SCS 3201 DATASTRUCTURES AND ALGORITHMS

TREES

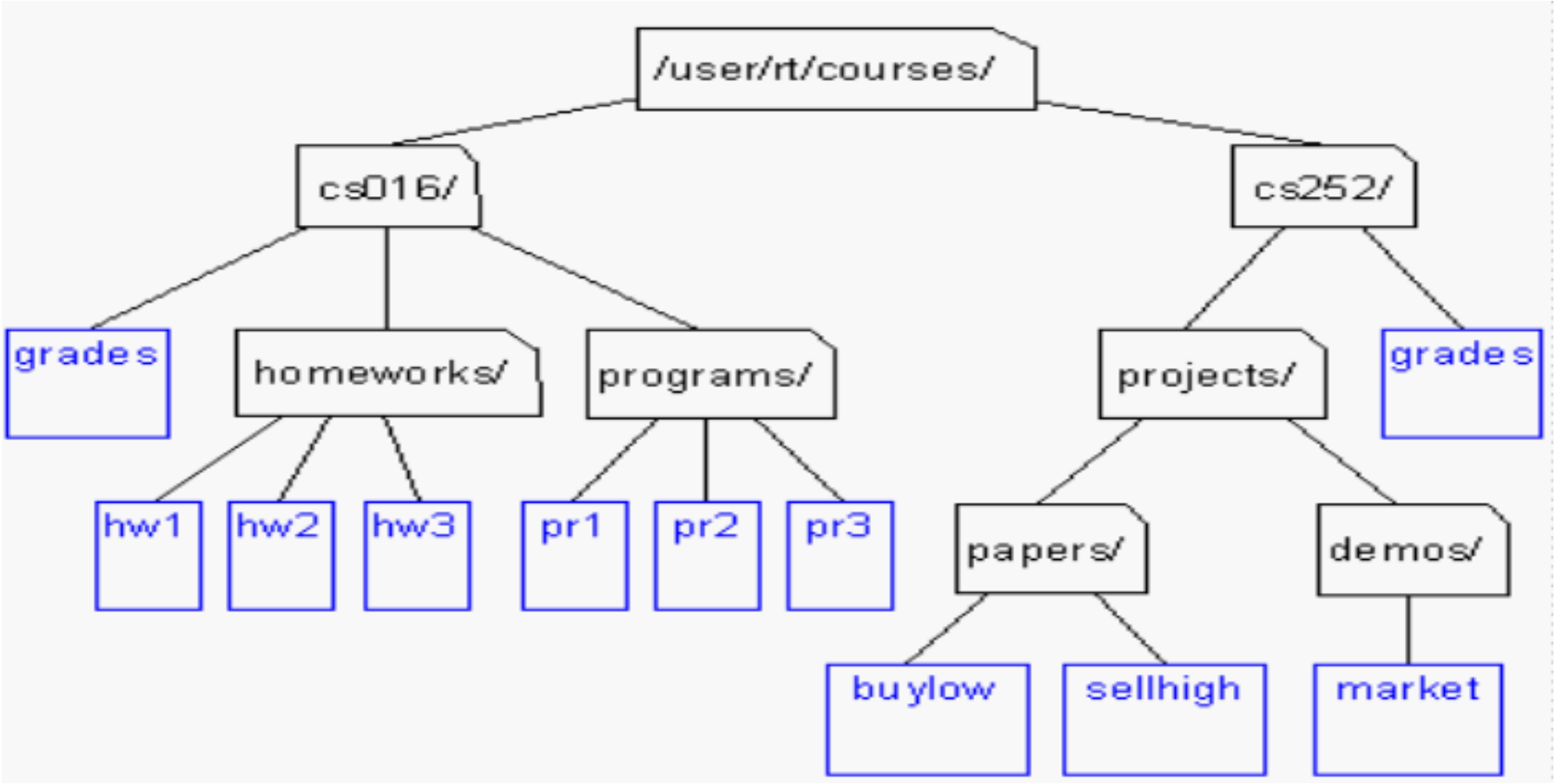
Trees in Data Structures-INTRODUCTION

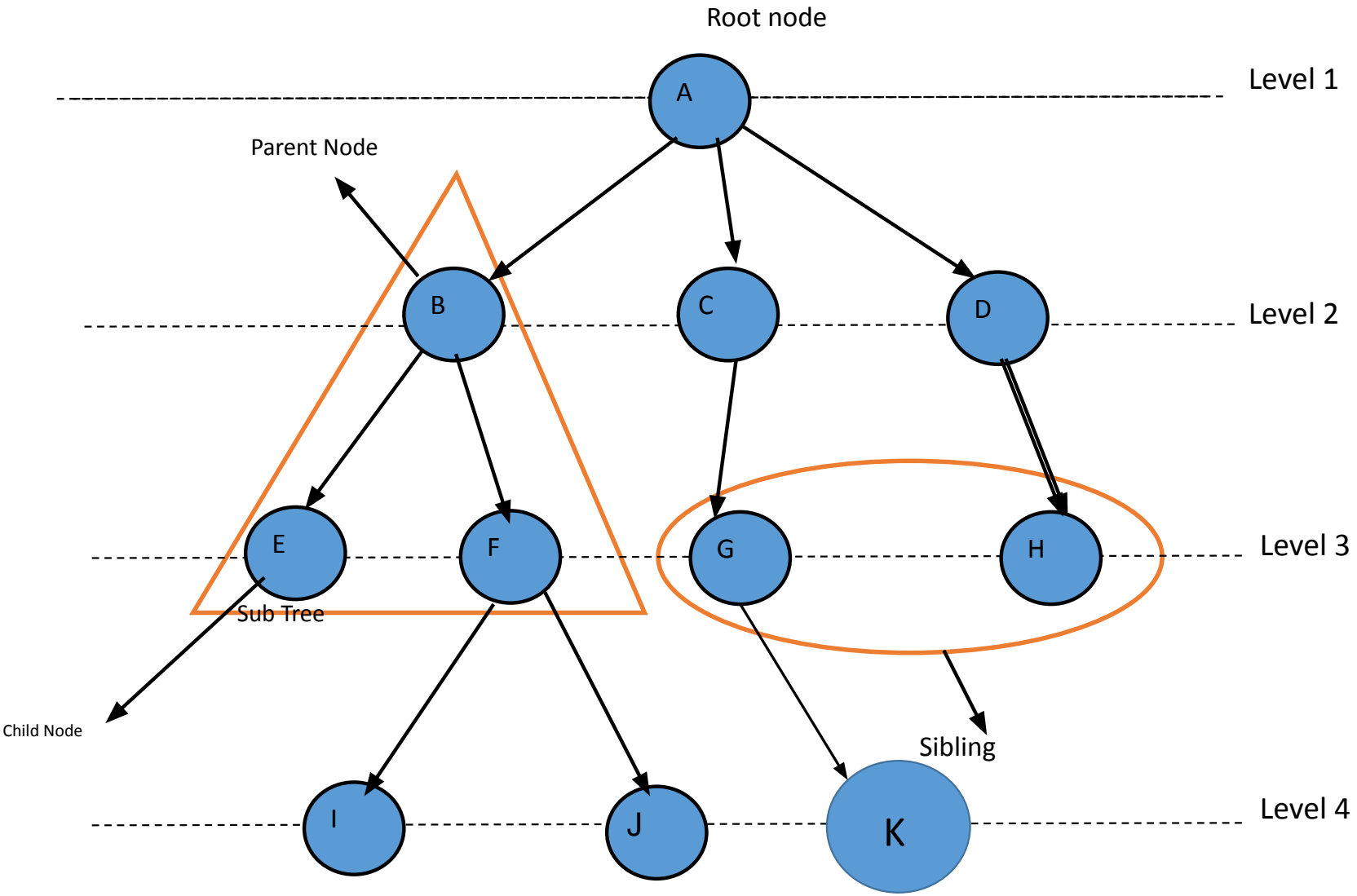
- Tree structure is used to explain hierarchical relationships, e.g. family tree, animal kingdom classification, etc.
- This hierarchical structure of trees is used in Computer science as an abstract data type for various applications like data storage, search and sort algorithms.
- It’s fascinating to know that with the observation of leaves of real world trees and branches of a tree, one came up with a structure to store and organise data in the real world which now is widely being used in multiple application libraries and has really changed the world.

Some key points of the Tree data structure

- A tree data structure is defined as a collection of objects or entities known as nodes that are linked together to represent a hierarchy.
- It's a non linear data structure as it does not store data in a sequential manner, but stores in a hierarchical fashion.
- In the Tree data structure, the first node is known as a root node i.e. from which the tree originates.
- Each node contains some data and also contains references to child nodes.
- A root node can never have a parent node.

Example of a Tree : Unix / Windows file structure




Root node
A
Level 1
Parent Node
B
C
D
Level 2
E
F
G
H
Level 3
Sub Tree
Child Node
Sibling
I
J
K
Level 4

Tree Data Structure Terminologies

Terminology	Description	Diagram
Root	Root node is a node from which the entire tree originates. It does not have a parent	Node A
Parent Node	An immediate predecessor of any node is its parent node.	B is parent of E & F
Child Node	All immediate successors of a node are its children. The relationship between the parent and child is considered as the parent-child relationship.	F & E are children of B

Tree Data Structure Terminologies

Terminology	Description	Diagram
Leaf	Node which does not have any child is a leaf. Usually the boundary nodes of a tree or last nodes of the tree are the leaf or collectively called leaves of the tree.	E, J, K, H, I are the leaf nodes.
Edge	Edge is the connection represented by a line between one node to another. In a tree with n nodes, there will be ‘n-1’ edges in a tree.	Connecting line between A&B OR A&C OR B&F OR any other nodes connecting each other.

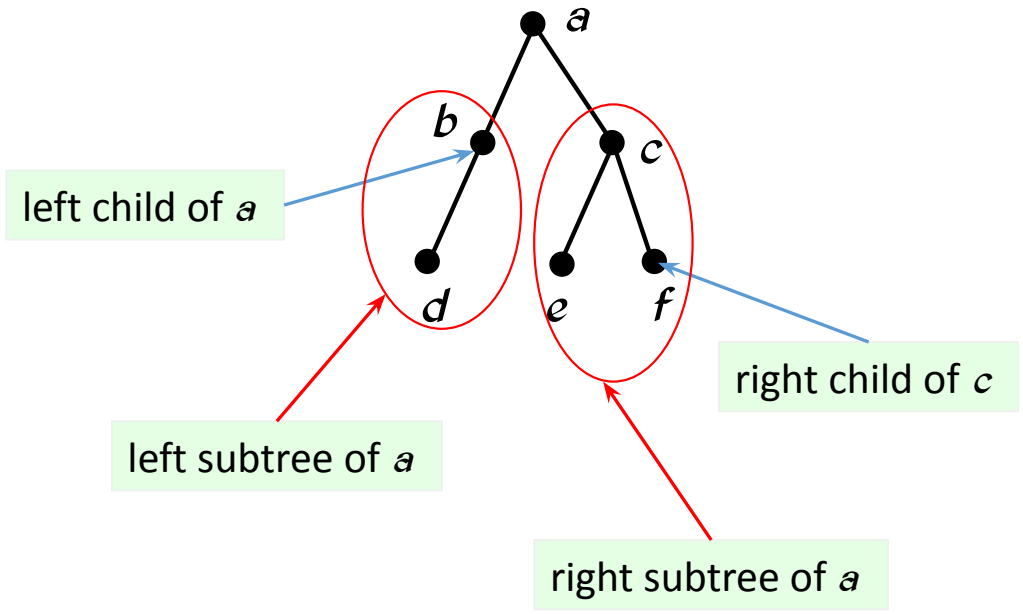
Terminology	Description	Diagram
Siblings	In trees in the data structure, nodes that belong to the same parent are called siblings	H&I are siblings
Path	Path is a number of successive edges from source node to destination node.	A ,C, G, K, is path from node A to K
Height of Node	Height of a node represents the number of edges on the longest path between that node and a leaf.	A, C, G, K form aheight. Height of A is no. of edges between A and K which is 3. Similarly the height of G is 1 as it has just one edge until the next leaf node.

Terminology	Description	Diagram
Levels of node	Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on	Level of H, I & J is 3. Level of D, E, F & G is 2
Degree of Node	Degree of a node implies the number of child nodes a node has.	Degree of D is 1 and of A is 3
Visiting	When you've iterated or traversed to a specific node programmatically, accessing value or checking value of the current node is called visiting.	

Terminology	Description	Diagram
Internal Node	A node that has at least one child is known as an internal node.	All the nodes except E, J, K, H, I are internal.
Traversing	Traversing is a process of visiting each node in a specific order in a tree data structure.	There are three types of traversals: inorder, preorder, postorder traversal.
Ancestor node	An ancestor or ancestors to a node are all the predecessor nodes from root until that node. I.e. any parent or grandparent and so on of a specific node are its ancestors.	A, C & G are ancestor to K nodes

Terminology	Description	Diagram
Descendant	Immediate successor of a node is its descendent.	K is descendent of G
Sub tree	Descendants of a node represent subtree. Tree being a recursive data structure can contain many subtrees inside of it.	Nodes B, E, F represent one subtree.

Def:


a
b
c
d
e
f
left child of a
right child of c
left subtree of a
right subtree of a

The Necessity for a Tree in Data Structures

- Other data structures like arrays, linked-list, stacks, and queues are linear data structures, and all these data structures store data in sequential order.
- Time complexity increases with increasing data size to perform operations like insertion and deletion on these linear data structures. But it is not acceptable for today's world of computation.
- The non-linear structure of trees enhances the data storing, data accessing, and manipulation processes by employing advanced control methods traversal through it.

Advantages of trees

- Trees are so useful and frequently used, because they have some very serious advantages:
- Trees reflect structural relationships in the data
- Trees are used to represent hierarchies
- Trees provide an efficient insertion and searching
- Trees are very flexible data, allowing to move sub trees around with minimum effort

Tree Node

- A node is a structure that contains a key or value and pointers in its child node in the tree data structure.
- In the tree data structure, you can define the tree node as follows:

```
struct node
{
Int data;
struct node *leftchild;
struct node *rightchild;
}
```

Root Node

- The above representation depicts what a tree looks like on a memory.
- According to it, each node consists of three fields.
- Left part of the node consists of the memory address of the left child, the right part of the node consists of the memory address of the right child and the center part holds the data for this node.

```
class Node: def __init__(self, data): self.left = None self.right = None self.data = data
```

Implementing a Tree Node from Scratch

- To create a tree in Python, we first have to start by creating a node, class that will represent a single node. This node class will contain 3 variables; the first is the left pointing to the left child, the second variable data containing the value for that node, and the right pointing to the right child.
- class Node:
  - def _init_(self, data):
    - Self.left = none
    - Self.right = none
    - Self.data = data

Lets initialize the tree

root = node(10)

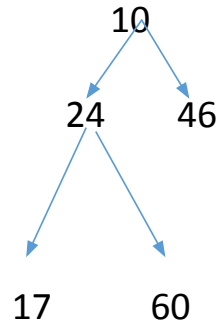
root.left = node (24)

root. right = node(46)

root.left .left = node(17)

root. left.right = node(60)

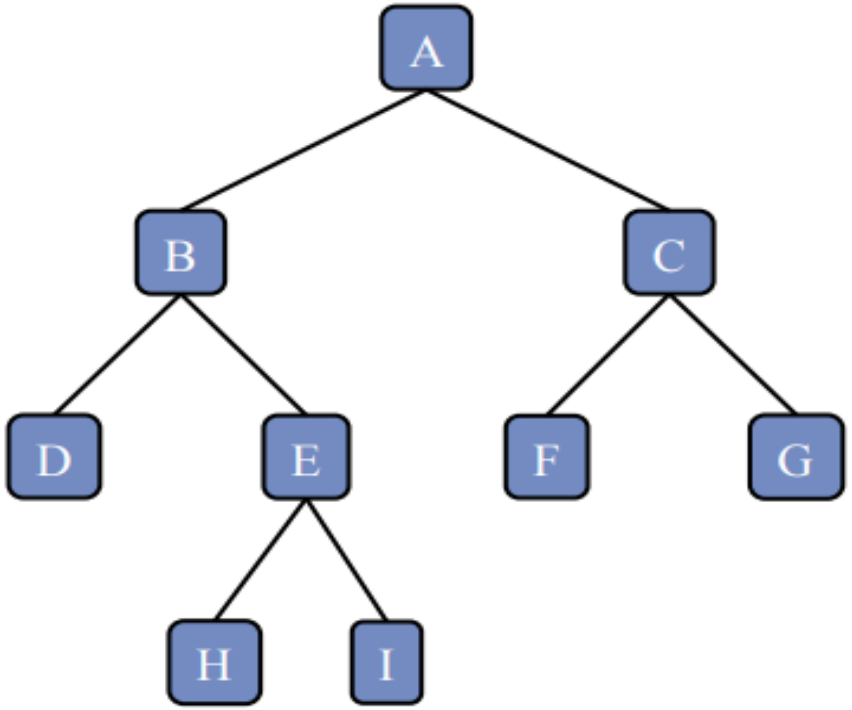
The tree will look like



Binary Trees

- In a normal tree, every node can have any number of children.
- Binary tree is a special type of tree data structure in which every node can have a maximum of 2 children.
- One is known as left child and the other is known as right child.
- A tree in which every node can have a maximum of two children is called as Binary Tree.
- In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.

