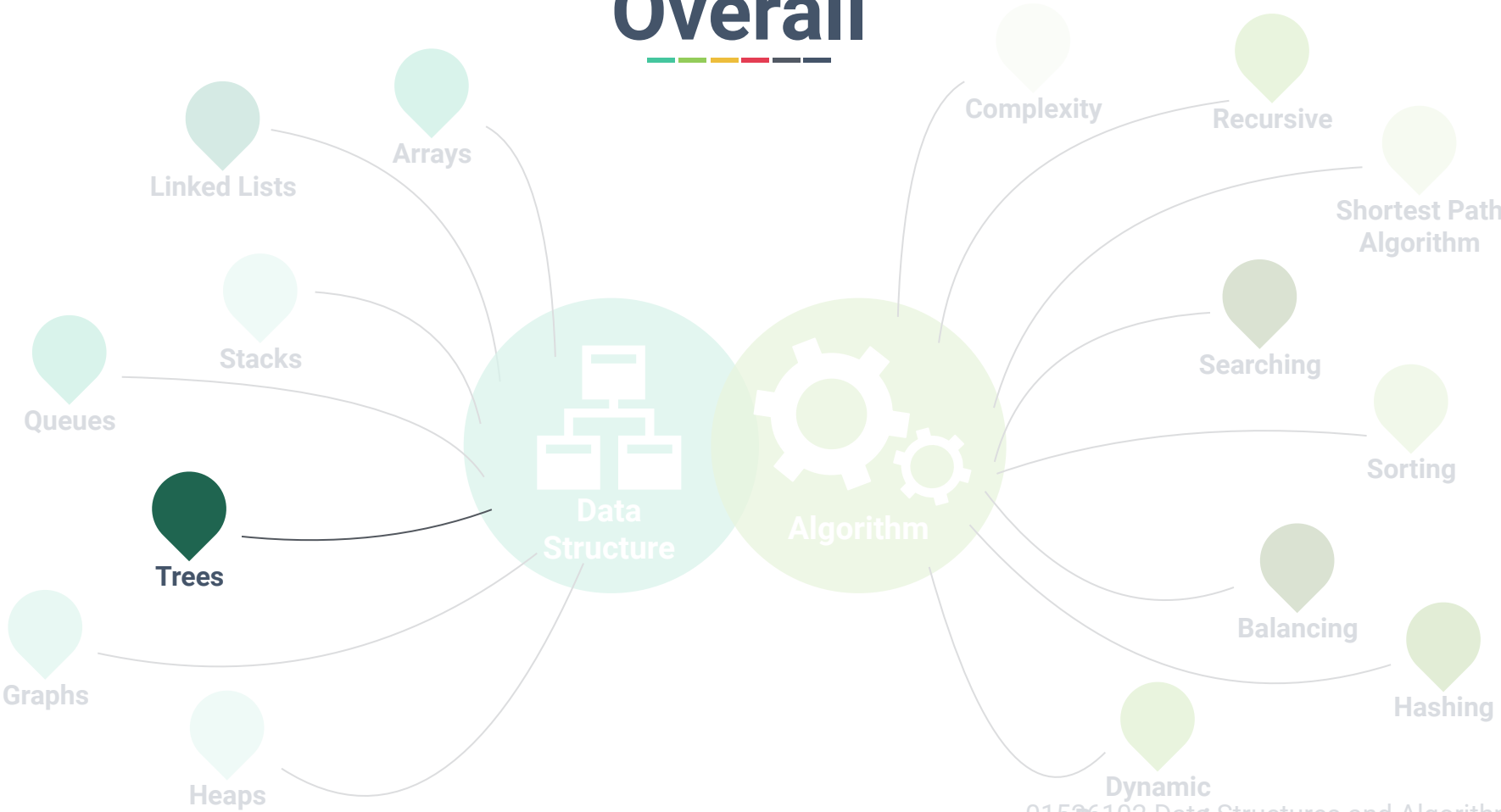
Chapter 6: Trees

Dr. Sirasit Lochanachit

Overall


Linked Lists
Arrays
Stacks
Queues
Trees
Graphs
Heaps
Data Structure
Algorithm
Complexity
Recursive
Shortest Path Algorithm
Searching
Sorting
Balancing
Hashing
Dynamic Programming

Abstract Data Type

Linear:

- Arrays, Stacks, Queues, and Linked Lists
- Operations: push, pop, enqueue, dequeue
- Algorithms: Searching and sorting, insertion, deletion

Non-Linear:

- Trees and Graphs
- Algorithms: Traversal, Insertion and Deletion, Balancing, and etc.

Today's Outline

General Trees:

- Definition and examples
- Elements of Tree Structure

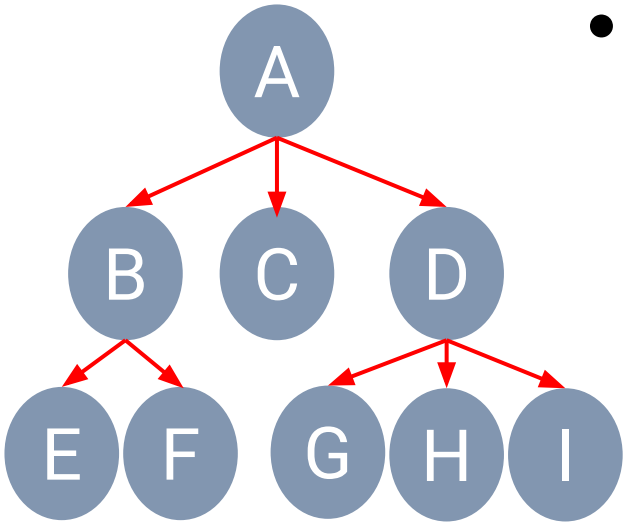
Binary Trees:

- Definition, examples, properties, and types

Tree traversal algorithms:

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

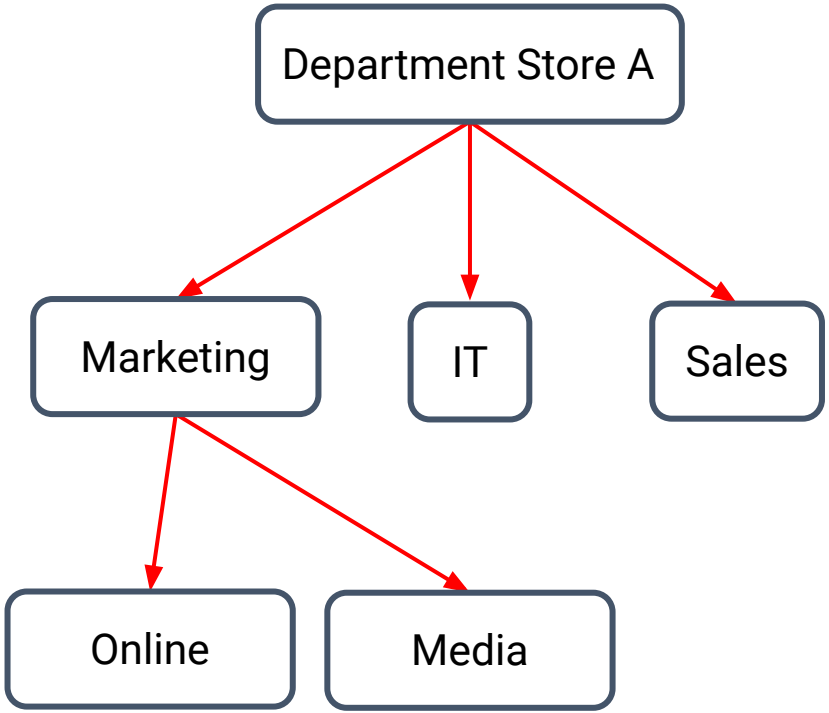
What is a Tree?

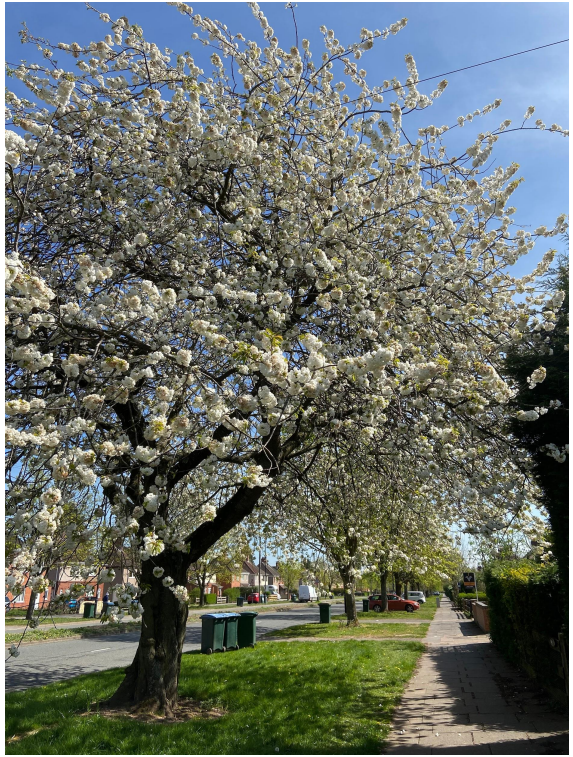


- A **tree** stores elements in a hierarchical structure.

