UNIVERSITY OF TEXAS ARLINGTON

Chapter 6.2
Trees and Their Representations

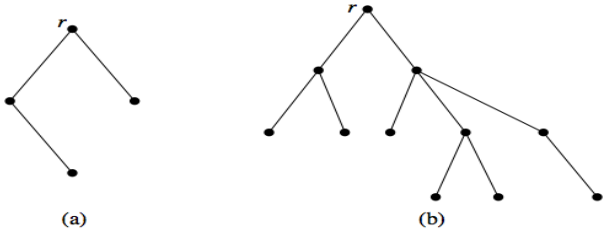
Instructor: Abhishek Santra

Email: abhishek.santra@uta.edu

1

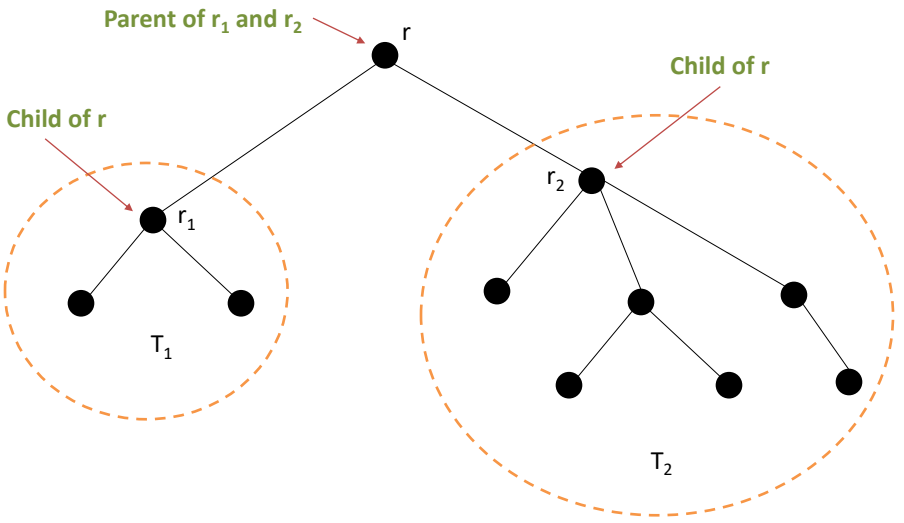
Tree Terminology

- Tree is a *special type of graph*
- A **tree** is an **acyclic, connected** graph with one node designated as the **root** of the tree.
- An acyclic, connected graph with *no designated root node* is called a **non-rooted tree** or a **free tree.**



2

Defining Trees Recursively



3

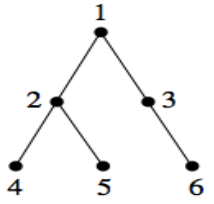
Defining Trees Recursively

- A single node is a tree (with that node as its root).
- If $T_1, T_2, \ldots, T_t$ are **disjoint trees** with roots $r_1, r_2, \ldots, r_t$, the graph formed by attaching a new node r by a single arc to each of $r_1, r_2, \ldots, r_t$ is a tree with root r.
- The nodes $r_1, r_2, \ldots, r_t$ are **children** of r
- The node r is a **parent** of $r_1, r_2, \ldots, r_t$.

4

Tree Terminology

- **Depth of a node** in a tree: Length of the path from the root to *the node*
  - The root itself has depth 0.
- **Height of the Tree:** Maximum depth of any node in the tree
  - *or,* Length of the longest path from the root to any node.
- A node with no children is called a **leaf** of the tree
- All non-leaves are **internal nodes**

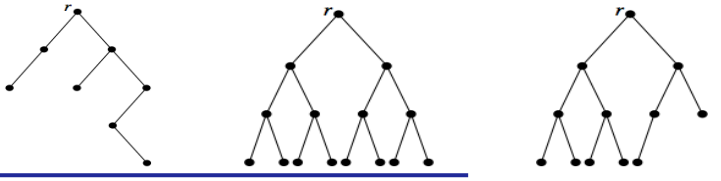


Root: Node 1
Depth of node 4: 2
Height of the tree: 2
Leaves: 4, 5, 6
Internal nodes: 1, 2, 3

5

Tree Terminology

- **Binary Trees:** Each node has *at most* two children
- Each child of a node is designated as either the **left child** or the **right child.**
- **Full Binary Tree** (as seen in the middle figure below): All internal nodes have two children and all leaves are at the same depth.
- **Complete Binary tree** (as seen in the right figure below) is an almost-full binary tree; the bottom level of the tree is filling from left to right but may not have its full complement of leaves.



6

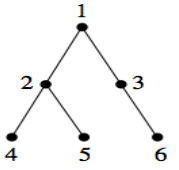
Binary Tree Representation

- Representations for graphs can also be used for trees.
- Binary trees have special characteristics that we want to capture in the representation, namely, the identity of the left and right child

- Equivalent of an adjacency matrix is a **two-column array (n x 2)** where the data for each node (in a row) is the **left and right child of that node**.

7

Binary Tree Representation Example



The tree represented by the figure above has the following

(a) Adjacency Matrix

	Left child	Right child
1	2	3
2	4	5
3	0	6
4	0	0
5	0	0
6	0	0

(a)

8

Binary Tree Representation

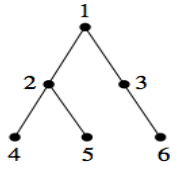
- Representations for graphs can also be used for trees.
- Binary trees have special characteristics that we want to capture in the representation, namely, the identity of the left and right child

- Equivalent of an adjacency matrix is a **two-column array (n x 2)** where the data for each node (in a row) is the **left and right child of that node**.