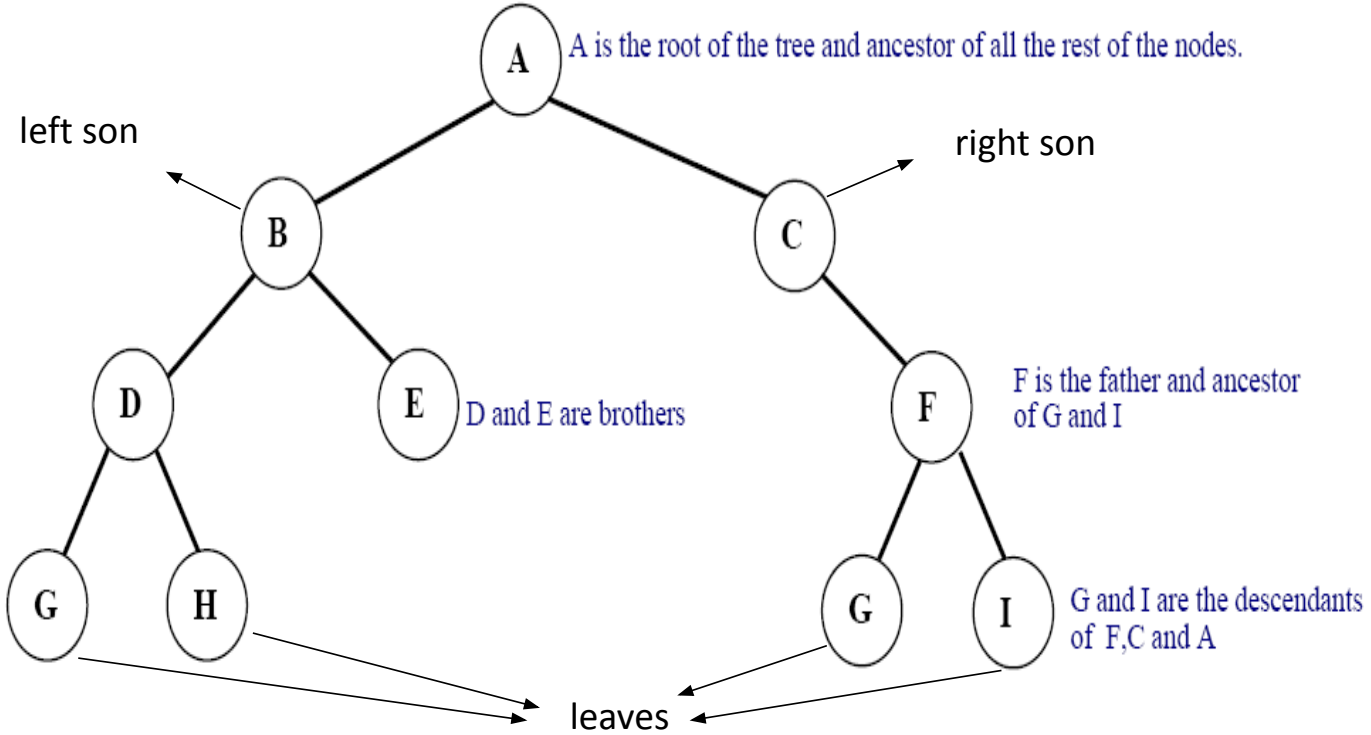
Example of a Binary Tree



BINARY TREES: BASIC DEFINITIONS

- If A is the root of a binary tree and B is the root of its left or right subtrees, then A is said to be the ***father*** of B and B is said to be the ***left son*** of A.
- A node that has no sons is called the ***leaf***.
- Node *n1* is the ***ancestor*** of node *n2* if n1 is either the father of *n2* or the father of some ancestor of *n2*. In such a case *n2* is a ***descendant*** of *n1*.
- Two nodes are ***brothers*** if they are left and right sons of the same father.

BINARY TREES: BASIC DEFINITIONS



Types of Binary Tree

- There are different types of binary trees and they are...

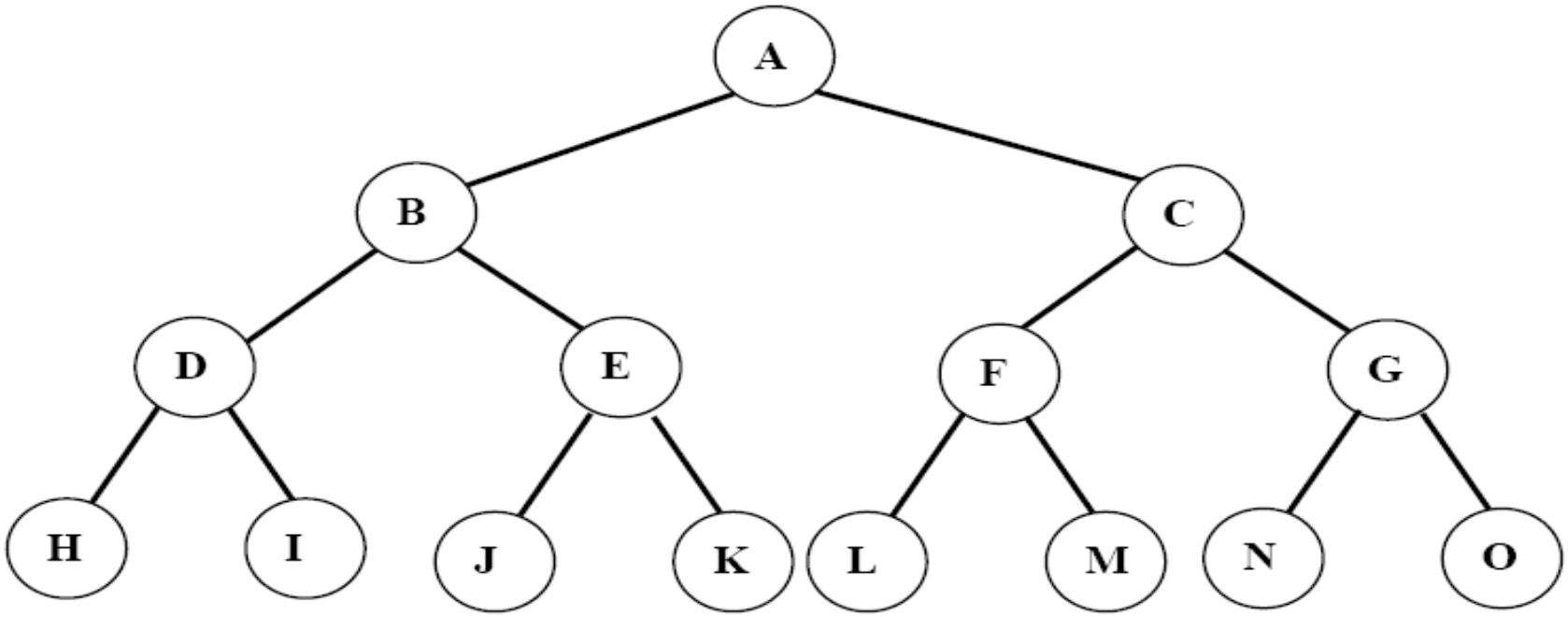
1. **Strictly Binary Tree /Full Binary Tree or Proper Binary Tree or 2-Tree**

- A strictly Binary Tree can be defined as follows... A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree.

2. **Complete Binary Tree /Perfect Binary Tree**

- In a binary tree, every node can have a maximum of two children.
- In complete binary tree all the nodes must have exactly two children and at every level of complete binary tree there must be 2 level number of nodes. For example at level 2 there must be 2^2 = 4 nodes and at level 3 there must be 2^3 = 8 nodes

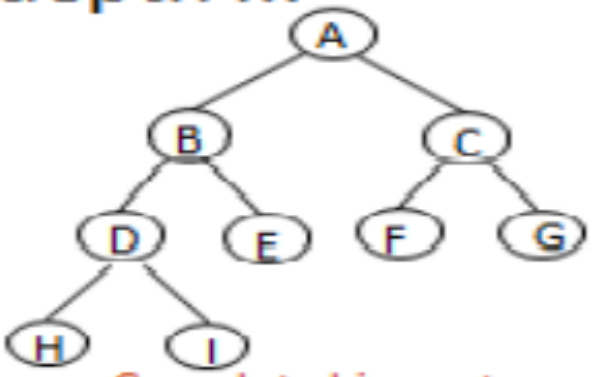
BINARY TREES: BASIC DEFINITIONS



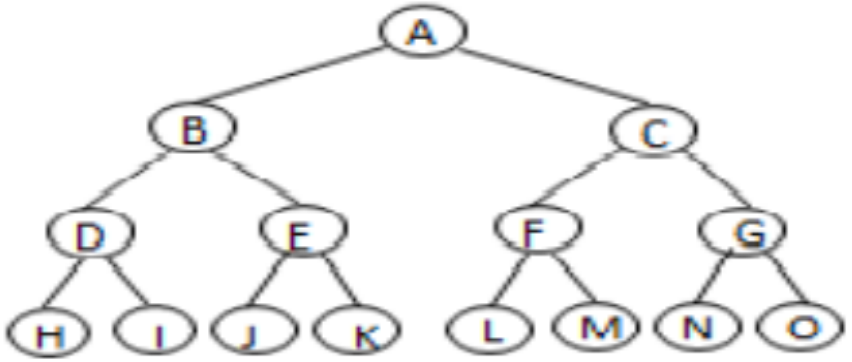
A complete Binary Tree of depth 3

Full BT VS Complete BT

- A full binary tree of depth k is a binary tree of depth k having 2^k-1 nodes, $k>=0$.
- A binary tree with n nodes and depth k is complete *iff* its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k.



Complete binary tree



Full binary tree of depth 4

Types of Binary Tree

3. Extended Binary Tree

- A binary tree can be converted into Full Binary tree by adding dummy nodes to existing nodes wherever required.
- The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.

TRAVERSING BINARY TREES

- One of the common operations of a binary tree is to ***traverse*** the tree. Traversing a tree is to pass through all of its nodes once. You may want to print the contents of each node or to process the contents of the nodes. In either case each node of the tree is visited.
- There are three main traversal methods where traversing a binary tree involves visiting the root and traversing its left and right subtrees. The only difference among these three methods is the order in which these three operations are performed.

Binary Tree Traversal

- Traversal is the process of visiting every node once
- Visiting a node entails doing some processing at that node, but when describing a traversal strategy, we need not concern ourselves with what that processing is

Preoder, Inorder, Postorder

- In Preorder, the root
  is visited before (pre)
  the subtrees traversals
- In Inorder, the root is
  visited in-between left
  and right subtree traversal
- In Preorder, the root
  is visited after (pre)
  the subtrees traversals

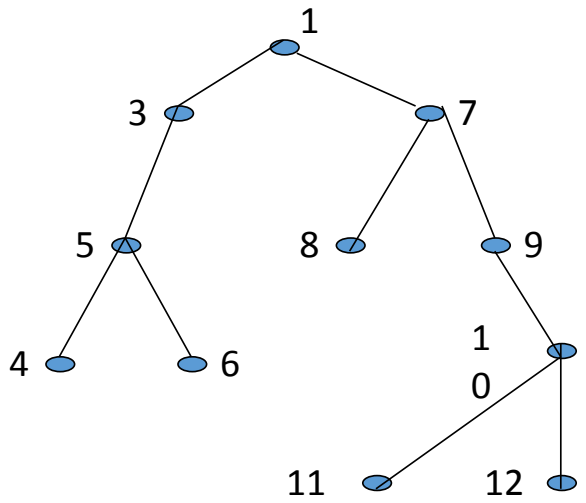
Preorder Traversal:
1. Visit the root
2. Traverse left subtree
3. Traverse right subtree

Inorder Traversal:
1. Traverse left subtree
2. Visit the root
3. Traverse right subtree

Postorder Traversal:
1. Traverse left subtree
2. Traverse right subtree
3. Visit the root

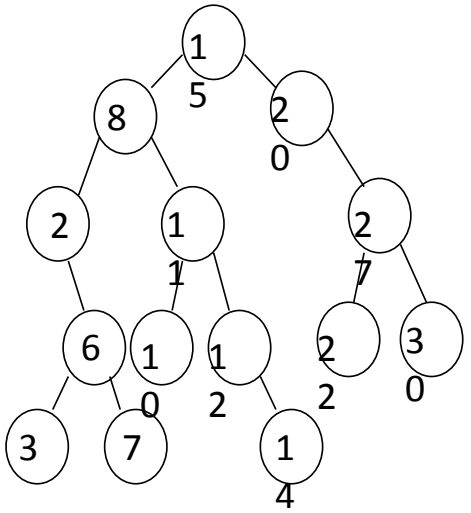
Illustrations for Traversals

- Assume: visiting a node
  is printing its label
- Preorder:
  1 3 5 4 6 7 8 9 10 11 12
- Inorder:
  4 5 6 3 1 8 7 9 11 10 12
- Postorder:
  4 6 5 3 8 11 12 10 9 7 1



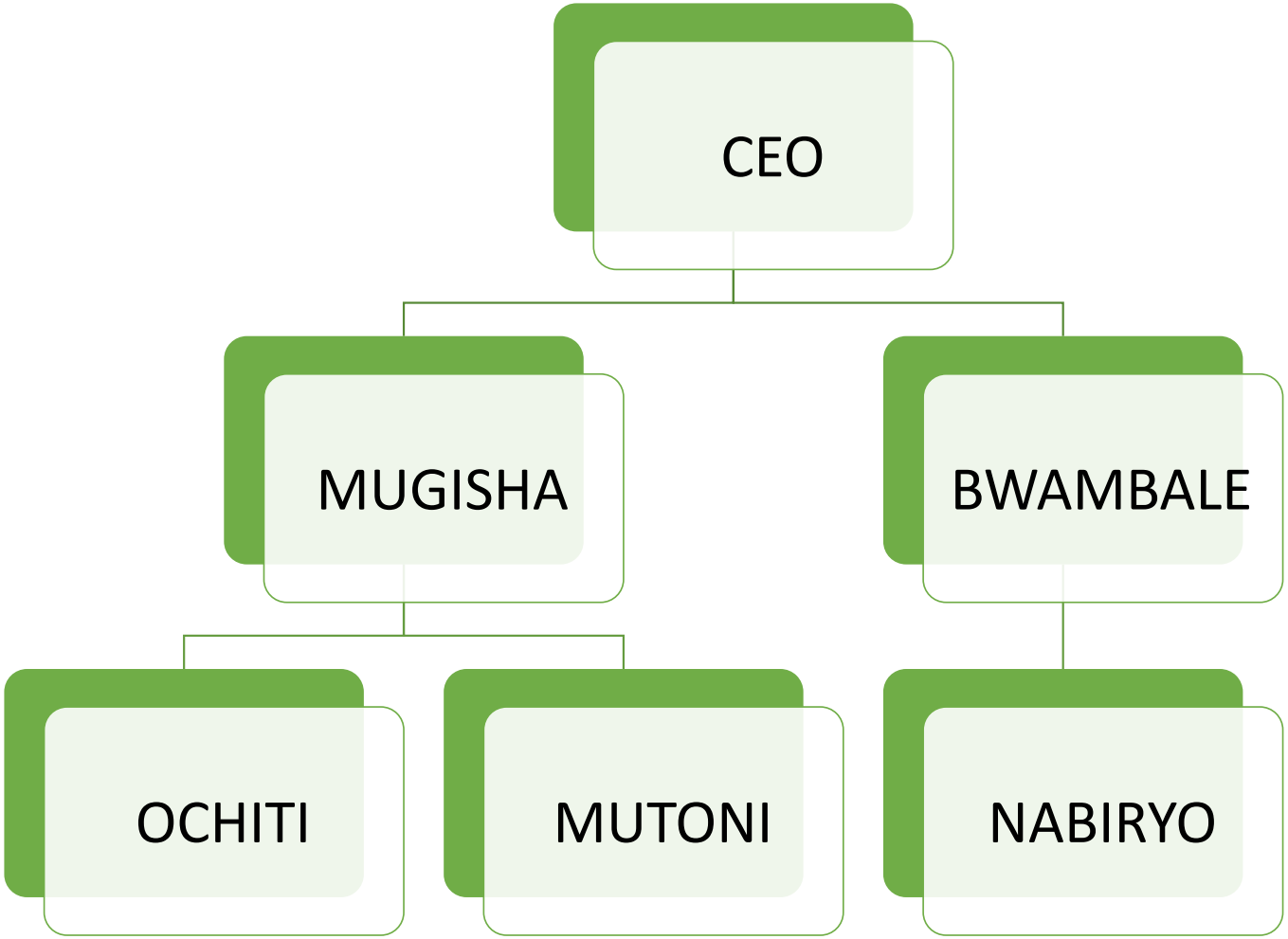
Illustrations for Traversals (Contd.)

