JURUSAN
TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG

# Week 14
# ~Tree~

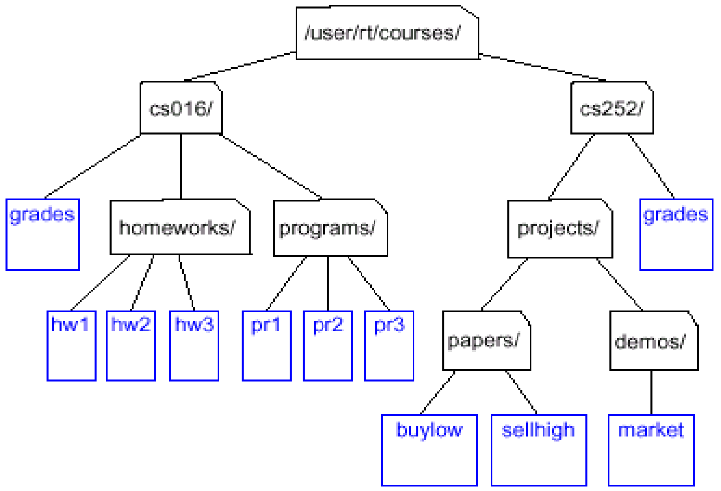
Teaching Team of Algorithm and Data Structure
Information Technology Dept.

## Learning Outcome

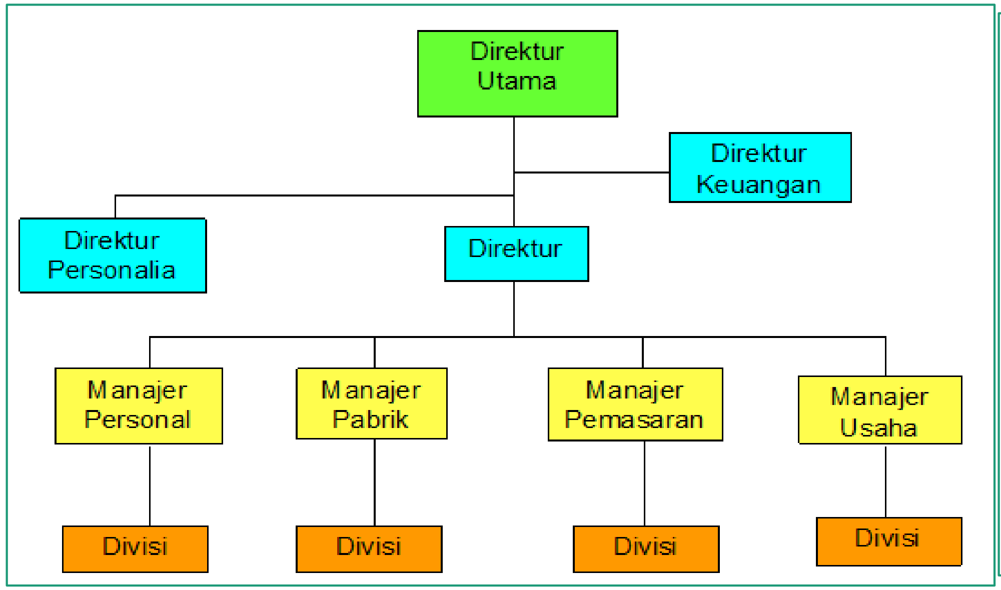
- Students must be able to understand basic concept of tree
- Students must be able to understand basic concept of binary tree and binary search tree, as the special case of tree
- Students must be able to have a good knowledge of tree operation

## Introduction

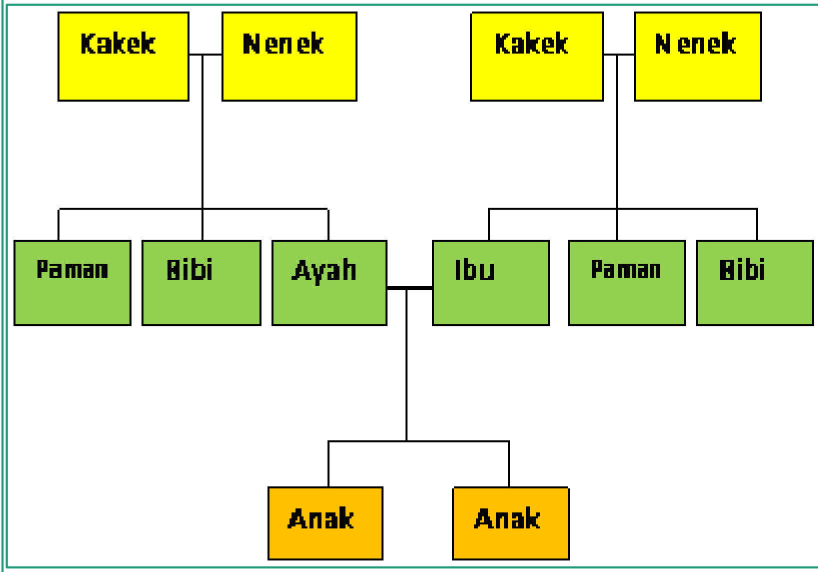
- *Linked List, Stack, and Queue are linear data structures (Linear List).*
- *Linear List* → used for sequential serial data.
  - Example: queue, name of day of week, name of month of year, etc.
- Tree is a non-linear data structure that has special properties.
- Tree is one form of implementation of many linked lists (double linked lists) that are usually used to describe the hierarchical relationship between the elements that exist.
- *Tree* → used for hierarchical (one to many) data.
  - Example :
    - Family tree
    - The organizational structure.



File System



Organization structure



Family tree

## Definition of Tree

- *Tree is a collection of elements that are interconnected in a hierarchical manner (one to many).*
- *Elements in a tree are called nodes.*

**Rules :**

- A node can only have one parent. Except root, it doesn't have a parent.
- Each node can have zero or many children (one to many).
- Nodes that do not have children are called leaf.

## Why tree?

- <u>Problems:</u>

  Public transportation routes from Arjosari (bus station) to tourist attractions in Malang.

  ❖Tlogomas : Arjosari→blimbing→tlogomas→Tlogomas Swimming Pool

  ❖Sengkaling : Arjosari→blimbing→tlogomas→jetis→Sengkaling Swimming Pool

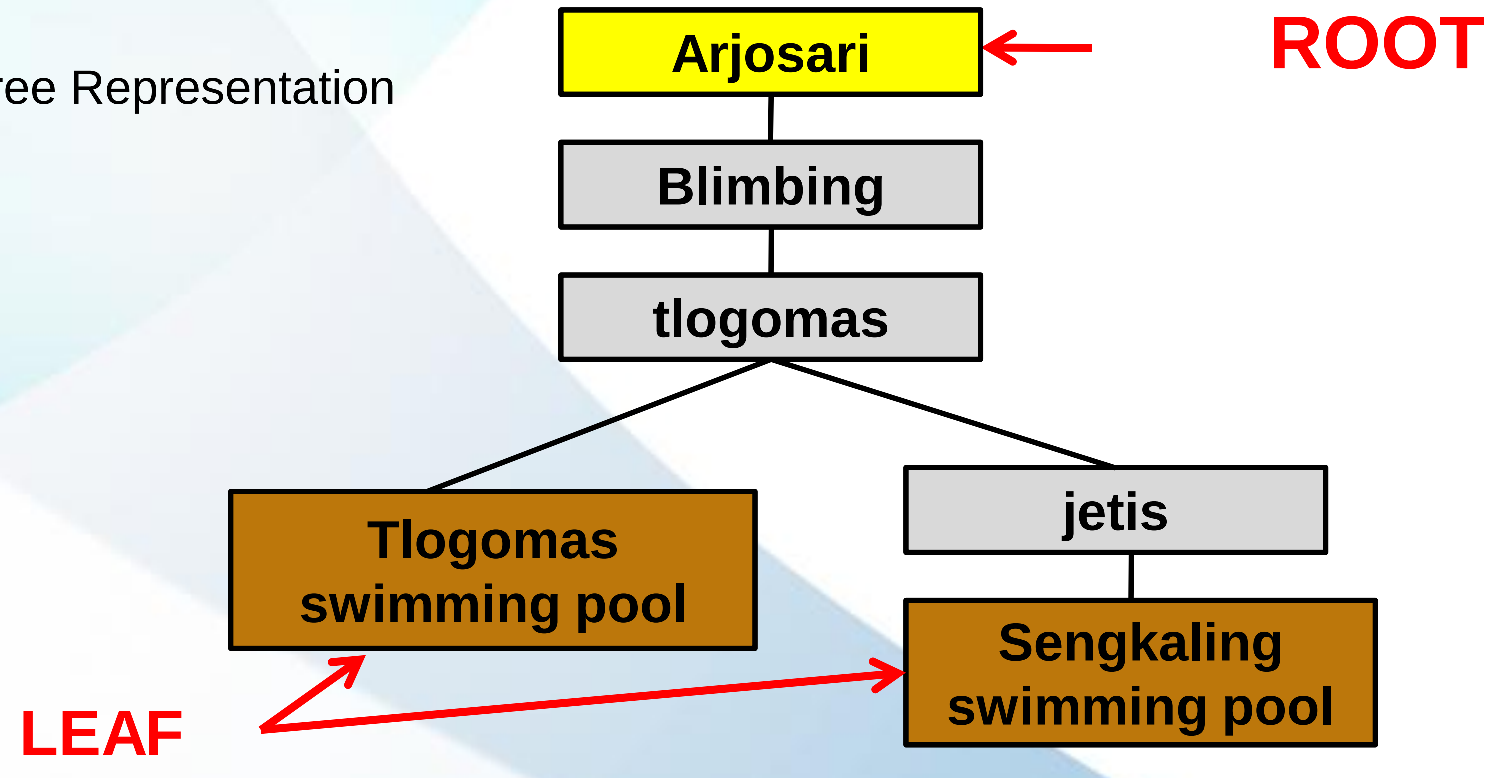
- Could it be represented by linear list?

  Yes it could, but it will be complicated.

- It will be easier to represent by using **tree** than linear list data structure. Since the data is hierarchical (*one to many*).

## Why Tree?

Tree Representation



**PATH**

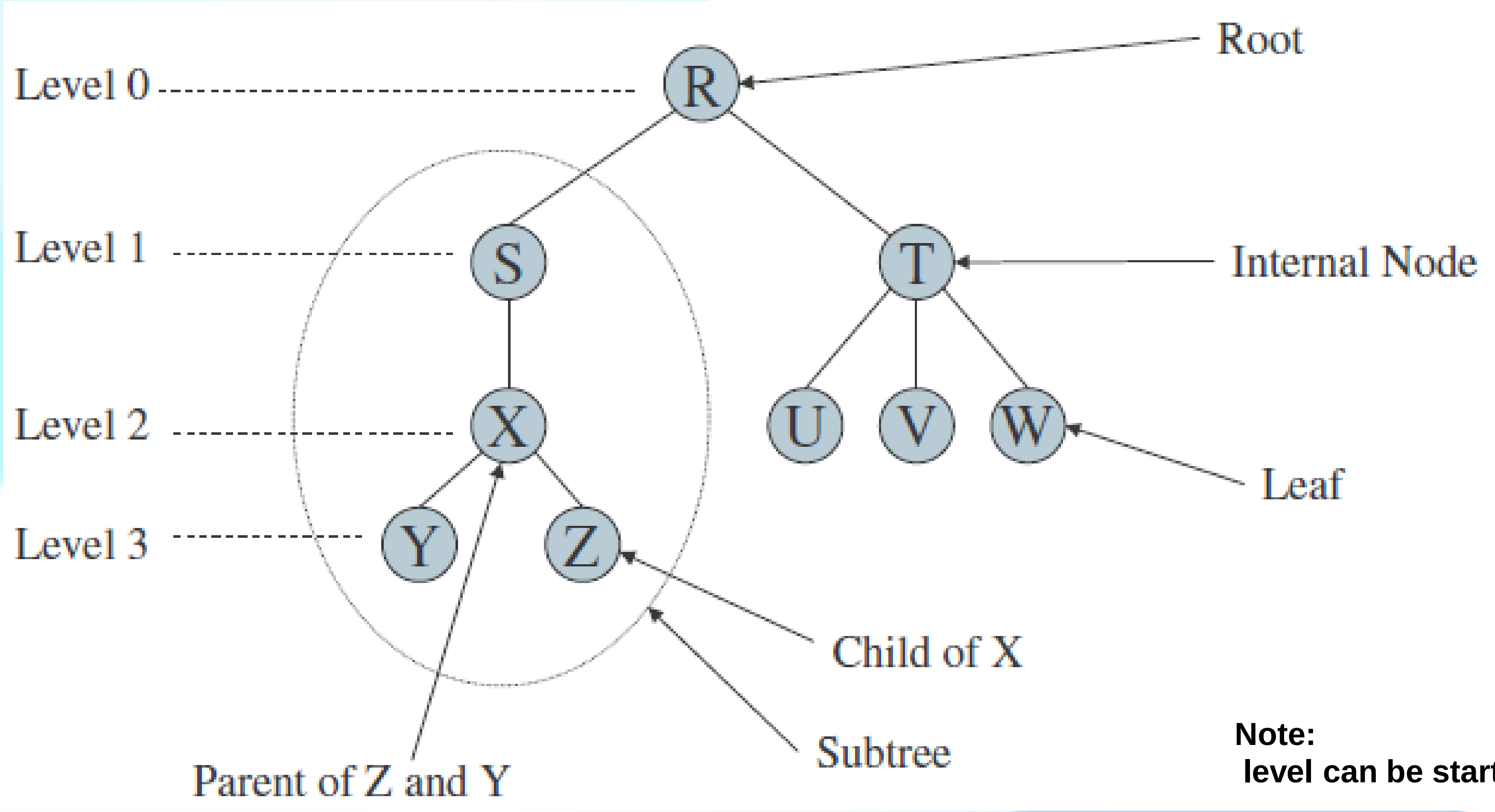
Path 1 : arjosari → blimbing → tlogomas → tlogomas swimming pool

Path 2 : arjosari → blimbing → tlogomas →jetis → sengkaling swimming pool

## Linear List vs Tree

- Linear list → to represent serial data.
  - Example : list od students, list of days, list of months etc.
- Tree → to represent hierarchical data (one to many).
  - Example : hierarchy of organizational structure, hierarchy of file system, etc.

