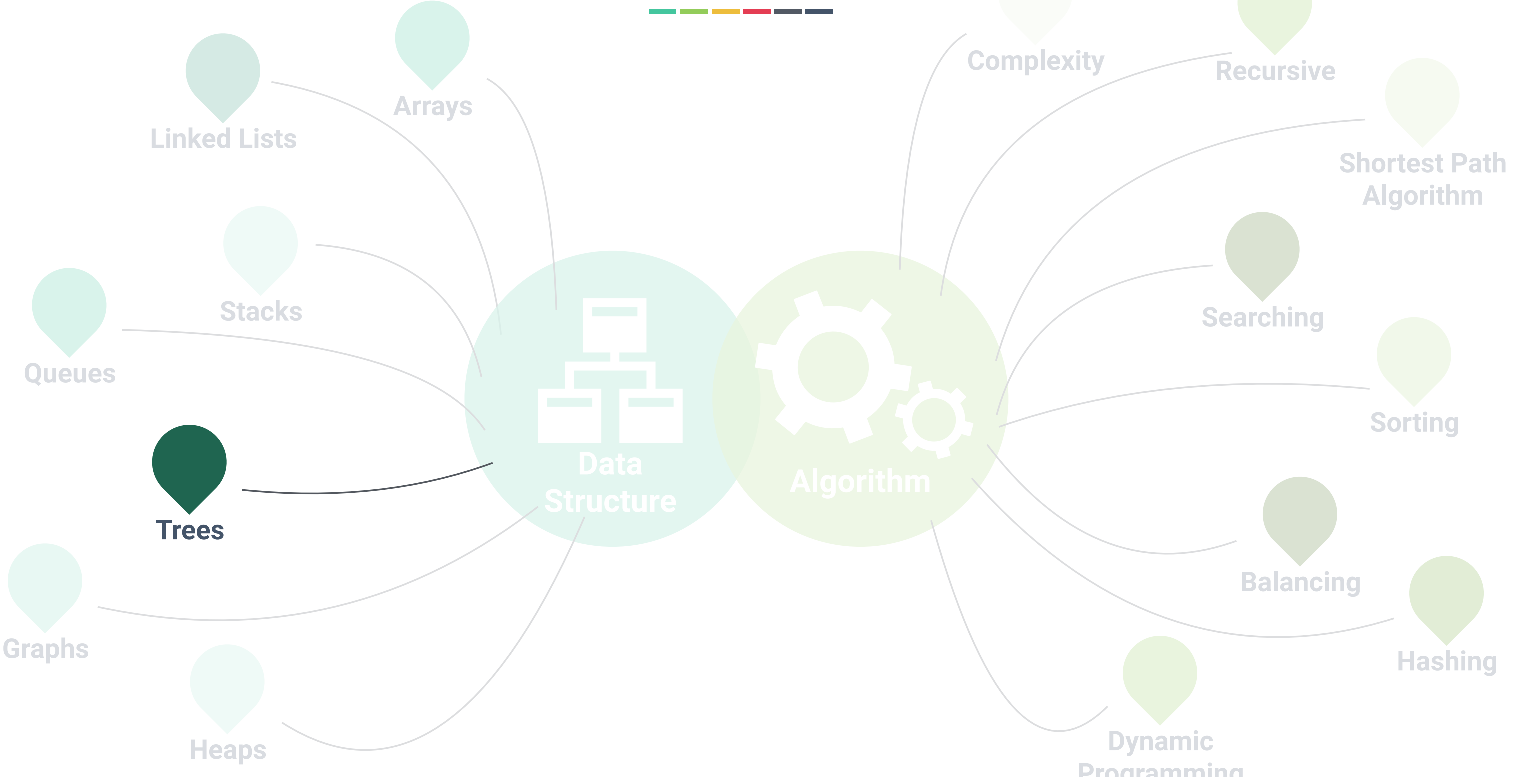
Chapter 6: Trees

(Tree and Binary Tree)

Dr. Sirasit Lochanachit

Overall



Abstract Data Type

Linear:

- Arrays, Stacks, Queues, and Linked Lists
- Algorithms: Sequential search, Binary search, and sorting (soon)

Non-Linear:

- Trees and Graphs
- There are much more algorithms than linear data structure!!

Outline

General Trees:

- Definition and examples
- Elements of Tree Structure

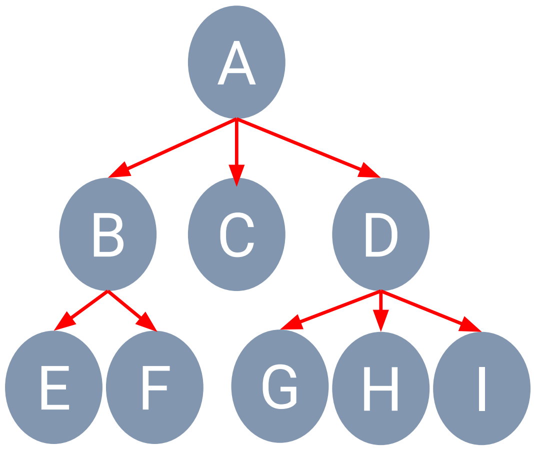
Binary Trees:

- Definition, examples, properties, and types
- Applications (i.e. Expression Tree)

Tree traversal algorithms:

- Depth-first Traversal (Preorder, postorder, and inorder)
- Breadth-first Traversal

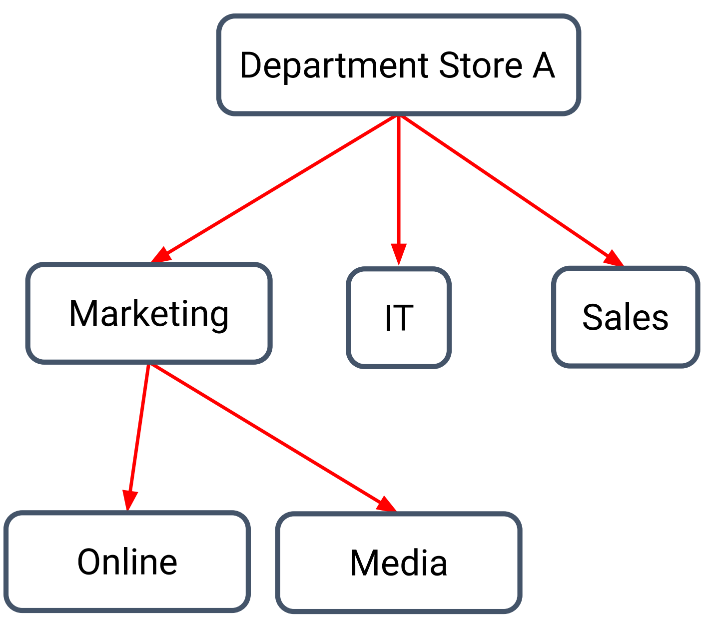
General Trees



- A **tree** stores elements in a hierarchical structure.
- Each element in a tree has a **parent** element and zero or more **children** elements, except the top element which is the **root** of the tree, having only children element(s).
- The term **parent/child** and **ancestor/descendant** are commonly used to describe tree structure.

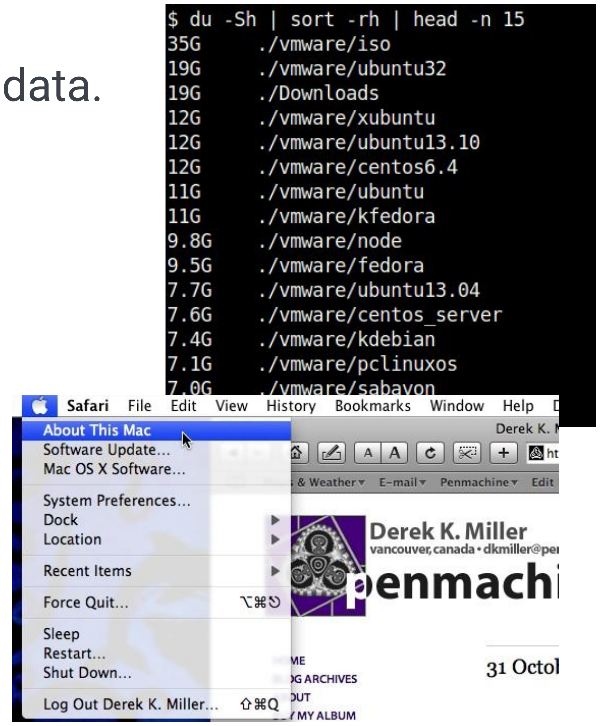
[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

General Trees

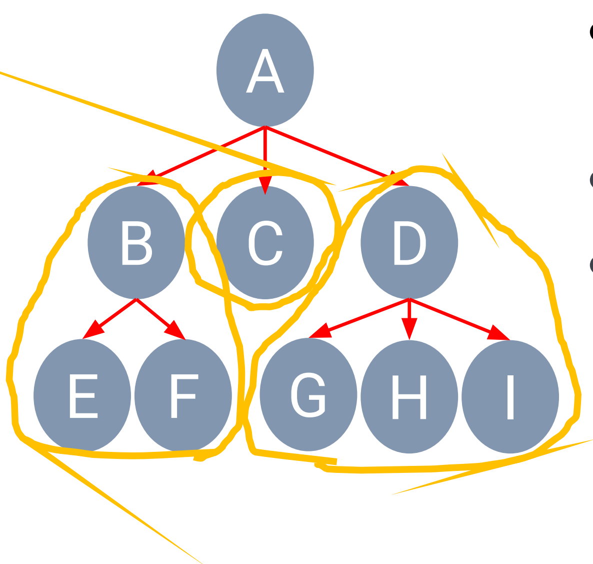


General Trees

- Trees represents natural organisation for data.
- Tree structure has been used widely in
  - File systems (Directory)
  - Graphical User Interfaces
  - Databases (Sub-categories, etc.)
  - Websites



Formal Tree Definition

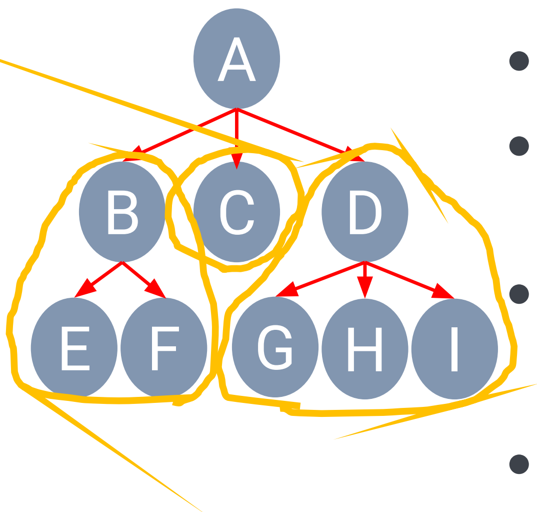


- A **tree** is a collection of nodes that store elements with a **parent-child** relationship.
- If tree T is empty, it does not have any nodes.
- If tree T is nonempty, it has a **root** node (r), which is the root of T
  - It also has a set of **subtrees** whose roots are the children of r.

[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Formal Tree Definition

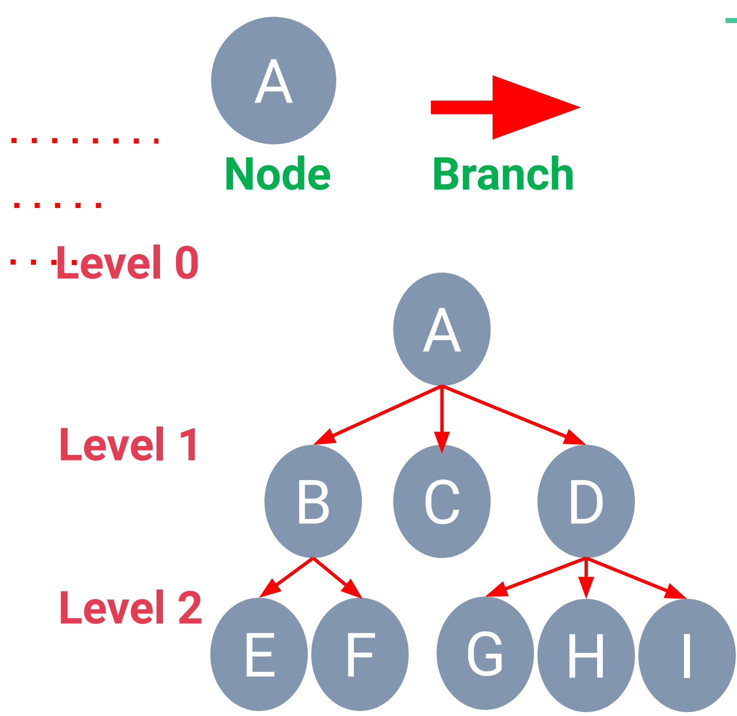


- **Siblings** are children nodes with the same parent.
- **External** or **<u>leaf</u>** node is a node with no children.
- A node is **internal** when there is one or more children.
- Node A is an **ancestor** of node E
  - Node A is a parent of the parent (B) of node E.
- In contrast, node E is a **descendant** of node A.

