DATA STRUCTURES AND ALGORITHM

Module 7

- To understand what a tree data structure is and how the importance to use it.
- To understand the different tree terminologies
- To construct trees according to its algorithm

TREES

Trees

- A non-linear data structure
- It consists of nodes connected by edges
- Tree imposes a **Hierarchical** structure, on a collection of items. It could be above or below on the objects

TREES

Trees

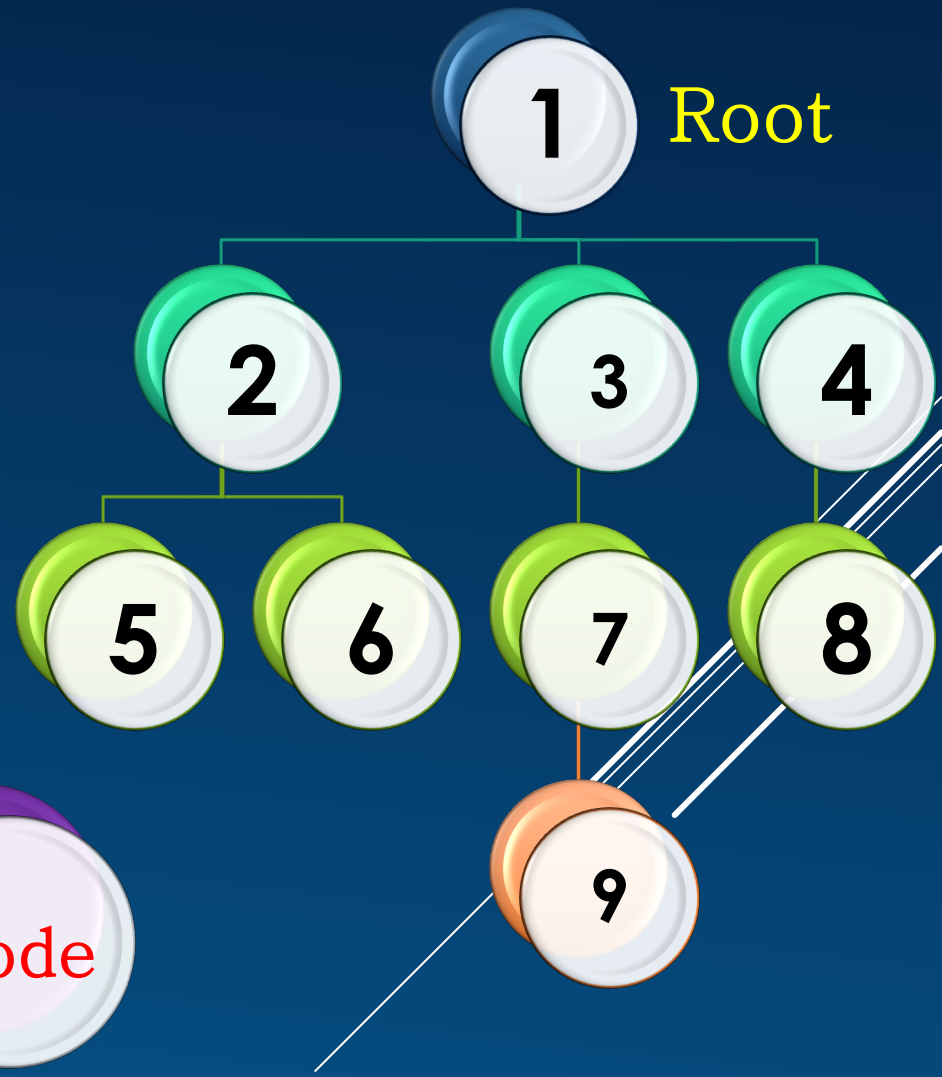
A tree can be theoretically defined as a finite set of one or more data items (or nodes[N]) such that :

1. If **Tree** is nonempty, it has a special node, called the **root** of *Tree*, that has no parent.
2. Each node **v** of *Tree* different from the root has a unique **parent** node ***w***; every node with parent ***w*** is a ***child*** of w.

TREES

BASIC TERMINOLOGIES

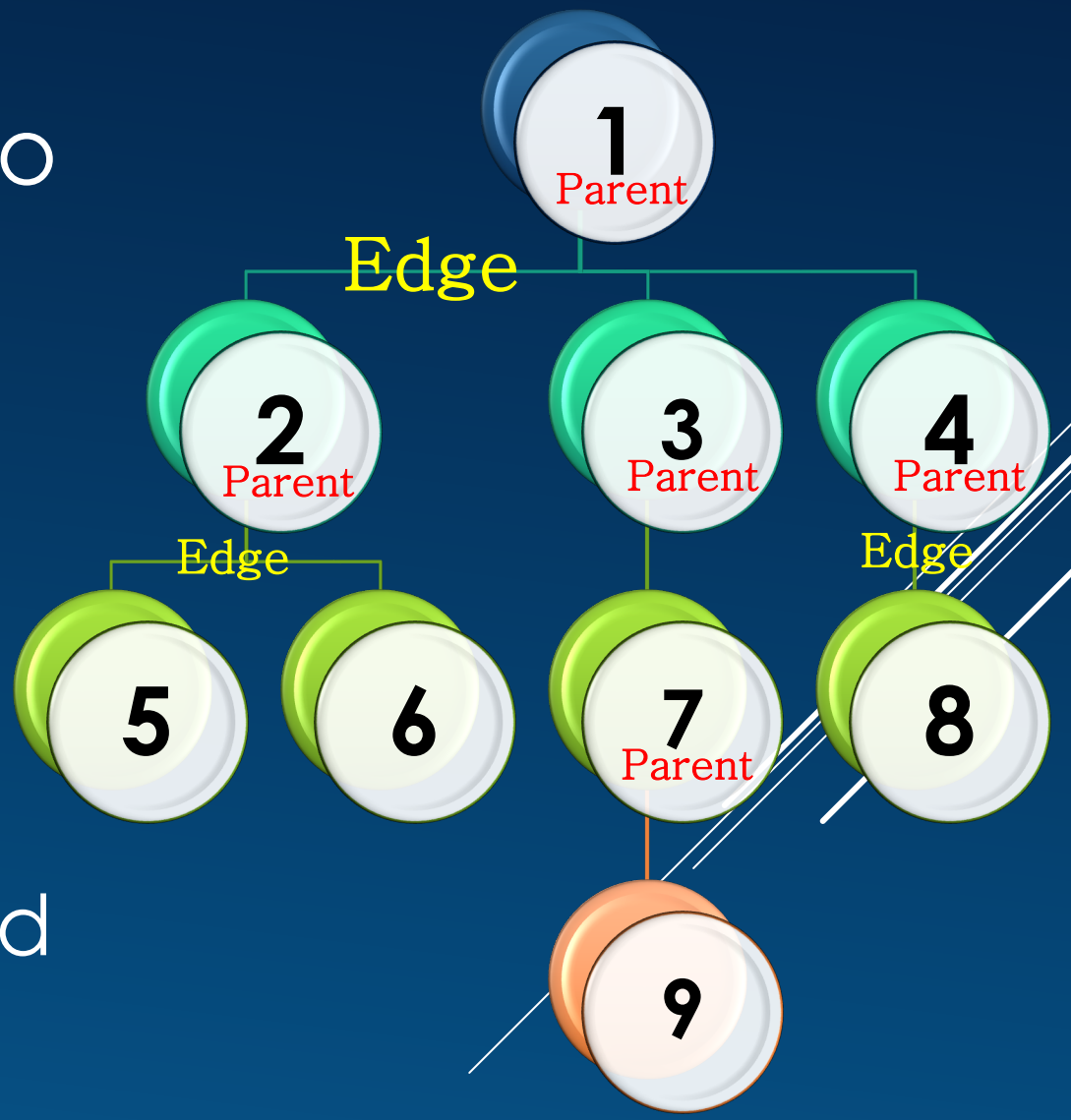
- **ROOT** is a specially designed node (or data items) in a tree. It is the first node in the hierarchical arrangement of the data items.
- **Node** of a tree stores the data and its role is the same as in the linked list. Nodes are connected by the means of links with other nodes.



TREES

BASIC TERMINOLOGIES

- **Edge** connects two nodes to show that there is a relationship between them.
- **Parent** of the nodes it connects to with outgoing edges or having a child



Examples:

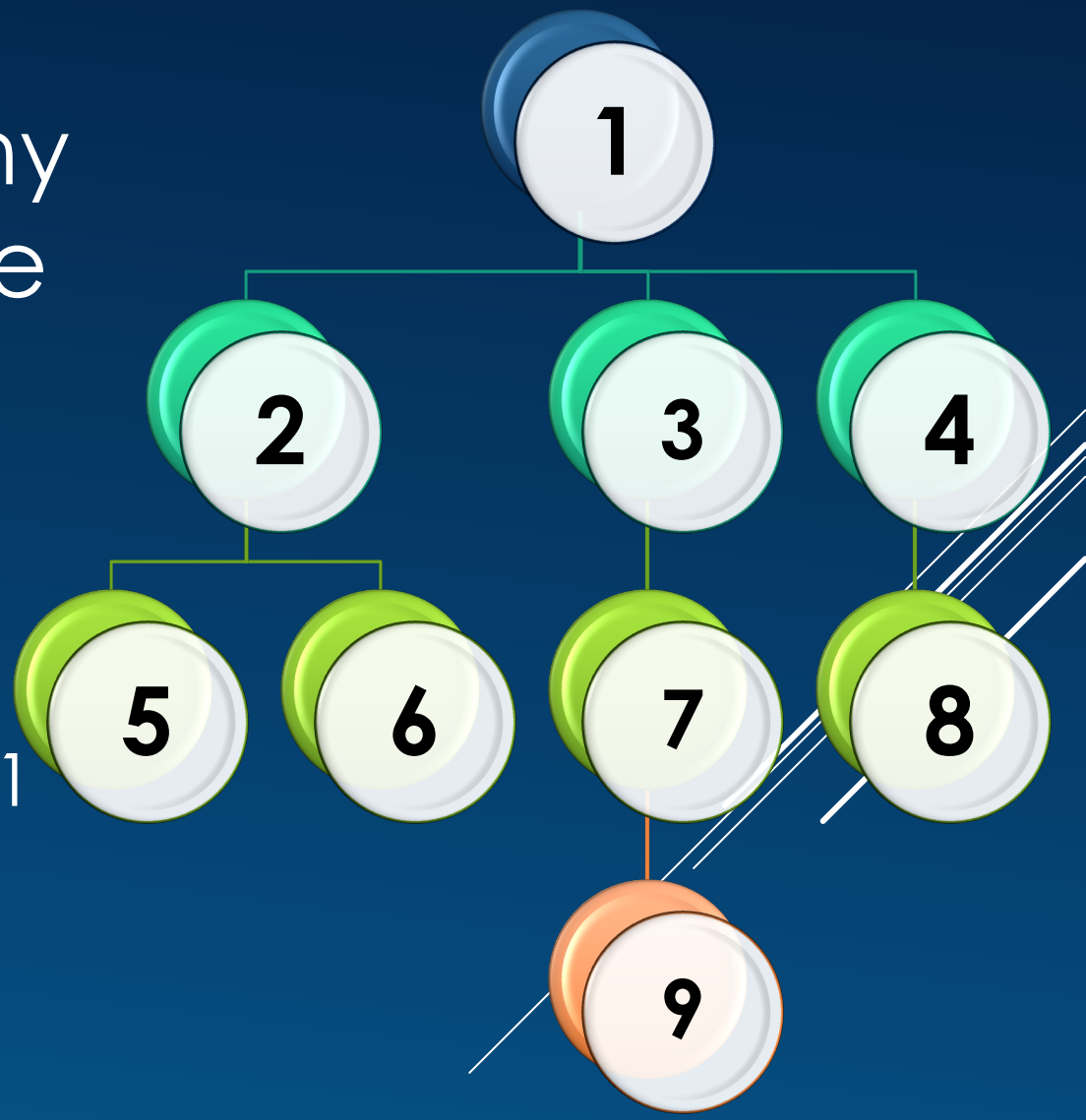
Node 2 is the parent of node 5 and node 6

Node 7 is the parent of node 9

TREES

BASIC TERMINOLOGIES

- **Child** successor nodes or any node can have one or more lines running downward to other nodes