- Equivalent of the adjacency list representation is a collection of **records with three fields** containing, respectively, the current node, a pointer to the record for the left-child node, and a pointer to the record for the right-child node.

9

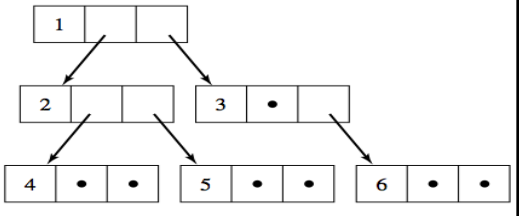
Binary Tree Representation Example



The tree represented by the figure above has the following
(a) Adjacency Matrix and **(b) Adjacency List** representations.

	Left child	Right child
1	2	3
2	4	5
3	0	6
4	0	0
5	0	0
6	0	0

(a)



(b)

10

Tree Traversal Algorithms

- Need a **systematic** mechanism for **writing out the data values stored at all the nodes**.
- Accomplished by ***traversing* the tree**, that is, visiting each of the nodes in the tree structure.

- Three common **tree traversal** algorithms are preorder, inorder, and postorder traversal.
- The terms *preorder, inorder,* and *postorder* refer to the order in which the **root of a tree** is visited compared to the sub-tree nodes.

11

Tree Traversal Algorithms - Preorder

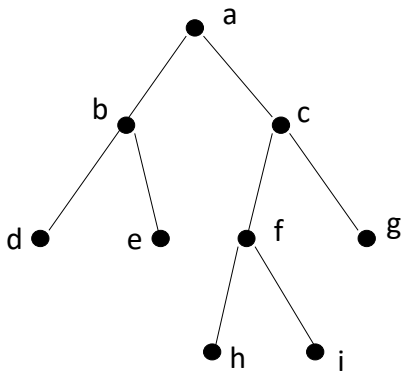
- In **preorder traversal**, the root of the tree is visited first and then the *sub-trees* are processed *left to right*, each in preorder.

- **ALGORITHM Preorder**

```
Preorder(tree T)
//Writes the nodes of a tree with root r in preorder
    write(r)
    for i =1 to t do // labeled left to right
            Preorder(T_i)
    end for
end Preorder
```