- Assume: visiting a node
  is printing its data
- Preorder: 15 8 2 6 3 7
  11 10 12 14 20 27 22 30
- Inorder: 2 3 6 7 8 10 11
  12 14 15 20 22 27 30
- Postorder: 3 7 6 2 10 14
  12 11 8 22 30 27 20 15



Case study

Mrs. Otim Charles, the CEO, believes that any officer of the company should not have more than two direct reports. She herself has two vice presidents, Mugisha and Bwambale, reporting to her. Each vice president can have up to two managers. In this case, Mutoni and Ochiti report to Mugisha, and Nabiryo reports to Bwambale.


CEO
MUGISHA
BWAMBALE
OCHITI
MUTONI
NABIRYO

Postorder traversal sequence

The postorder traversal sequence is “OCHITI”, “MUTONI”, “MUGISHA”, “NABIRYO”, “BWAMBALE”, “CEO”.

The order is indicated in the figure as sequence numbers below each node.

inorder traversal

The inorder traversal sequence is “OCHITI”, “MUGISHA”, “MUTONI”, “CEO”, “NABIRYO”, “BWAMBALE”.

The order is indicated in the figure as sequence numbers below each node.

Pre order traversal

By now we have visited all the nodes in the following order: “CEO”, “MUGISHA”, “OCHITI”, “MUTONI”,

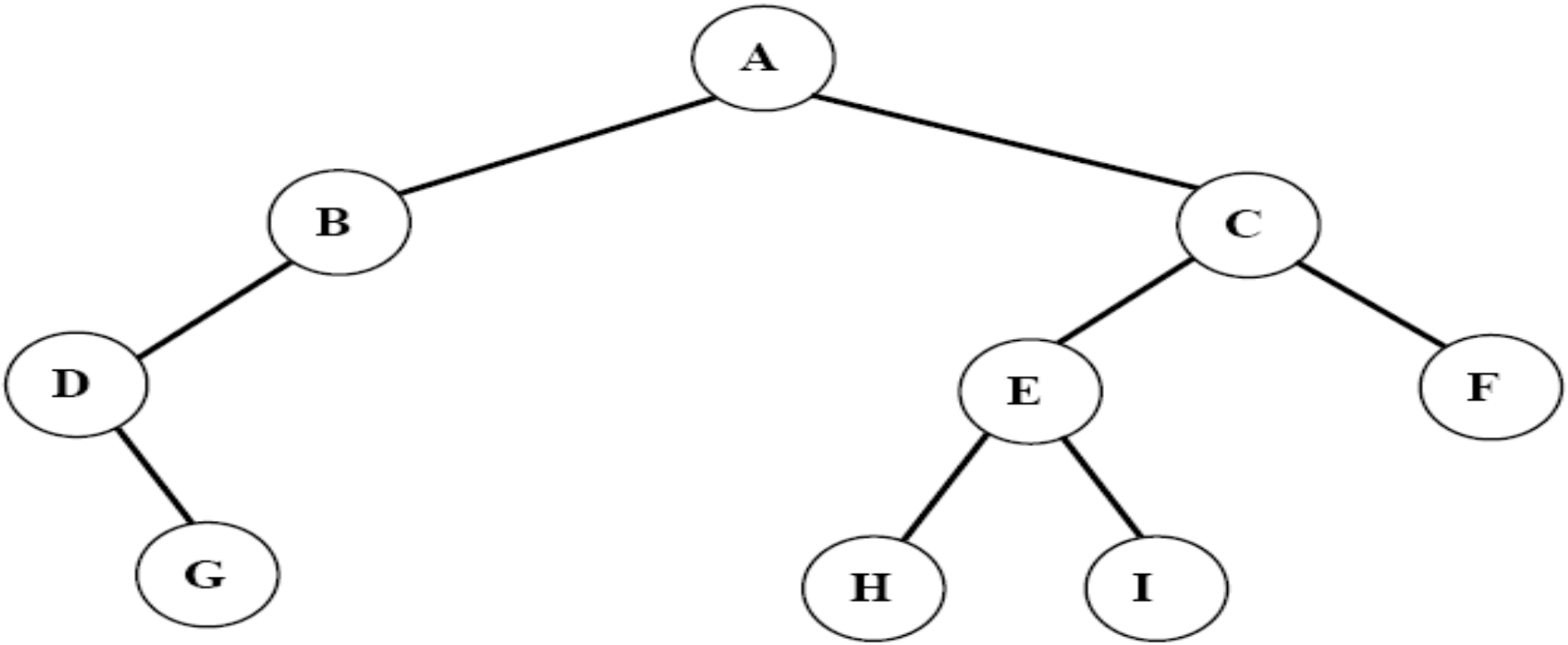
“BWAMBALE”, “NABIRYO”.

The order is indicated in in the figure as sequence numbers below each node.

TRAVERSING BINARY TREES

- Traversing a binary tree in ***preorder*** (depth-first order)
  1. Visit the ***root***.
  2. Traverse the ***left subtree*** in preorder.
  3. Traverse the ***right subtree*** in preorder.

Traversing a binary tree in *preorder*



Preorder*: ABDGCEHIF*

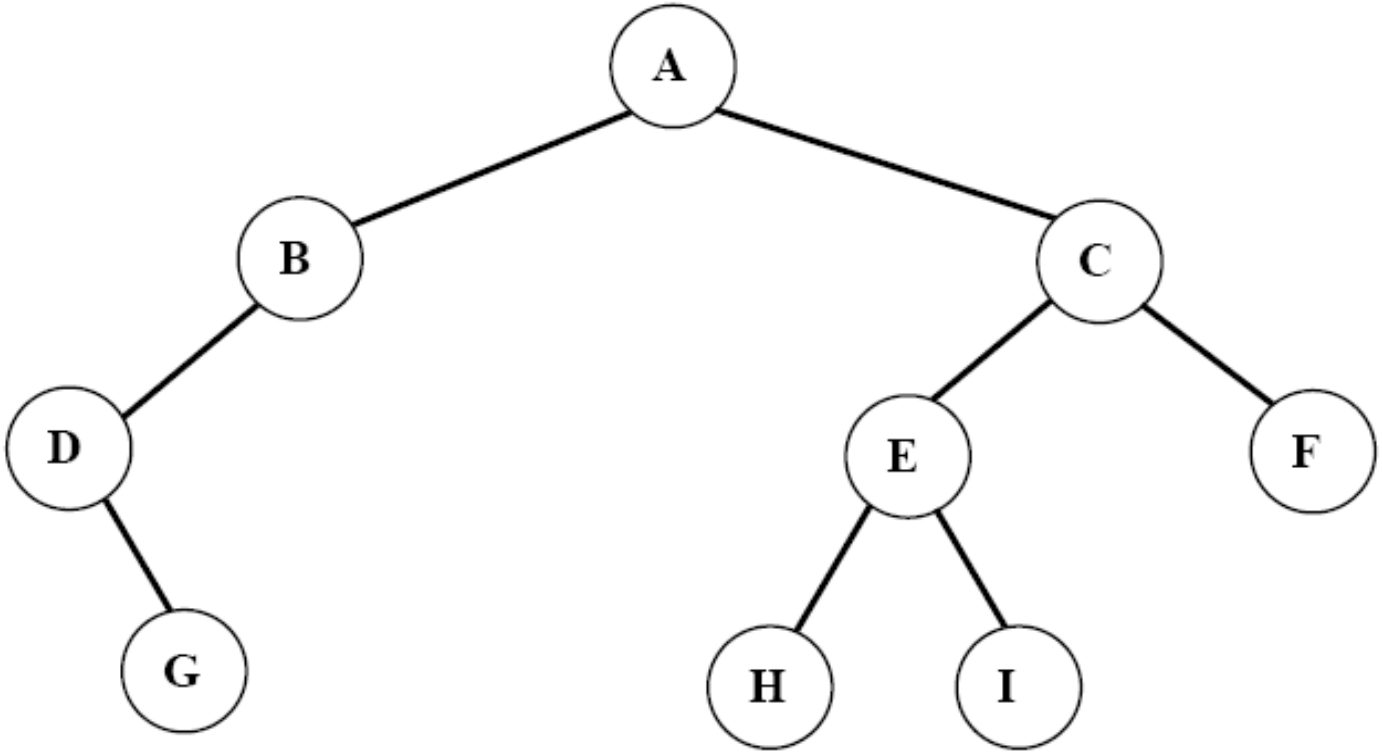
TRAVERSING BINARY TREES

Traversing a binary tree in ***inorder***

(or symmetric order)

1. Traverse the ***left subtree*** in inorder.
2. Visit the ***root***.
3. Traverse the ***right subtree*** in inorder.

Traversing a binary tree in *inorder*

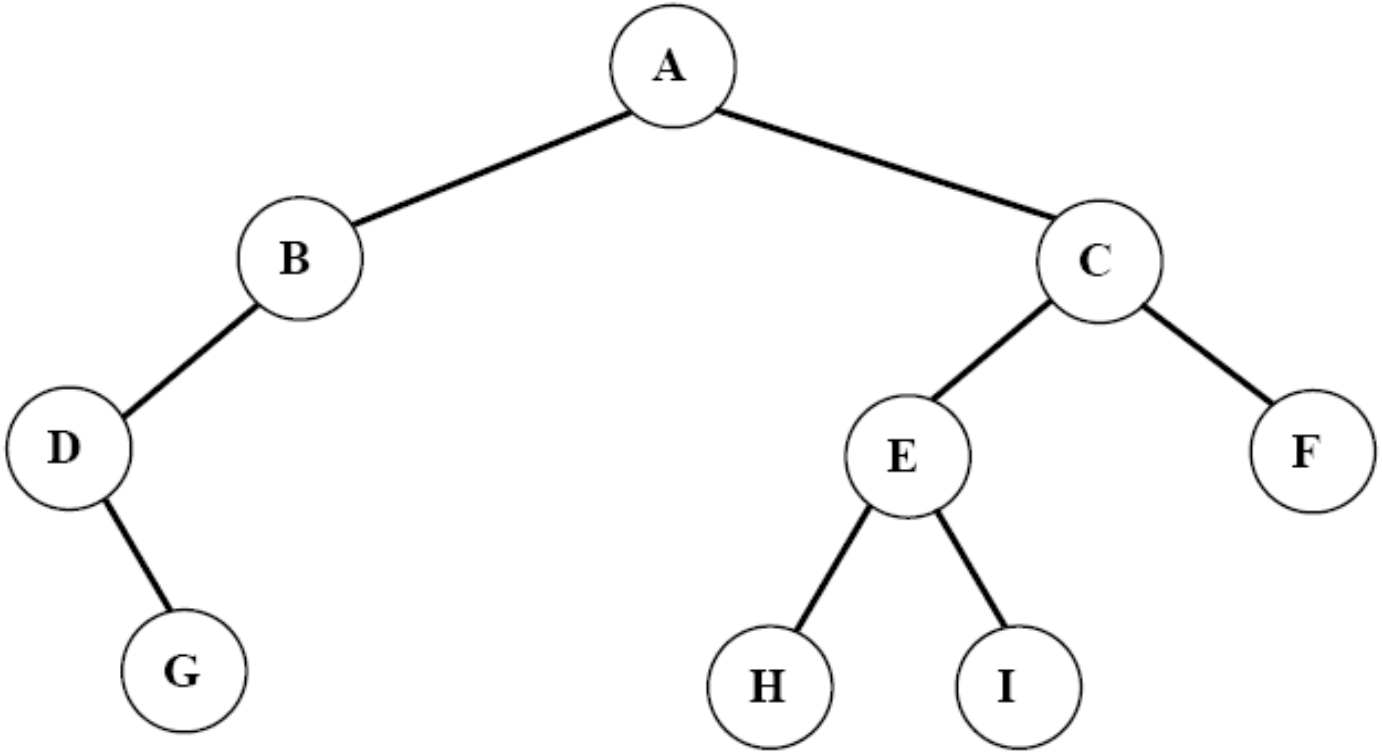


Inorder*: DGBAHEICF*

TRAVERSING BINARY TREES

- Traversing a binary tree in ***postorder***
  1. Traverse the ***left subtree*** in postorder.
  2. Traverse the ***right subtree*** in postorder.
  3. Visit the ***root***.