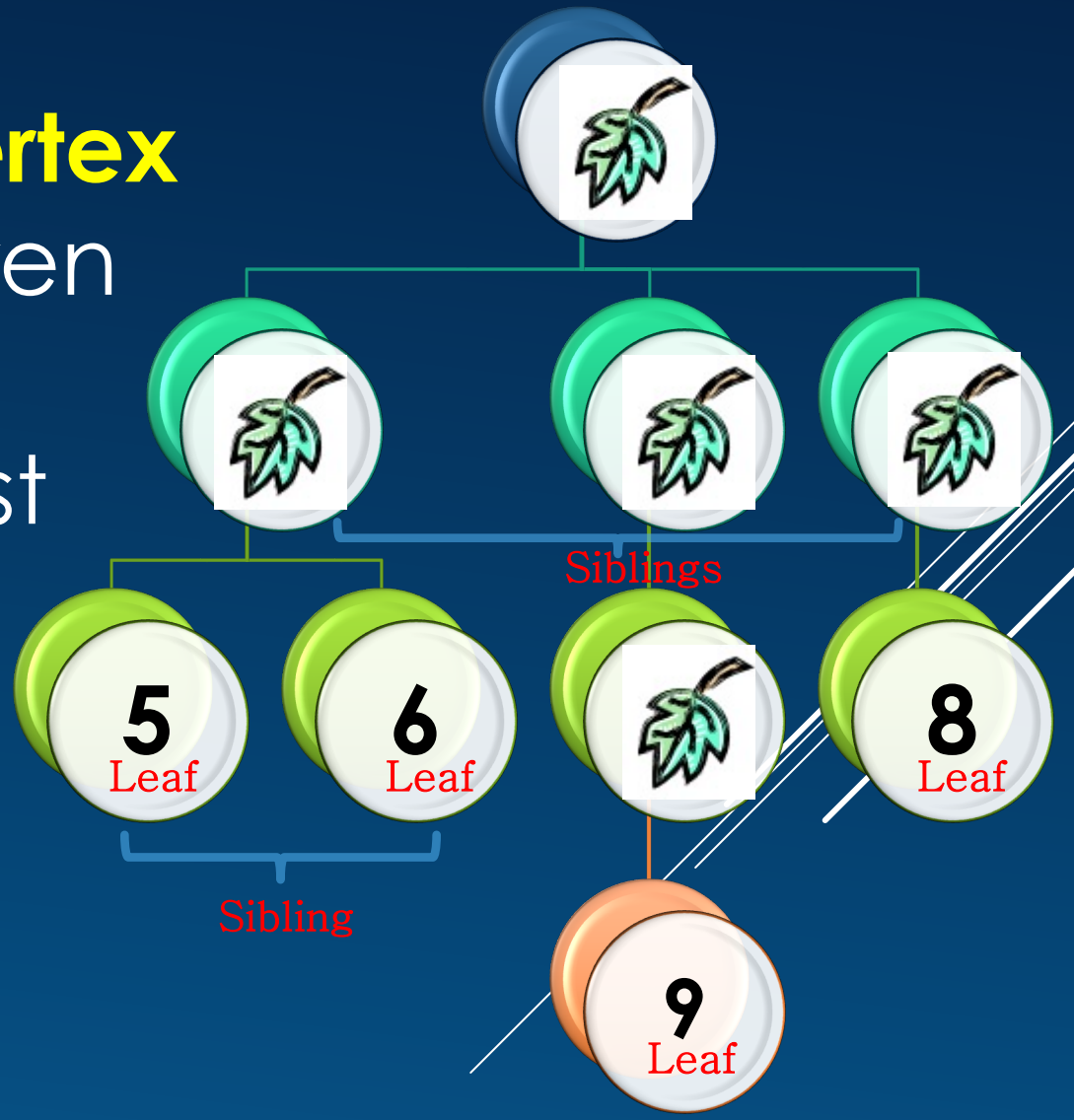
Example:
Nodes 2, 3, 4 are the children of Node 1
Node 7 is the child of Node 3
Node 8 is the child of Node 4

TREES

BASIC TERMINOLOGIES

- **Leaf | Leaves | Terminal Vertex**
  A node that has no children
-  **Internal Node**
  Any node that has at least one non-emptyChild
- **Sibling**
  The child node of same parent

Example:
Sibling of Node 5 is Node 6



TREES

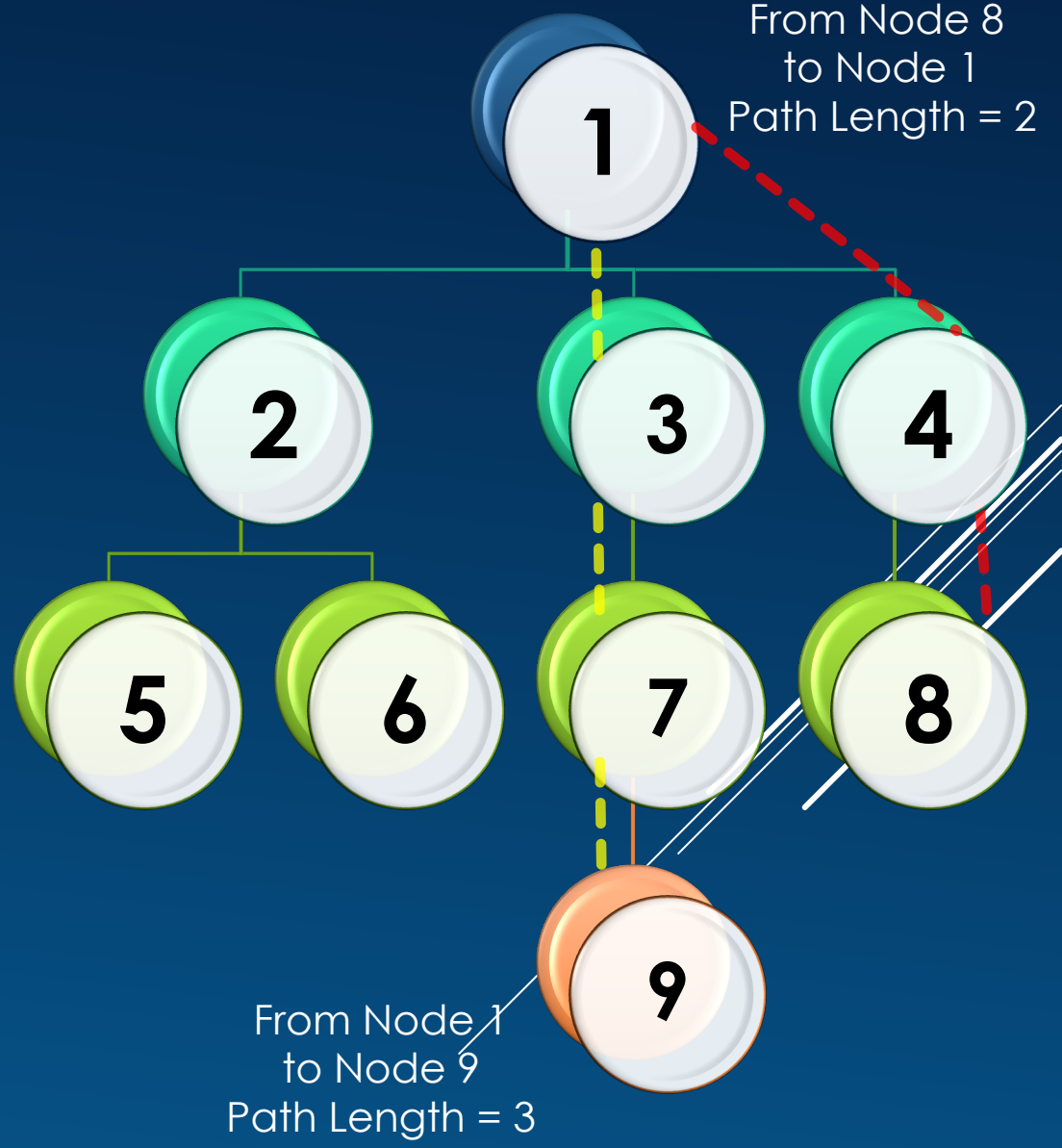
BASIC TERMINOLOGIES

- **Path** resulting sequence of nodes from source to destination.

  - If $n_1, n_2, \ldots n_k$ is a sequence of nodes such that n_i is the parent of n_i+1, then that sequence is a path.

- **Path length**

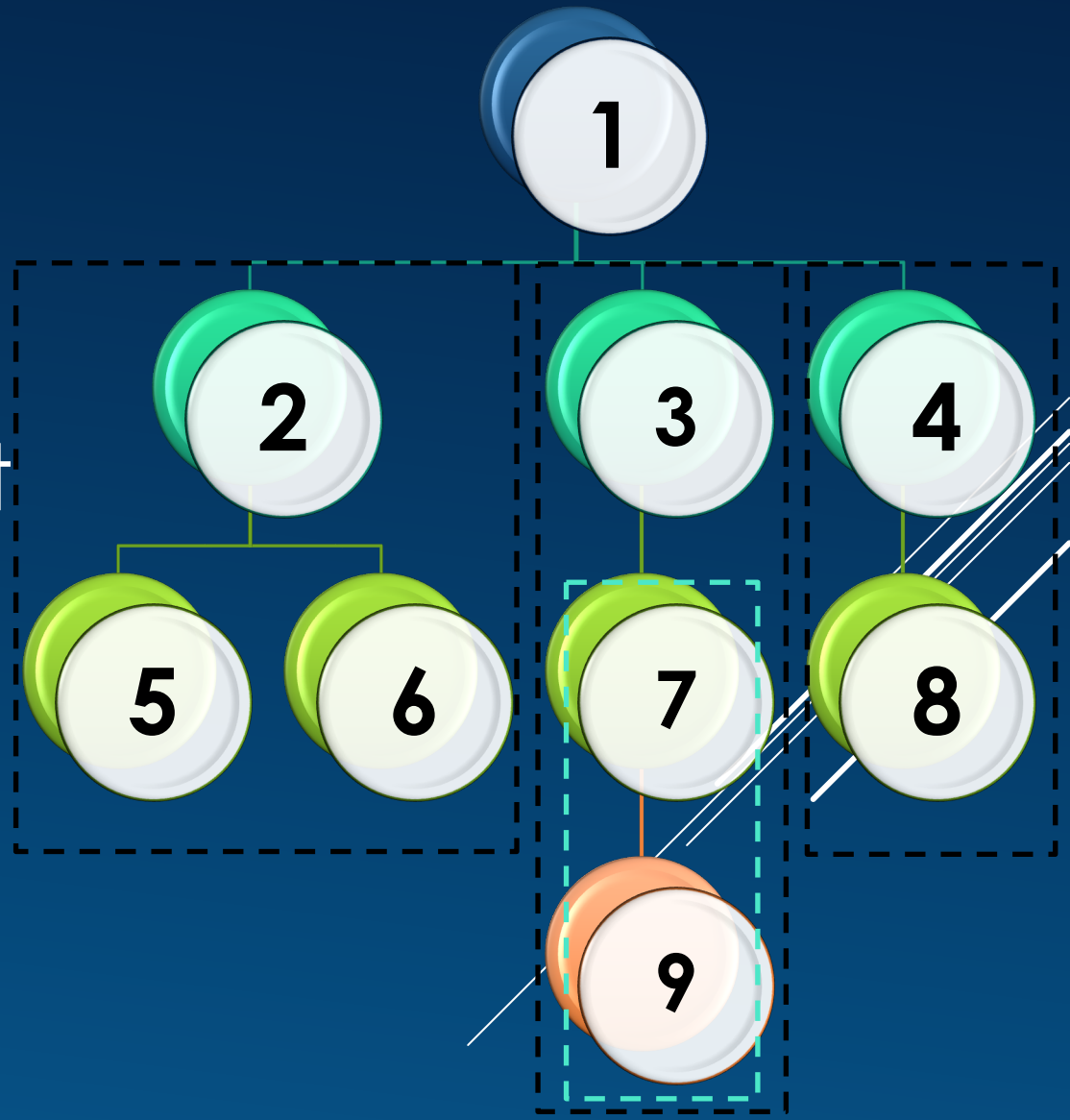
  Is the number of successive edges from source node to destination node.



