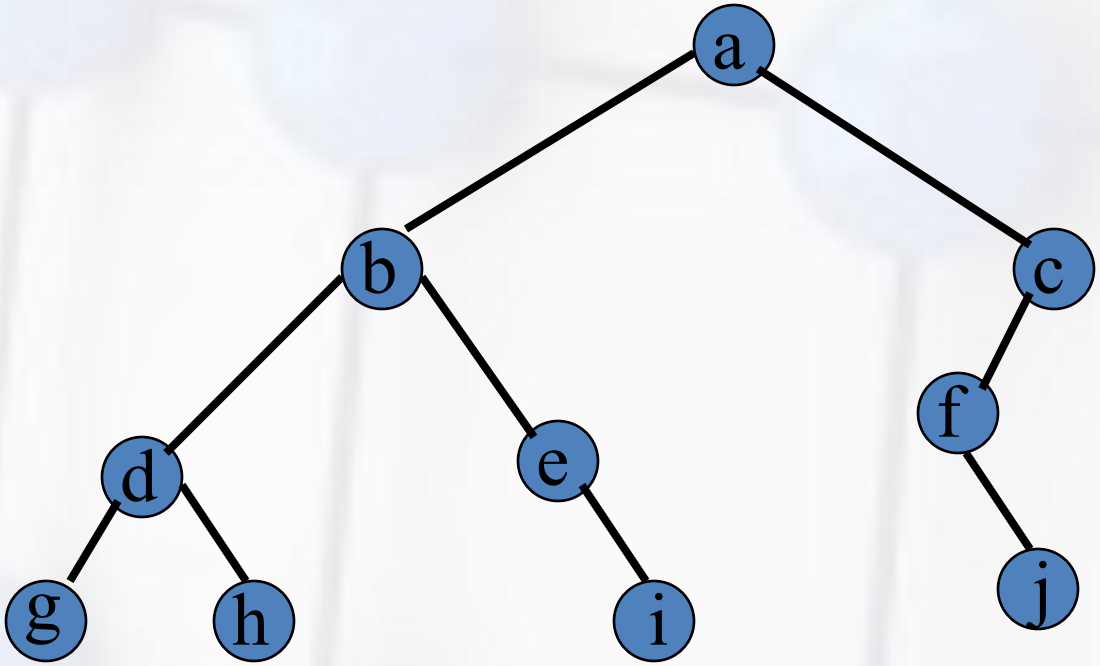
```
void inOrder(treePointer ptr)
{
    if (ptr != NULL)
    {
        inOrder(ptr->leftChild);
        visit(ptr);
        inOrder(ptr->rightChild);
    }
}
```

Inorder Example (Visit = print)



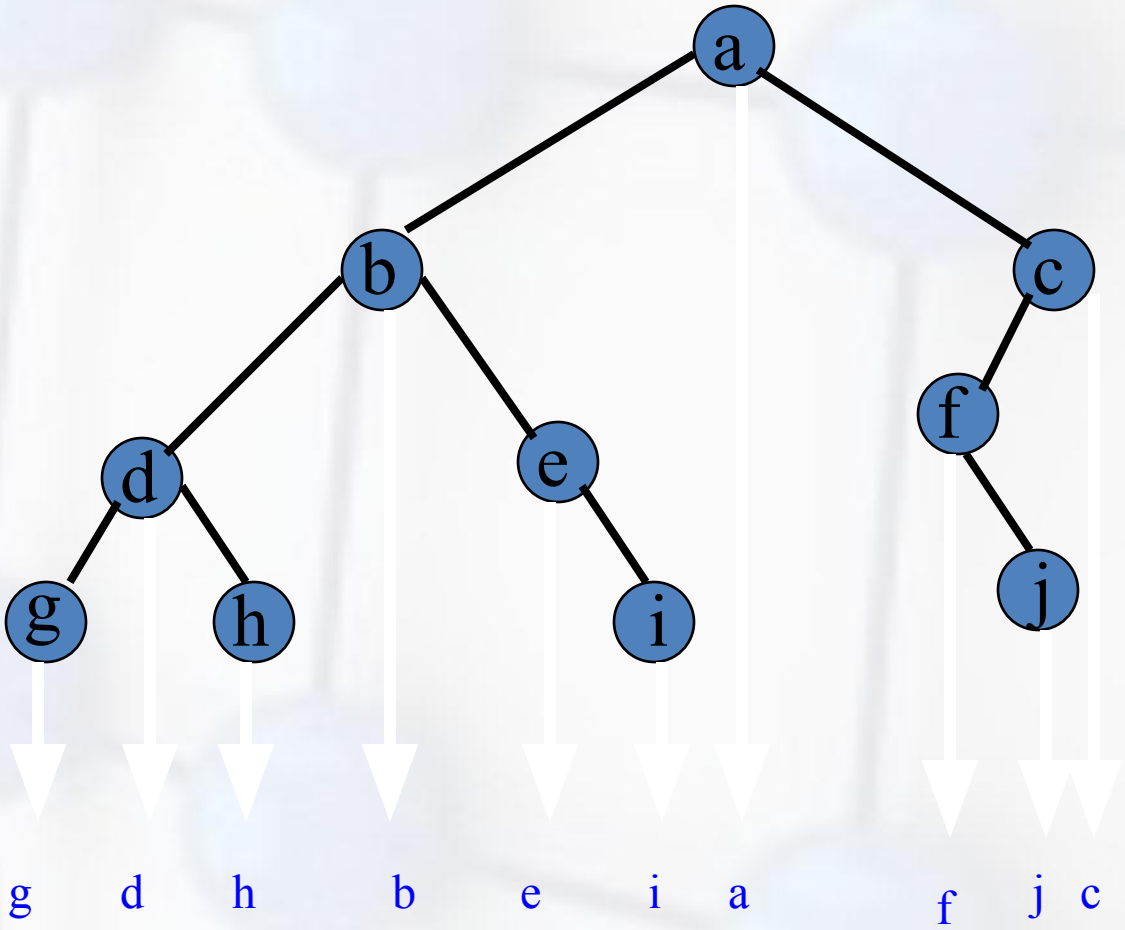
b a c

Inorder Example (Visit = print)

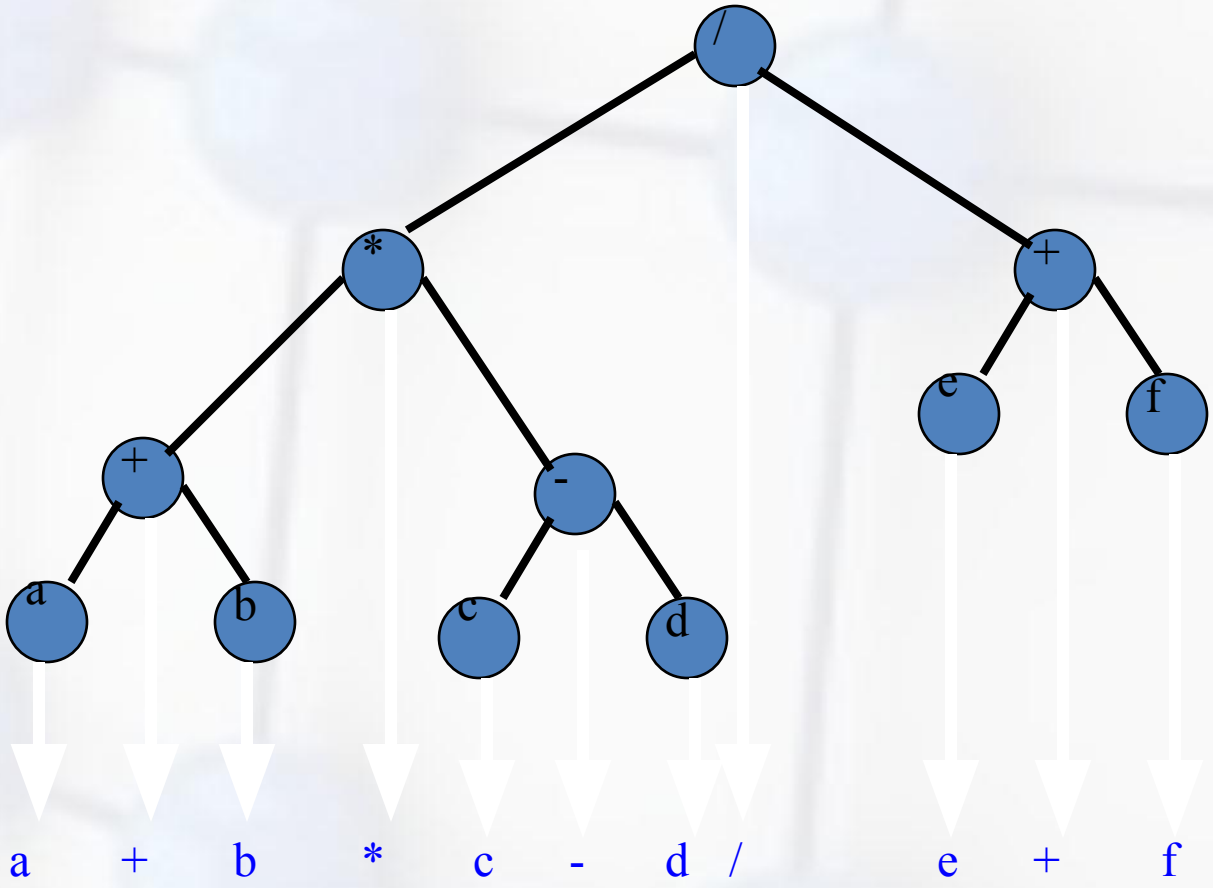


g d h b e i a f j c

Inorder By Projection (Squishing)



Inorder Of Expression Tree



Gives infix form of expression (sans parentheses)!

