Introduction to Binary Trees

Lesson 6.1

Learning Objectives

LO 6.1.1 **Convert** conceptual binary tree representations into parenthetical notations and v.v.

LO 6.1.2 **Identify** the valid properties of a binary tree

LO 6.1.3 **Perform** preorder, inorder, and postorder traversals given a binary tree definition

LO 6.1.4 **Create** a binary tree using predefined binary tree traversals

Learning Objectives

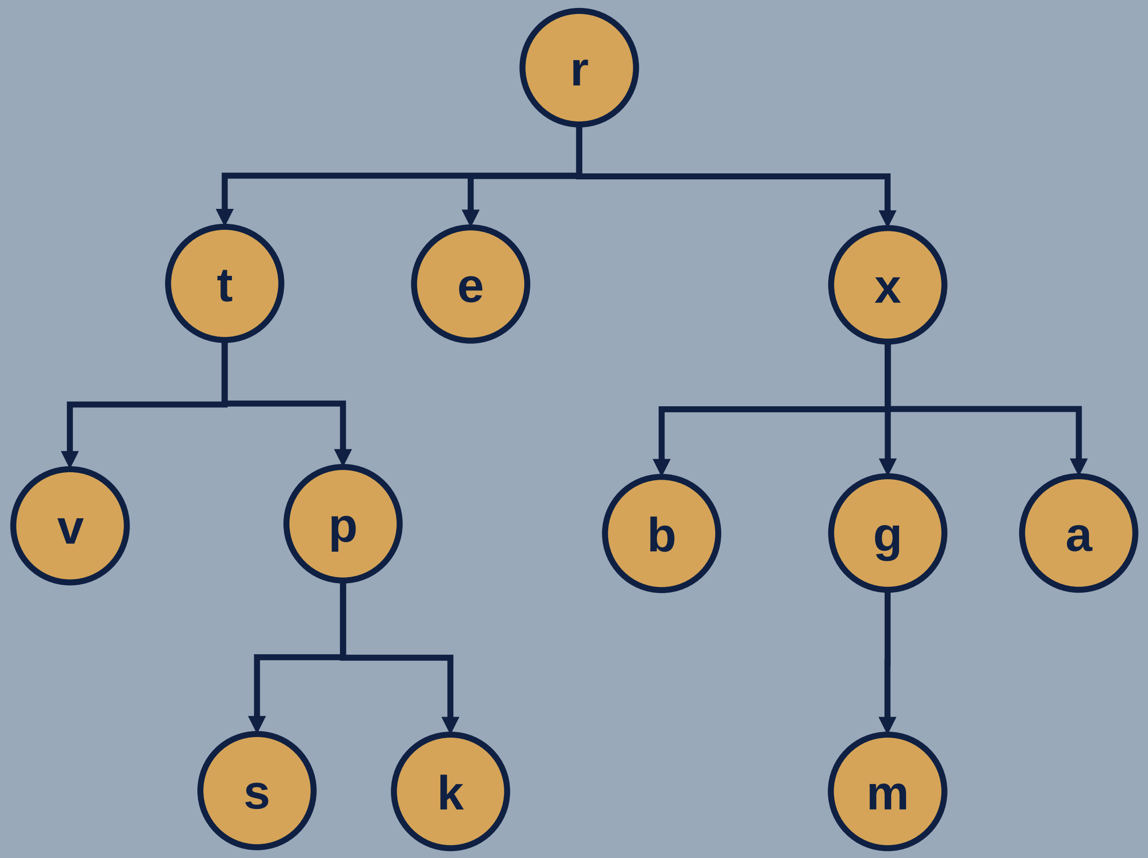
LO 6.1.5 **Compute** the balance factor of a binary tree