[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

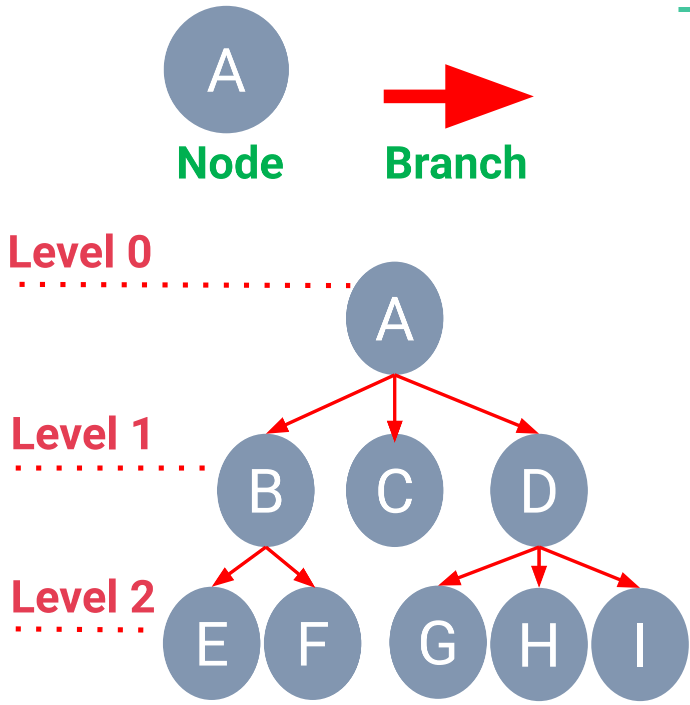
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Basic Elements of Tree Structure



- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height is 3

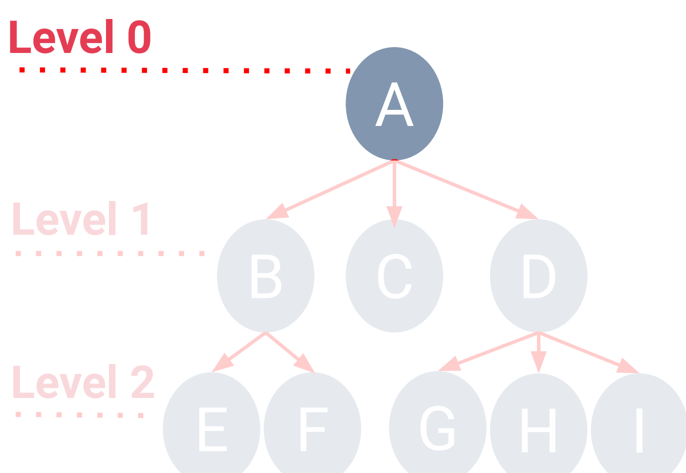
Basic Elements of Tree Structure



- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height is 3

Basic Elements of Tree Structure

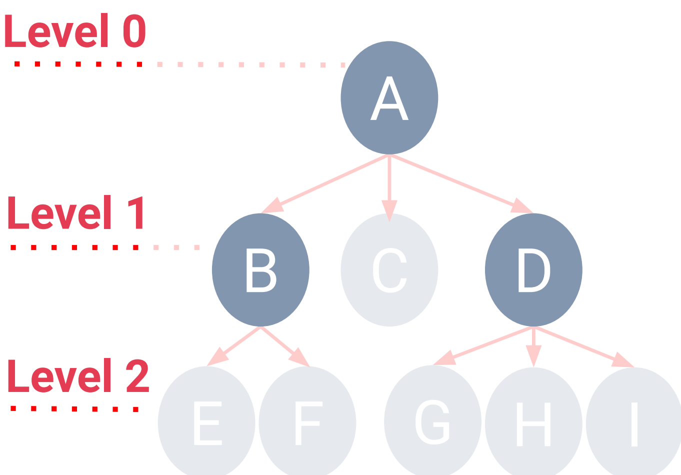
Root node is the *top element* of tree.
It is the only item that has no parent.



- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height is 3

Basic Elements of Tree Structure

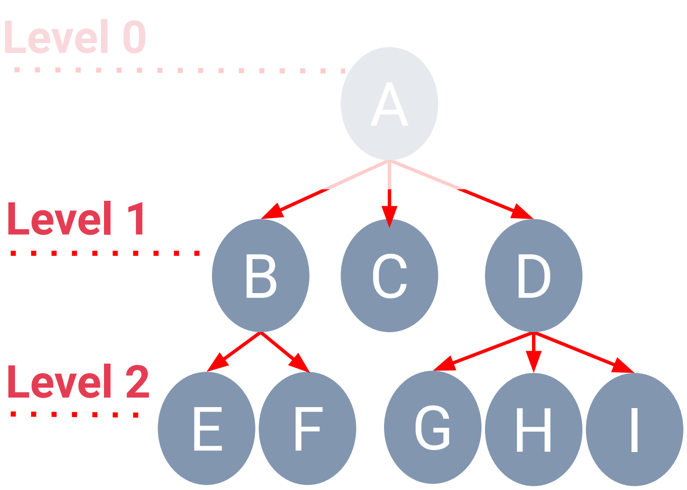
Parent node is an internal node that has one or more children nodes.



- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height is 3

Basic Elements of Tree Structure

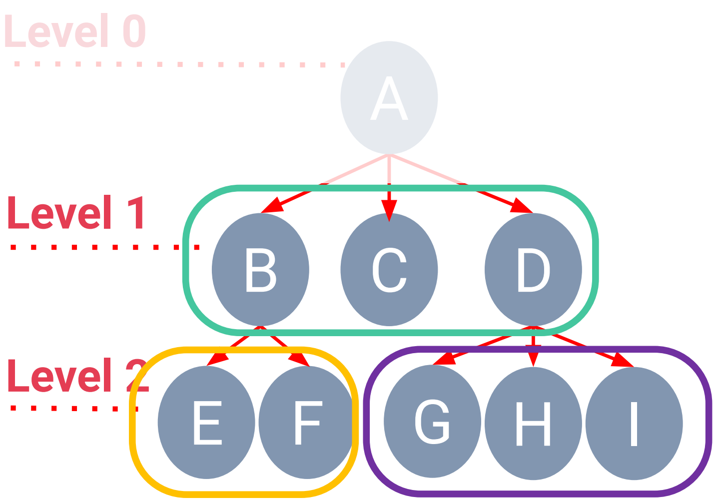
Children node is a node that has parent node.



- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height is 3

Basic Elements of Tree Structure

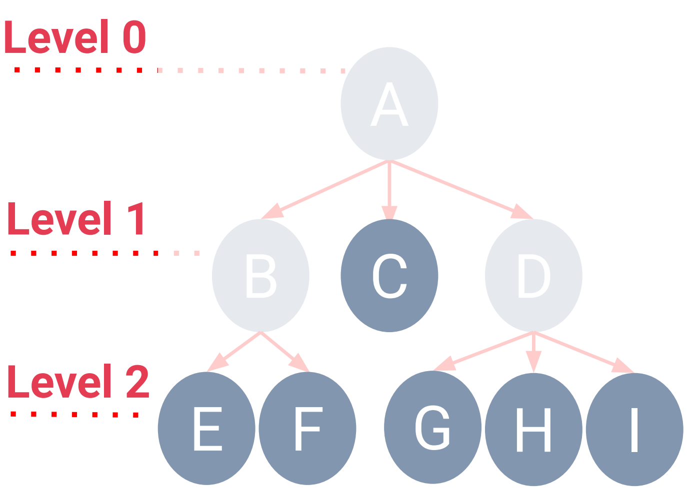
Two or more children nodes are ***siblings*** when they have the same parent node.



- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height is 3

Basic Elements of Tree Structure

Leaf node is a node without any children.

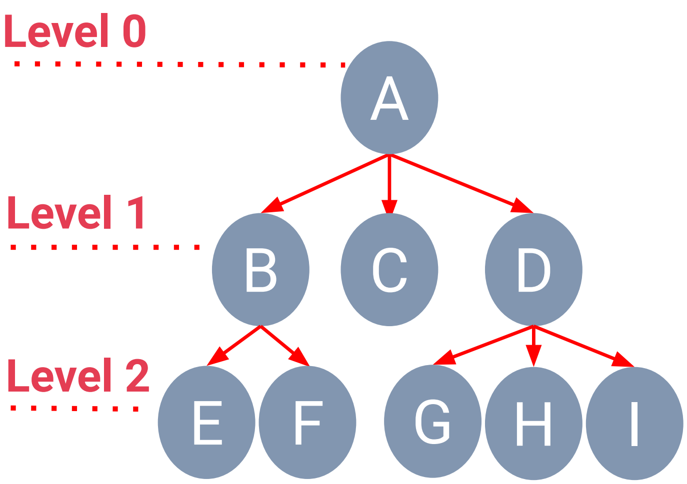


- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height is 3

Basic Elements of Tree Structure

A ***Degree of tree*** is the number of subtrees or edges/links.

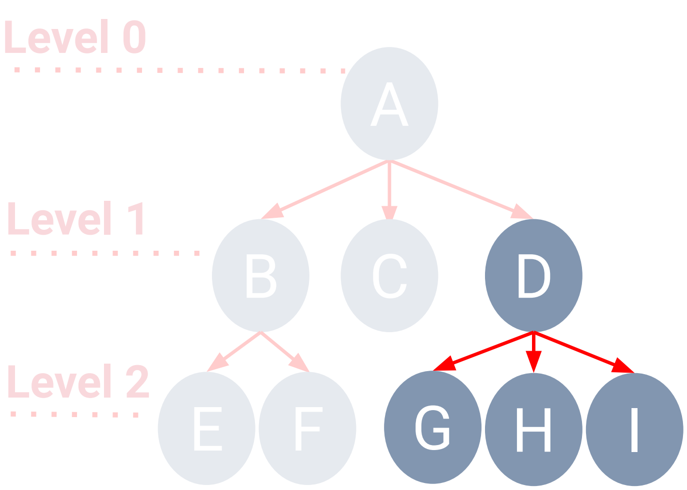
Edge is a pair of parent and child node.



- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height is 3

Basic Elements of Tree Structure

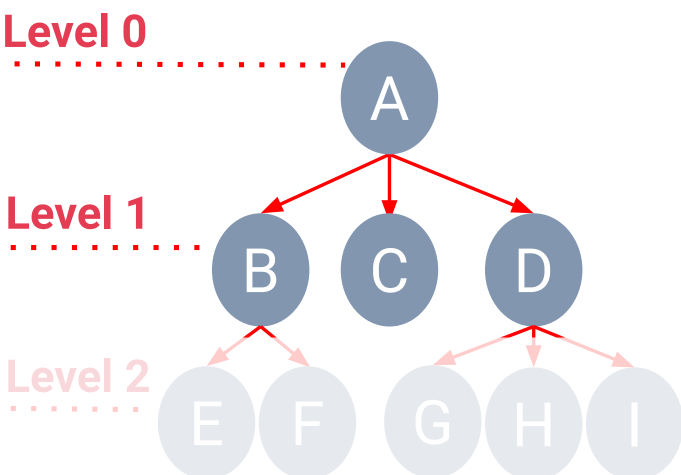
A ***Degree of node*** is the number of edges connected to the node.



- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height is 3

Basic Elements of Tree Structure

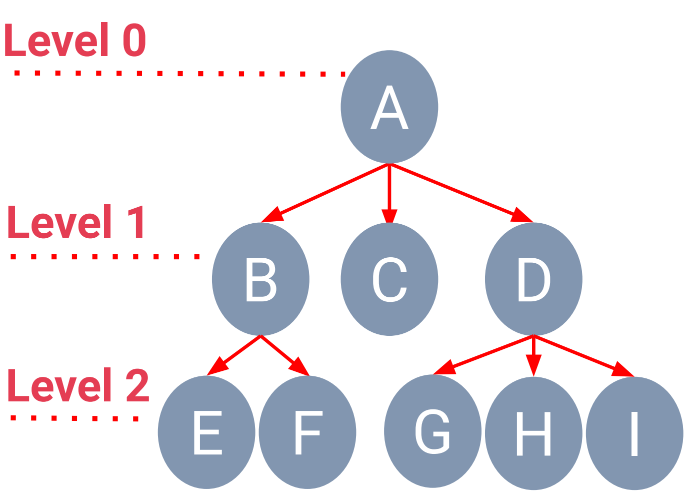
Depth of a node is the length of the path to its root.



- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height of tree is 3
- Depth of node D is 1

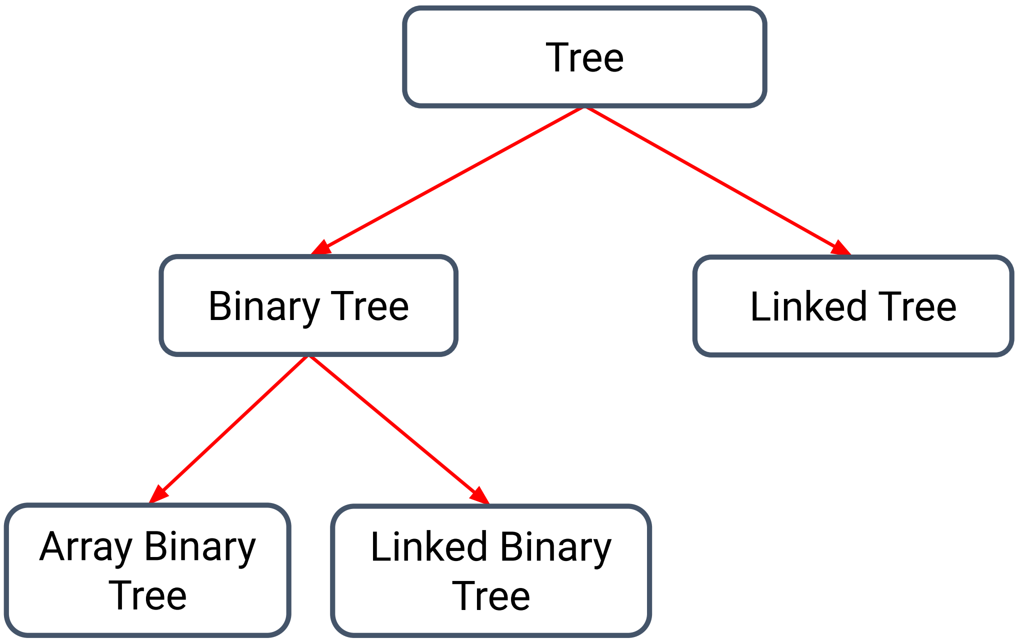
Basic Elements of Tree Structure

Height of tree is the number of levels starting from the root node.