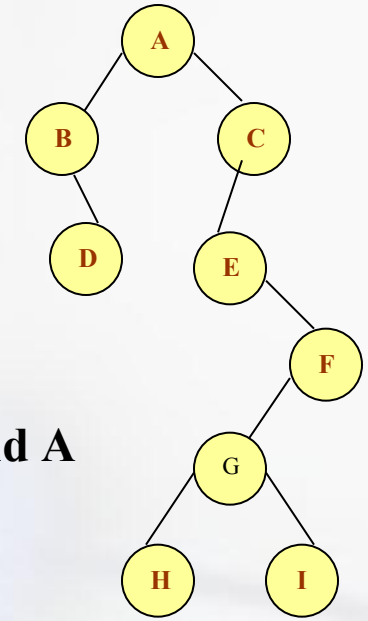
Post-order Algorithm

- To traverse a non-empty binary tree in **post-order**, the following operations are performed recursively at each node.
- The algorithm starts with the root node of the tree and continues by,

✔ Traversing the left subtree

✔ Traversing the right subtree

✔ Visiting the root node



D, B, H, I, G, F, E, C and A

POSTORDER

- Step 1 Repeat steps 2 to 4 while Tree! = NULL

  STEP 2 POSTORDER (TREE LEFT)

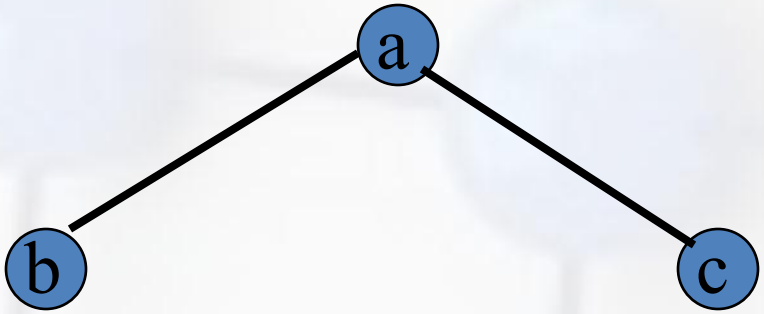
  STEP 3 POSTORDER (TREE RIGHT)

  STEP 4 Write TREE- DATA

  End of Loop

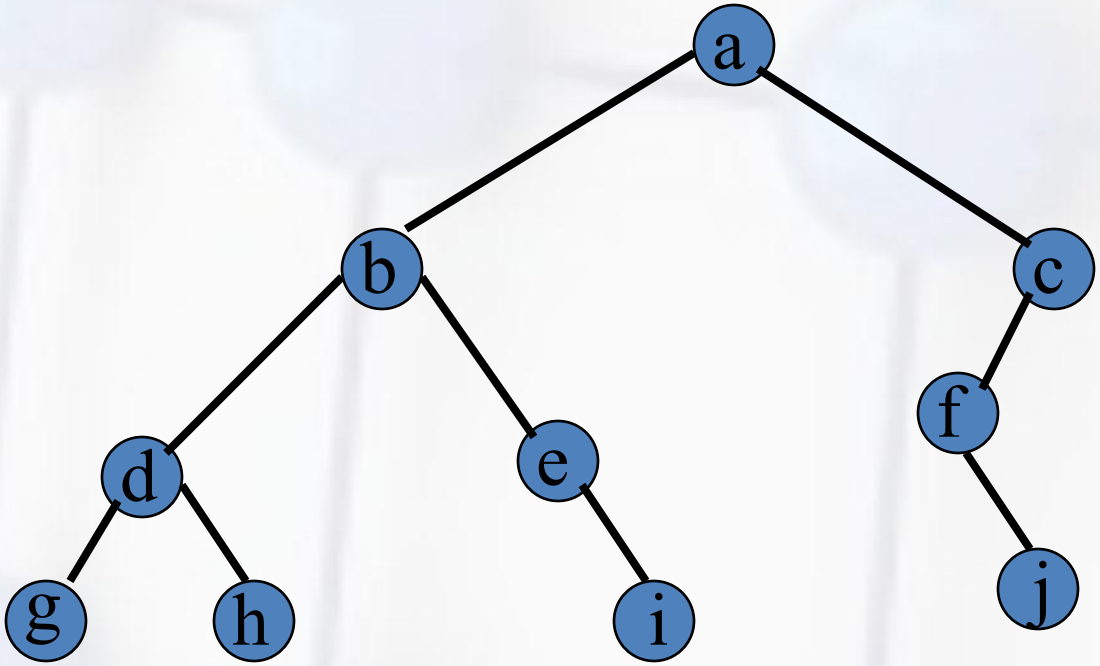
  STEP 5 END

Postorder Example (Visit = print)



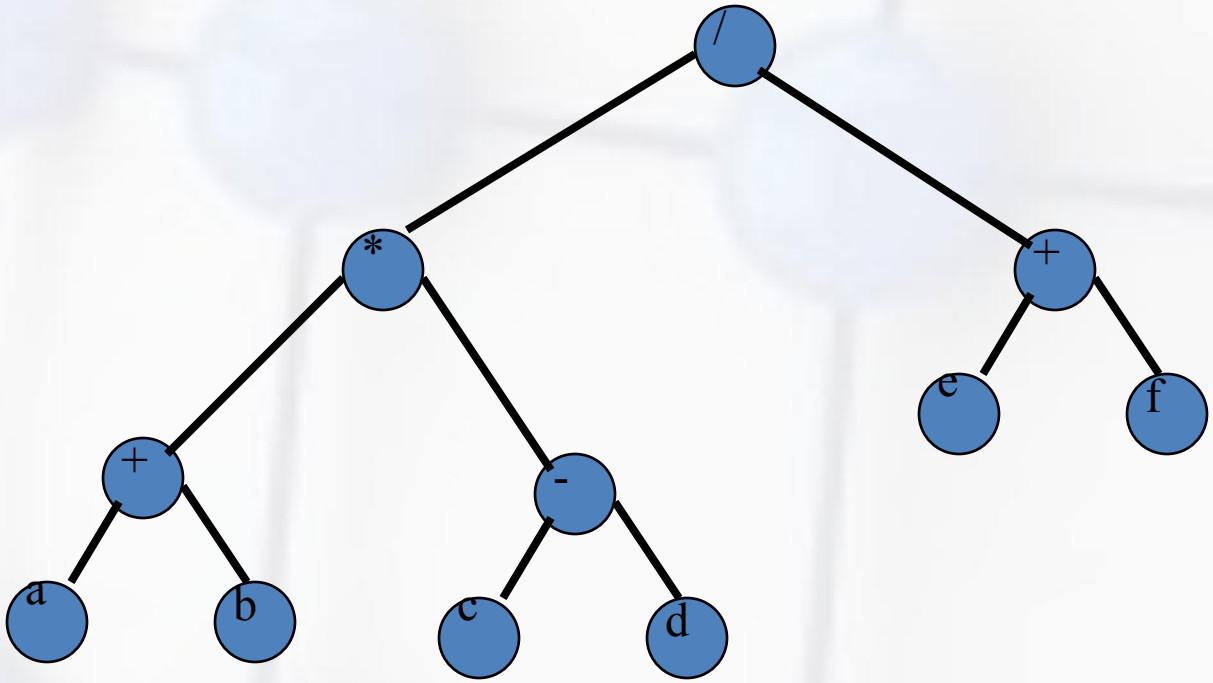
b c a

Postorder Example (Visit = print)



g h d i e b j f c a

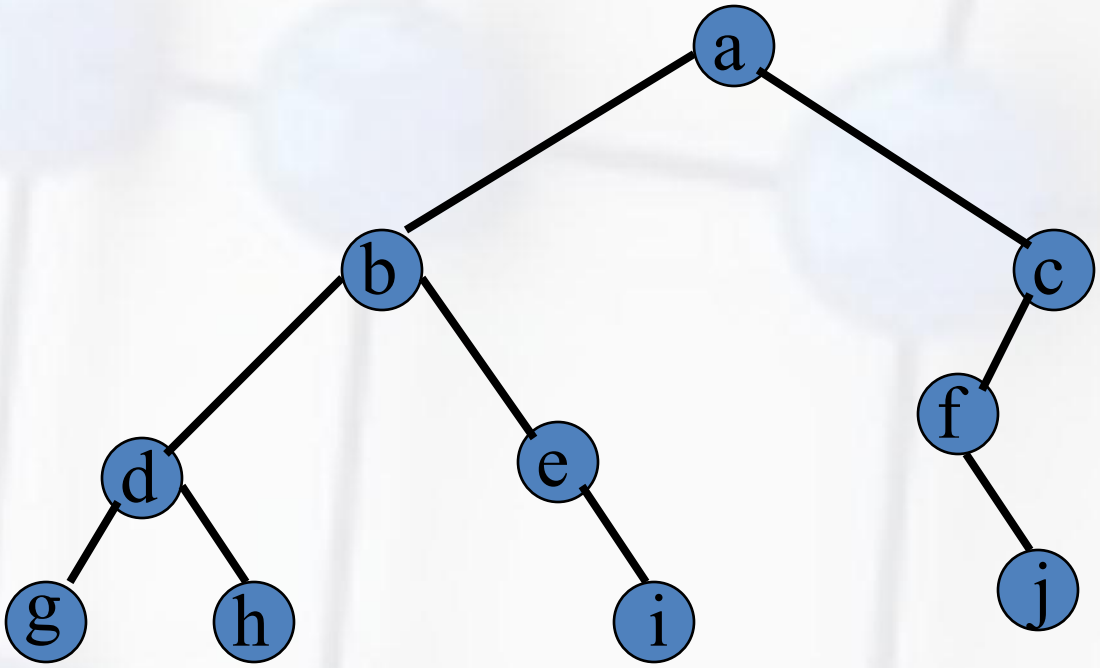
Postorder Of Expression Tree



a b + c d - * e f + /

Gives postfix form of expression!

