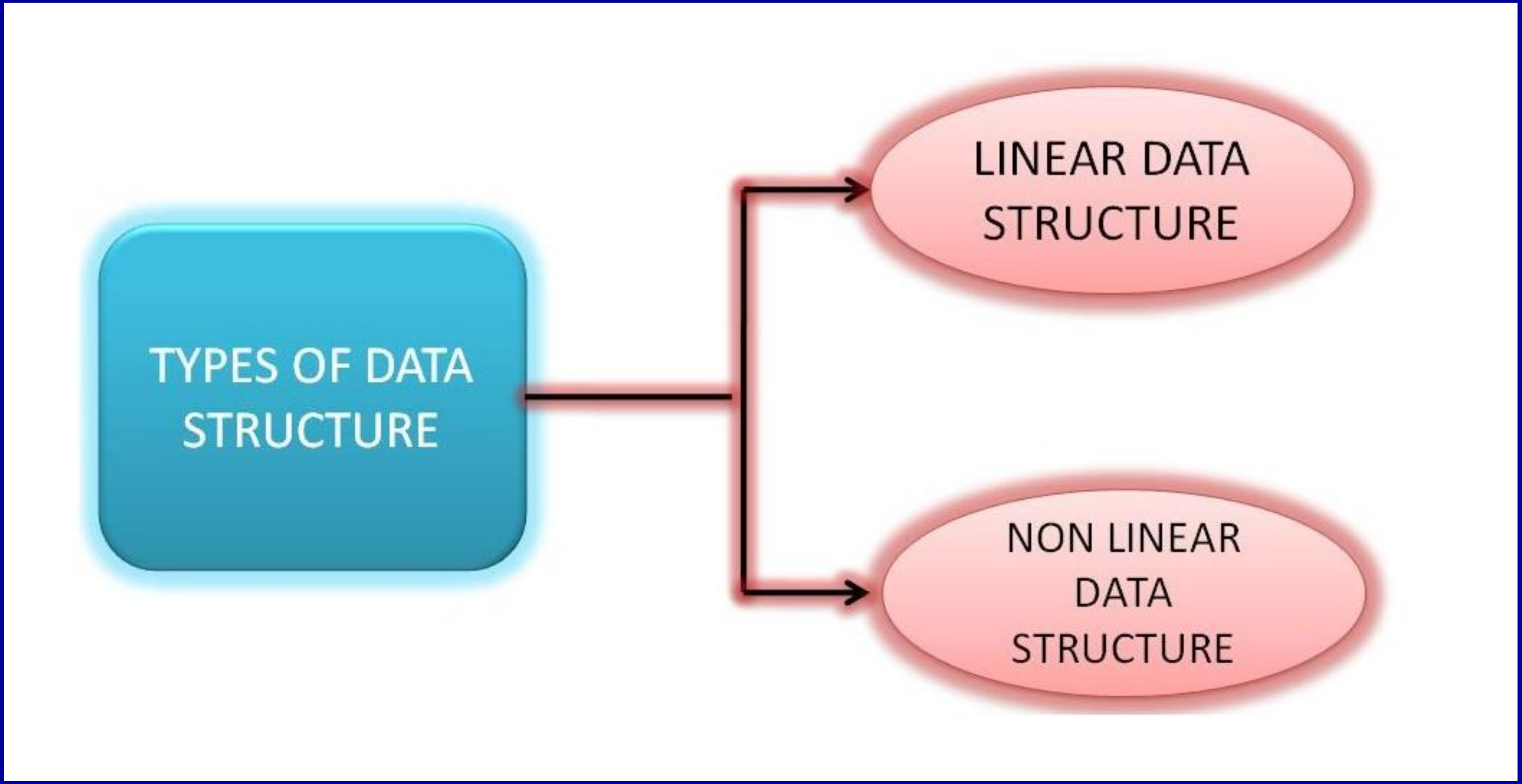
CSE-s201 - Data Structures

Dr. Ravindra Nath
CSJM University Kanpur

Topics

- Types of Data Structures
- Examples for each type
- Tree Data Structure
- Basic Terminology
- Tree ADT
- Traversal Algorithms
- Binary Trees
  - Binary Tree Representations
  - Binary Tree ADT
  - Binary Tree Traversals

Types of Data Structure



Linear Data Structures

- In linear data structure, member elements form a sequence. Such linear structures can be represented in memory by using one of the two basic strategies
- By having the linear relationship between the elements represented by means of sequential memory locations. These linear structures are called arrays. By having relationship between the elements represented by pointers. These structures are called linked lists.

Dr. Ravindra Nath UIET CSJM
UNIVERSITY KANPUR UP INDIA

When to use them

- Arrays are useful when number of elements to be stored is fixed.
- Operations like traversal searching and sorting can easily be performed on arrays.
- On the other hand, linked lists are useful when the number of data items in the collection are likely to change.

Nonlinear Data Structures

- In nonlinear data structures, data elements are not organized in a sequential fashion. A data item in a nonlinear data structure could be attached to several other data elements to reflect a special relationship among them and all the data items cannot be traversed in a single run.
- Data structures like multidimensional arrays, trees and graphs are some examples of widely used nonlinear data structures.

Examples

- A Multidimensional Array is simply a collection of one-dimensional arrays.
- A Tree is a data structure that is made up of a set of linked nodes, which can be used to represent a hierarchical relationship among data elements.
- A Graph is a data structure that is made up of a finite set of edges and vertices. Edges represent connections or relationships among vertices that stores data elements.

Difference

- Main difference between linear and nonlinear data structures lie in the way they organize data elements.
- In linear data structures, data elements are organized sequentially and therefore they are easy to implement in the computer's memory.
- In nonlinear data structures, a data element can be attached to several other data elements to represent specific relationships that exist among them.

Difficult to Implement

- Due to this nonlinear structure, they might be difficult to implement in computer's linear memory compared to implementing linear data structures.
- Selecting one data structure type over the other should be done carefully by considering the relationship among the data elements that needs to be stored.

A Specific Example

- Imagine that you are hired by company XYZ to organize all of their records into a computer database.
- The first thing you are asked to do is create a database of names with all the company's management and employees.
- To start your work, you make a list of everyone in the company along with their position and other details.

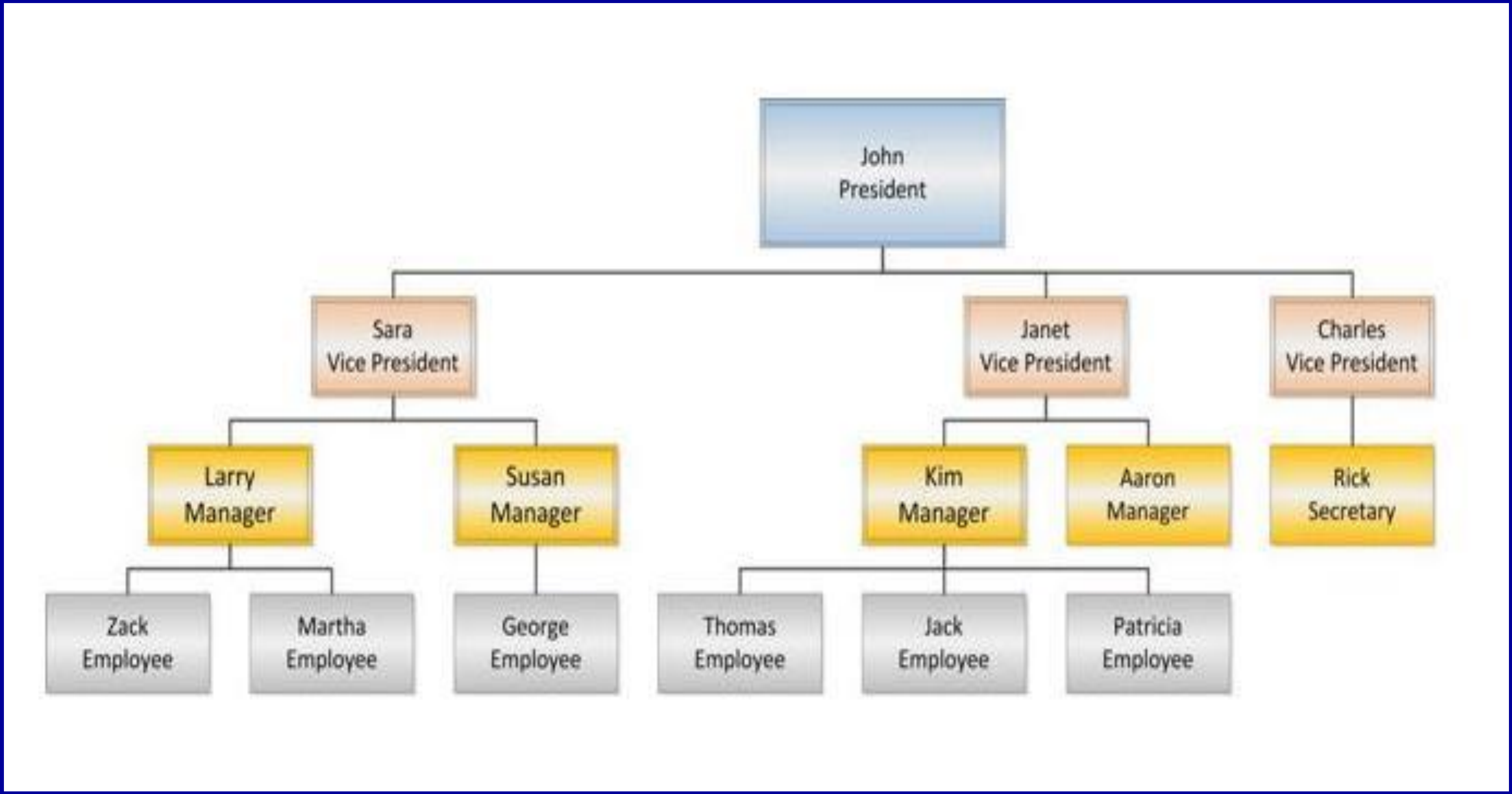
Employees Table

Name	Position
Aaron	Manager
Charles	VP
George	Employee
Jack	Employee
Janet	VP
John	President
Kim	Manager
Larry	Manager
Martha	Employee
Patricia	Employee
Rick	Secretary
Sarah	VP
Susan	Manager
Thomas	Employee
Zack	Employee

Disadvantages of Tables

- But this list only shows one view of the company. You also want your database to represent the relationships between management and employees at XYZ.
- Although your list contains both name and position, it does not tell you which managers are responsible for which workers and so on.
- After thinking about the problem for a while, you decide that a tree diagram is a much better structure for showing the work relationships at XYZ.

Better Representation



Comparison

- These two diagrams are examples of different data structures.
- In one of the data structures, your data is organized into a list. This is very useful for keeping the names of the employees in alphabetical order so that we can locate the employee's record very quickly.
- However, this structure is not very useful for showing the relationships between employees. A tree structure is much better suited for this purpose.

Tree Data Structure

- A Tree is a nonlinear data structure that models a hierarchical organization.
- The characteristic features are that each element may have several successors (called its “children”) and every element except one (called the “root”) has a unique predecessor (called its “parent”).
- Trees are common in computer science: Computer file systems are trees, the inheritance structure for C++/Java classes is a tree.

What is a Tree ?

- In Computer Science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization Charts
  - File Systems
  - Programming Environment

Tree Definition

- Here is the recursive definition of an (unordered) tree:
  - A *tree* is a pair (*r*, *S*), where *r* is a node and *S* is a set of disjoint trees, none of which contains *r*.
- The node *r* is called the *root* of the tree *T*, and the elements of the set *S* are called its *subtrees*.
- The set *S*, of course, may be empty.
- The elements of a tree are called its *nodes*.

Root, Parent and Children

- If $T = (x, S)$ is a tree, then
  - x is the root of T and
  - S is its set of subtrees $S = \{T_1, T_2, T_3, \ . \ . \ ., T_n\}$.
- Each subtree T_j is itself a tree with its own root r_j .
- In this case, we call the node r the *parent* of each node r_j, and we call the r_j the *children* of r. In general.

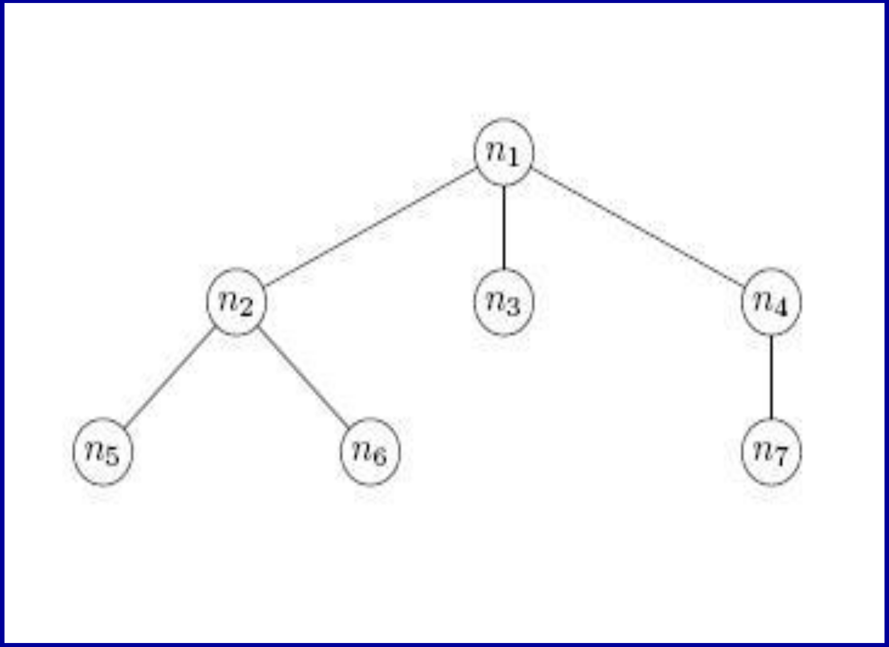
Dr. Ravindra Nath UIET CSJM UNIVERSITY KANPUR UP INDIA

Basic Terminology

- A node with no children is called a *leaf*. A node with at least one child is called an *internal node*.
- The Node having further sub-branches is called *parent node.*
- Every node *c* other than the root is connected by an edge to some one other node *p* called the parent of *c*.
- We also call *c* a child of *p*.