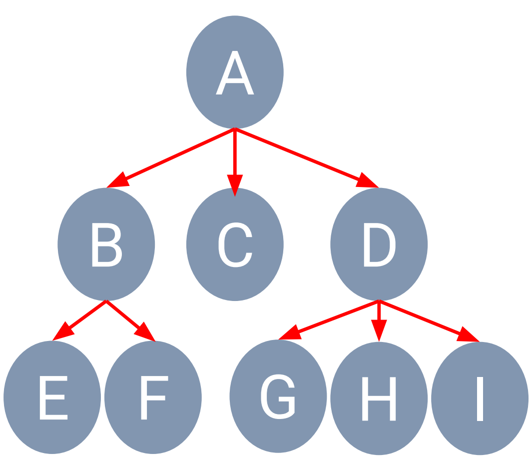


- ***Root*** Node: *A*
- ***Parents***: *A*, *B*, and *D*
- ***Children***: *B*, *E*, *F*, *C*, *D*, *G*, *H* and *I*
- ***Siblings***: {*B*, *C*, *D*}, {*E*, *F*}, {*G*, *H*, *I*}
- ***Leaf*** Nodes: *C*, *E*, *F*, *G*, *H* and *I*
- Degree of Tree is 8
- Degree of Node D is 3
- Height of tree is 3

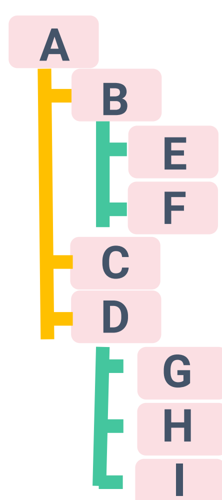
Types of Trees



Implementation of Trees



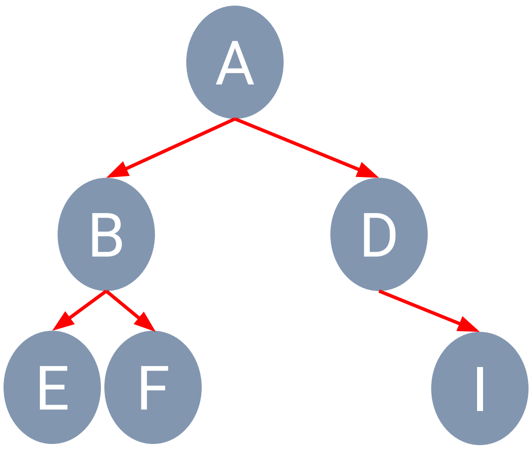
(a) General Form



(b) Tab Form

(c) Set Form

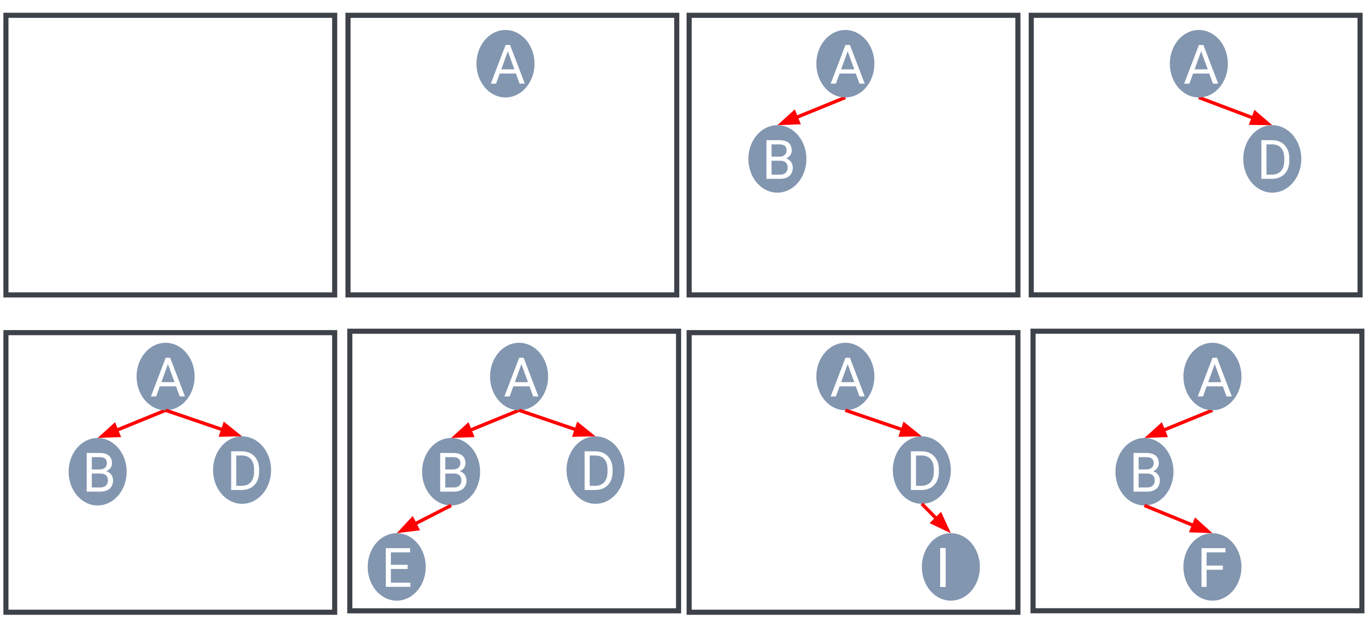
Binary Trees



- A **binary tree** is a tree in which any node can have only two children at maximum.
  - In other words, a node can have either zero, one, or two subtrees.
- Each child node is designated as being either a **left child** or **right child**.

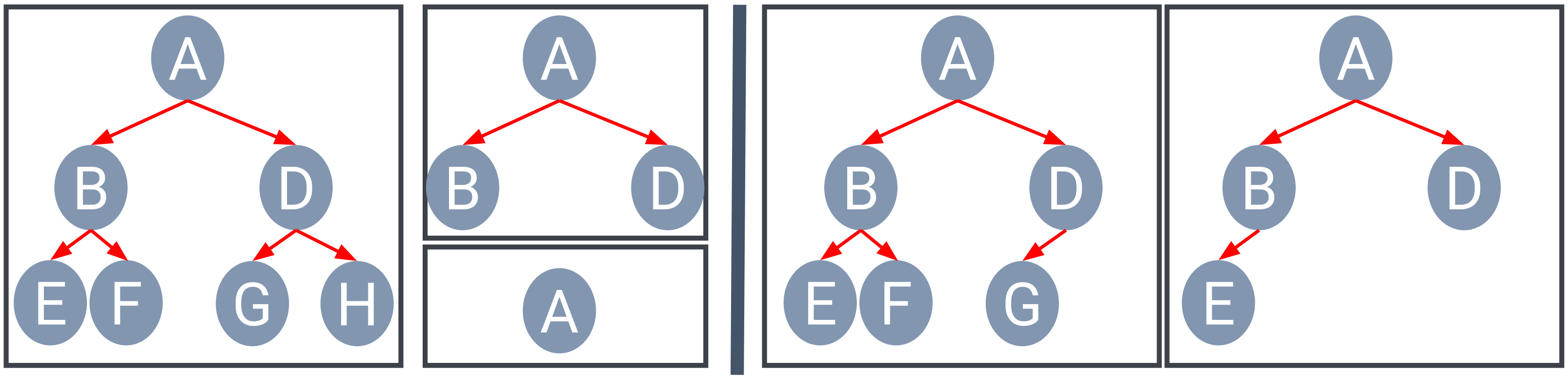
[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

Binary Trees



Types of Binary Trees

- A binary tree is nearly complete when every level, except the last, is completely filled and all leaf nodes are as far left as possible.



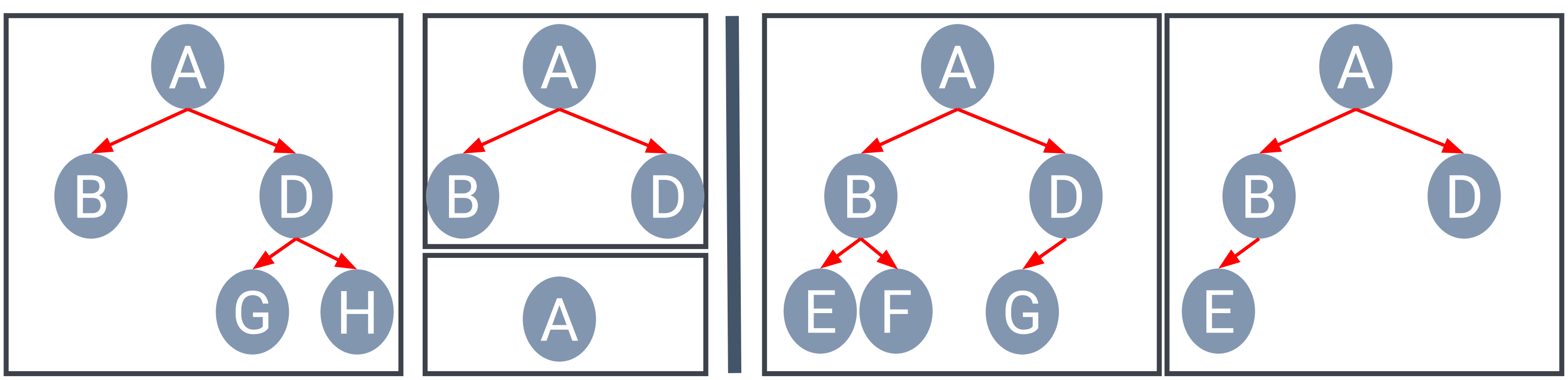
(a) *Complete/Perfect* Binary Tree

(b) *Nearly Complete* Binary Tree at level 2

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Types of Binary Trees

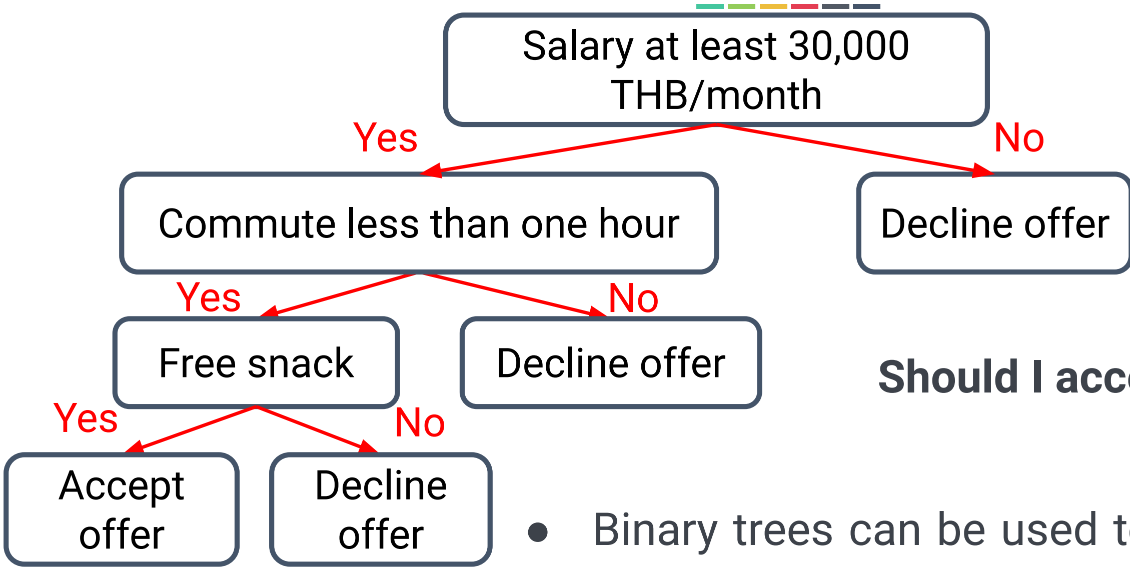
- A binary tree is proper or full when every node has zero or two children.



(a) *Proper/Full* Binary Tree
Each node has either 0 or 2 children

(b) *Improper* Binary Tree

Examples of Binary Trees

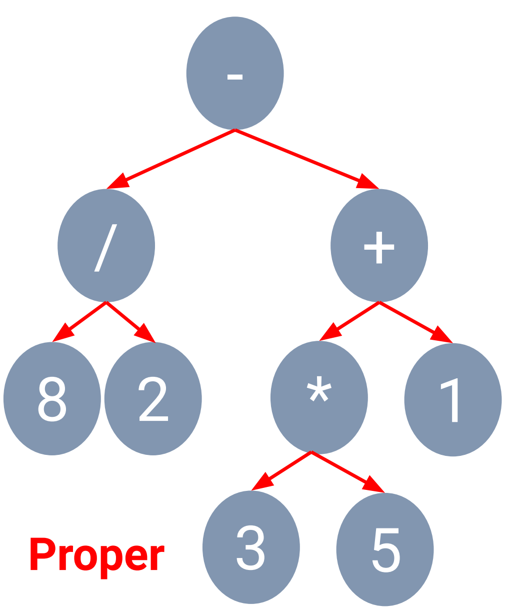


Proper

Should I accept a job offer?