Traversal Applications



- Make a clone.
- Determine height.
- Determine number of nodes.

Level Order

Let ptr be a pointer to the tree root.

while (ptr != NULL)

{

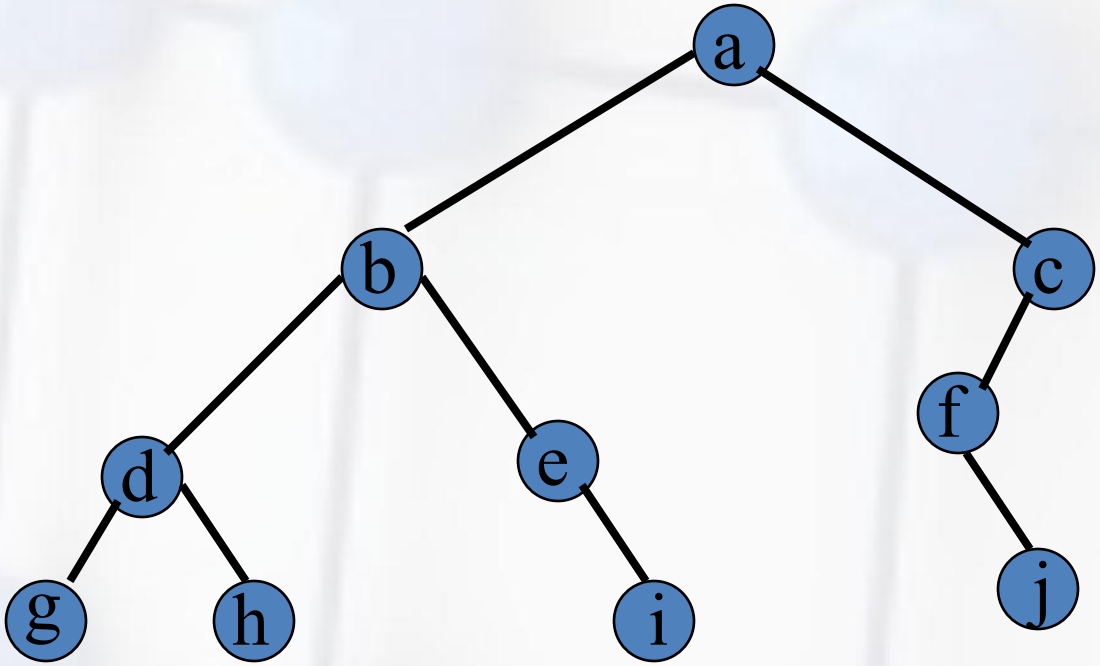
visit node pointed at by ptr and put its children on a FIFO queue;

if FIFO queue is empty, set ptr = NULL;

otherwise, delete a node from the FIFO queue and call it ptr;

}

Level-Order Example (Visit = print)



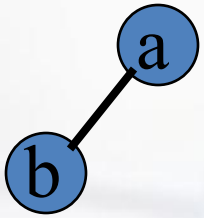
a b c d e f g h i j

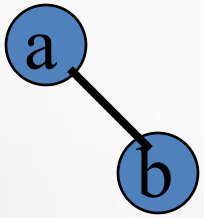
Binary Tree Construction

- Suppose that the elements in a binary tree are distinct.
- Can you construct the binary tree from which a given traversal sequence came?
- When a traversal sequence has more than one element, the binary tree is not uniquely defined.
- Therefore, the tree from which the sequence was obtained cannot be reconstructed uniquely.

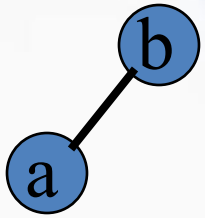
Some Examples

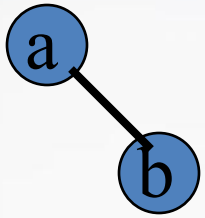
preorder = ab



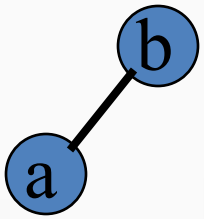


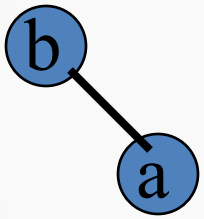
inorder = ab



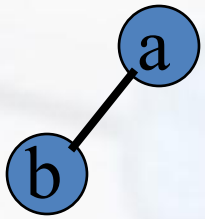


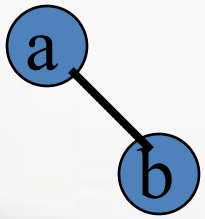
postorder = ab





level order = ab





Binary Tree Construction

- Can you construct the binary tree, given two traversal sequences?
- Depends on which two sequences are given.

Preorder And Postorder

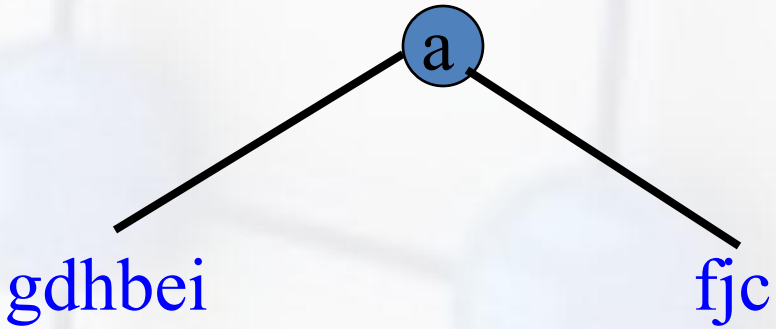
preorder = ab

postorder = ba

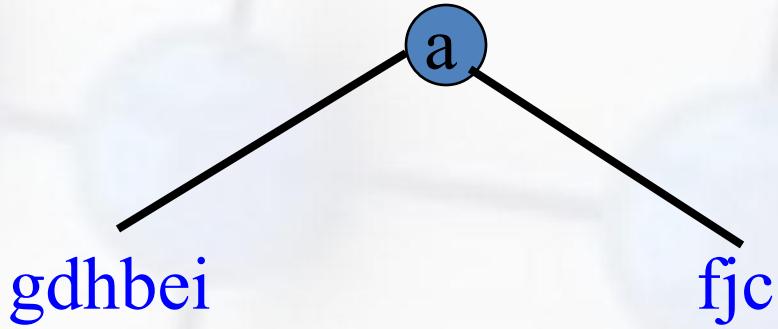
- Preorder and postorder do not uniquely define a binary tree.
- Nor do preorder and level order (same example).
- Nor do postorder and level order (same example).

Inorder And Preorder

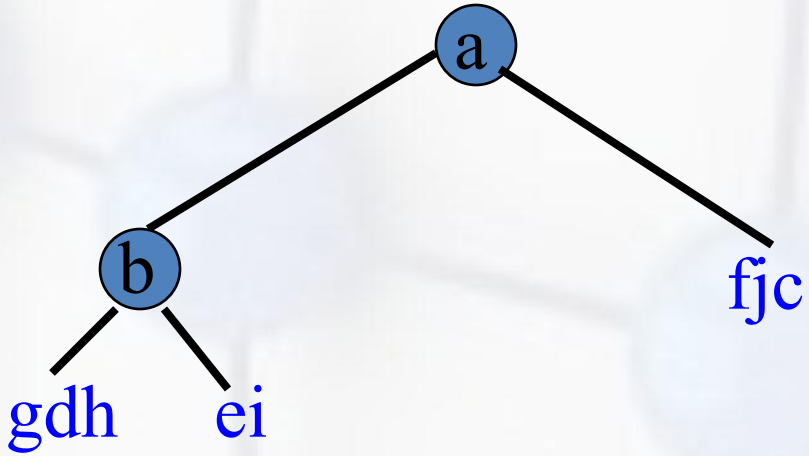
- inorder = g d h b e i a f j c
- preorder = a b d g h e i c f j
- Scan the preorder left to right using the inorder to separate left and right subtrees.
- a is the root of the tree; gdhbei are in the left subtree; fjc are in the right subtree.