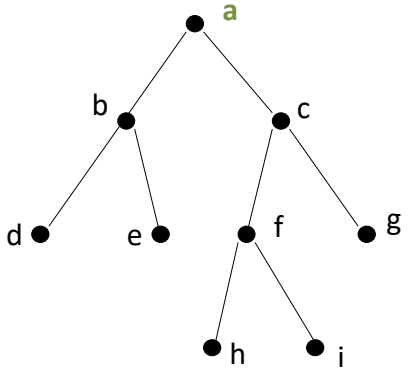

12

Preorder (Root, *Left, Right*) Example – *root is 'a'*



15

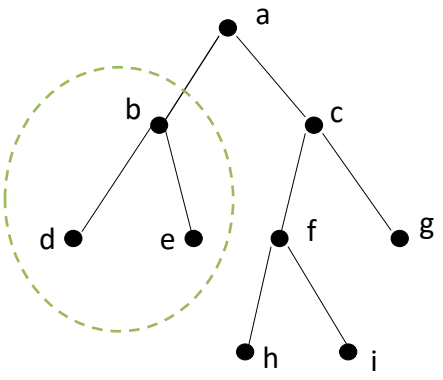
Preorder (Root, *Left, Right*) Example



a

16

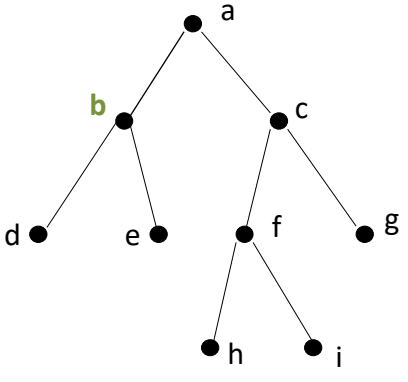
Preorder (Root, *Left, Right*) Example



a

17

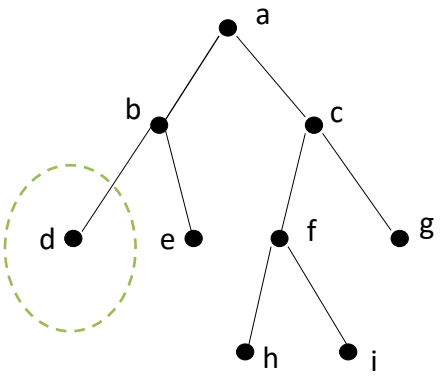
Preorder (Root, *Left, Right*) Example



a, b

18

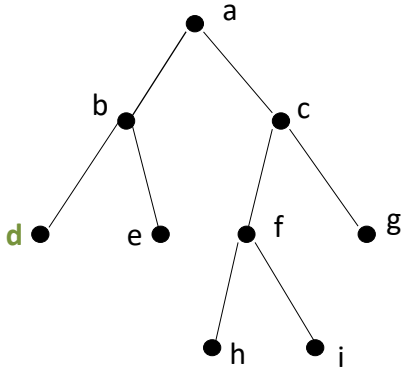
Preorder (Root, *Left, Right*) Example



a, b

19

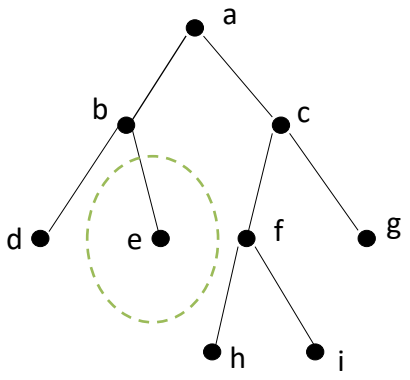
Preorder (Root, *Left, Right*) Example



a, b, d

20

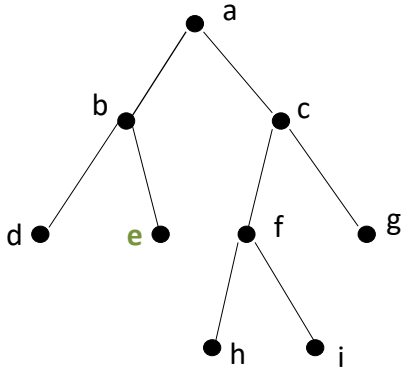
Preorder (Root, *Left, Right*) Example



a, b, d

21

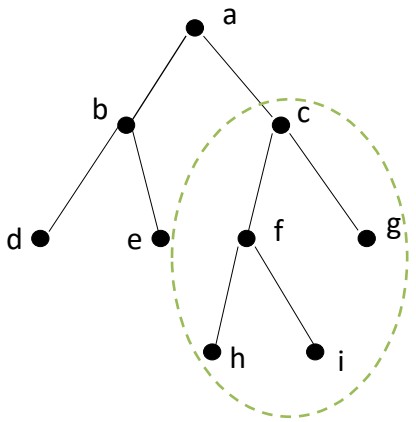
Preorder (Root, *Left, Right*) Example



a, b, d, e

22

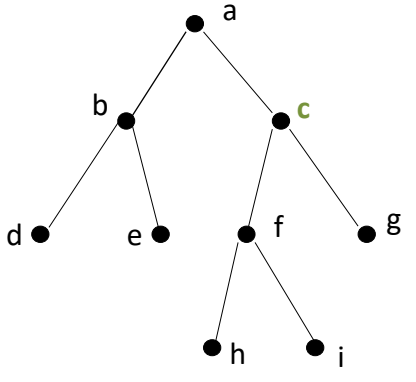
Preorder (Root, *Left, Right*) Example



a, b, d, e

23

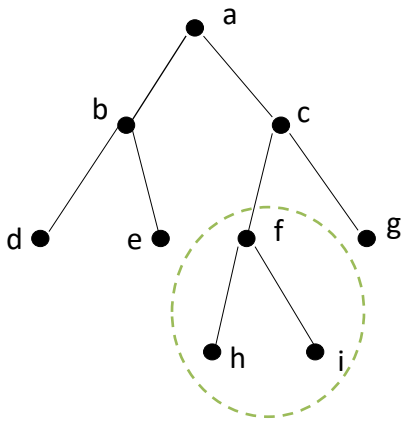
Preorder (Root, *Left, Right*) Example



a, b, d, e, c

24

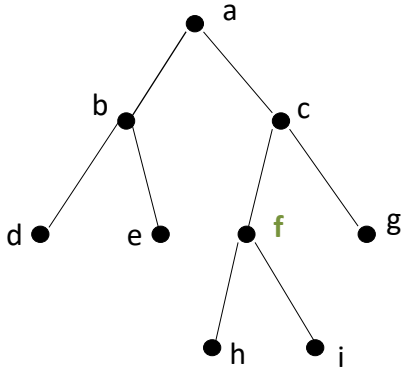
Preorder (Root, *Left, Right*) Example



a, b, d, e, c

25

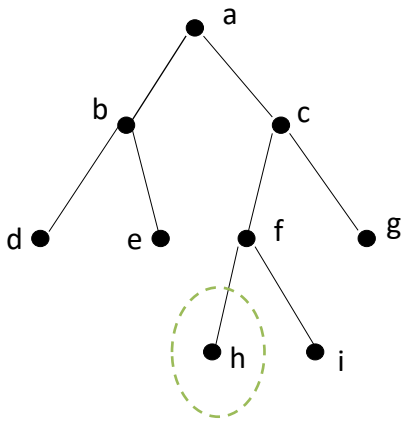
Preorder (Root, *Left, Right*) Example



a, b, d, e, c, f

26

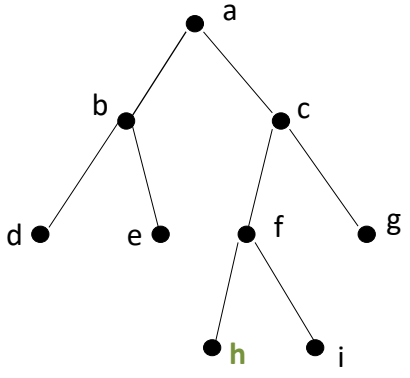
Preorder (Root, *Left, Right*) Example