- Binary trees can be used to represent a number of different outcomes resulting from a series of yes/no questions. Such binary trees is known as **decision trees**.

Examples of Binary Trees

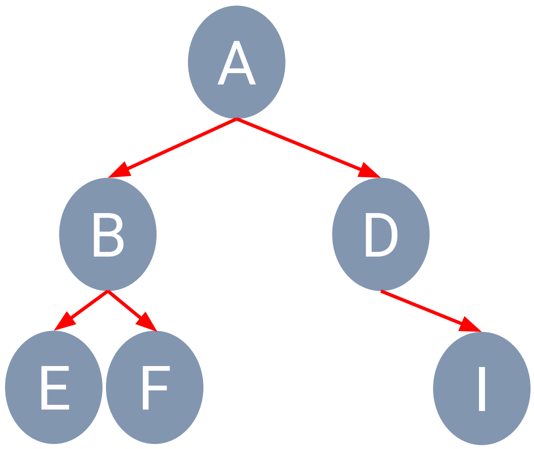


Proper

- Binary trees can be used to represent an **arithmetic expression**.
- **Leaves** are associated with **variables** or **constants**.
- **Root** and **Internal** nodes are associated with **operators**.
- **Subtree** is a sub-expression.

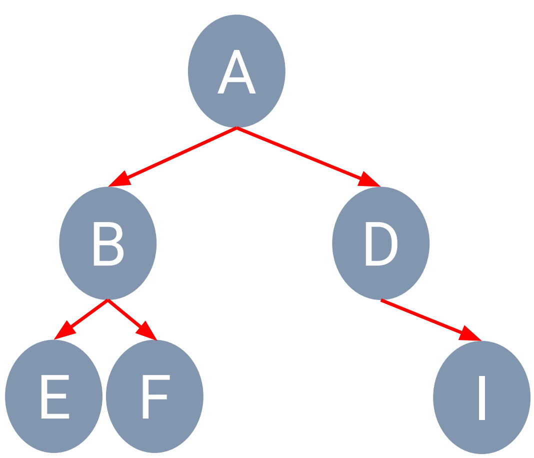
Expression: (8/2) - ((3*5)+1)

Recursive Binary Trees



- A **binary tree** is either empty or consists of:
  - A root node that stores an element
  - A binary left subtree (possibly empty)
  - A binary right subtree (possibly empty)

Binary Trees Operations

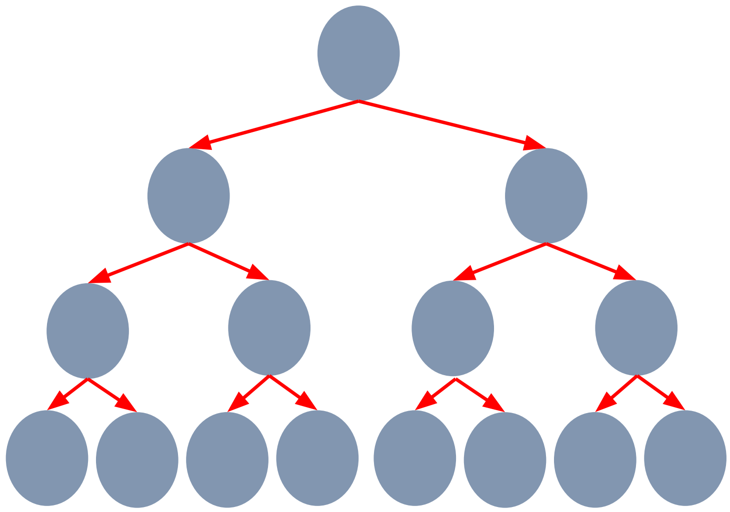


- A **binary tree** has three

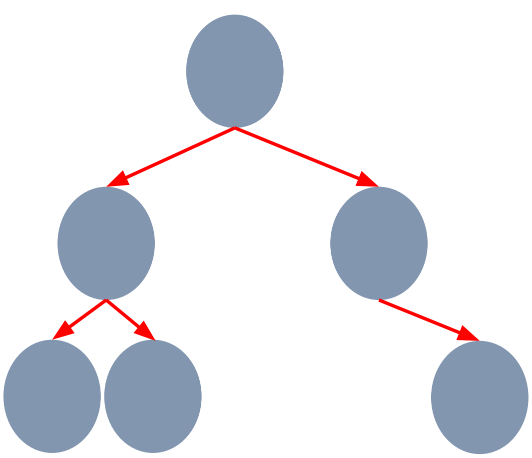
[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

Binary Trees' Properties

	Nodes
Level 0	1
Level 1	2
Level 2	4
Level 3	8
Level d	2^d

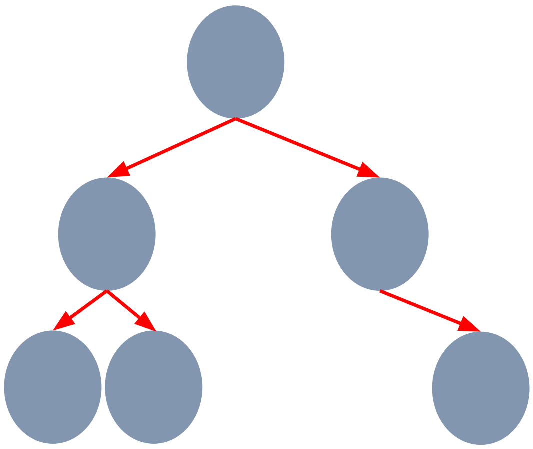


Binary Trees' Properties



- n denotes the number of nodes
- h denotes the height of tree
- **Maximum height** of binary trees, $h_{max} = n$
- **Minimum height** of binary trees, $h_{min} = \log_2 n + 1$
- **Minimum number** of nodes in a tree, $n_{min} = h$
- **Maximum number** of nodes in a tree, $n_{max} = 2^h - 1$

Binary Trees Implementation



Arrays (List of Lists)

or

Doubly Linked Lists

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Outline

General Trees:

- Definition and examples
- Elements of Tree Structure

Binary Trees:

- Definition, examples, properties, and types
- Applications (i.e. Expression Tree)

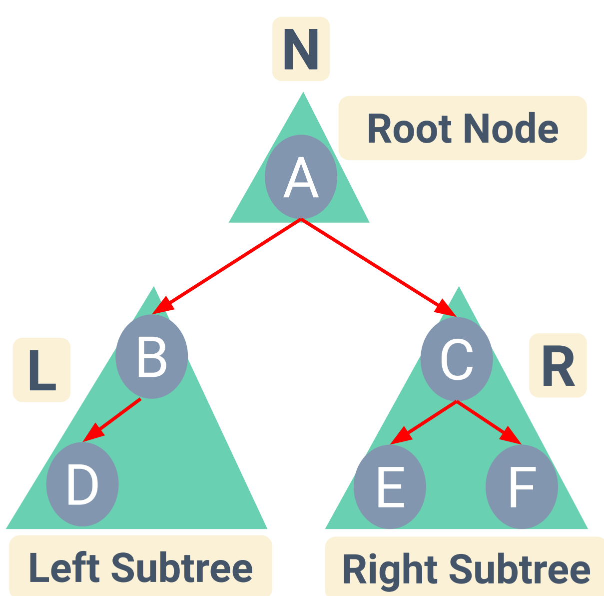
Tree traversal algorithms:

- Depth-first Traversal (Preorder, postorder, and inorder)
- Breadth-first Traversal

Tree Traversal

- A **traversal** of a tree is a method to access all the tree nodes.
- Two main approaches: Depth-first and Breadth-first
  - Depth-first
    - Preorder traversal - start at the root node, then the subtrees from left-to-right.
    - Postorder traversal - start at subtrees first from left-to-right, then the root.
    - Inorder traversal - visiting nodes from left-to-right.
  - Breadth-first - visit all the nodes at depth d before moving to depth d+1.