## Tree Anatomy

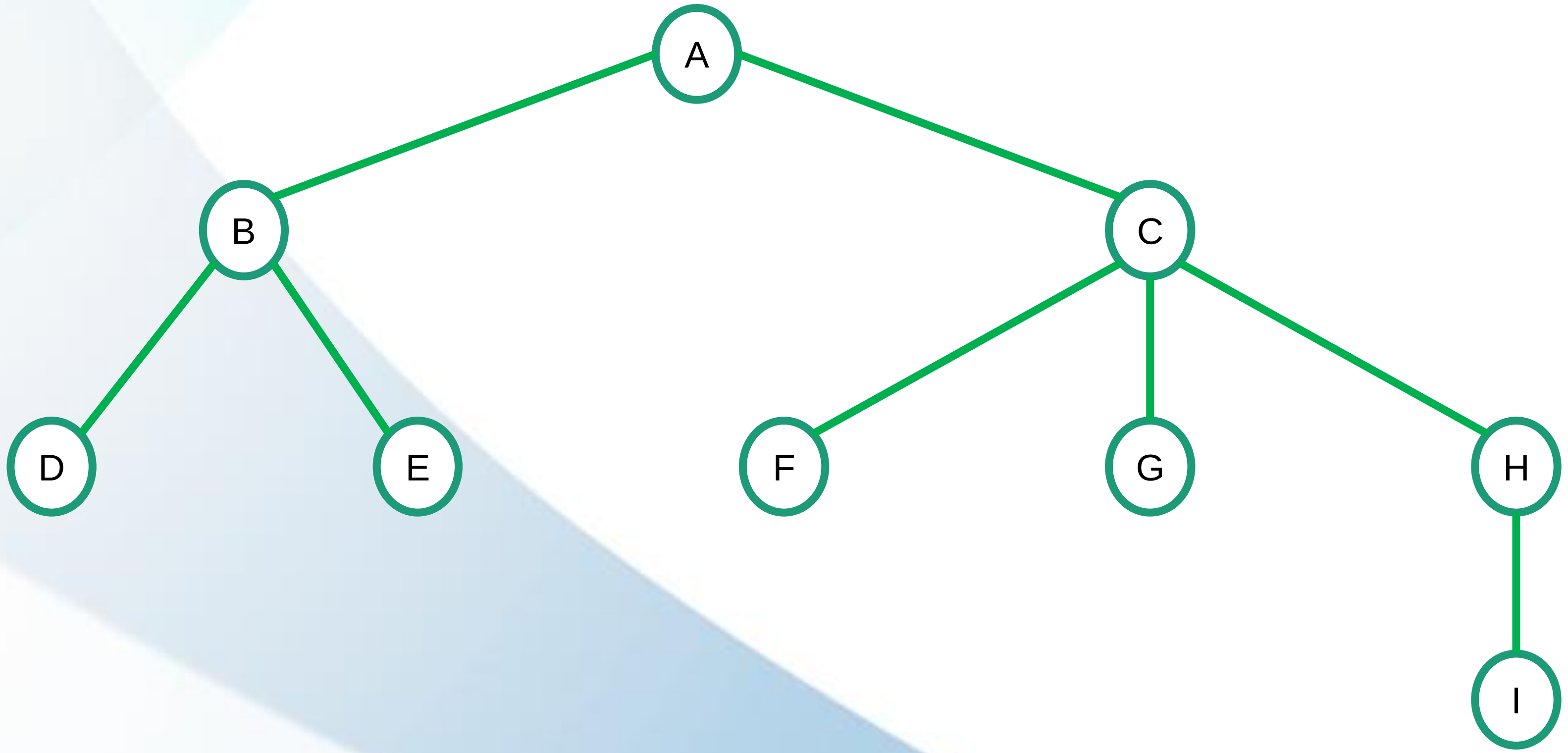


**Note:**
**level can be started from 0 or from 1.**

## Terminology in Tree

- **Node:** an element in a tree that contains information.
- **Predecessor**: nodes above certain node
- **Succesor**: node under a certain node.
- **Ancestor**: all nodes located before a certain node and located on the same path
- **Descendant**: All nodes located after a certain node and located on the same path.
- **Parent**: Predecessor that is one level above one node.
- **Child**: Successor that is one level below one node.
- **Sibling**:  nodes that have the same parent as one node.
- **Subtree**: The part of a tree (consist of a node and its descendants)
- **Size**: The number of nodes in a tree.
- **Height**: The number of levels in a tree.
- **Root**: A special node in a tree that has no predecessor.
- **Leaf**: Nodes in a tree that have no successor.

## Example: *Tree*



Exercise, which part (node) of the tree above, which is in accordance with the terminology in the t

## Answer

- **Node** : There are 9 nodes in the example tree , are node {A, B, C, D,E, F, G, H, I}
- **Predesesor** : node B is the predecessor of node {D, E}.
- **Succesor** : node {D, E} is the successor of node B
- **Ancestor**: node {C, A} is the ancestor of node F
- **Descendant**: node {D, E} is descendant of node B
- **Parent**: node C is the parent of node {F, G, H}
- **Child**: node {B, C} is child of node A
- **Sibling**: node {F, G, H} is sibling, has the same parent, node C.
- **Subtree**: there are 2 subtree, namely subtree from node {B, D, E} and from node {C, F, G, H, I}
- **Size**: in the tree there are 9 nodes.
- **Height**: the level of the tree is 4.
- **Root**: node A is the root of the illustration tree.
- **Leaf:** the leaf node is {D, E, F, G, I}.
- **Degree**: node B has degrees 2 {D, E} and node C has degrees 3 {F, G, H}

## *About Tree*

- Each node, except root, has ONLY one parent.
- Subtree is a collection of children from a node, which forms a smaller tree structure

## Binary Tree

## ***Binary*** *Tree*

- Special case of **tree**.
- In binary tree, each node has **no more** than 2 childs/subtrees/degrees.
- They are called left-child and right-child.

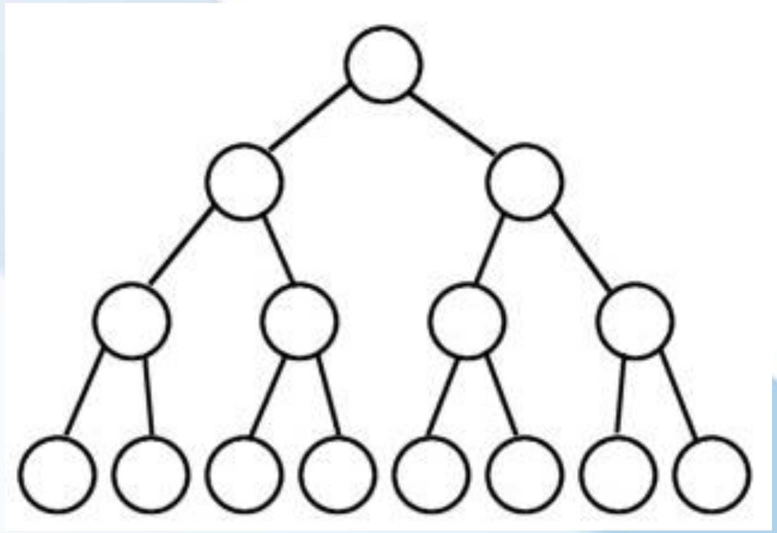
## Type of Binary Tree

There are several types of binary trees based on the structure of the nodes in it, including:

- *Full Binary Tree*
- *Strict Binary Tree*
- *Complete Binary Tree*
- *Incomplete Binary Tree*
- *Skewed Binary Tree*

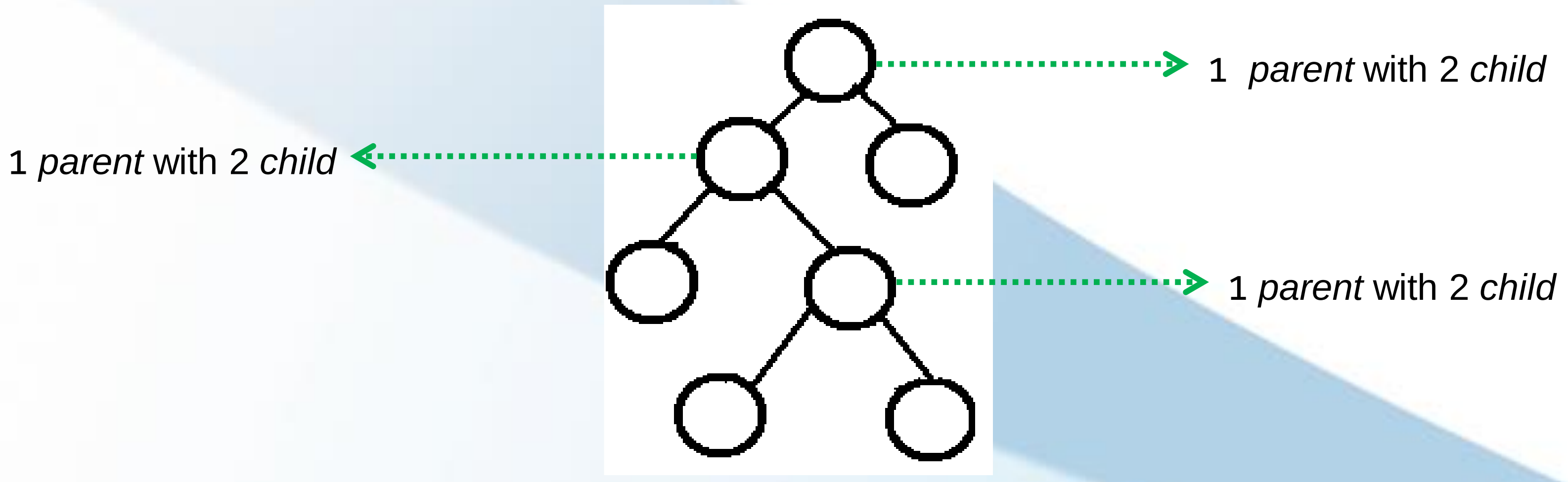
## *Full Binary Tree*

- All nodes (except leaves) have 2 children and each branch has the same segment length. Or, each subtree has the same path length
- Also called **maximum binary tree** or **perfect binary tree**.



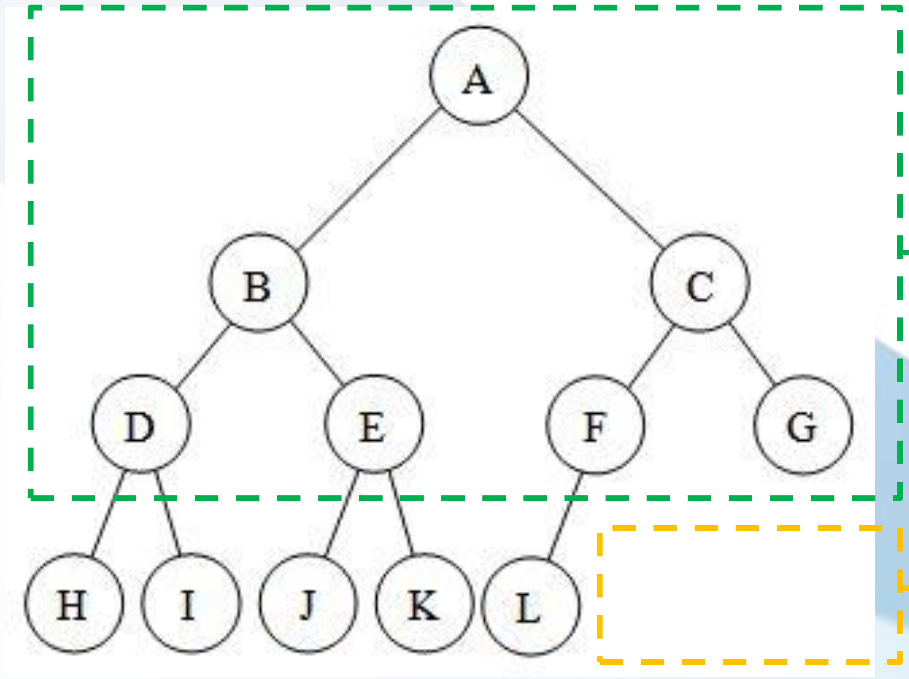
## *Strict Binary Tree*

- Each parent node **MUST** have **2 children, no parent node** ONLY has **one child**.
- Each subtree is not required to have the same path length as the other subtree as in the full binary tree.