Dr. Ravindra Nath UIET CSJM
UNIVERSITY KANPUR UP INDIA

An Example



- n_1 is the parent of n_2 ,n_3 and n_4, while n_2 is the parent of n_5 and n_6. Said another way, n_2, n_3, and n_4 are children of n_1, while n_5 and n_6 are children of n_2.

Dr. Ravindra Nath UIET CSJM UNIVERSITY KANPUR UP INDIA

Connected Tree

- A tree is *connected* in the sense that if we start at any node n other than the root, move to the parent of n, to the parent of the parent of n, and so on, we eventually reach the root of the tree.
- For instance, starting at n_7, we move to its parent, n_4, and from there to n_4's parent, which is the root, n_1.

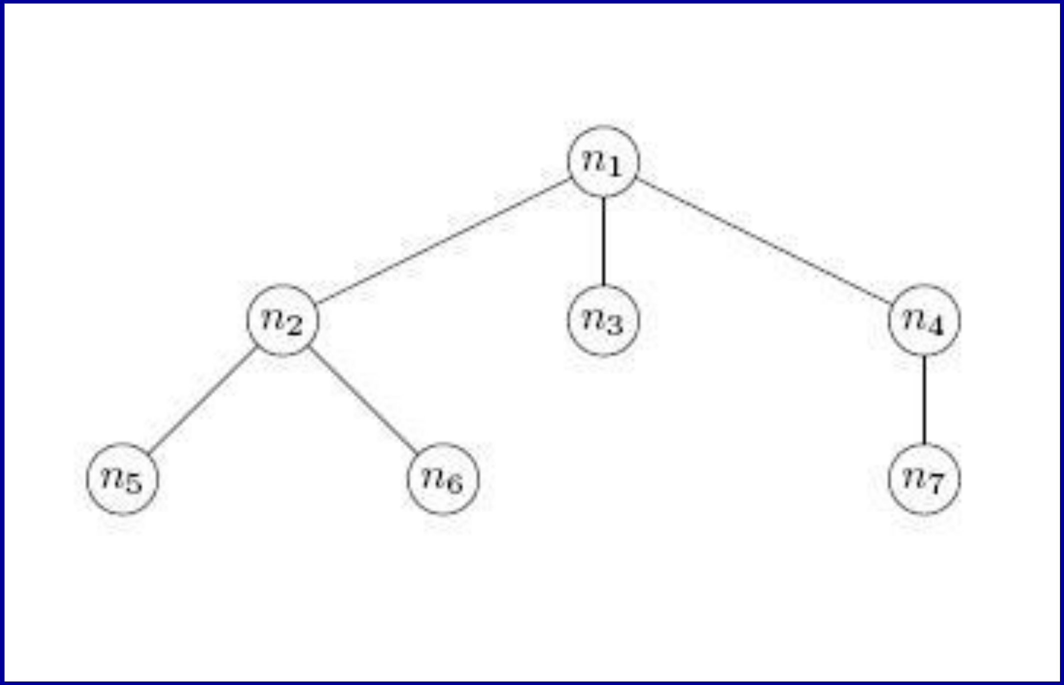
Ancestors and Descendants

- The parent-child relationship can be extended naturally to ancestors and descendants.
- Informally, the ancestors of a node are found by following the unique path from the node to its parent, to its parent's parent, and so on.
- The descendant relationship is the inverse of the ancestor relationship, just as the parent and child relationships are inverses of each other.

Path and Path Length

- More formally, suppose $m_1, m_2, . . . , m_k$ is a sequence of nodes in a tree such that m_1 is the parent of m_2, which is the parent of m_3, and so on, down to m_{k-1}, which is the parent of m_k. Then $m_1, m_2, . . . , m_k$ is called a *path* from m1 to m_k in the tree. The *path length* or length of the path is $k - 1$, one less than the number of nodes on the path.

In our Example



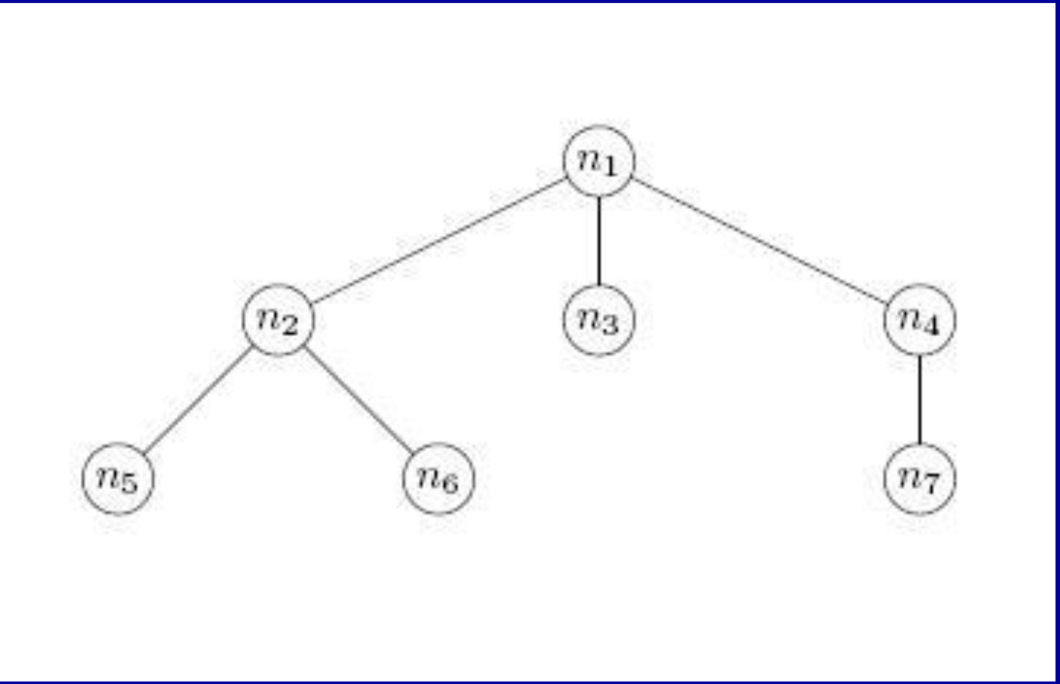
- n_1, n_2, n_6 is a path of length 2 from the root n_1 to the node n_6.

Dr. Ravindra Nath UIET CSJM UNIVERSITY KANPUR UP INDIA

Height and Depth

- In a tree, the *height* of a node n is the length of a longest path from n to a leaf. The height of the tree is the height of the root.
- The *depth*, or *level*, of a node n is the length of the path from the root to n.

In our Example

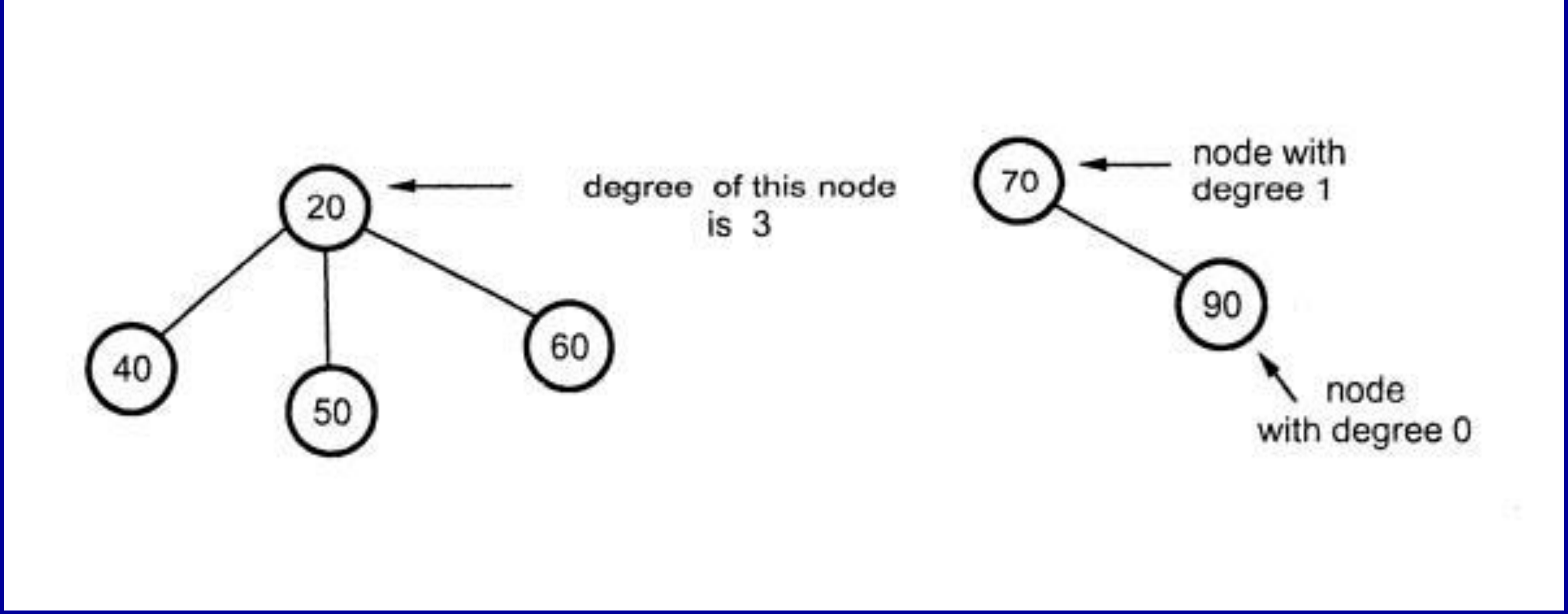


- node n_1 has height 2, n_2 has height 1, and leaf n_3 has height 0. In fact, any leaf has height 0. The height of the tree is 2. The depth of n_1 is 0, the depth of n_2 is 1, and the depth of n_5 is 2.

Other Terms

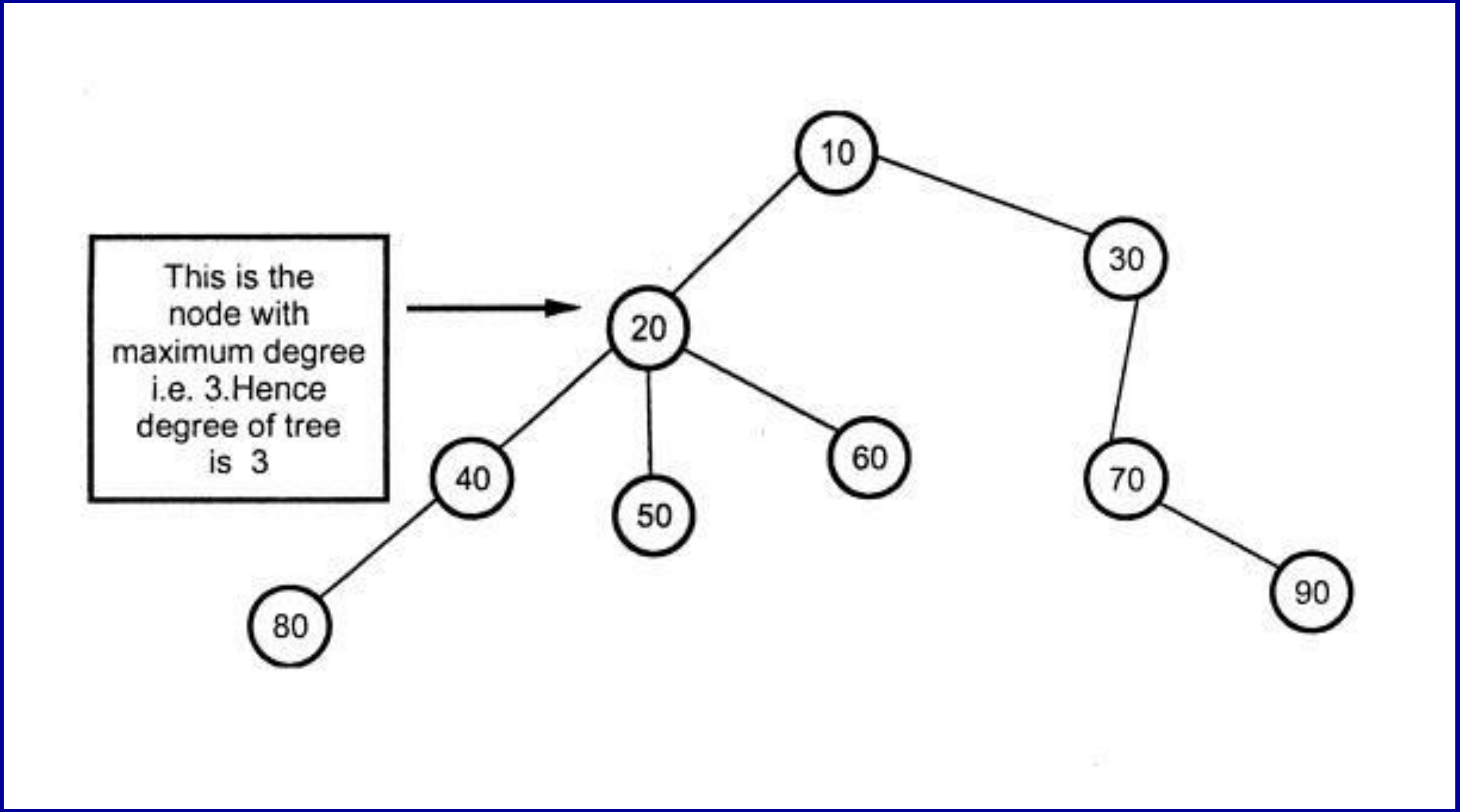
- The *size of a tree* is the number of nodes it contains.
- The total number of subtrees attached to that node is called the *degree of the node*.
- *Degree of the Tree* is nothing but the maximum degree in the tree.
- The nodes with common parent are called *siblings* or *brothers*.