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

Tree Example

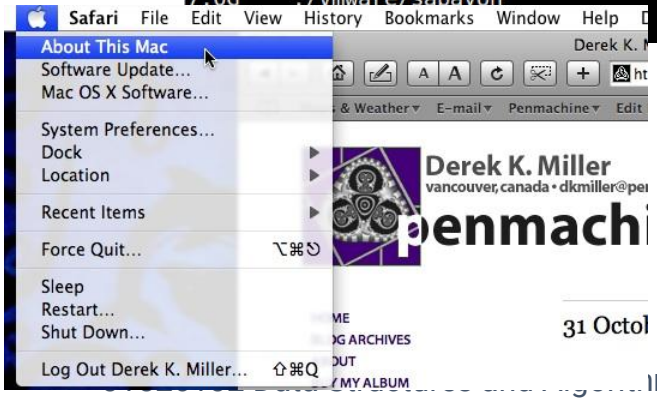




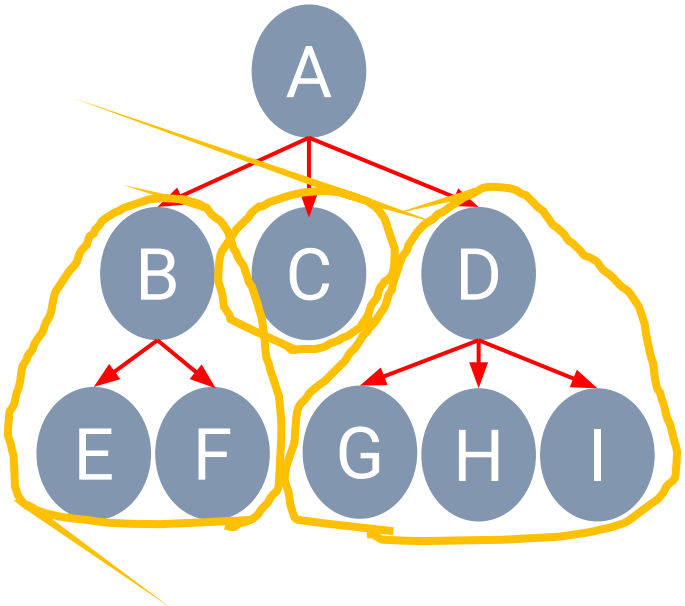
Tree Applications

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

```
$ du -Sh | sort -rh | head -n 15
35G     ./vmware/iso
19G     ./vmware/ubuntu32
19G     ./Downloads
12G     ./vmware/xubuntu
12G     ./vmware/ubuntu13.10
12G     ./vmware/centos6.4
11G     ./vmware/ubuntu
11G     ./vmware/kfedora
9.8G    ./vmware/node
9.5G    ./vmware/fedora
7.7G    ./vmware/ubuntu13.04
7.6G    ./vmware/centos_server
7.4G    ./vmware/kdebian
7.1G    ./vmware/pclinuxos
7.0G    ./vmware/sabayon
```



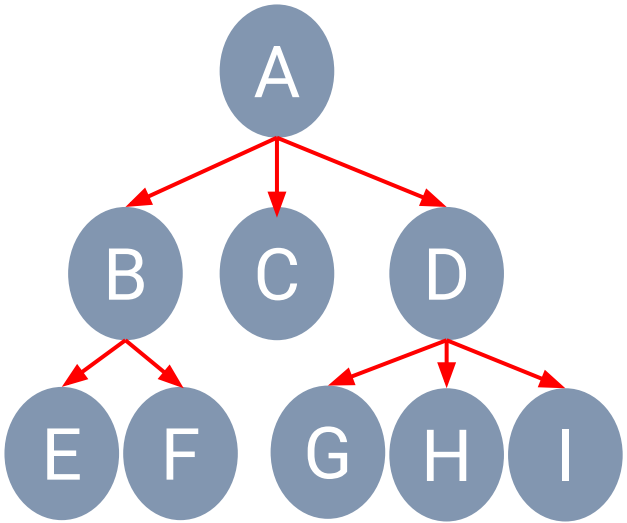
Formal Definition



- A **tree** is a collection of nodes that store elements with a **parent-child** relationship.

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

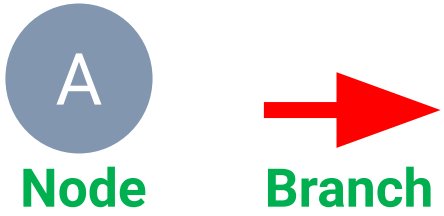
Formal Definition

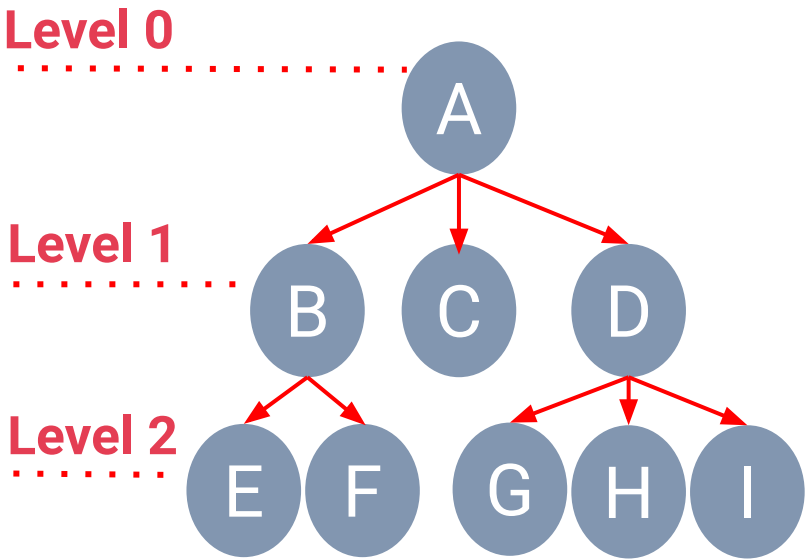


- **Siblings**
- **External** or **leaf** node
- **Internal**
- **Ancestor**
- **Descendant**

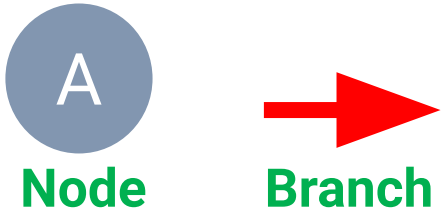
[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