LO 6.1.6 **Generate** a binary expression tree given an infix expression

LO 6.1.7 **Synthesize** a valid postfix expression from a binary expression tree

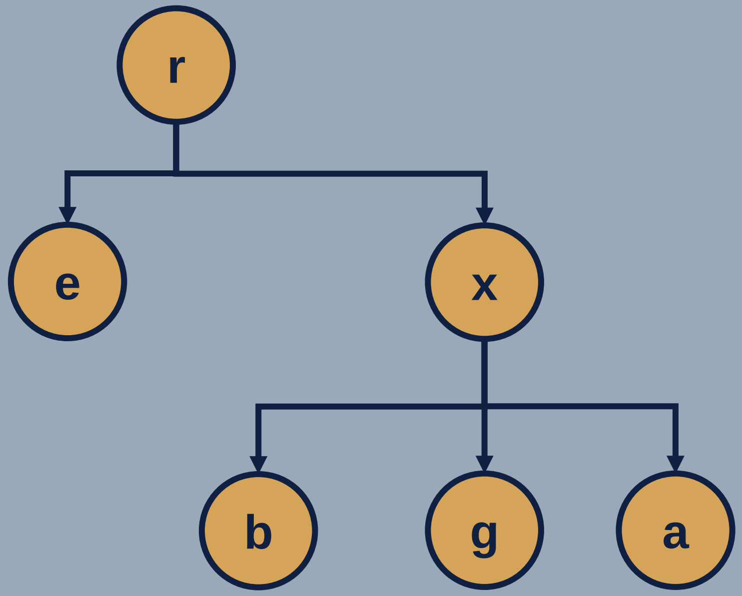
Tree

- Tree is an ADT that consists of a finite set of elements, called tree **nodes**, and a finite set of directed lines, called **branches**, that connect to the nodes.
- Tree, similar to linked lists, is also used as a data structure for storage but is *non-linear* in form.



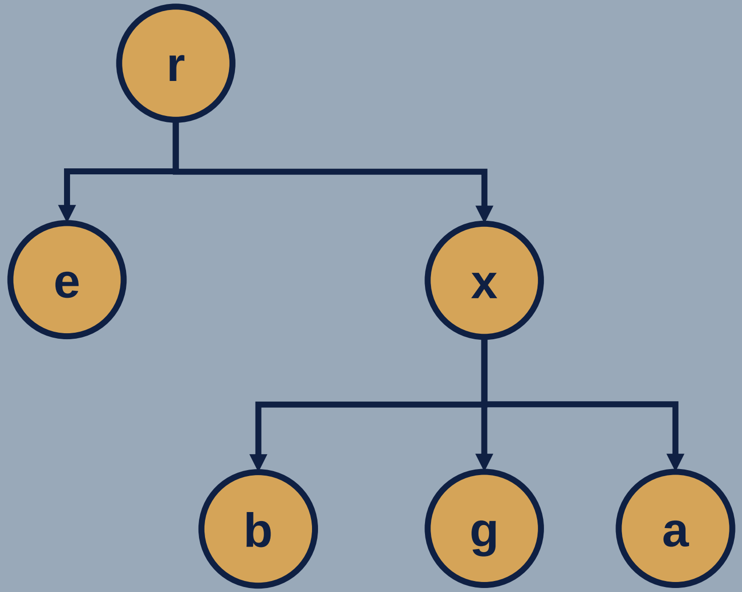
Tree Terminologies

- The number of branches associated with a tree node is the **degree** of the tree node.
- When the branch is directed toward the tree node, it is an **indegree** branch.
- When the branch is directed away from the tree node, it is an **outdegree** branch.
- The sum of the indegree and outdegree branches is the degree of the node.



*The tree node 'x' has a **degree** of 4. It has an **indegree** of 1 and an **outdegree** of 3.*

Tree Terminologies

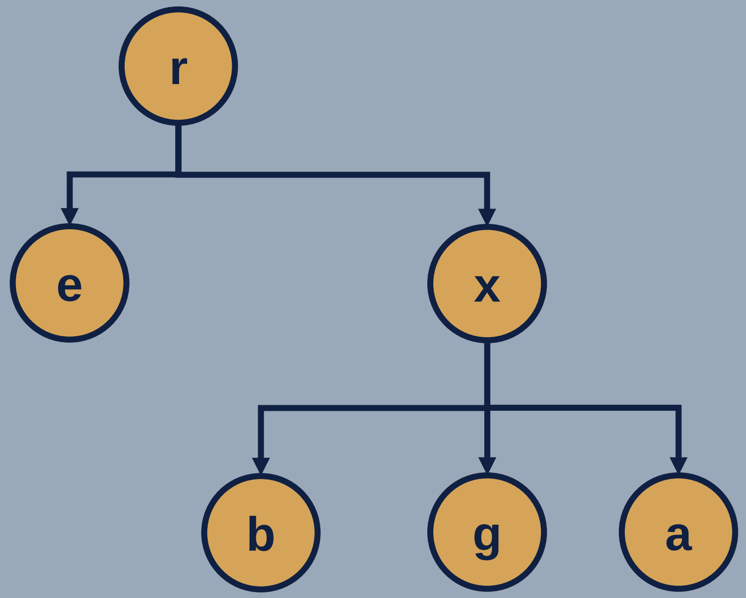


*The tree node 'r' is the **root** of the tree above.*

- If the tree is not empty, the first tree node is called the **root**. The indegree of the root is, by definition, zero.
- With the exception of the root, all of the nodes in a tree must have an indegree of *exactly one*.