Degree

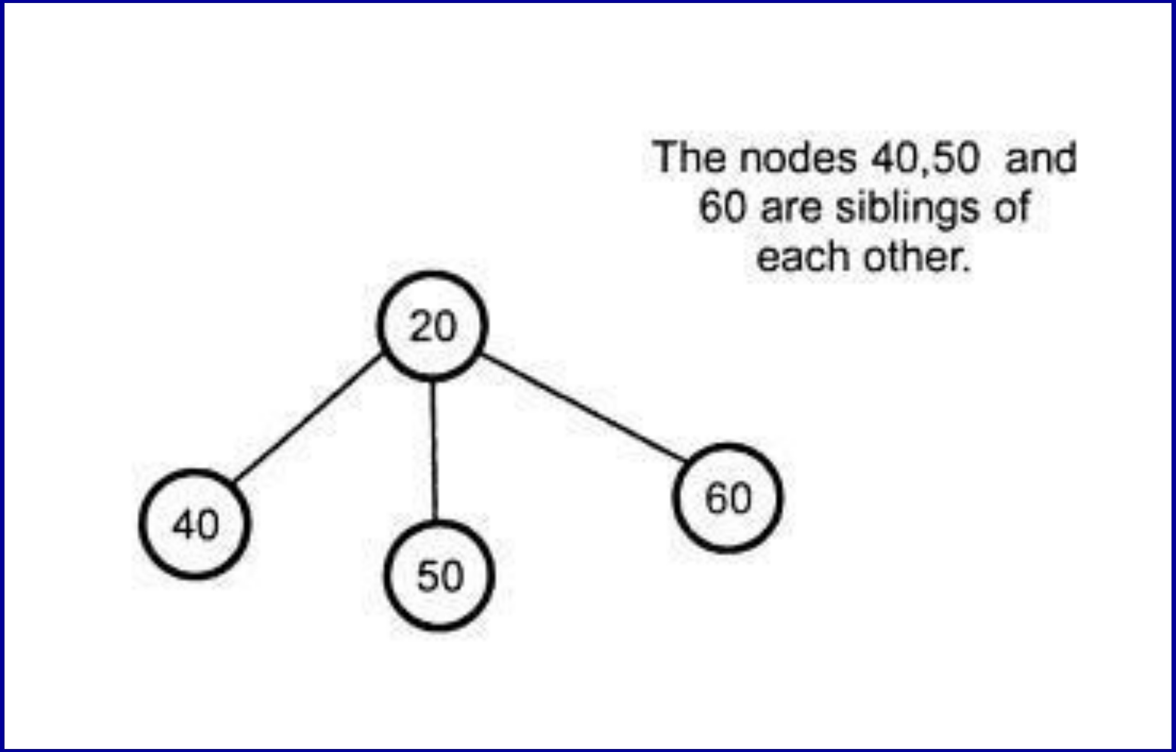


20
40
50
60
degree of this node
is 3
70
90
node with
degree 1
node
with degree 0

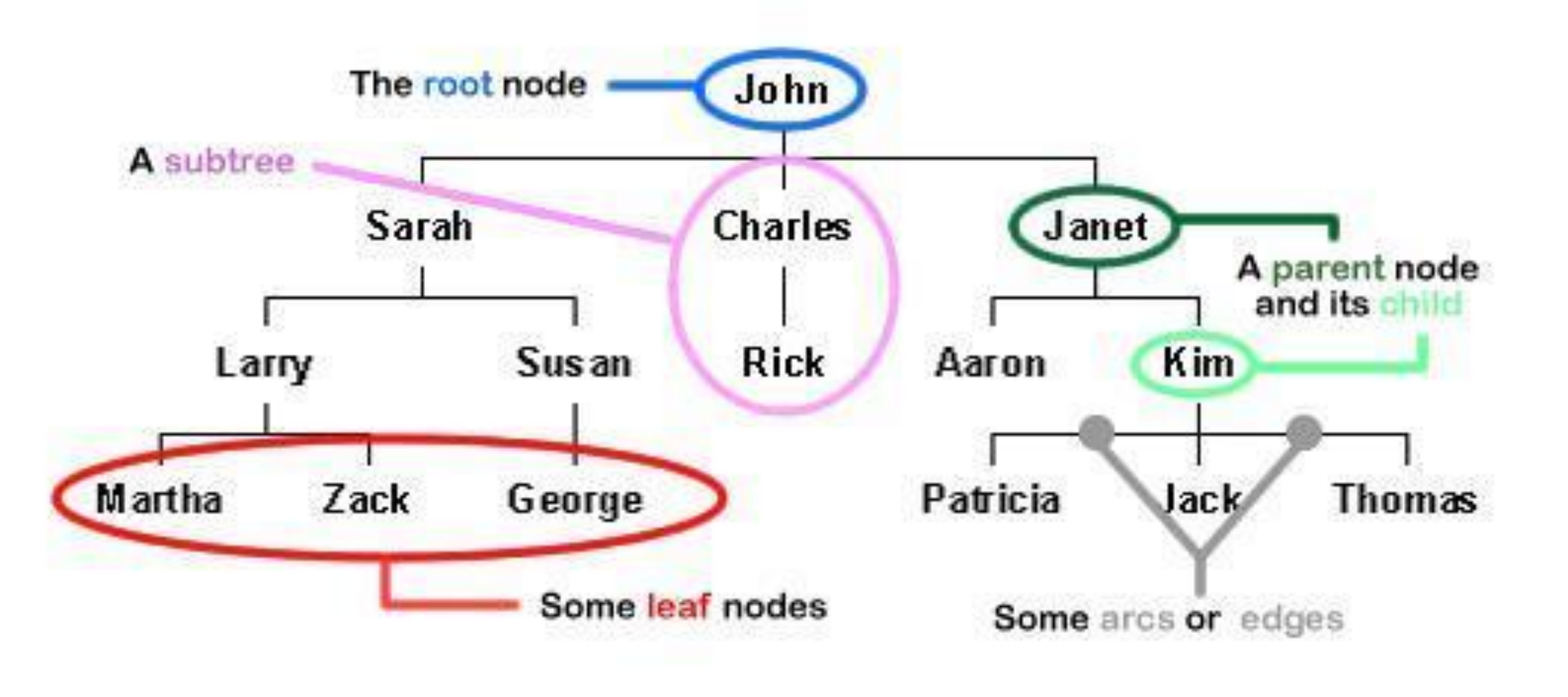
Degree of the Tree



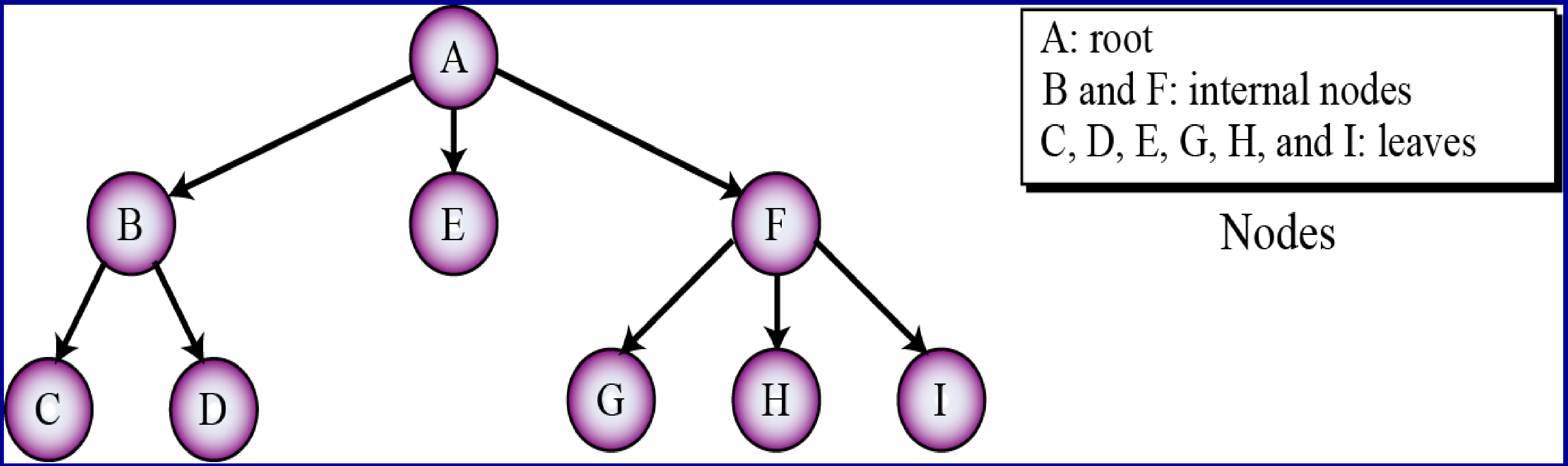
Siblings



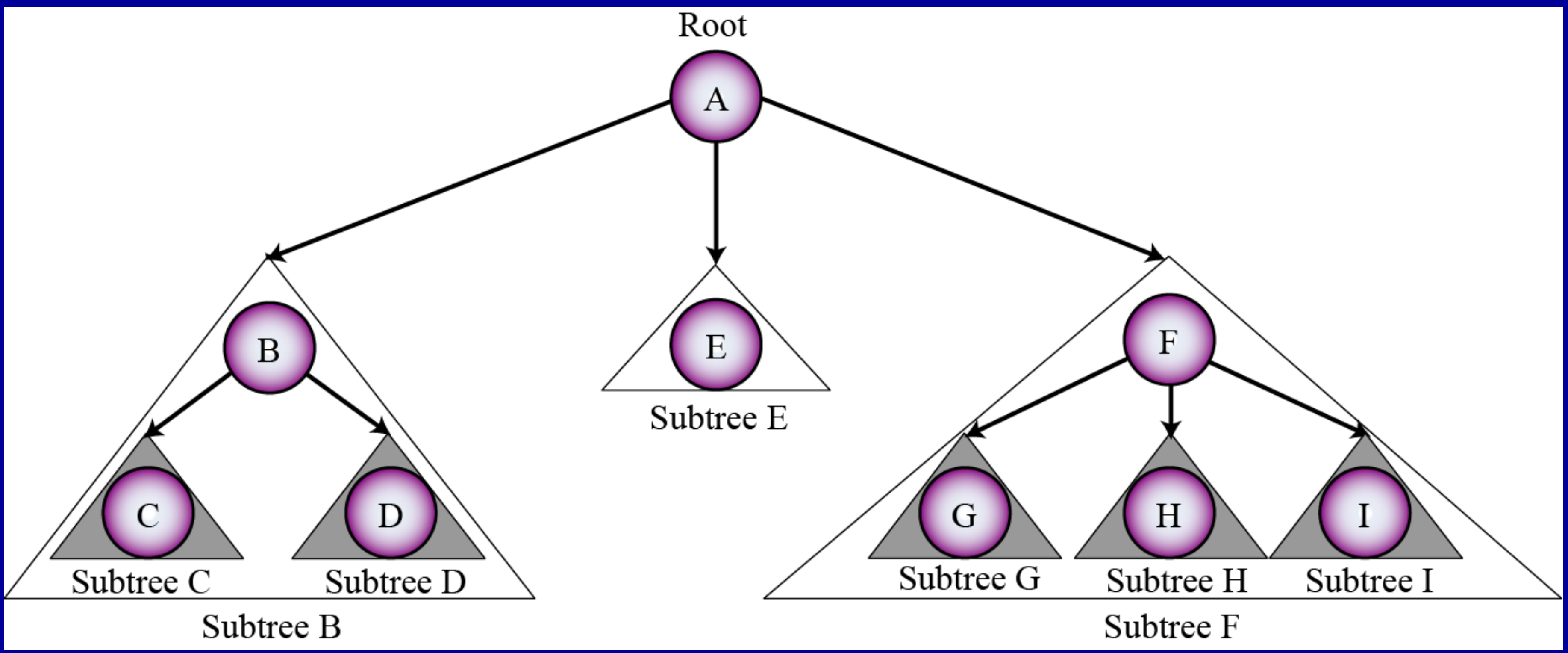
Terminology Explained



Another Example

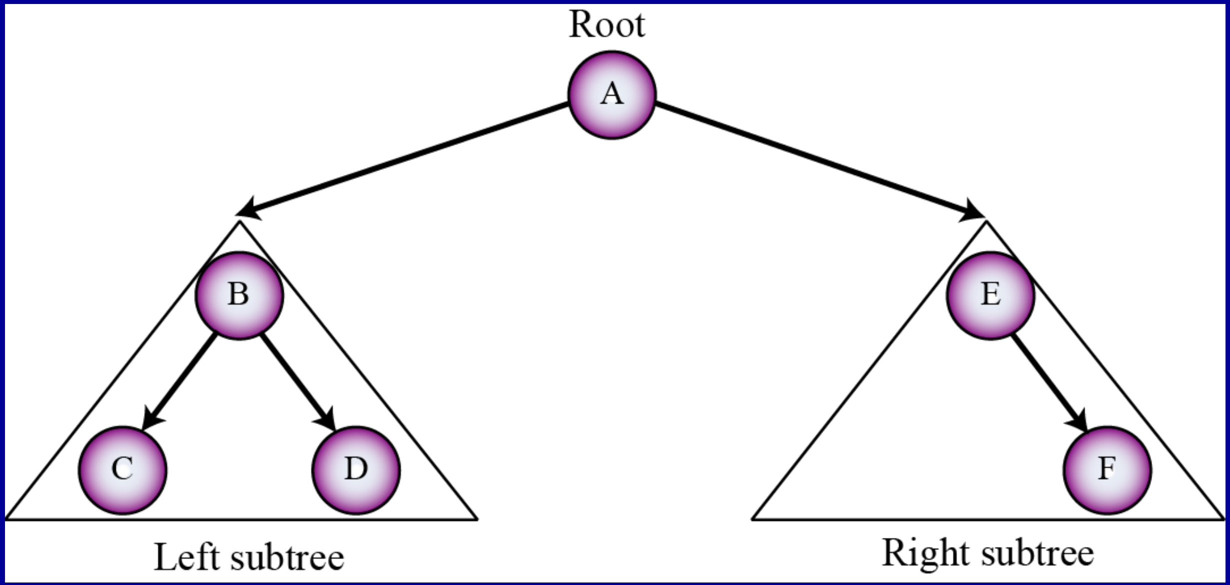


Subtrees



Root
A
B
E
F
C
D
G
H
I
Subtree C
Subtree D
Subtree E
Subtree G
Subtree H
Subtree I
Subtree B
Subtree F

Binary Tree



- A binary tree is a tree in which no node can have more than two subtrees. In other words, a node can have zero, one or two subtrees.

Dr. Ravindra Nath UIET CSJM UNIVERSITY KANPUR UP INDIA

Tree ADT

- The tree ADT stores elements at positions, which are defined relative to neighboring positions.
- Positions in a tree are its nodes, and the neighboring positions satisfy the parent-child relationships that define a valid tree.
- Tree nodes may store arbitrary objects.

Dr. Ravindra Nath UIET CSJM
UNIVERSITY KANPUR UP INDIA

Methods of a Tree ADT

- As with a list position, a position object for a tree supports the method: element() : that returns the object stored at this position (or node).
- The tree ADT supports four types of methods:
  - Accessor Methods
  - Generic Methods
  - Query Methods
  - Update Methods

Accessor Methods

- We use positions to abstract nodes. The real power of node positions in a tree comes from the accessor methods of the tree ADT that return and accept positions, such as the following:
  - root(): Return the position of the tree’s root; an error occurs if the tree is empty.
  - parent(p): Return the position of the parent of p; an error occurs if p is the root.
  - children(p): Return an iterable collection containing the children of node p.
    - Iterable - you can step through (i.e. *iterate*) the object as a collection

Make a note of it

- If a tree T is ordered, then the iterable collection, children(p), stores the children of p in their linear order.
- If p is an external node, then children(p) is empty.
- Any method that takes a position as argument should generate an error condition if that position is invalid.

Generic Methods

- Tree ADT also supports the following Generic Methods:
  - size(): Return the number of nodes in the tree.
  - isEmpty(): Test whether the tree has any nodes or not.
  - Iterator(): Return an iterator of all the elements stored at nodes of the tree.
  - positions(): Return an iterable collection of all the nodes of the tree.

Query Methods

- In addition to the above fundamental accessor methods, the tree ADT also supports the following Boolean query methods:
  - isInternal(p): Test whether node p is an internal node