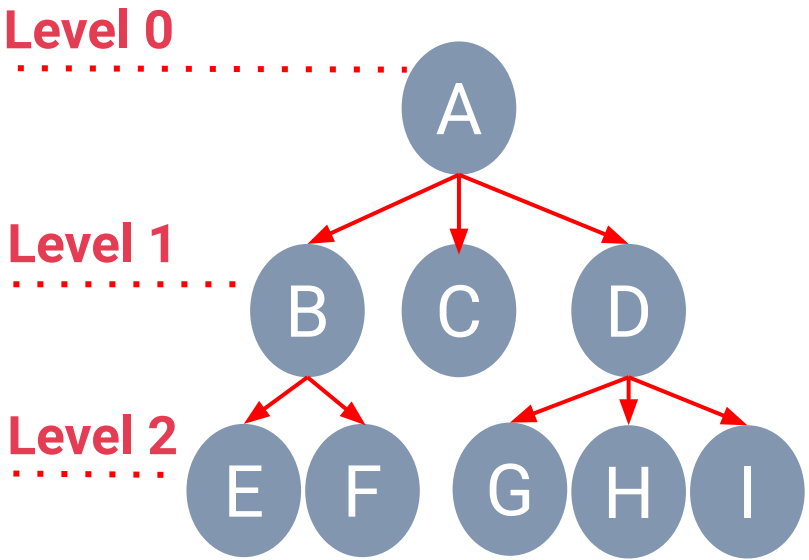
Basic Elements of Tree Structure





Basic Elements of Tree Structure

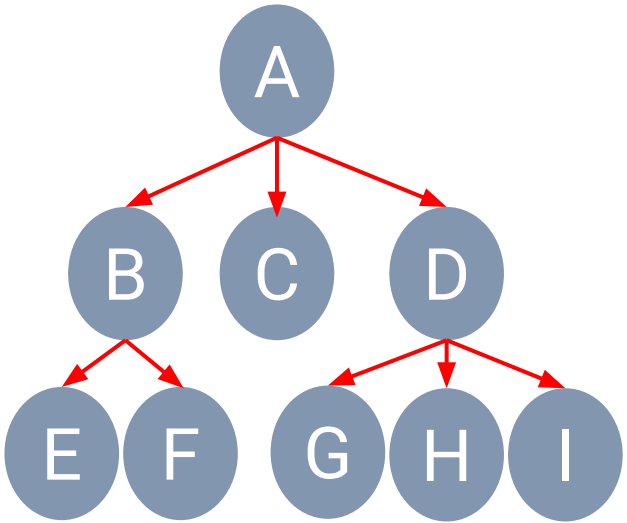




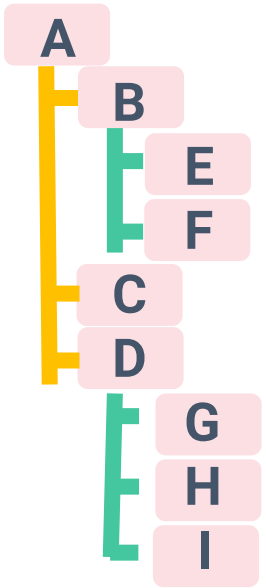
Implementation of Trees

Tree

Other Implementation of Trees



(a) General Form

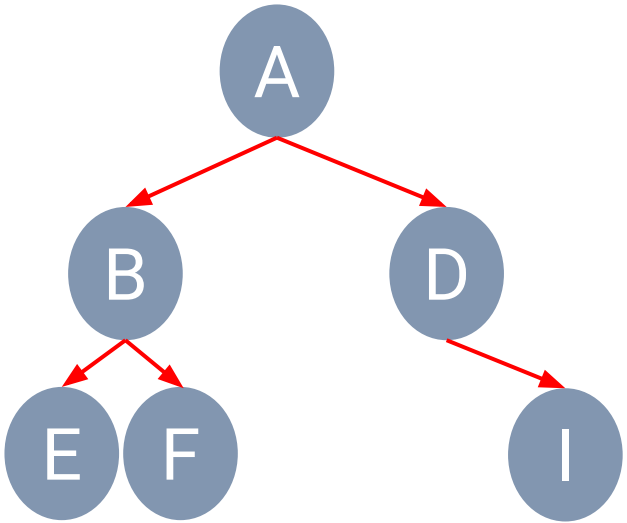


(b) Tab Form

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

(c) Set Form

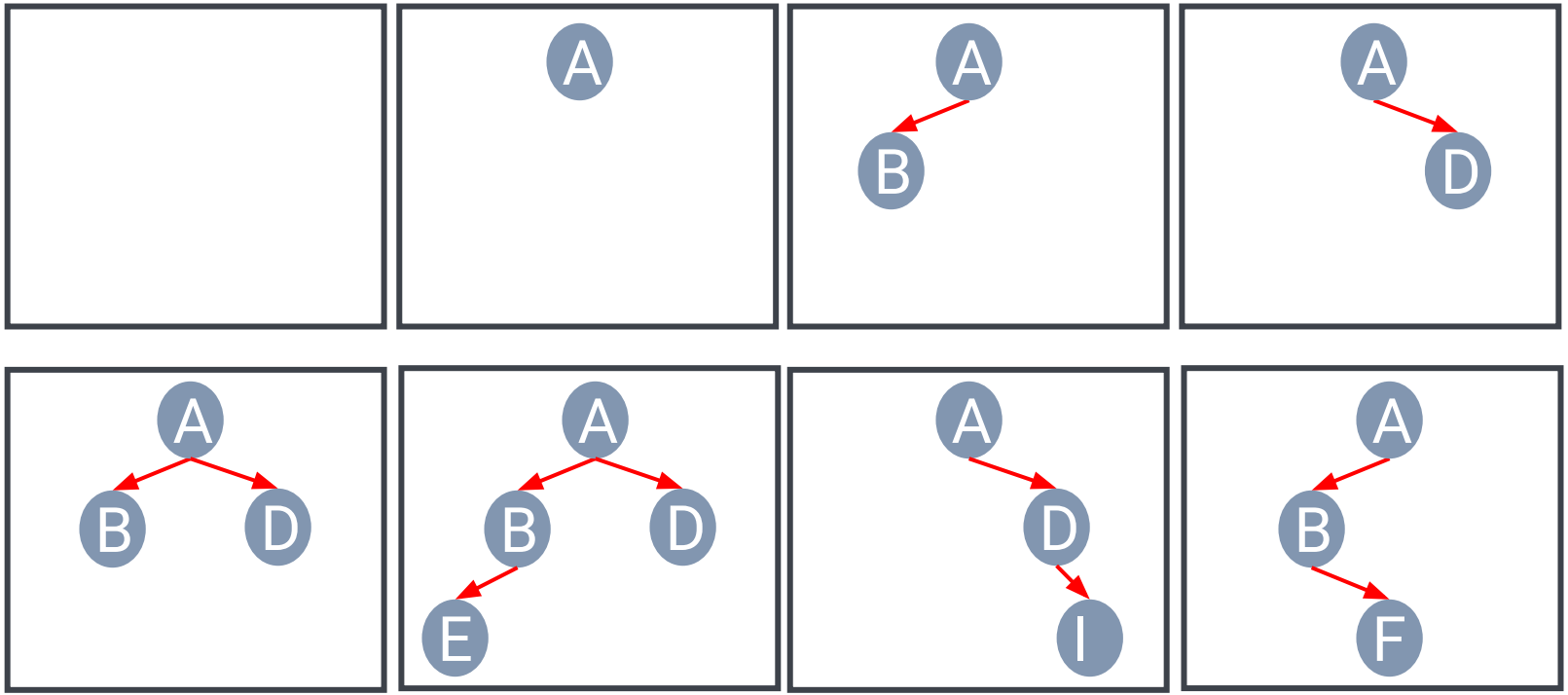
Binary Trees



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

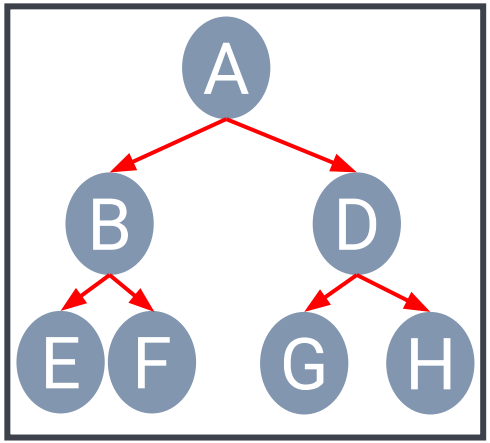
[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

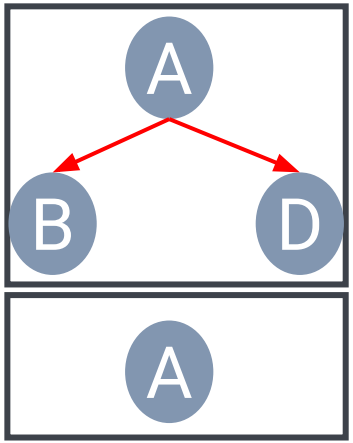
Binary Trees

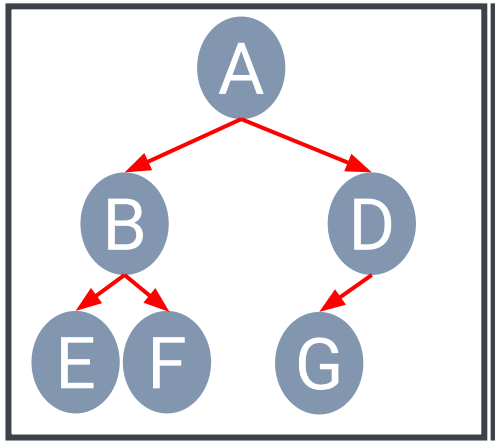


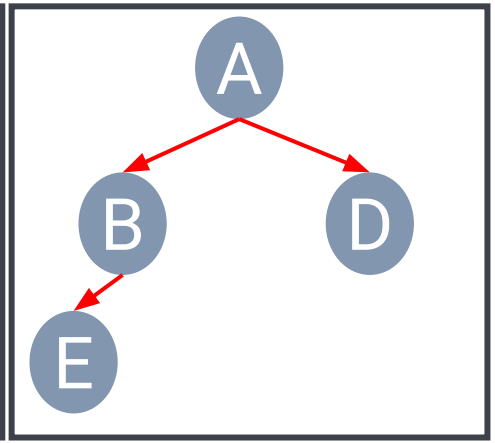
Types of Binary Trees

- 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.