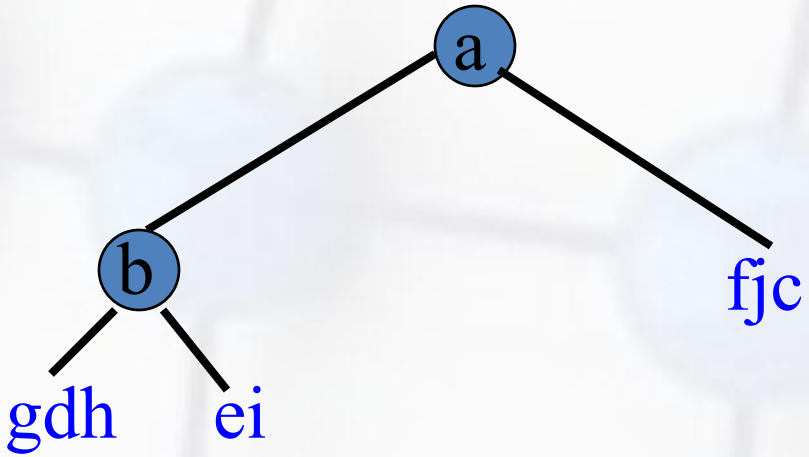
Inorder And Preorder



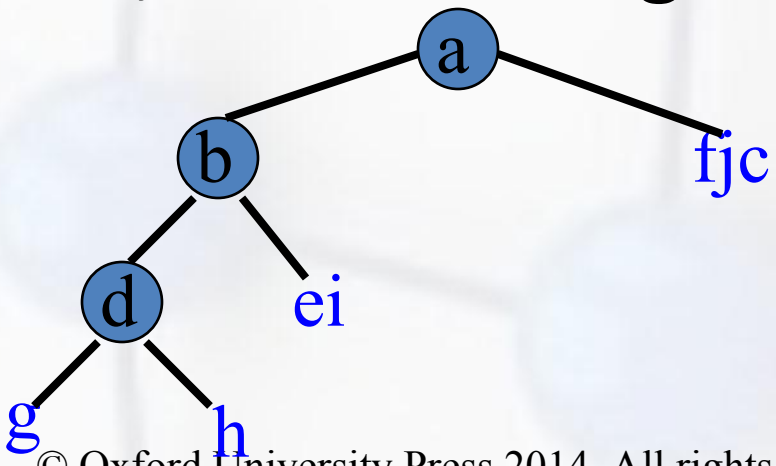
- preorder = a b d g h e i c f j
- b is the next root; gdh are in the left subtree; ei are in the right subtree.



Inorder And Preorder



- preorder = a b d g h e i c f j
- d is the next root; g is in the left subtree; h is in the right subtree.



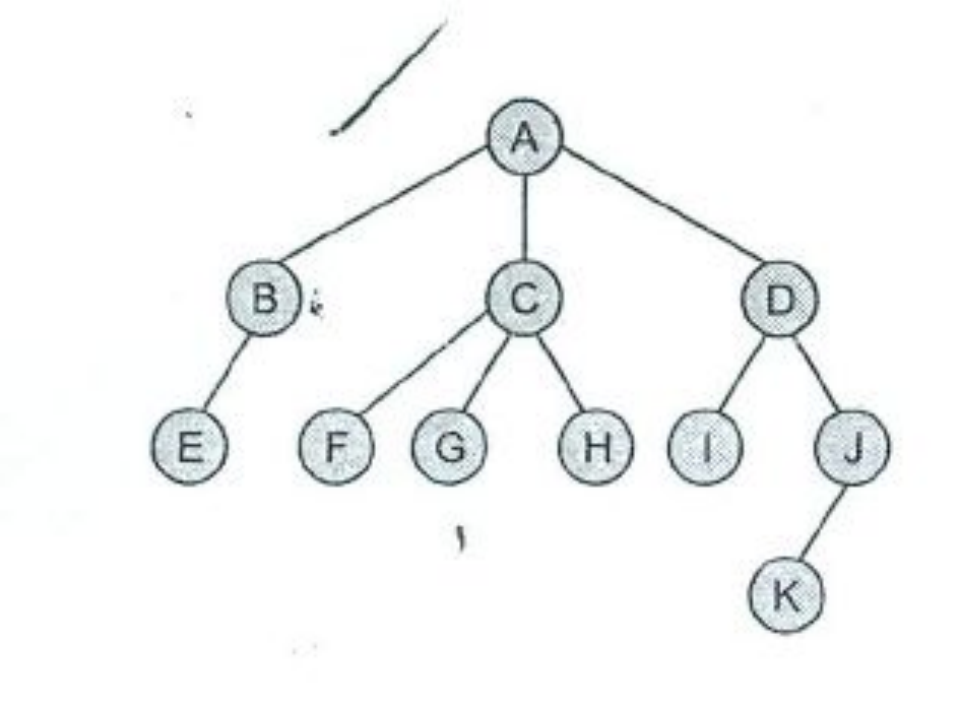
Inorder And Postorder

- Scan postorder from right to left using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- postorder = g h d i e b j f c a
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

Inorder And Level Order

- Scan level order from left to right using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- level order = a b c d e f g h i j
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

Creating a binary tree from a general tree



Rules

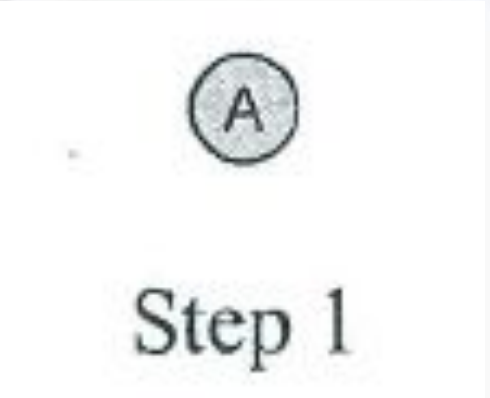
Rule 1: Root of the binary tree = Root of the general tree

Rule 2: Left child of a node = Leftmost child of the node
in the binary tree in the general tree

Rule 3: Right child of a node
in the binary tree = Right sibling of the node in the general tree

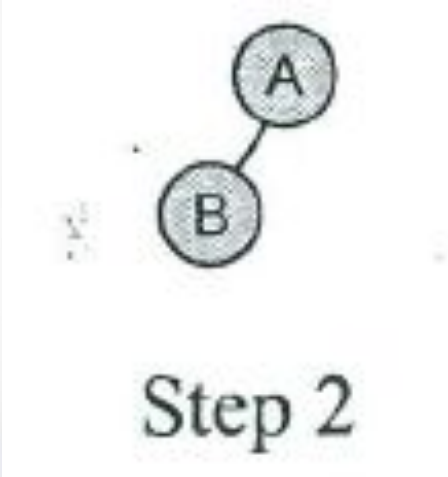
Step -1

Step 1: Node A is the root of the general tree, so it will also be the root of the binary tree.



Step-2

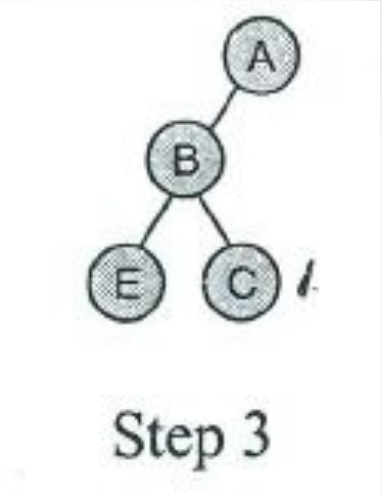
Step 2: Left child of node A is the leftmost child of node A in the general tree and right child of node A is the right sibling of the node A in the general tree. Since node A has no right sibling in the general tree, it has no right child in the binary tree.



Step 2

Step-3

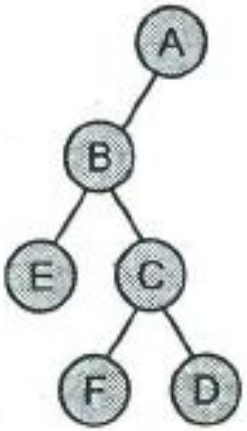
Step 3: Now process node B. Left child of B is E and its right child is C (right sibling in general tree).



Step 3

Step-4

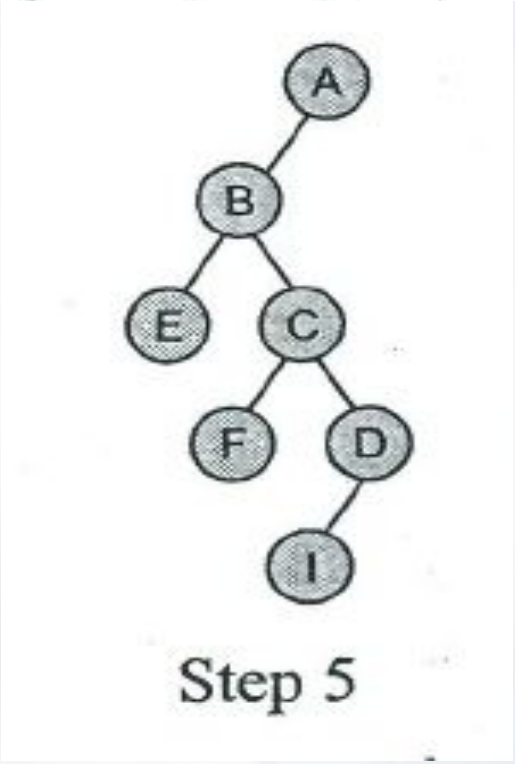
Step 4: Now process node C. Left child of C is F (leftmost child) and its right child is D (right sibling in general tree).



Step 4

Step-5

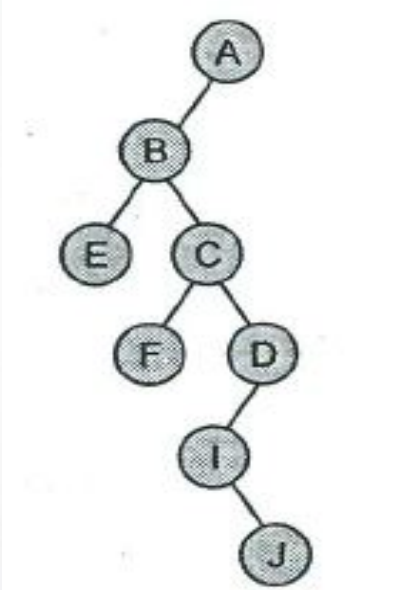
Step 5: Now process node D. Left child of D is I (leftmost child). There will be no right child of D because it has no right sibling in the general tree.