  - isExternal(p): Test whether node p is an external node
  - isRoot(p): Test whether node p is the root node

  (These methods make programming with tree easier and more readable, since we can use them in the conditionals of if -statements and while -loops, rather than using a non-intuitive conditional).

Update Methods

- The tree ADT also supports the following update method:
  - replace(p, e): Replace with e and return the element stored at node p.

  (Additional update methods may be defined by data structures implementing the tree ADT)

Dr. Ravindra Nath UIET CSJM UNIVERSITY KANPUR UP INDIA

Tree ADT Exceptions

- An interface for the tree ADT uses the following exceptions to indicate error conditions:
  - InvalidPositionException: This error condition may be thrown by any method taking a position as an argument to indicate that the position is invalid.
  - BoundaryViolationException: This error condition may be thrown by method parent() if it's called on the root.
  - EmptyTreeException: This error condition may be thrown by method root() if it's called on an empty tree.

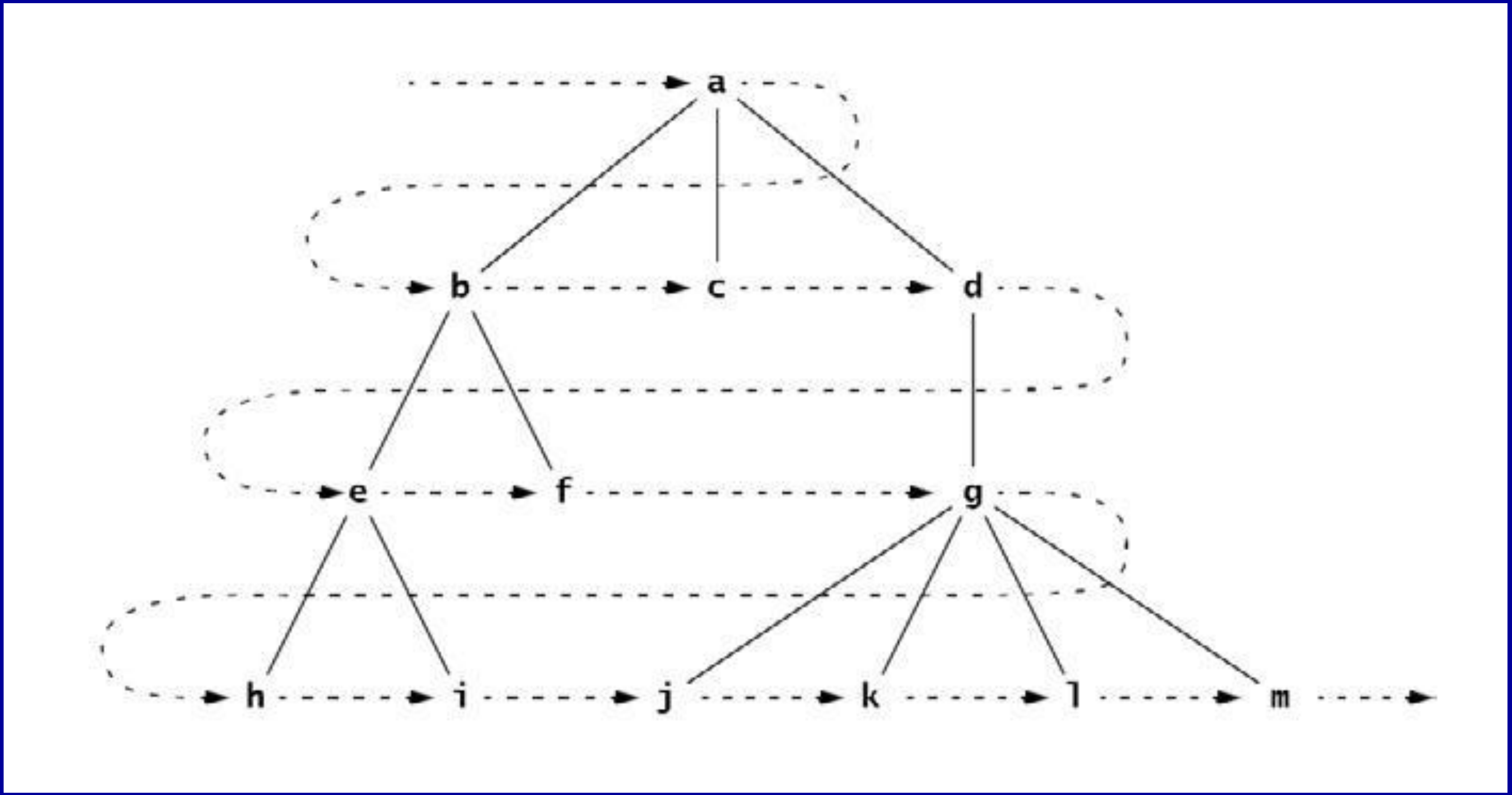
Traversal Algorithms

- A traversal (circumnavigation) algorithm is a method for processing a data structure that applies a given operation to each element of the structure.
- For example, if the operation is to print the contents of the element, then the traversal would print every element in the structure.
- The process of applying the operation to an element is called visiting the element. So executing the traversal algorithm causes each element in the structure to be visited.

Level Order Traversal

- The level order traversal algorithm visits the root, then visits each element on the first level, then visits each element on the second level, and so forth, each time visiting all the elements on one level before going down to the next level.
- If the tree is drawn in the usual manner with its root at the top and leaves near the bottom, then the level order pattern is the same left-to-right top-to-bottom pattern that you follow to read English text.

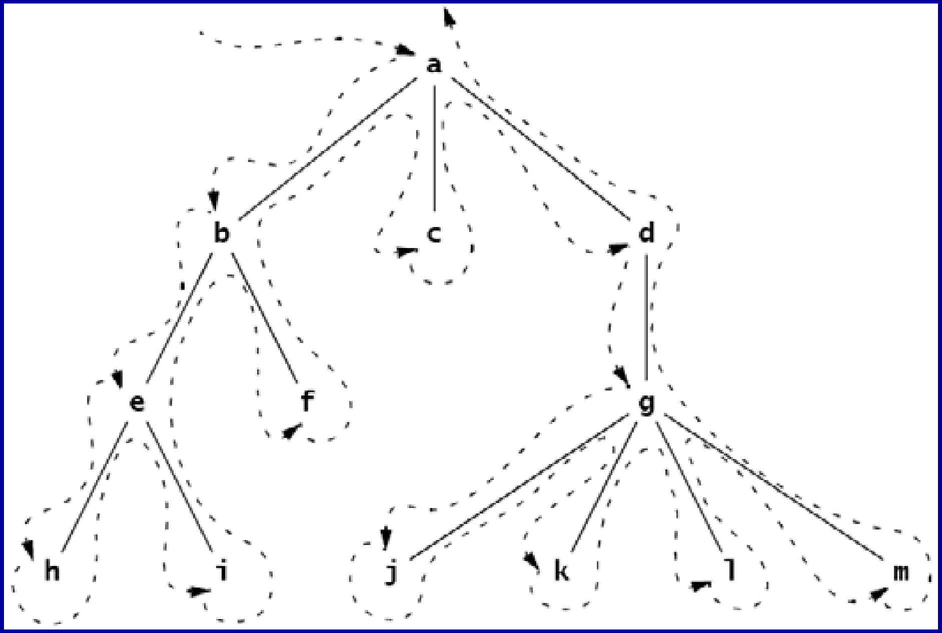
Level Order Example



Level order Traversal - Algorithm

- To traverse a nonempty ordered tree:
  1. Initialize a queue.
  2. Enqueue the root.
  3. Repeat steps 4–6 until the queue is empty.
     4. Dequeue node x from the queue.
     5. Visit x.
     6. Enqueue all the children of x in order.

Preorder Traversal



The preorder traversal of the tree shown above would visit the nodes in this order: a, b, e, h, i, f, c, d, g, j, k, l, m. Note that the preorder traversal of a tree can be obtained by circumnavigating the tree, beginning at the root and visiting each node the first time it is encountered on the left.