Traversing a binary tree in *postorder*



Postorder: *GDBHIEFCA*

Code for the Traversal Techniques

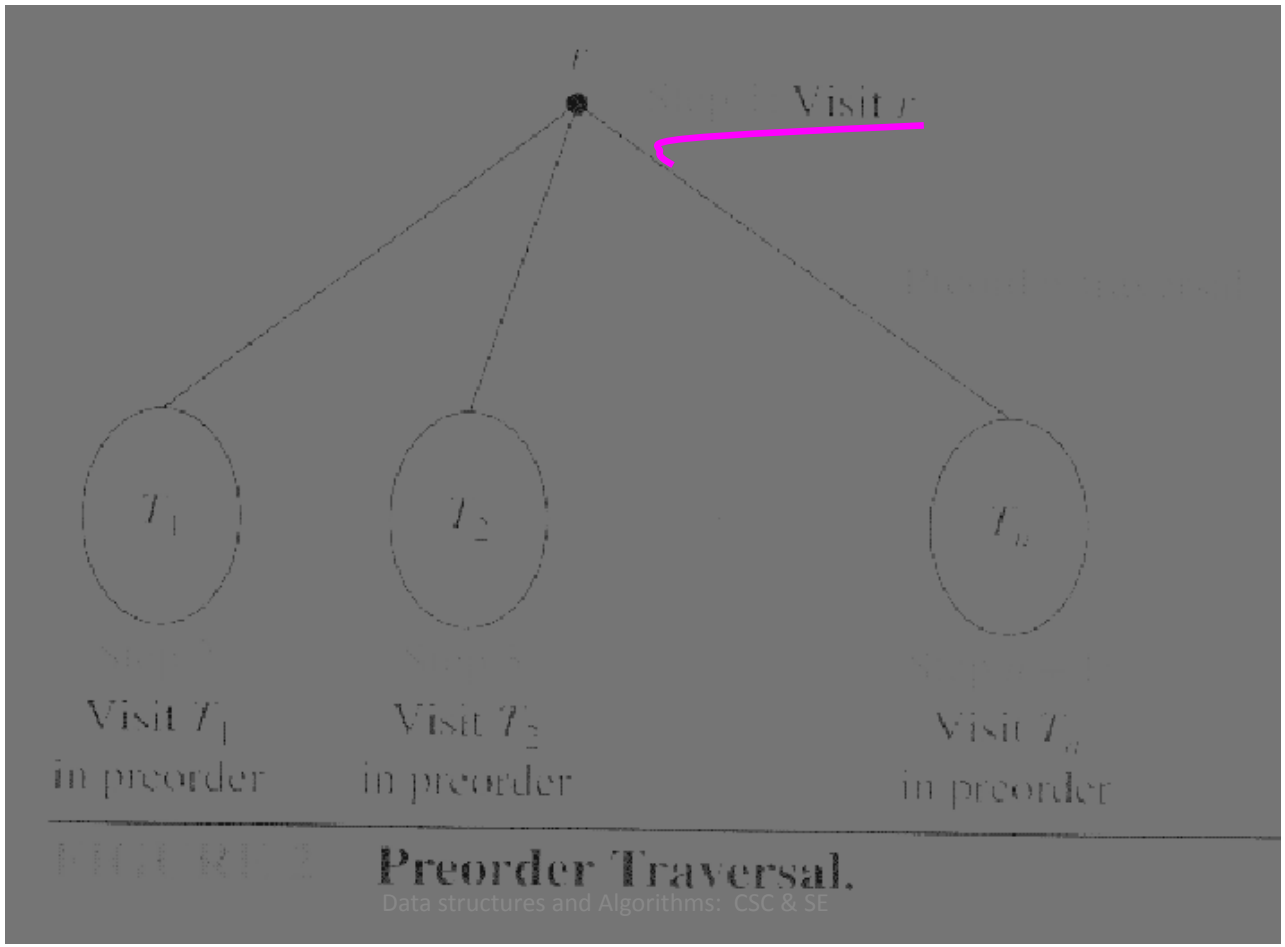
- The code for visit is up to you to provide, depending on the application
- A typical example for visit(…) is to print out the data part of its input node

```
void preOrder(Tree *tree){
   if (tree->isEmpty( ))      return;
   visit(tree->getRoot( ));
   preOrder(tree->getLeftSubtree());
                              preOrder(tree-
>getRightSubtree());
}
```

```
void inOrder(Tree *tree){
   if (tree->isEmpty( ))      return;
   inOrder(tree->getLeftSubtree( ));
   visit(tree->getRoot( ));
   inOrder(tree->getRightSubtree( ));
}
```

```
void postOrder(Tree *tree){
   if (tree->isEmpty( ))      return;
     postOrder(tree->getLeftSubtree(
));
    postOrder(tree->getRightSubtree(
));
   visit(tree->getRoot( ));
}
```

Steps for Preorder Traversal



Preorder Traversal.

Inorder Traversal

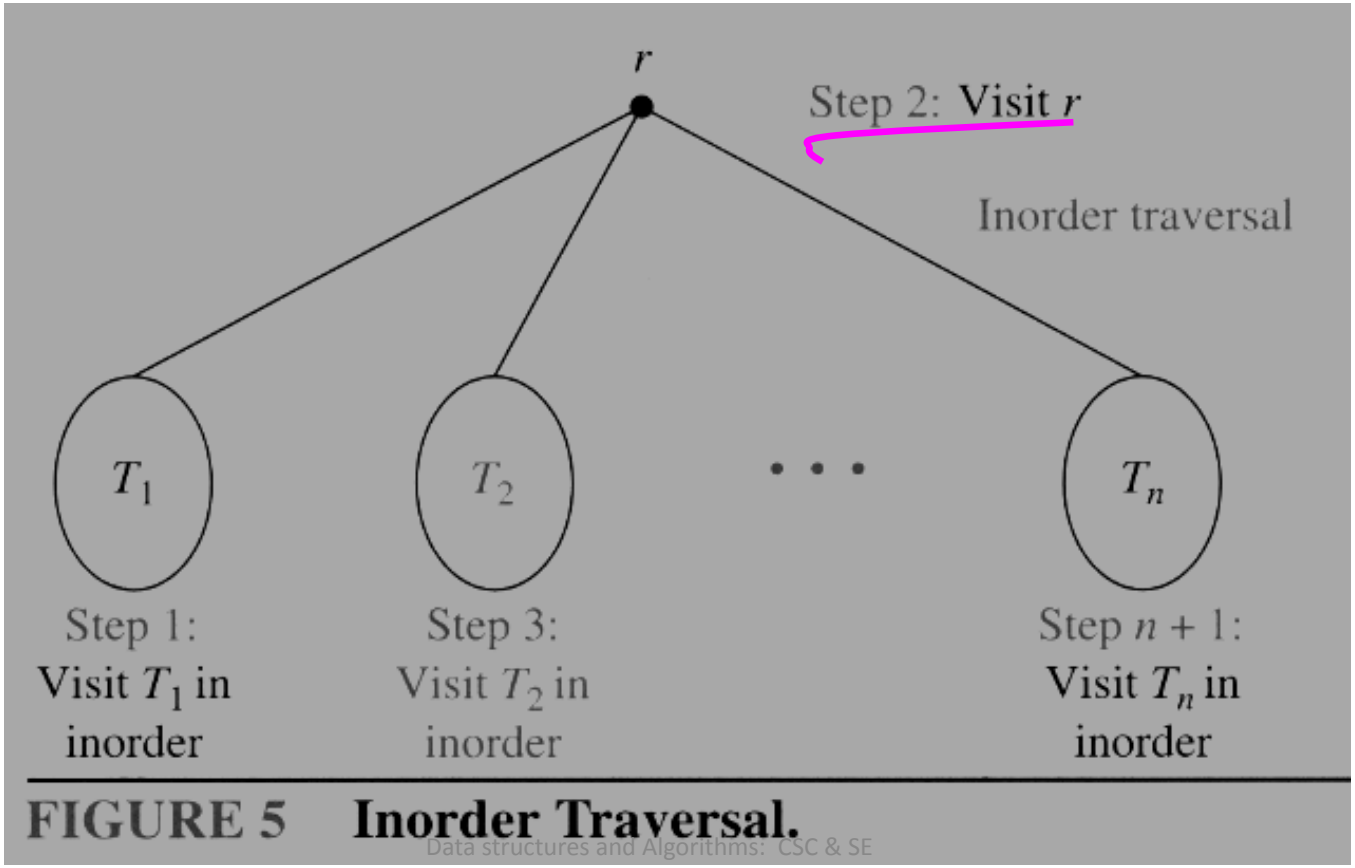


FIGURE 5 Inorder Traversal.

Postorder Traversal

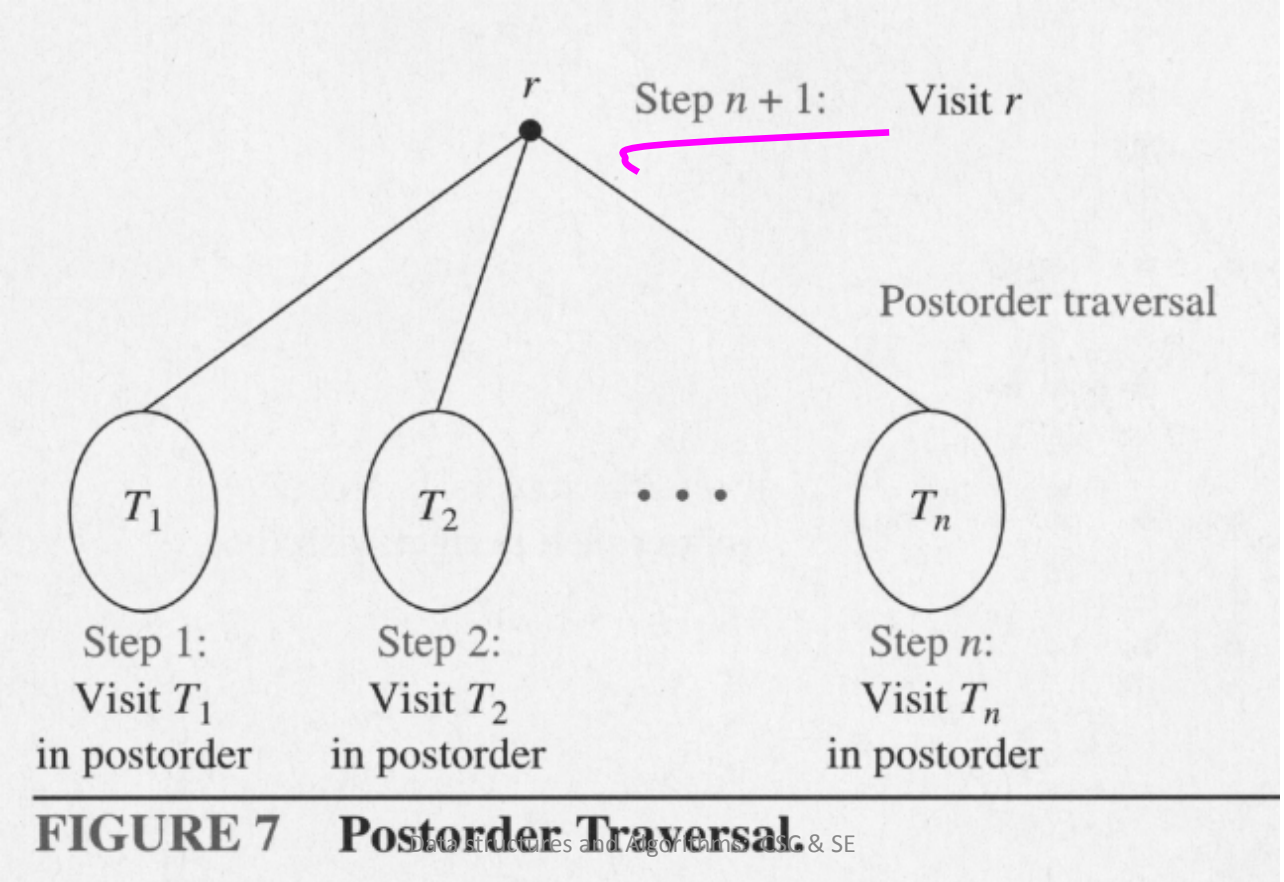
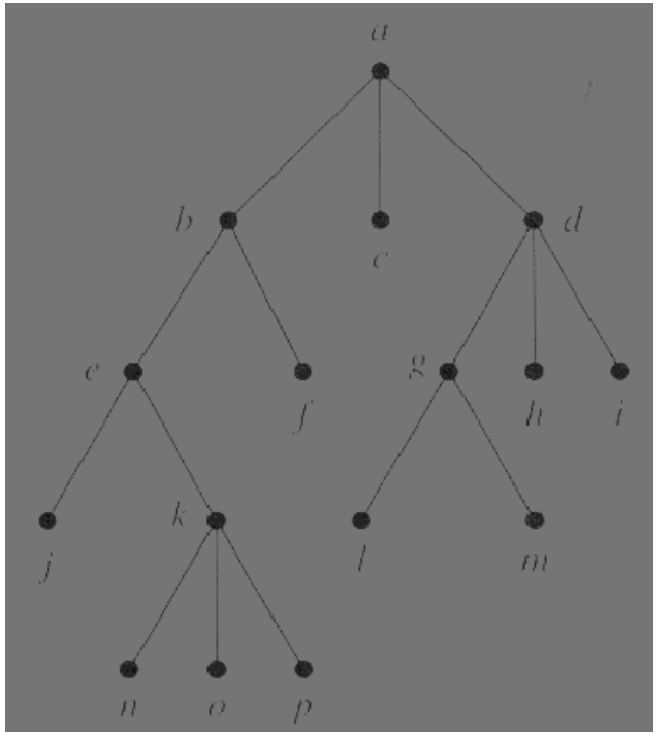


FIGURE 7 Postorder Traversal.

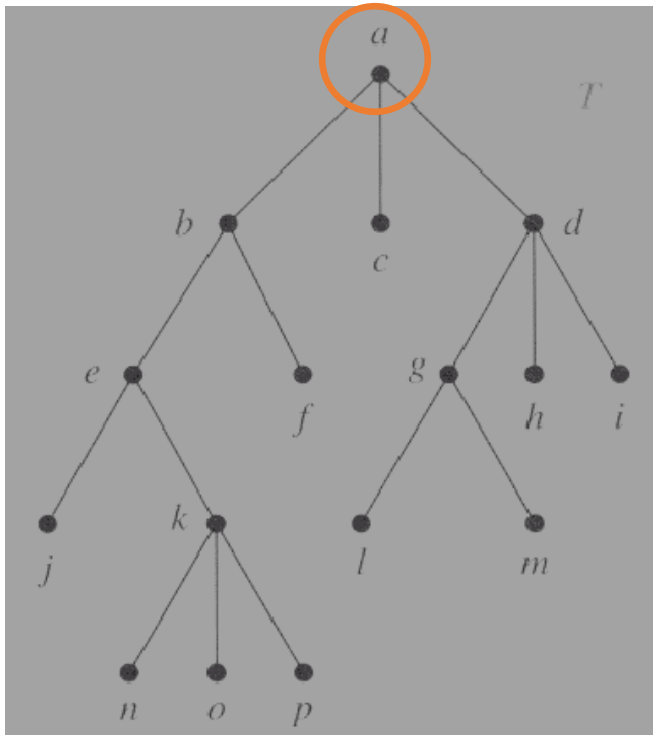
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output:

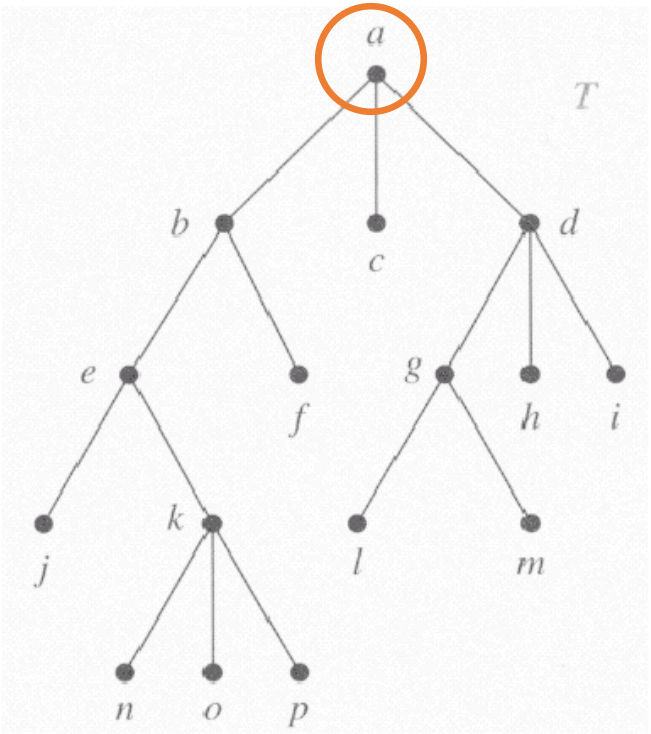
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output: a

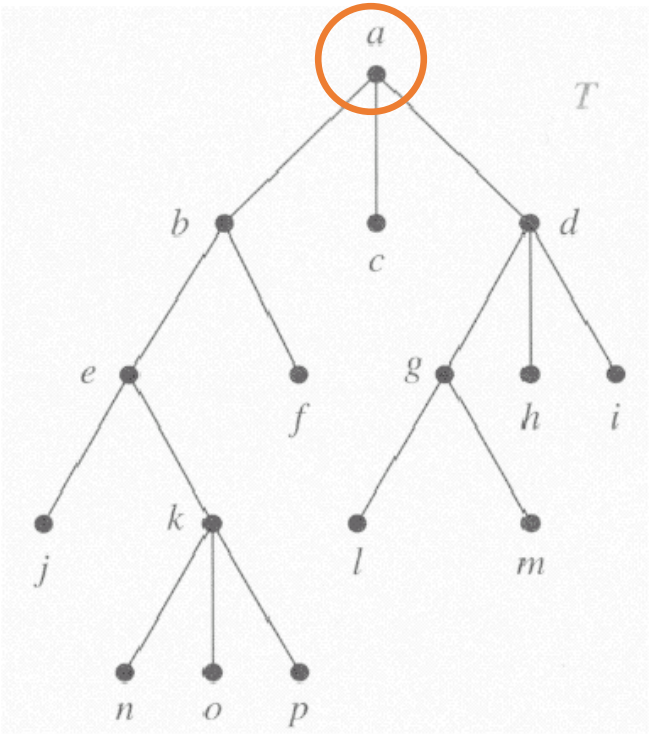
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {b, c, d}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a