Step 5

Step-6

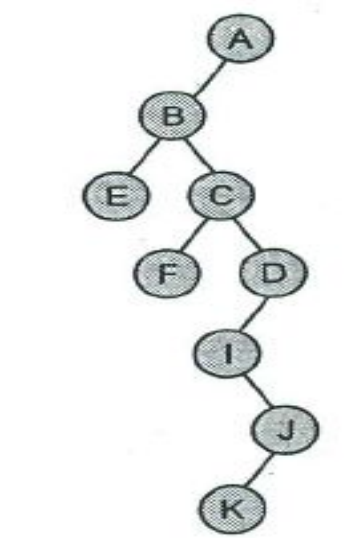
Step 6: Now process node I. There will be no left child of I in the binary tree because I has no left child in the general tree. However, I has a right sibling J, so it will be added as the right child of I.



Step 6

Step-7

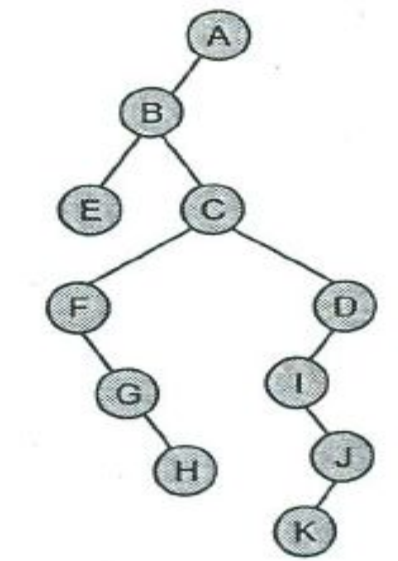
Step 7: Now process node J. Left child of J is K (leftmost child). There will be no right child of J because it has no right sibling in the general tree.



Step 7

Step-8

Step 8: Now process all the unprocessed nodes (E, F, G, H, K) in the same fashion, so the resultant binary tree can be given as follows.



Step 8

Chapter 11

MULTI WAY SEARCH TREE

INTRODUCTION

- Binary Search tree contain one value and two pointers left and right
- Which points to node s left and right sub tree

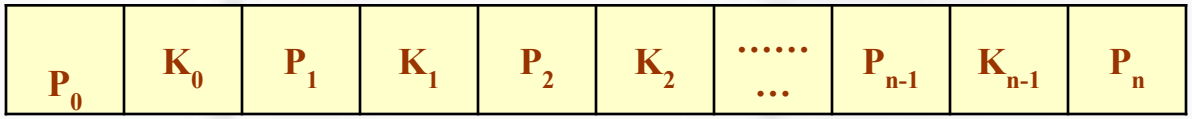
Same concept in M way search trees

Which has m-1 values per node and M sub trees

M is called degree of the Tree.

Introduction

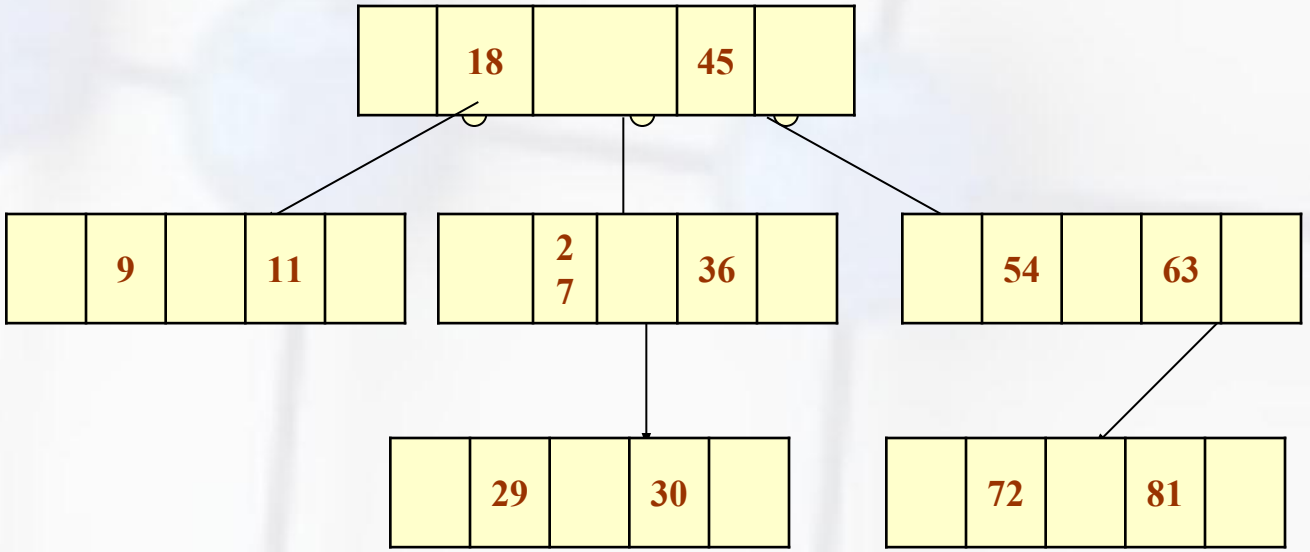
- In such a tree M is called the degree of the tree. Note in a binary search tree M = 2, so it has one value and 2 sub-trees. In other words, every internal node of an M-way search tree consists of pointers to M sub-trees and contains M – 1 keys, where M > 2.



- In the structure, P0, P1, P2,…, Pn are pointers to the node's sub-trees and K0, K1, K2,…, Kn-1 are the key values of the node. All the key values are stored in ascending order. That is, Ki < Ki+1 for 0 ≤ i ≤ n-2.

- In an M-way search tree, it is not compulsory that every node has exactly (M-1) values and have exactly M sub-trees. Rather, the node can have anywhere from 1 to (M-1) values, and the number of sub-trees may vary from 0 (for a leaf node) to 1 + *i*, where *i* is the number of key values in the node. M is thus a *fixed upper limit* that defines how much key values can be stored in the node.

Introduction



- In the structure, P0, P1, P2,…, Pn are pointers to the node's sub-trees and K0, K1, K2,…, Kn-1 are the key values of the node. All the key values are stored in ascending order. That is, Ki < Ki+1 for 0 ≤ i ≤ n-2.

- In an M-way search tree, it is not compulsory that every node has exactly (M-1) values and have exactly M sub-trees. Rather, the node can have anywhere from 1 to (M-1) values, and the number of sub-trees may vary from 0 (for a leaf node) to 1 + *i*, where *i* is the number of key values in the node. M is thus a *fixed upper limit* that defines how much key values can be stored in the node.

B-Trees

- A B-tree is a specialized m- way tree that is widely used for disk access.
- Used to index the Data in a Data base of very large size and provide fast Access.
- B tree of order m can have maximum m-1 keys and m pointers to its sub-trees. A B-tree may contain a large number of key values and pointers to sub-trees. Storing a large number of keys in a single node keeps the height of the tree relatively small.
- A B-tree is designed to store sorted data and allows search, insert, and delete operations to be performed in logarithmic amortized time. A B-tree of order *m* (the maximum number of children that each node can have) is a tree with all the properties of an m-way search tree and in addition has the following properties:
- Every node in the B-tree has at most (maximum) *m* children.

B-Trees

A B-tree of order m can be defined as an m-way search tree which is either empty or satisfies the following properties-

(i) All leaf nodes are at the same level.

(ii) All non leaf nodes (except root node) should have at least $\lceil m/2 \rceil$ children.

(iii) All nodes (except root node) should have at least $\lceil m/2 \rceil - 1$ keys.

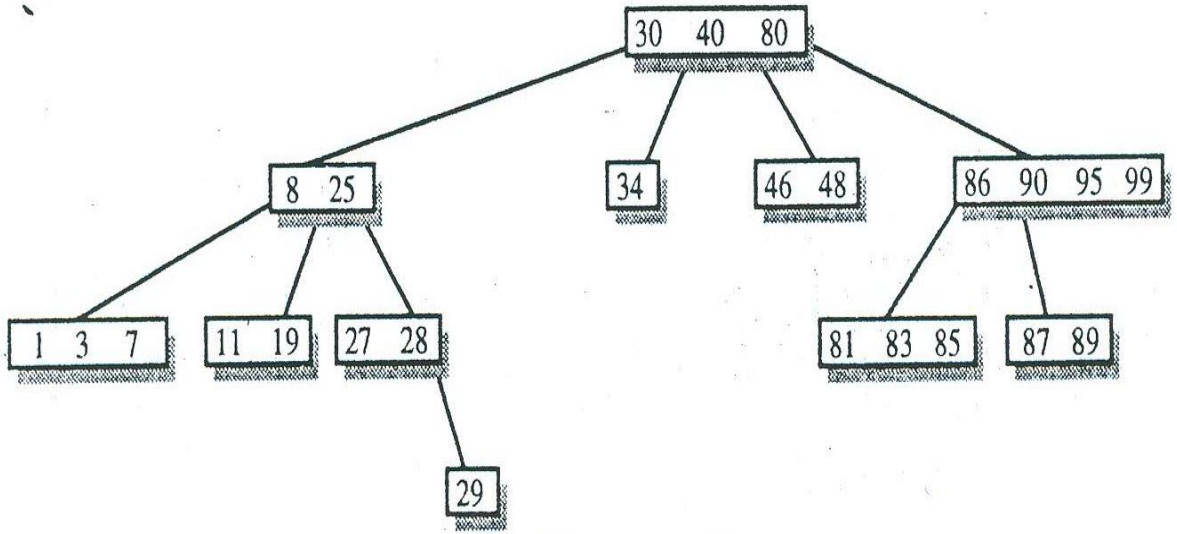
(iv) If the root node is a leaf node(only node in the tree), then it will have no children and will have at least one key. If the root node is a non leaf node, then it will have at least 2 children and at least one key.