Dr. Ravindra Nath UIET CSJM UNIVERSITY KANPUR UP INDIA

Preorder Traversal - Algorithm

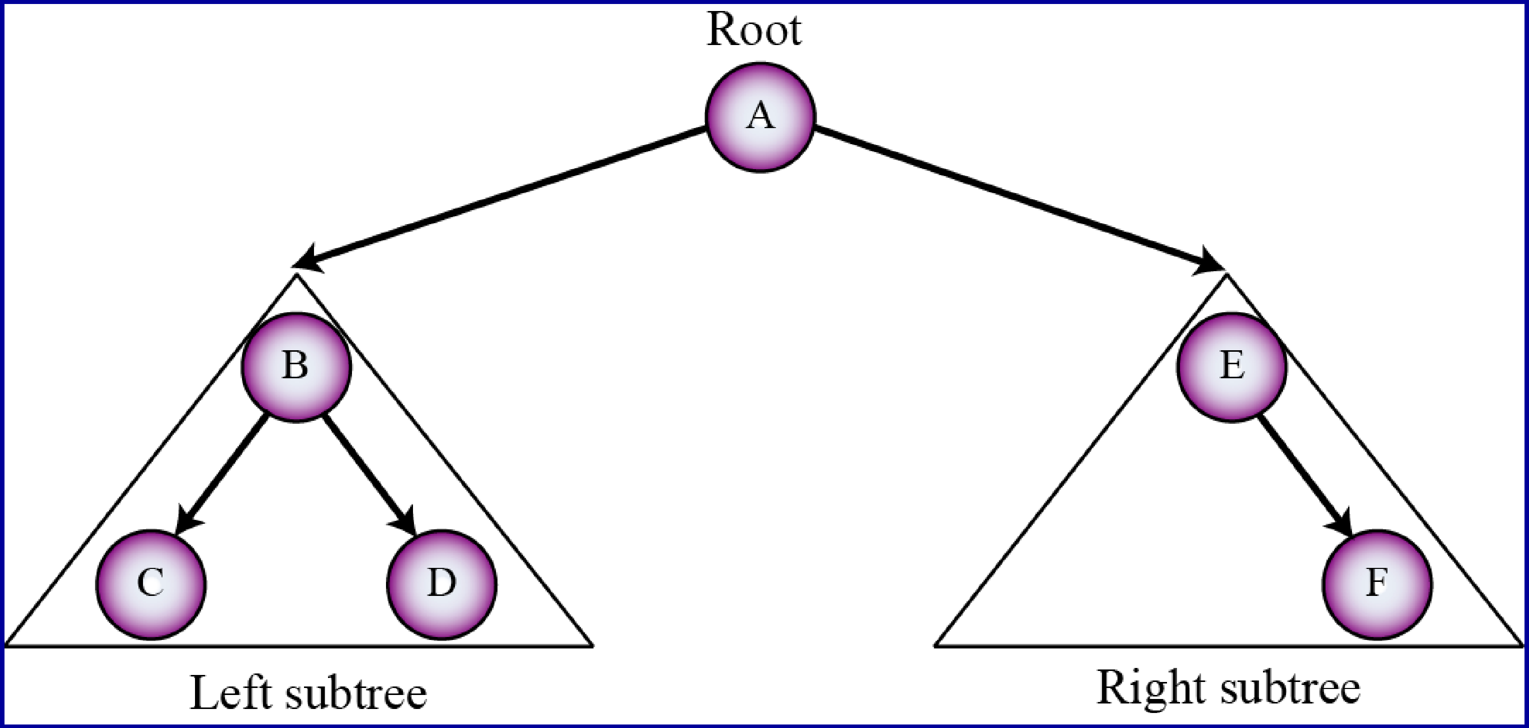
- To traverse a nonempty ordered tree:
  1. Visit the root.
  2. Do a recursive preorder traversal of each subtree in order.
- The postorder traversal algorithm does a postorder traversal recursively to each subtree before visiting the root.

Dr. Ravindra Nath UIET CSJM UNIVERSITY KANPUR UP INDIA

Binary Trees

- A binary tree is either the empty set or a triple T = (x, L, R), where x is a node and L and R are disjoint binary trees, neither of which contains x.
- The node x is called the root of the tree T, and the subtrees L and R are called the left subtree and the right subtree of T rooted at x.

Binary Tree



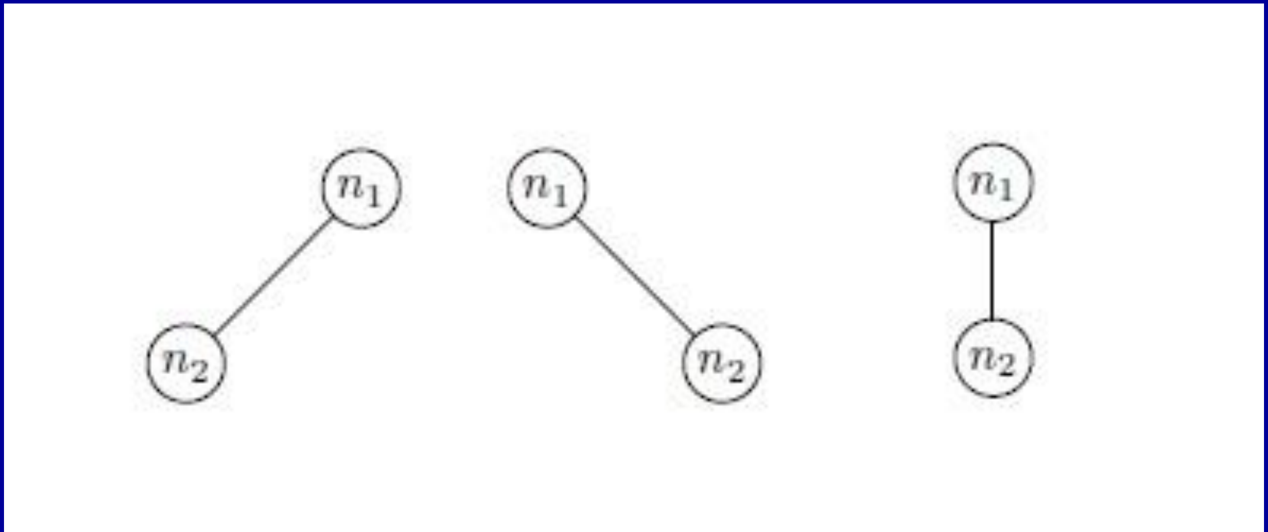
Terminologies

- The definitions of the terms *size*, *path*, *length* of a path, *depth* of a node, *level*, *height*, *interior* node, *ancestor*, *descendant*, and *subtree* are the same for binary trees as for general trees.

Trees and Binary Trees - Difference

- It is important to understand that while binary trees require us to distinguish whether a child is either a left child or a right child, ordinary trees require no such distinction.
- There is another technical difference. While trees are defined to have at least one node, it is convenient to include the empty tree, the tree Empty tree with no nodes, among the binary trees.

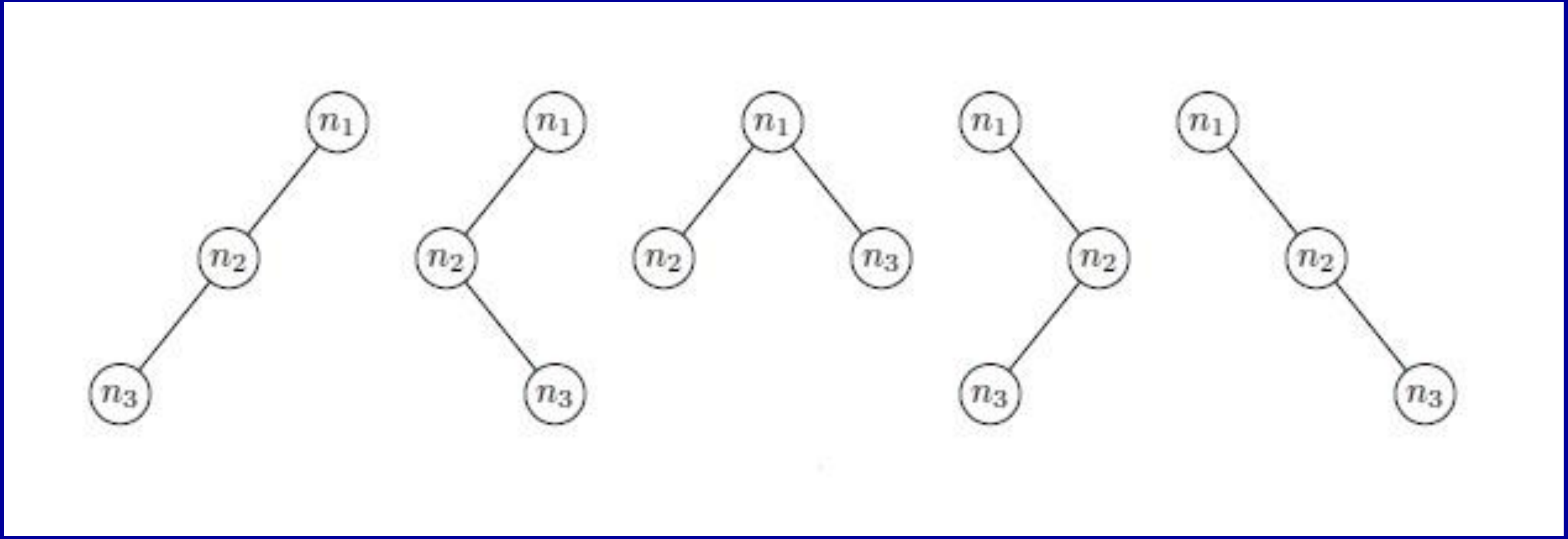
Difference Continued



- That is, binary trees are not just trees all of whose nodes have two or fewer children.
- Not only are the two trees in the above Figure are different from each other, but they have no relation to the ordinary tree consisting of a root and a single child of the root:

Dr. Ravindra Nath UIET CSJM UNIVERSITY KANPUR UP INDIA

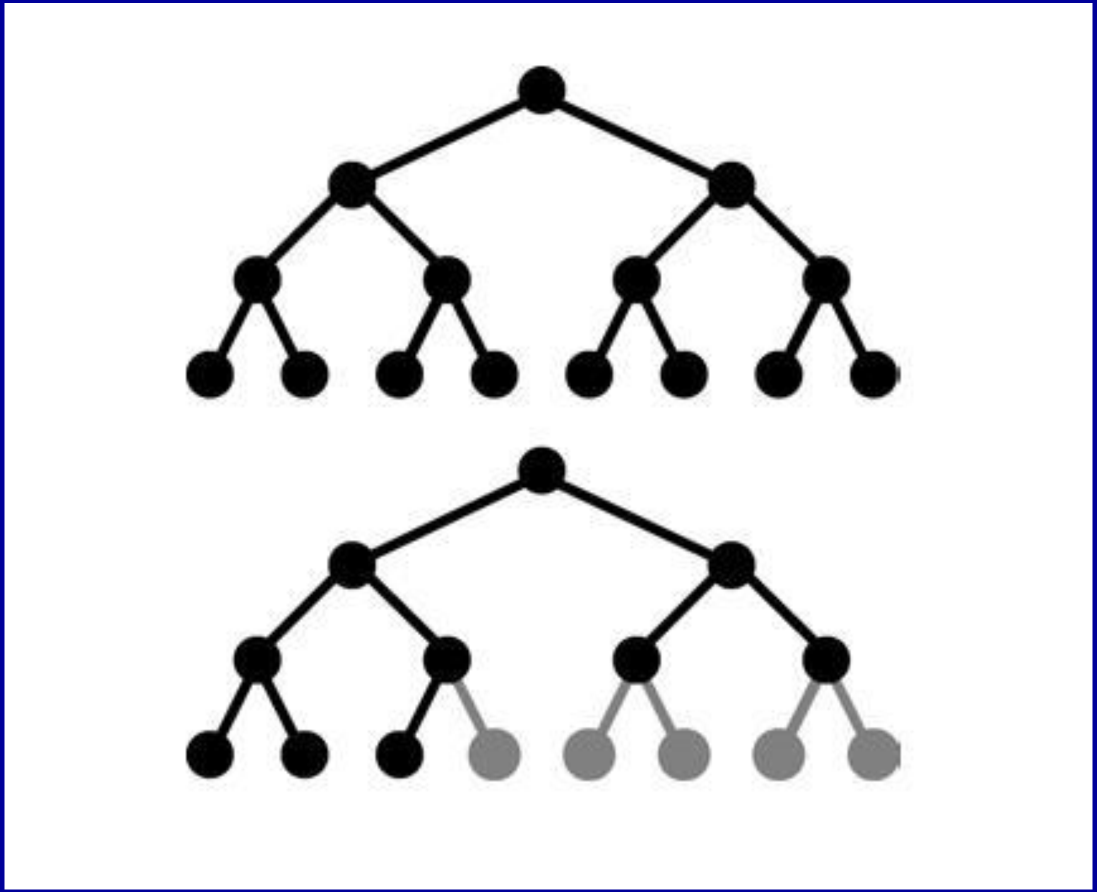
Five binary trees with three nodes



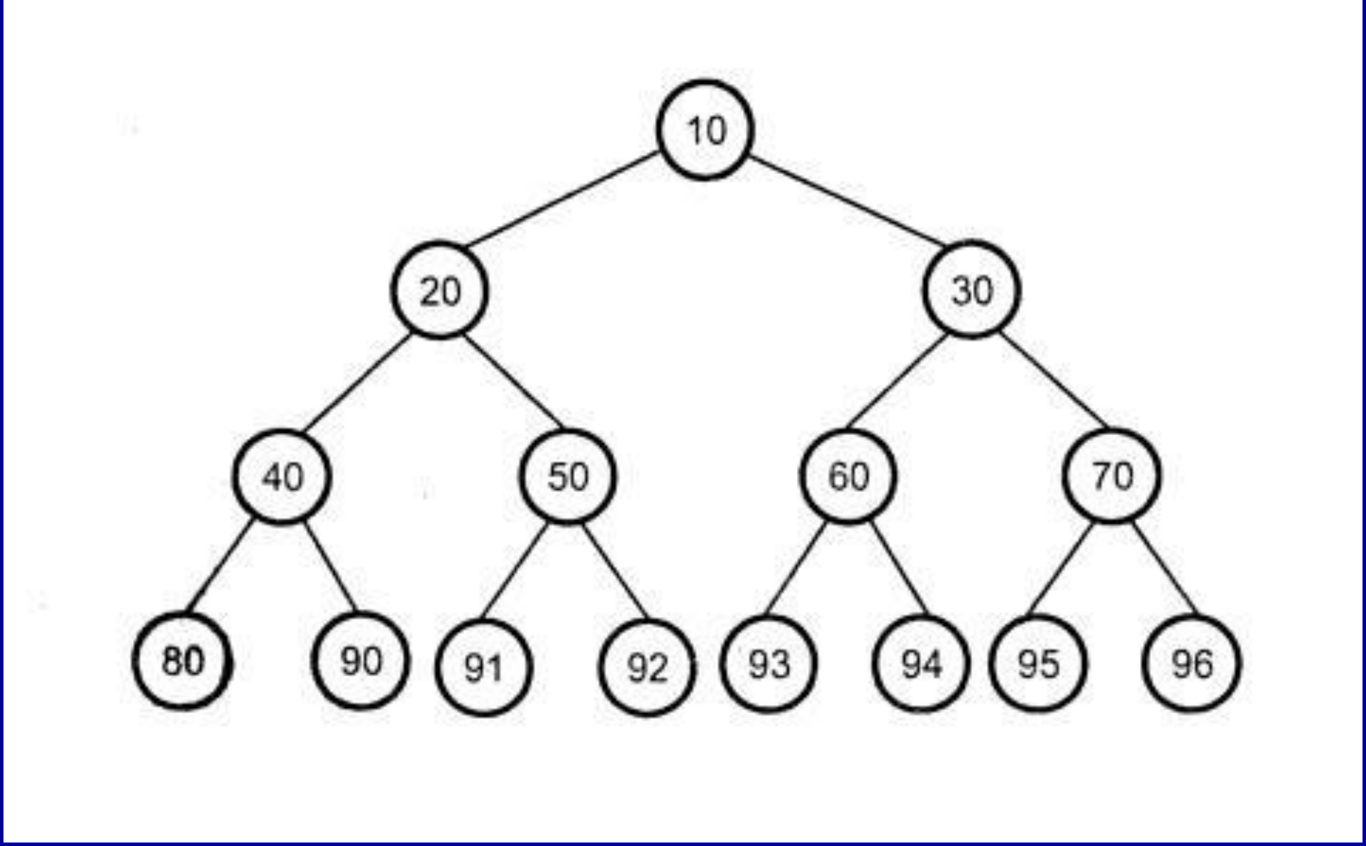
Full and Complete Binary Trees

- A binary tree is said to be full if all its leaves are at the same level and every interior node has two children.
- A complete binary tree is either a full binary tree or one that is full except for a segment of missing leaves on the right side of the bottom level.

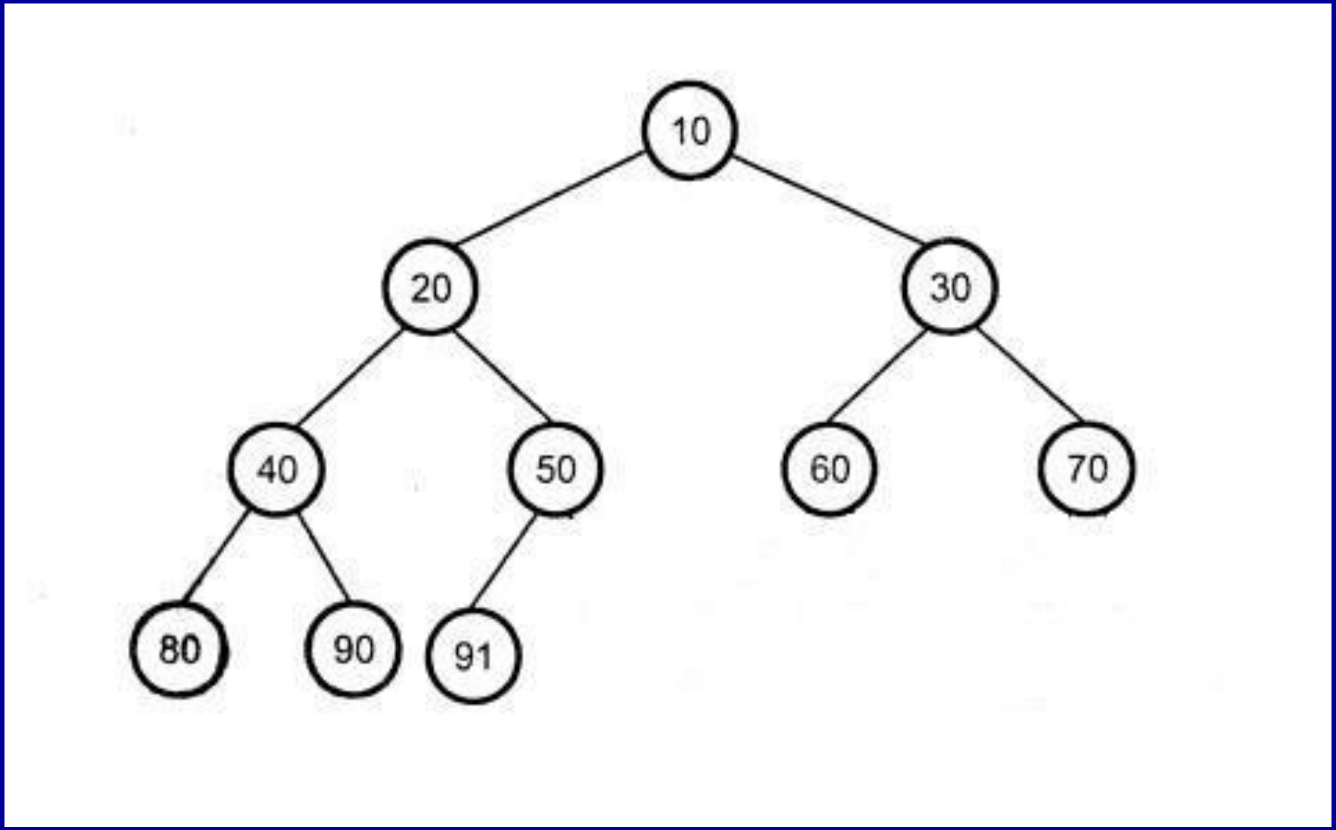
Full and Complete Binary Trees



Full Binary Tree



Complete Binary Tree



Full Binary Trees

- The full binary tree of height h has $l = 2^h$ leaves and $m = 2^h - 1$ internal nodes.
- The full binary tree of height h has a total of $n = 2^{h+1} - 1$ nodes.
- The full binary tree with n nodes has height $h = lg(n+1) - 1$.

Complete Binary Tree

- In a complete binary tree of height h,

  $h + 1 \leq n \leq 2^{h+1} - 1$ and $h = \lfloor \lg n \rfloor$

Dr. Ravindra Nath UIET CSJM
UNIVERSITY KANPUR UP INDIA

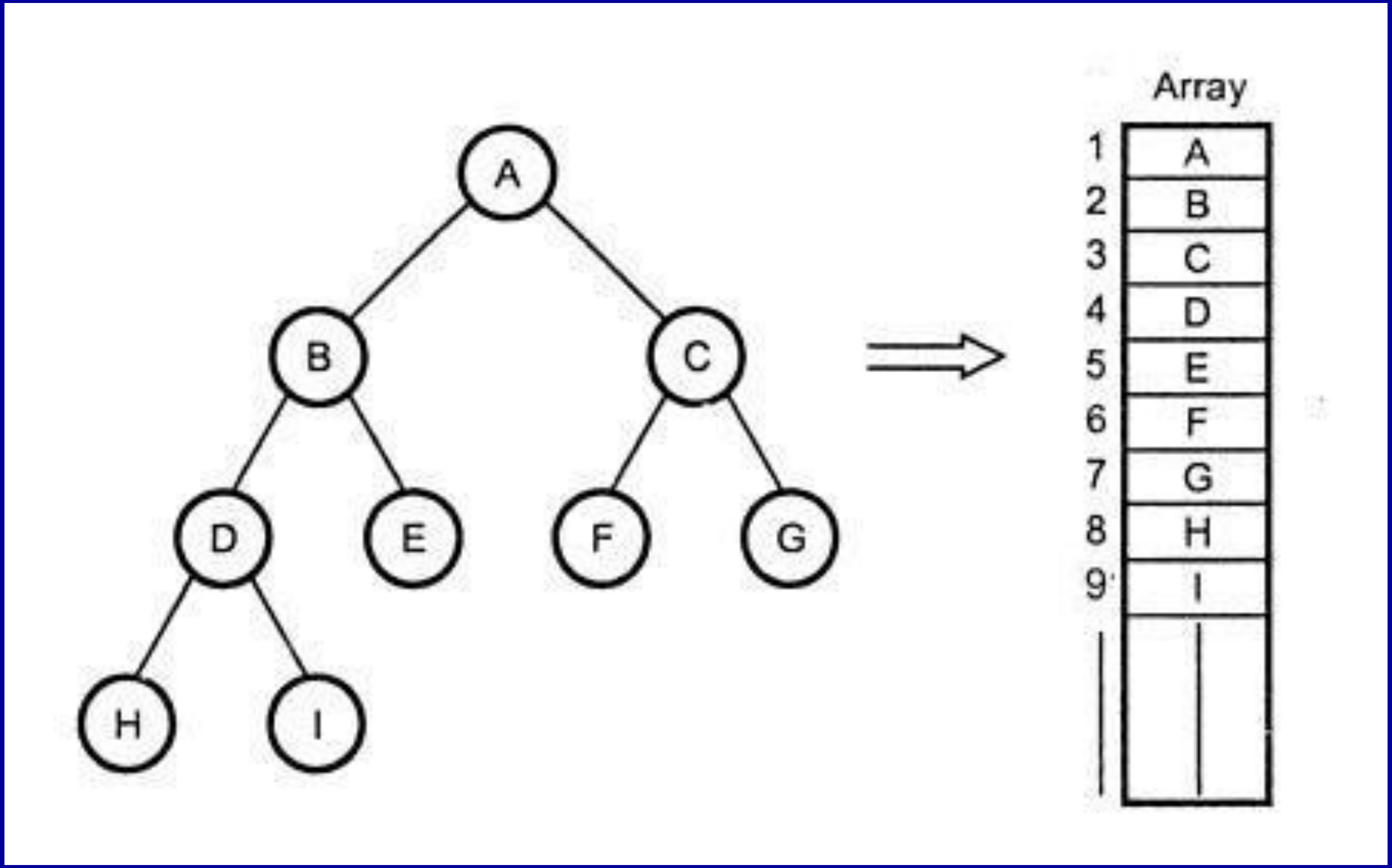
Make a Note

- Complete binary trees are important because they have a simple and natural implementation using ordinary arrays.
- The *natural mapping* is actually defined for any binary tree: Assign the number 1 to the root; for any node, if i is its number, then assign $2i$ to its left child and $2i+1$ to its right child (if they exist).
- This assigns a unique positive integer to each node. Then simply store the element at node i in a[i], where a[] is an array.

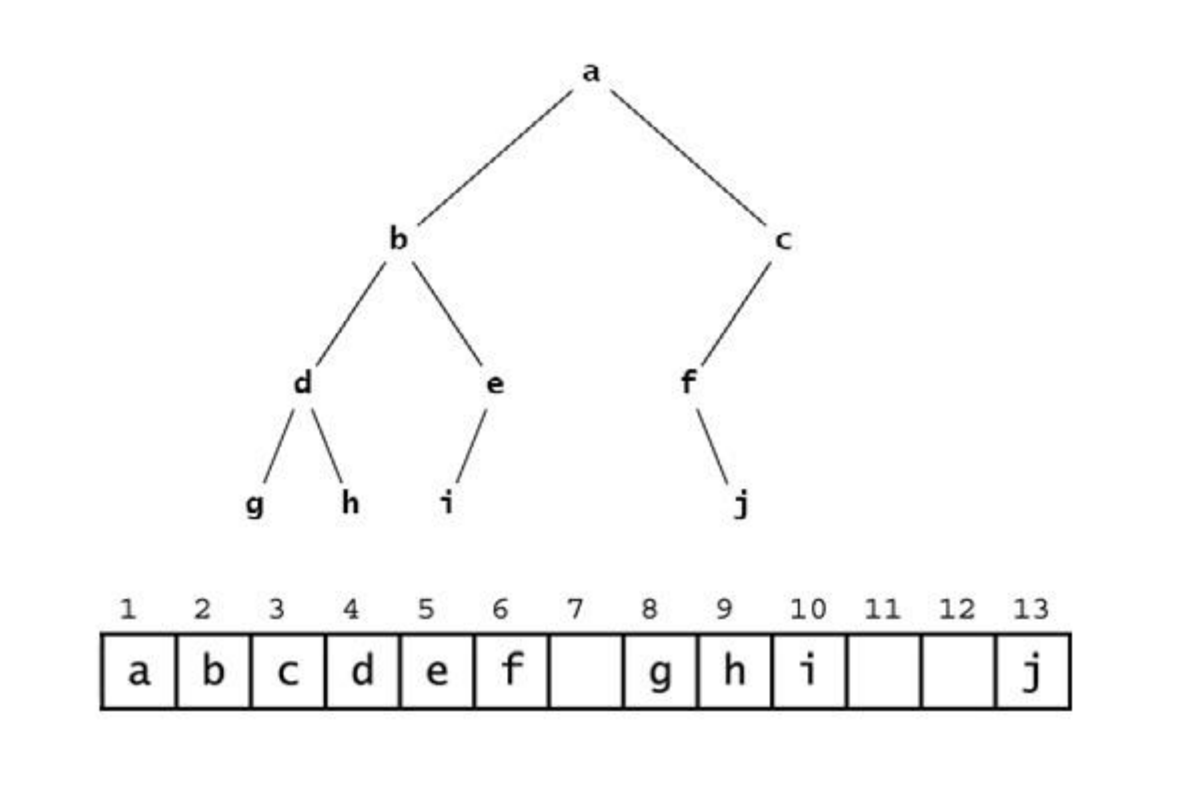
Binary Tree Representations

- If a complete binary tree with n nodes (depth = log n + 1) is represented sequentially, then for any node with index i, $1 \leq i \leq n$, we have:
  - *parent*(i) is at $i/2$ if i!=1. If i=1, i is at the root and has no parent.
  - *leftChild*(i) is at $2i$ if $2i \leq n$. If $2i > n$, then i has noleft child.
  - *rightChild*(i) is at $2i+1$ if $2i + 1 \leq n$. If $2i + 1 > n$, then i has no right child.