TREES

BASIC TERMINOLOGIES

- **Subtree** a set of nodes and edges comprised of a parent and all the descendants of that parent

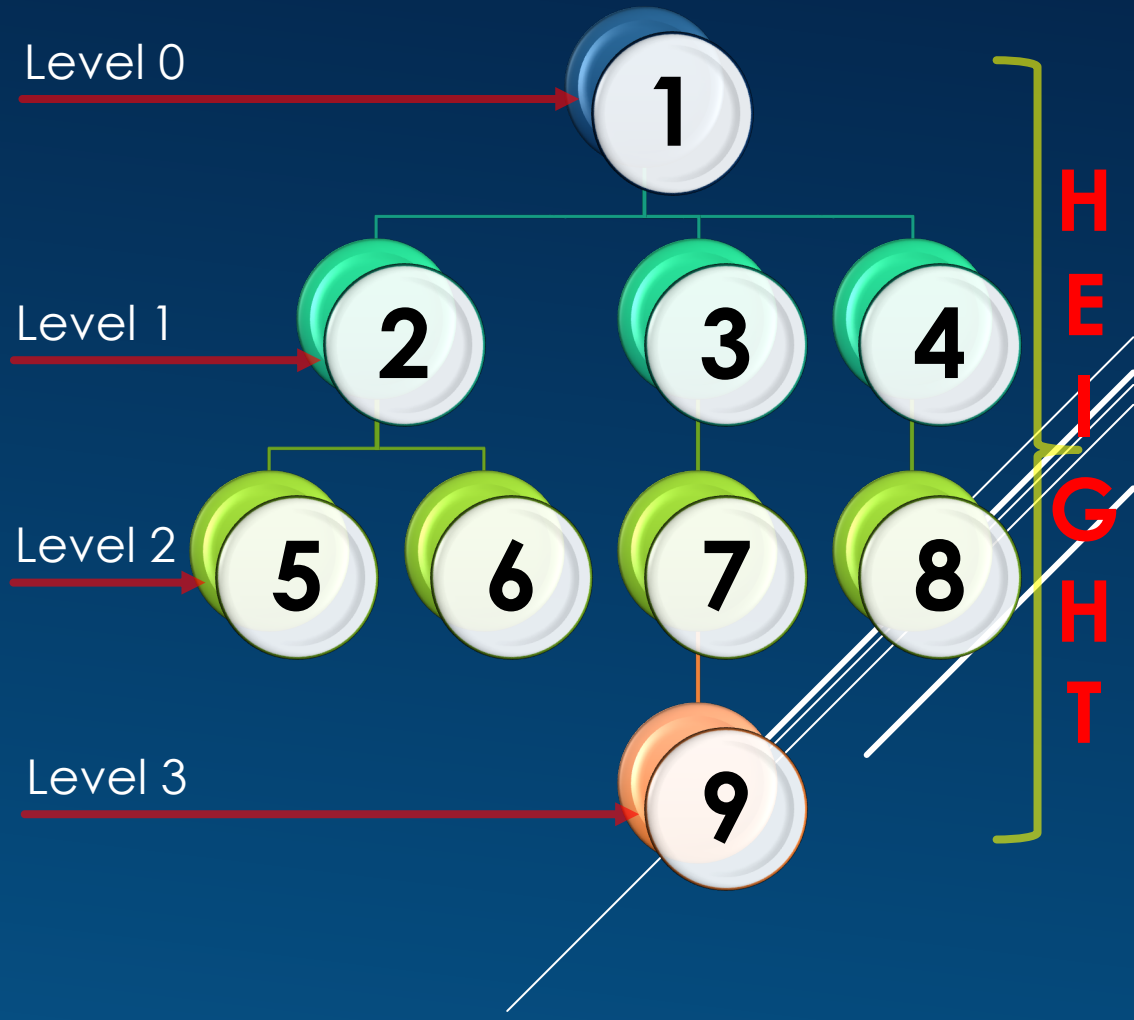


TREES

BASIC TERMINOLOGIES

- **Level** a particular node refers to how many generations the node is from the root.
- **Height** the highest number of nodes that is possible in a way starting from the first node (ROOT) to a leaf node is called the height of tree. Also used to denote the depth

The formula for finding the height of a tree $h = i_{max} + 1$,
where h is the height
i is the max level of the tree



TREES

BASIC TERMINOLOGIES

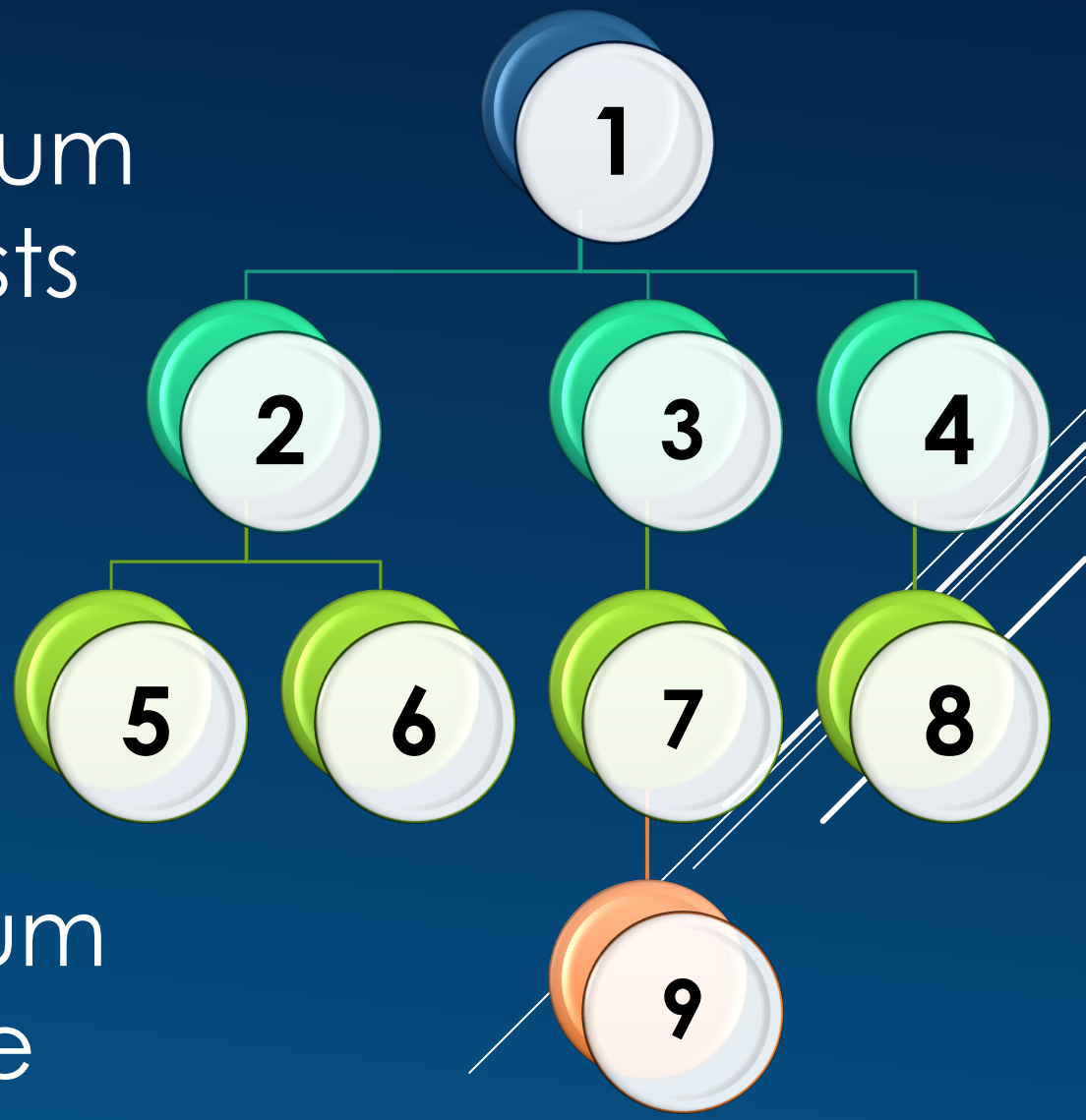
- **Degree of Node** The maximum number of children that exists for a node

Example:

The degree of node 1 is 3
The degree of node 2 is 2
The degree of node 3 is 1
The degree of node 4 is 1

Degree of a tree is the maximum degree of node in a given tree

The degree of a tree is 3



BINARY TREES

- Which no more than two nodes/children. Typically these children are described as left child and right child of the parent node

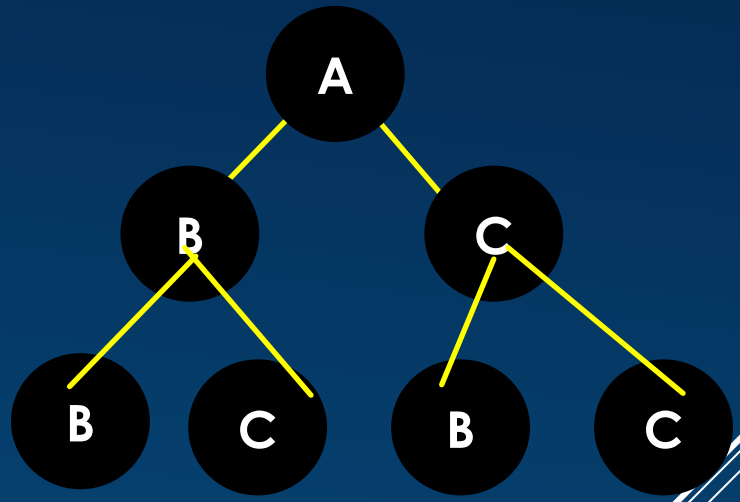
THE MAIN DIFFERENCE BETWEEN A BINARY TREE AND ORDINARY TREE IS:

BINARY TREE	ORDINARY TREE
Can be empty tree	Can't be empty tree
Has exactly two subtrees	Have any number of subtrees
Each element in a binary tree are ordered, left and right subtrees	Subtrees in a tree are unordered

BINARY TREES

The basic operations that are commonly performed on Binary Tree

- Create an empty Binary Tree
- Traversing a Binary Tree
- Inserting a New Node
- Deleting a Node
- Searching for a Node
- Copying the mirror image of a tree
- Determine the total number of Nodes
- Determine the total number leaf Nodes



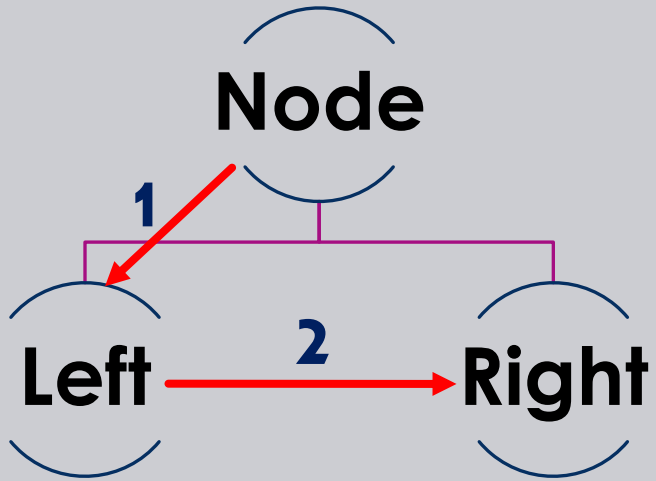
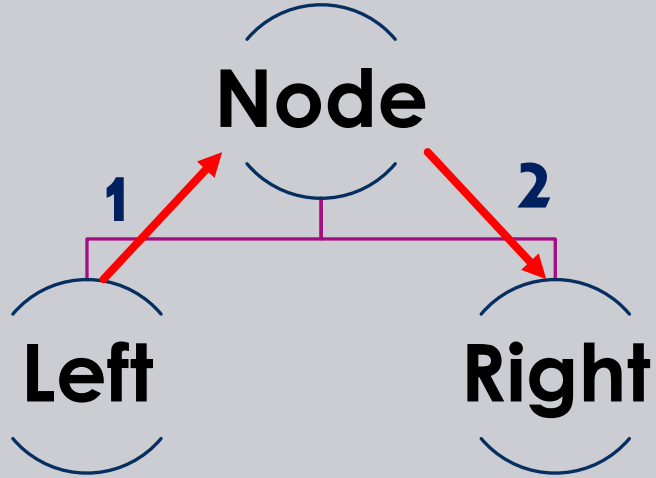
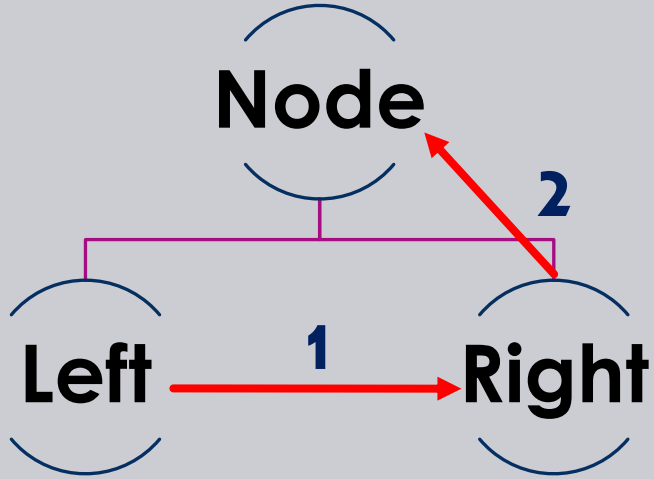
BINARY TREES

The basic operations that are commonly performed on Binary Tree

- Determine the total number non-leaf Nodes
- Find the smallest element in a Node
- Finding the largest element
- Find the Height of the tree
- Finding the Parent | Child | Right Child of an arbitrary node

BINARY TREES

There are three standard ways of traversing a binary tree.