1 *parent* with 2 *child*
1 *parent* with 2 *child*
1 *parent* with 2 *child*

## *Complete Binary Tree*

- A tree which has **complete nodes at all levels**, **except** for the **last level**.
- At the last level, nodes **can be filled incompletely** and it must be filled from **from left to right.**

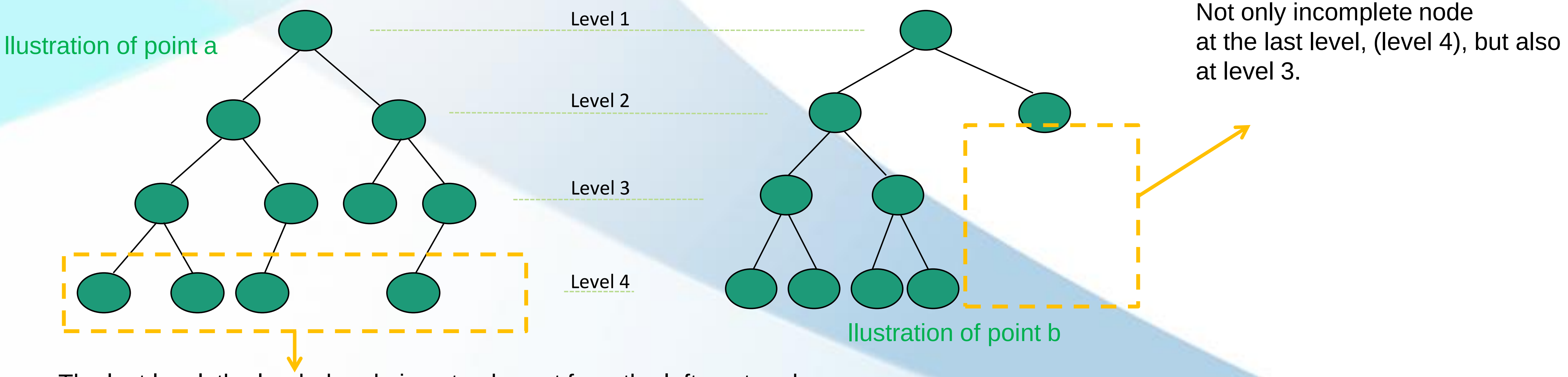


Nodes at all levels are full filled, except the last level

Fill the node at the last level Must be from left to right

# *Incomplete Binary Tree*

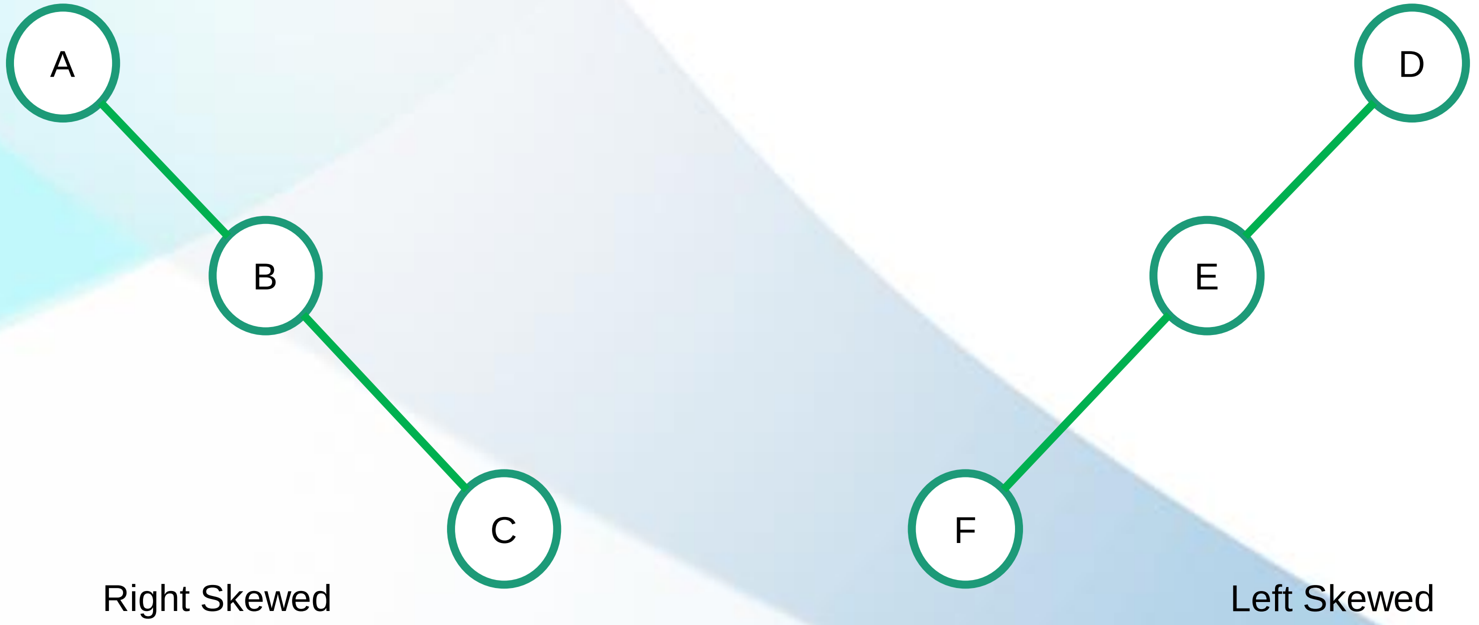
a. All nodes at the last level are filled in unevenly from left to right first.
b. There are empty nodes in the tree in addition to the last level position.


Ilustration of point a
Level 1
Level 2
Level 3
Level 4
Not only incomplete node at the last level, (level 4), but also at level 3.
Ilustration of point b

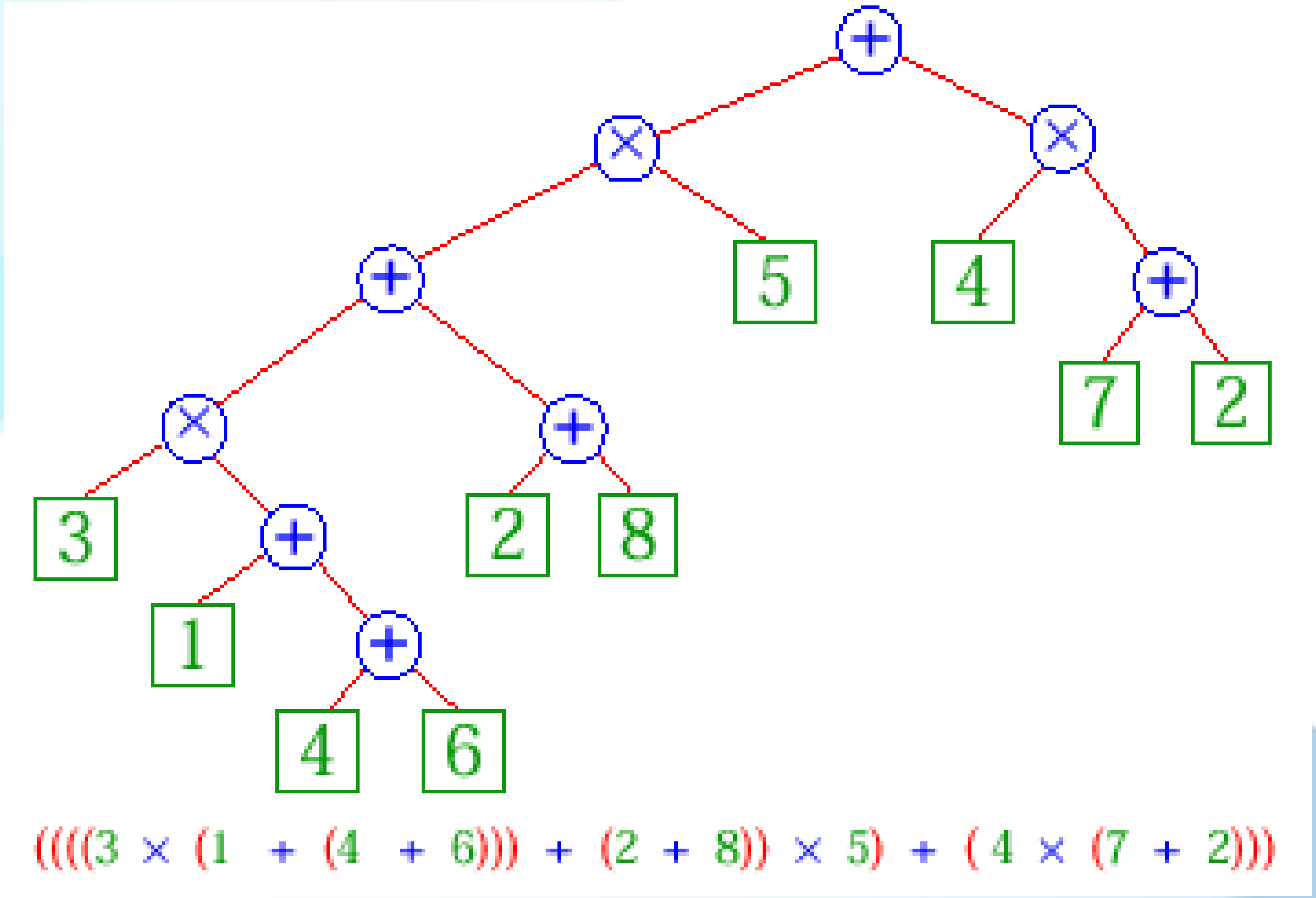
The last level, the loaded node is not coherent from the leftmost node.

## *Skewed Binary Tree*

- *A binary tree where all nodes (except leaves) have only one child.*
- *Also called a binary tree minimum.*



## Example of a Binary Tree



## Exercise

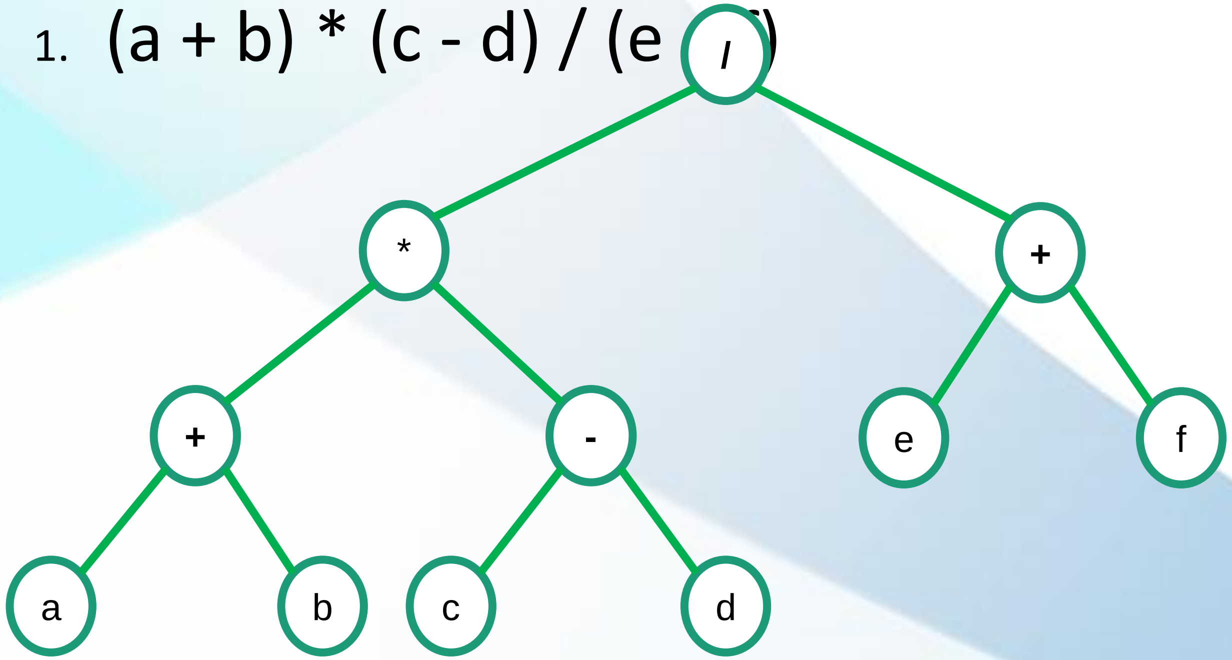
Create a binary tree from the following arithmetic expression:

1. (a + b) * (c - d) / (e + f)
2. (a + b) * ((b - c) + d)

## Exercise – Answer 1

Create a binary tree from the following arithmetic expression:

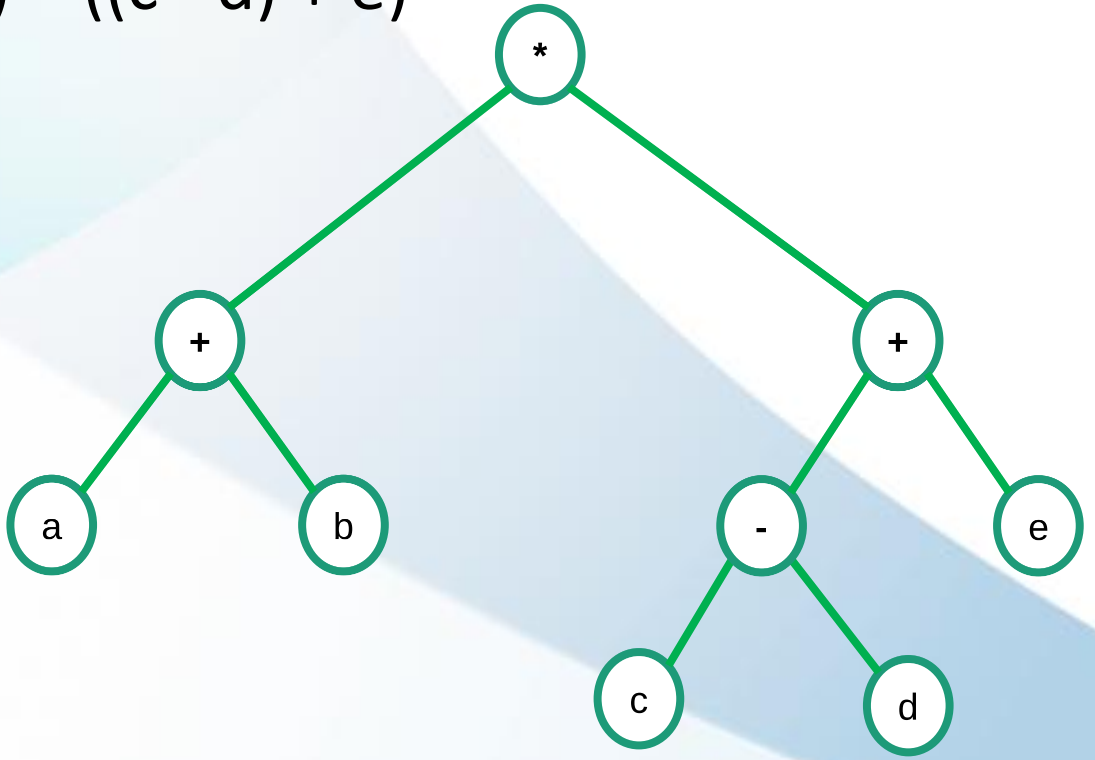
1. (a + b) * (c - d) / (e )



## Exercise – Answer 2

Create a binary tree from the following arithmetic expression:

2. (a + b) * ((c - d) + e)



## Exercise Task 1

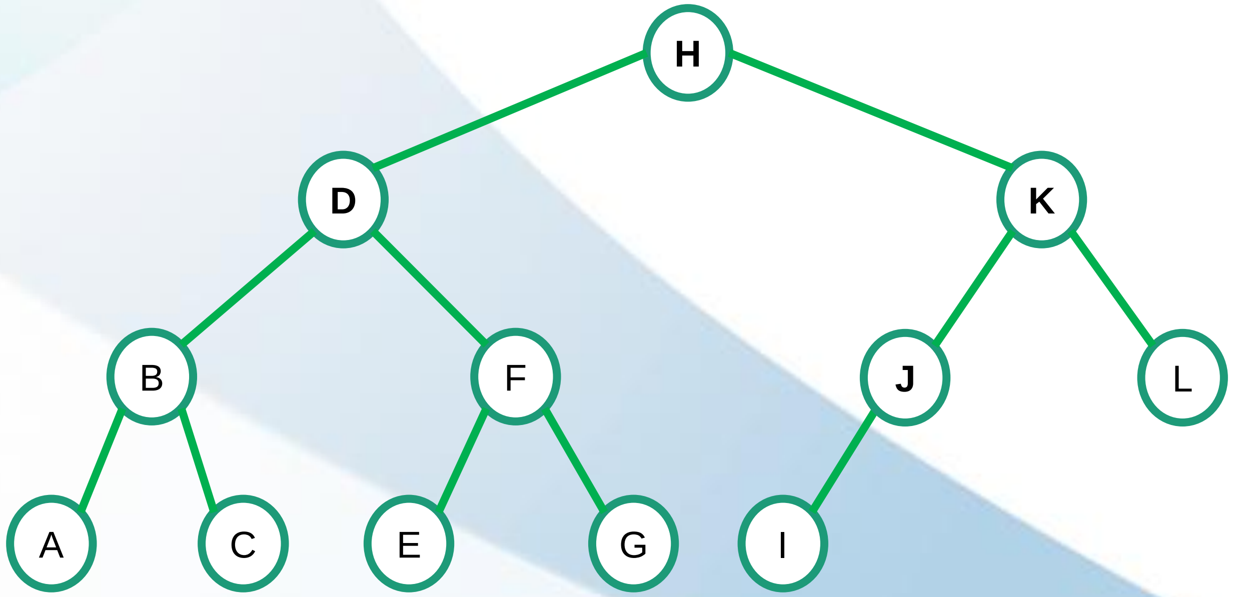
Create a binary tree from the following arithmetic expression:

1. a * (b + c) / (e + (f – g))
2. ((a * b) * c) + (d / e) * f

## Binary Tree Representation

# Binary Tree Representation

- Binary Tree can be represented by using an **array** or **double linked list**.



## Tree Representation with **Array**

The position of the node can be determined based on the following formula:

- The root assumption starts at index-0:
  - `The left child of node i is at index: 2 * i + 1`
  - `Right child of node i is at index: 2 * i + 2`
- The root assumption starts at index 1:
  - The left child of node i is at index: $2 * i$
  - Right child of node i is in index: $2 * i + 1$

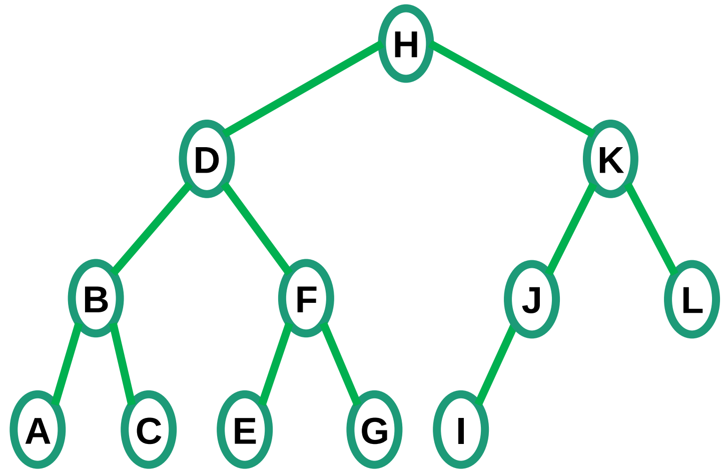
# Tree Representation with Array (cont.)

- The root assumption starts at index-0:

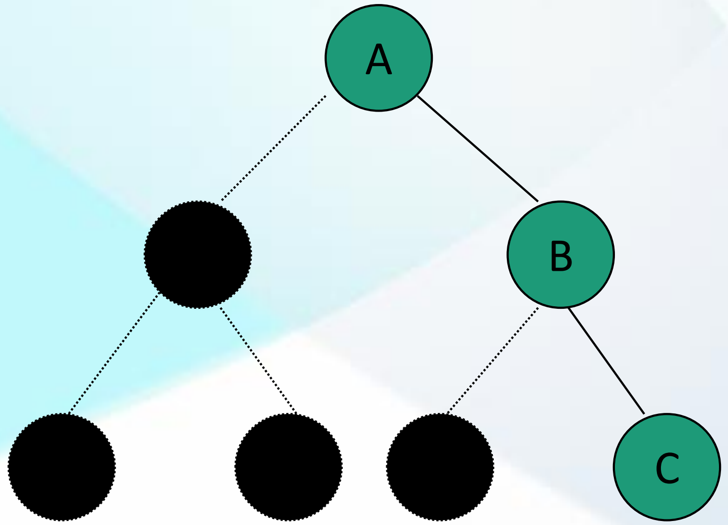
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| H | D | K | B | F | J | L | A | C | E | G | I |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

- The root assumption starts at index 1:

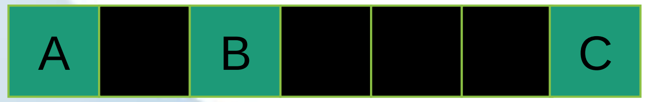
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | H | D | K | B | F | J | L | A | C | E | G | I |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |



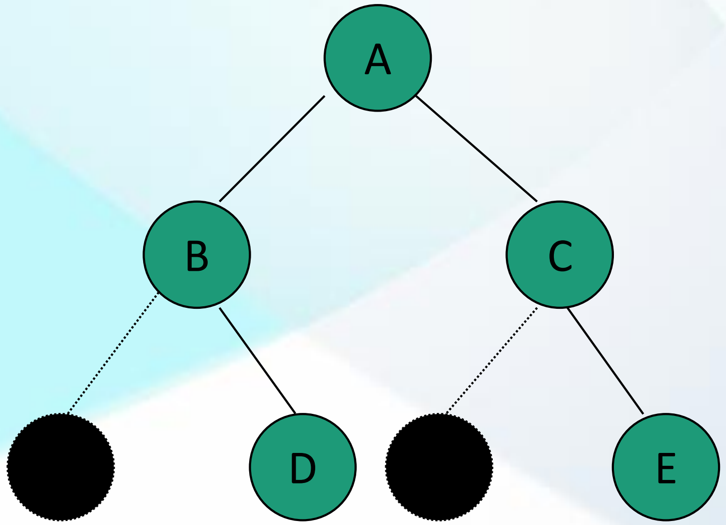
## Tree Representation with Array (cont.)



- The binary tree aside has 3 nodes {A, B, C}
- Black nodes represent missing elements



## Tree Representation with Array (cont.)

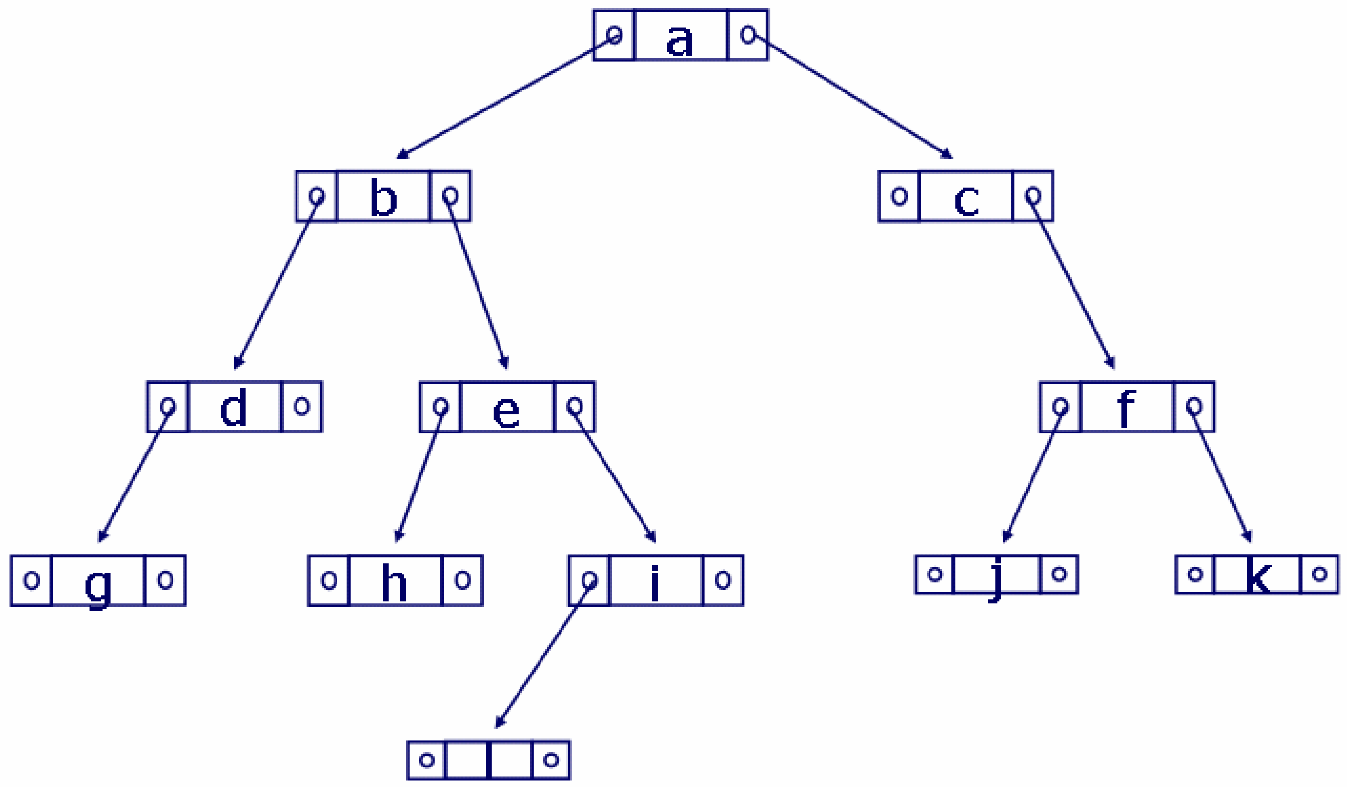


- The binary tree aside has 5 nodes {A, B, C, D, E}
- Black nodes represent missing elements

