INFORMATIKA UMN

UMN
UNIVERSITAS MULTIMEDIA NUSANTARA

# IF232
# ALGORITHMS & DATA STRUCTURES

08
TREES

DENNIS GUNAWAN

# REVIEW

**Queues:**

Array Representation of Queues

Operations on Queues

Linked Representation of Queues

Operations on Linked Queues

Applications of Queues

# OUTLINE

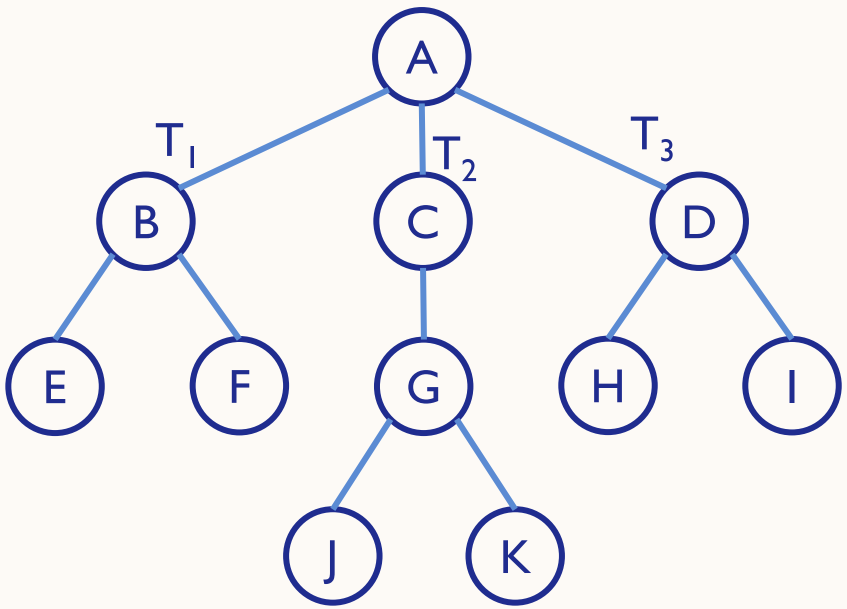
Basic Terminology

Types of Trees

Traversing a Binary Tree

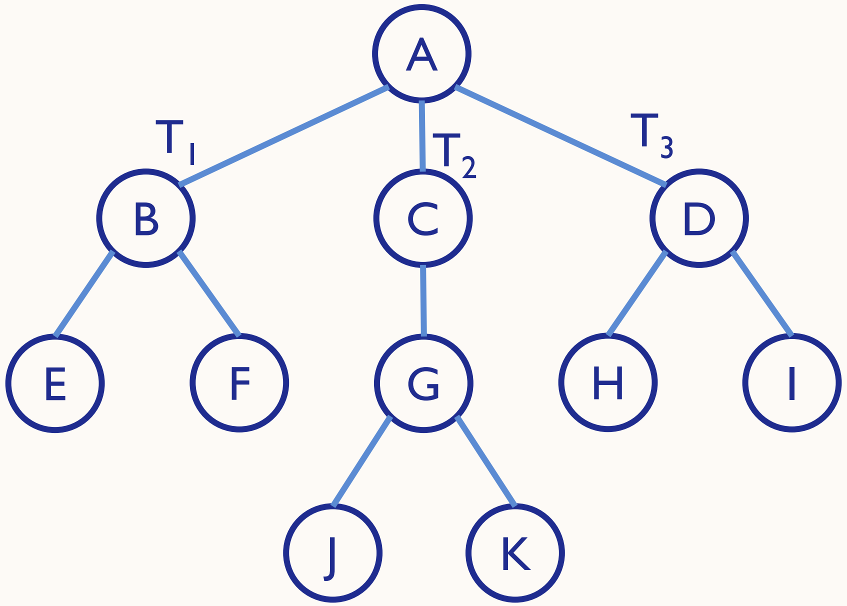
Applications of Trees

# 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**
- Node **A** is the **root node**
- Nodes **B**, **C**, and **D** are **children of the root node** and **form sub-trees** of the tree rooted at node A

# BASIC TERMINOLOGY




**Root node**

- The root node R is the **topmost node** in the tree
- If **R = NULL**, then it means the tree is empty

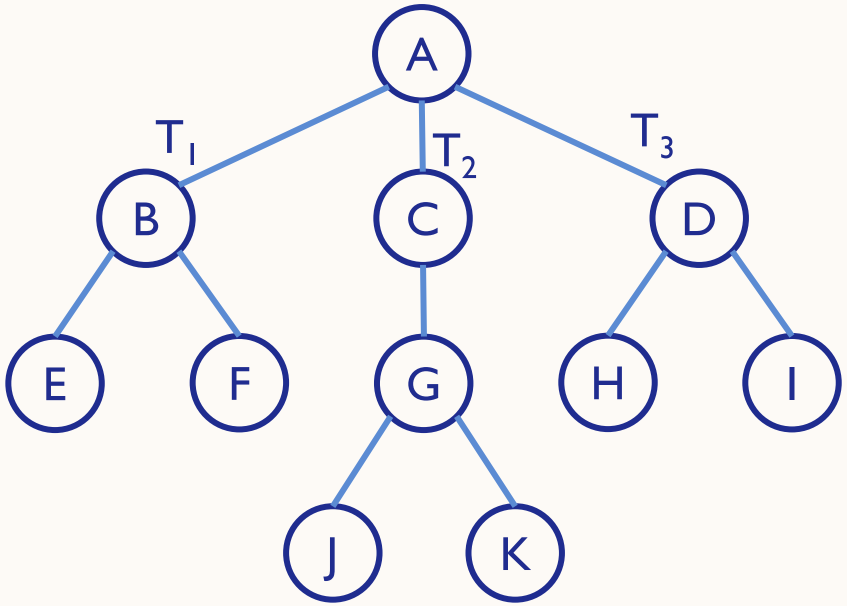
**Sub-trees**

- If **R != NULL**, then the trees $\mathbf{T_1}$, $\mathbf{T_2}$, and $\mathbf{T_3}$ are called the sub-trees of R

**Leaf node**

- A node that **has no children** is called the leaf node or the terminal node

# BASIC TERMINOLOGY



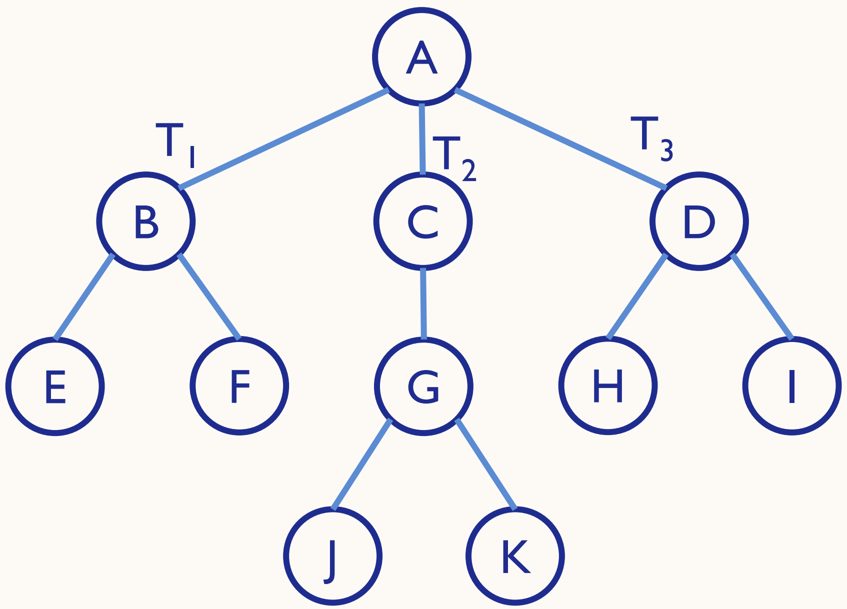

**Path**

- A **sequence of consecutive edges** is called a path
- The path from the root node A to node I is given as: **A**, **D**, and **I**

**Level number**

- Every node in the tree is assigned a level number in such a way that the **root node is at level 0**, children of the root node are at level number 1
- Every node is at **one level higher than its parent**
- All **child** nodes have a level number given by **`parent's level number + 1`**