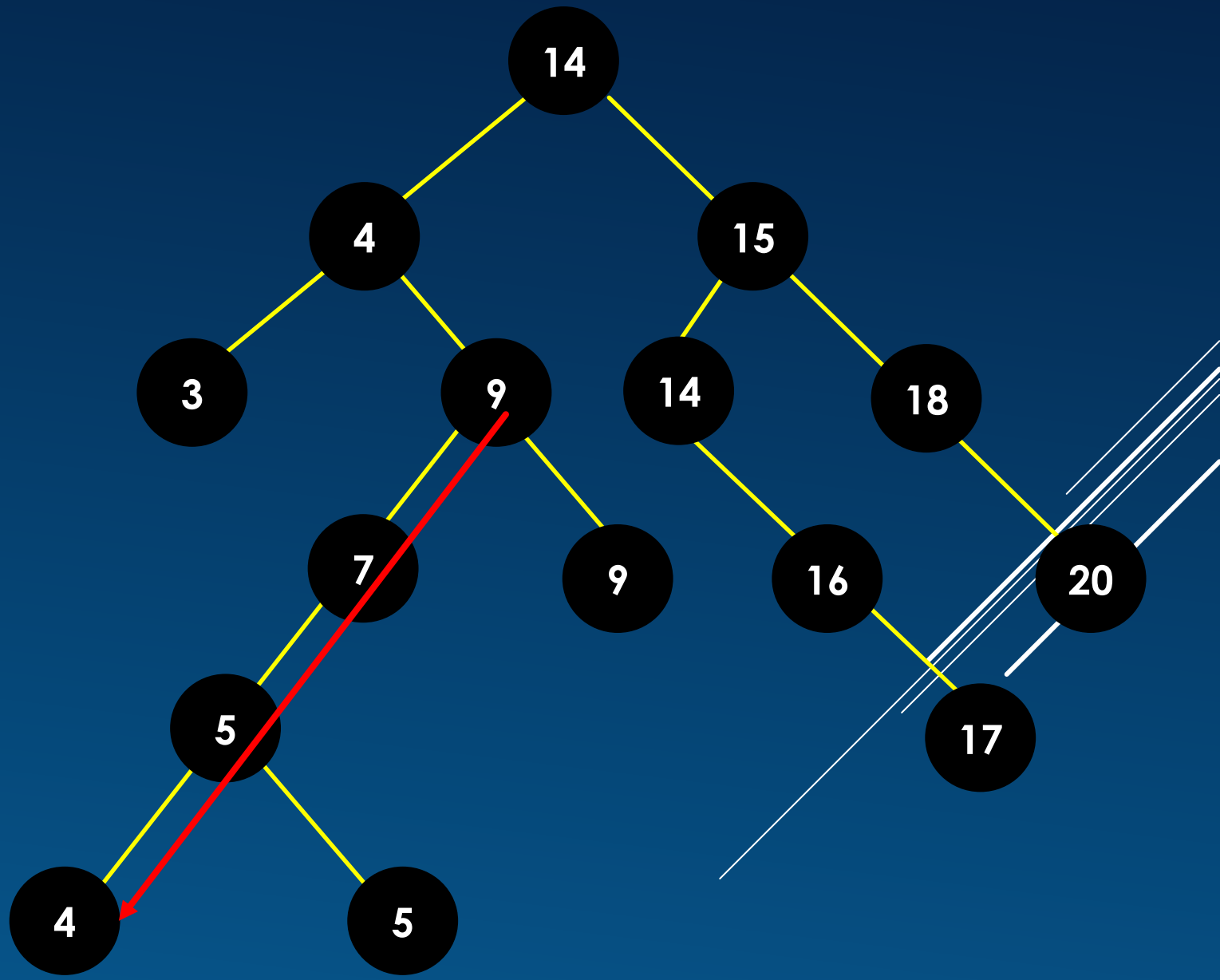
PRE ORDER Traversal (Node-Left-right)	IN ORDER Traversal (Left-node-right)	POST ORDER Traversal (Left-right-node)
		

BINARY TREES

Pre-order Traversal?

Node – Left -Right

14, 4 ,3, 9,7,5,4,5,9,
15,14,16,17,18,20



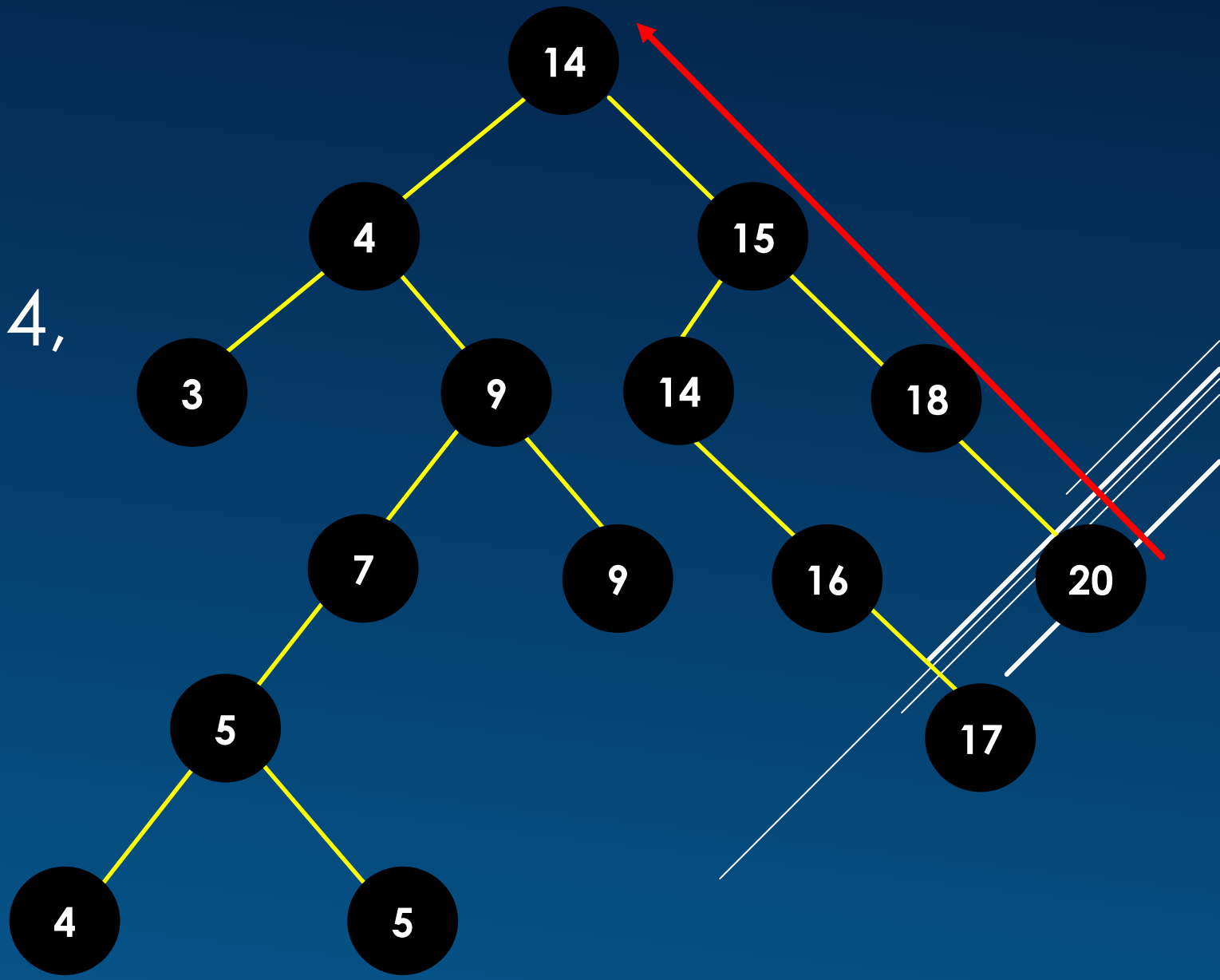
BINARY TREES

Post-order Traversal?

Left – Right –Node

3,4,5,5,7,9,9,4,17,16,14,

20,18,15,14

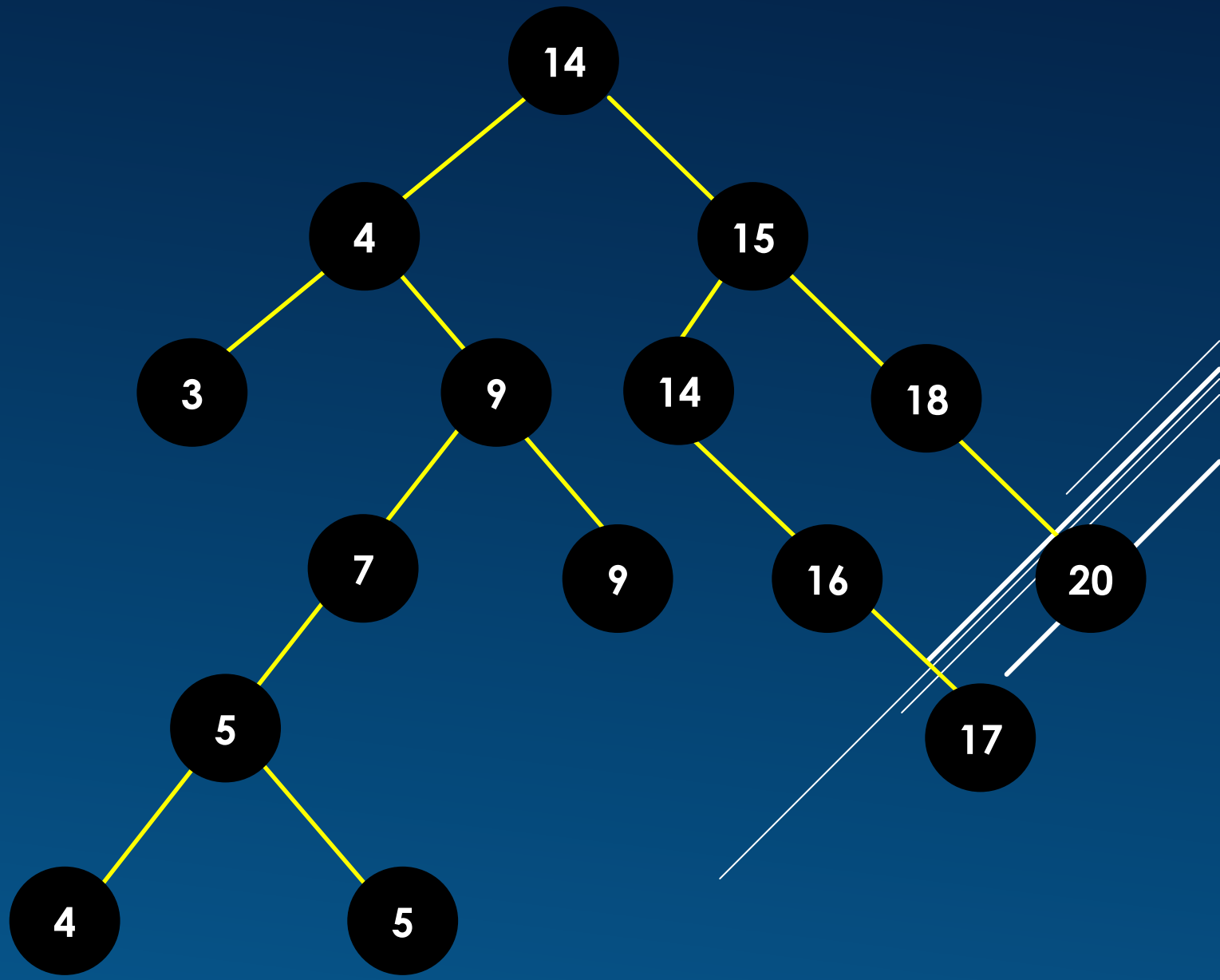


BINARY TREES

In-order Traversal?