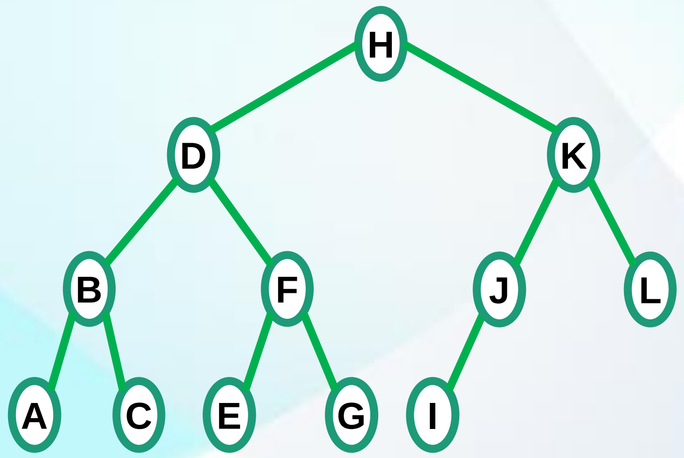
| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | ■ | D | ■ | E |

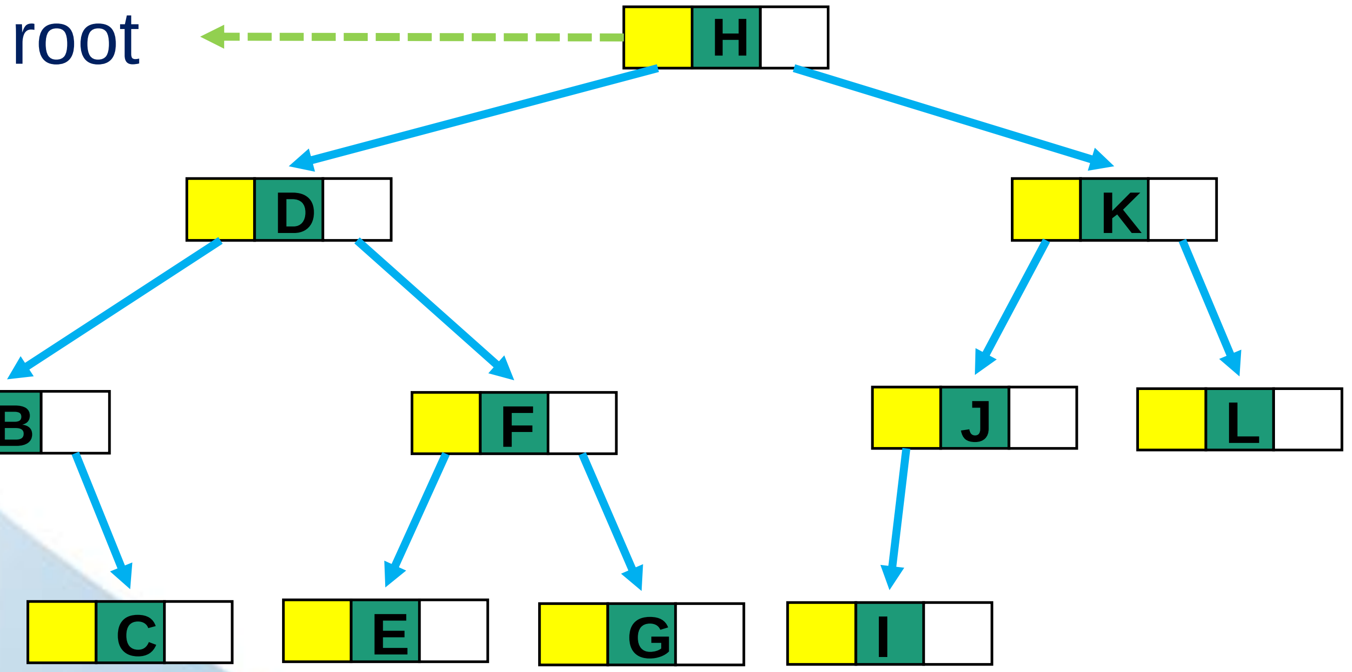
## Tree Representation with ***Linked List***

- The binary tree implementation can be done using a data linked list structure; each node consists of 3 parts, i.e
  1. Left Pointer
  2. Data / element info
  3. Right Pointer



# Tree Representation with ***Linked List*** *(cont.)*

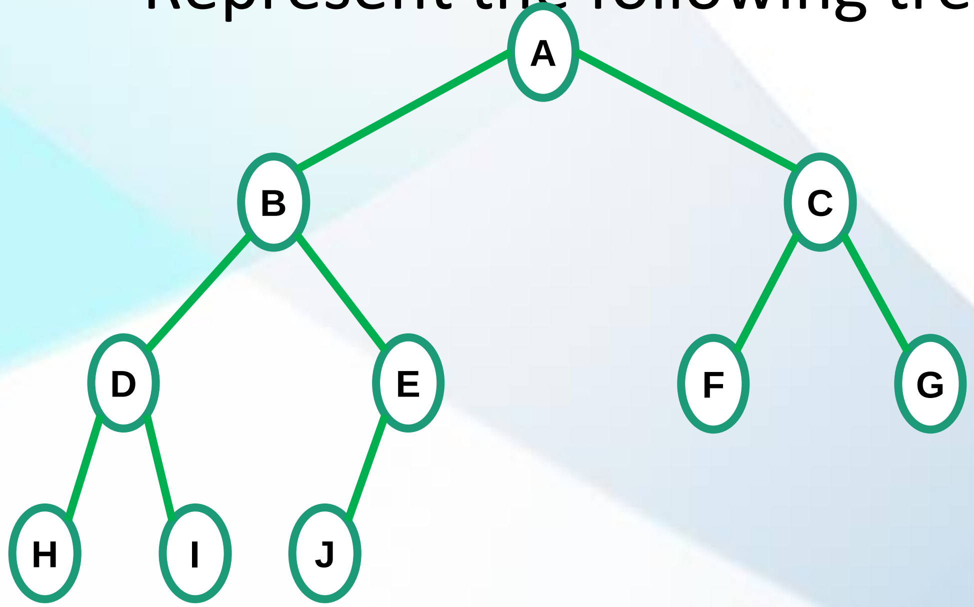




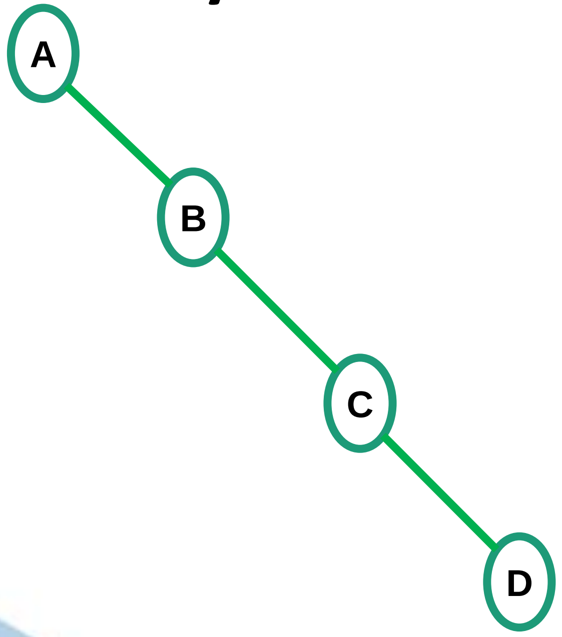
🟨 leftChild 🟩 element ⬜ rightChild

## Exercise Task 2

- Represent the following tree with illustration **array** and **linked lists.**



Exercise 1



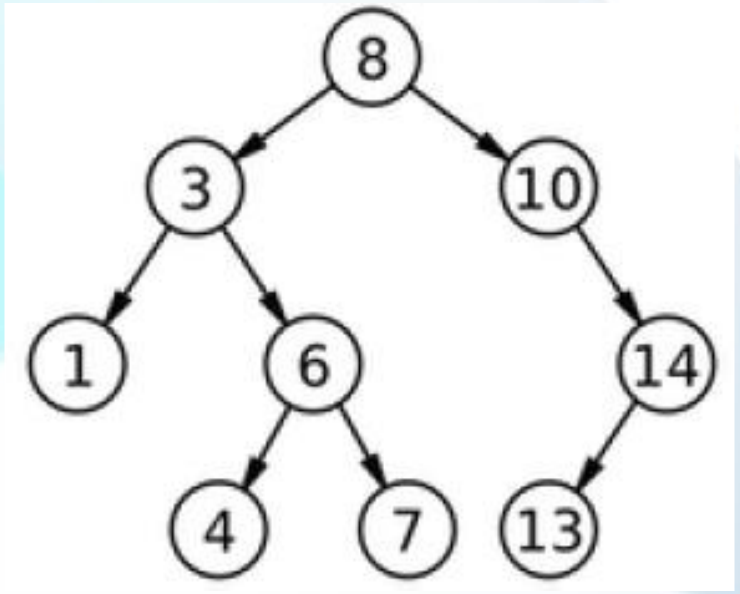
Exercise 2

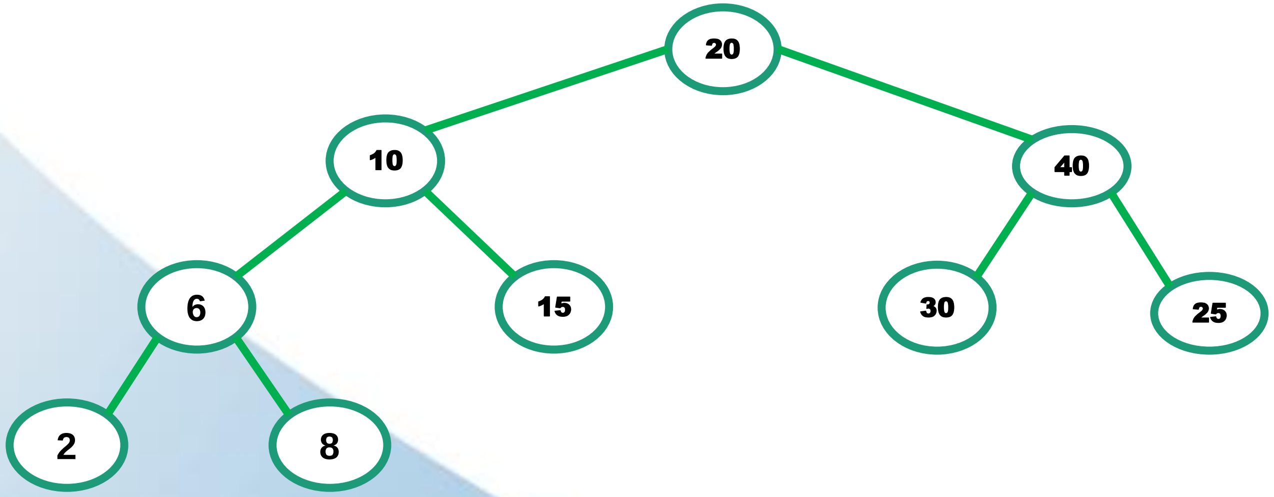
## Binary Search Tree

## *Binary Search Tree* (BST)

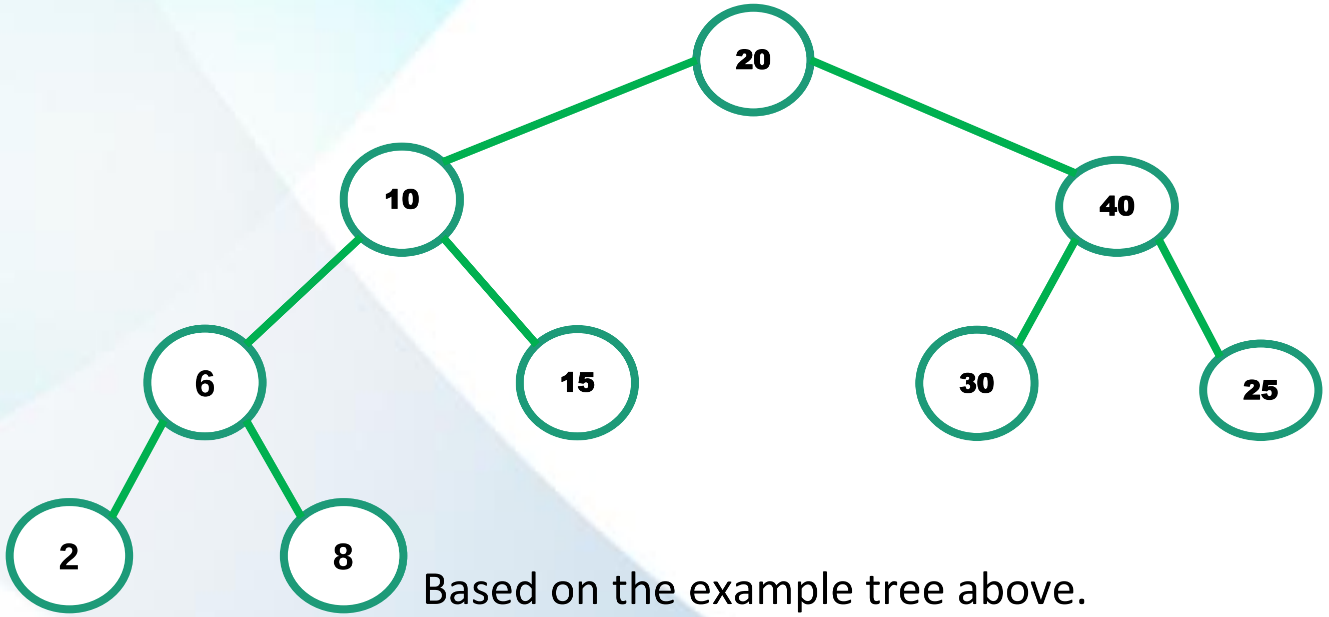
- Binary Search Tree is a **special case** of binary tree with the characteristic that all **left-childs** must be smaller than the **right-childs** and its parent.
- Binary search tree is made to overcome **weaknesses** in ordinary binary tree → it is hard to search data/node in the binary tree, that is managed randomly (unordered binary tree)
- Also called an **ordered Binary tree**, which is a binary tree that **all children** from each **node are sorted**.