# BASIC TERMINOLOGY



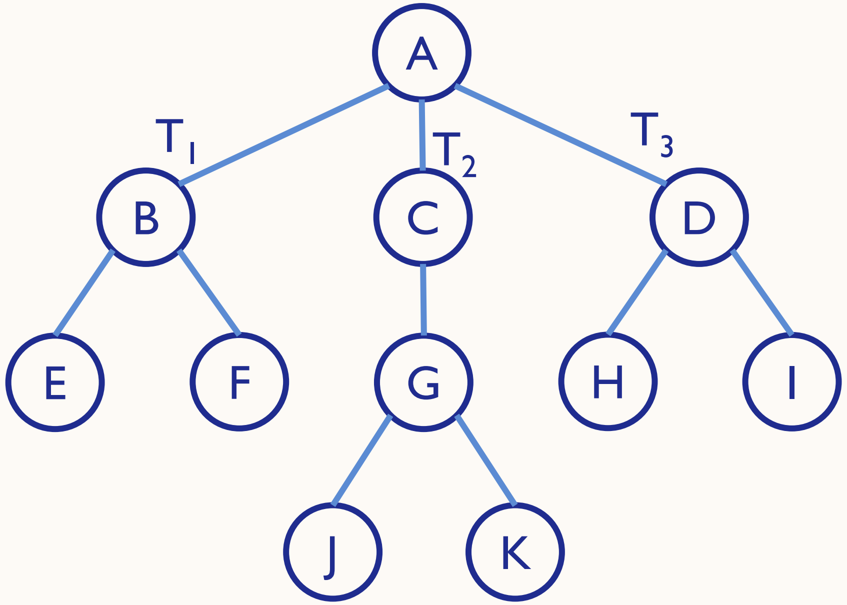

**Ancestor node**

- An ancestor of a node is **any predecessor node** on the path from root to that node
- The root node does not have any ancestors
- Nodes **A**, **C**, and **G** are the ancestors of node K

**Descendant node**

- A descendant node is **any successor node** on any path from the node to a leaf node
- Leaf nodes do not have any descendants
- Nodes **C**, **G**, **J**, and **K** are the descendants of node A

# BASIC TERMINOLOGY




**Degree**

- Degree of a node is equal to the **number of children** that a node has
- The degree of a leaf node is zero

**In-degree**

- In-degree of a node is the **number of edges arriving** at that node

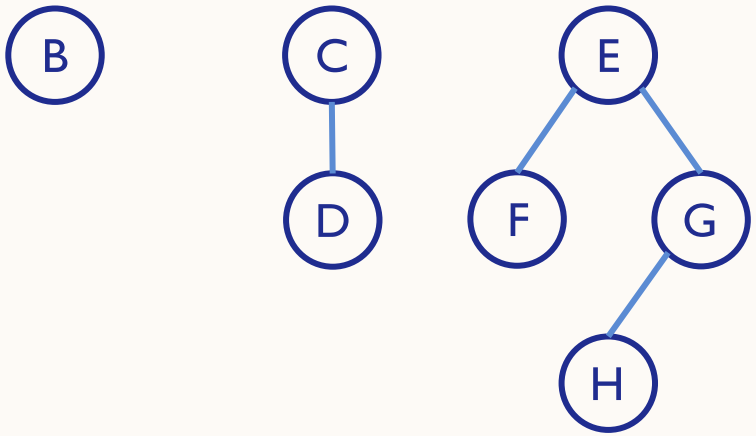
**Out-degree**

- Out-degree of a node is the **number of edges leaving** that node

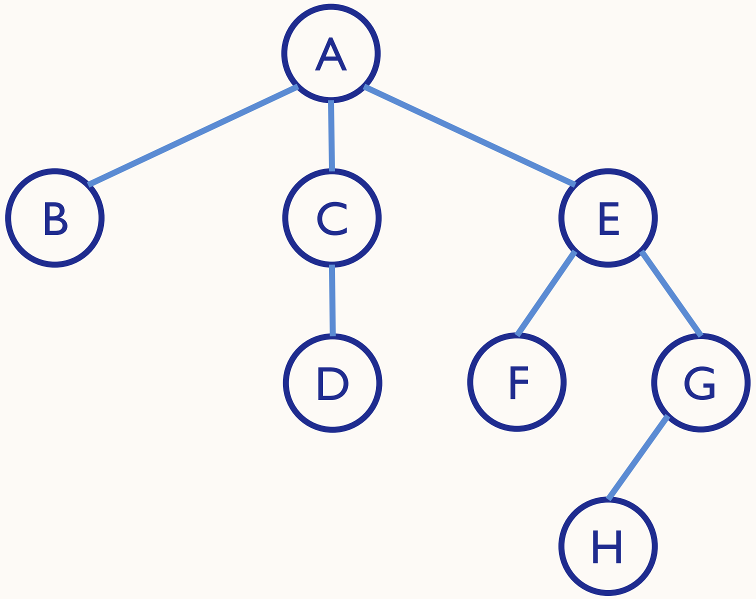
# GENERAL TREES

- A node in a general tree may have zero or more sub-trees
- The number of sub-trees for any node may be variable
- The algorithms for searching, traversing, adding, and deleting nodes become much more complex as there are multiple possibilities
- To overcome the complexities of a general tree
  - Represent as a graph
  - Convert into binary trees

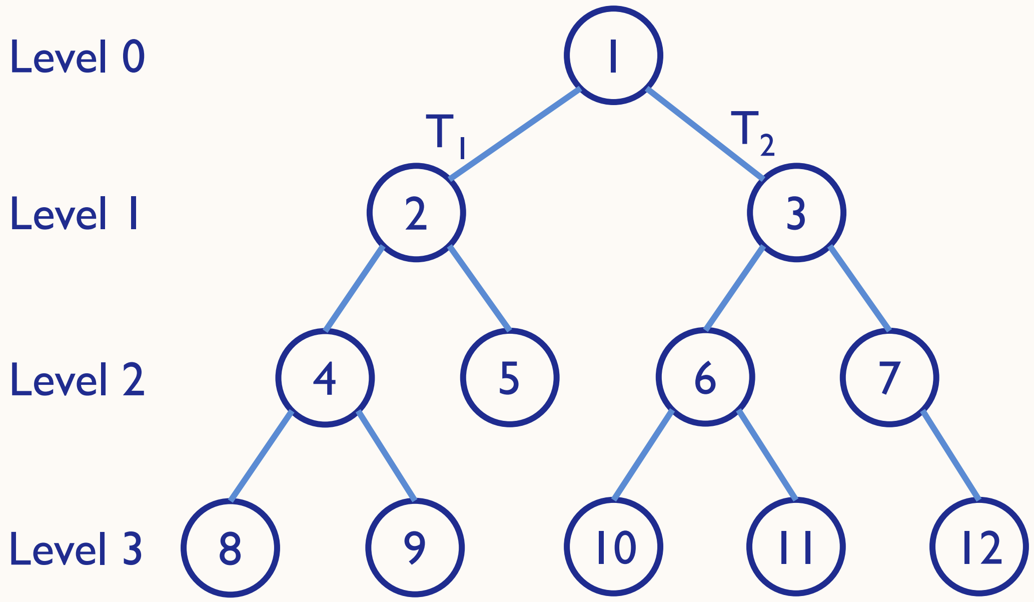
# FORESTS




- A forest is a disjoint union of trees
- A set of disjoint trees (forest) is obtained by deleting the root and the edges connecting the root node to nodes at level 1
- 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
- A forest can be converted into a tree by adding a single node as the root node of the tree

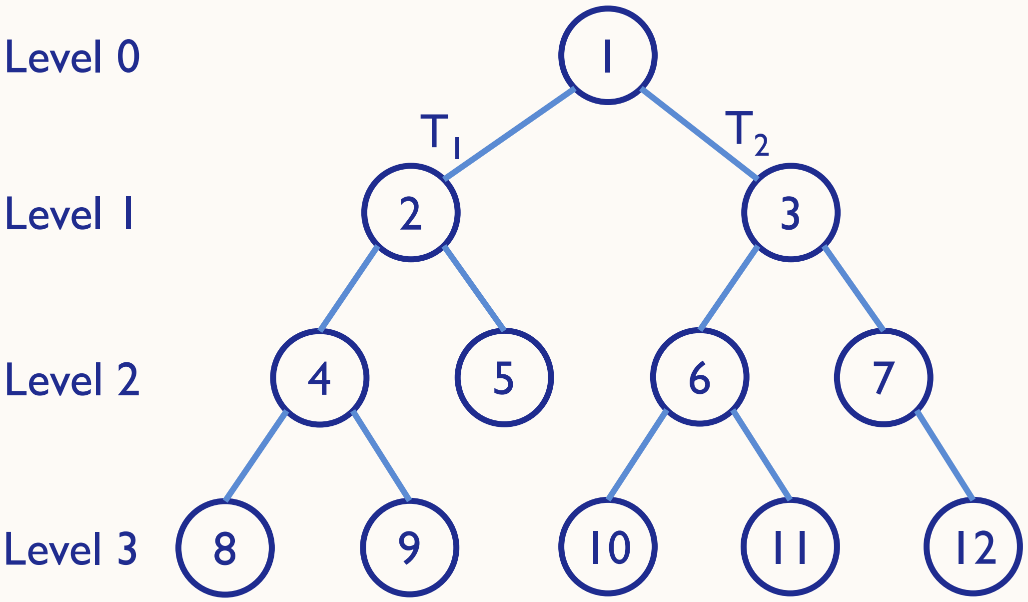


# BINARY TREES




- Each node has 0, 1, or at the most 2 children
- $T_1$ and $T_2$ are called the left and right sub-trees of the root node R
  - $T_1$ is said to be the left successor of R
  - $T_2$ is called the right successor of R
- Node 2 has two successor nodes: 4 and 5
  Node 5 has no successor
  Node 7 has only one successor: 12
- A binary tree is recursive by definition as every node in the tree contains a left sub-tree and a right sub-tree
  - The terminal nodes contain an empty left sub-tree and an empty right sub-tree