a, b, d, e, c, f

27

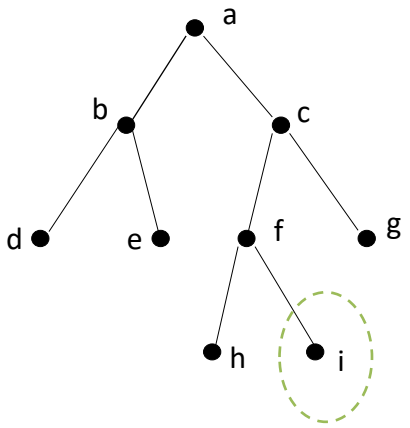
Preorder (Root, *Left, Right*) Example



a, b, d, e, c, f, h

28

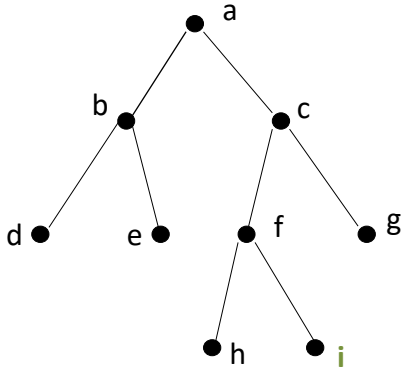
Preorder (Root, *Left, Right*) Example



a, b, d, e, c, f, h

29

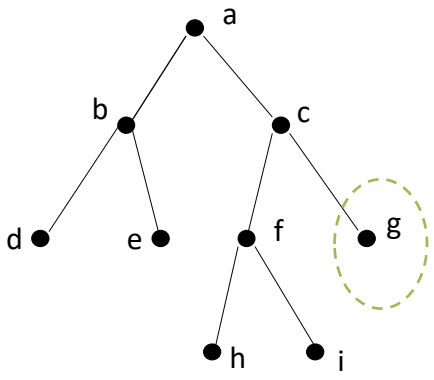
Preorder (Root, *Left, Right*) Example



a, b, d, e, c, f, h, i

30

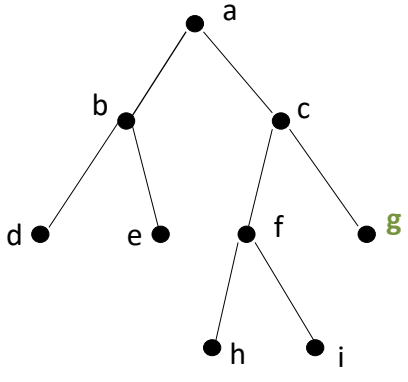
Preorder (Root, *Left, Right*) Example



a, b, d, e, c, f, h, i

31

Preorder (Root, *Left, Right*) Example



a, b, d, e, c, f, h, i, g

Final Output

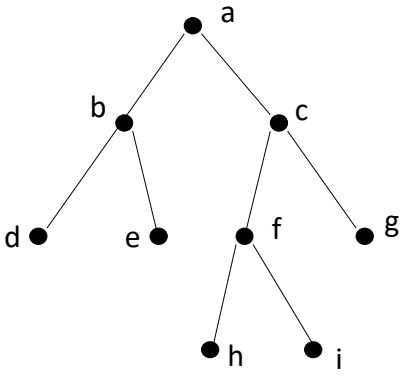
32

Tree Traversal Algorithms - Inorder

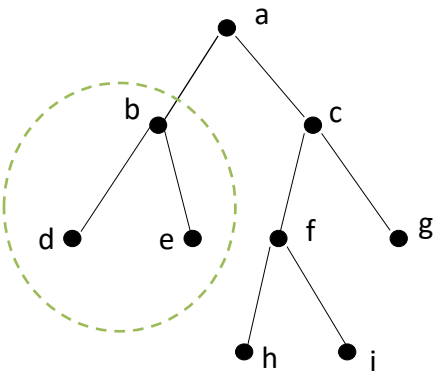
- In **inorder traversal**, the *left sub-tree* is processed by an inorder traversal, then *the root is visited*, and then the remaining sub-trees are *processed from left to right*, each in inorder
  - If the tree is binary, the result is that the **root is visited between processing of the two sub-trees**.
- **ALGORITHM Inorder**

```
Inorder(tree T )
//Writes the nodes of a tree with root r in inorder
Inorder(T1)
      write(r)
      for i =2 to t do // labeled from left to right
                  Inorder(Ti)
      end for
end Inorder
```