Depth-first Traversal



- **Pre**order Traversal

  N L R

- **In**order Traversal

  L N R

- **Post**order Traversal

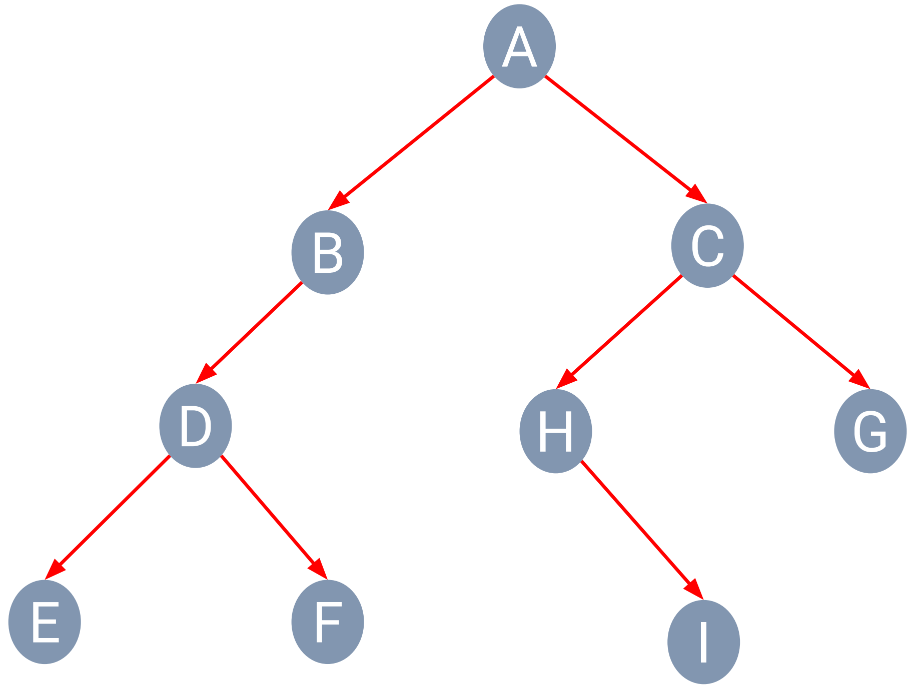
  L R N

Preorder Traversal

- **Preorder Traversal**

  N **L** **R**

- **Output**



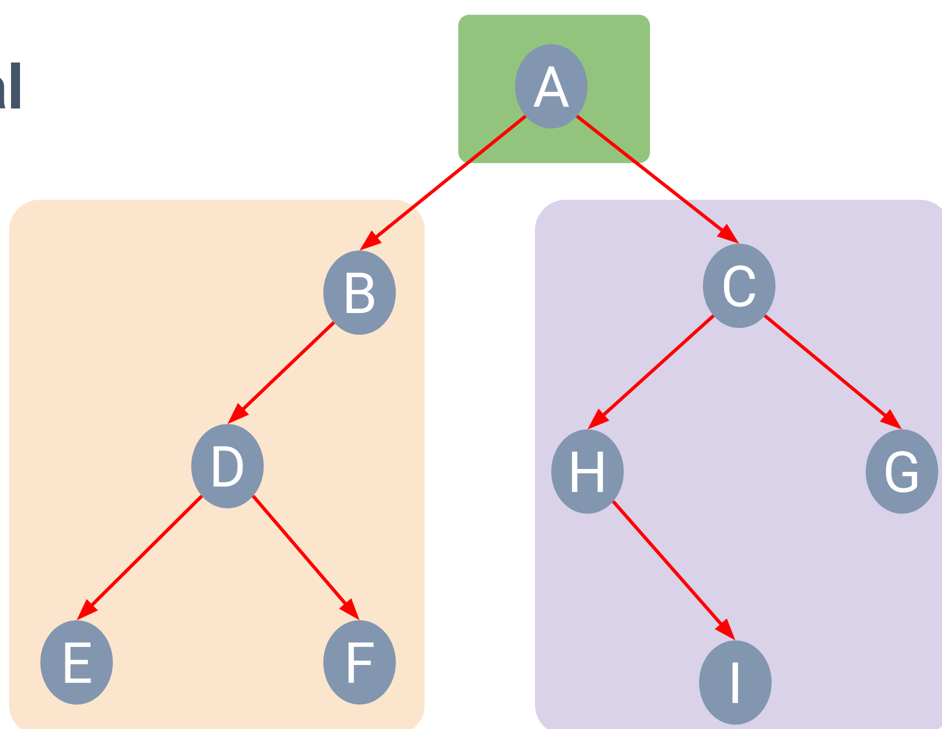
Preorder Traversal

- **Preorder Traversal**

  N **L** **R**

- **Output**

A



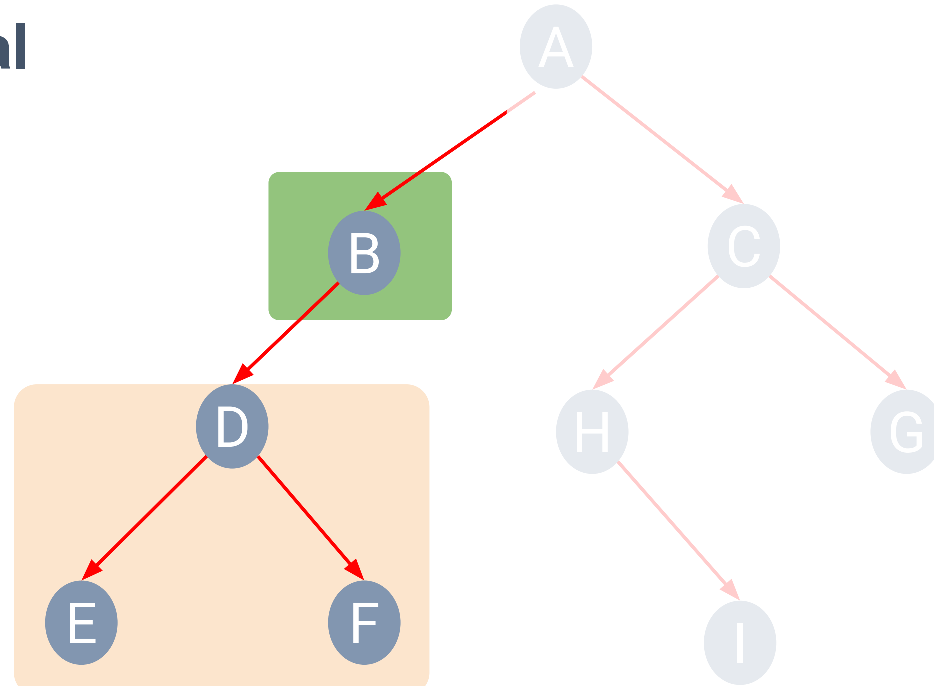
Preorder Traversal

- **Preorder Traversal**

  N L R

- **Output**

  A B



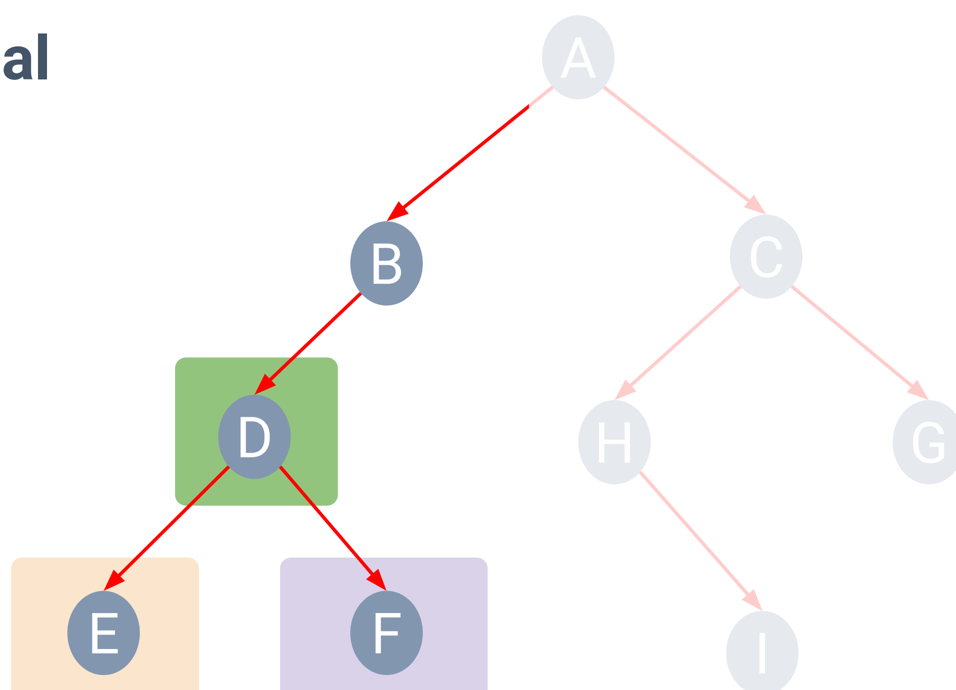
Preorder Traversal

- **Preorder Traversal**

  N L R

- **Output**

  A B D E F



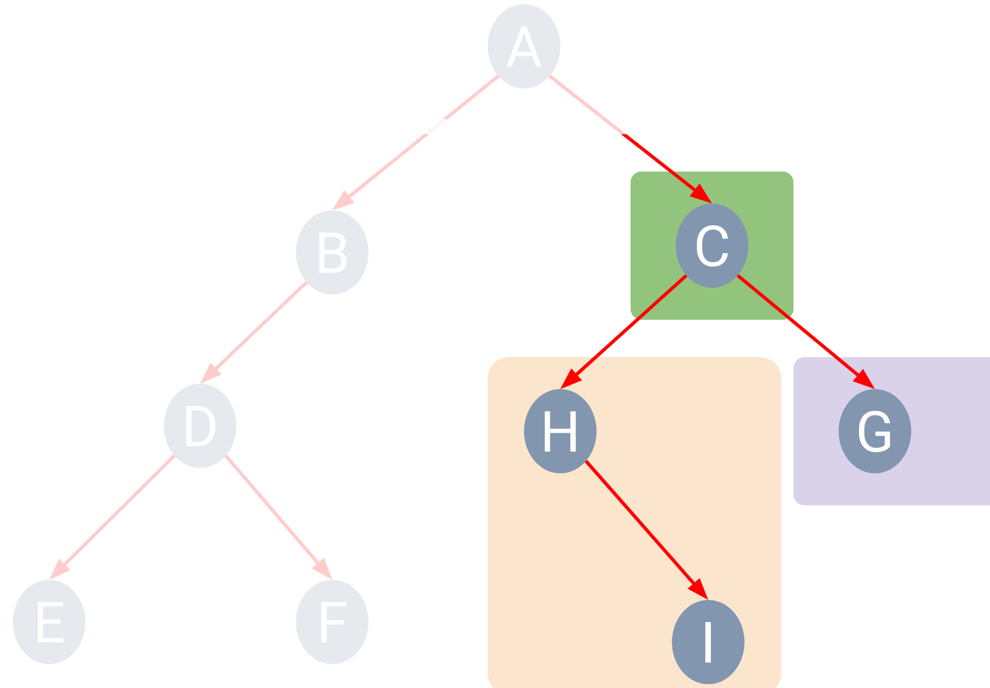
Preorder Traversal

- **Preorder Traversal**

  N L R

- **Output**

  A B D E F



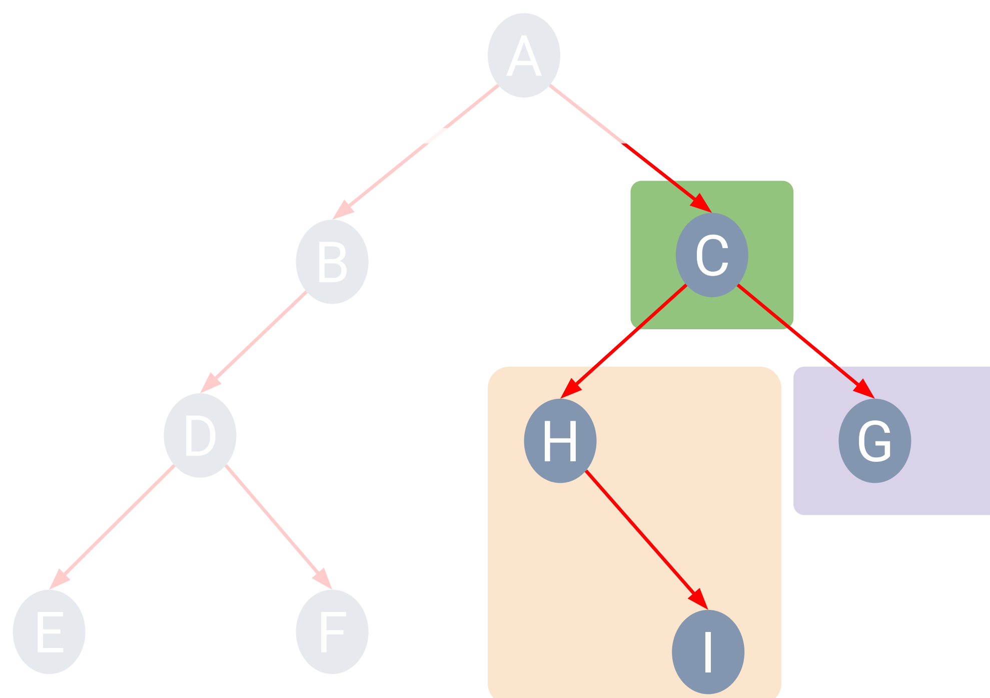
Preorder Traversal

- **Preorder Traversal**

  N L R

- **Output**

  A B D E F
  C



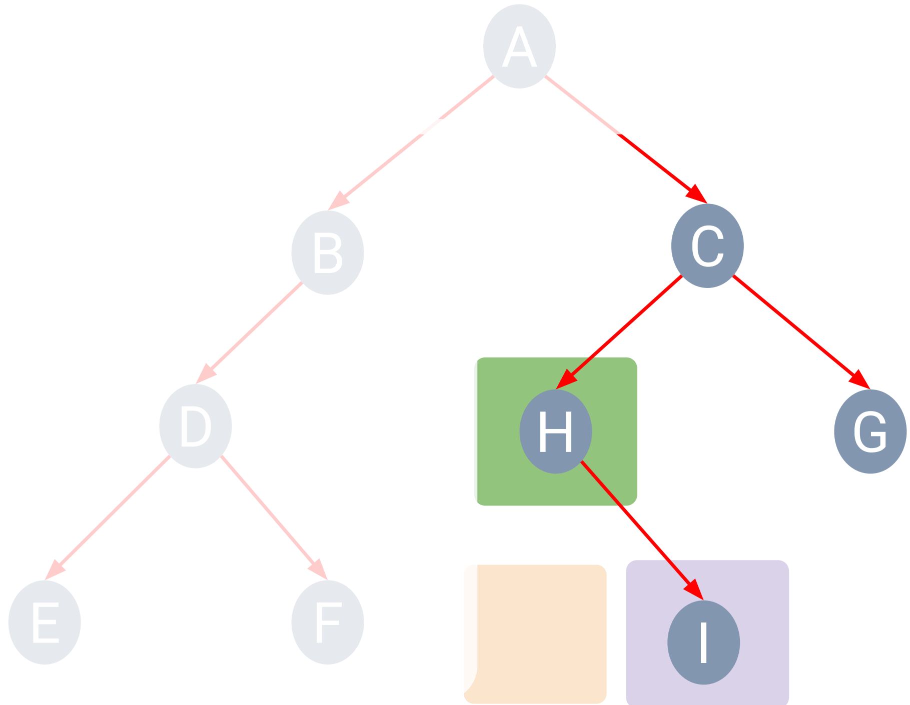
Preorder Traversal

- **Preorder Traversal**

N L R

- **Output**

A B D E F
C H I



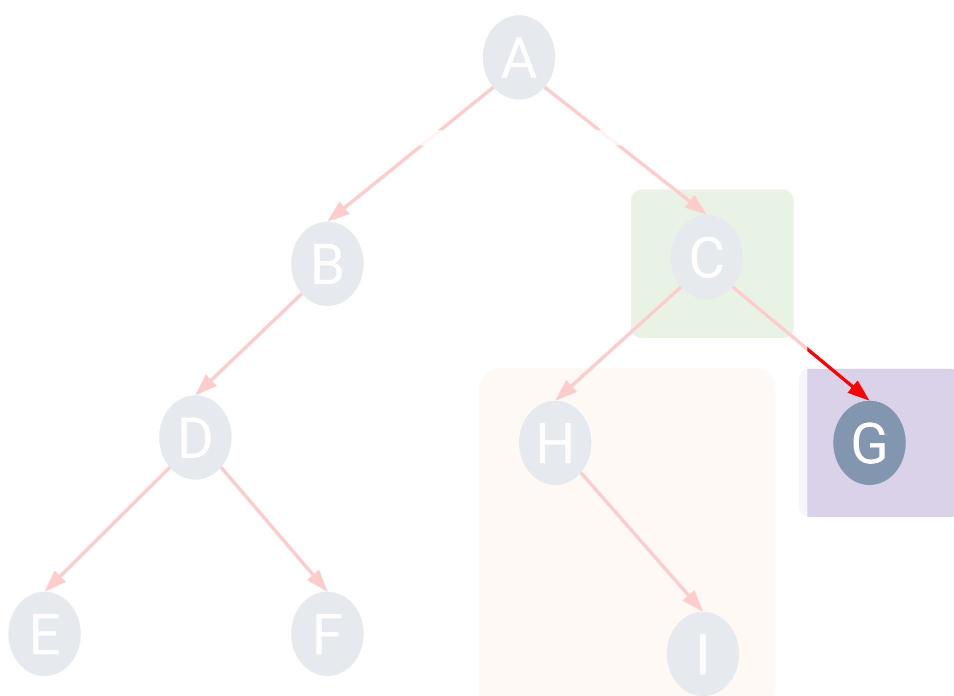