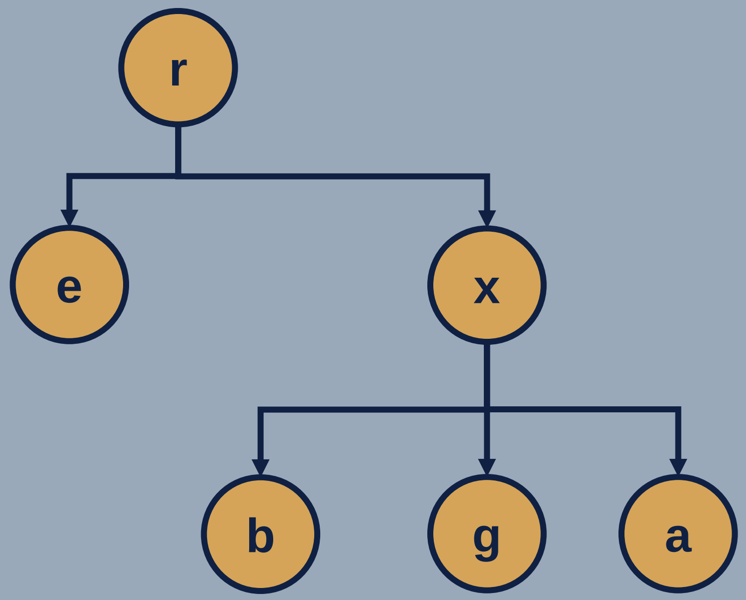
Tree Terminologies

- A **leaf** is any tree node with an outdegree of zero, a node with no successors.
- A tree node that is *not* a *root nor* a *leaf* is known as an **internal node**.
- Tree node A is a **parent** node of tree node B if the outgoing branch of A directs as an ingoing branch of B.
- With the similar definition, B is a **child** node of A.



*Tree nodes 'e', 'b', 'g', and 'a' are **leaf nodes** while 'x' is an **internal node**. Tree node 'x' is a **parent** node of 'a' and at the same time a **child** node of 'r'.*

Tree Terminologies

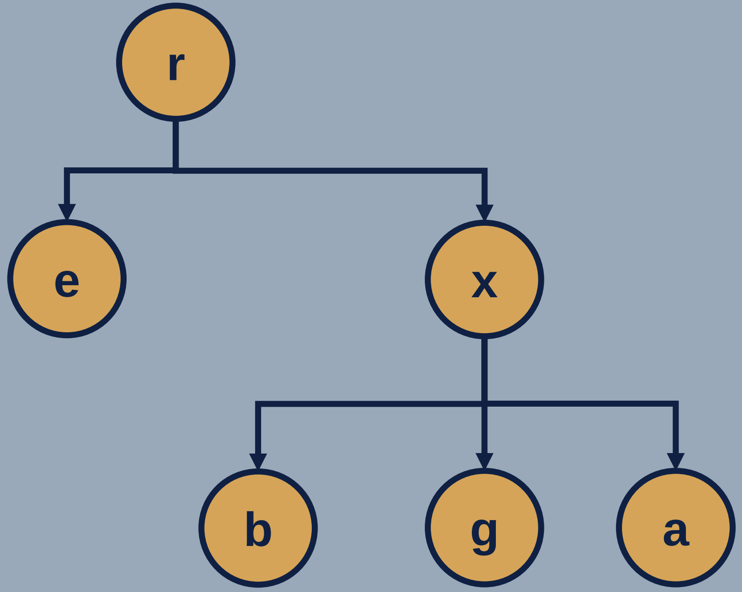


*The tree nodes 'e' and 'x' are **siblings**. Based on the **path** from tree node 'r' to 'a': <u>r – x – a</u>, 'r' is an **ancestor** of 'a', and 'x' is a **descendent** of 'r'.*

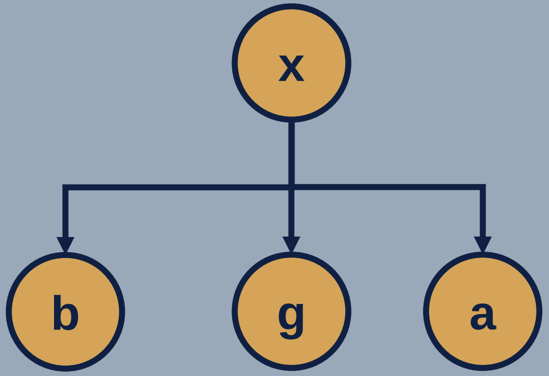
- Two or more tree nodes with the same parent node are **siblings**.
- A **path** is a sequence of nodes along the branches of a tree.
- An **ancestor** of a tree node is any tree node in the path from the root to that tree node.
- A **descendent** of a tree node is any tree node in the path from that tree node to any leaf node below (*outgoing*) that tree node.