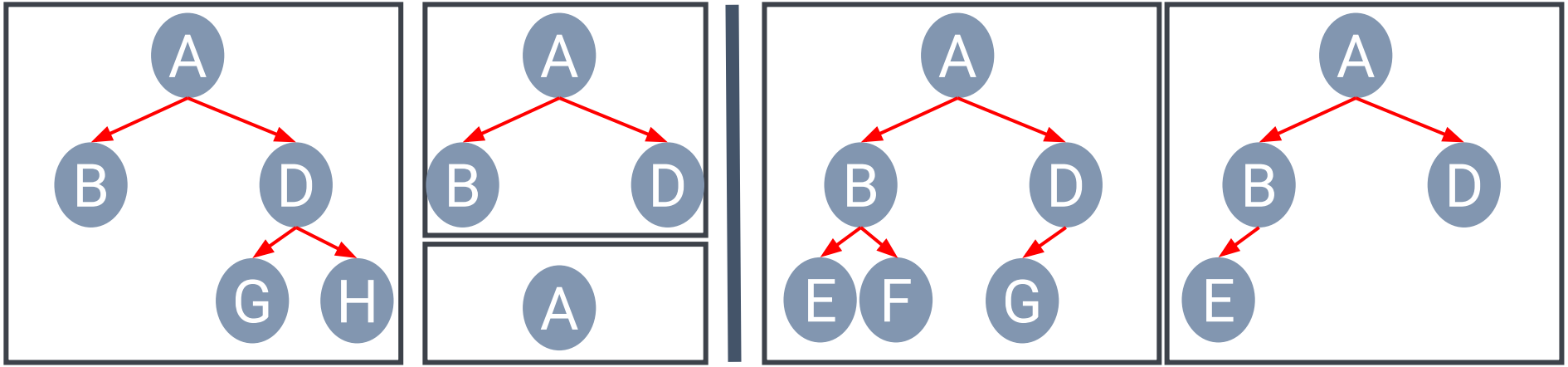


(a) *Complete/Perfect* Binary Tree

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

Types of Binary Trees

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

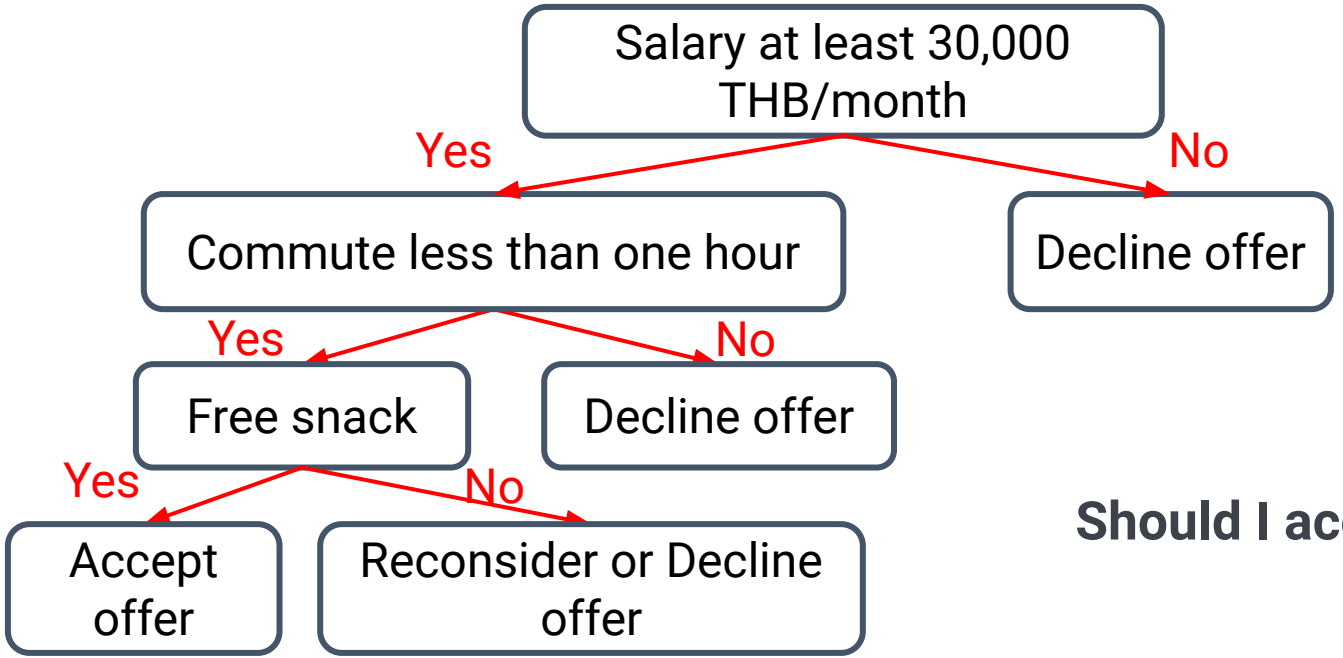


(a) *Proper/Full* Binary Tree

Each node has either 0 or 2 children

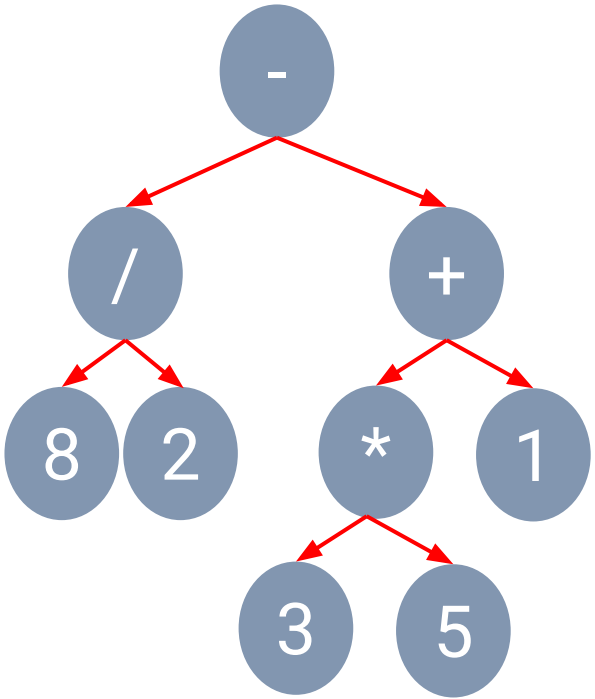
(b) *Improper* Binary Tree

Examples of Binary Trees



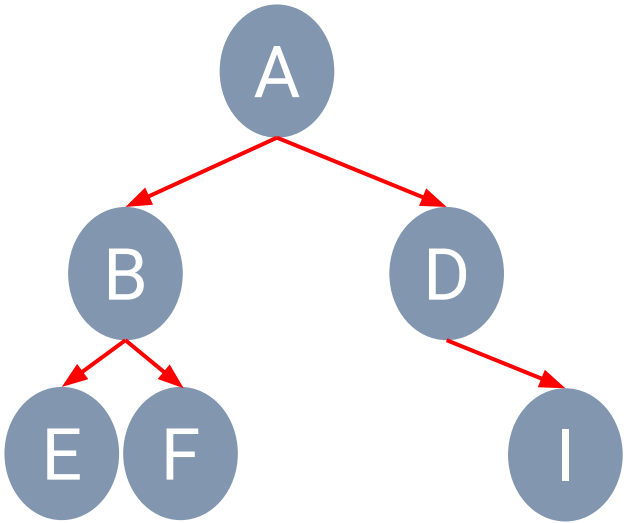
Should I accept a job offer?

Examples of Binary Trees



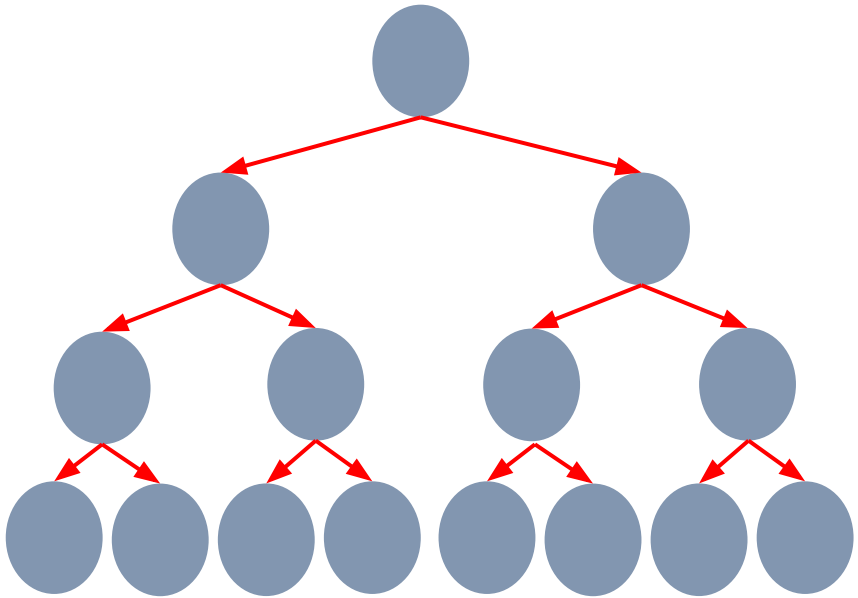
- Binary trees can be used to represent an **arithmetic expression**.
- **Leaves** are associated with **variables** or **constants**.
- **Root** and **Internal** nodes are associated with **operators**.
- **Subtree** is a sub-expression.

Recursive Binary Trees



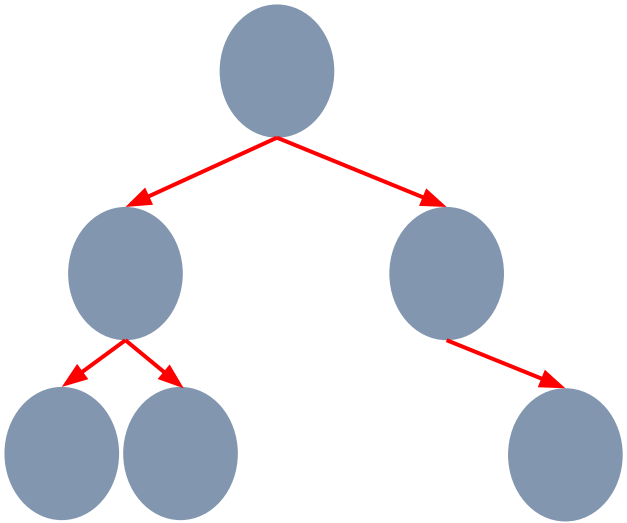
- A **binary tree** is either empty or consists of:
  - A root node that stores an element
  - A binary left subtree (possibly empty)
  - A binary right subtree (possibly empty)