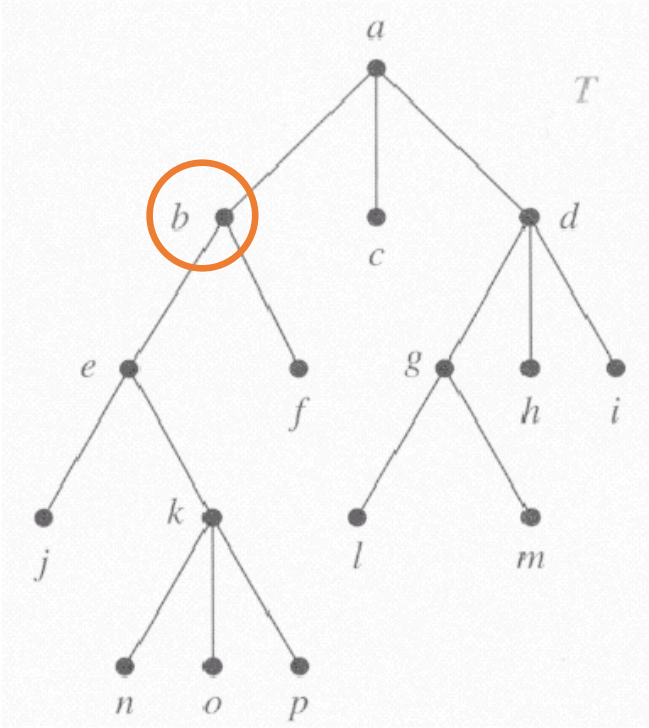
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {b, c, d}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a

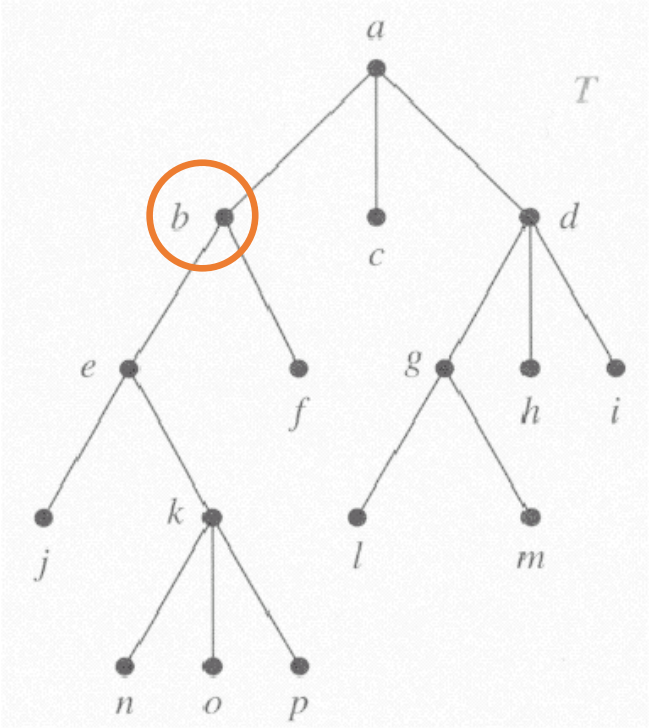
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output: a b

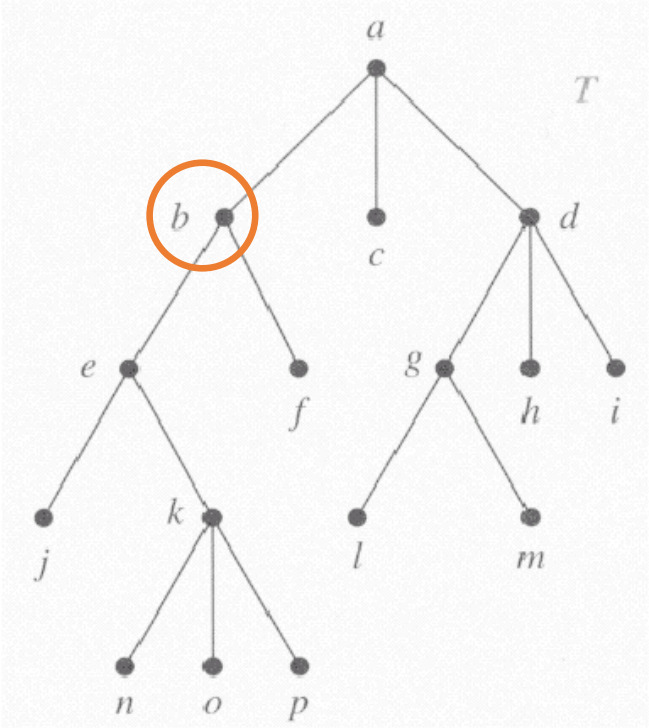
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for                    for each of {e, f}
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

output: a b

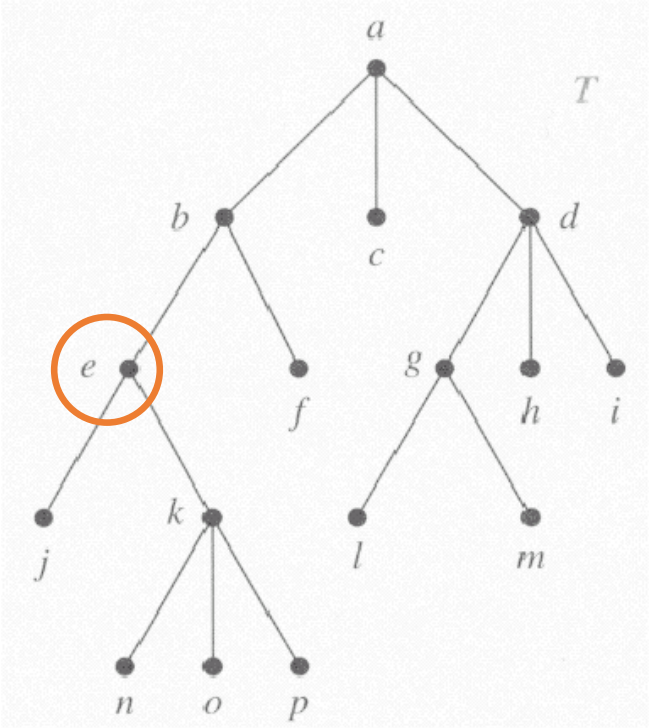
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {e, f}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b

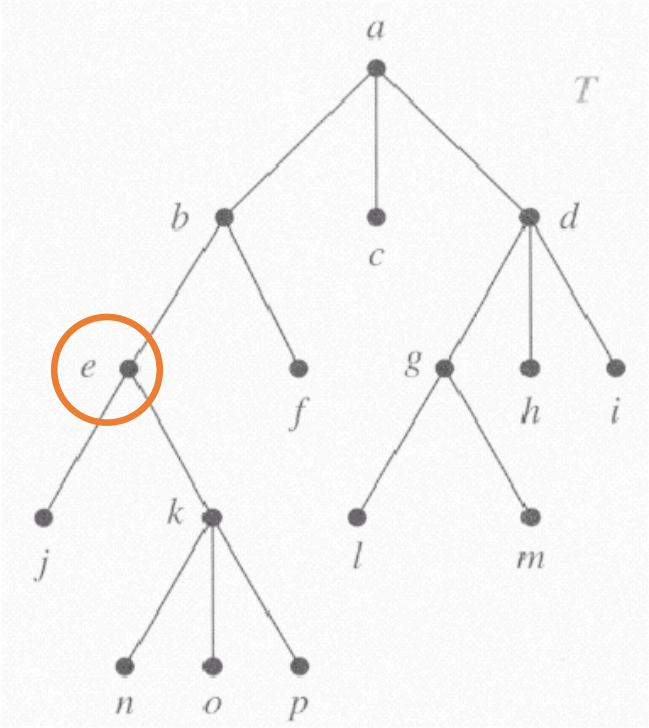
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output: a b e

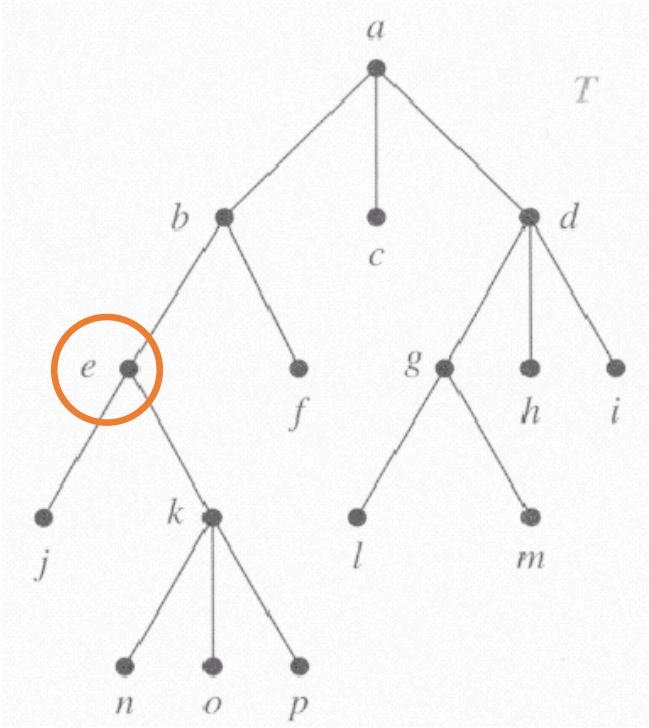
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for                    for each of {j, k}
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

output: a b e

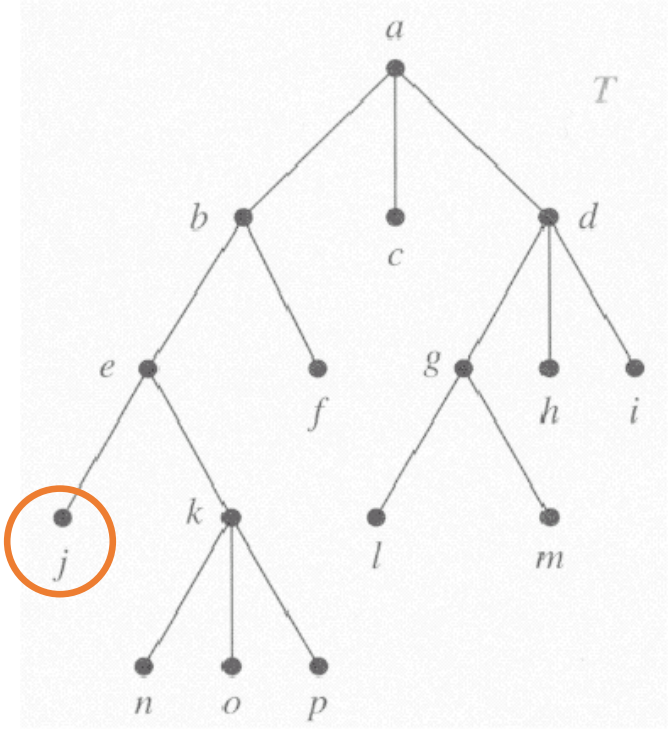
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {j, k}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b e

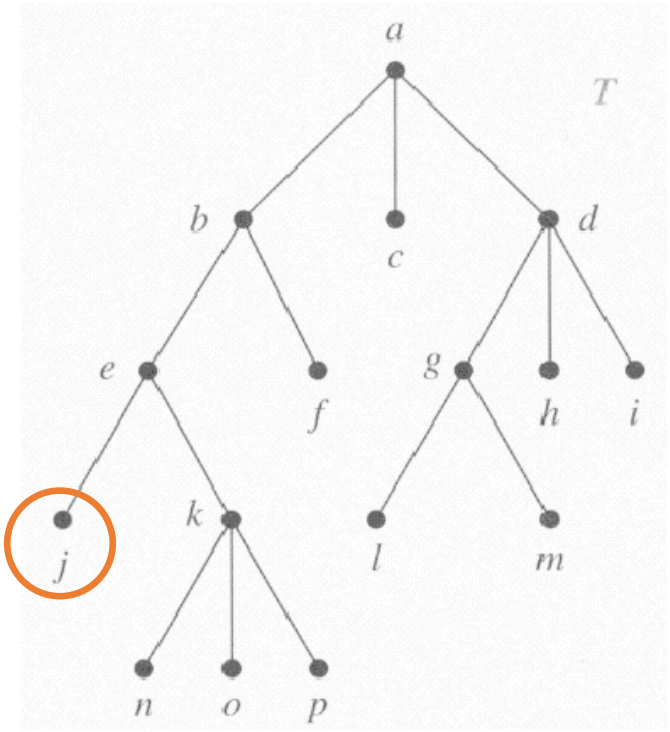
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output: a b e j

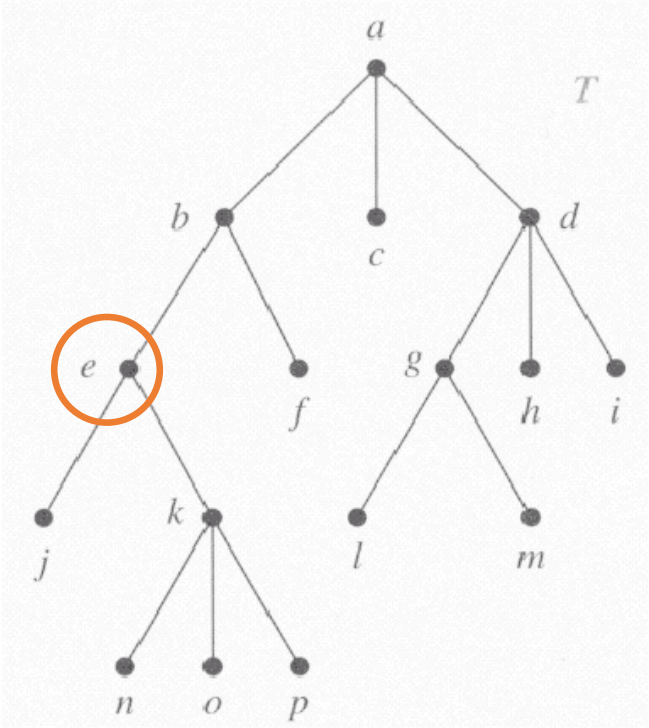
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for                    for each of {}
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

output: a b e j

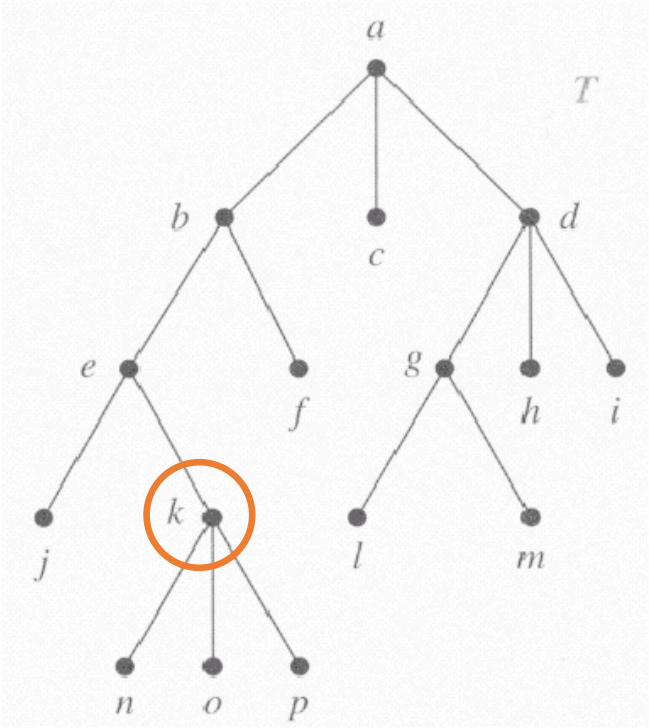
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {j, k}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b e j