Tree Terminologies

- A **level** of a tree node is the number of branches it takes to traverse from the root to that tree node. The *root*, by definition, has a *level* of *0*.
- A **subtree** is any connected structure below the root. A subtree based from any tree node makes that tree node the root of that subtree.
- The **height** of a tree/subtree is the highest leaf node level plus 1.



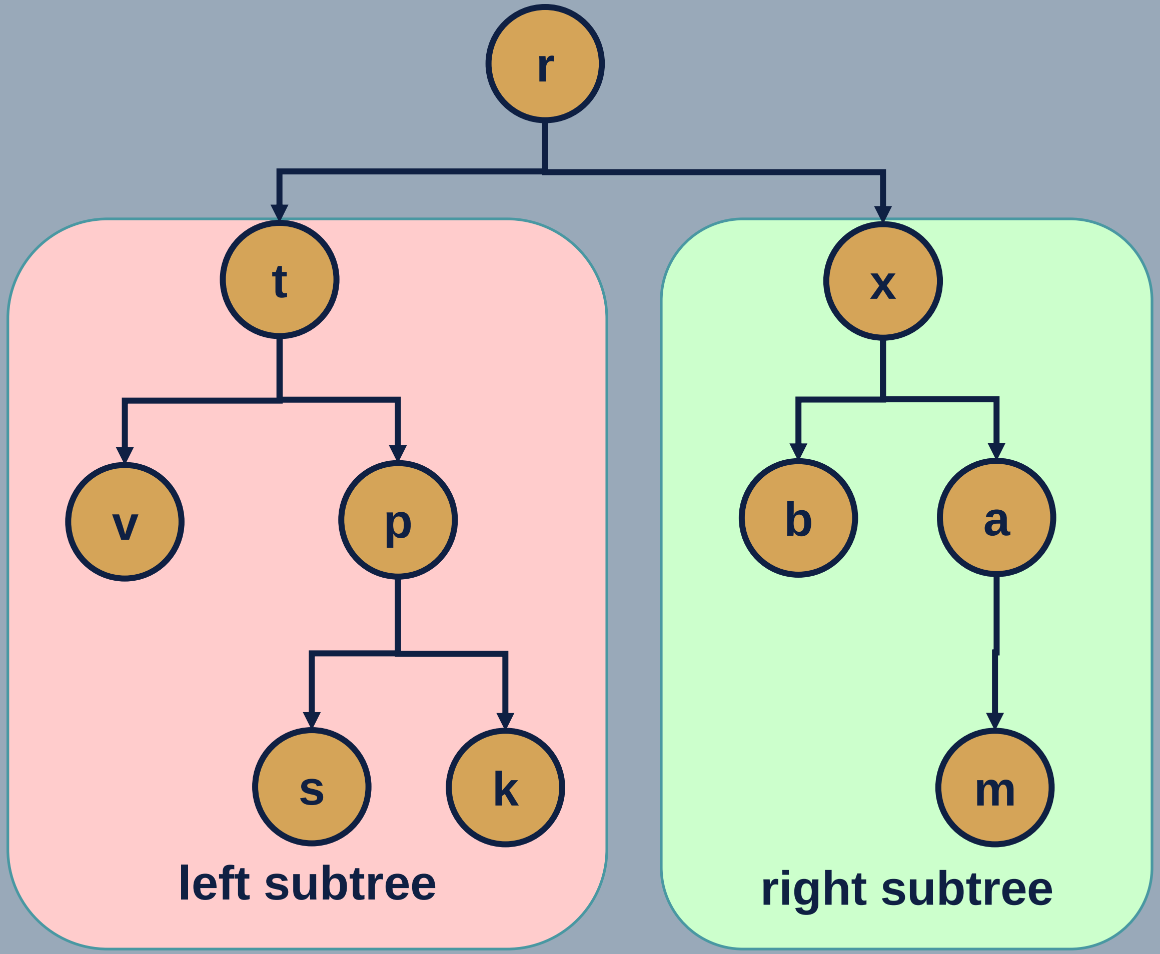
*Tree node 'g' has a **level** of 2.*



*The tree above is a **subtree** from the tree above it. The **height** of this subtree is 2, while the **height** of the uppermost tree is 3.*

Binary Tree

- A **binary tree** is a tree in which no tree node can have more than two *subtrees*; the maximum outdegree for any tree node is **2**.
- These subtrees are designated as the **left subtree** and **right subtree**. Note that each subtree itself is a binary tree.