Binary Trees' Properties



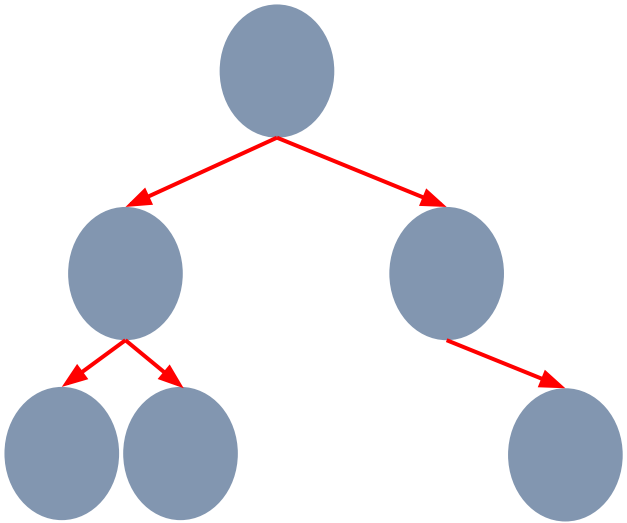
	Nodes
Level 0	1
Level 1	2
Level 2	4
Level 3	8
Level d	2^d

Binary Trees' Properties



- *n* denotes the number of nodes
- *h* denotes the height of tree

Binary Trees Implementation



Arrays (List of Lists)

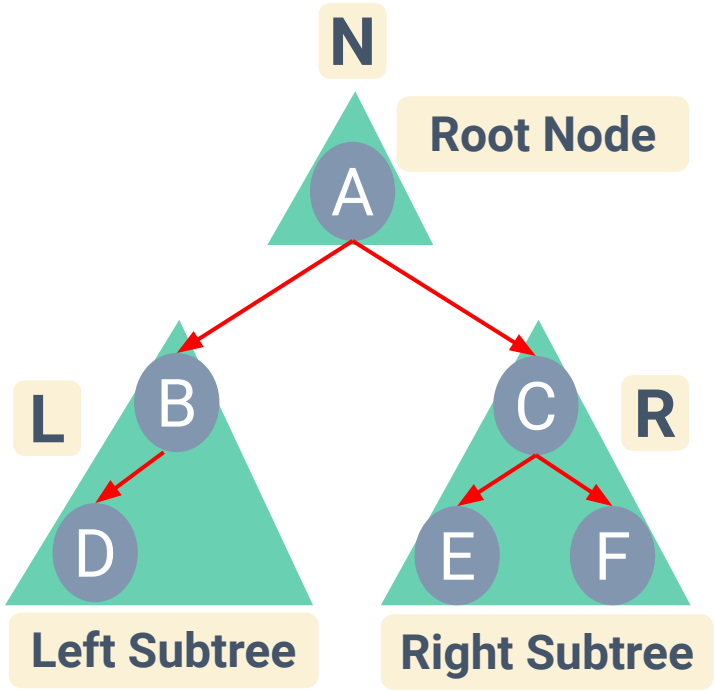
or

Doubly Linked Lists

Binary 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
    - Postorder traversal
    - Inorder traversal
  - Breadth-first - visit all the nodes at depth *d* before moving to depth *d*+1.

Depth-first Traversal



- **Preorder Traversal**

N L R

- **Inorder Traversal**

L N R

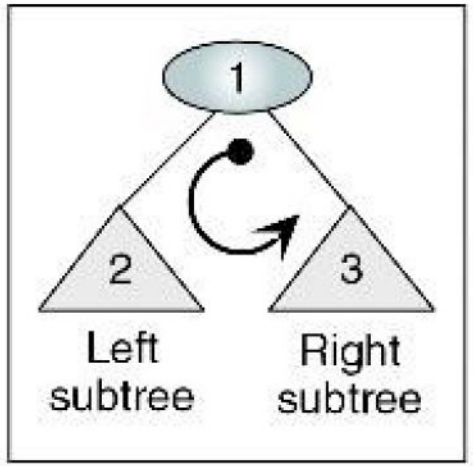
- **Postorder Traversal**

L R N

Preorder Traversal

- Root -> Left subtree -> Right subtree

```
Algorithm preOrder(root):
    if (root is not null):
        process(root)
        preOrder(leftSubtree)
        preOrder(rightSubtree)
    end if
end preOrder
```