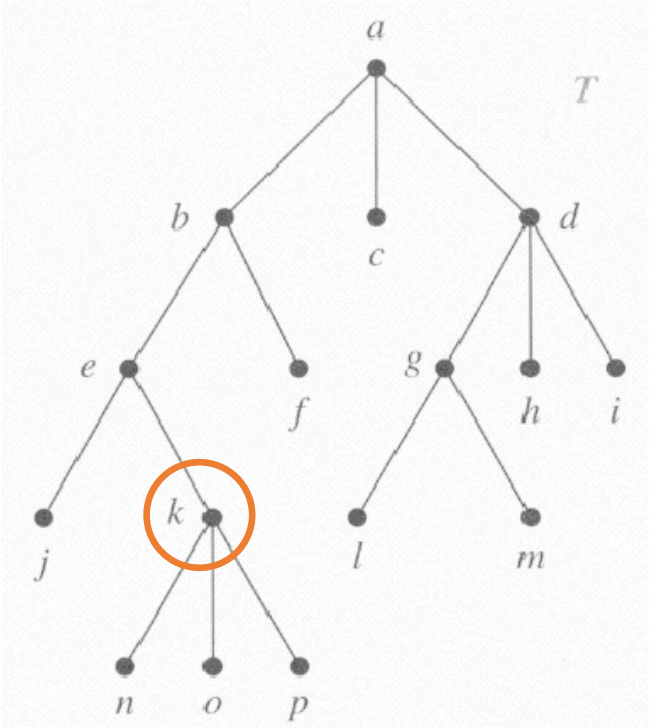
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output: a b e j k

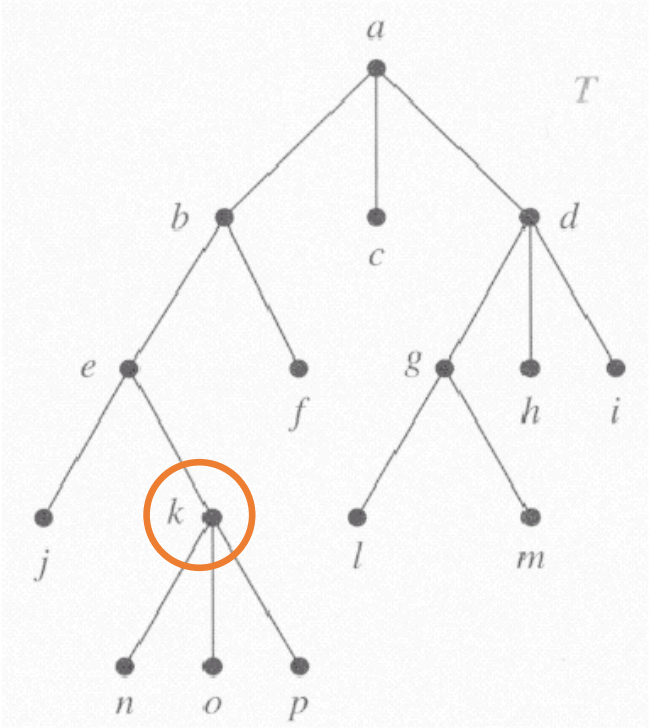
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {n, o, p}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b e j k

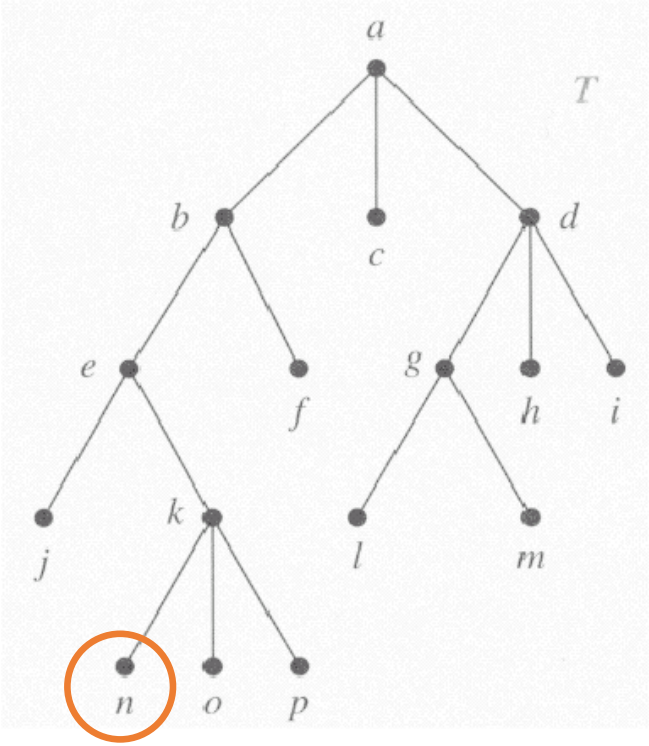
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {n, o, p}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b e j k

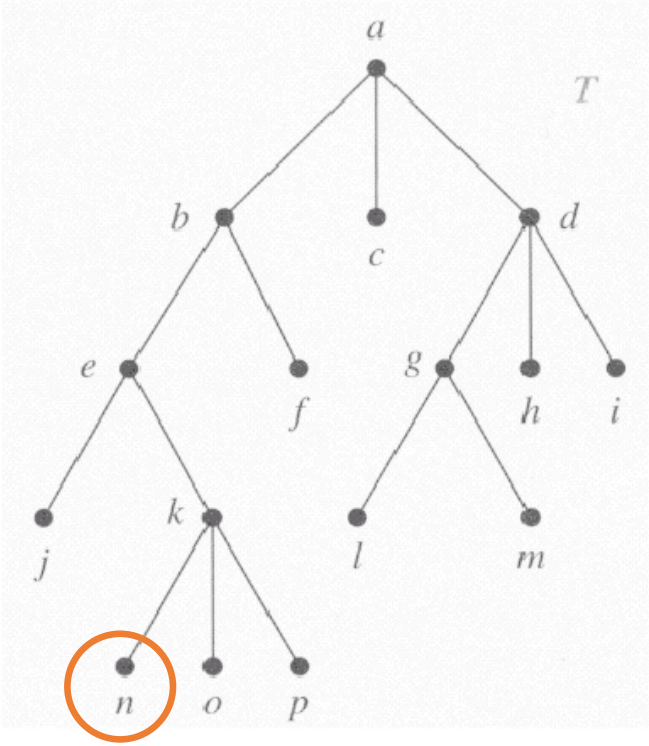
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output: a b e j k n

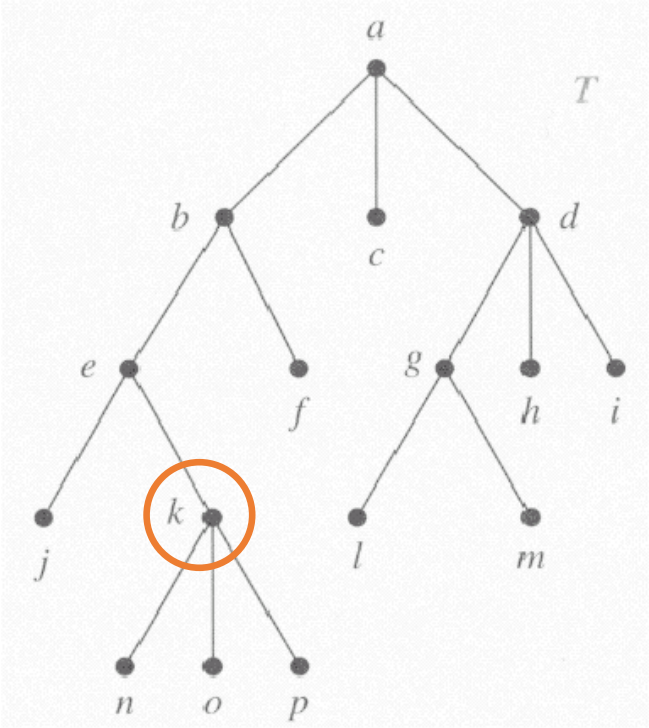
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for                    for each of {}
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

output: a b e j k n

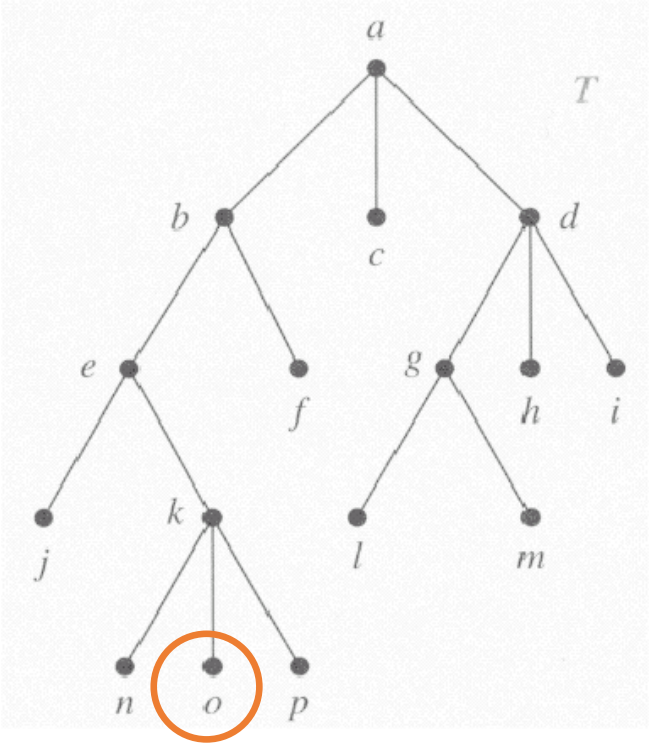
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {n, o, p}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b e j k n

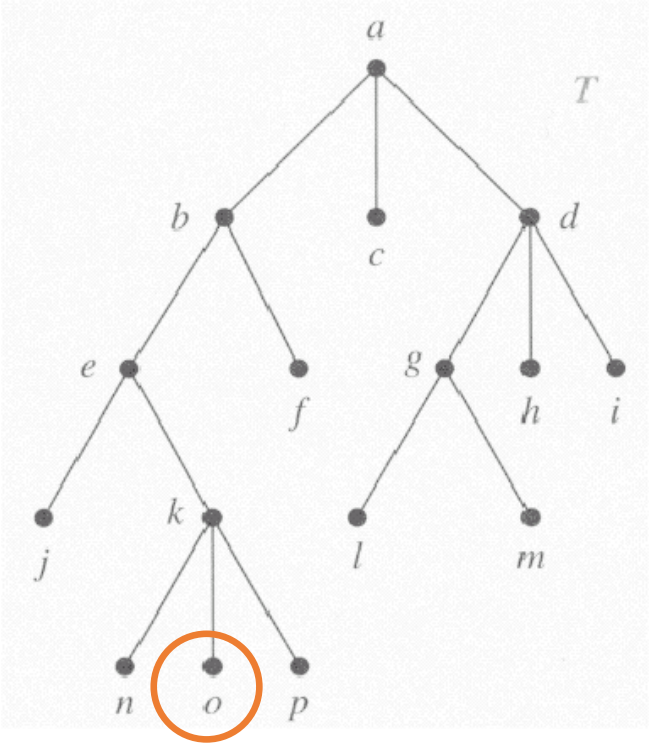
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