Left – Node –Right

3,4,4,5,5,7,9,9,14,
14,16,17,15,18,20



SCHEDULE OF ACTIVITIES

May 16 – Monday (5:00PM – 7:00PM)

Long Quiz (Trees) -

Long Quiz (Graph) -

May 17 – Tuesday (5:00PM – 7:00PM)

Final Examination Lecture

Topics

Introduction of Data Structures, Array, Linked List, StacksQueues, Sorting, Trees, Graphs

BINARY SEARCH TREE (BST)

Stores keys in the nodes in a way so that searching, insertion and deletion can be done efficiently.

Binary search tree is either empty or each node of tree satisfies the following property

- The Key value in the **left child** is **not more than** the value of root
- The key value in the **right child** is **more than or identical** to the value of root
- All the sub-trees, i.e. left and right sub-trees follow the two rules mention above.

BINARY SEARCH TREE (BST)

INSERTION IN BST

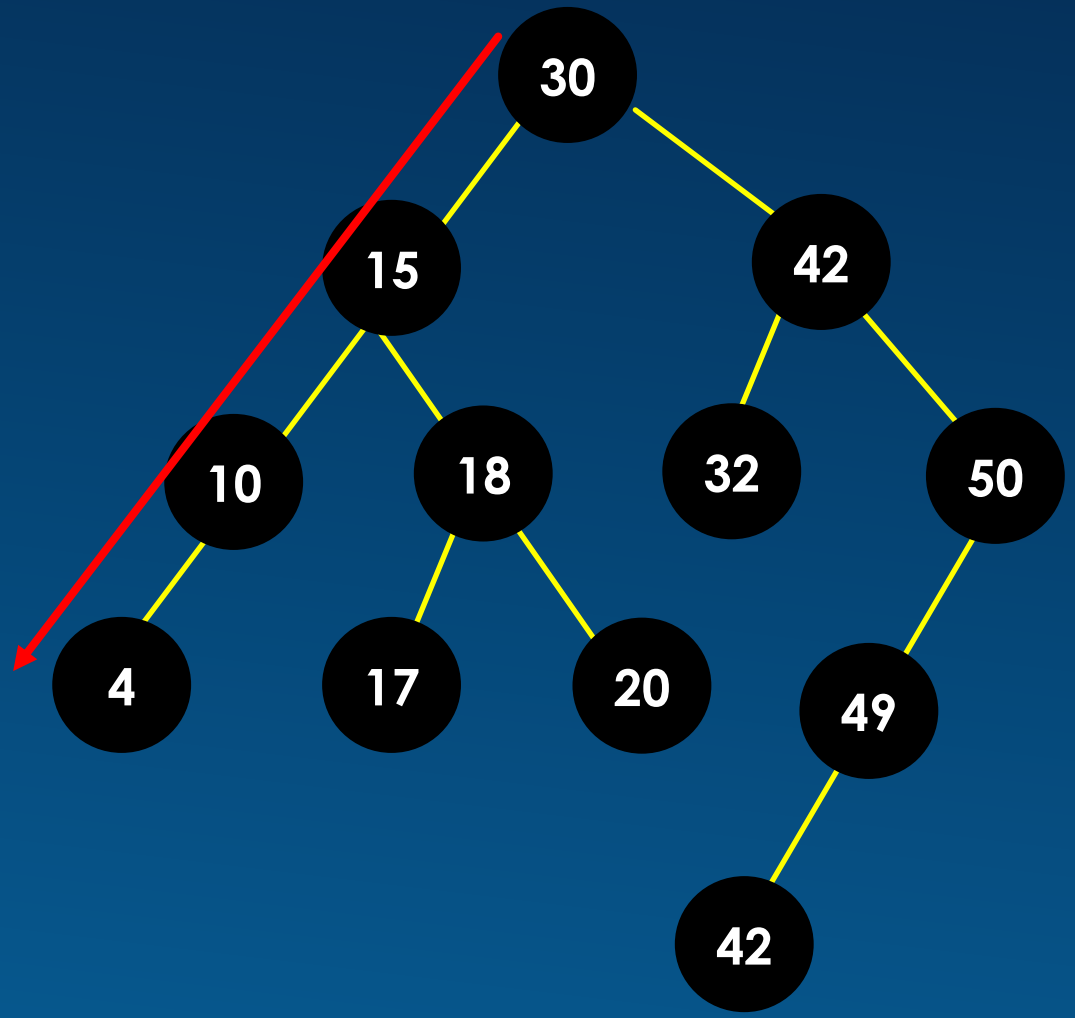
Three steps of insertion

- If the root of the tree is **NULL** then insert the first node and root points to that node.
- If the inserted number is lesser than the root node then insert the node in the left sub-tree.
- If the inserted number is greater than the root node then insert the node in the right sub-tree.

BINARY SEARCH TREE (BST)

Input list of numbers:

30, 15 , 42 , 18 , 50 , 49 , 42 , 20 , 32 , 17 , 10 , 4



Pre order (NLR)

30, 15, 10, 4, 18, 17, 20, 42, 32, 50, 49, 42

In order (LNR)

4, 10, 15, 17, 18, 20, 30, 32, 42,42, 49, 50

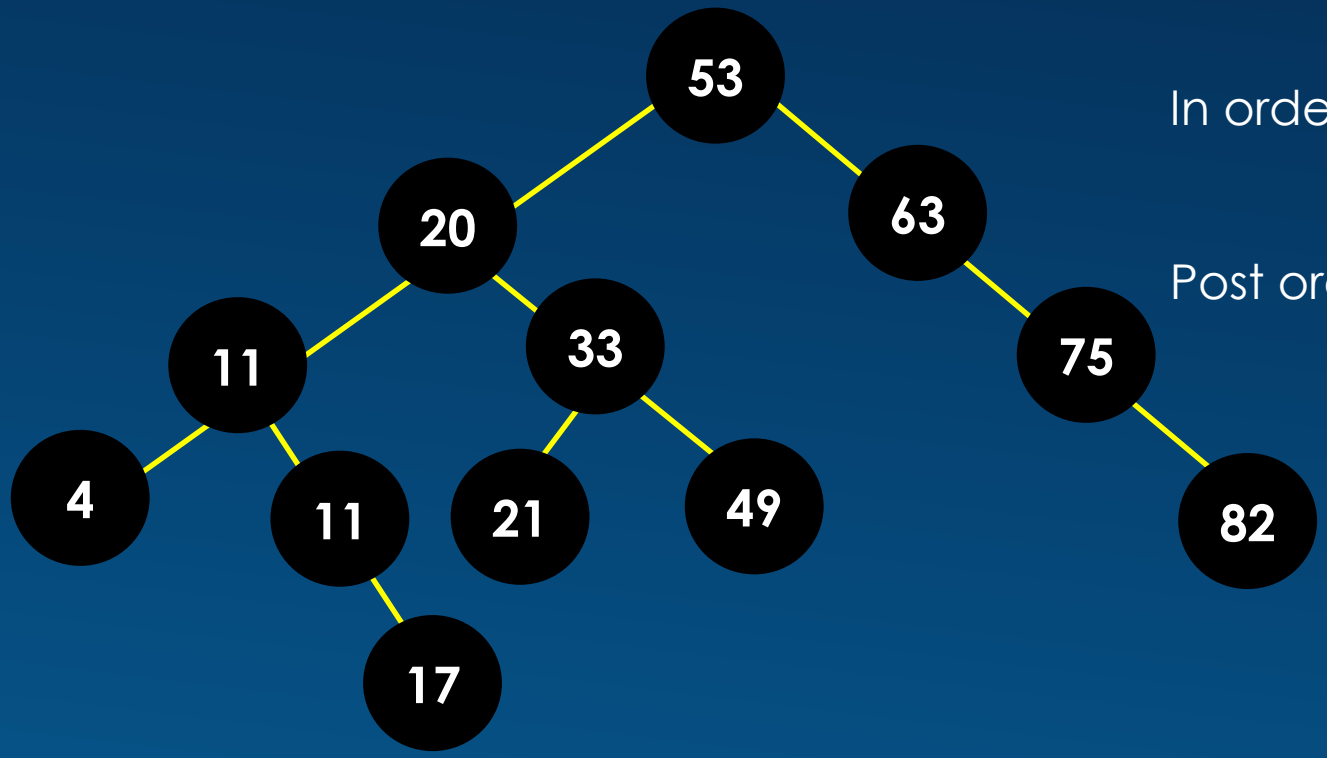
Post order (LRN)

4, 10, 17, 20, 18, 15, 32, 42, 49, 50, 42, 30

BINARY SEARCH TREE (BST)

Input list of numbers:

53, 20, 11, 33, 11, 21, 49, 63, 75, 17, 82, 4



Pre order (NLR)

53, 20,11, 4, 11, 17, 33, 21, 49, 63, 75, 82

In order (LNR)

4, 11, 11, 17, 20, 21, 33, 49, 53, 63, 75, 82

Post order (LRN)

4, 17, 11, 11, 21, 49, 33, 20, 82, 75, 63, 53

BINARY SEARCH TREE (BST)

DELETION IN BST

A deletion of a node, need to consider how it take care of the children of the deleted node.

This has to be done such that the property of the search tree is maintained.

Three cases:

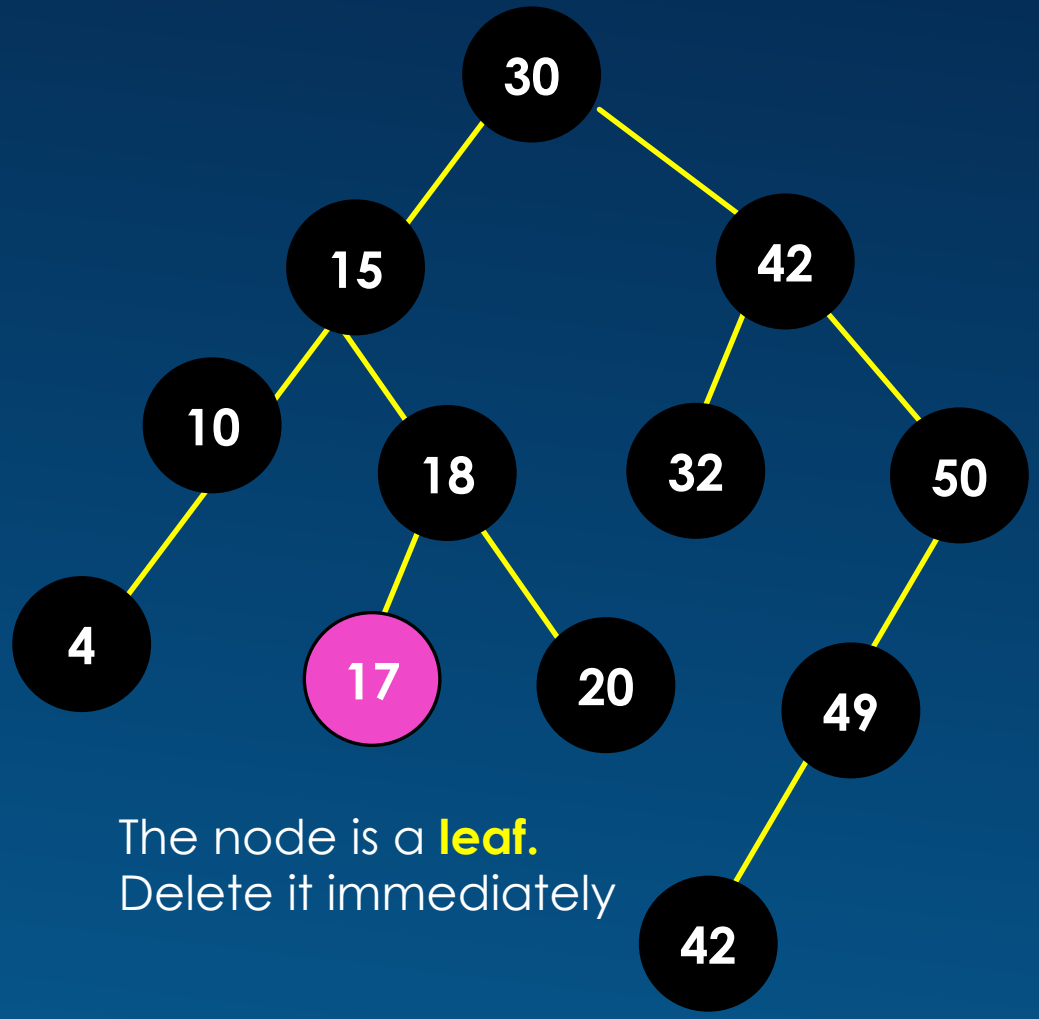
(1) The node is a leaf. Delete it immediately