(v) A non leaf node with n-1 key values should have n non NULL children.

The following figure shows the minimum and maximum number of children in any non root and non leaf node of B-trees of different orders.

Order of the tree	Minimum Children	Maximum Children
3	2	3
4	2	4
5	3	5
6	3	6
7	4	7
.....	.....	.....
M	⌈M/2⌉	M

The tree in figure 6.139 is a multiway search tree of order 5.



(i) The leaf nodes in this tree are [1,3,7], [11,19], [29], [34], [46,48], [81,83,85], [87,89] and it can be clearly seen that they are not at the same level.

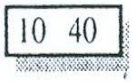
(ii)The non leaf node [27, 28] has 2 keys but only one non NULL child, and the non leaf node [86,90,95,99] has 4 keys but only 2 non NULL children.

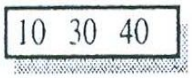
(iii) The minimum number of keys for a B-tree of order 5 is $\lceil 5/2 \rceil - 1 = 2$ while in the above tree there are 2 nodes [34], [29] which have less than 2 keys.

insertion

10, 40, 30, 35, 20, 15, 50, 28, 25, 5, 60, 19, 12, 38, 27, 90, 45, 48

(a) Insert 10 

(b) Insert 40 

(c) Insert 30 

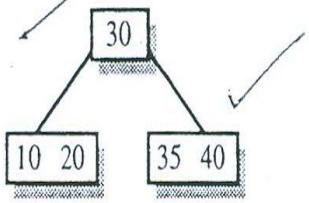
30 will be inserted between 10 and 40 since all the keys in a node should be in ascending order.

(d) Insert 35 

The maximum number of permissible keys for a node of a B-tree of order 5 is 4, so now after the insertion of 35, this node has become full.

(e) Insert 20

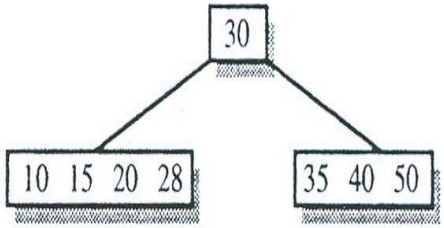
The node [10, 30, 35, 40] will become overfull [10, 20, **30**, 35, 40], so splitting is done at the median key 30. A new node is allocated and keys to the right of median key i.e. 35 and 40 are moved to the new node and the keys to the left of the median key i.e. 10 and 20 remain in the same node. Generally after splitting, the median key goes to the parent node, but here the node that is being split is the root node, so a new node is allocated which contains the median key and now this new node becomes the root of the tree.



Since the root node has been split, the height of the tree has increased by one.

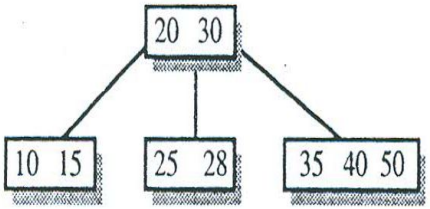
(f) Insert 15, 50, 28

15 and 28 are less than 30 so they are inserted in the left node at appropriate place and 50 is greater than 30 so it is inserted in the right node.

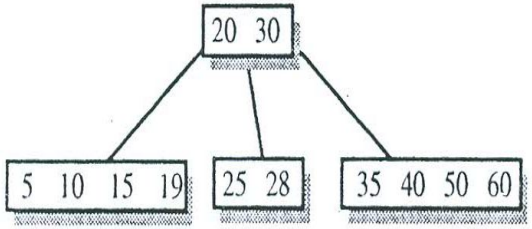


(g) Insert 25

The node [10, 15, 20, 28] will become overfull [10, 15, **20**, 25, 28], so splitting is done and the median key 20 goes to the parent node.

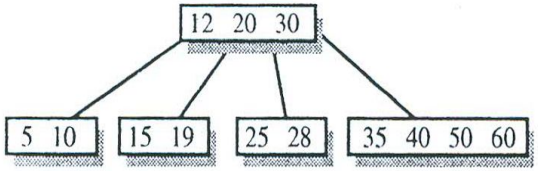


(h) Insert 5, 60, 19



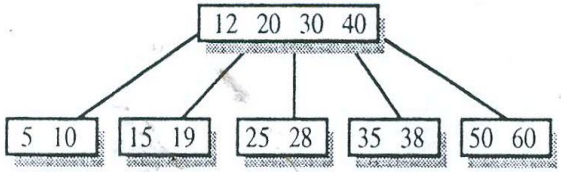
(i) Insert 12

The node [5, 10, 15, 19] will become overfull [5, 10, **12**, 15, 19], so splitting is done and the median key 12 goes to the parent node.

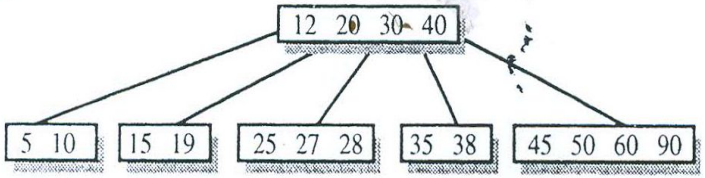


(j) Insert 38

The node[35, 40, 50, 60] will become overfull [35, 38, **40**, 50, 60], so splitting is done and the median key 40 goes to the parent node.

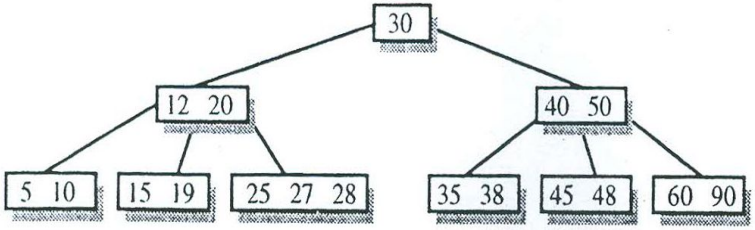


(k) Insert 27, 90, 45

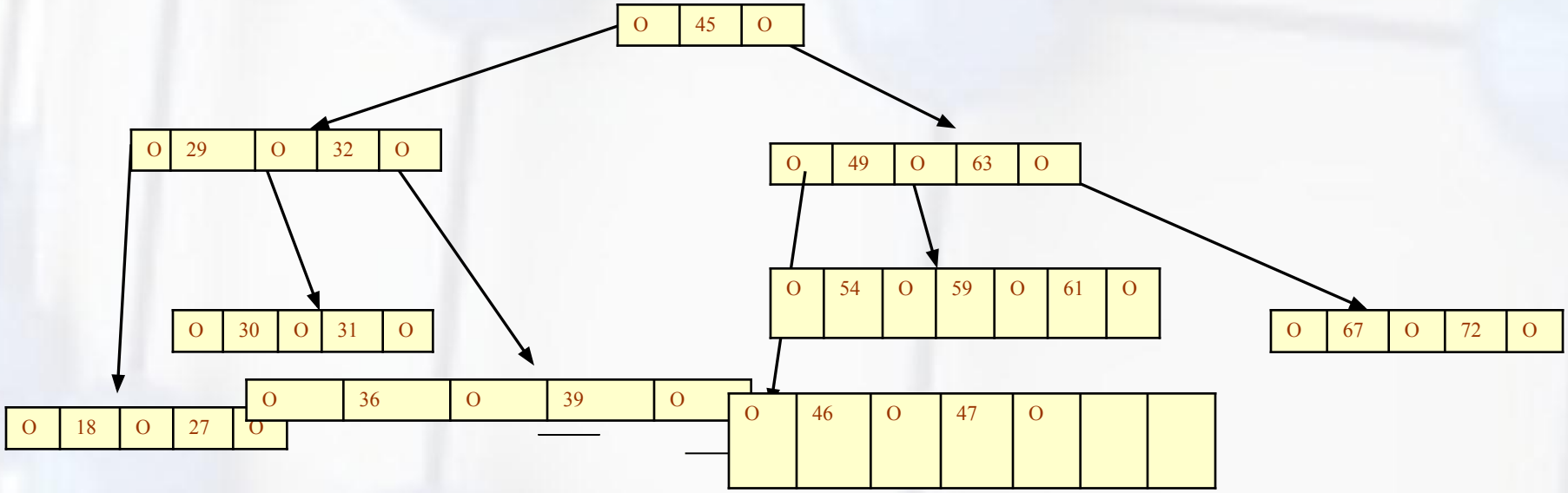


(l) Insert 48

The node [45, 50, 60, 90] will become overfull [45, 48, **50**, 60, 90], so splitting is done and the median key 50 goes to the parent node. After insertion of 50 the parent node also becomes overfull [12, 20, **30**, 40, 50] so again splitting is done and this time root node is splitted so a new root is formed and the tree becomes taller.