output: a b e j k n o

```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

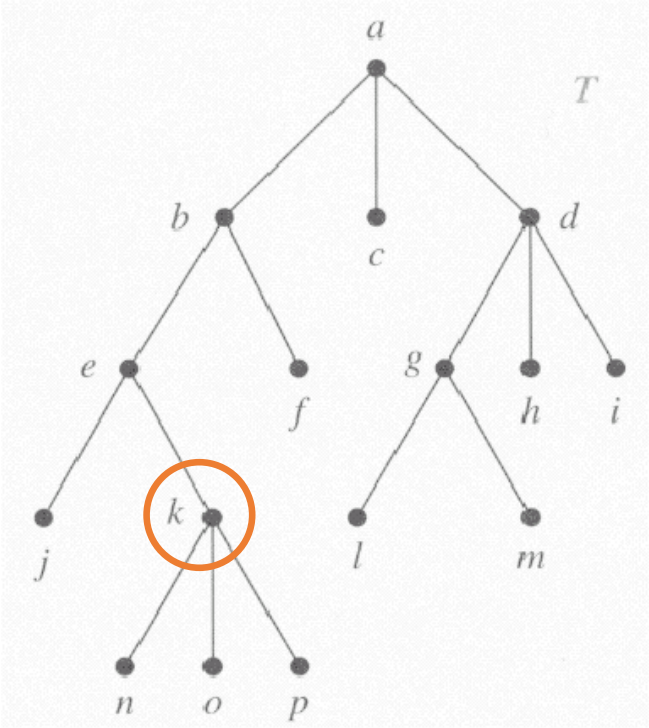
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for                    for each of {}
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

output: a b e j k n o

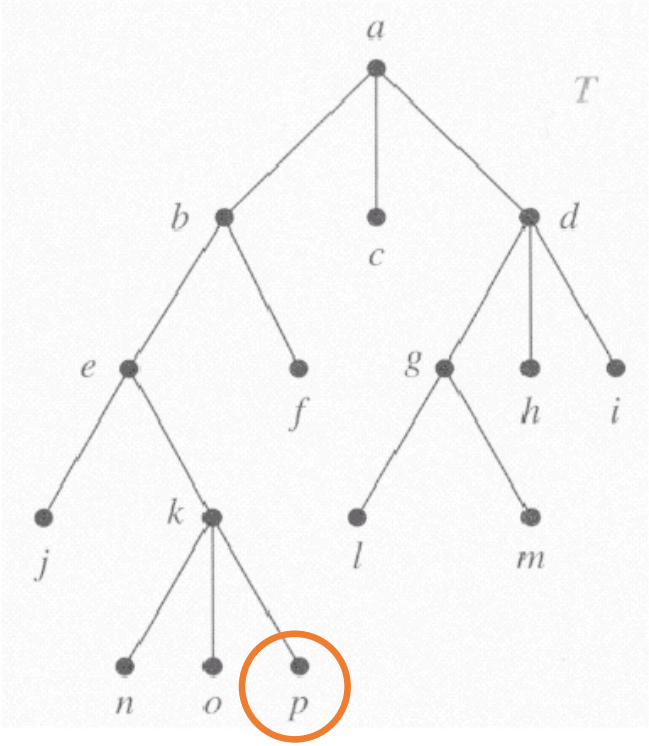
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {n, o, p}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b e j k n o

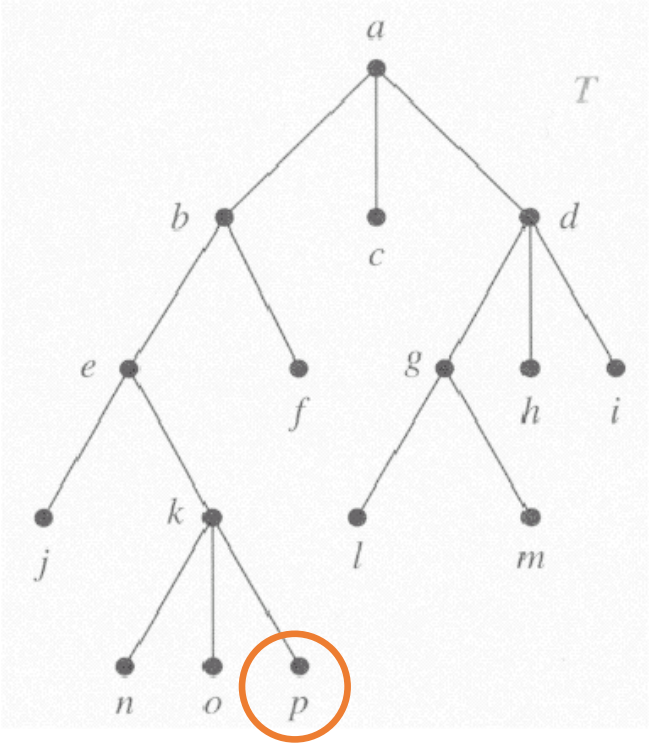
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output: a b e j k n o p

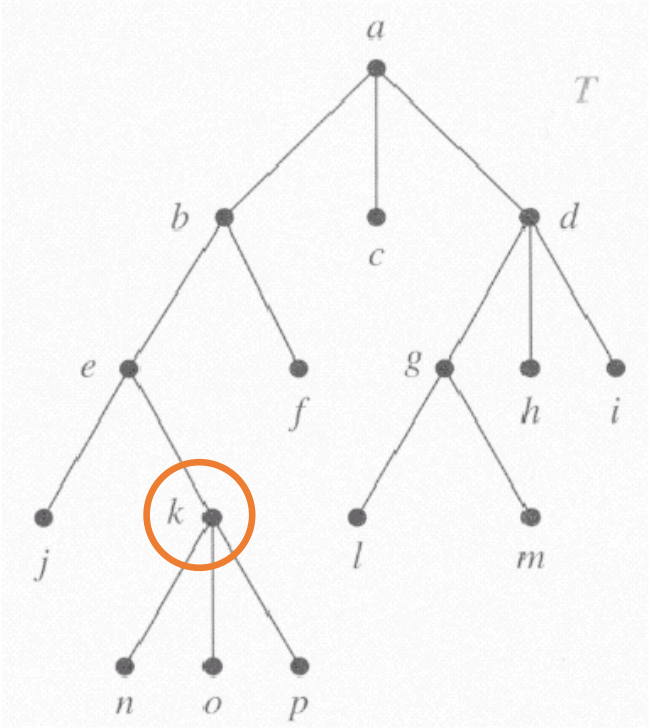
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for                    for each of {}
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

output: a b e j k n o p

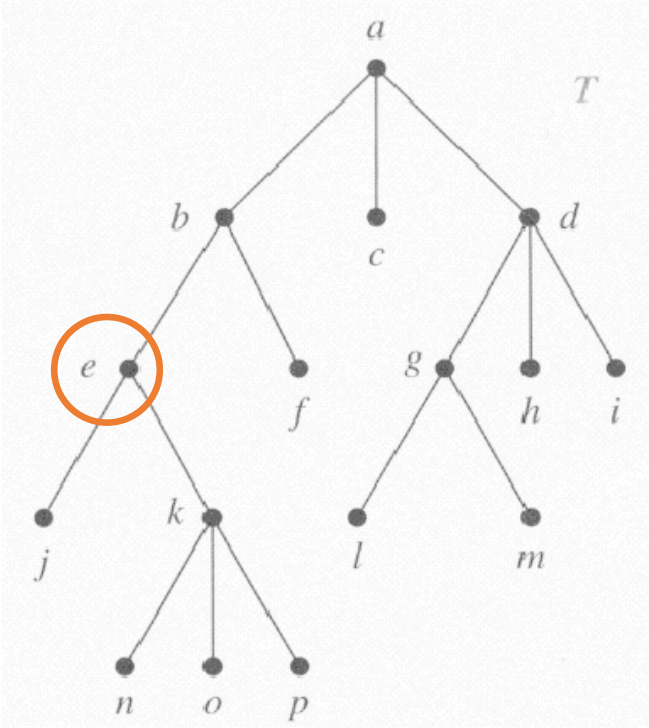
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {n, o, p}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b e j k n o p

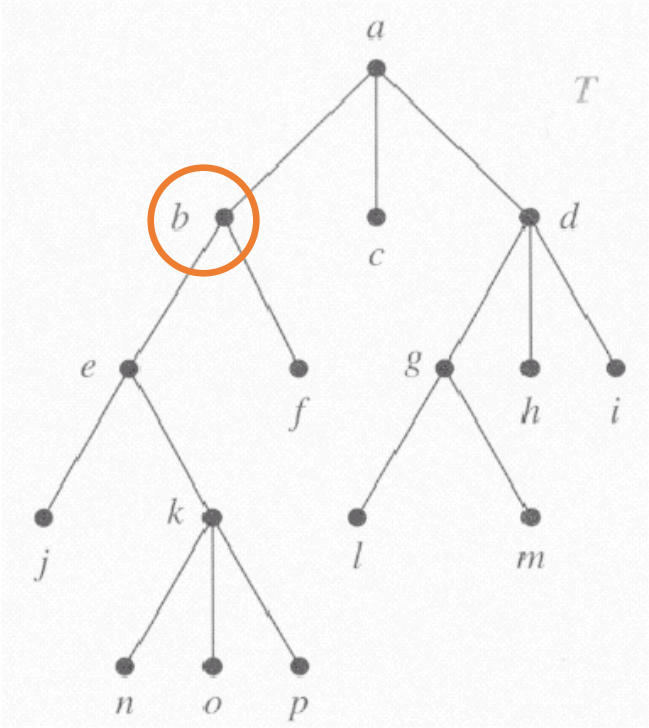
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for                    for each of {j, k}
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

output: a b e j k n o p

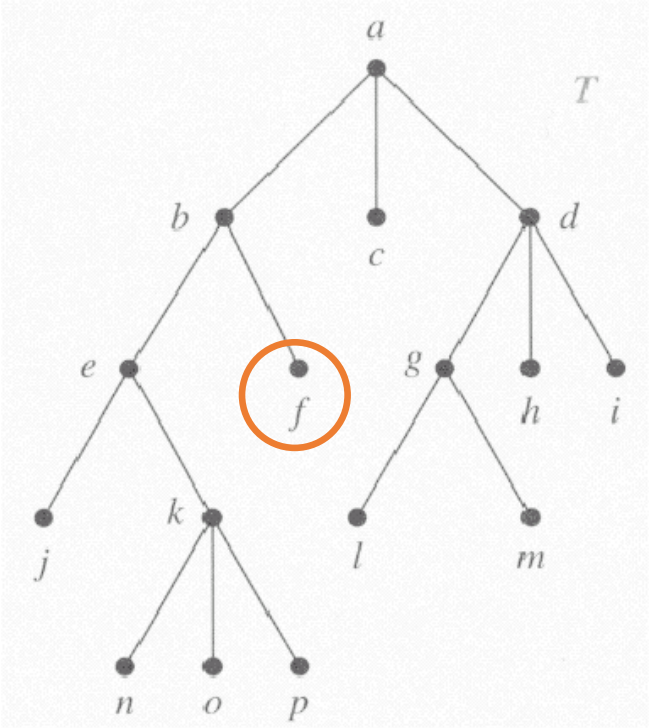
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for                    for each of {e, f}
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

output: a b e j k n o p

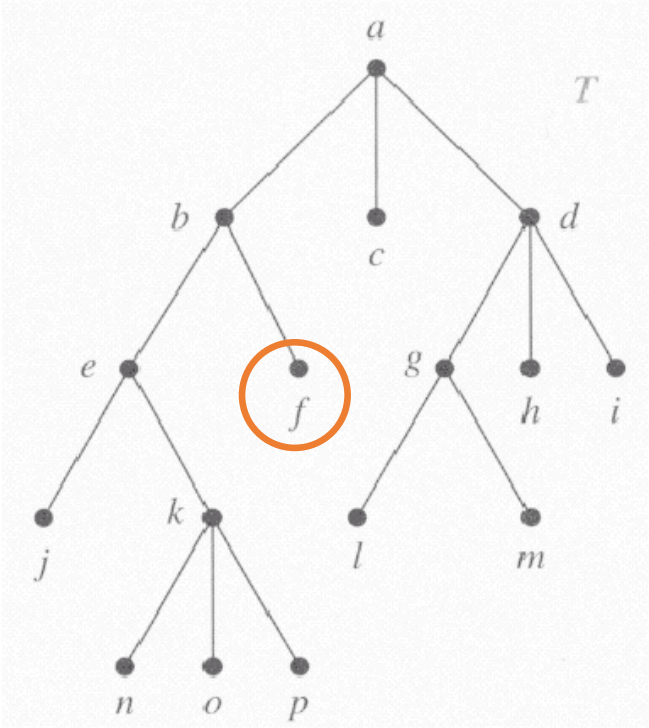
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output: a b e j k n o p f

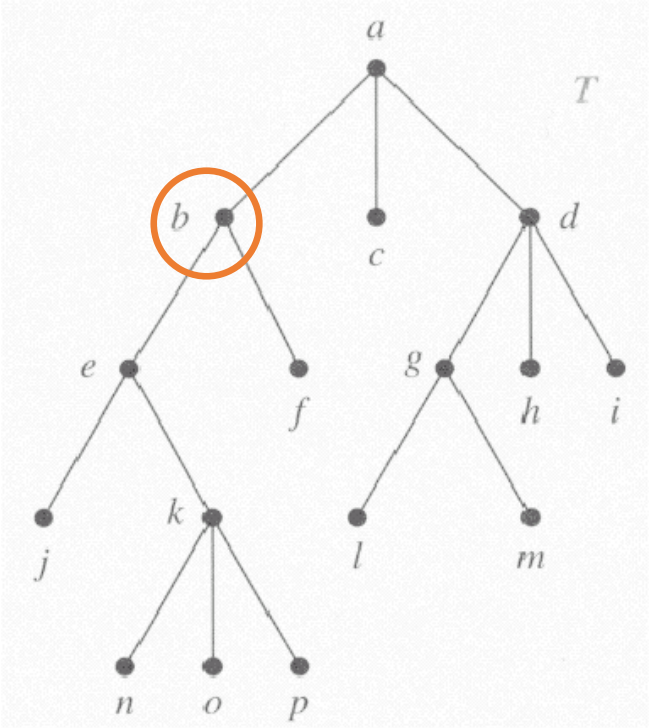
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b e j k n o p f

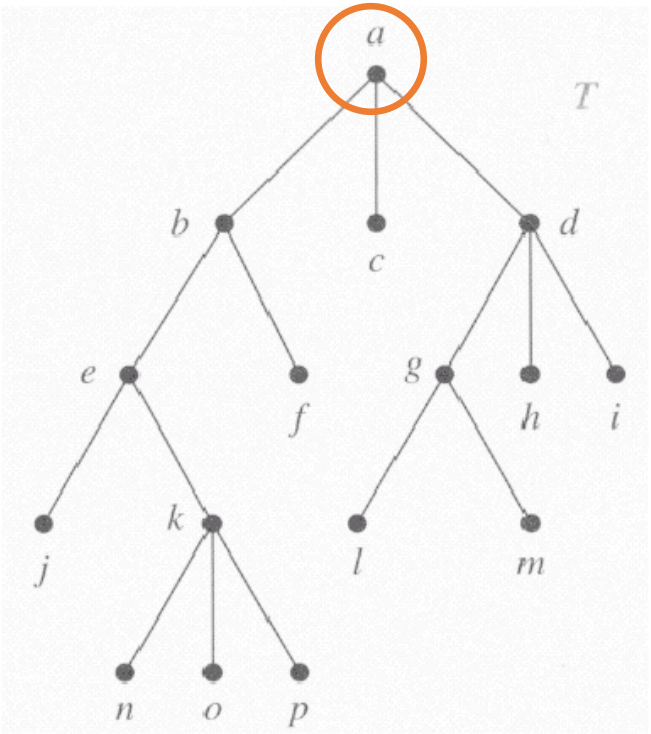
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for                    for each of {e, f}
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

output: a b e j k n o p f

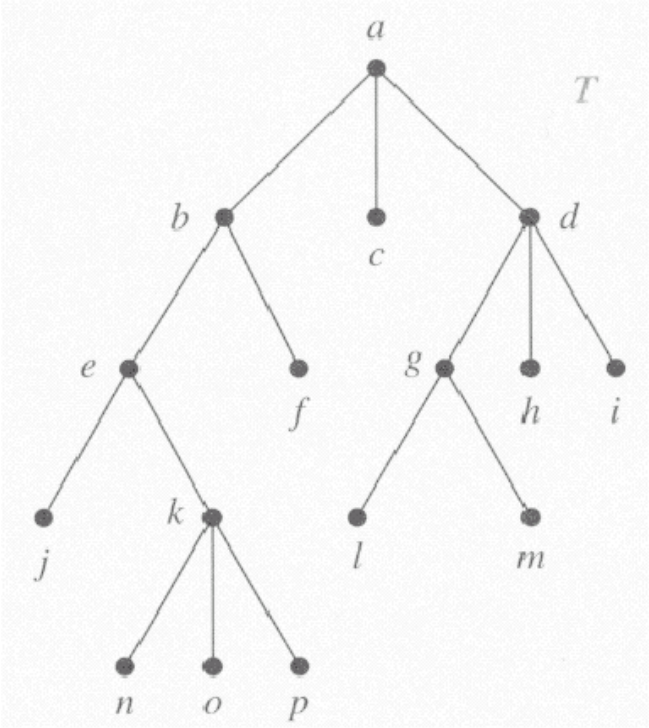
In which order does a preorder traversal visit the vertices in this ordered rooted tree?



procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for for each of {b, c, d}
begin
  T(*c*) := subtree with *c* as its root
  preorder(*T*(*c*))
end

output: a b e j k n o p f

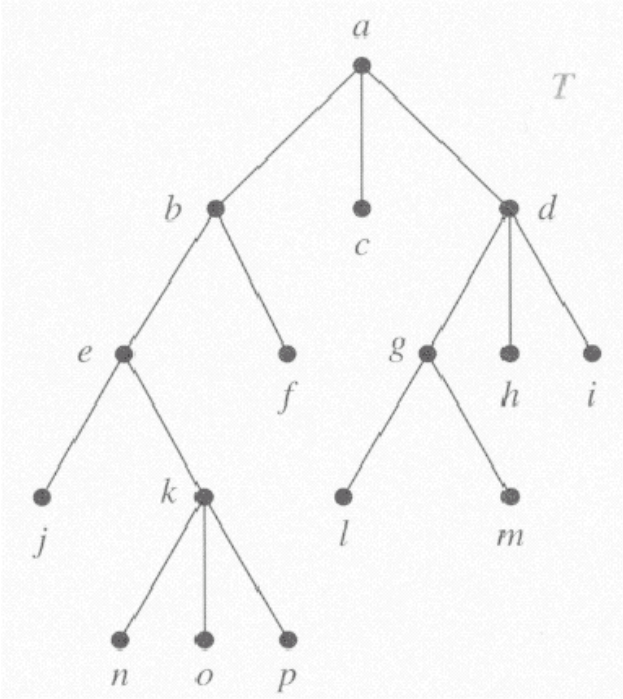
In which order does a preorder traversal visit the vertices in this ordered rooted tree?