# BINARY TREES



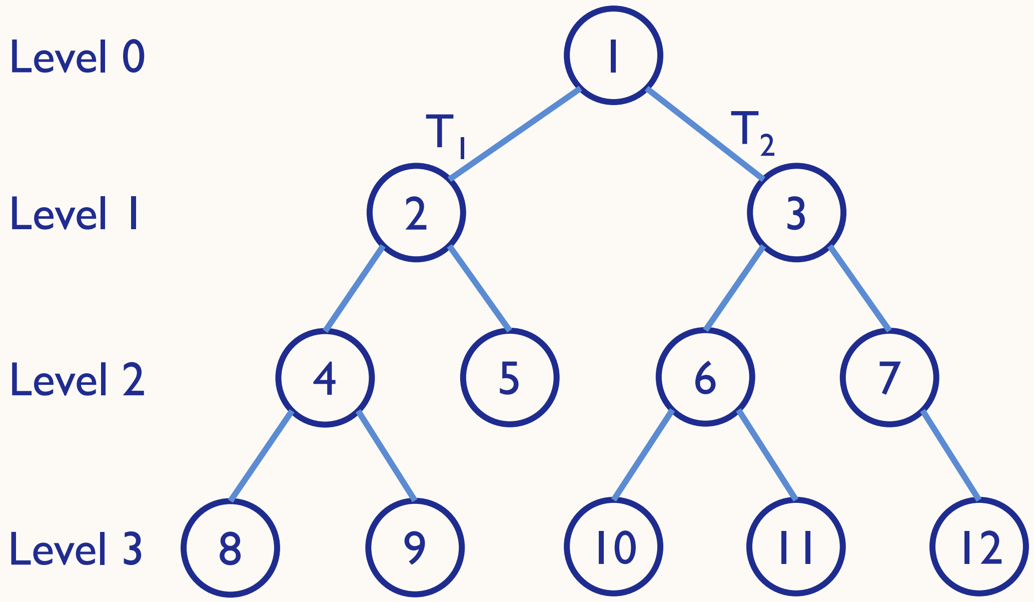

**Parent**

- If N is any node in T that has left successor $S_1$ and right successor $S_2$, then N is called the parent of $S_1$ and $S_2$
- $S_1$ and $S_2$ are called the left child and the right child of N
- Every node other than the root node has a parent

**Sibling**

- All nodes that are at the **same level** and share the **same parent** are called siblings
- Nodes 2 and 3; nodes 4 and 5; nodes 6 and 7; nodes 8 and 9; and nodes 10 and 11 are siblings

# BINARY TREES




**Leaf node**

- The leaf nodes in the tree are: 8, 9, 5, 10, 11, and 12

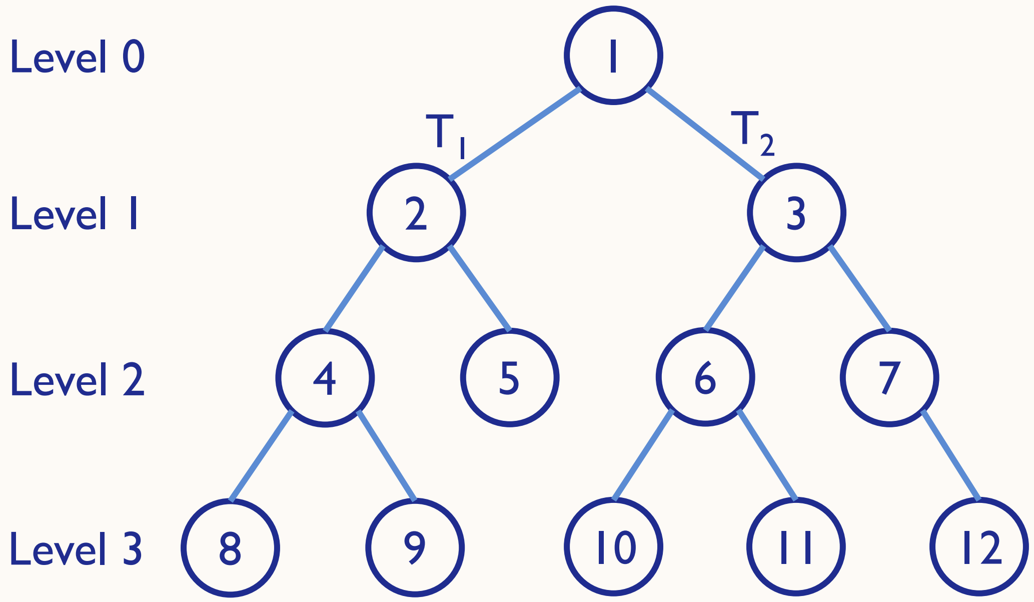
**Degree of a node**

- Degree of node 4 is 2
- Degree of node 5 is 0
- Degree of node 7 is 1

**In-degree / out-degree of a node**

- The root node is the only node that has an in-degree equal to zero

# BINARY TREES




**Edge**

- The line connecting a node N to any of its successors
- A binary tree of **n** nodes has exactly **n−1** edges because every node except the root node is connected to its parent via an edge

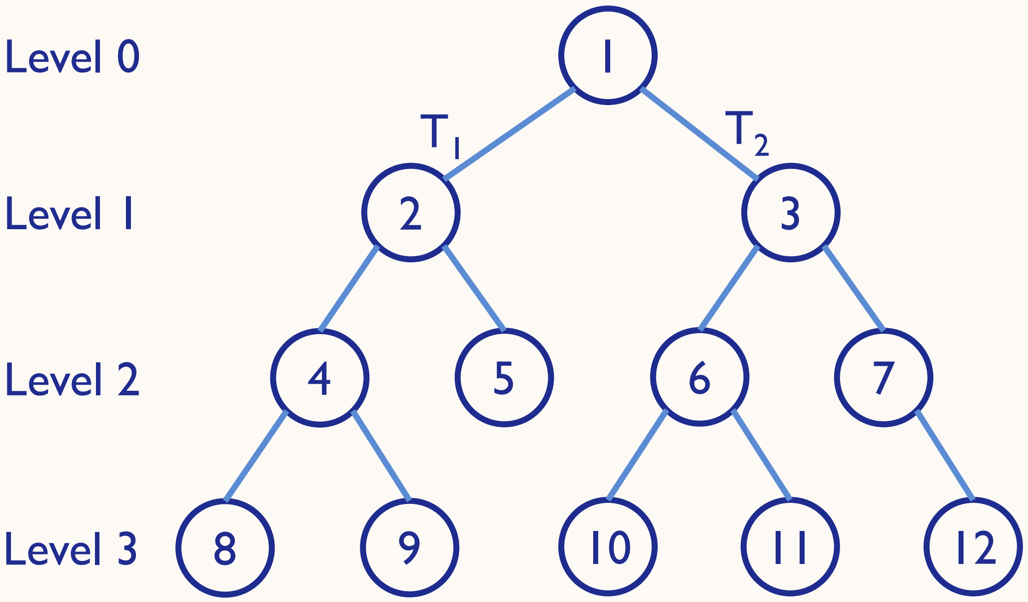
**Path**

- The path from the root node to the node 8 is given as: 1, 2, 4, and 8

**Depth**

- The depth of a node N is given as the **length of the path** from the root R to the node N
- The depth of the root node is zero