Array Implementation



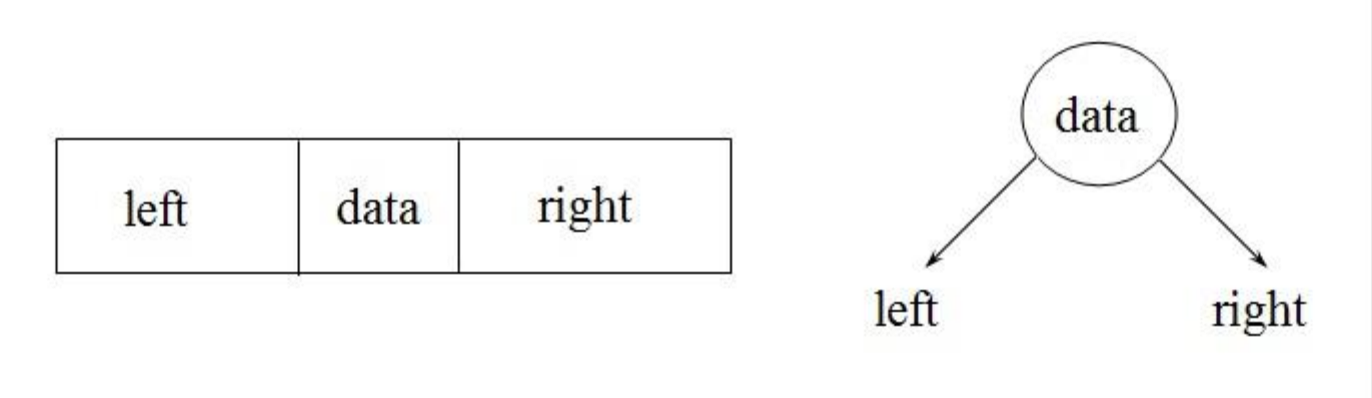
The Disadvantage



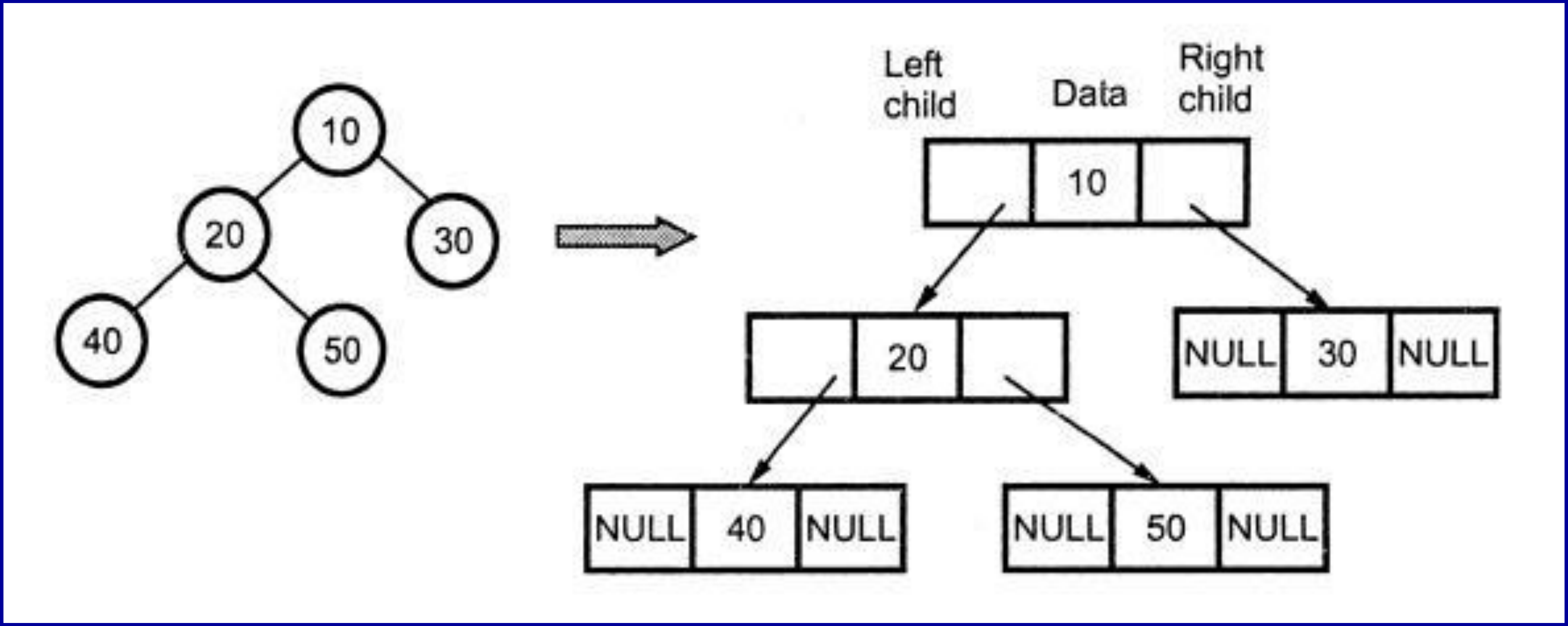
- Figure above shows the incomplete binary tree and the natural mapping of its nodes into an array which leaves some gaps.

Linked List Implementation

```
typedef struct tnode *ptnode;
typedef struct tnode
{
  int data;
  ptnode left, right;
};
```

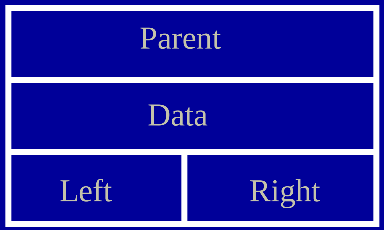


Linked List Implementation



Include One more Pointer

- A natural way to implement a *tree T* is to use a *linked structure*, where we represent each *node p* of *T* by a *position* object with the following fields:
  - A *link* to the *parent* of ***p,*** A *link* to the *LeftChild* named *Left*, a *link* to the *RightChild* named *Right* and the *Data.*



Array Implementation

- Advantages
  - Direct Access
  - Finding the Parent / Children is fast
- Disadvantages
  - Wastage of memory
  - Insertion and Deletion will be costlier
  - Array size and depth

Linked List Implementation

- Advantages
  - No wastage of memory
  - Insertion and Deletion will be easy
- Disadvantages
  - Does not provide direct access
  - Additional space in each node.

Binary Tree ADT

- The Binary Tree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT, in addition to that it supports the following additional accessor methods:
  - position left(p): return the left child of p, an error condition occurs if p has no left child.
  - position right(p): return the right child of p, an error condition occurs if p has no right child.
  - boolean hasLeft(p): test whether p has a left child
  - boolean hasRight(p): test whether p has a right child

Binary Tree ADT (Cont.)

- Update methods may be defined by data structures implementing the Binary Tree ADT.
- Since Binary trees are ordered trees, the iterable collection returned by method chilrden(p) (inherited from the Tree ADT), stores the left child of p before the right child of p.

Dr. Ravindra Nath UIET CSJM
UNIVERSITY KANPUR UP INDIA

Binary Tree Traversals

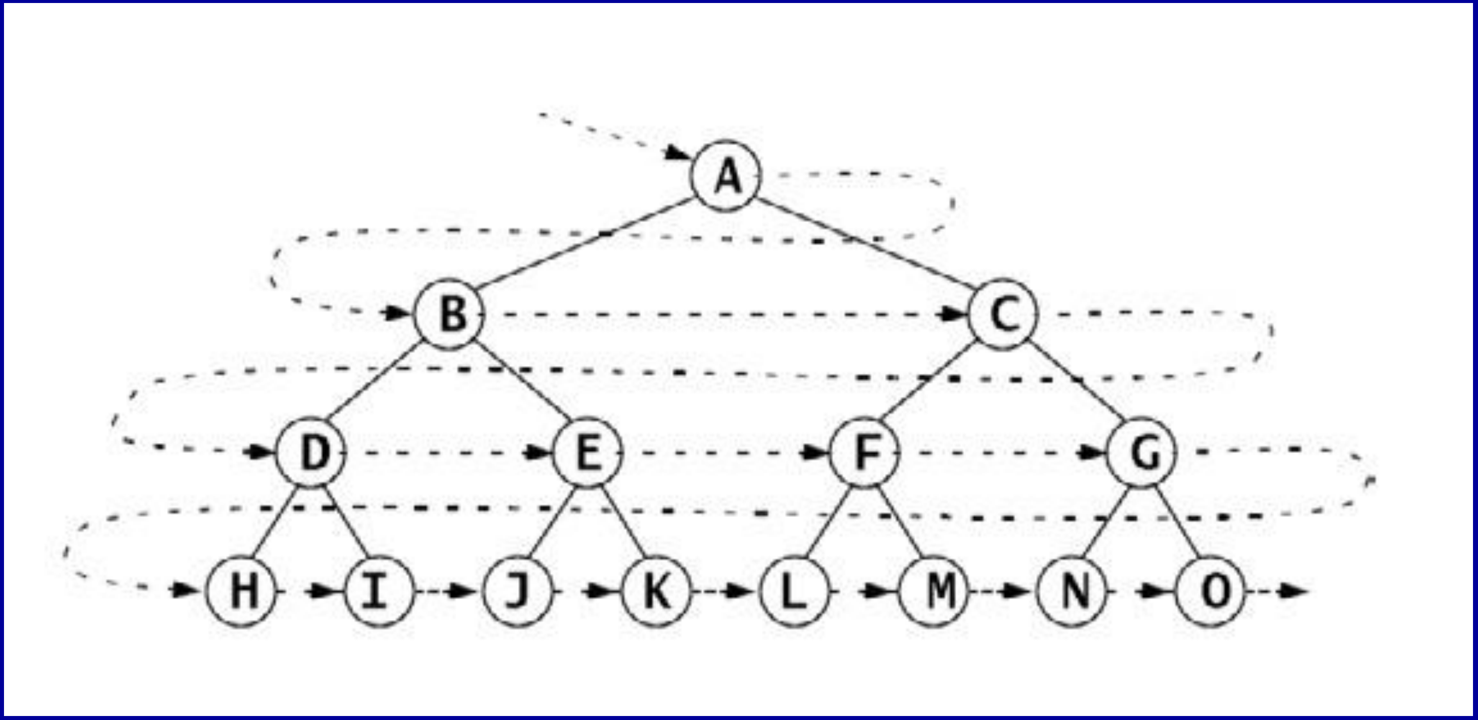
- The three traversal algorithms that are used for general trees (see Chapter 10) apply to binary trees as well: the preorder traversal, the postorder traversal, and the level order traversal.
- In addition, binary trees support a fourth traversal algorithm: the inorder traversal. These four traversal algorithms are given next.

Level Order Traversal

- To traverse a nonempty binary tree:

  1. Initialize a queue.
  2. Enqueue the root.
  3. Repeat steps 4–7 until the queue is empty.
     4. Dequeue a node x from the queue.
     5. Visit x.
     6. Enqueue the left child of x if it exists.
     7. Enqueue the right child of x if it exists.