## Example of a Binary Search Tree





## Question



Based on the example tree above.

- What if there are additional nodes with a value of 13?
- What if there are additional nodes with a value of 50?
- What if there are additional nodes with a value of 27?

***Answer:*** *By tracing each node element in the tree before adding it by traversal.*

***Binary Tree Traversal***

## Binary Tree Traversal

## Binary Tree Traversal

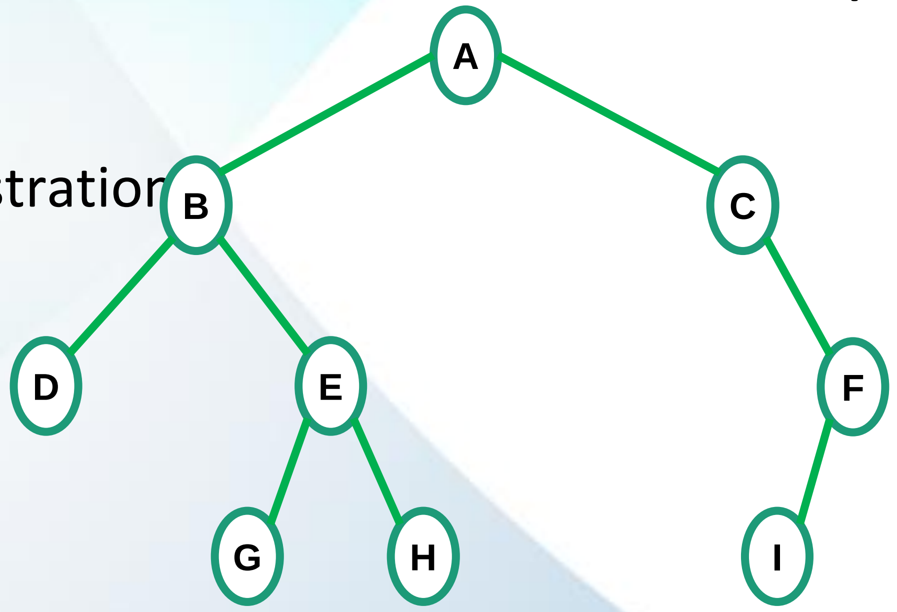
- A method of tracing all nodes in the binary tree.
- There are several methods, including:
  - Preoder
  - Inorder
  - Postorder
  - Level order

## *Binary Tree Traversal - Preorder*

- *Preorder Traversal Steps:*
  1. Visit and print data on root
  2. Recursively visit and print all data in the **left** subtree starting from the **left-child.**
  3. Recursively visit and print all data in the **right** subtree starting from the **left-child.**

# *Binary Tree Traversal - Preorder (cont.)*

- Example illustration



Preorder Traversal → A, B, D, E, G, H, C, F, I

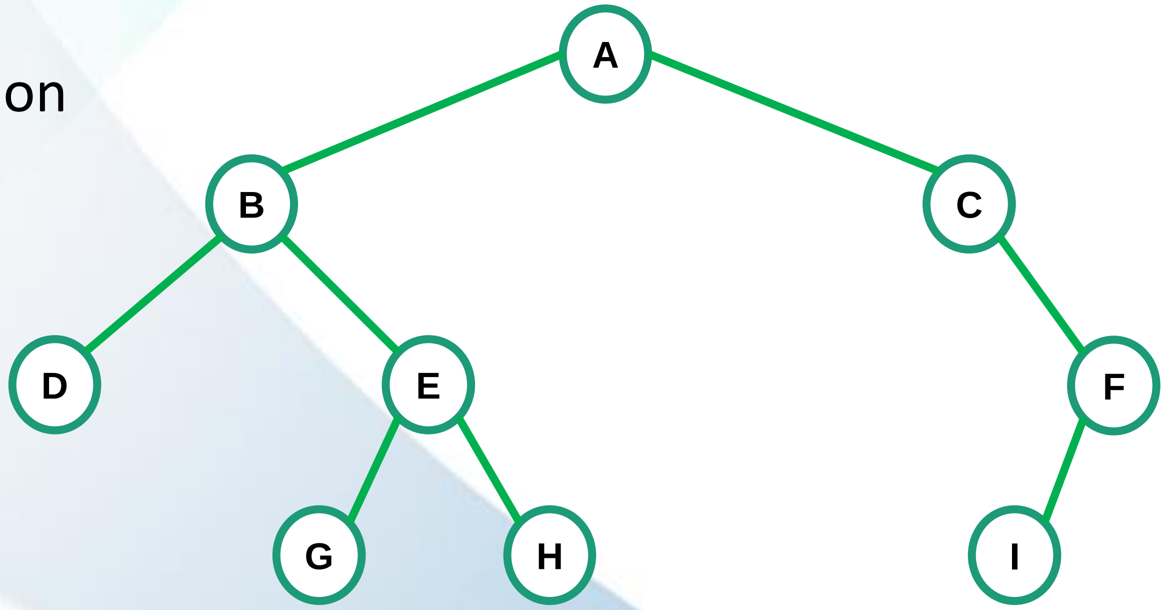
Video illustration: *https://youtu.be/1WxLM2hwL-U*

## *Binary Tree Traversal - Inorder*

- *Inorder Traversal Steps:*
  1. Recursively visit and print all data in the **left** subtree.
  2. Visit and print data on root
  3. Recursively visit and print all data in the **right** subtree.

## Binary Tree Traversal - Inorder (cont.)

- Example illustration



Inorder Traversal → D, B, G, E, H, A, C, I, F

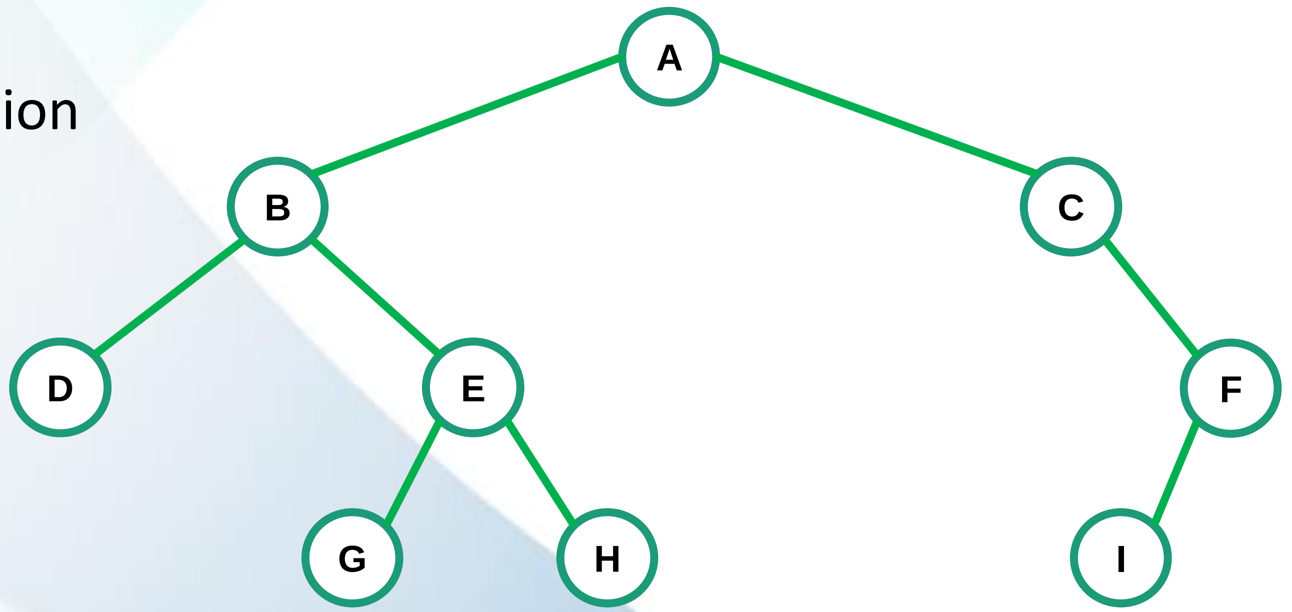
Video illustration : *https://youtu.be/5dySuyZf9Qg*

## *Binary Tree Traversal - Postorder*

- *Postorder Traversal Steps:*
  1. Recursively visit and print all data in the **left** subtree.
  2. Recursively visit and print all data in the **right** subtree.
  3. Visit and print data on root

## Binary Tree Traversal - Postorder (cont.)

- Example illustration



Postorder Traversal → D, G, H, E, B, I, F, C, A

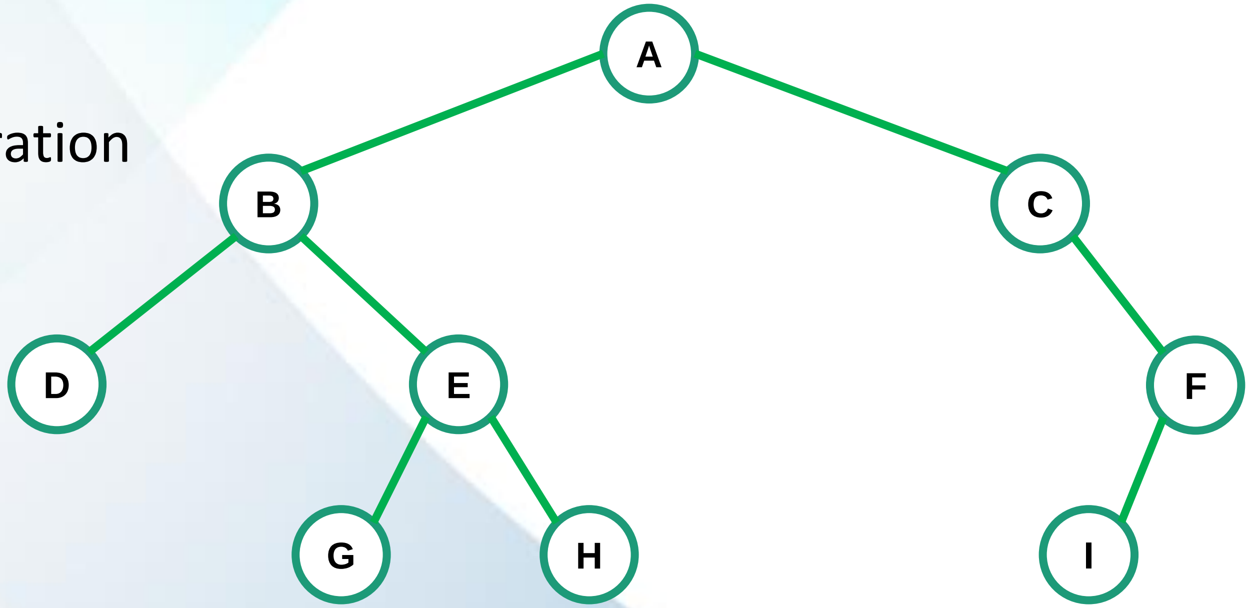
Video illustration : [https://youtu.be/4zVdfkpcT6U](https://youtu.be/4zVdfkpcT6U)

## *Binary Tree Traversal – Level order*

- The order level is also called the Breath First Order
- Level order Traversal Steps:
  1. Visit and print data on root
  2. Recursively visit the nodes under root from the left to the right
  3. Go back to 2nd point to the lowest level.