Preorder Traversal

- **Preorder Traversal**

N L R

- **Output**

A B D E F
C H I G

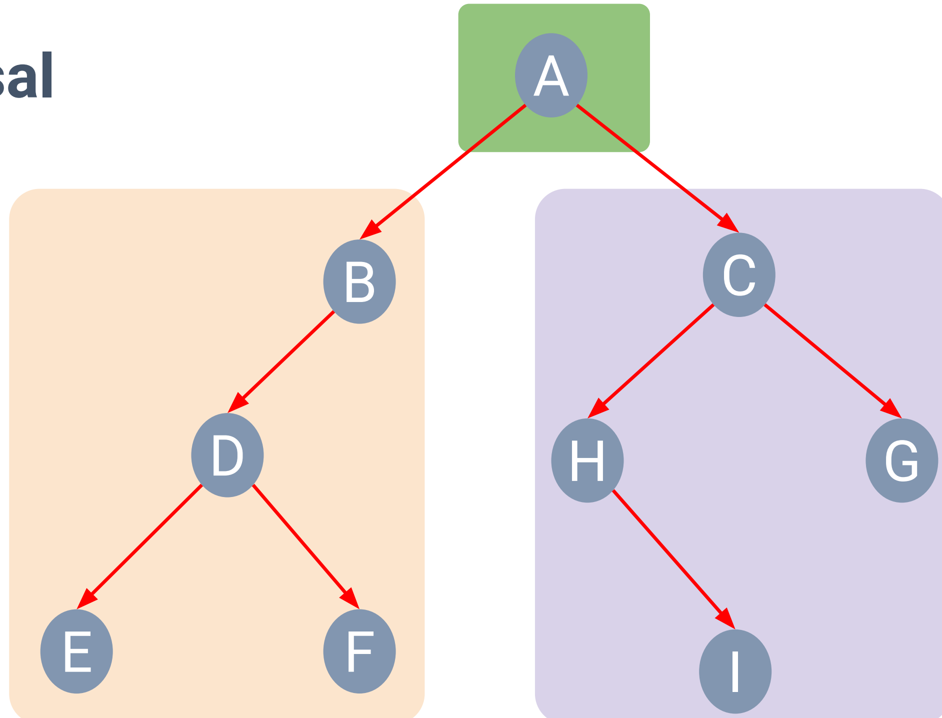


Postorder Traversal

- **Postorder Traversal**

  L R N

- **Output**



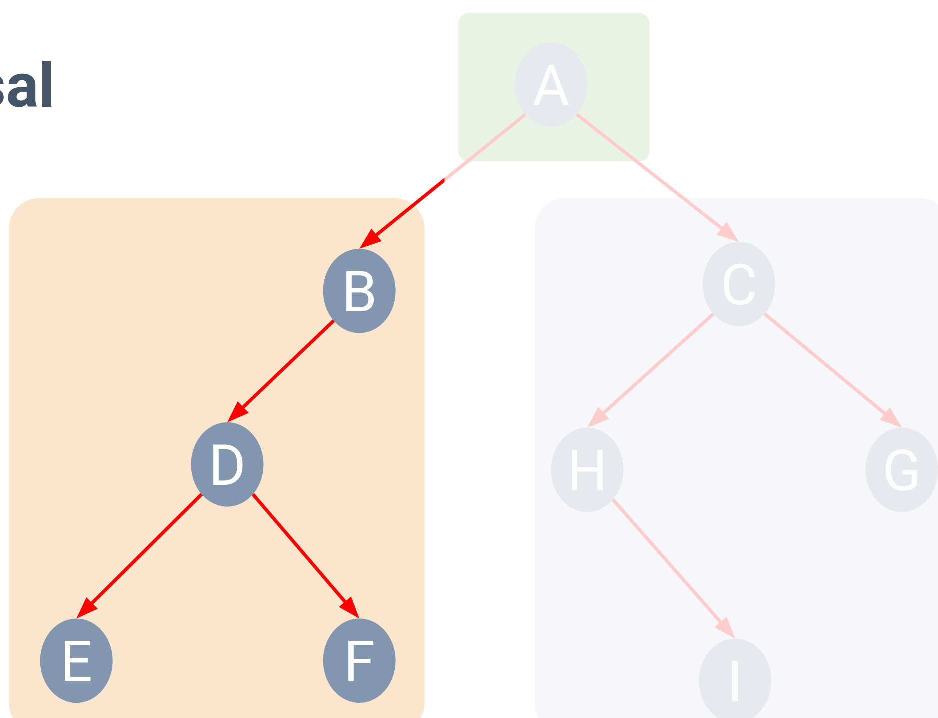
Postorder Traversal

- **Postorder Traversal**

  L R N

- **Output**

  E F D B



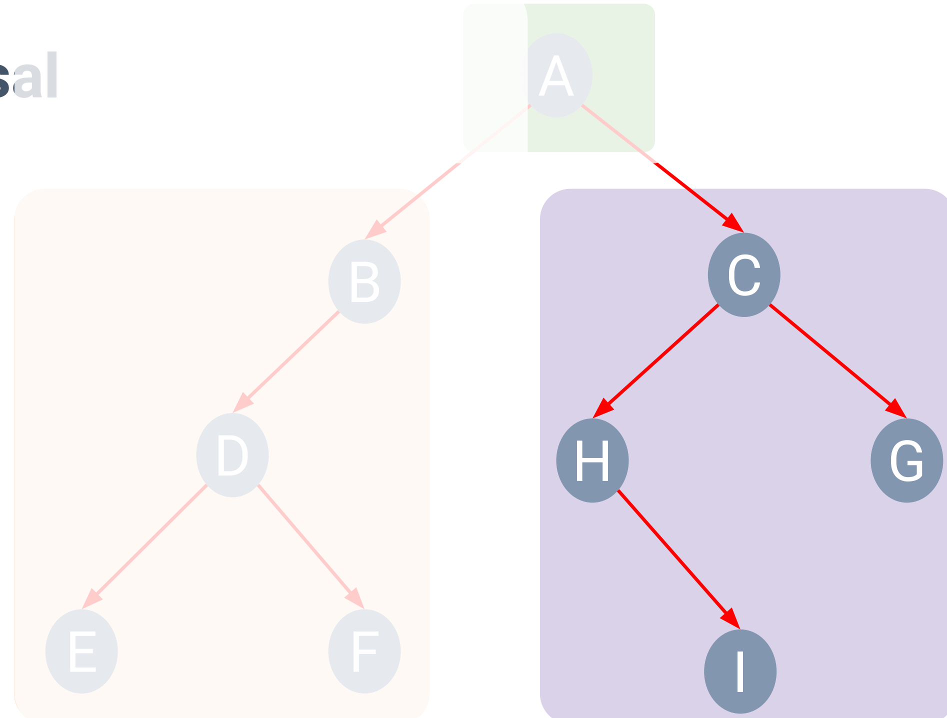
Postorder Traversal

- **Postorder Traversal**

L **R** **N**

- **Output**

E F D B
I H G C



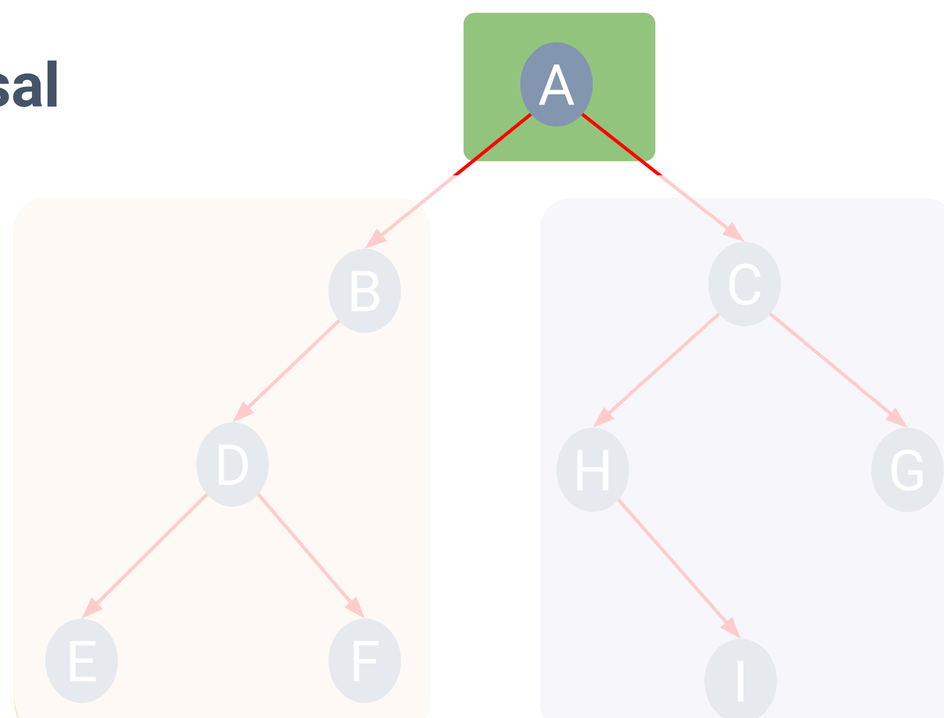
Postorder Traversal

- **Postorder Traversal**

L **R** **N**

- **Output**

E F D B
I H G C
A

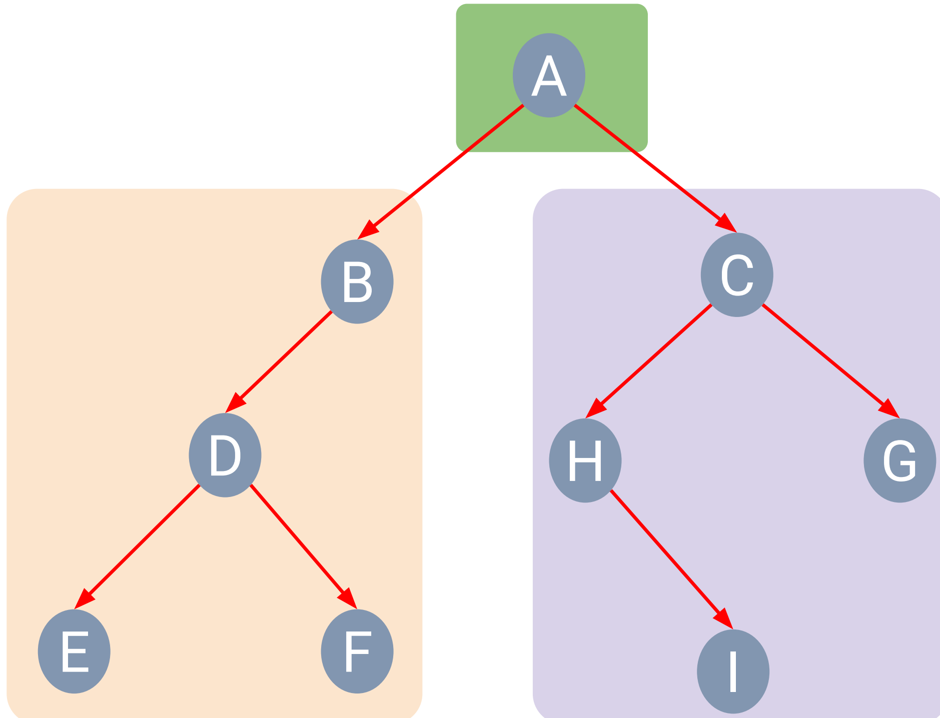


Inorder Traversal

- **Inorder Traversal**

L N R

- **Output**



Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

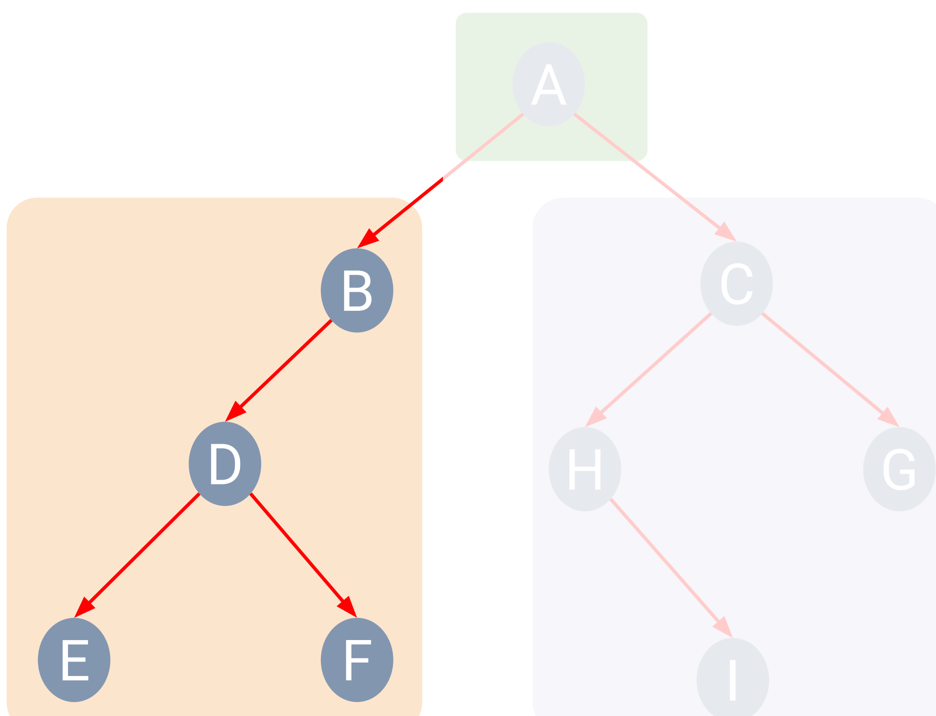
Inorder Traversal

- **Inorder Traversal**

L N R

- **Output**

E D F B



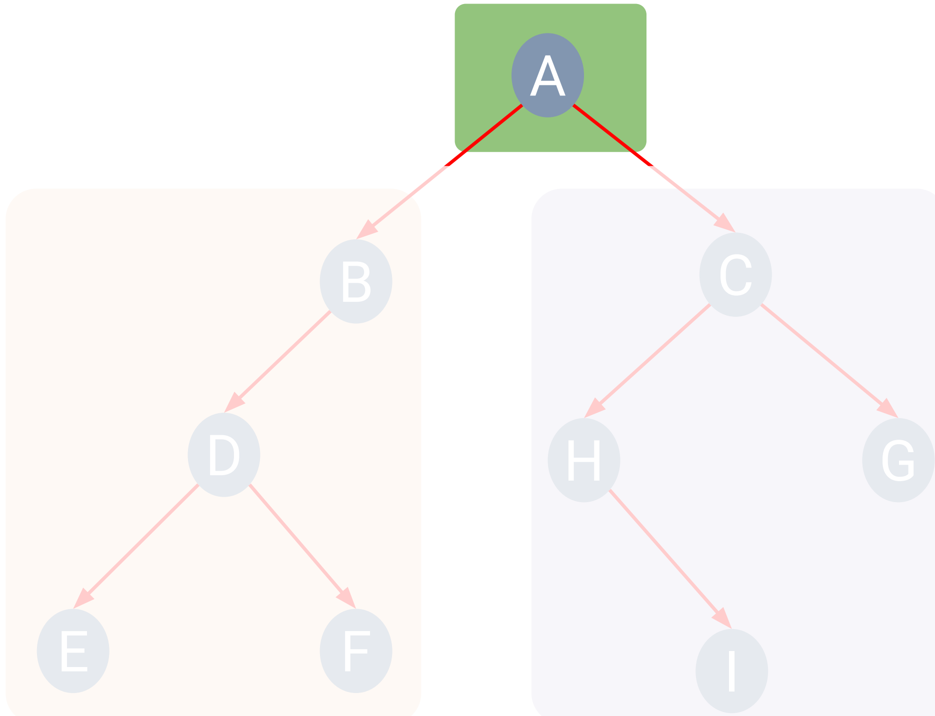
Inorder Traversal

- **Inorder Traversal**

L N R

- **Output**

E D F B A



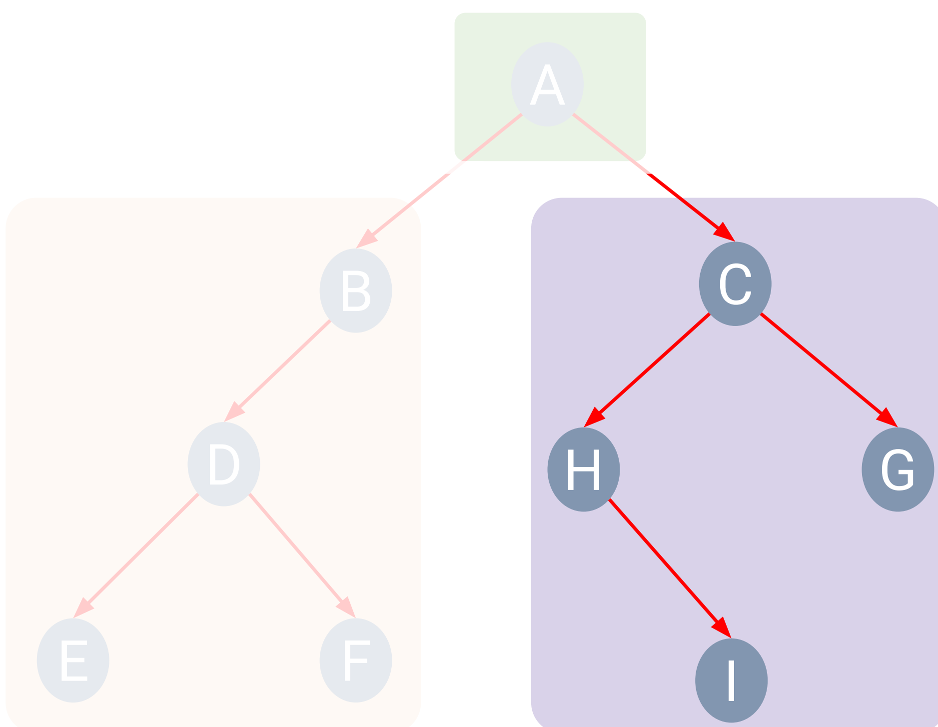