B-Trees



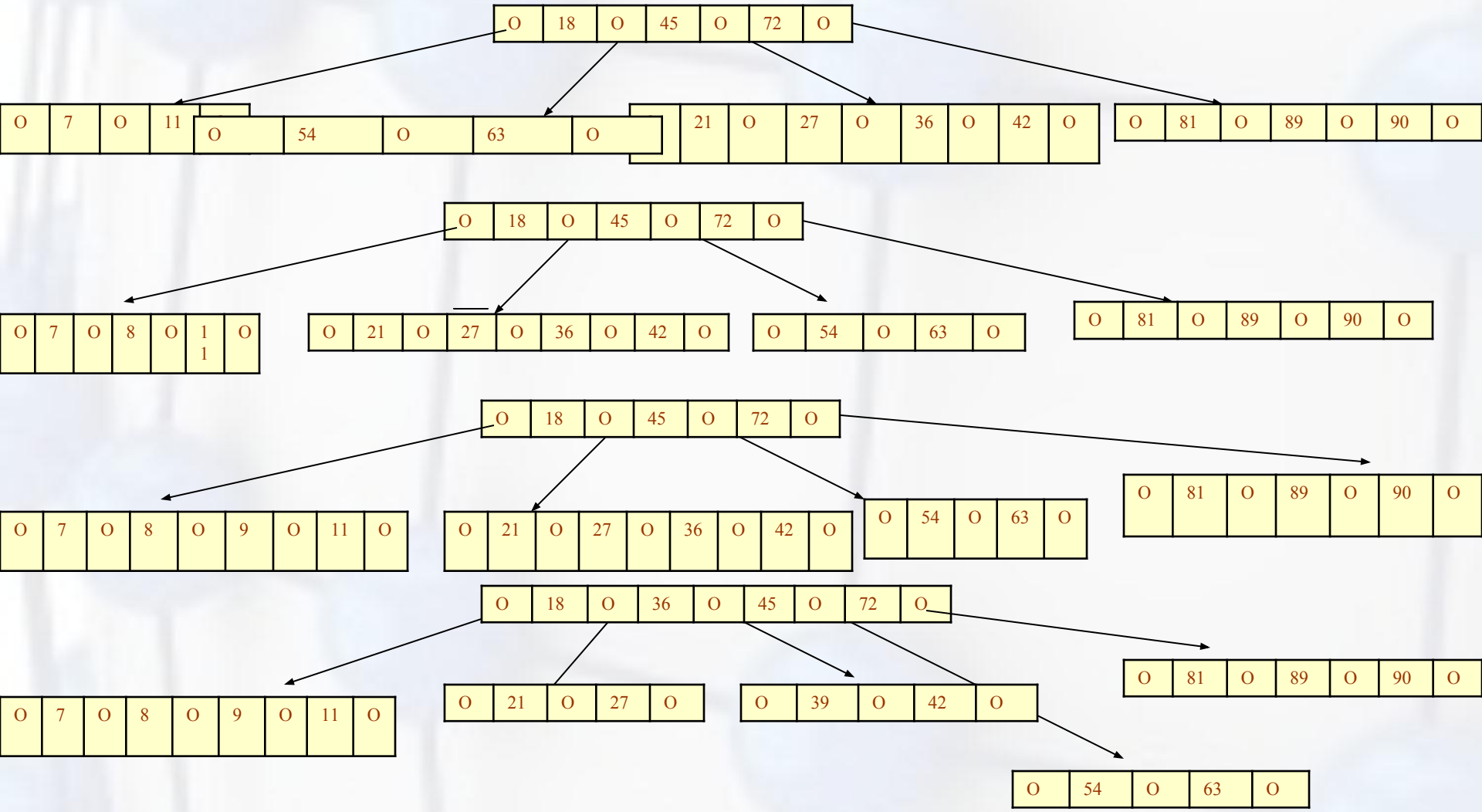
B-Trees

Insert Operation

- In a B tree all insertions are done at the leaf node level. A new value is inserted in the B tree using the algorithm given below.
- Search the B tree to find the leaf node where the new key value should be inserted.
- If the leaf node is not full, that is it contains less than m-1 key values then insert the new element in the node, keeping the node's elements ordered.
- If the leaf node is full, that is the leaf node already contain m-1 key values then insert the new value in order into the existing set of keys split the node at its median into two nodes. note that the split nodes are half full. Push the median element up to its parent’s node. If the parent’s node is already full, then split the parent node by following the same steps.

B-Trees

Example: Look at the B tree of order 5 given below and insert 8, 9, 39 into it.



Deletion

- Like insertion, deletion is also done from the leaf nodes. There are two cases of deletion. First, a leaf node has to be deleted. Second, an internal node has to be deleted. Let us first see the steps involved in deleting a leaf node.
- Locate the leaf node which has to be deleted
- If the leaf node contains more than minimum number of key values (more than m/2 elements), then delete the value.
- Else, if the leaf node does not contain even m/2 elements then, fill the node by taking an element either from the left or from the right sibling

  If the left sibling has more than the minimum number of key values (elements), push its largest key into its parent's node and pull down the intervening element from the parent node to the leaf node where the key is deleted. Else if the right sibling has more than the minimum number of key values (elements), push its smallest key into its parent node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

Deletion

- Else, if both left and right siblings contain only minimum number of elements, then create a new leaf node by combining the two leaf nodes and the intervening element of the parent node (ensuring that the number of elements do not exceed the maximum number of elements a node can have, that is, m). If pulling the intervening element from the parent node leaves it with less than minimum number of keys in the node, then propagate the process upwards thereby reducing the height of the B tree.

- To delete an internal node, promote the successor or predecessor of the key to be deleted to occupy the position of the deleted key. This predecessor or successor will always be in the leaf node. So further the processing will be done as if a value from the leaf node has been deleted.

- Deletion in a B tree can be classified in two cases
- Deletion from leaf node
- Deletion from non leaf node

(a) Delete 7, 52 from tree in figure 6.142

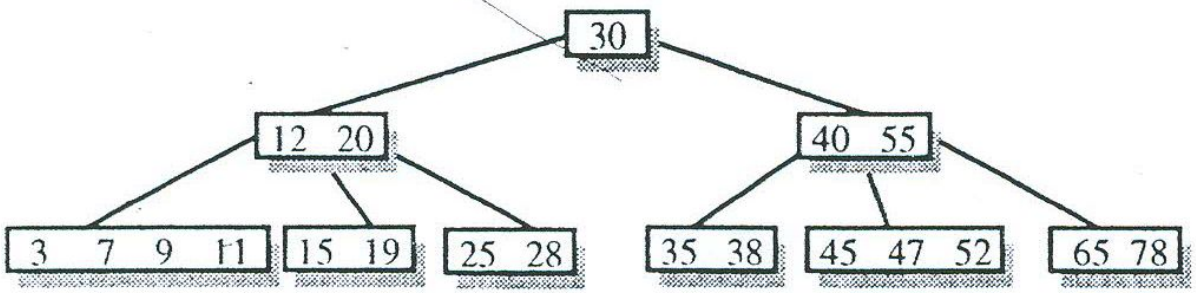


Figure 6.142

IF NODe has MIN Keys

(b) Delete 15 from tree in figure 6.143.

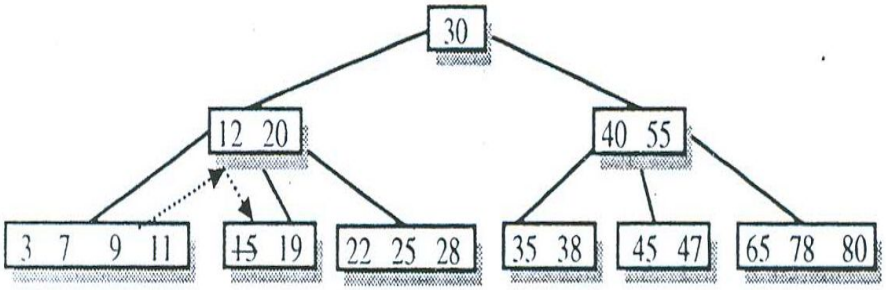
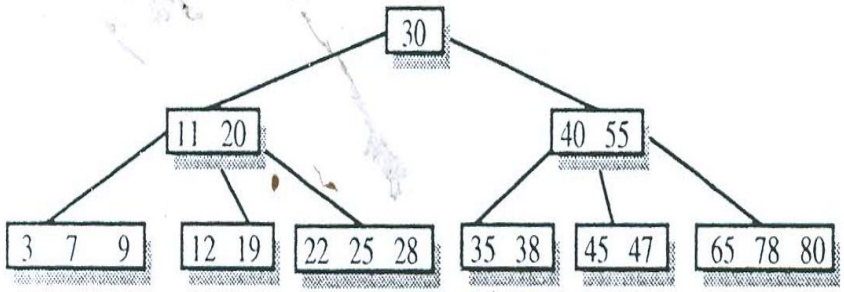
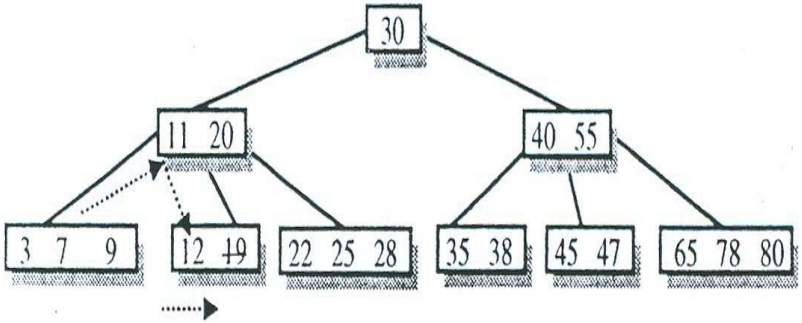