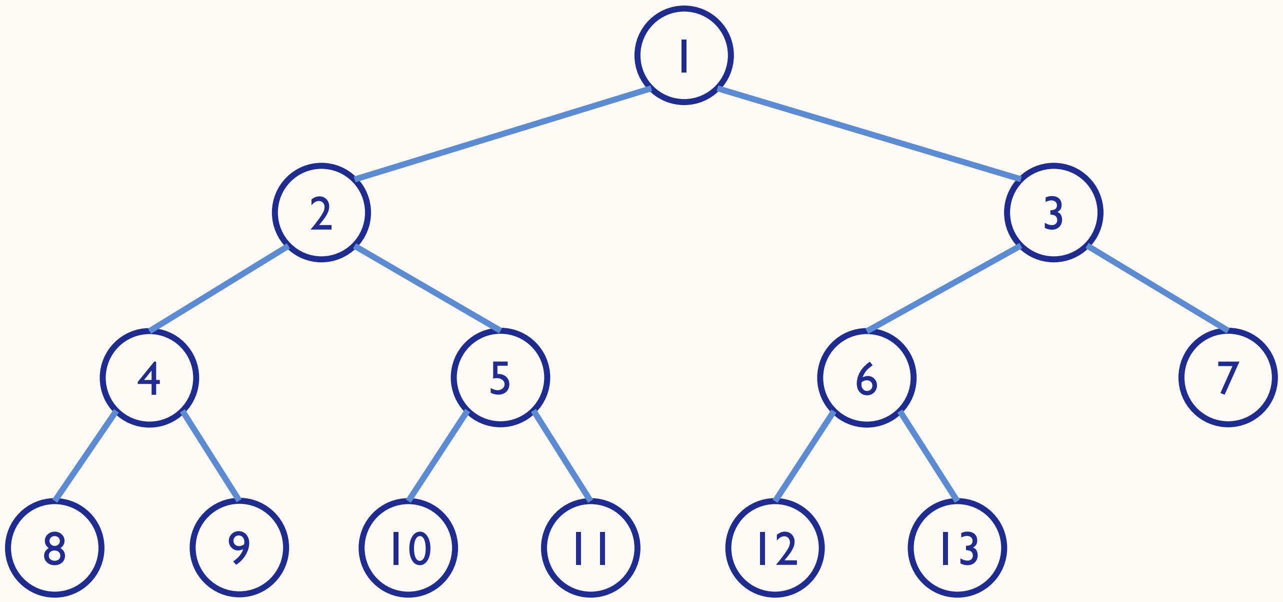
# BINARY TREES




**Height of a tree**

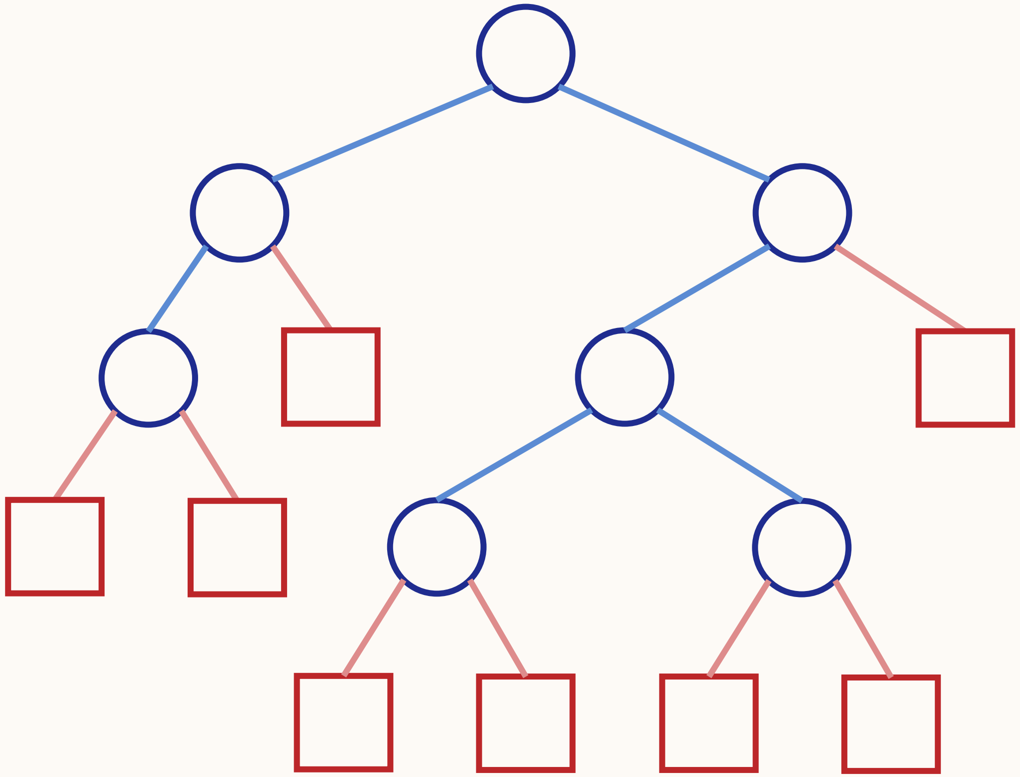
- The total number of nodes on the path from the root node to the deepest node in the tree
- A tree with only a root node has a height of 1
- A binary tree of height **h** has at least **h** nodes and at most $2^h - 1$ nodes
- The height of a binary tree with **n** nodes is at least $\log_2(n+1)$ and at most **n**

# COMPLETE BINARY TREES




- A complete binary tree is a binary tree that satisfies two properties
  - Every level, except possibly the last, is **completely filled**
  - All nodes appear **as far left as possible**

# 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**
  - Nodes having two children are called **internal nodes**
  - Nodes having no children are called **external nodes**

- To convert an ordinary binary tree into an extended binary tree, every empty sub-tree is replaced by a new node

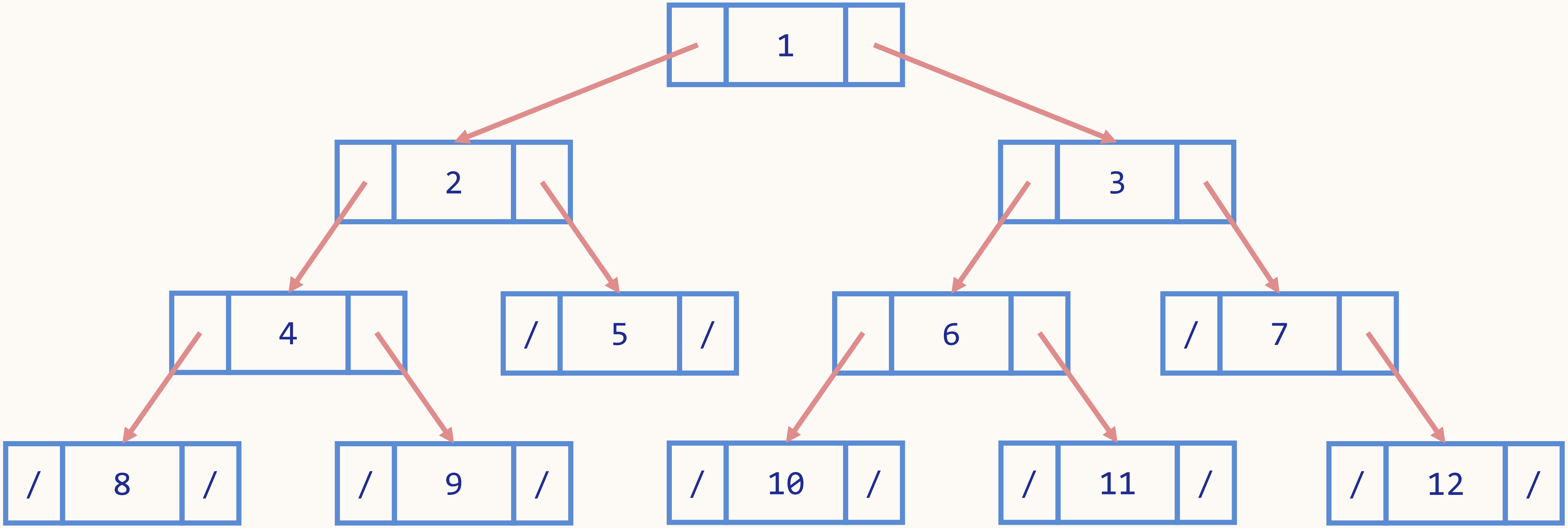
# LINKED REPRESENTATION OF BINARY TREES

- Every node will have three parts
  - The data element
  - A pointer to the left node
  - A pointer to the right node