Inorder Traversal

- **Inorder Traversal**

L N R

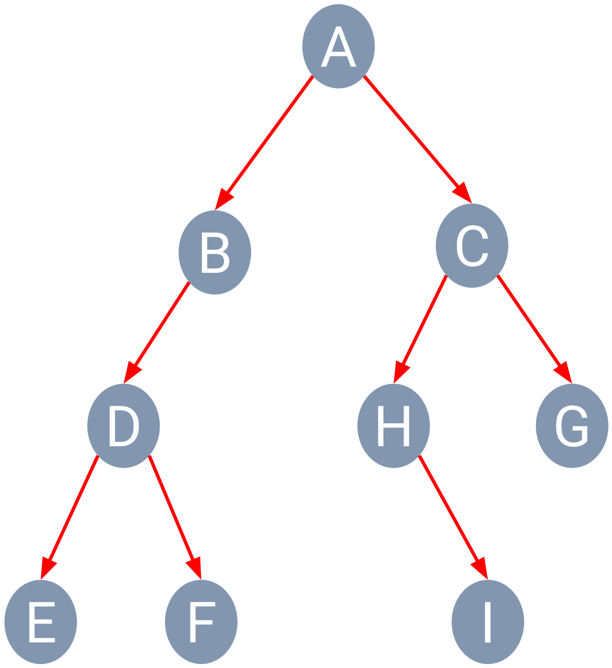
- **Output**

E D F B A
H I C G



Depth-first Traversal

Traversal of tree: *O*(?)



- **Pre**order Traversal N L R

A B D E F C H I G

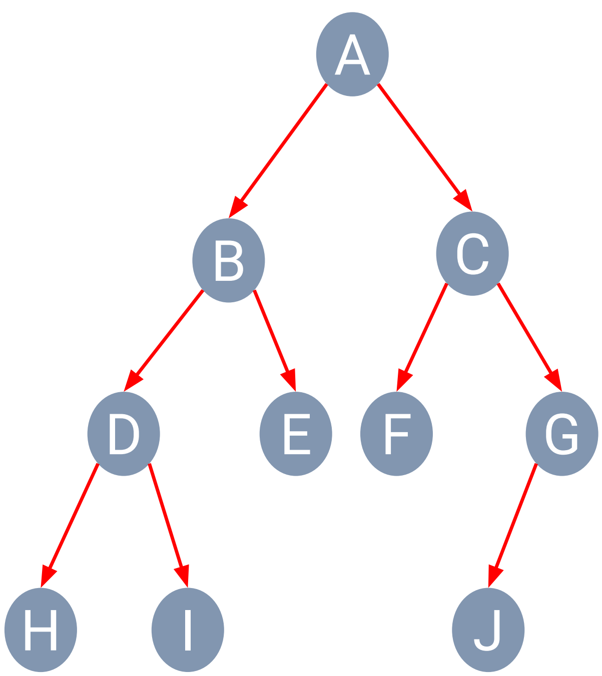
- **In**order Traversal L N R

E D F B A H I C G

- **Post**order Traversal L R N

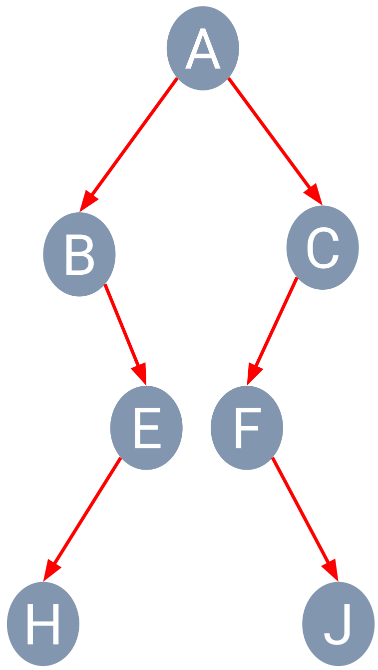
E F D B I H G C A

Depth-first Traversal Exercise 1



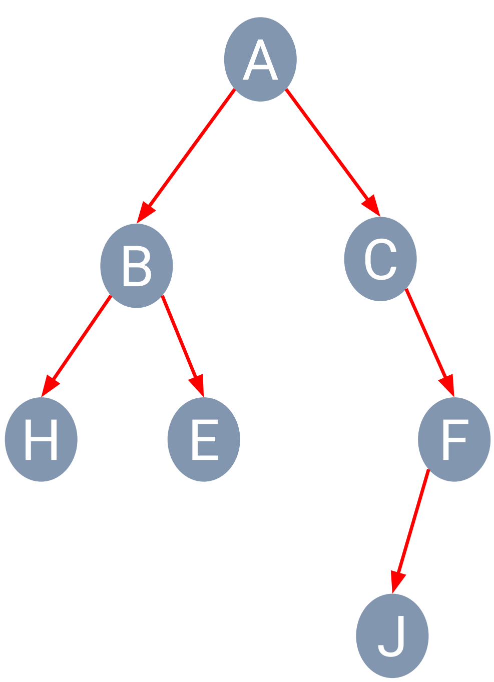
- **Pre**order Traversal N L R
- **In**order Traversal L N R
- **Post**order Traversal L R N

Depth-first Traversal Exercise 2



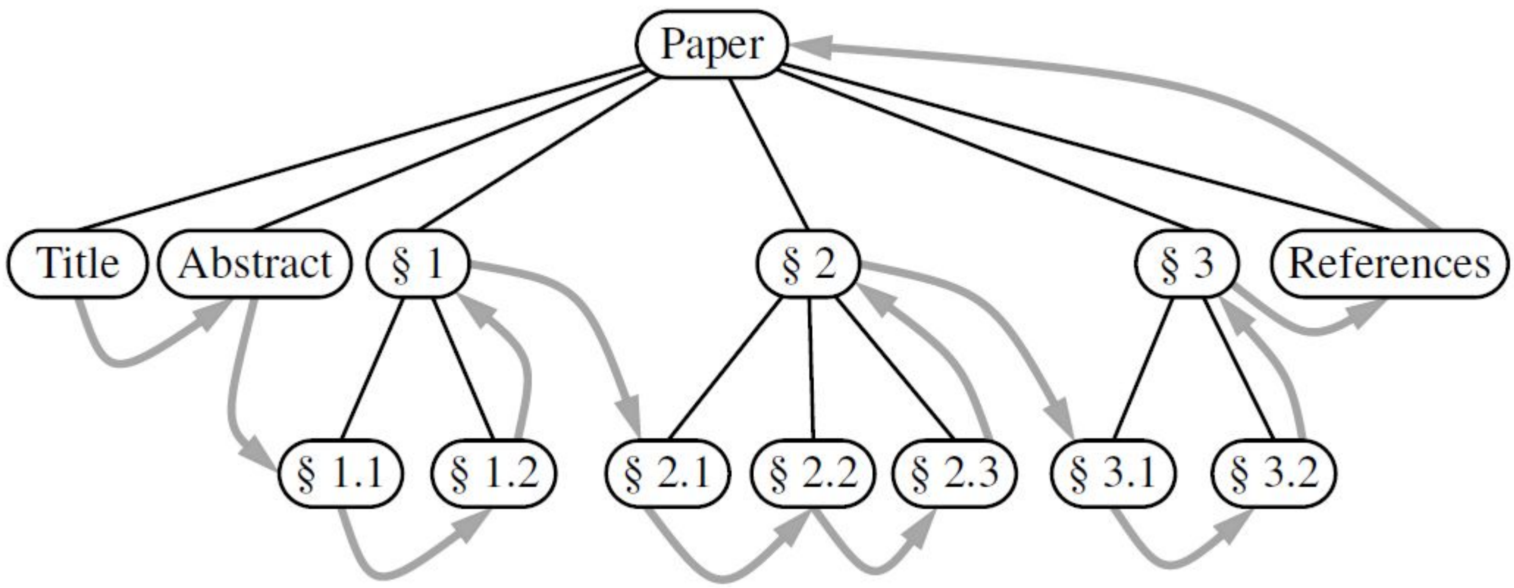
- **Pre**order Traversal N L R
- **In**order Traversal L N R
- **Post**order Traversal L R N

Depth-first Traversal Exercise 3



- **Pre**order Traversal N L R
- **In**order Traversal L N R
- **Post**order Traversal L R N

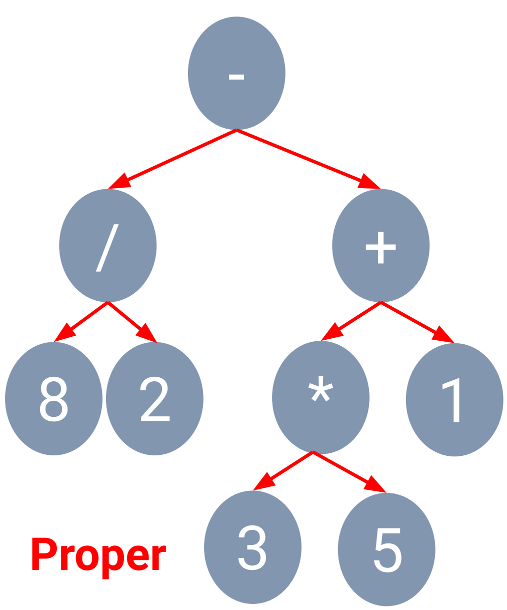
Depth-first Traversal Application



- **Preorder** traversal of an ordered tree - Table of contents.