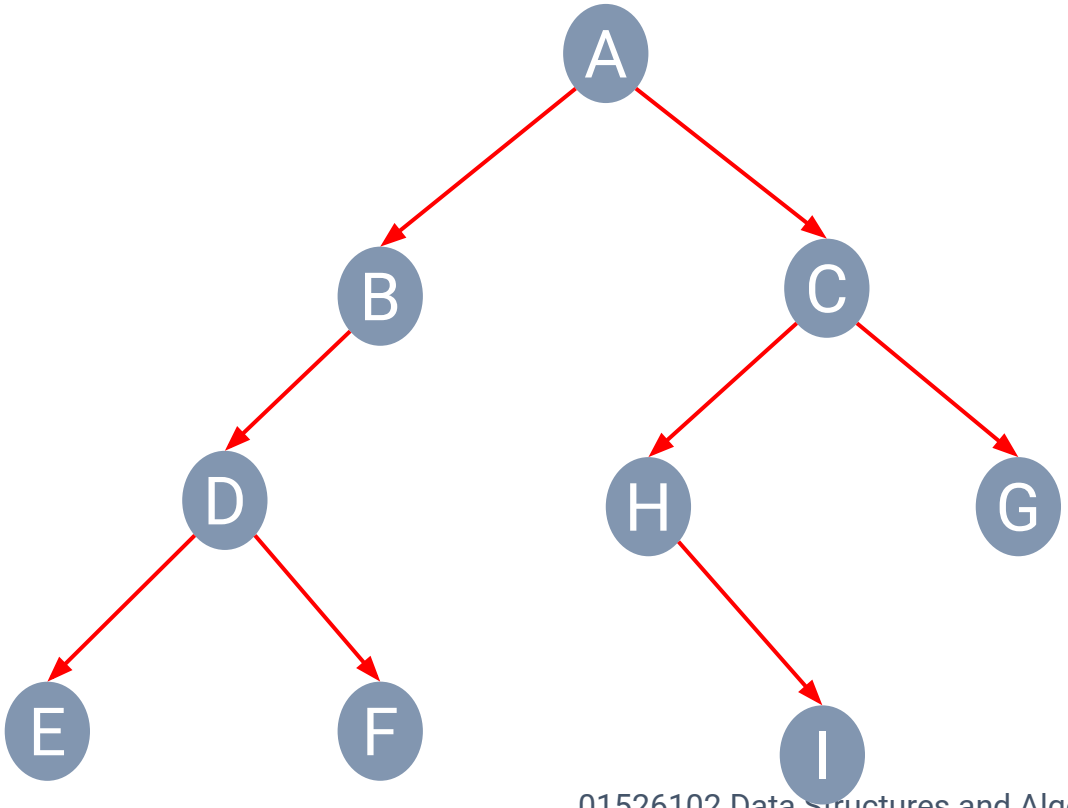


Preorder Traversal

- **Preorder Traversal**

  N L R

- **Visited nodes**



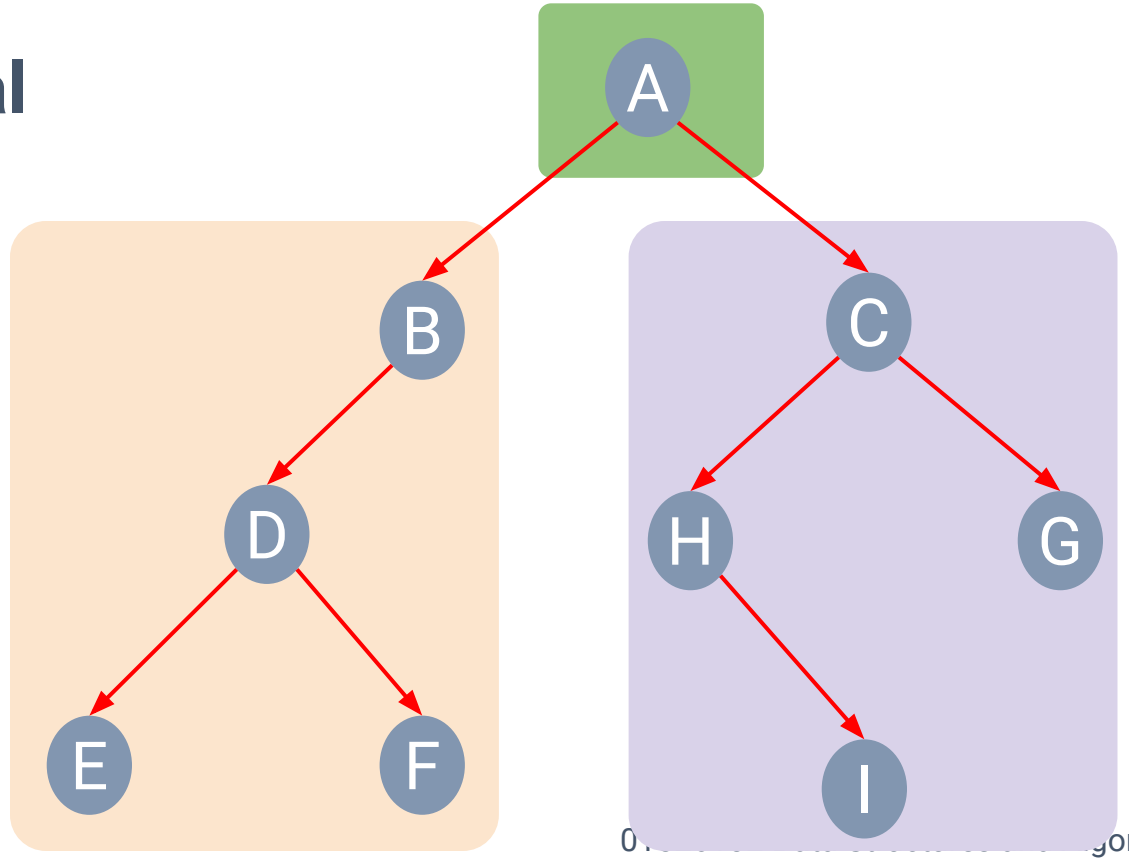
Preorder Traversal

- Preorder Traversal

N L R

- Visited nodes

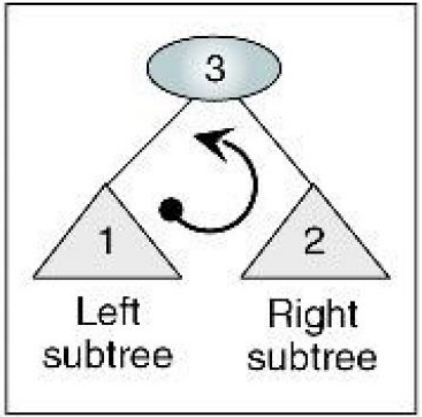
A



Postorder Traversal

- Left subtree -> Right subtree -> root

```
Algorithm postOrder(root):
    if (root is not null):
        postOrder(leftSubTree)
        postOrder(rightSubtree)
        process(root)
    end if
end postOrder
```

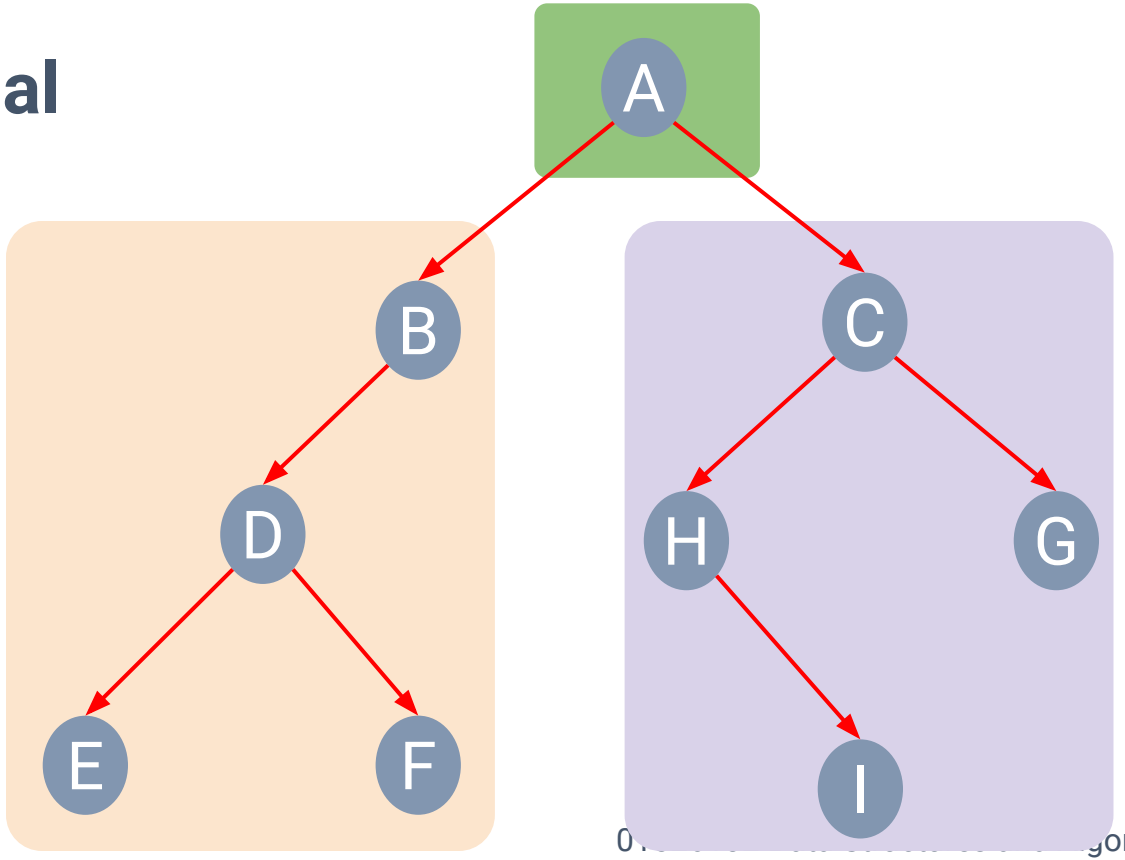


Postorder Traversal

- **Postorder Traversal**

L **R** **N**

- **Visited nodes**

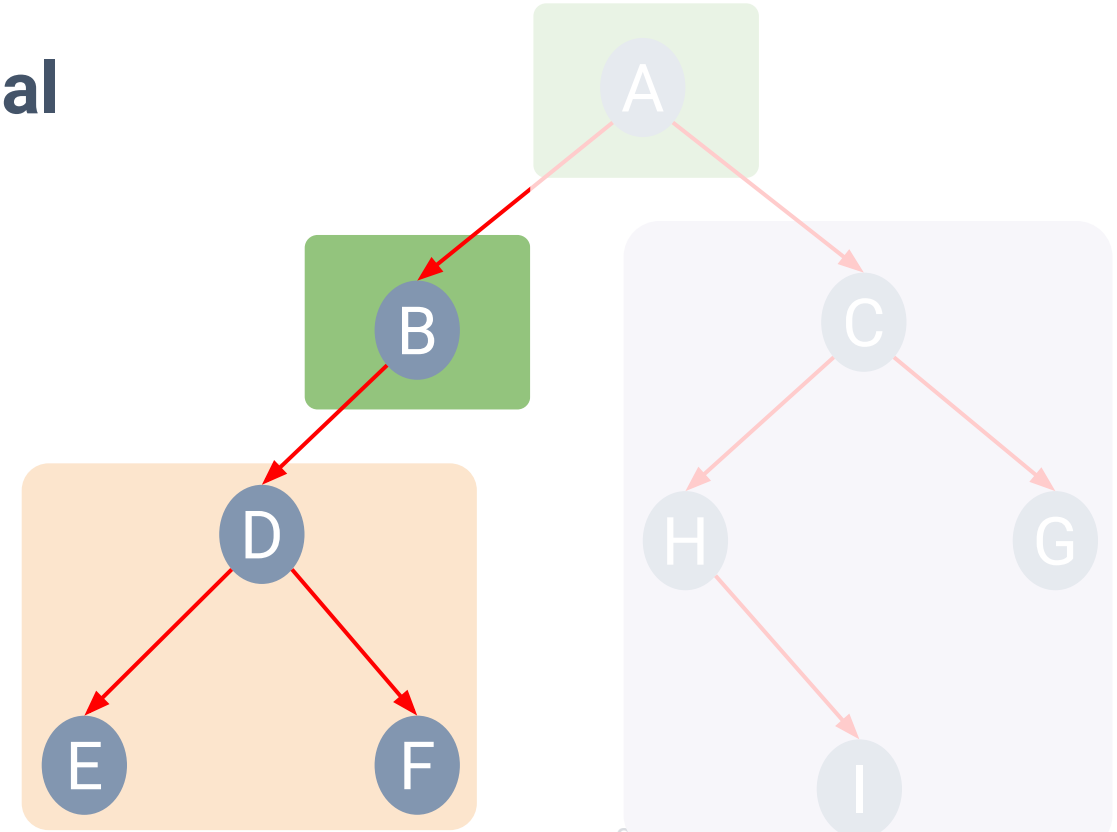


Postorder Traversal

- **Postorder Traversal**

L R N

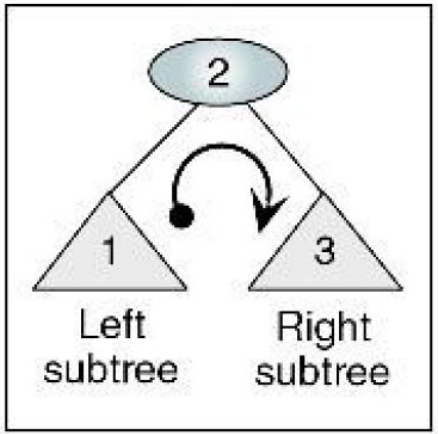
- **Visited nodes**



Inorder Traversal

- Left subtree -> root -> Right subtree

```
Algorithm inOrder(root):
    if (root is not null):
        inOrder(leftSubTree)
        process(root)
        inOrder(rightSubtree)
    end if
end inOrder
```