Inorder (*Left,* Root, *Right*) Example – *root is 'a'*

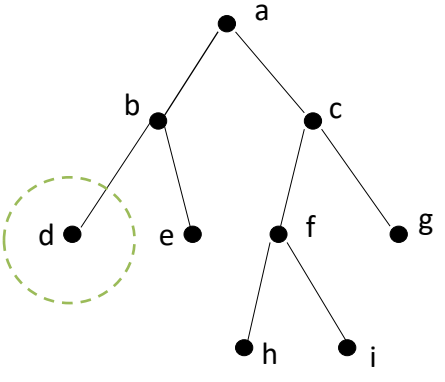


Inorder (*Left,* Root, *Right*) Example



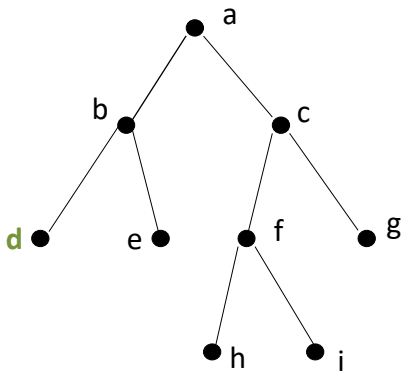
35

Inorder (*Left,* Root, *Right*) Example



36

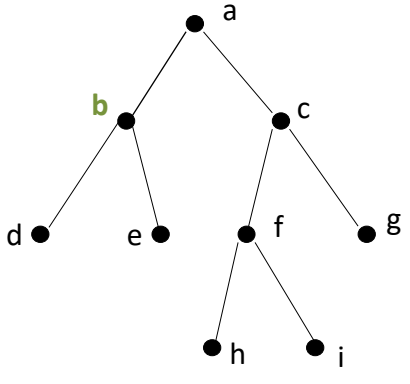
Inorder (*Left,* Root, *Right*) Example



d

37

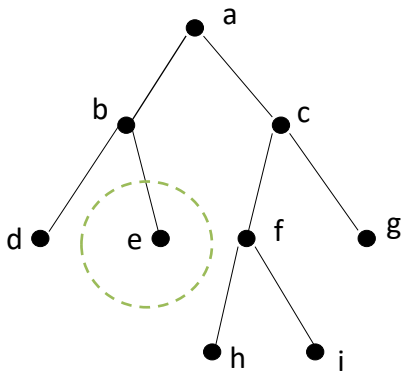
Inorder (*Left,* Root, *Right*) Example



d, b

38

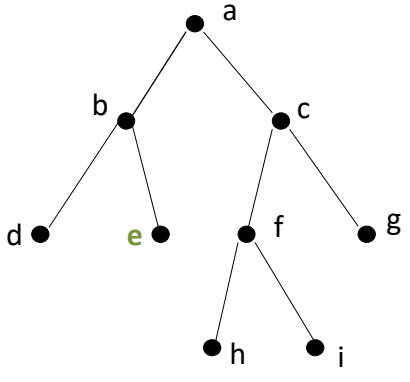
Inorder (*Left,* Root, *Right*) Example



d, b

39

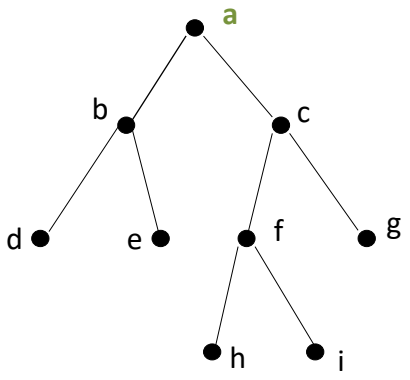
Inorder (*Left,* Root, *Right*) Example



d, b, e

40

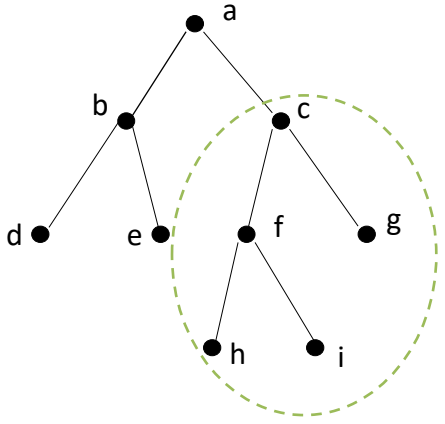
Inorder (*Left,* Root, *Right*) Example



d, b, e, a

41

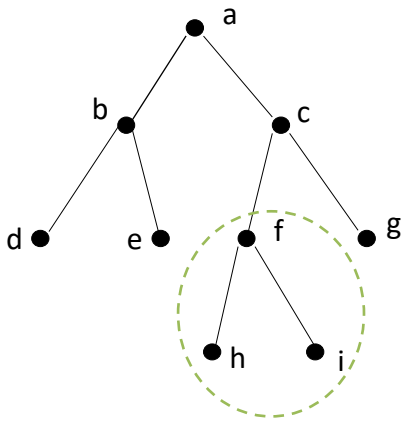
Inorder (*Left,* Root, *Right*) Example



d, b, e, a

42

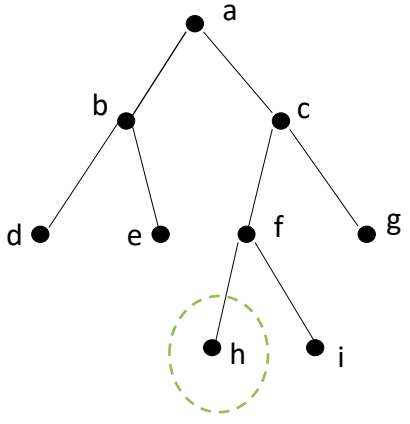
Inorder (*Left,* Root, *Right*) Example



d, b, e, a

43

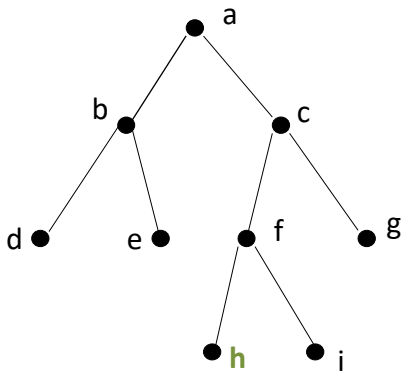
Inorder (*Left,* Root, *Right*) Example



d, b, e, a

44

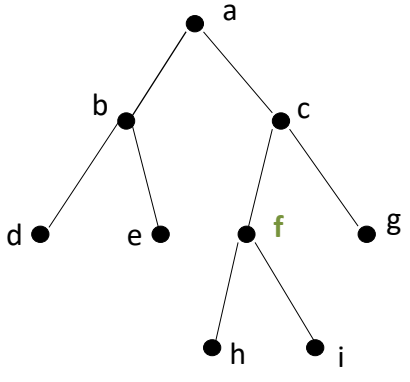
Inorder (*Left,* Root, *Right*) Example



d, b, e, a, h

45

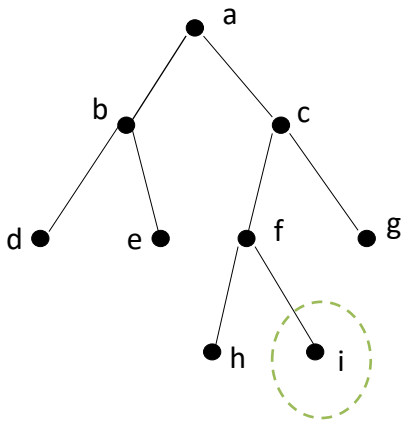
Inorder (*Left,* Root, *Right*) Example



d, b, e, a, h, f

46

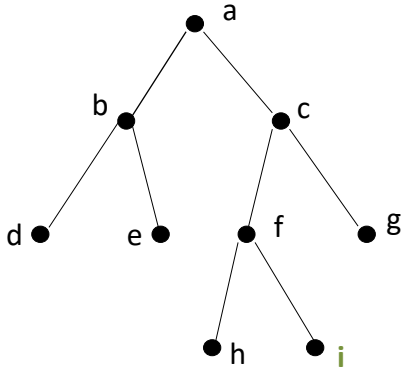