(2) The node has one child. Adjust a pointer from the parent to bypass that node

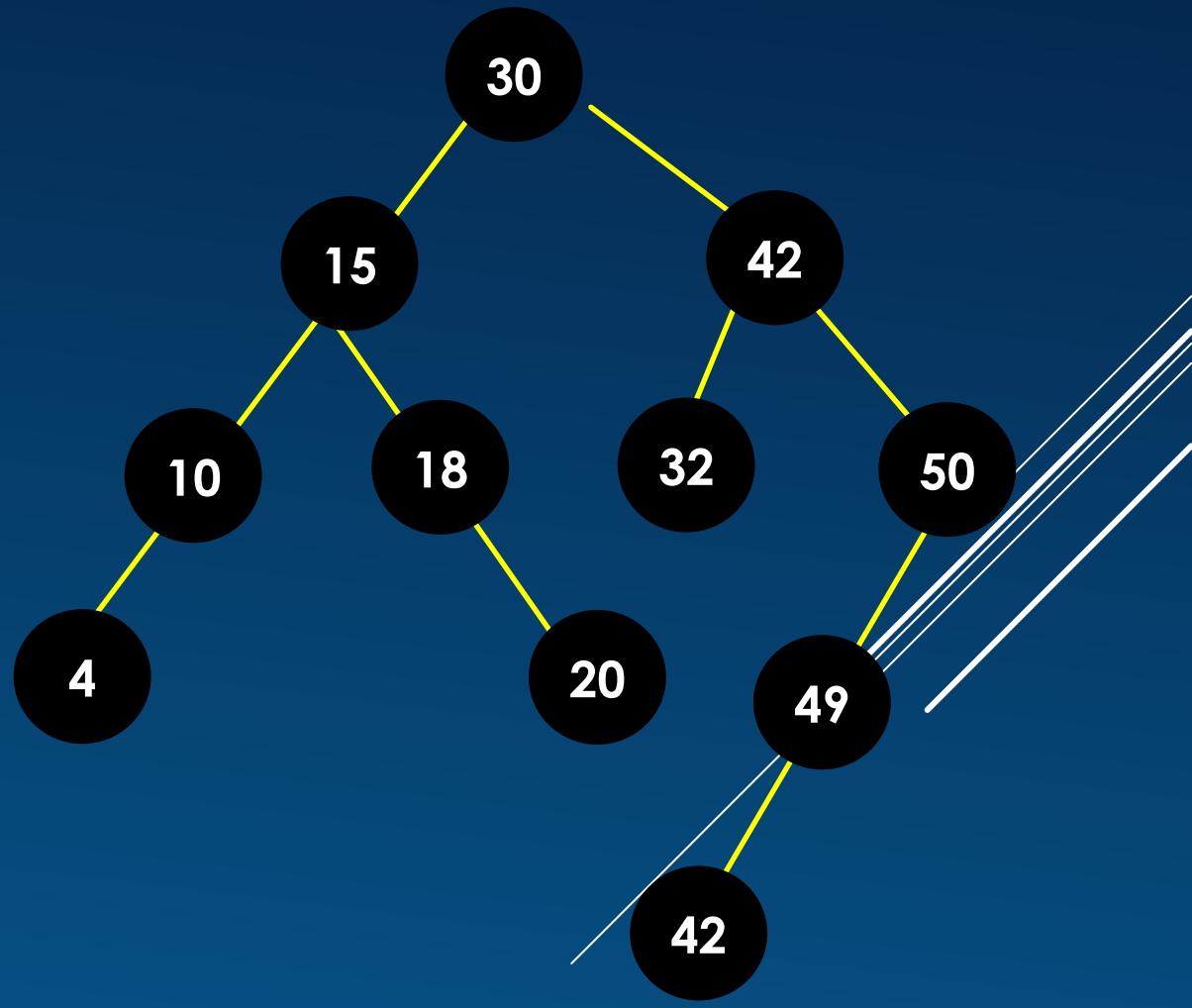
(3) The node has 2 children. Replace the key of that node with the minimum element at the right subtree. Delete the minimum element

BINARY SEARCH TREE (BST)

Delete 17

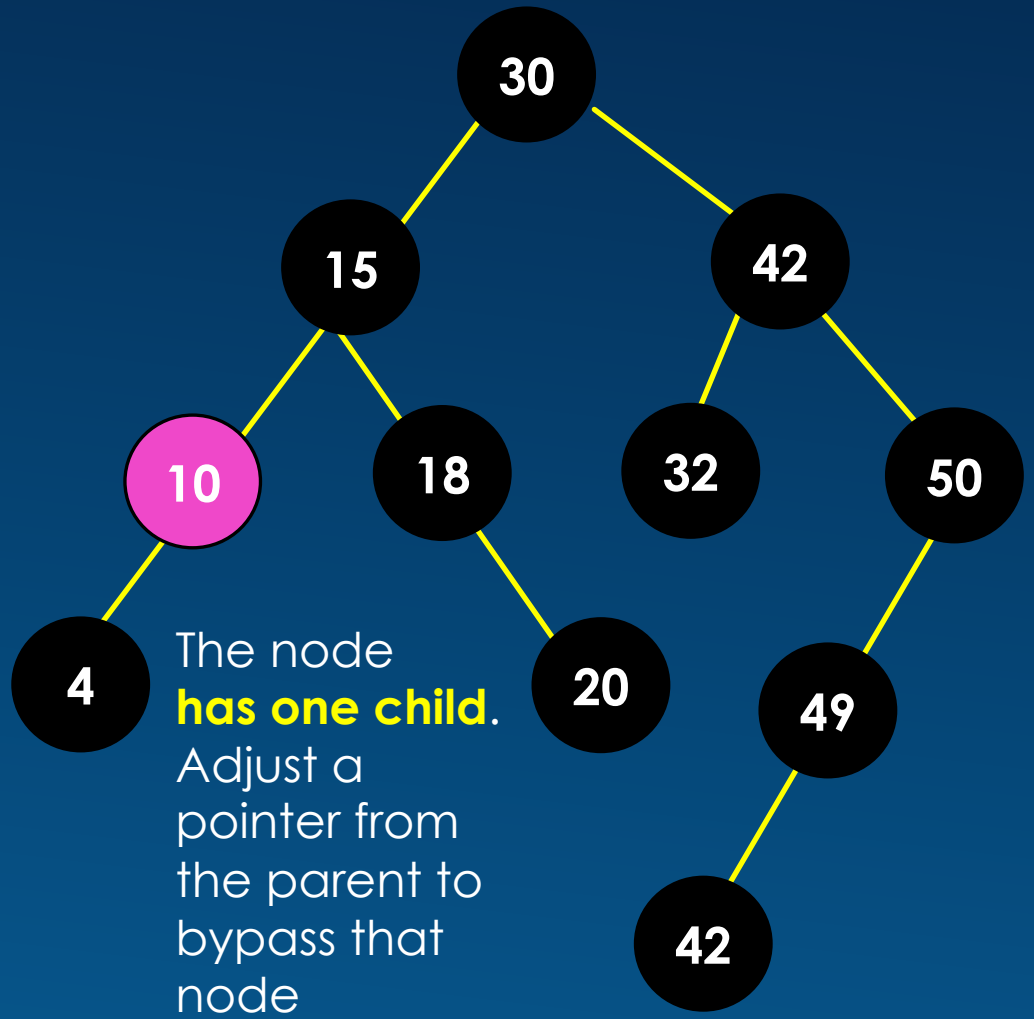


The node is a **leaf.**
Delete it immediately

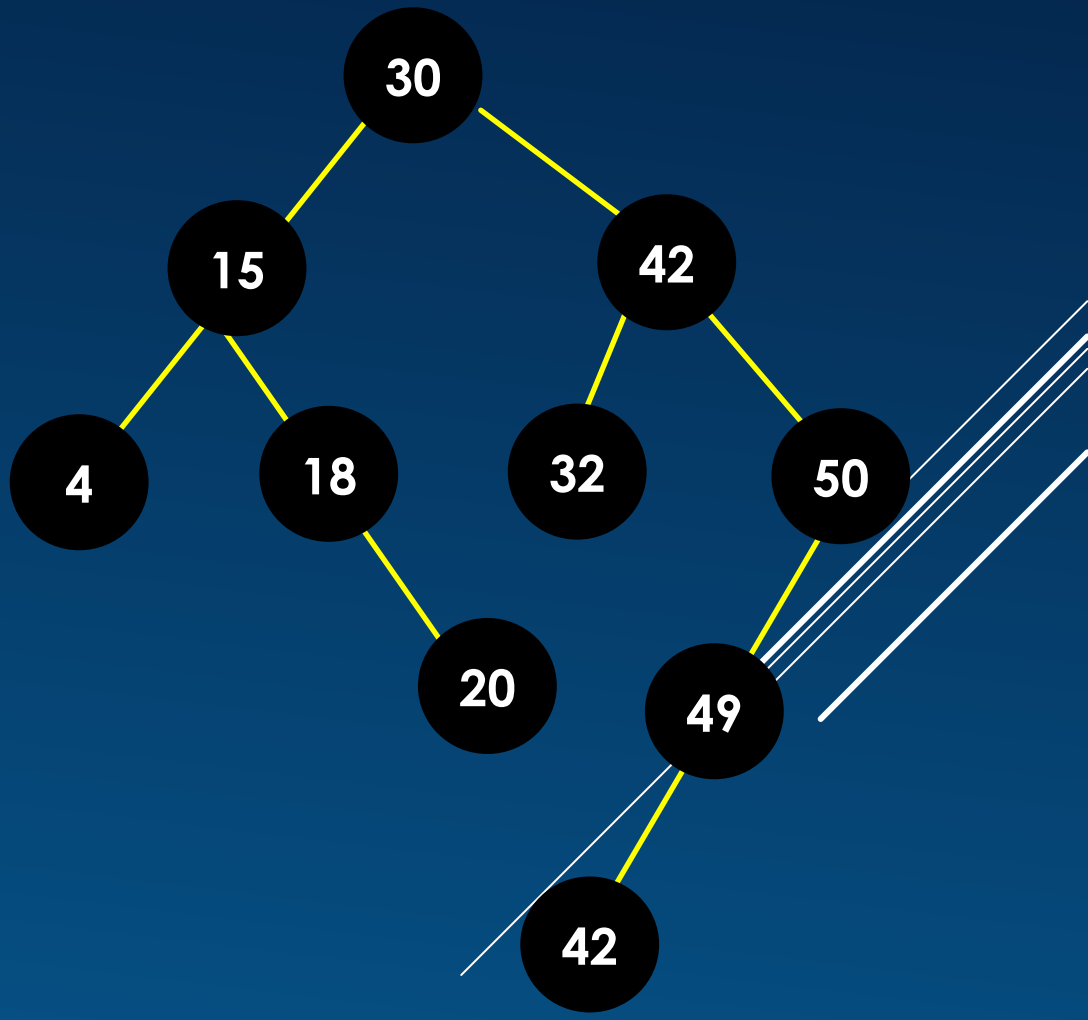


BINARY SEARCH TREE (BST)