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Depth-first Traversal Application

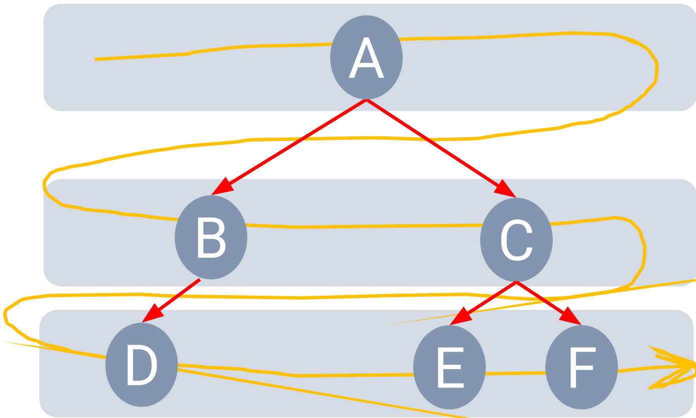


Proper

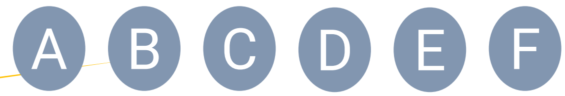
- Binary trees can be used to represent an **arithmetic expression**.
- The **inorder** traversal visits node in a consistent order with the expression.

Expression: (8/2) - ((3*5)+1)

Breadth-first Traversal

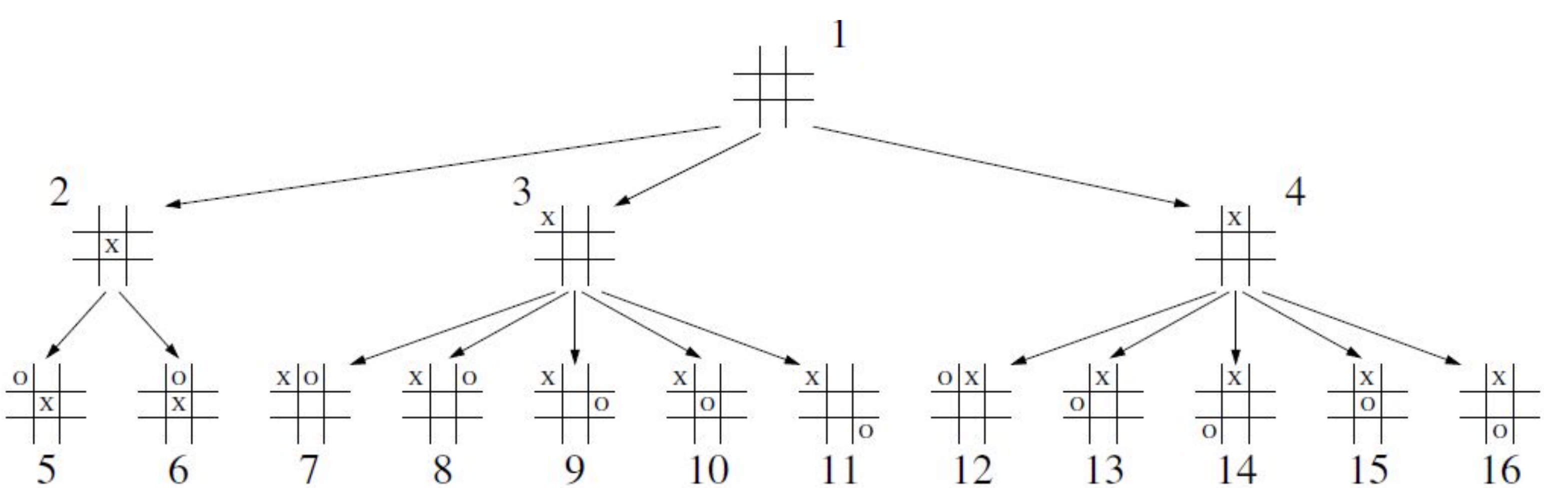


- **Output**



- Visit all the nodes at depth d before moving to depth d+1.

Breadth-first Traversal Application



- Tic-tac-toe possible choices of moves by a computer

Outline

General Trees:

- Definition and examples
- Elements of Tree Structure

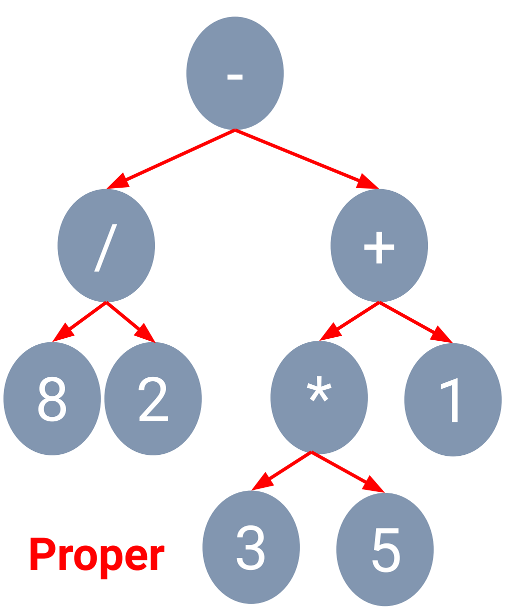
Binary Trees:

- Definition, examples, properties, and types
- Applications (i.e. Expression Tree)

Tree traversal algorithms:

- Depth-first Traversal (Preorder, postorder, and inorder)
- Breadth-first Traversal

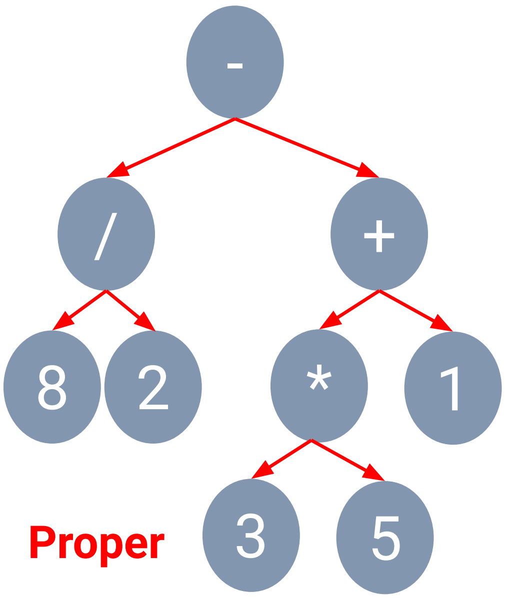
Expression Tree



- Binary trees can be used to represent an **arithmetic expression**.
- Three properties:
  - **Leaves** are associated with **variables** or **constants**.
  - **Root** and **Internal** nodes are associated with **operators**.
  - **Subtree** is a sub-expression.

Expression: (8/2) - ((3*5)+1)

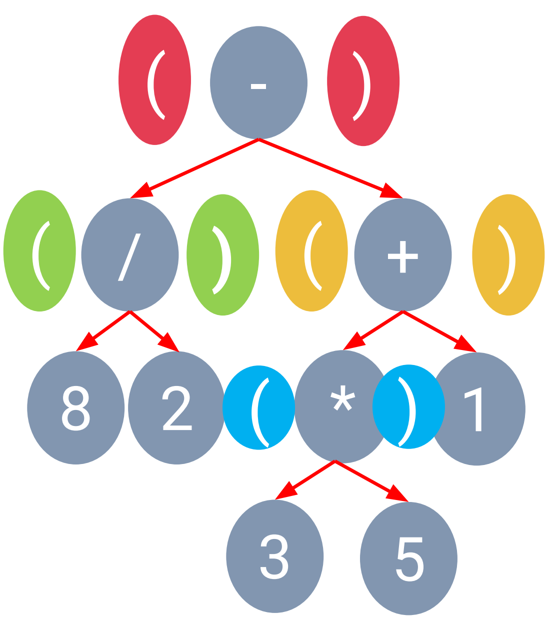
Expression Tree

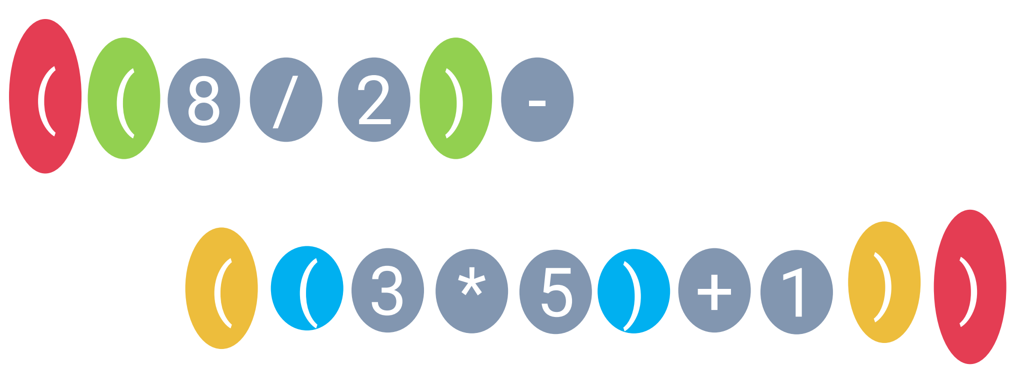


- Depth-First Traversals
  1. Preorder = Prefix form
     - -(/82)(+(*35)1)
  2. Inorder = Infix form
     - (8/2) - ((3*5)+1)
  3. Postorder = Postfix form
     - (82/)((35*)1+)-

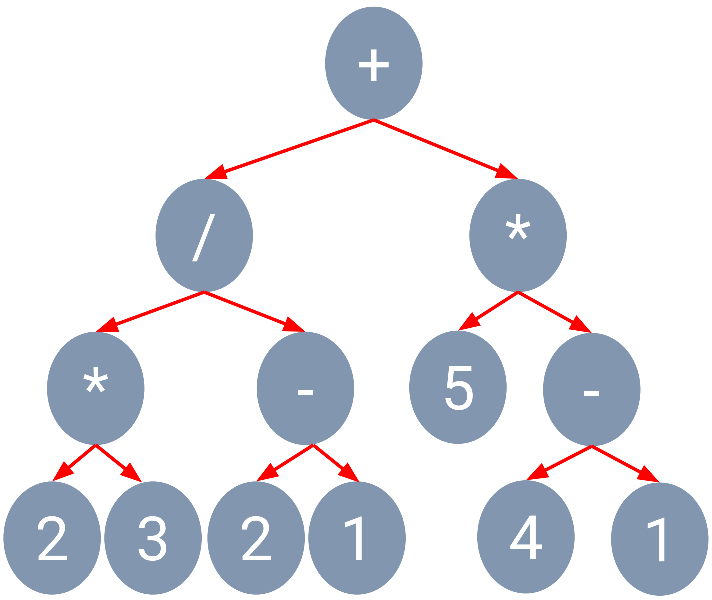
Expression: (8/2) - ((3*5)+1)

Expression Tree - Infix Form





Expression Tree Exercise 1

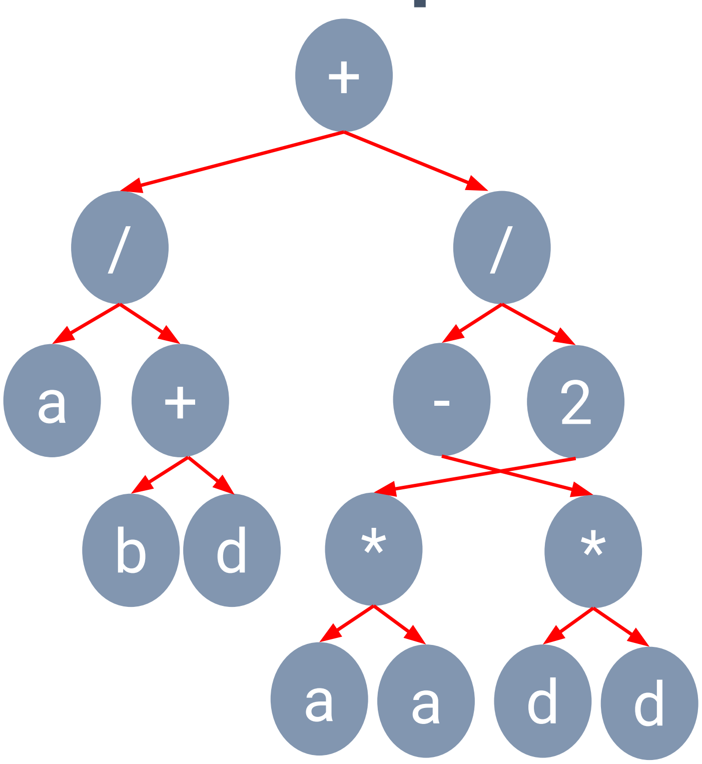


- **Preorder Traversal** N L R
- **Inorder Traversal** L N R
- **Postorder Traversal** L R N

Expression: (2*3/(2-1))+(5*(4-1))

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Expression Tree Exercise 2



- **Preorder Traversal** N L R
- **Inorder Traversal** L N R
- **Postorder Traversal** L R N

Expression: $\frac{a}{b+d}+\frac{a^2-d^2}{2}$

Individual Assignment

- Assignment#4: Linked Lists
- Due 09.00 am, Tuesday 08/09/2020.
- Submission
  - Email: sirasit@it.kmitl.ac.th
  - Paper: in classroom next week
- Can be either written by hand or typing.
- Make sure to submit on time!!
  - Late submission has penalty on the score.
- If unable to submit on time for reasonable reasons, let me know asap.