Inorder (*Left,* Root, *Right*) Example



d, b, e, a, h, f

47

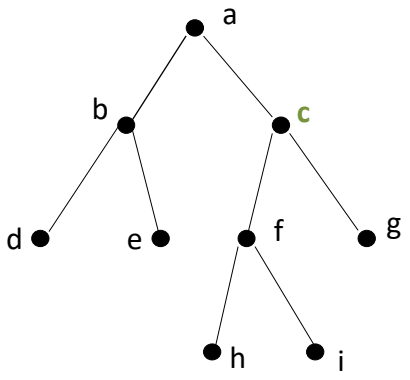
Inorder (*Left,* Root, *Right*) Example



d, b, e, a, h, f, i

48

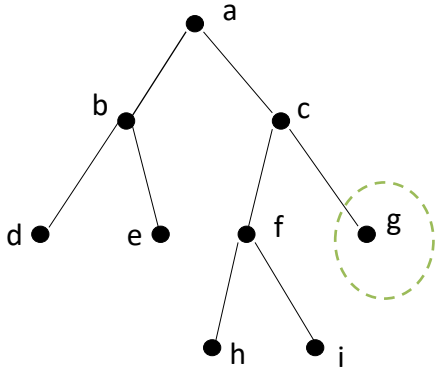
Inorder (*Left,* Root, *Right*) Example



d, b, e, a, h, f, i, c

49

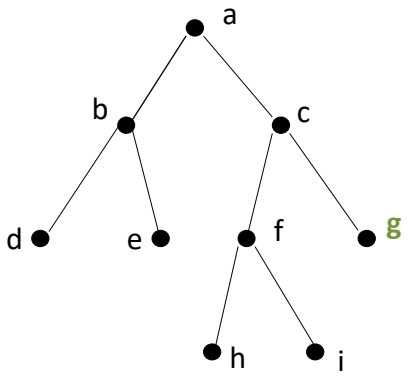
Inorder (*Left,* Root, *Right*) Example



d, b, e, a, h, f, i, c

50

Inorder (*Left,* Root, *Right*) Example



d, b, e, a, h, f, i, c, g

Final Output

51

Tree Traversal Algorithms - Postorder

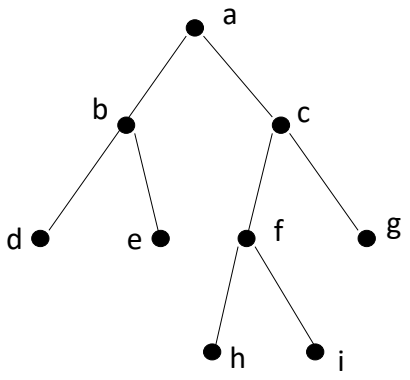
- In **postorder traversal,** the *root is visited last*, after all sub-trees have been processed from *left to right in postorder*.

- **ALGORITHM Postorder**

```
Postorder(tree T )
//Writes the nodes of a tree with root r in postorder
    for i =1 to t do // labeled from left to right
            Postorder(T_i)
    end for
    write(r)
end Postorder
```

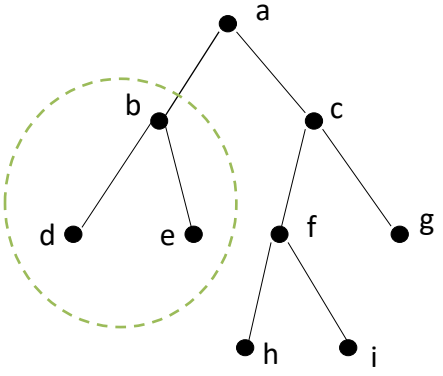

52

Postorder (*Left*, *Right,* Root) Example – *root is 'a'*



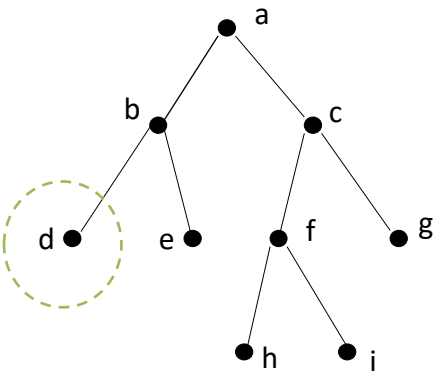
53

Postorder (*Left*, *Right,* Root) Example



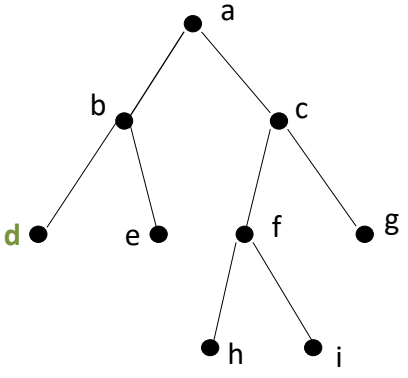
54

Postorder (*Left*, *Right*, Root) Example



55

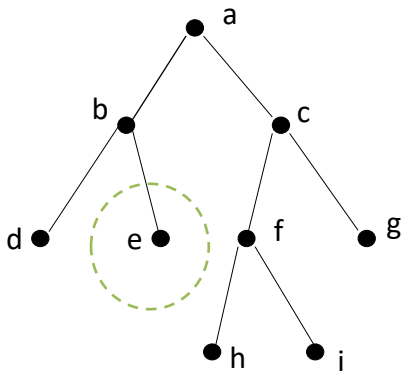
Postorder (*Left*, *Right*, Root) Example