Delete 10

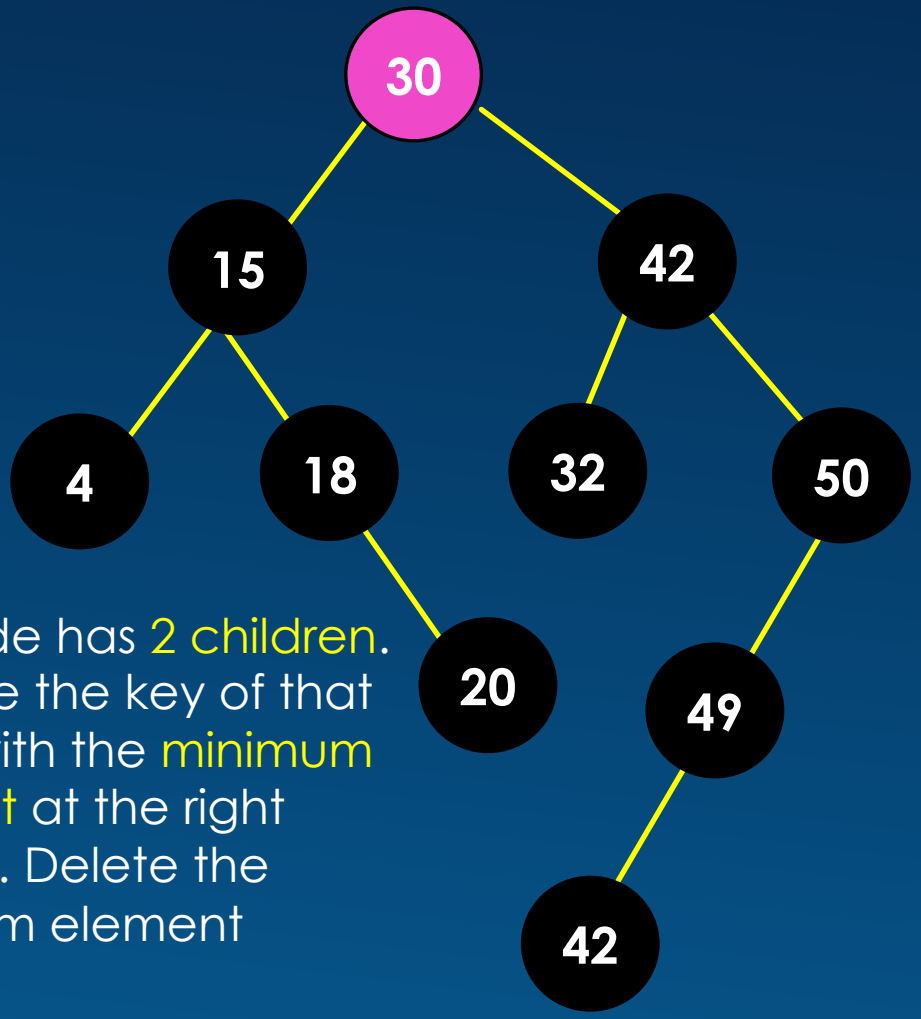


The node **has one child**. Adjust a pointer from the parent to bypass that node

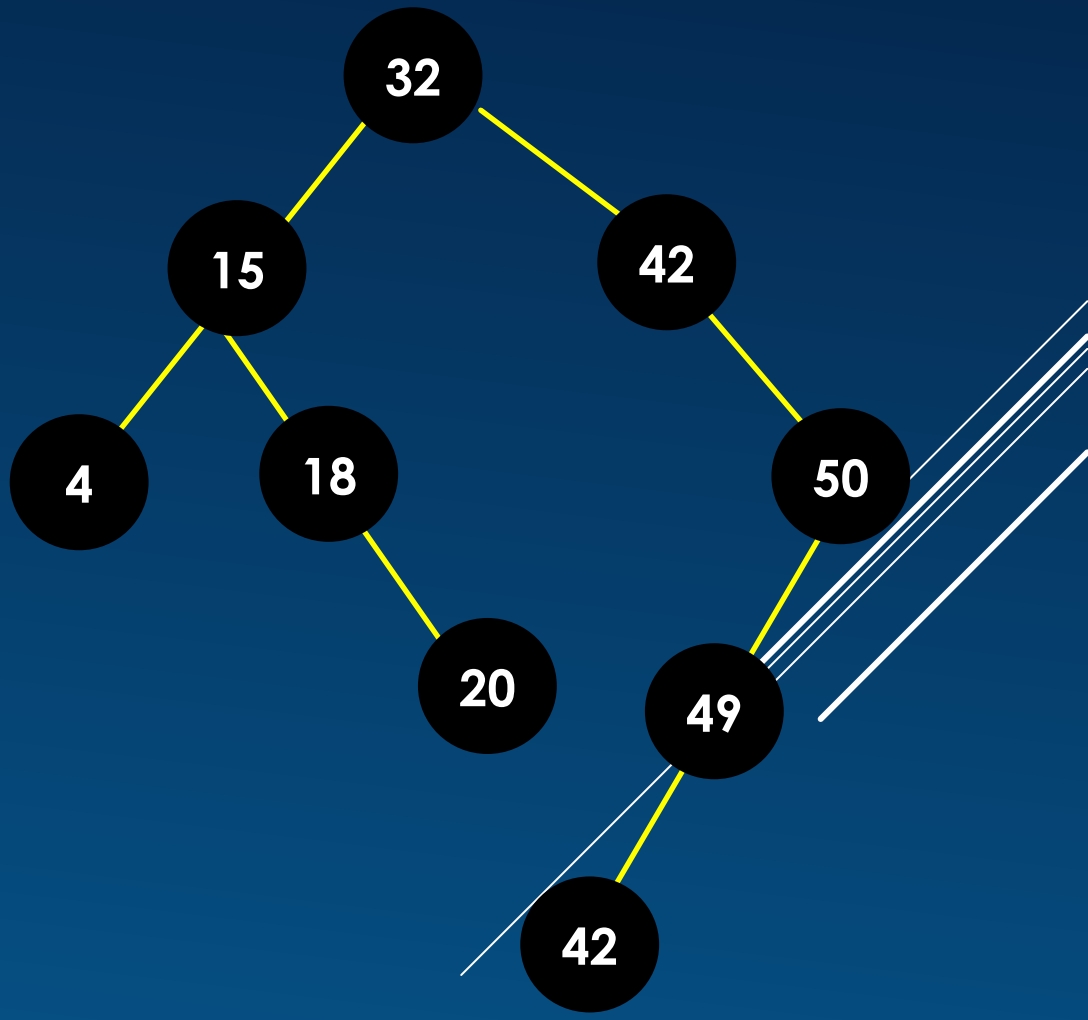


BINARY SEARCH TREE (BST)

Delete 30



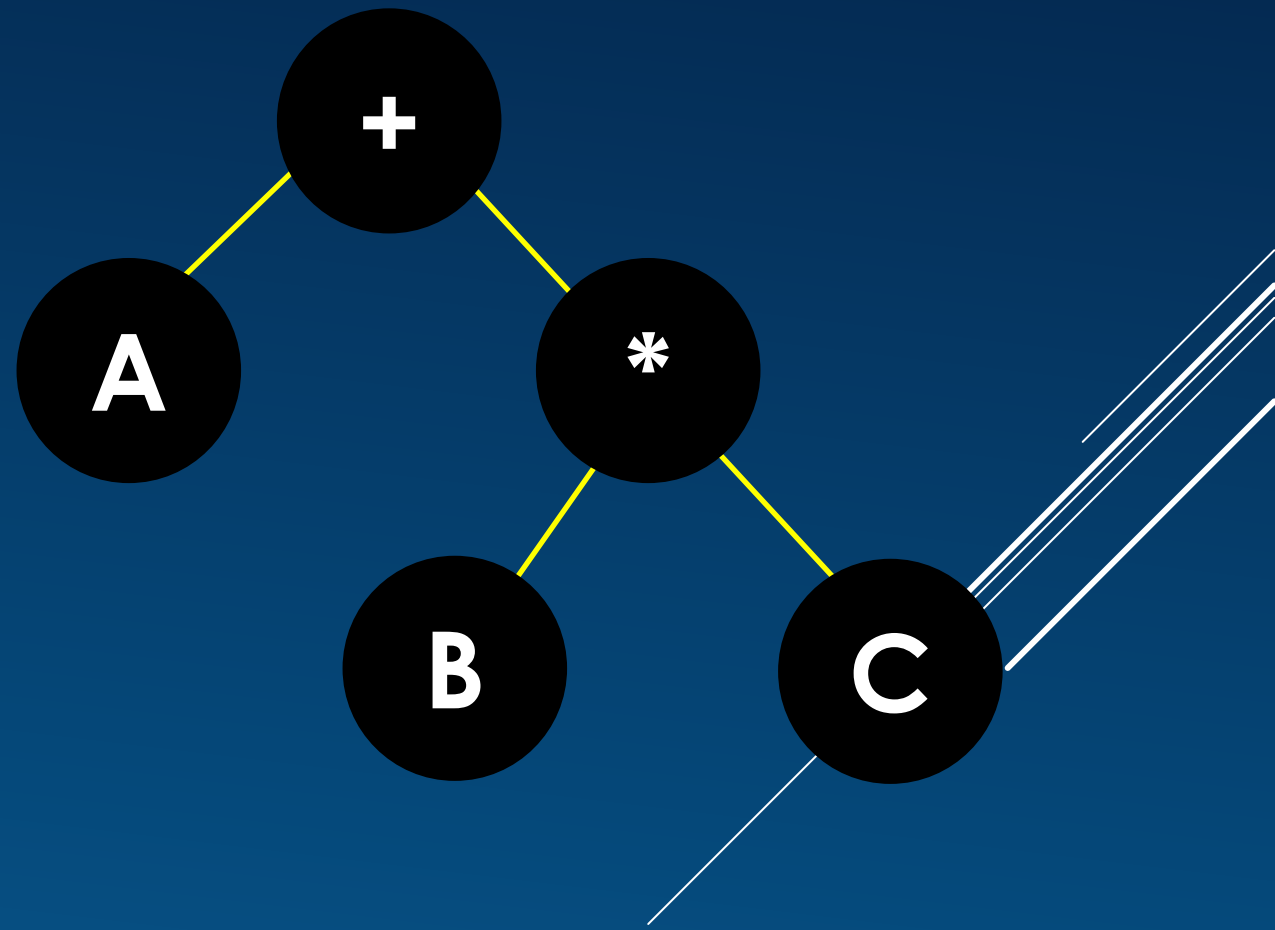
The node has 2 children. Replace the key of that node with the minimum element at the right subtree. Delete the minimum element



EXPRESSION TREE P E MD AS

A + B * C

Node/Parent (Operator)
Leaf/Child (Operand)