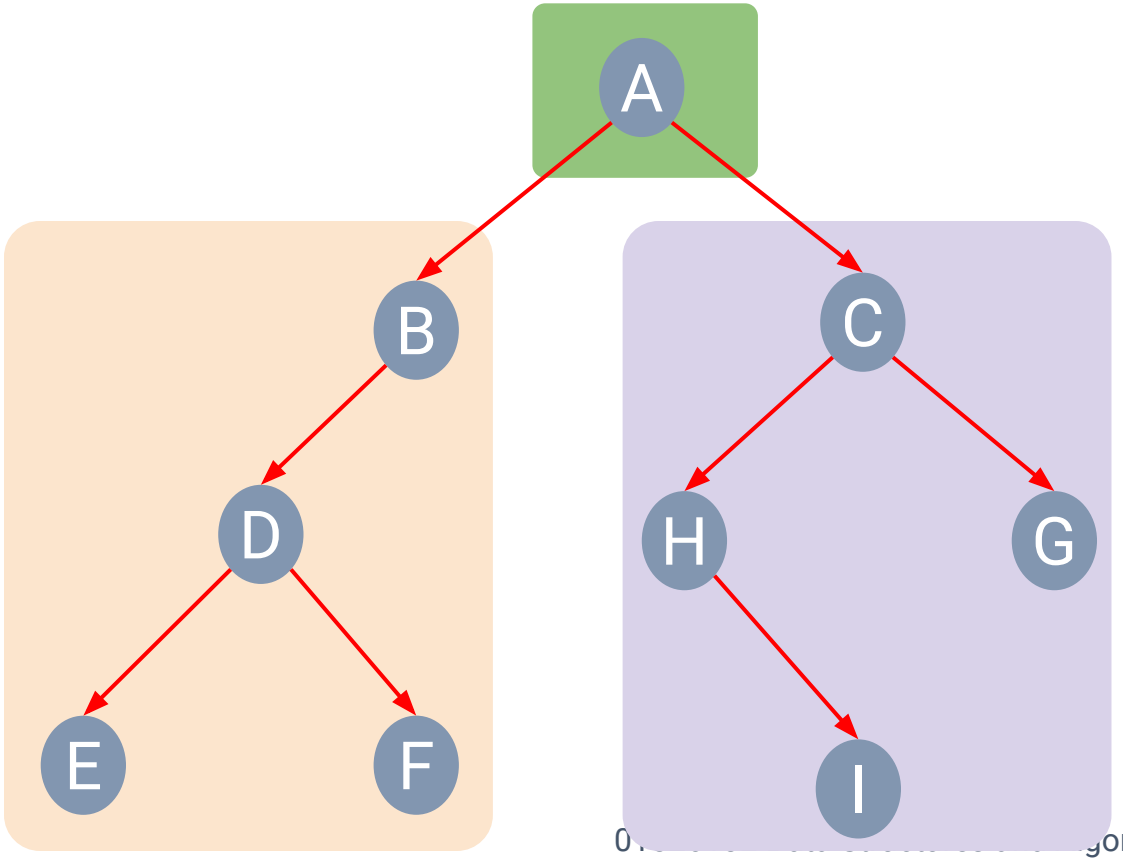


Inorder Traversal

- Inorder Traversal

L N R

- Visited nodes



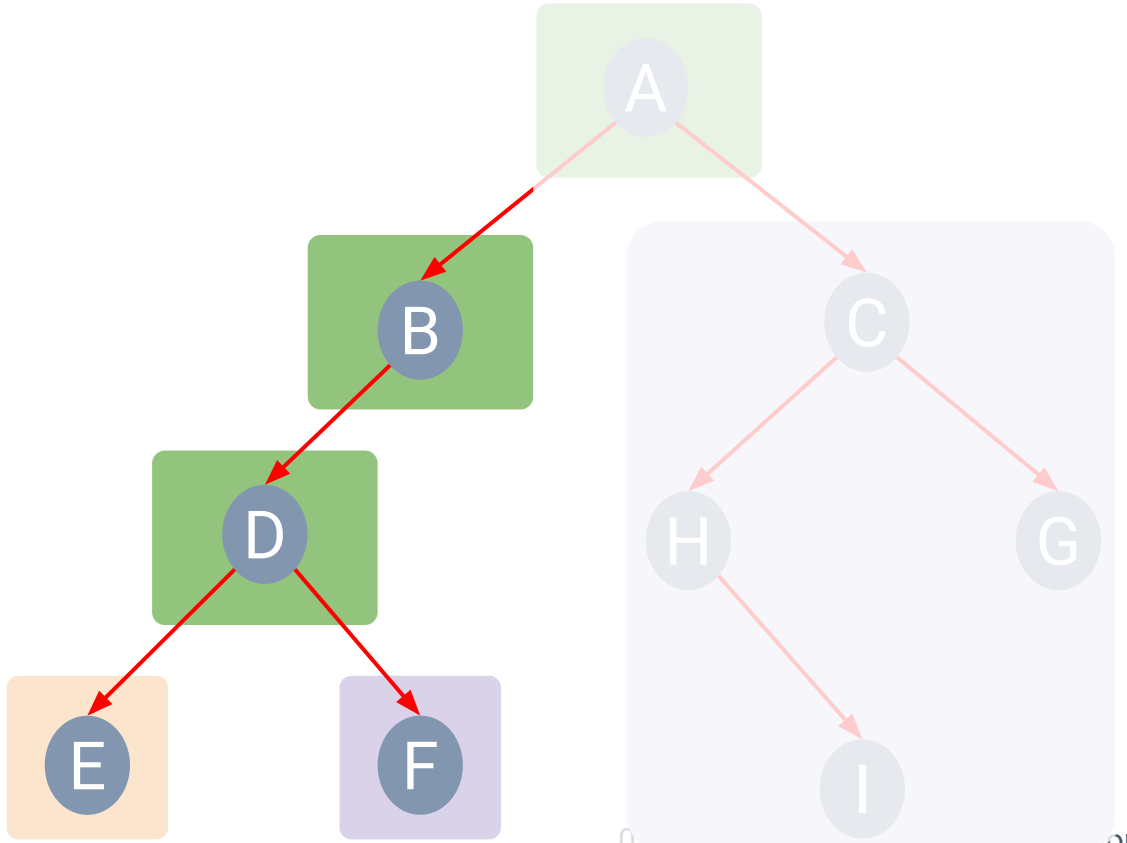
Inorder Traversal

- Inorder Traversal

L N R

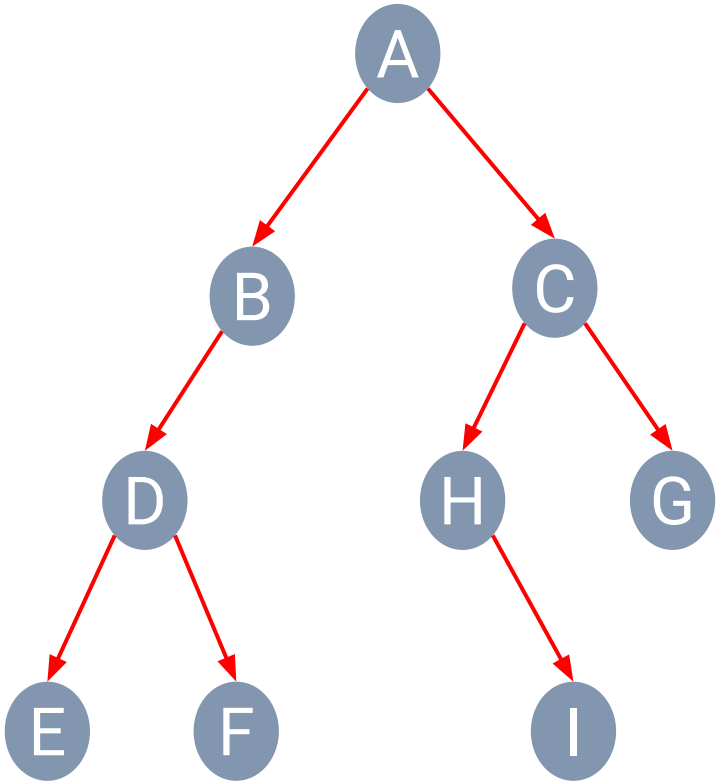
- Visited nodes

E D F



Depth-first Traversal

Traversal of tree: $O(?)$



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

A B D E F C H I G

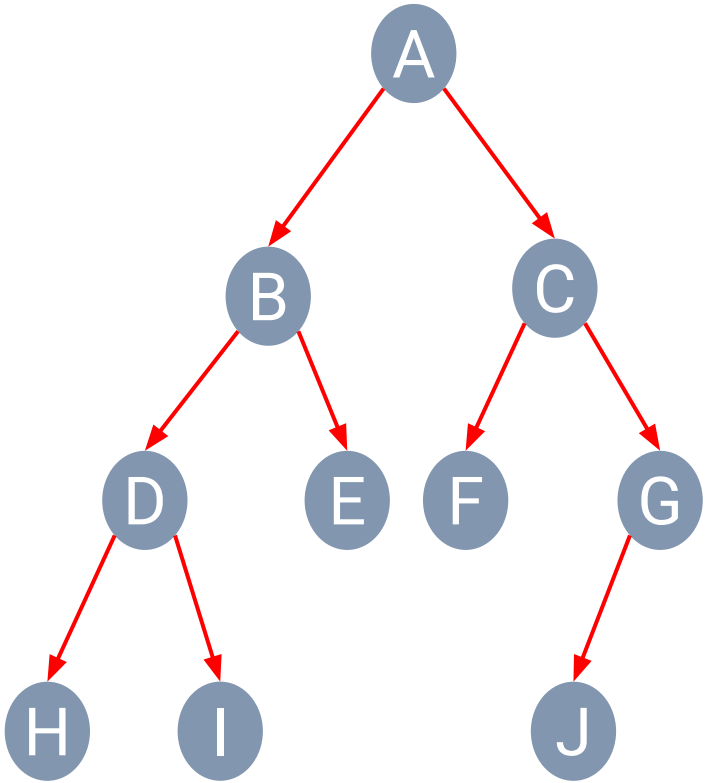
- **In**order Traversal **L N R**

E D F B A H I C G

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

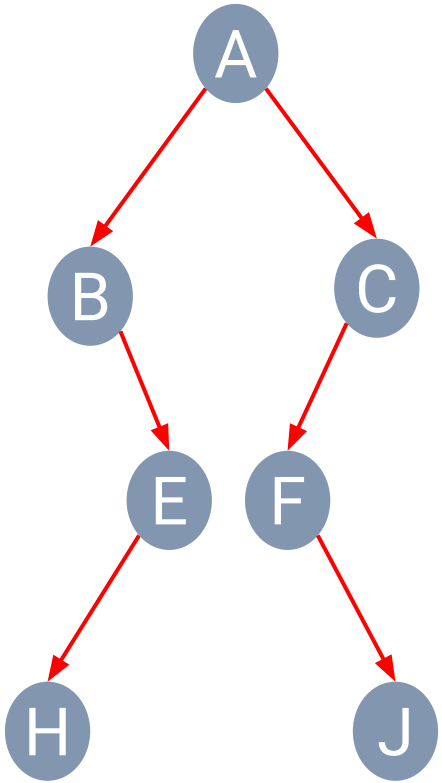
E F D B I H G C A

Depth-first Traversal Exercise 1



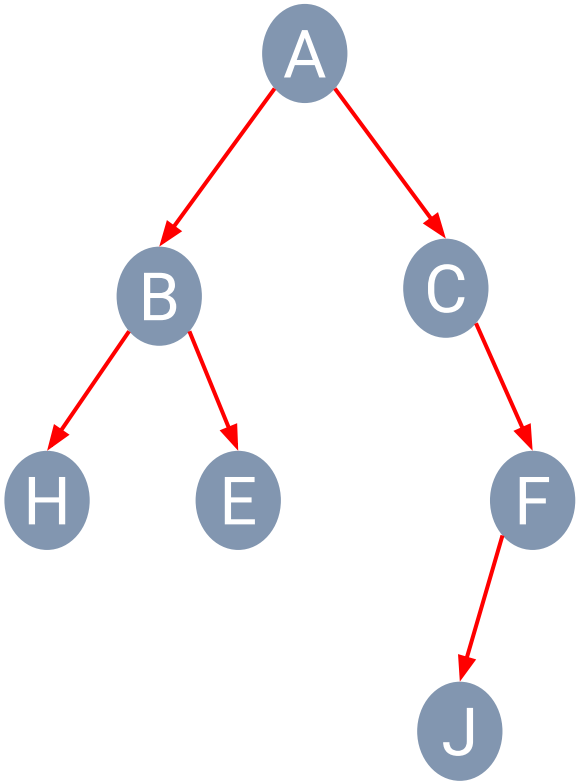
- Preorder Traversal **N L R**
- Inorder Traversal **L N R**
- Postorder Traversal **L R N**

Depth-first Traversal Exercise 2