d

56

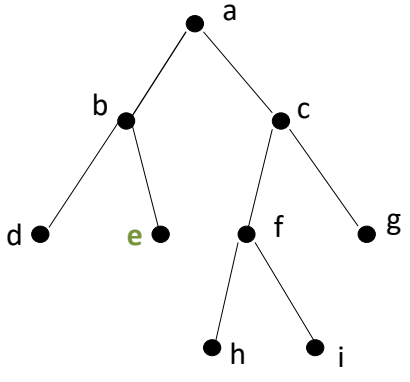
Postorder (*Left*, *Right*, Root) Example



d

57

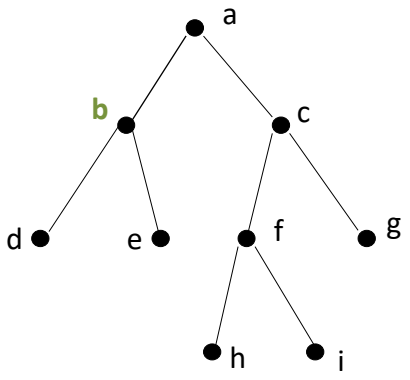
Postorder (*Left*, *Right*, Root) Example



d, e

58

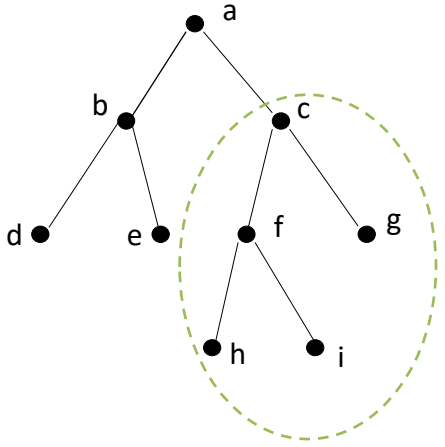
Postorder (*Left*, *Right*, Root) Example



d, e, b

59

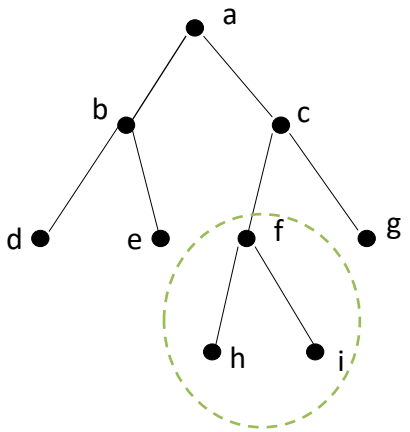
Postorder (*Left*, *Right*, Root) Example



d, e, b

60

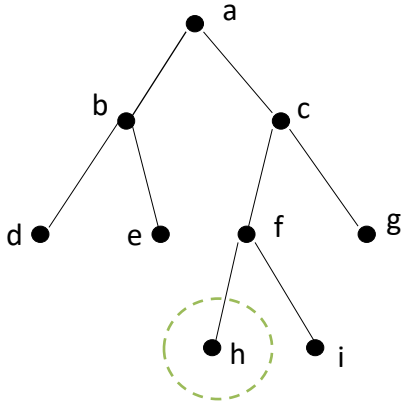
Postorder (*Left*, *Right*, Root) Example



d, e, b

61

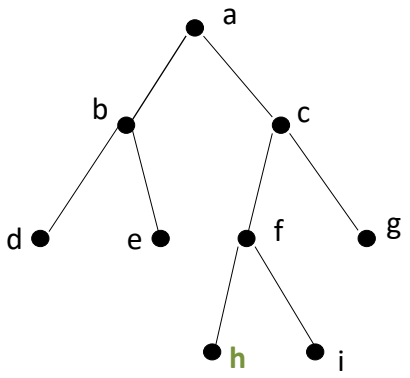
Postorder (*Left*, *Right*, Root) Example



d, e, b

62

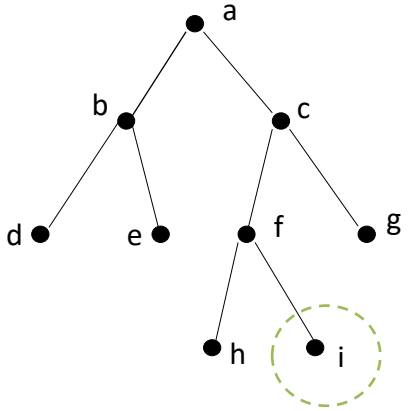
Postorder (*Left*, *Right*, Root) Example



d, e, b, h

63

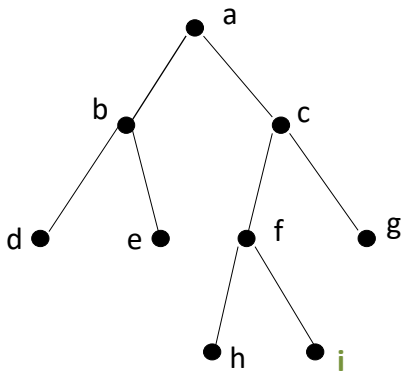
Postorder (*Left*, *Right*, Root) Example