## Binary Tree Traversal – Level order (cont.)

- Example illustration

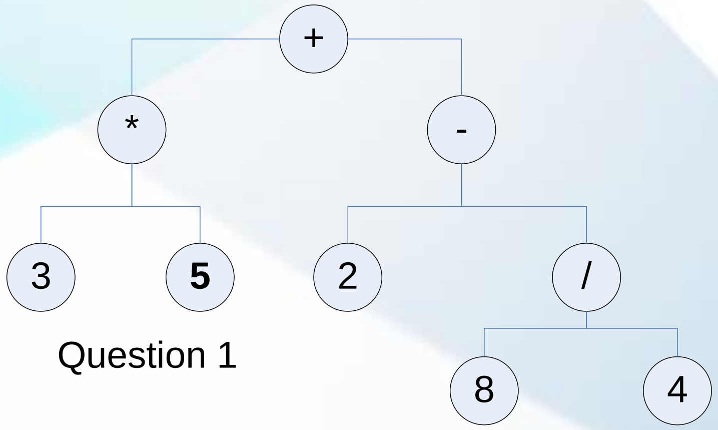


Level order Traversal → A, B, C, D, E, F, G, H, I

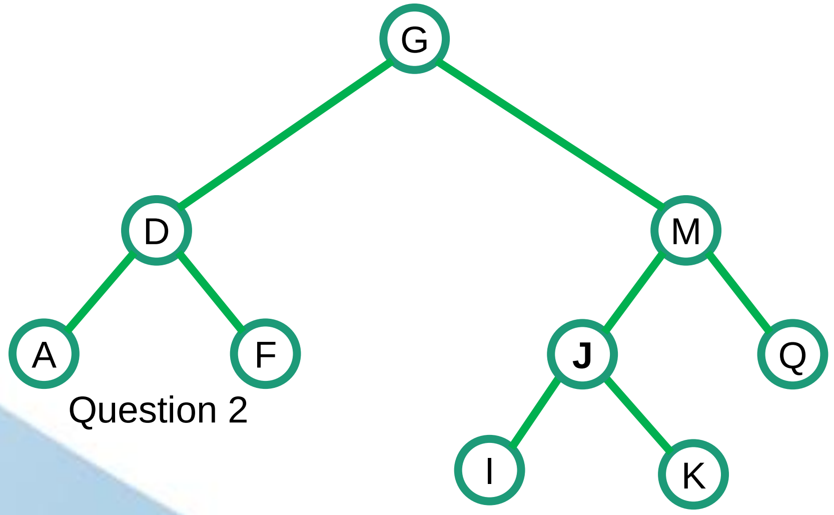
Video illustration: [https://youtu.be/IozGo2kwRYE](https://youtu.be/IozGo2kwRYE)

## Assignment Exercise 3

- Browse the following binary trees using the preorder, inorder, postorder, and order traversal level methods.



Question 1



Question 2

## Operations on the Binary Search Tree

## Operations on BST

In BST there are several operations that are used to manage binary trees, including:

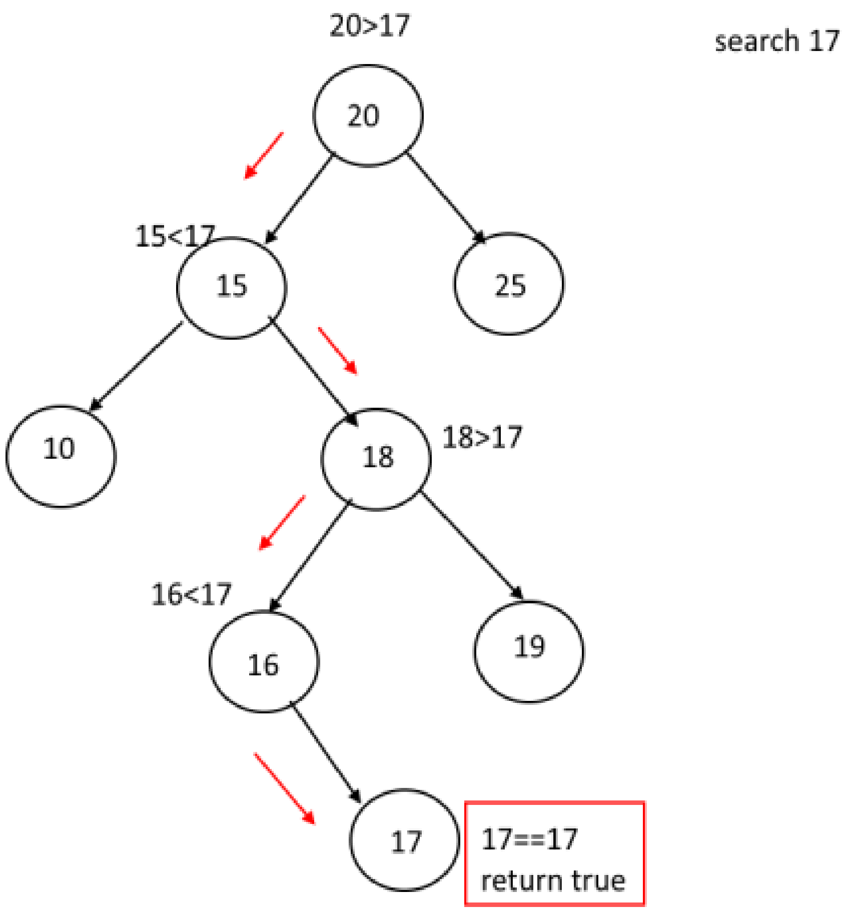
- **Find**: data search operation in the same tree as n (the value to be searched for).
- **Insert**: Operation of adding data / nodes to a tree
- **Delete**: The operation of deleting data / nodes in a tree
- **Display**: Operation display data

## Operation - Find

1. Start from root, and compare the root value with n (the key).
2. If the value of n is smaller than root, do a search on the left subtree of root, and ignore the remaining right subtree
3. If the value of n is greater than root, search the right subtree of the root, and ignore the remaining right subtree
4. Search nodes under root in the same way as step in number 2 & 3.
5. If found, return true.
6. If at the end of the node (leaf) is not found, return false.

## Operation - Find*(cont.)*

- **Ilustration**



Video ilustration *Find/search element* : [*https://youtu.be/a7xOXL7hn94*](https://youtu.be/a7xOXL7hn94)

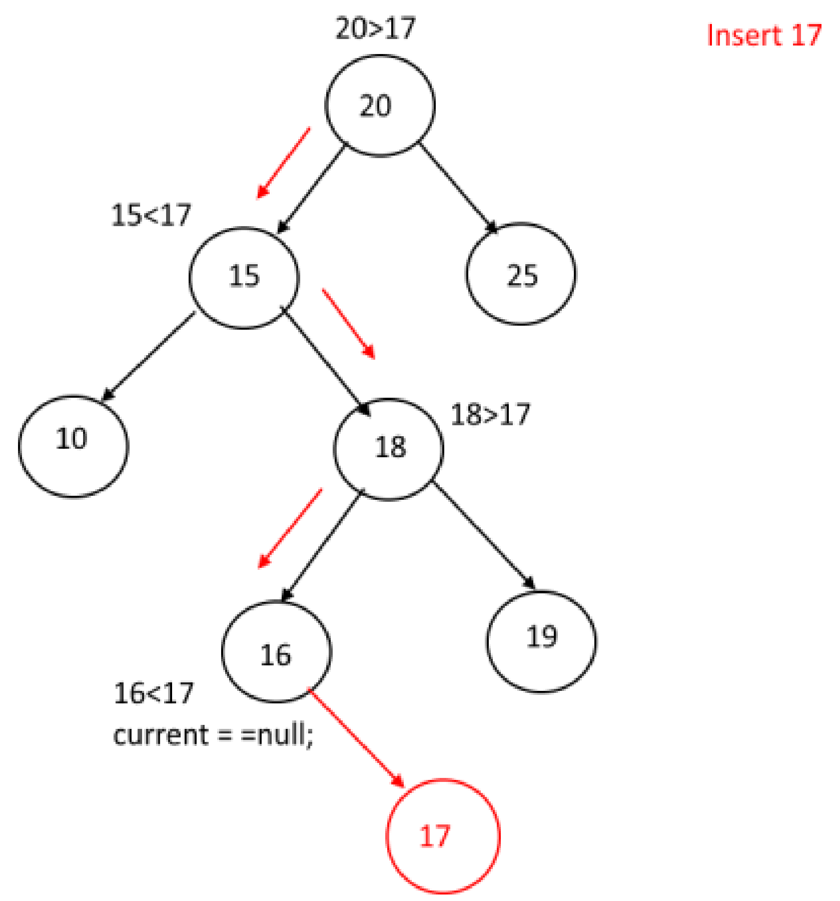
## Operation – *Insert*

The insert operation is similar to the find operation. To insert a node, we must first, find the right place to put the node. The steps are:

1. Create a temporary variable named **current** node, initialize **current** with root.
2. Compare the **current** value with n (the new data to be entered).
3. If the **current** value is greater than n, look for a place to the left subtree.
4. If the **current** value is smaller than n, look for a place to the right subtree.
5. If you find the **current** null value, which means that the search has reached the leaf (end of the node), place the new node containing the n into the **current** position.

## Operation – *Insert (cont.)*

- **Ilustration**



Video ilustration *Insert element* : [https://youtu.be/WoRa9vnHkDM](https://youtu.be/WoRa9vnHkDM)

## Operation – *Delete*

Possible deletion scenario that can be done:

1. The node to be deleted is the leaf node (no children).
2. The node to be deleted only has one child node.
3. The node to be deleted has two child nodes.

Video ilustration *Delete element* : *https://youtu.be/c1zKNiABiHk*

## Operation – *Delete* (scenario 1)

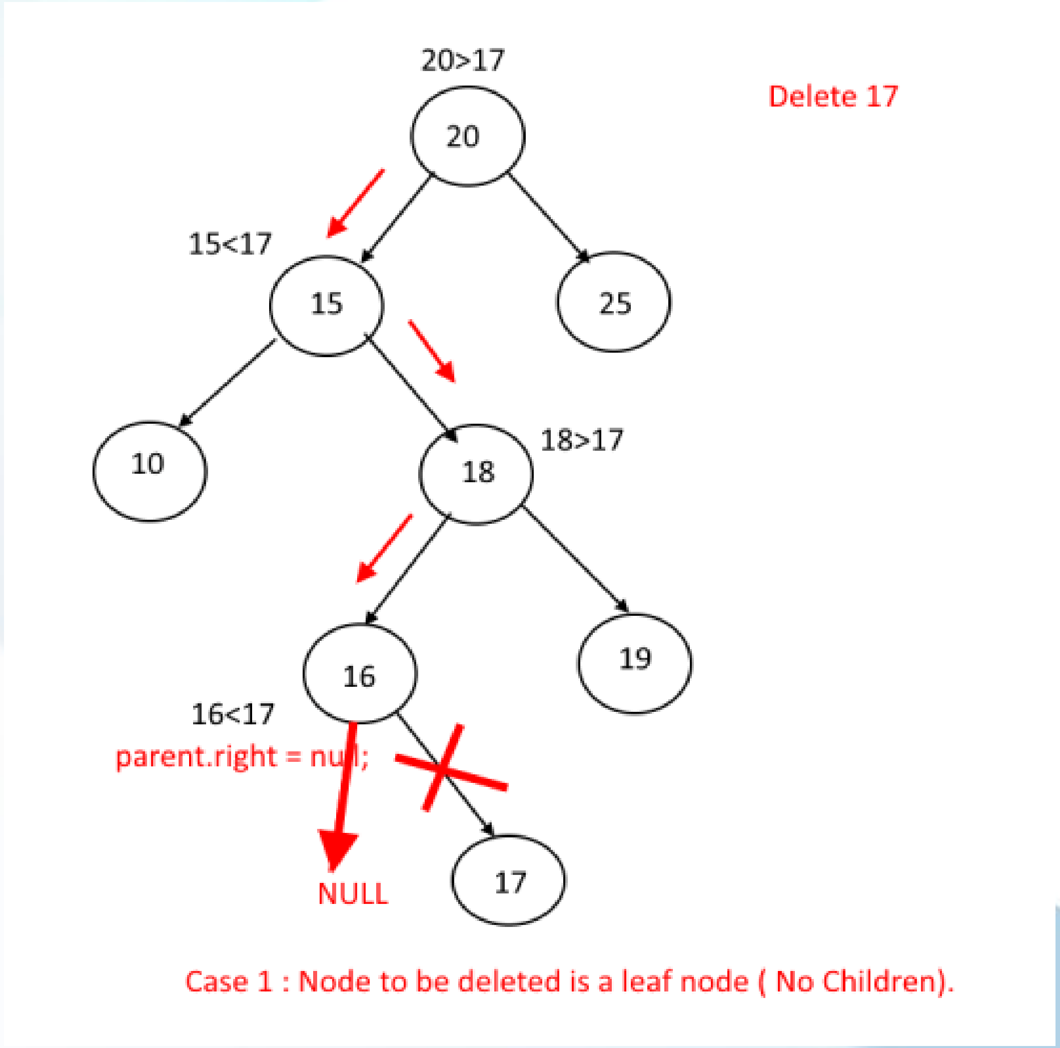
**The node to be deleted is the leaf node (no children).**

Steps:

1. Traverse the tree by comparing each node visited.
2. If the value of n (the value of the node to be deleted) is smaller than value of visited node, then continue traversing to the left.
3. If the value of n (the value of the node to be deleted) is bigger than value of visited node, then continue traversing to the left, continue traversing to the right.
4. If you find it, set the pointer
   - left = null → if the value of n (the node to be deleted or child) is smaller than the parent (node n is leftchild of its parent), or
   - rigth = null → if the value of n (the node to be deleted or child) is greater than the parent (node n is rigthchild of its parent).