*The 't' subtree is the **left subtree** and 'x' subtree is the **right subtree** of the tree node 'r'.*

Binary Tree Parenthetical Notations

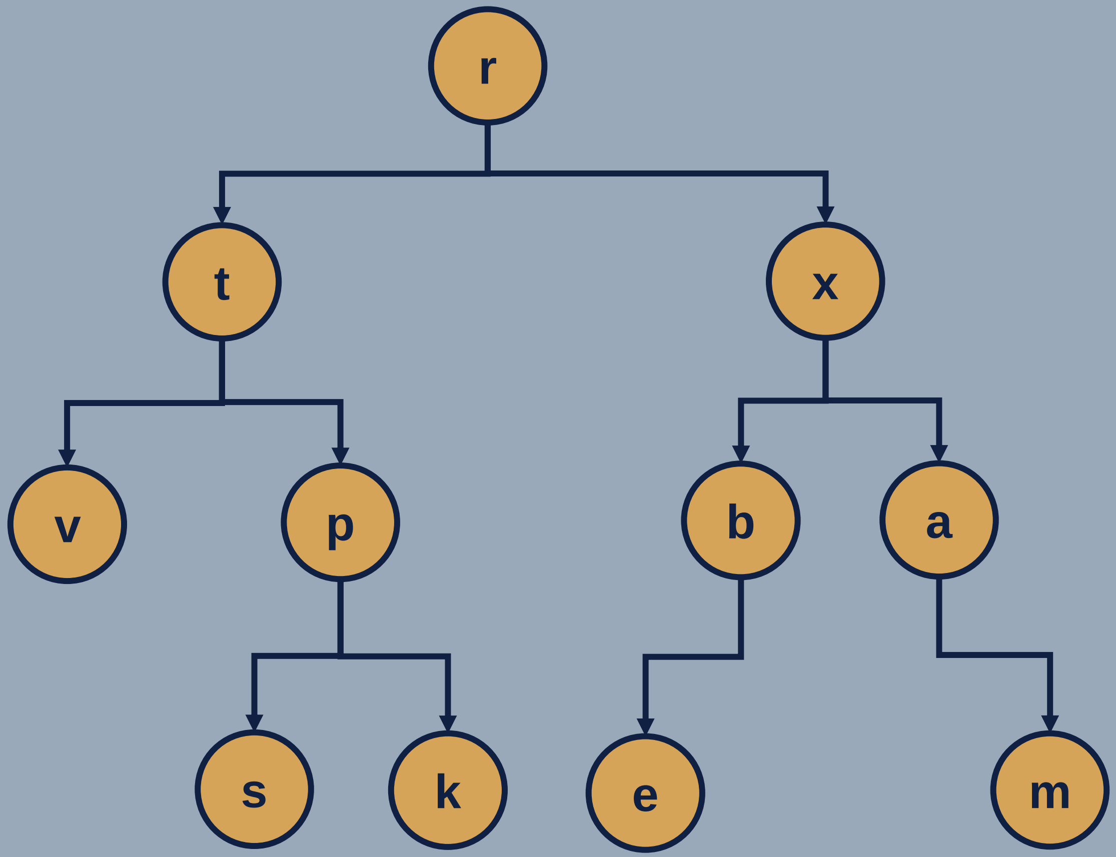
- Most binary trees are written in the form a diagram representation but it can also be represented as a text through **parenthetical notations**.

- The parenthetical notation follows a recursive grammar:

<parent node data> [(**<left child node data> <right child node data>**)]

- The brackets (‘[‘ and ‘]’) means it is optional, i.e. in the case of leaf nodes where the node has no child nodes. If in any case there is no left child of a parent node, a symbol ‘-’ is written while if there is a left child but no right child, only the left child node data is written. The delimiter used will be space symbol.