Figure 6.143

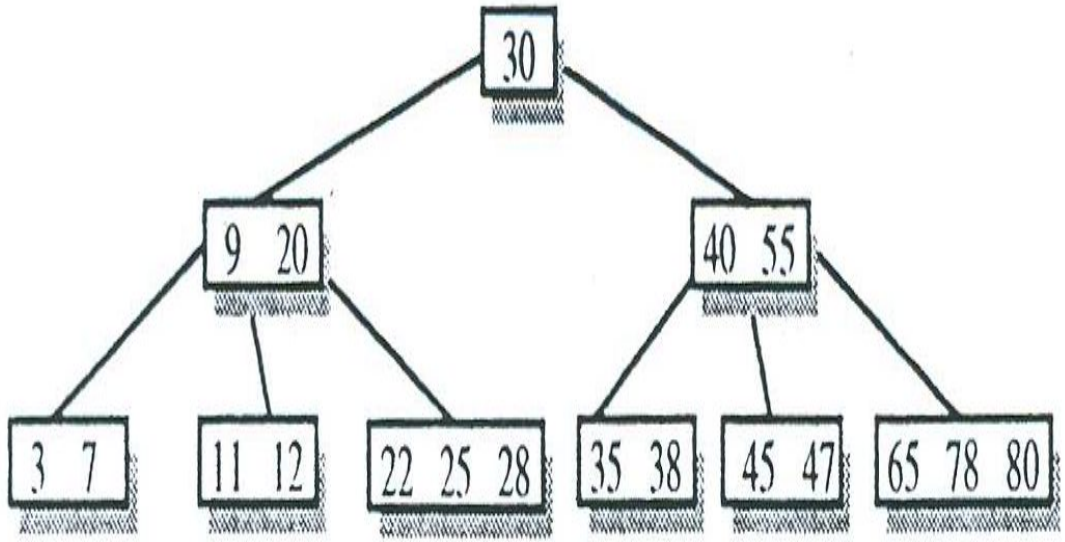
Here key 15 is to be deleted from node [15, 19], since this node has only MIN keys we will try to borrow from its left sibling [3, 7, 9, 11] which has more than MIN keys. The parent of these nodes is node [12, 20] and the separator key is 12. So the last key of left sibling(11) is moved to the place of separator key and the separator key is moved to the underflow node. The resulting tree after deletion of 15 will be-



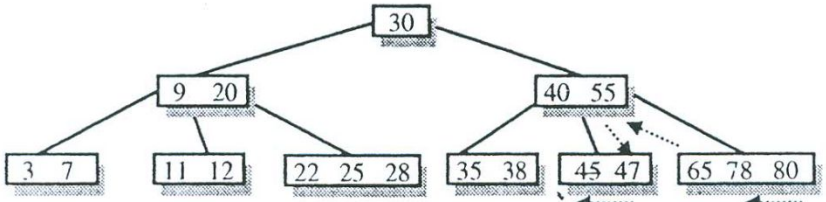
(c) Delete 19 from the tree in figure 6.144



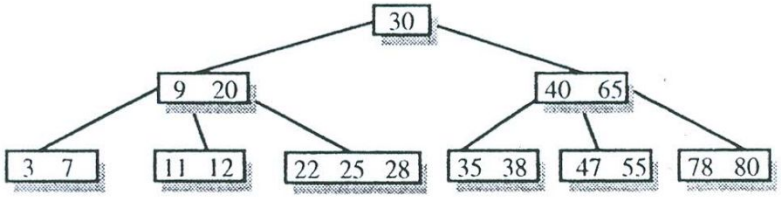
We will borrow from the left sibling so key 9 will be moved up to the parent node and 11 will be shifted to the underflow node. In the underflow node the key 12 will be shifted to the right to make place for 11. The resulting tree is-



(d) Delete 45 from tree in figure 6.145



The left sibling of [45,47] is [35,38] which has only MIN keys so we can't borrow from it, hence we will try to borrow from the right sibling [65, 78, 80]. The first key of the right sibling(65) is moved to the parent node and the separator key from the parent node(55) is moved to the underflow node. In the underflow node, 47 is shifted left to make room for 55. In the right sibling, 78 and 80 are moved left to fill the gap created by removal of 65. The resulting tree is-



If both left and right siblings of underflow node have MIN keys, then we can't borrow from any of the siblings. In this case, the underflow node is combined with its left (or right) sibling.

(e) Delete 28 from the tree in figure 6.146.

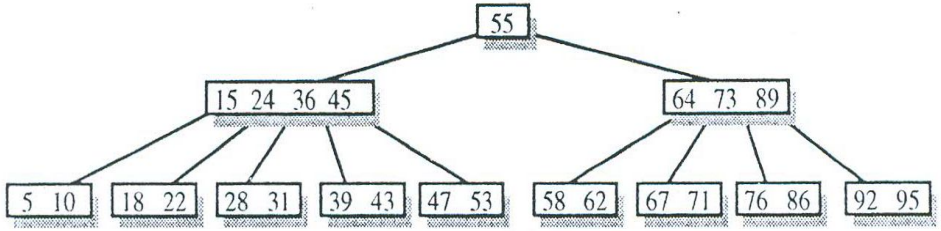
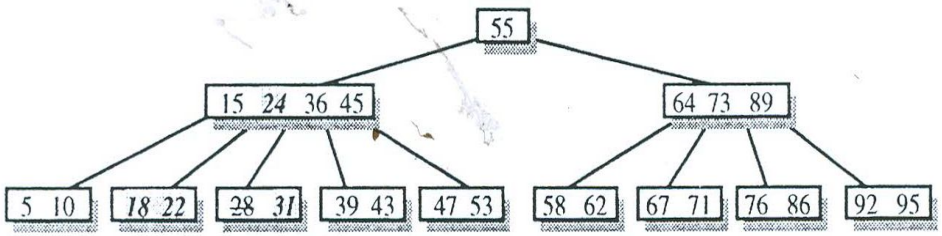
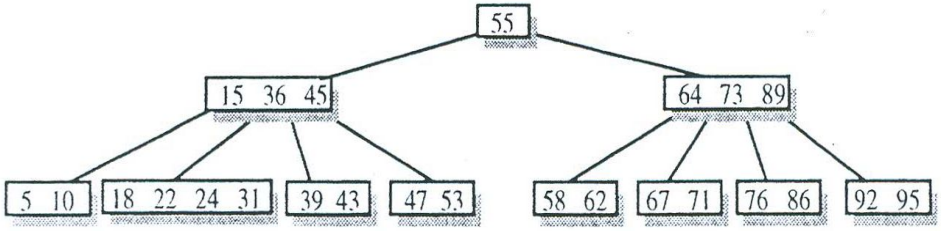


Figure 6.146

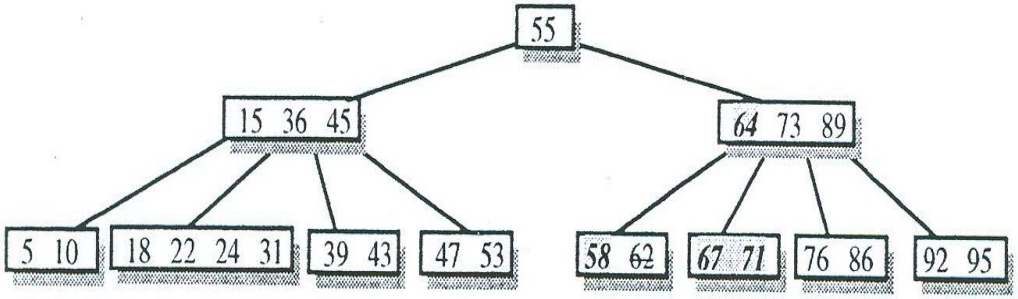
We can see that the node [28,31] has only MIN keys so we'll try to borrow from left sibling [18, 22], but it also has MIN keys so we'll look at the right sibling [39,43] which also has only MIN keys. So after deletion of 28 we'll combine the underflow node with its left sibling. For combining these two nodes the separator key(24) from the parent node will move down in the combined node.



The resulting tree after deletion of 28 is-



(f) Delete 62 from the tree in figure 6.147.



Here the key is to be deleted from [58, 62] which is leftmost child of its parent, and hence it has no left sibling. So here we'll look at the right sibling for borrowing a key, but the right sibling has only MIN keys, so we'll delete 62 and combine the underflow node with the right sibling. The resulting tree after deletion of 62 is-

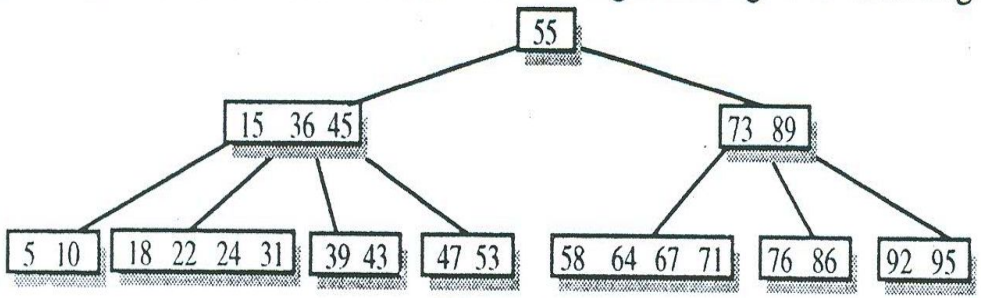


Figure 6.148