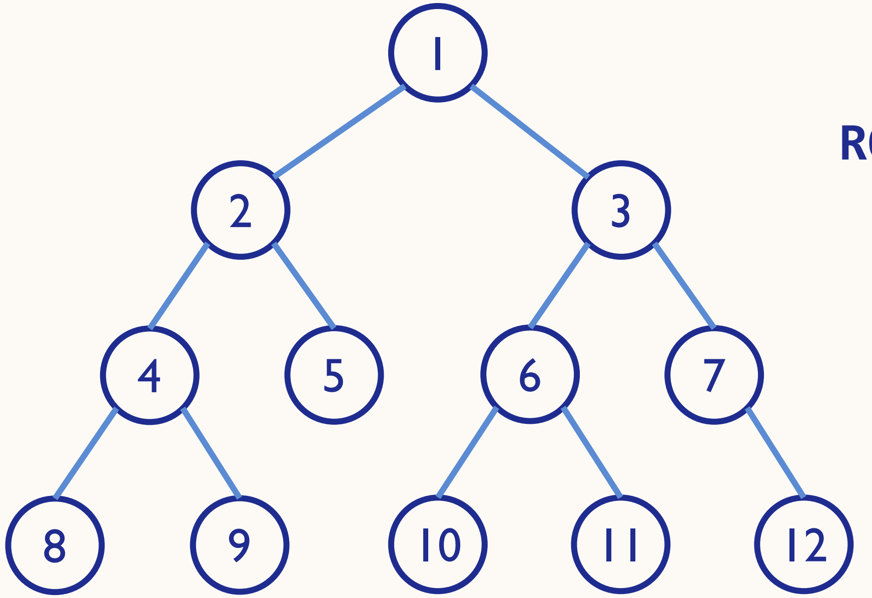
```
struct tbintree{
    struct tbintree *left;
    int data;
    struct tbintree *right;
};
```

- Every binary tree has a pointer **ROOT**, which points to the root element (topmost element) of the tree
  - If **ROOT = NULL**, then the tree is <u>empty</u>

# LINKED REPRESENTATION OF BINARY TREES



# LINKED REPRESENTATION OF BINARY TREES



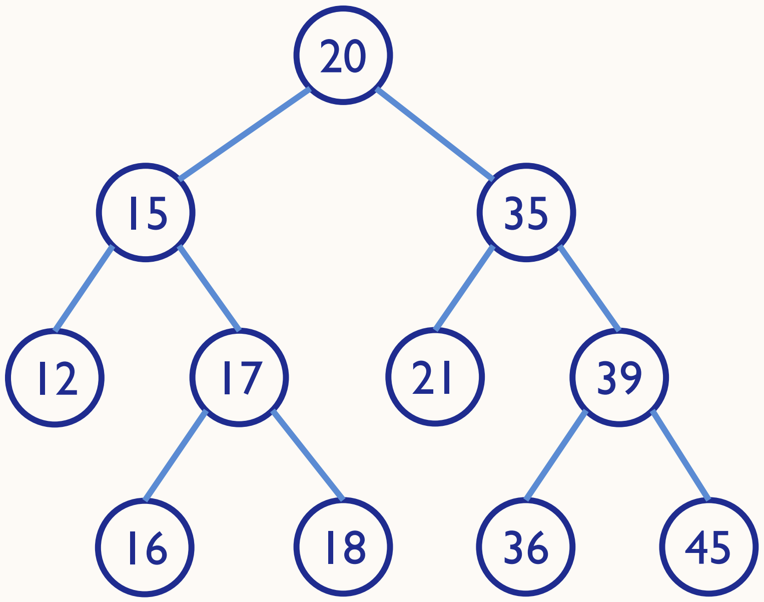

| | | LEFT | DATA | RIGHT |
|---|---|---|---|---|
| | 1 | -1 | 8 | -1 |
| | 2 | -1 | 10 | -1 |
| ROOT | 3 | 5 | 1 | 8 |
| | 4 | | | |
| | 5 | 9 | 2 | 14 |
| | 6 | | | |
| | 7 | | | |
| | 8 | 20 | 3 | 11 |
| | 9 | 1 | 4 | 12 |
| | 10 | | | |

| | LEFT | DATA | RIGHT |
|---|---|---|---|
| 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 |

# SEQUENTIAL REPRESENTATION OF BINARY TREES

- Sequential representation of trees is done using single or one-dimensional arrays
- Though it is the simplest technique for memory representation, it is inefficient as it requires a lot of memory space

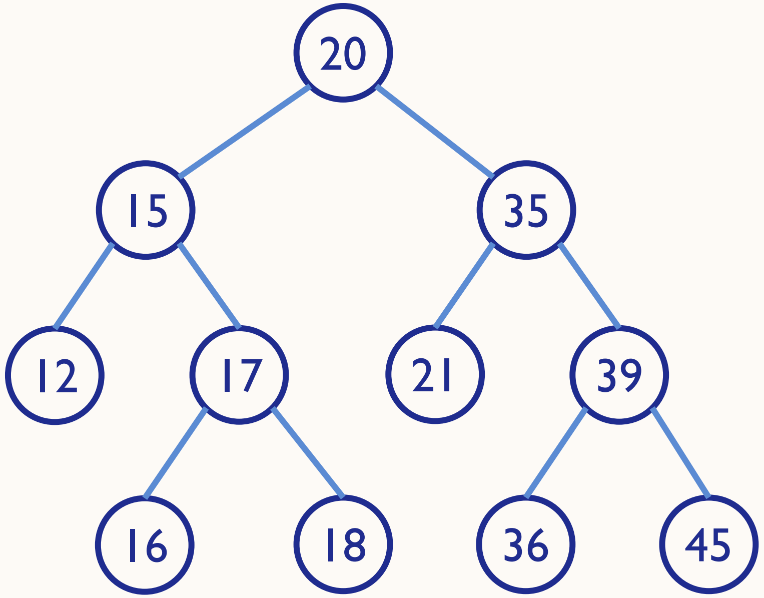
# SEQUENTIAL REPRESENTATION OF BINARY TREES




| | | | |
|---|---|---|---|
| **1** | 20 | **11** | 18 |
| **2** | 15 | **12** | |
| **3** | 35 | **13** | |
| **4** | 12 | **14** | 36 |
| **5** | 17 | **15** | 45 |
| **6** | 21 | | |
| **7** | 39 | | |
| **8** | | | |
| **9** | | | |
| **10** | 16 | | |

- A one-dimensional array, called **`TREE`**, is used to store the elements of tree
- The root of the tree will be stored in the first location
  - **`TREE[1]`** will store the data of the root element
- An empty tree or sub-tree is specified using NULL
  - If **`TREE[1] = NULL`**, then the tree is empty

# SEQUENTIAL REPRESENTATION OF BINARY TREES




| | | | |
|---|---|---|---|
| 1 | 20 | 11 | 18 |
| 2 | 15 | 12 | |
| 3 | 35 | 13 | |
| 4 | 12 | 14 | 36 |
| 5 | 17 | 15 | 45 |
| 6 | 21 | | |
| 7 | 39 | | |
| 8 | | | |
| 9 | | | |
| 10 | 16 | | |

- The children of a node stored in location **K** will be stored in locations **(2K)** and **(2K + 1)**
- The maximum size of the array **TREE** is given as $(2^h - 1)$, where **h** is the height of the tree

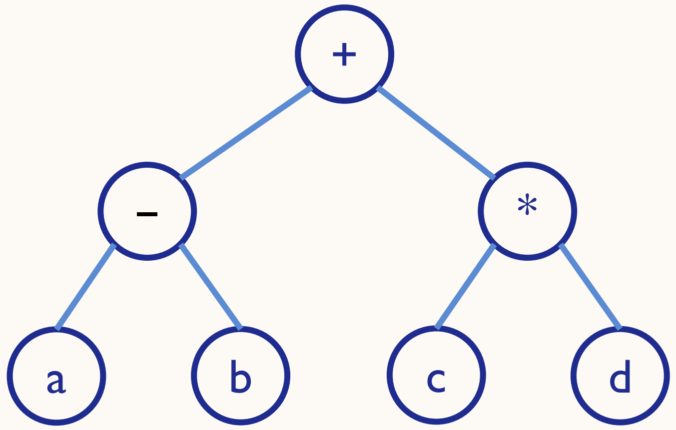
# BINARY SEARCH TREES

- A binary search tree, also known as an ordered binary tree, is a variant of binary tree in which the nodes are arranged in an order