Level Order



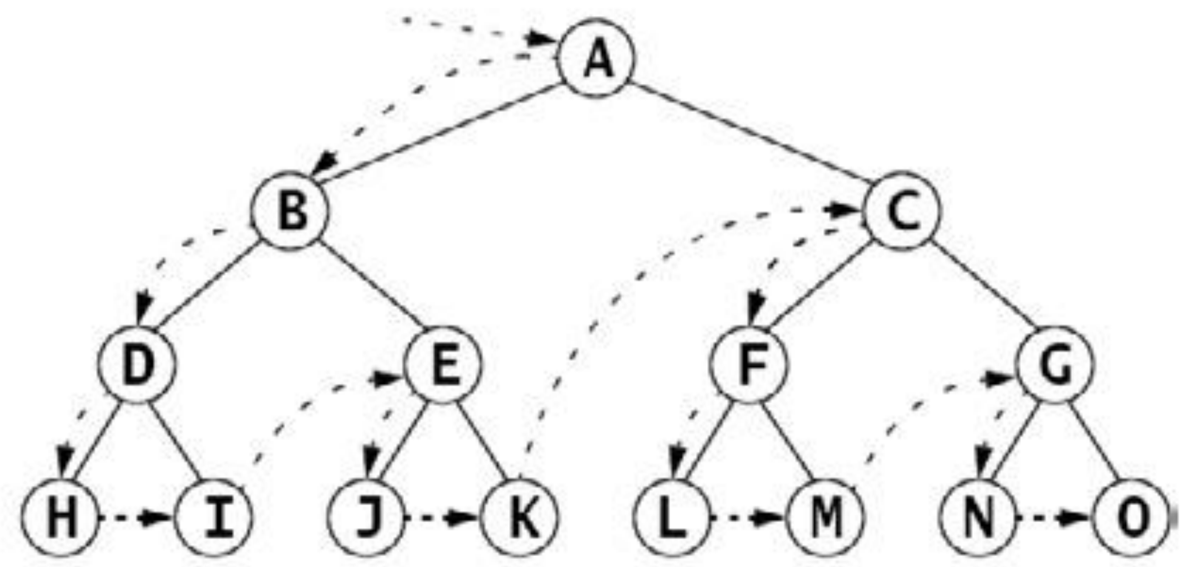
Preorder Traversal

- To traverse a nonempty binary tree:

  1. Visit the root.
  2. If the left subtree is nonempty, do a preorder traversal on it.
  3. If the right subtree is nonempty, do a preorder traversal on it.

Dr. Ravindra Nath UIET CSJM UNIVERSITY KANPUR UP INDIA

Preorder

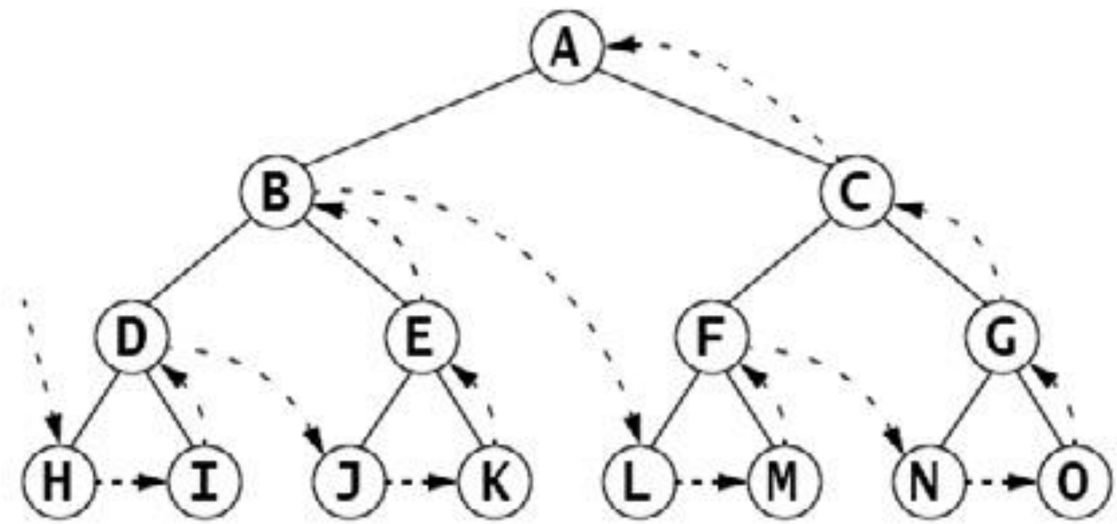


Postorder Traversal

- To traverse a nonempty binary tree:

1. If the left subtree is nonempty, do a postorder traversal on it.
2. If the right subtree is nonempty, do a postorder traversal on it.
3. Visit the root.

Postorder

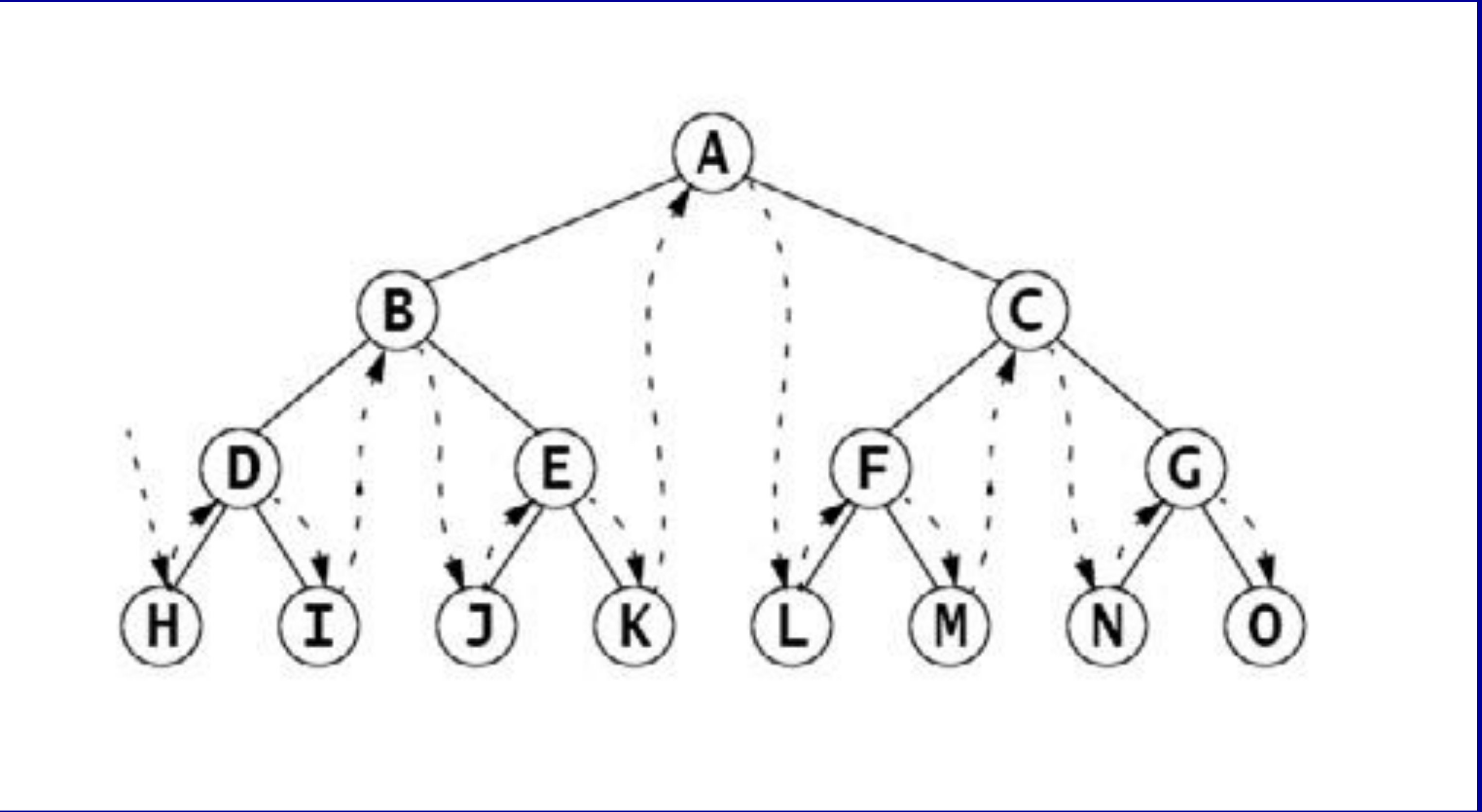


Inorder Traversal

- To traverse a nonempty binary tree:

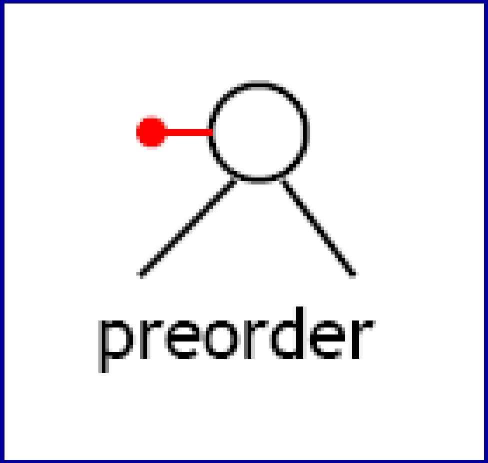
1. If the left subtree is nonempty, do a preorder traversal on it.
2. Visit the root.
3. If the right subtree is nonempty, do a preorder traversal on it.

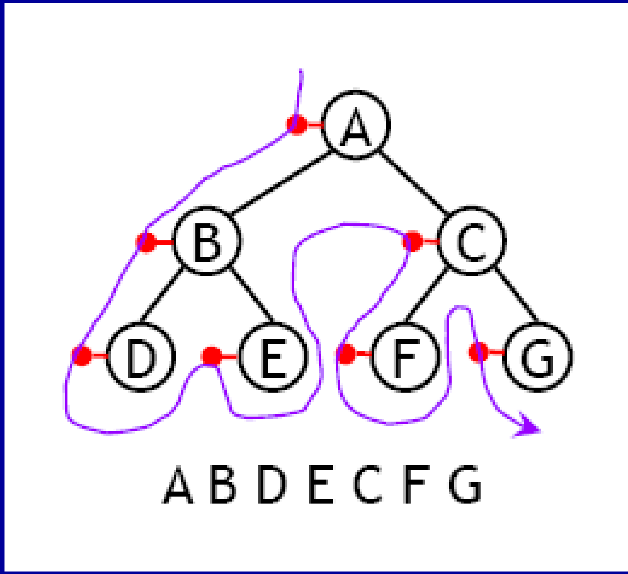
Inorder



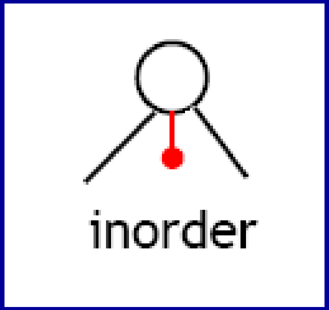
Traversal Using Flag

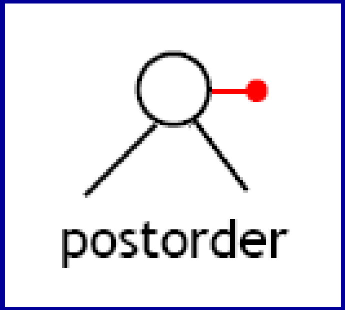
- The order in which the nodes are visited during a tree traversal can be easily determined by imagining there is a "flag" attached to each node, as follows:
- To traverse the tree, collect the flags:

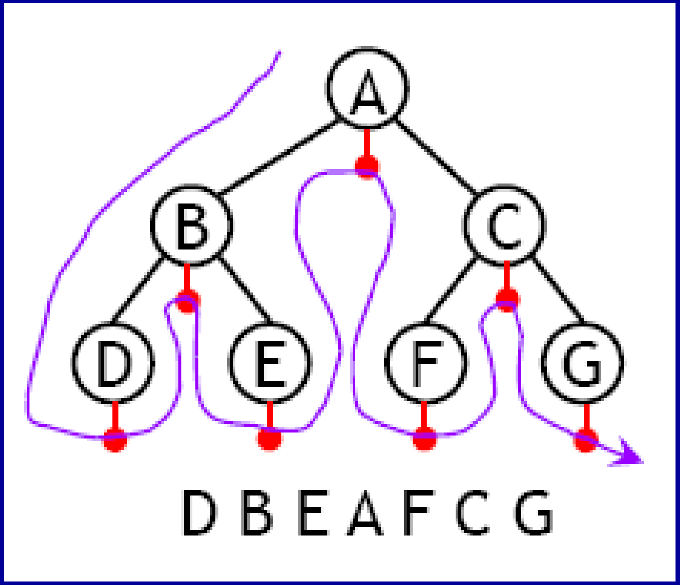


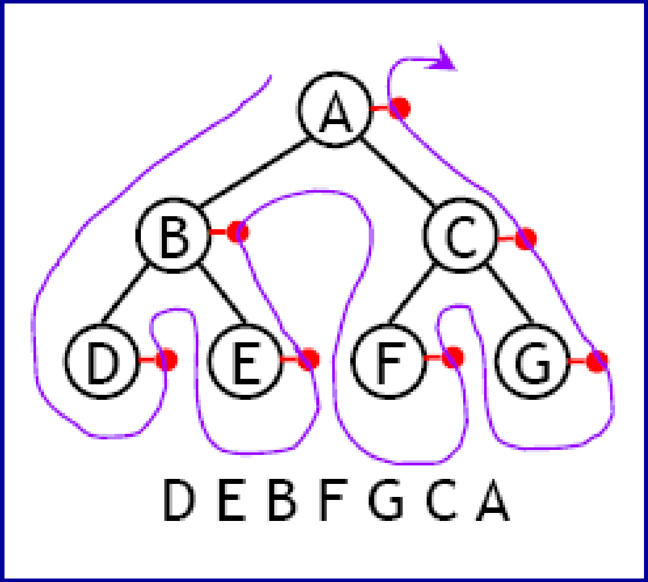


Inorder and Postorder









Dr. Ravindra Nath UIET CSJM
UNIVERSITY KANPUR UP INDIA

Thanks
and BEST of LUCK