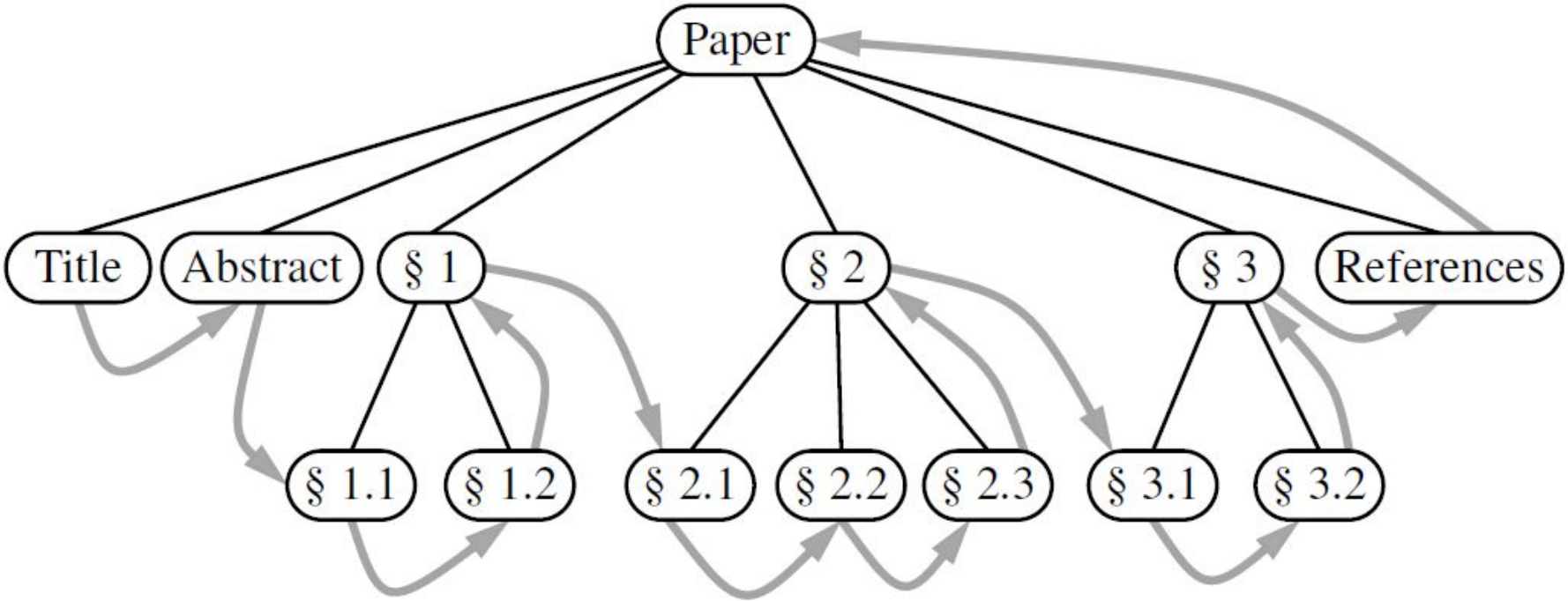
- Preorder Traversal **N** **L** **R**
- Inorder Traversal **L** **N** **R**
- Postorder Traversal **L** **R** **N**

Depth-first Traversal Exercise 3



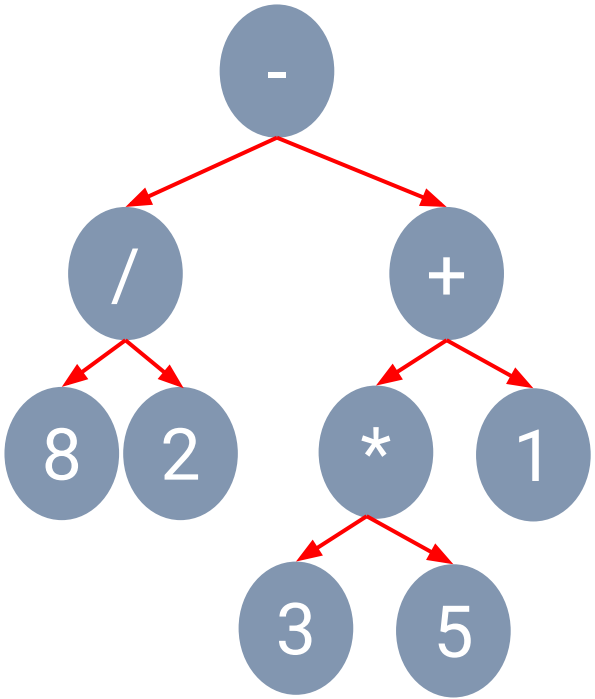
- Preorder Traversal **N** **L** **R**
- Inorder Traversal **L** **N** **R**
- Postorder Traversal **L** **R** **N**

Depth-first Traversal Application



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

Depth-first Traversal Application

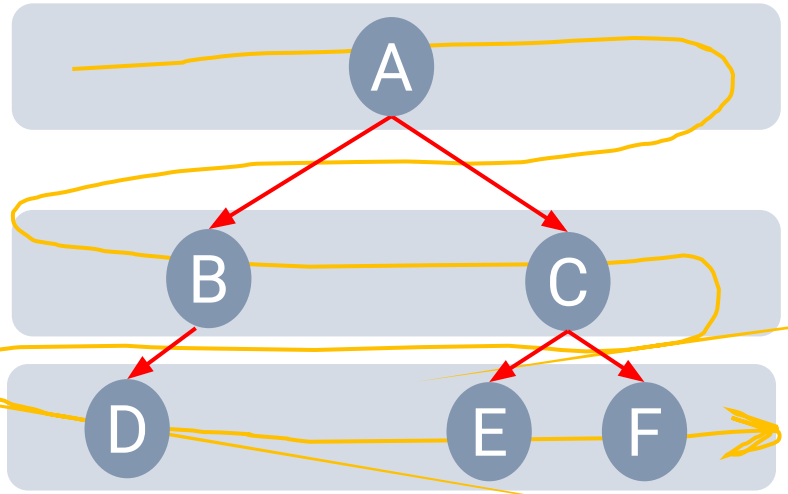


- Binary trees can be used to represent an **arithmetic expression**.
- The **inorder** traversal visits node in a consistent order with the expression.

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

L N R

Breadth-first Traversal



- Visited nodes

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