Binary Tree Parenthetical Notations



- The parenthetical notation of the tree on the left is:

r (t (v p (s k)) x (b (e) a (- m)))

Binary Tree Properties

- The maximum height of a binary tree with N tree nodes, H_{max} is

$$H_{max} = N$$

- The minimum height of a binary tree with N tree nodes, H_{min} is

$$H_{min} = \lfloor \log_2 N \rfloor + 1$$

- The minimum number of nodes, N_{min}, of a binary tree with height H is

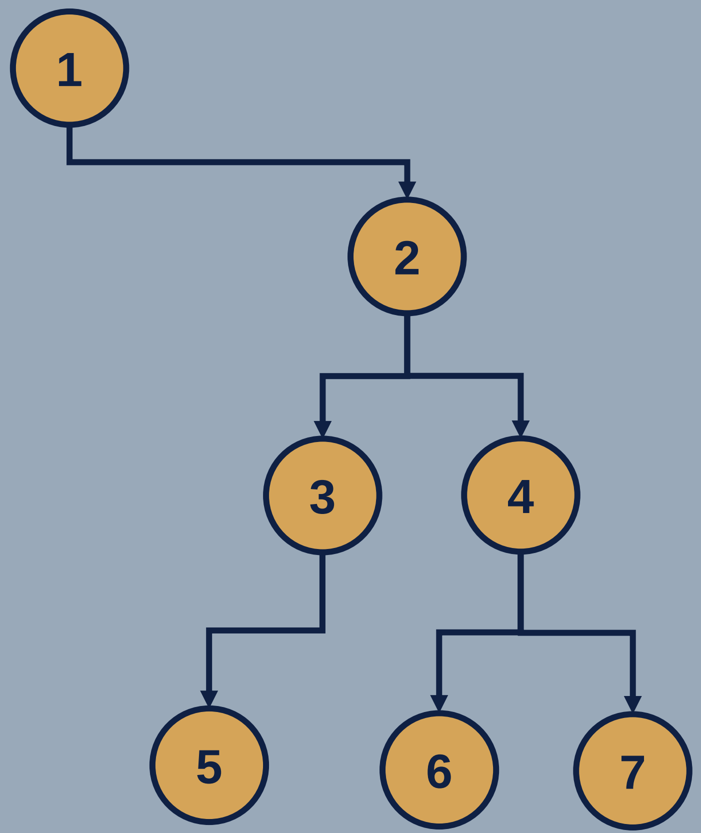
$$N_{min} = H$$

- The maximum number of nodes, N_{max}, of a binary tree with height H is

$$N_{max} = 2^H - 1$$

LO 6.1.1 Convert conceptual binary tree representations into parenthetical notations and v.v.

1. Convert the binary tree representation below into parenthetical notation:



LO 6.1.1 Convert conceptual binary tree representations into parenthetical notations and v.v.

2. Convert the parenthetical notation below into binary tree representation:

a (b (- c (d (e f) g (h (i)))) j)

LO 6.1.2 Identify the valid properties of a binary tree

Identify the following of the binary tree represented by the parenthetical notation below:

a (b (- c (d (e f) g (h (i)))) j)

1. degree, indegree, and outdegree of binary tree nodes
2. root
3. leaf nodes
4. internal nodes

LO 6.1.2 Identify the valid properties of a binary tree

Identify the following of the binary tree represented by the parenthetical notation below:

a (b (- c (d (e f) g (h (i)))) j)

5. parent of ‘f’
6. children of ‘b’
7. sibling of ‘g’
8. all ancestors of ‘c’
9. all descendents of ‘c’

LO 6.1.2 Identify the valid properties of a binary tree

Identify the following of the binary tree represented by the parenthetical notation below:

a (b (- c (d (e f) g (h (i)))) j)