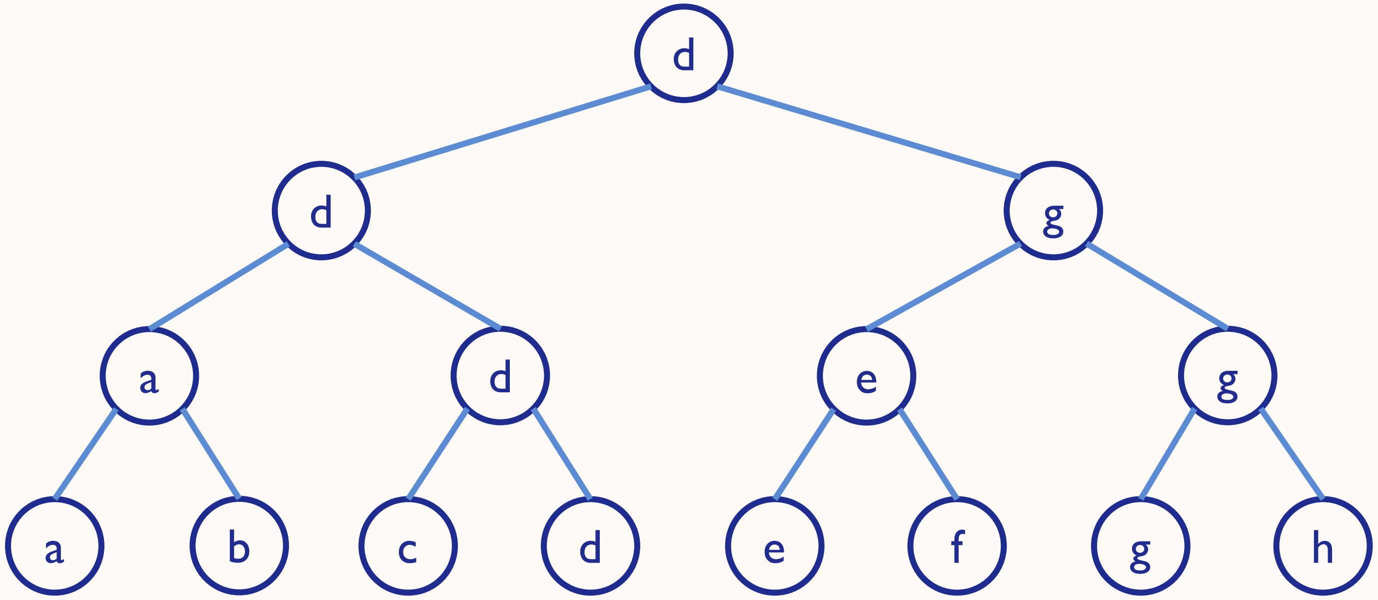
# EXPRESSION TREES

- Binary trees are widely used to store algebraic expressions
- Example

Exp = (a – b) + (c * d)



# 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

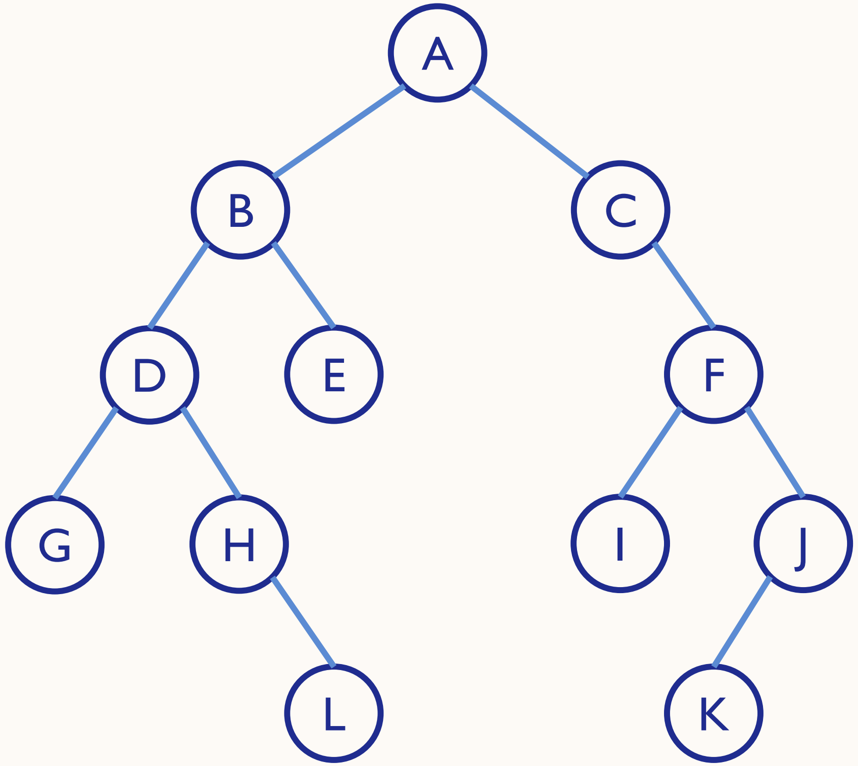
# TRAVERSING A BINARY TREE

- Traversing a binary tree is the process of **visiting each node** in the tree **exactly once** in a systematic way
- There are different algorithms for tree traversals
  - Pre-order Traversal
  - In-order Traversal
  - Post-order Traversal
- These algorithms differ in the order in which the nodes are visited

# PRE-ORDER TRAVERSAL

- The following operations are performed recursively at each node
  1. Visiting the root node
  2. Traversing the left sub-tree
  3. Traversing the right sub-tree

```
void preorder(struct tbintree *root){
    if(root != NULL){
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
```



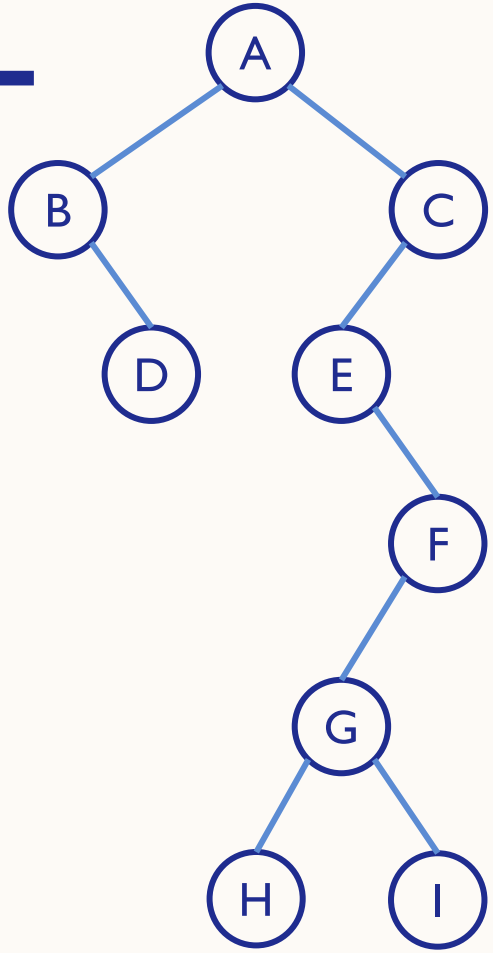

Traversal Order:

A, B, D, G, H, L, E, C, F, I, J, K

# PRE-ORDER TRAVERSAL

- The following operations are performed recursively at each node
  1. Visiting the root node
  2. Traversing the left sub-tree
  3. Traversing the right sub-tree

```
void preorder(struct tbintree *root){
    if(root != NULL){
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
```



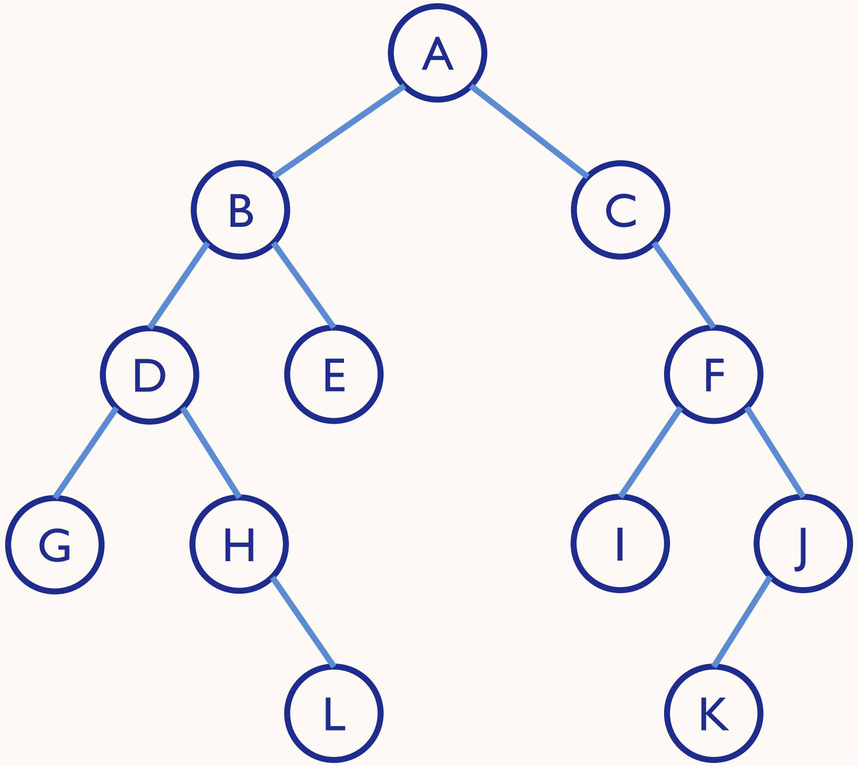

Traversal Order:

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

# IN-ORDER TRAVERSAL

- The following operations are performed recursively at each node
  1. Traversing the left sub-tree
  2. Visiting the root node
  3. Traversing the right sub-tree

```
void inorder(struct tbintree *root){
    if(root != NULL){
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
```

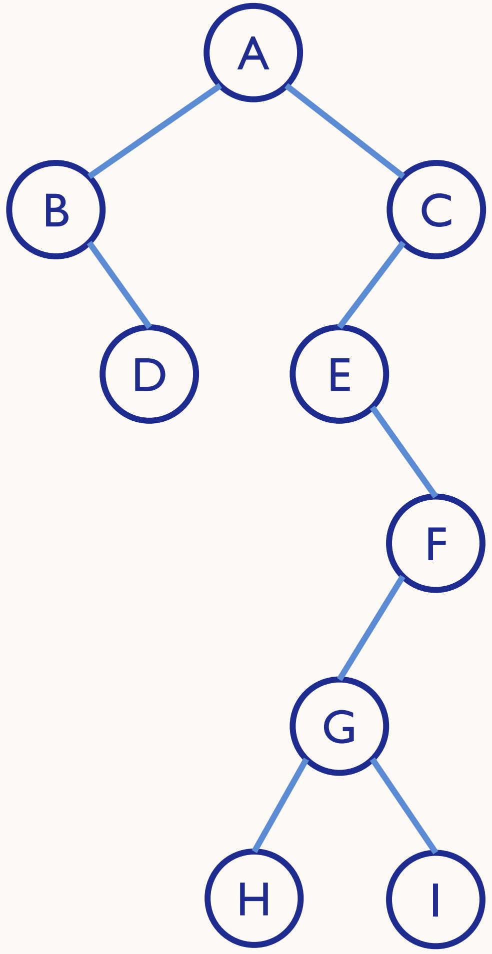



Traversal Order:
G, D, H, L, B, E, A, C, I, F, K, J

# IN-ORDER TRAVERSAL

- The following operations are performed recursively at each node
  1. Traversing the left sub-tree
  2. Visiting the root node
  3. Traversing the right sub-tree

```
void inorder(struct tbintree *root){
   if(root != NULL){
      inorder(root->left);
      printf("%d ", root->data);
      inorder(root->right);
   }
}
```

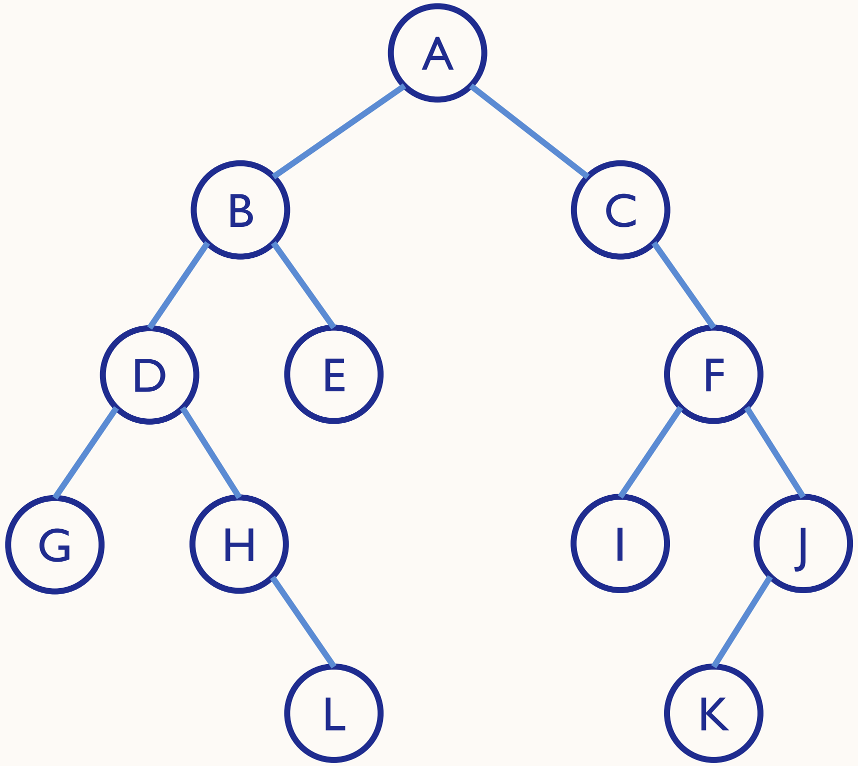



Traversal Order:
B, D, A, E, H, G, I, F, C

# POST-ORDER TRAVERSAL

- The following operations are performed recursively at each node
  1. Traversing the left sub-tree
  2. Traversing the right sub-tree
  3. Visiting the root node

```
void postorder(struct tbintree *root){
    if(root != NULL){
        postorder(root->left);
        postorder(root->right);
        printf(“%d ”, root->data);
    }
}
```



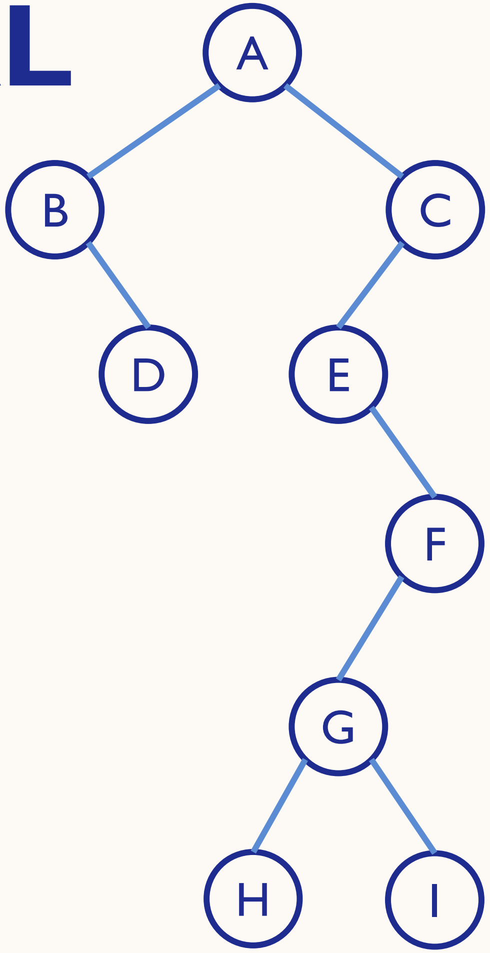

Traversal Order:

G, L, H, D, E, B, I, K, J, F, C, A

# POST-ORDER TRAVERSAL

- The following operations are performed recursively at each node
  1. Traversing the left sub-tree
  2. Traversing the right sub-tree
  3. Visiting the root node

```
void postorder(struct tbintree *root){
    if(root != NULL){
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}
```




Traversal Order:

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

# APPLICATIONS OF TREES

- Trees are used to store simple (integer, character) as well as complex data (structure, record)
- 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 multiprocessor computer operating system use
- Another variation of tree, B-trees are prominently used to store tree structures on disc. They are used to index a large number of records.