Pre-order (NLR)
+ A * B C

Post – order (LRN)
A B C * +

EXPRESSION TREE

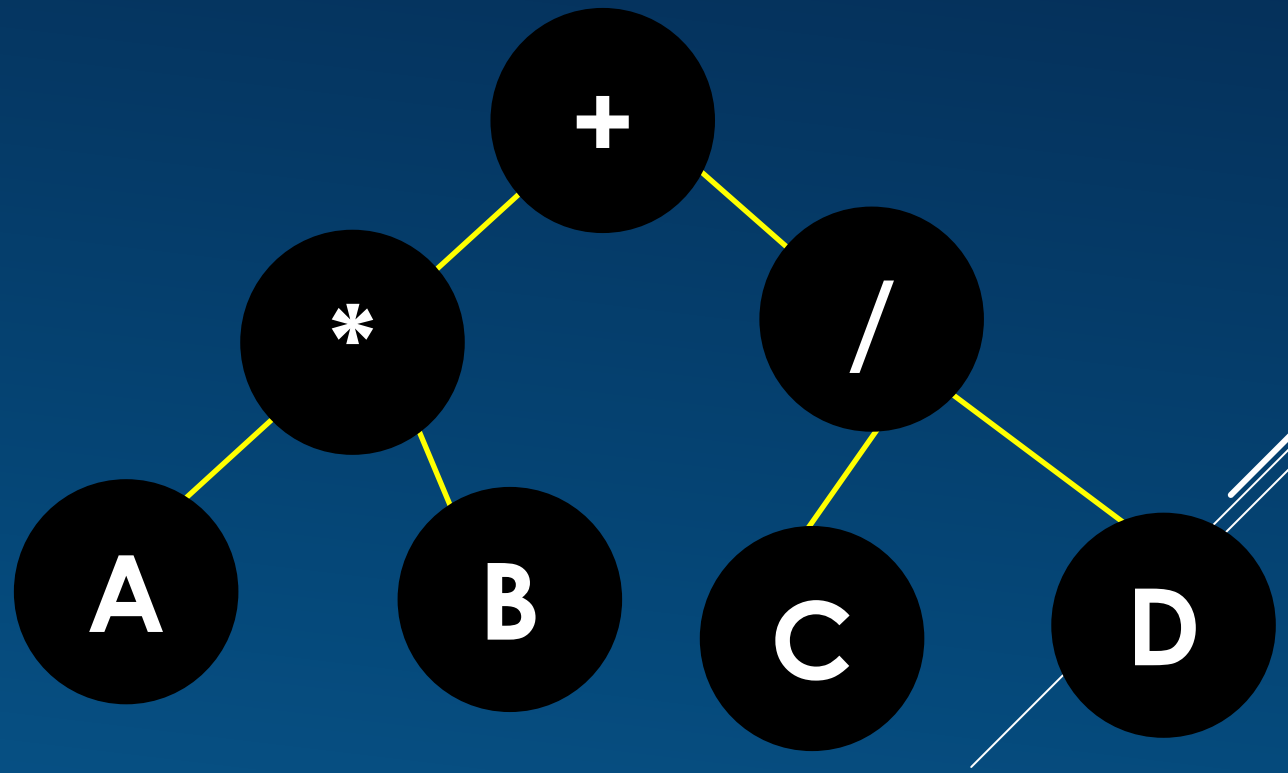
A * B + C / D

Pre-order

+ * A B / C D

Post – order

A B * C D / +



EXPRESSION TREE

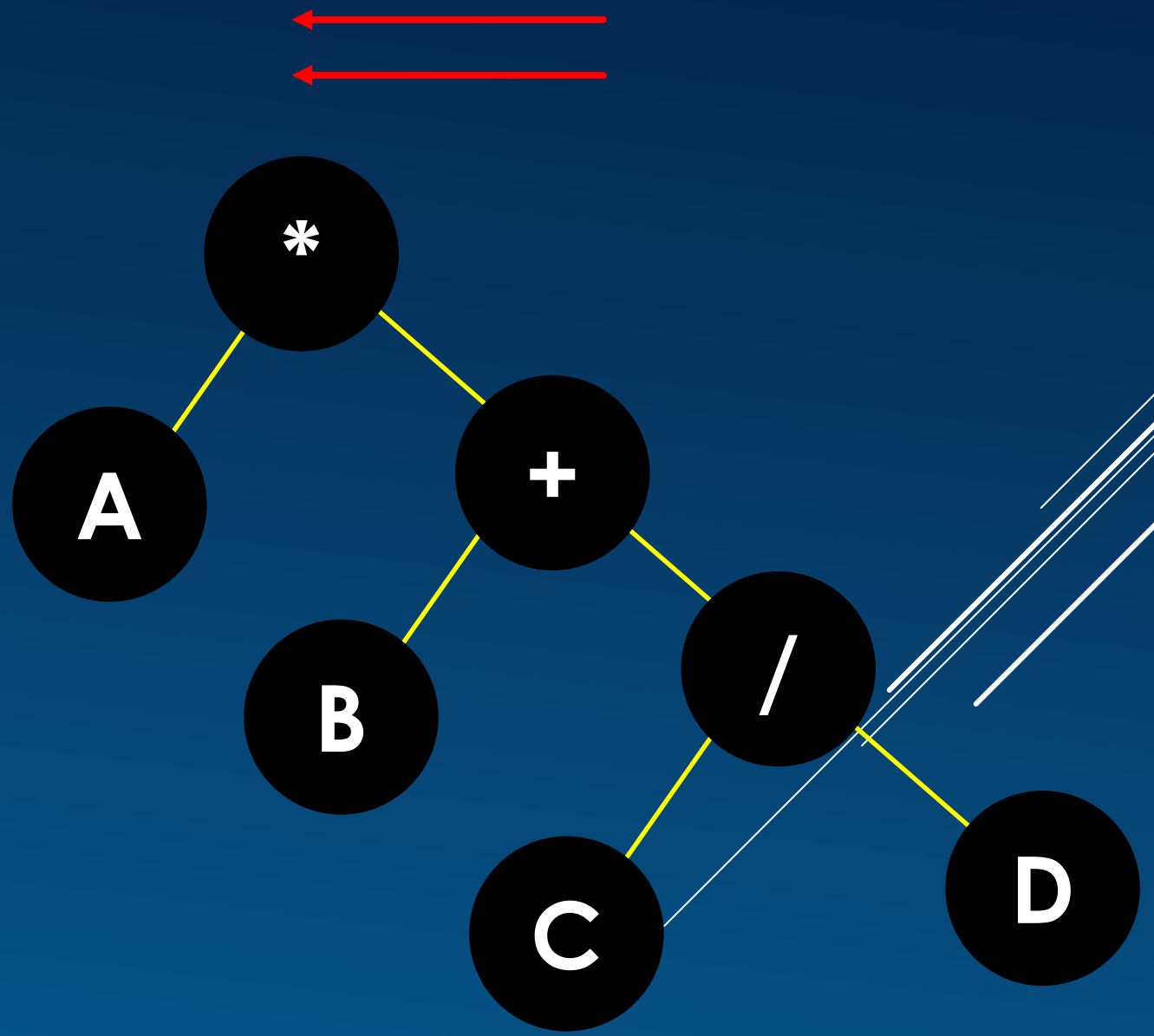
A * (B + C / D)

Pre-order

* A + B / C D

Post – order

A B C D / + *



EXPRESSION TREE

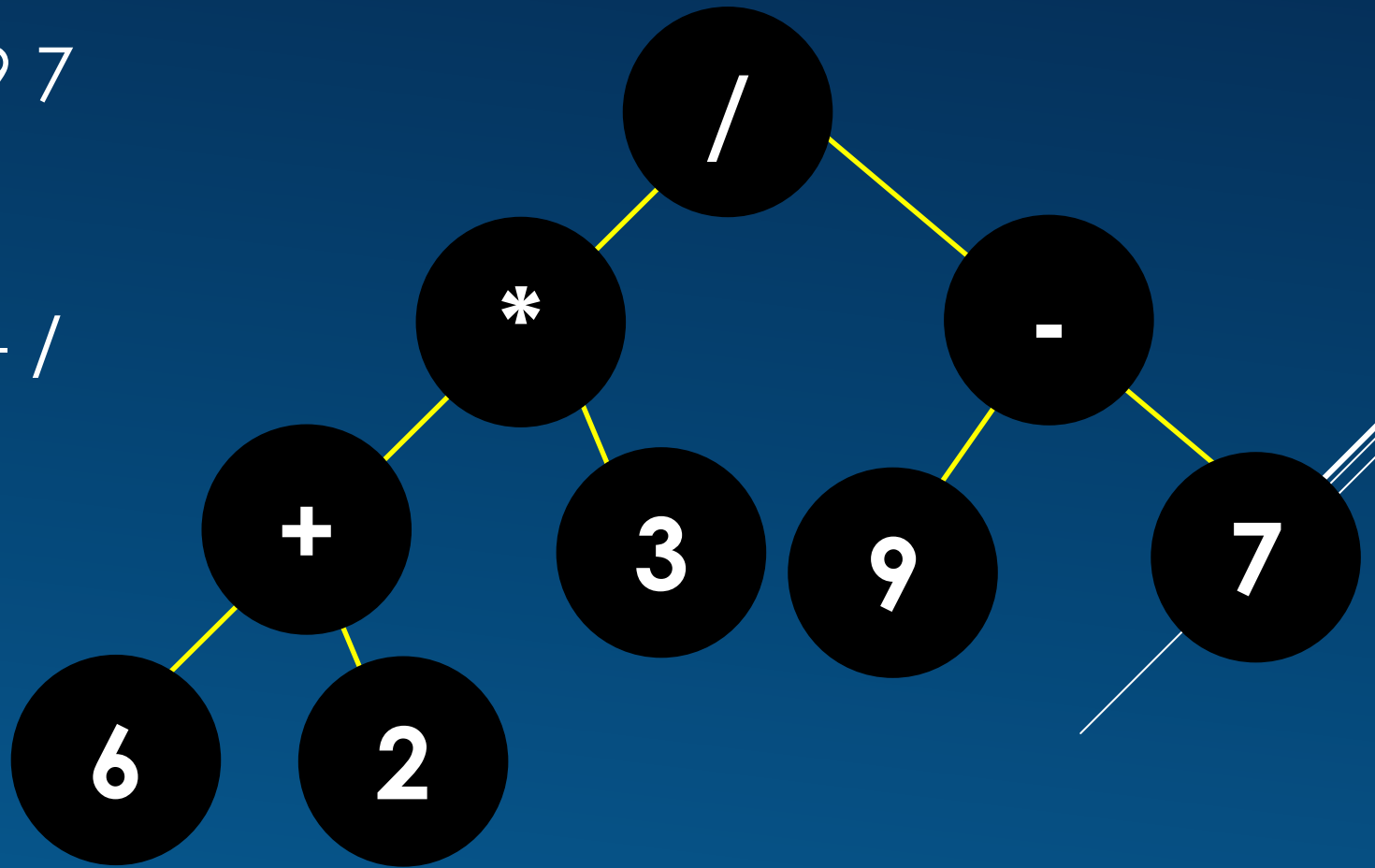
(6 + 2) * 3 / (9 - 7)

Pre-order

/ * + 6 2 3 – 9 7

Post – order

6 2 + 3 * 9 7 - /

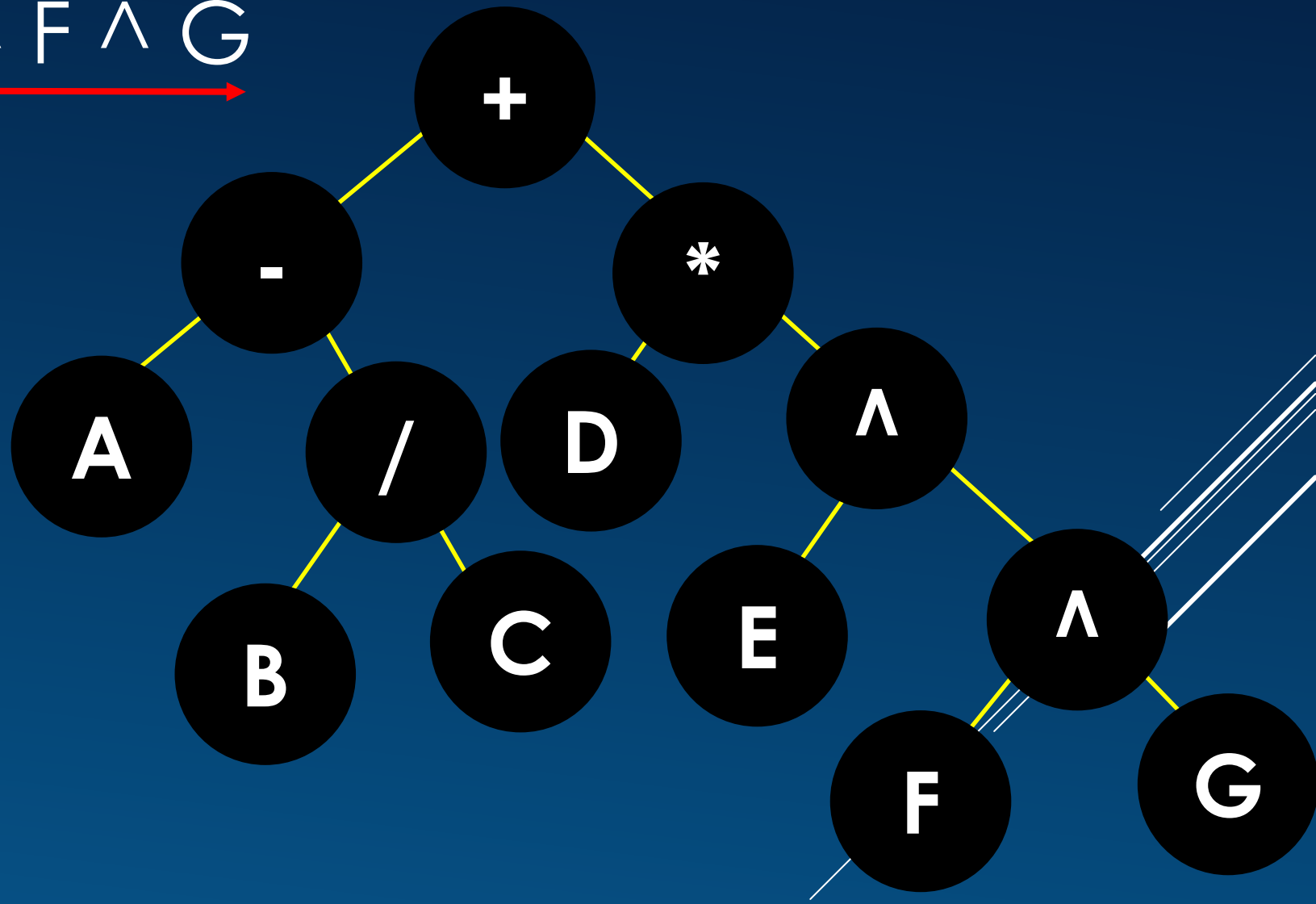


EXPRESSION TREE

A – B / C + D * E ^ F ^ G

Pre-order
+ - A / B C * D ^ E ^ F G

Post – order
A B C / - D E F G ^ ^ * +



SCHEDULE OF ACTIVITIES

May 16 – Monday (5:00PM – 7:00PM)

Long Quiz (Trees) -

Long Quiz (Graph) -

May 17 – Tuesday (5:00PM – 7:00PM)

Final Examination Lecture

Topics

Introduction of Data Structures, Array, Linked List, StacksQueues, Sorting, Trees, Graphs