10. Level of ‘h’
11. Level of ‘i’ in subtree ‘c’
12. Height of the binary tree
13. Height of subtree ‘b’

LO 6.1.2 Identify the valid properties of a binary tree

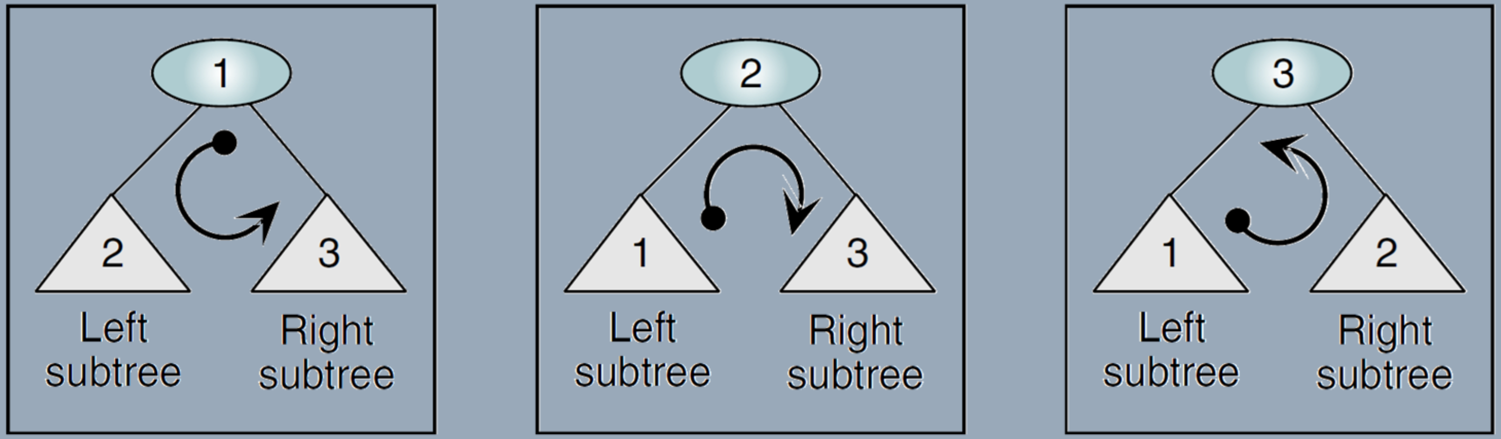
14. Maximum and minimum height of a binary tree with 10 nodes
15. Maximum and minimum number of nodes of a binary tree with a height of 4

Binary Tree Traversals

- A binary tree traversal requires that each node of the tree be processed once and only once in a predetermined sequence.
- The two general approaches to the traversal sequence are **depth first** and **breadth first**.

Binary Tree Traversals

- In *depth-first traversal*, processing proceeds along a path from the root through one child to the most distant descendent of that first child before processing a second child. In this traversal, we process all of the descendents of a child before going on to the next child. **Preorder**, **inorder**, and **postorder** traversals are depth-first traversals.



(a) Preorder traversal (b) Inorder traversal (c) Postorder traversal

Binary Tree Traversals

- *Preorder Traversal of a Binary Tree*

algorithm *preOrder* (*root*)
  Traverse a binary tree in node-left-right sequence.
  Pre: root is the entry node of a tree or subtree.
  Post: each node has been processed in order.
  if (*root* is not null)
    process (*root*)
    preOrder (*root*'s left child)
    preOrder (*root*'s right child)
  end if
end *preOrder*

Binary Tree Traversals

- *Inorder Traversal of a Binary Tree*

algorithm *inOrder* (*root*)
    Traverse a binary tree in left-node-right sequence.
    Pre: root is the entry node of a tree or subtree.
    Post: each node has been processed in order.
    if (*root* is not null)
        inOrder (*root*'s left child)
        process (*root*)
        inOrder (*root*'s right child)
    end if
end *inOrder*

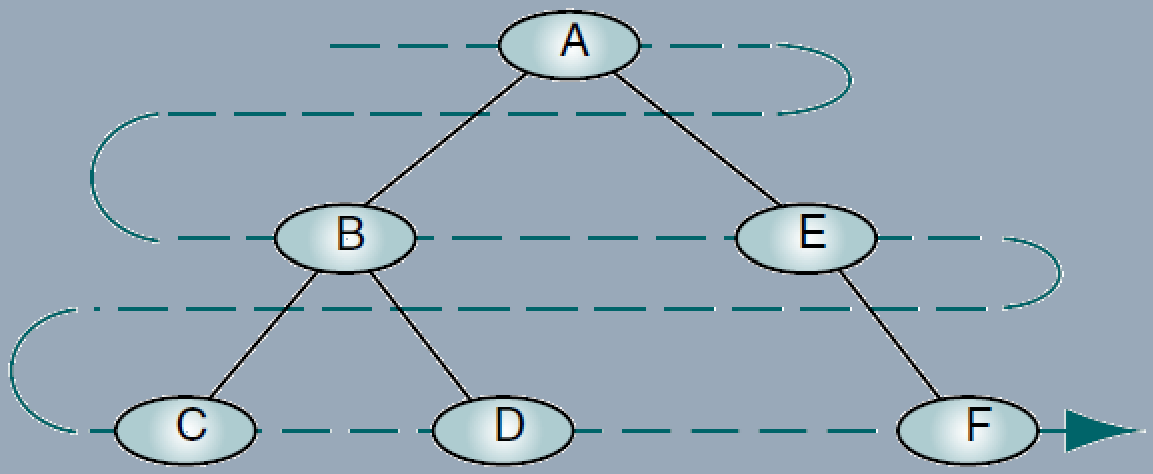
Binary Tree Traversals

- *Postorder Traversal of a Binary Tree*

algorithm *postOrder* (*root*)
    Traverse a binary tree in left-right-node sequence.
    Pre: root is the entry node of a tree or subtree.
    Post: each node has been processed in order.
    if (*root* is not null)
        postOrder (*root*'s left child)
        postOrder (*root*'s right child)
        process (*root*)
    end if
end *postOrder*

Binary Tree Traversals

- In *breadth-first traversal*, the processing proceeds horizontally from the root to all of its children, then to its children's children, and so on until all tree nodes have been processed.