# APPLICATIONS OF TREES

- B-trees are also used for secondary indexes in databases, where the index facilitates a select operation to answer some range criteria
- Trees are an important data structure 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

# PRACTICE

# EXERCISES

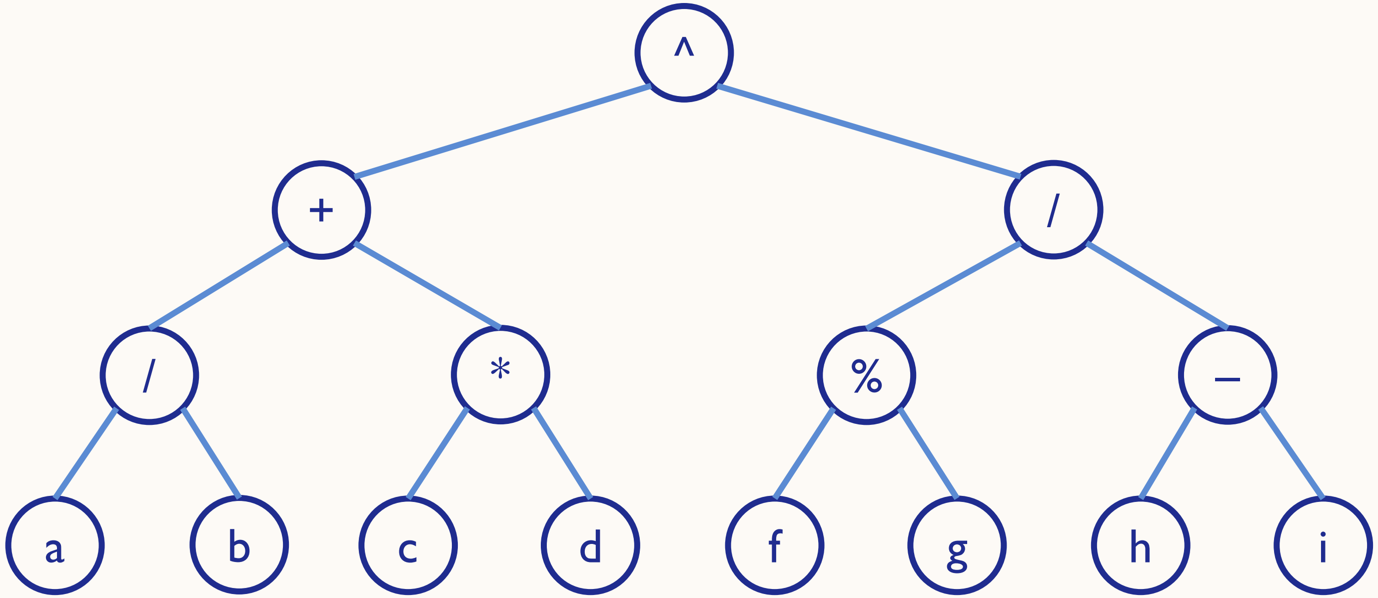
1. Given the memory representation of a tree that stores the names of family members, construct the corresponding tree from the given data.

| | LEFT | DATA | RIGHT |
|---|---|---|---|
| 1 | 12 | Keaton | -1 |
| 2 | | | |
| ROOT 3 | 9 | Penelope | 13 |
| 4 | | | |
| 5 | | | |
| 6 | 19 | Derwin | 17 |
| 7 | | | |
| 8 | | | |
| 9 | 1 | Rooney | -1 |
| 10 | | | |

| | LEFT | DATA | RIGHT |
|---|---|---|---|
| 11 | -1 | Twiggy | -1 |
| 12 | -1 | Jitters | -1 |
| 13 | 6 | Marshal | 20 |
| 14 | | | |
| 15 | -1 | Tiansheng | -1 |
| 16 | | | |
| 17 | -1 | Gaston | -1 |
| 18 | | | |
| 19 | -1 | Vesta | -1 |
| 20 | 11 | Quinn | 15 |

# EXERCISES

2. Given the binary tree, write down the expression that it represents.



# EXERCISES

3. Given the expressions below, construct the corresponding binary tree.

   a. `((a + b) - (c * d)) % ((e ^ f) / (g - h))`
   b. `a + b / c * d - e`

4. Draw the binary expression tree that represents the following postfix expression.

   **`A B + C * D -`**

5. Convert the prefix expression **`- / a b * + b c d`** into infix expression and then draw the corresponding expression tree.

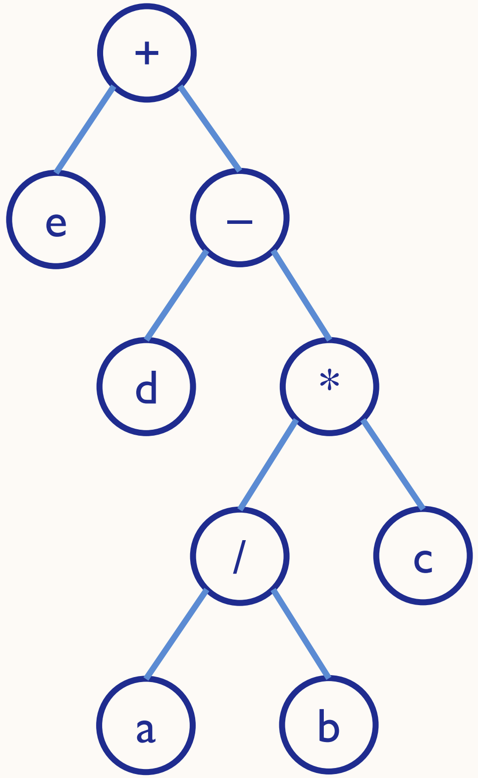
# EXERCISES

6. For the given expression tree, do the following.

   a. Extract the infix expression it represents

   b. Find the corresponding prefix and postfix expressions

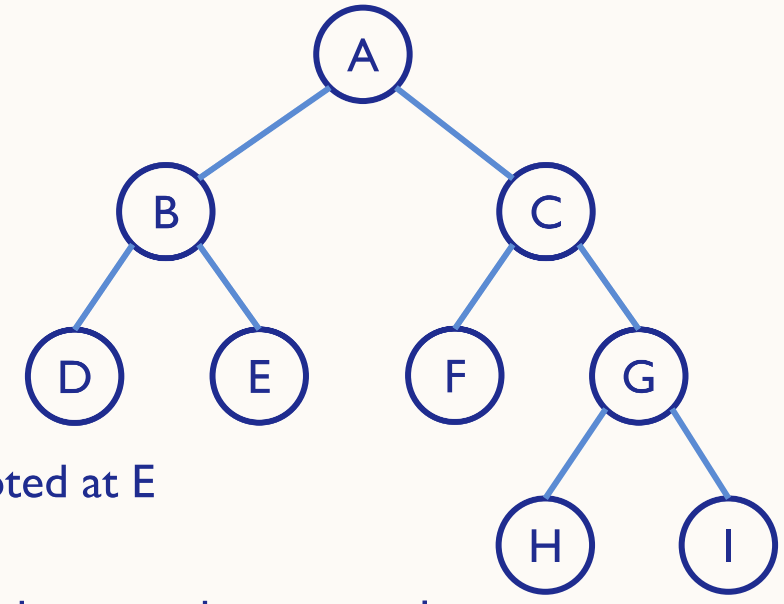
   c. Evaluate the infix expression, given a = 30, b = 10, c = 2, d = 30, e = 10



# EXERCISES

7. Consider the tree given below. Now, do the following.

   a. Name the leaf nodes
   b. Name the non-leaf nodes
   c. Name the ancestors of E
   d. Name the descendants of A
   e. Name the siblings of C
   f. Find the height of the tree
   g. Find the height of sub-tree rooted at E
   h. Find the level of node E
   i. Find the in-order, pre-order, and post-order traversal



# EXERCISES

8. What is the maximum number of levels that a binary search tree with 100 nodes can have?
9. What is the maximum height of a tree with 32 nodes?
10. What is the maximum number of nodes that can be found in a binary tree at levels 3, 4, and 12?

# EXERCISES

11. Construct the binary tree from the traversal results given below.

    a. In-order Traversal : D B E A F C G
       Pre-order Traversal : A B D E C F G

    b. In-order Traversal : D B H E I A F J C G
       Post-order Traversal : D H I E B J F G C A

# REFERENCES


- Deitel, P. and Harvey Deitel (2022), C How to Program (9th Edition), Pearson Education.
- Thareja, R. (2014), Data Structures Using C (2nd Edition), India: Oxford University Press.

# NEXT

**Efficient Binary Trees:**

Binary Search Trees

AVL Trees

# VISION

To become an **outstanding** undergraduate Computer Science program that produces **international-minded** graduates who are **competent** in software engineering and have **entrepreneurial spirit** and **noble character**.

# MISSION

1. To conduct studies with the best technology and curriculum, supported by professional lecturer
2. To conduct research in Informatics to promote science and technology
3. To deliver science-and-technology-based society services to implement science and technology

Without hard work,

nothing grows but weeds.

UMN

UNIVERSITAS MULTIMEDIA NUSANTARA

INFORMATIKA UMN

Have patience.

All things are difficult before they become easy.