d, e, b, h

64

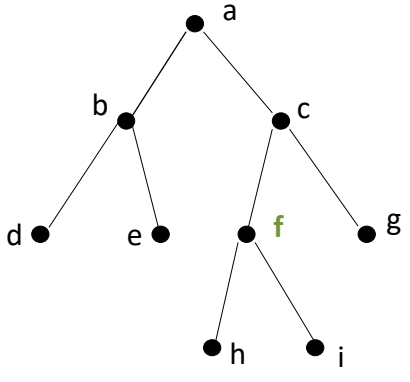
Postorder (*Left*, *Right*, Root) Example



d, e, b, h, i

65

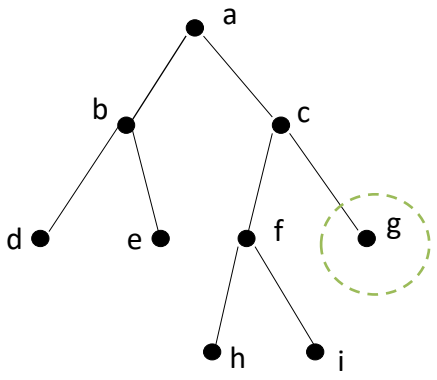
Postorder (*Left*, *Right*, Root) Example



d, e, b, h, i, f

66

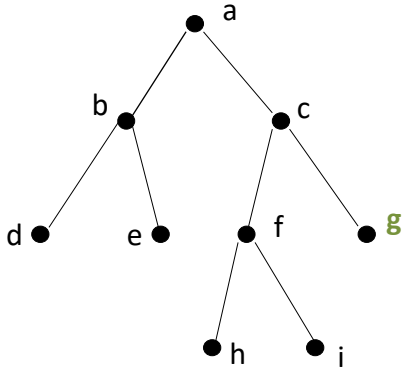
Postorder (*Left*, *Right,* Root) Example



d, e, b, h, i, f

67

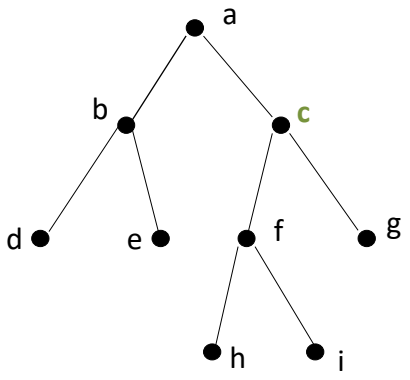
Postorder (*Left*, *Right,* Root) Example



d, e, b, h, i, f, g

68

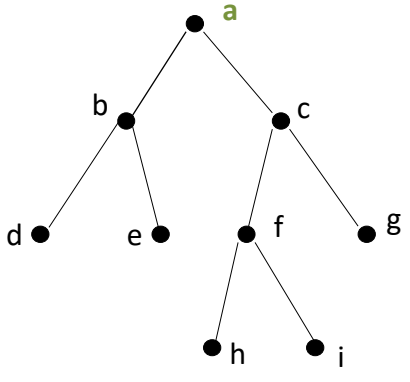
Postorder (*Left*, *Right*, Root) Example



d, e, b, h, i, f, g, c

69

Postorder (*Left*, *Right*, Root) Example



d, e, b, h, i, f, g, c, a

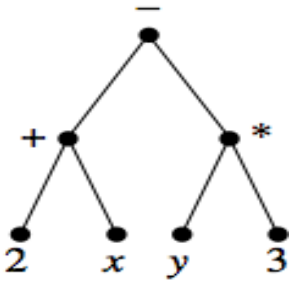
Final Output

70

Infix Notation

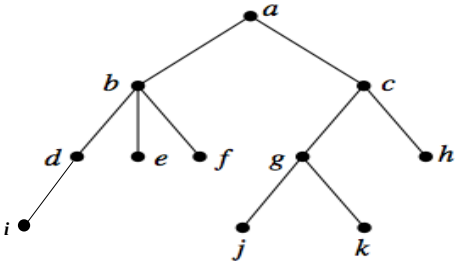
- Apply **inorder traversal** to the binary tree in the figure (known as expression tree)
  - [IMPORTANT] *Parentheses are added as we complete the processing of a sub-tree.*
- Output is the algebraic expression:

  (2 + *x*) – (*y* * 3)
- This is called **infix notation.**



Class/Home Exercise

- What is the preorder, inorder, and postorder traversal for the following tree (**root is *a***) ?



- The preorder (root, left, right) traversal produces: *a, b, d, i, e, f, c, g, j, k, h.*
- The inorder (left, root, right) traversal produces: *i, d, b, e, f, a, j, g, k, c, h.*
- The postorder (left, right, root) traversal produces: *i, d, e, f, b, j, k, g, h, c, a.*

Interesting Real-Life Graph Problems

- What is the shortest (or fastest) route between Dallas and Austin? *Used in Travel Industry and Navigation*
  - *Shortest Path Algorithms*
- Which is the most influential group of people on Social Media? *Used in Advertising*
  - *Dense Subgraphs (Community) Detection*
- Which are the top 10 search results for a Google Search? Most vulnerable computer or device in a network?
  - *Centrality Detection*
- Optimal Match Problems: Job applicants vs. Job Positions, Dating Portals
  - *Bipartite Graph Matching Algorithms*
- ***And many more applications, …***

74

Discussion

75