LO 6.1.3 Perform preorder, inorder, and postorder traversals given a binary tree definition

Display the preorder, inorder, and postorder traversals of the binary tree represented in parenthetical notation below:

a (b (- c (d (e f) g (h (i)))) j)

LO 6.1.4 Create a binary tree using predefined binary tree traversals

A binary tree has 10 nodes. The preorder and inorder traversals of the tree are shown below. Draw the tree.

Preorder	JCBADEFIGH
Inorder	ABCEDFJGIH

Balancing Binary Trees

- Tree ADT/data structures are popular for their O(log n) operations.
- This is only achievable if the length of the path to all leaf nodes of the tree is similar; if not equal, therefore the tree nodes must be spread uniformly on the tree.
- It stands to reason that the shorter the tree, the easier it is to locate any desired node in the tree.
- This concept leads us to a very important characteristic of a binary tree — its *balance*.

Balancing Binary Trees

- To determine whether a tree is balanced, we calculate its **balance factor**, B.
- The balance factor of a binary tree is the difference in height between its left and right subtrees.

$$B = H_L - H_R$$

- In a balanced binary tree, the height of its subtrees differs by no more than one (B = −1, 0, or +1), and its subtrees are also balanced.

LO 6.1.5 Compute the balance factor of a binary tree

Compute the balance factor/s of the binary tree represented in parenthetical notation below:

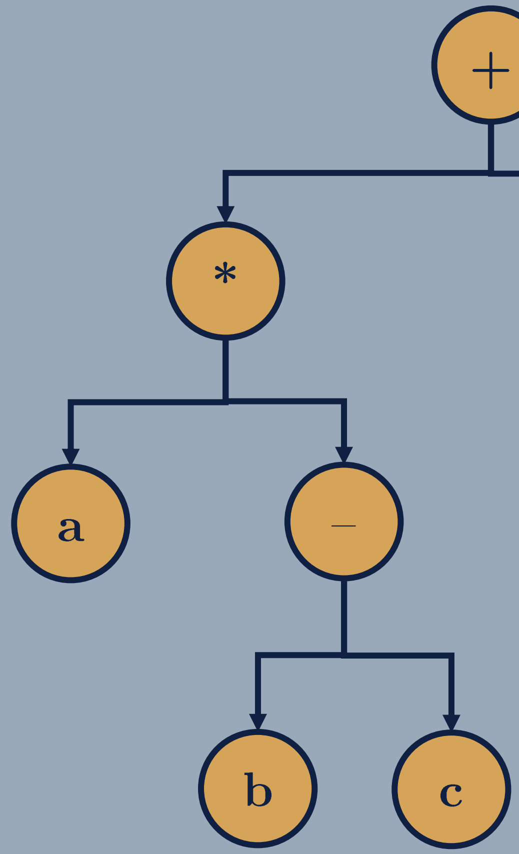
a (b (- c (d (e f) g (h (i)))) j)

Binary Expression Trees

- One interesting application of binary trees is *expression trees*.
- An **expression** is a sequence of tokens that follow prescribed rules.
- A **token** may be either an operand or an operator.
- In this discussion, we consider only binary arithmetic operators in the form *operand–operator–operand*. The standard operators are +, −, *, /, %, and ^.

Binary Expression Trees

- An **expression tree** is a binary tree with the following properties:
    1. Each leaf is an operand.
    2. The root and internal nodes are operators.
    3. Subtrees are subexpressions, with the root being an operator.



$$\boxed{a * (b + c) + d}$$

LO 6.1.6 Generate a binary expression tree given an infix expression
LO 6.1.7 Synthesize a valid postfix expression from a binary expression tree

Generate a binary expression tree given the infix expression below, and synthesize a valid postfix expression based from the binary expression tree generated.

$$3 + 2 \ \% \ 5 \ \hat{} \ 3 \ / \ 46 - 12 * 5 + 4 - 8 \ \hat{} \ 7 \ \% \ 10$$