## Operation – *Delete* (scenario 1) (cont.)

**Ilustration**



## Operation – *Delete* (scenario 2)

**The node to be deleted has one child.**

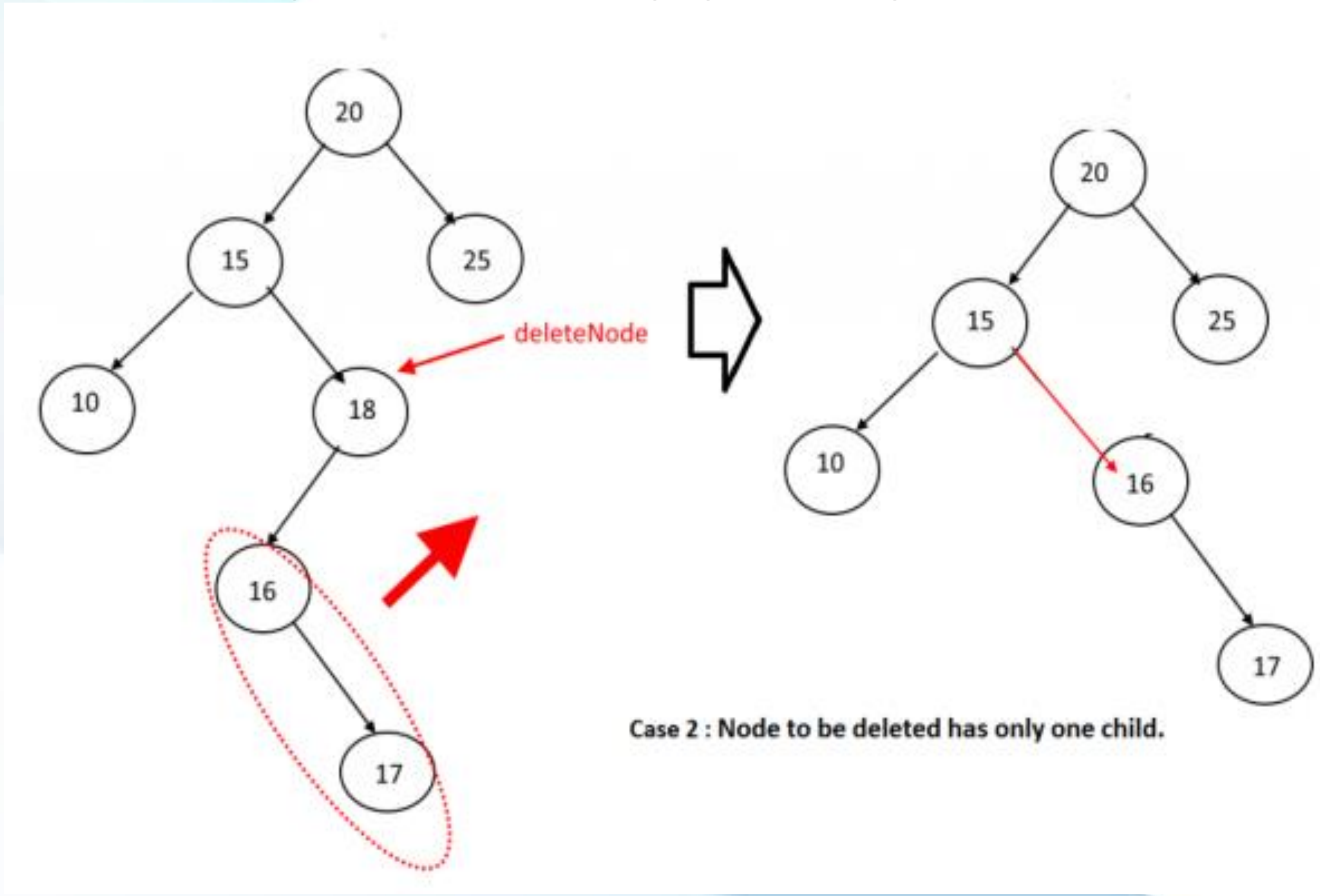
Steps:

1. Traverse the node to be deleted.
2. After finding the node to be deleted, **note** the parent of that node, and in which position that node to the parent (rightchild or leftchild).
3. From that node, check which side is null (because that node only has one child).
4. For example the node to be deleted has a child on the left. Then set the child of that node (along with its sub tree) and place it on the parent side to replace the position of that node to be deleted earlier.

JURUSAN TEKNOLOGI INFORMASI POLITEKNIK NEGERI MALANG

## Operation – *Delete* (scenario 2) (cont.)

**Ilustration**



Case 2 : Node to be deleted has only one child.

## Operation – *Delete* (scenario 3)

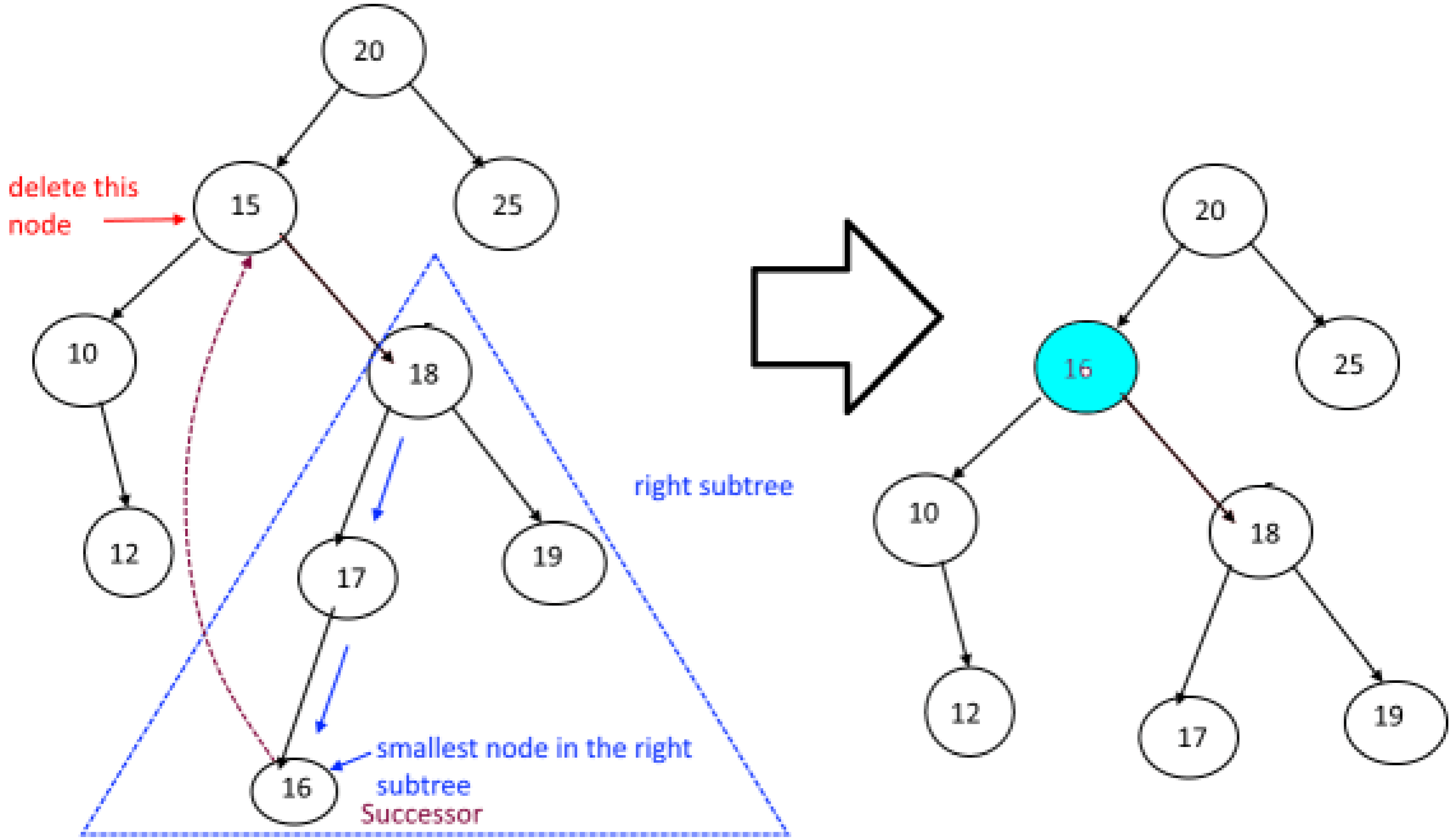
**The node to be deleted has two children.**

Steps:

1. Look for the successor of the node to be deleted, to replace the node to be deleted.
2. Successor is the smallest node from the right subtree on the node to be deleted.

## Operation – *Delete* (scenario 3) (cont.)

**Ilustration**

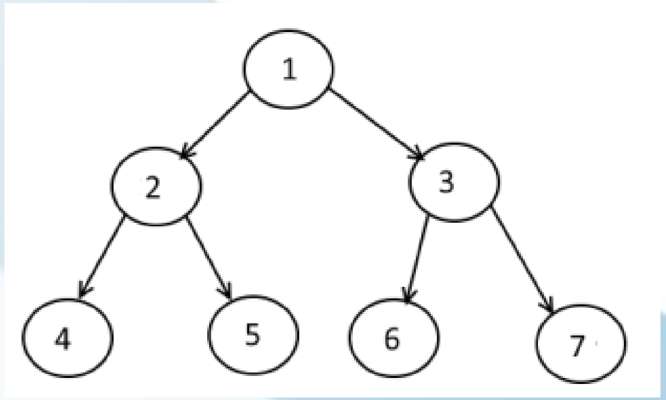


## Operation – *Display*

Prints all the nodes in the tree. The process of printing data uses the traversal method that has been studied, including the method

- *Preorder*
- *Inorder*
- *Postorder*
- *Level order / BFS*

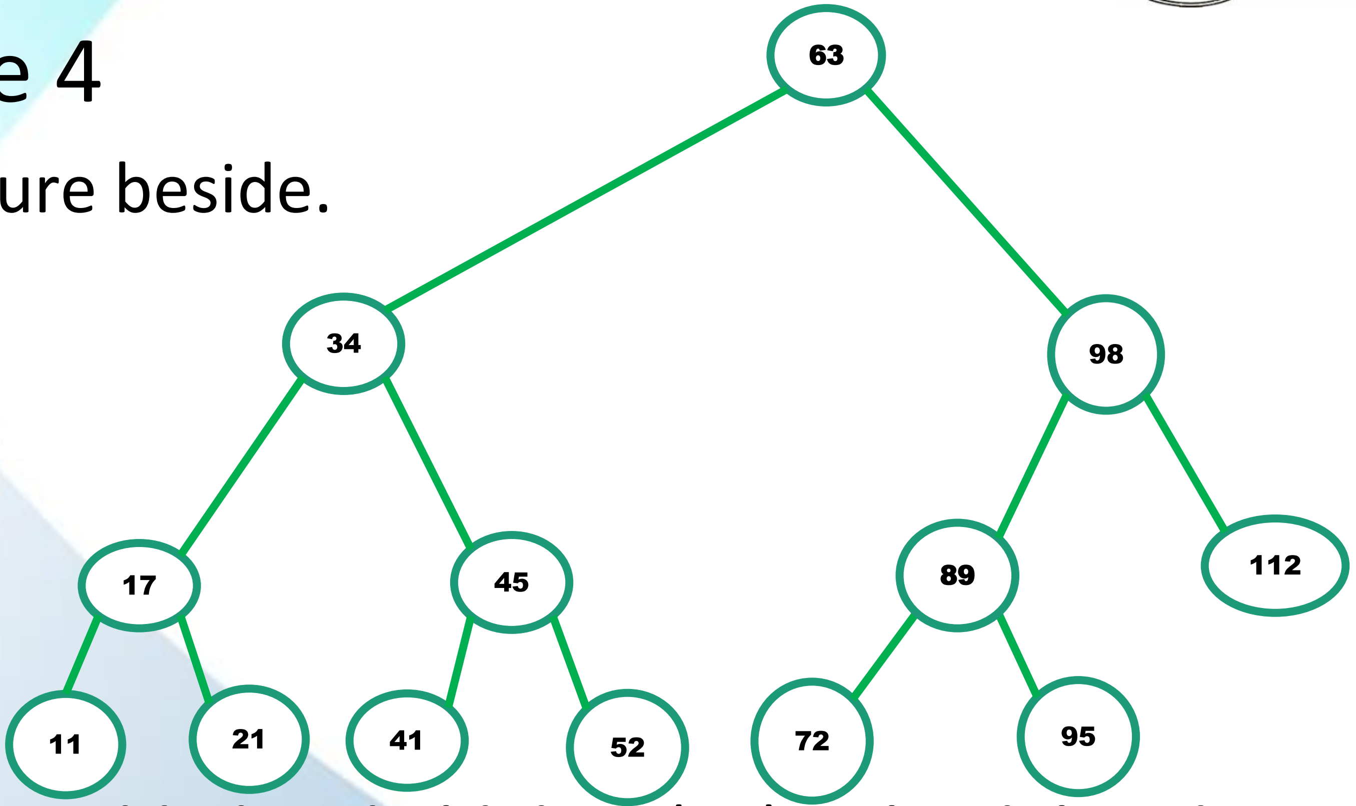
```
Inorder :     4, 2, 5, 1, 6, 3, 7
Preorder:     1, 2, 4, 5, 3, 6, 7
Postorder:    4, 5, 2, 6, 7, 3, 1
Level order:  1, 2, 3, 4, 5, 6, 7
```



For the illustration video the Display element follows the Binary Tree Traverse illustration video

## Assignment Exercise 4

There is a tree like the picture beside.



There is new data (40) to be added and old data (98) to be deleted.

Illustrate the operations (find, insert, delete, display) that will be performed to overcome the addition and deletion of that data.

Thank you ☺