```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  T(c) := subtree with c as its root
  preorder(T(c))
end
```

output: a b e j k n o p f c d g l m h i

In which order does a inorder traversal visit the vertices in this ordered rooted tree?



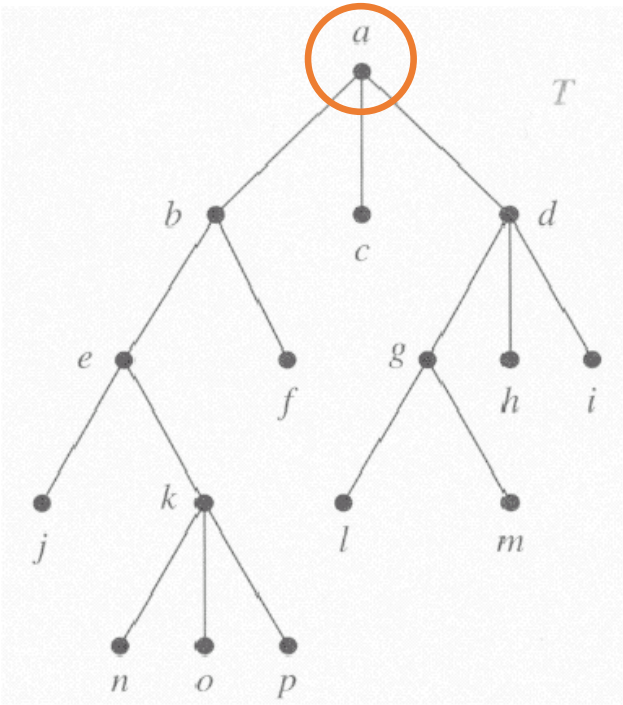
procedure *inorder*(*T*: ordered rooted tree)
r := root of *T*
if r is a leaf **then** list *r*
else
begin
  l:= first child of *r* from left to right
  T(l) := subtree with *l* as its root
  inorder(T(l))
  list *r*
  for each child *c* of *r* except for *l* left to right
    T(*c*) := subtree with *c* as its root
    preorder(*T*(*c*))
end

output:

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

call stack

r = *a*, *l* = ?
s = {}

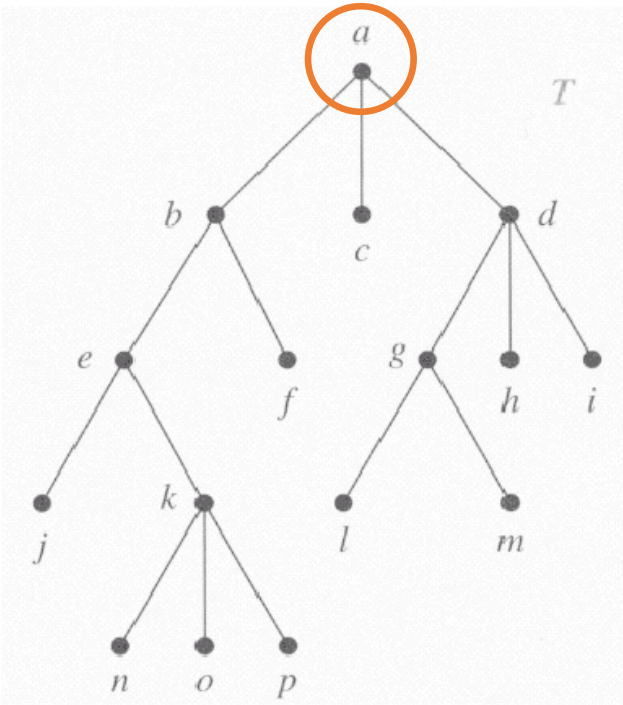


procedure *inorder*(*T*: ordered rooted tree)
r := root of *T*
if r is a leaf **then** list *r*
else
begin
  l:= first child of *r* from left to right
  T(l) := subtree with *l* as its root
  inorder(T(l))
  list *r*
  for each child *c* of *r* except for *l* left to right
    T(*c*) := subtree with *c* as its root
    preorder(*T*(*c*))
end

output:

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

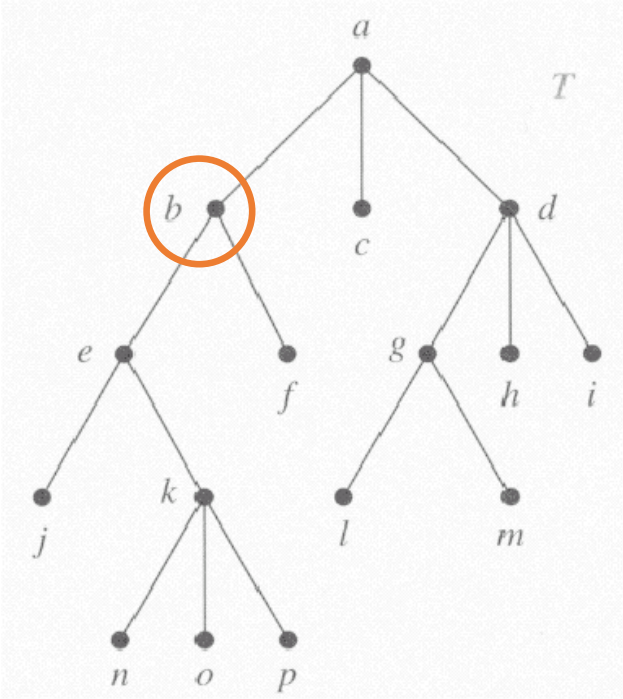
call stack
r = *a*, *l* = *b* s = {}



procedure *inorder*(*T*: ordered rooted tree)
r := root of *T*
if r is a leaf **then** list *r*
else
begin
l:= first child of *r* from left to right
T(l) := subtree with *l* as its root
inorder(T(l))
list *r*
for each child *c* of *r* except for *l* left to right
T(*c*) := subtree with *c* as its root
preorder(*T*(*c*))
end

output:

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

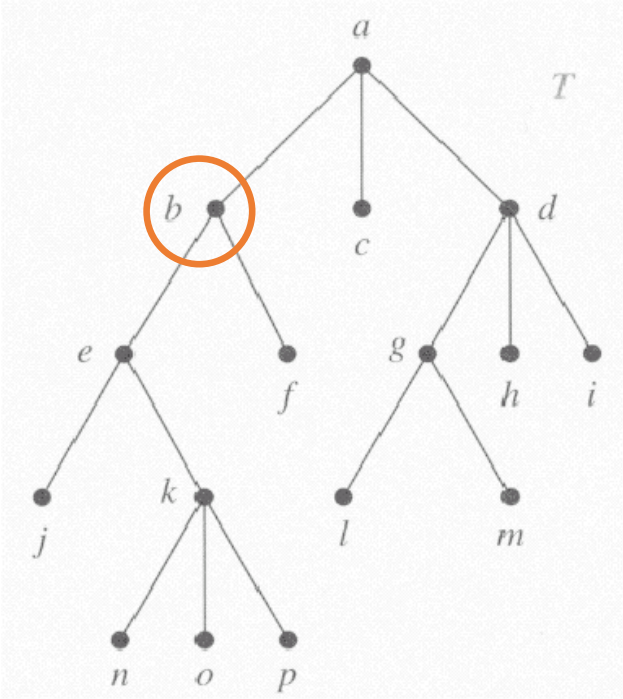



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left to right
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = ? s = {}

output:

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

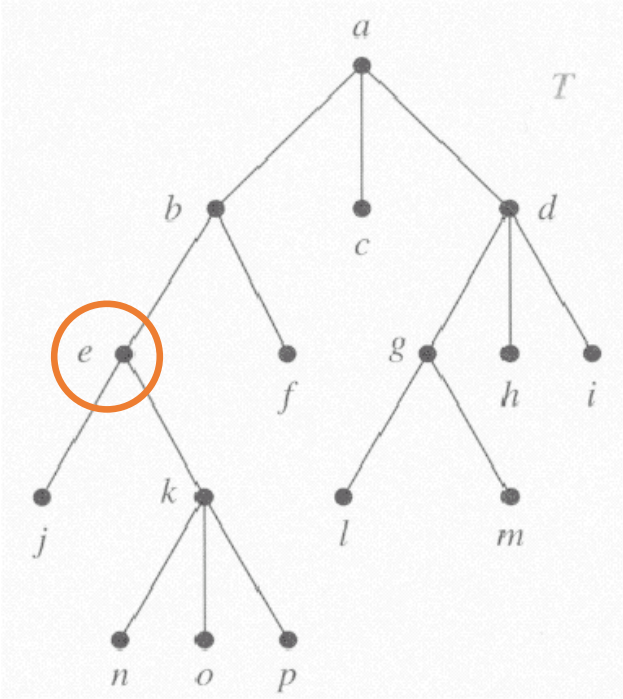



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left to right
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}

output:

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

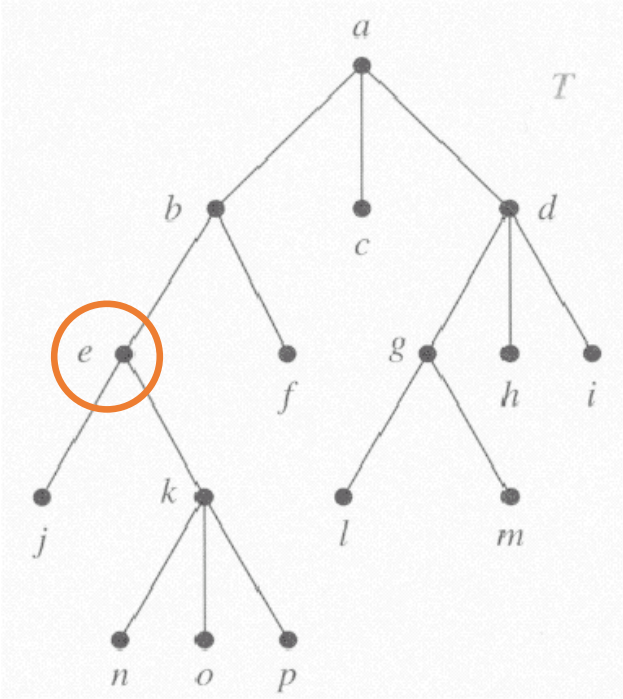



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left to right
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = ? s = {}

output:

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

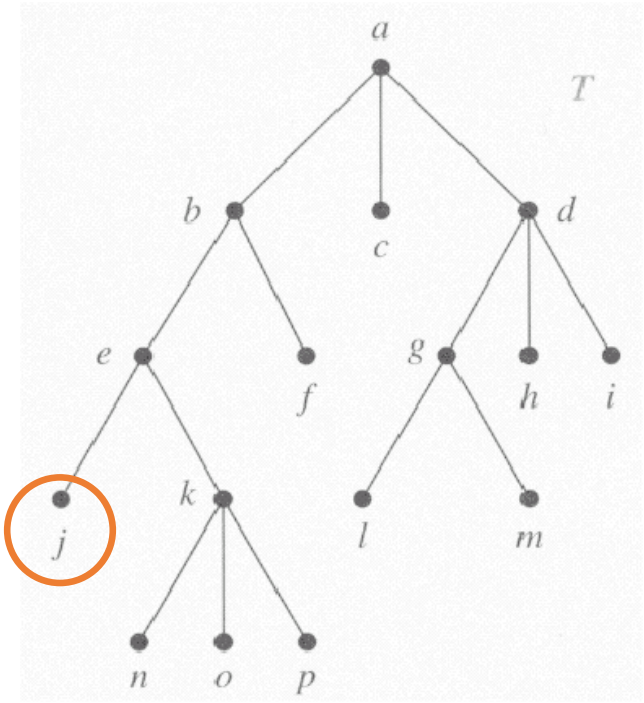



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left to right
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {}

output: j

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

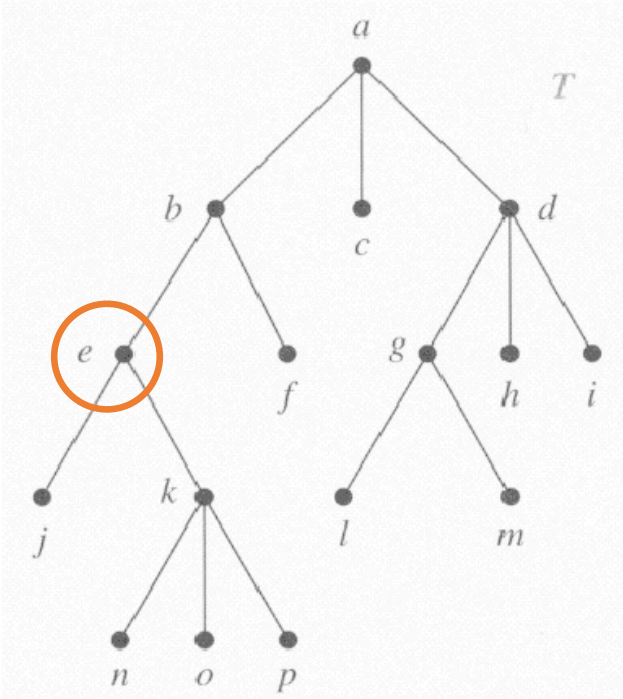



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left to right
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {}
r = j, l = ? s = {}

output: j

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

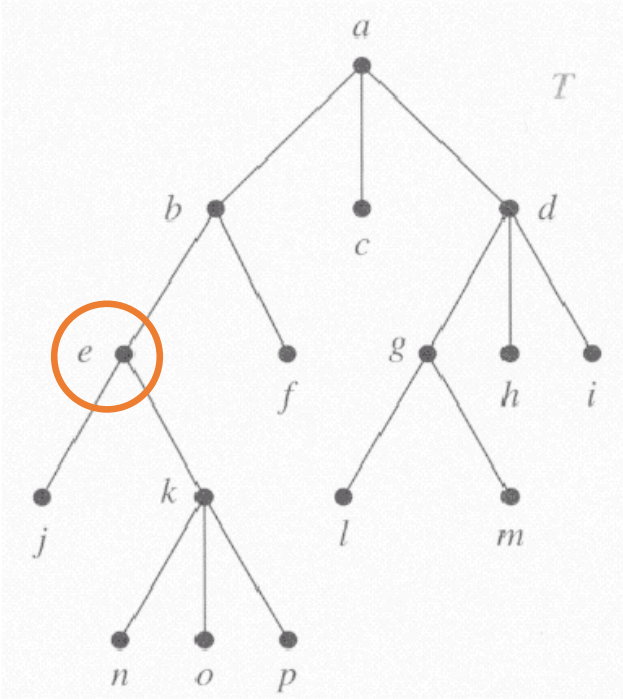



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left to right
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r =e, l = j s = {}

output: j e

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

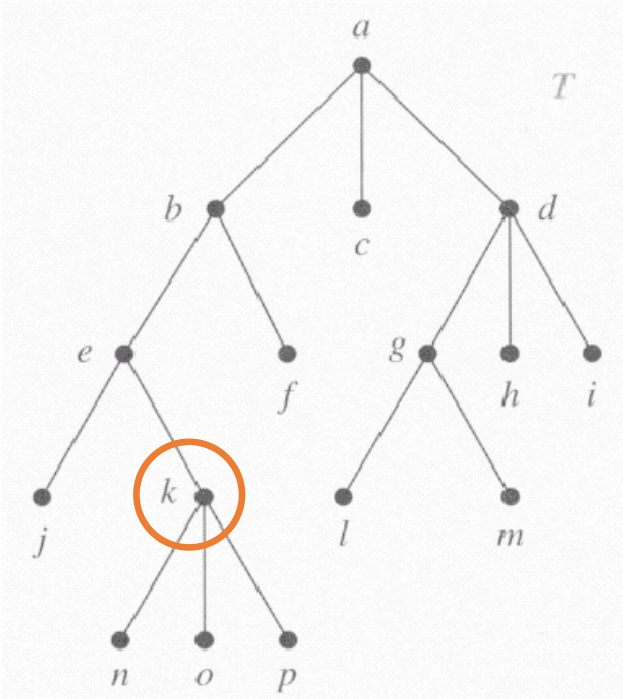



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left to right
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}

output: j e

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

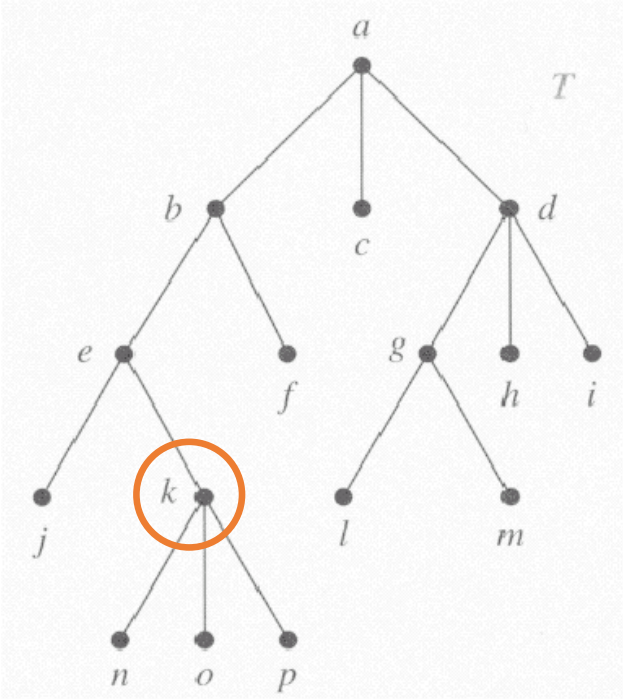



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left to right
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}
r = k, l = ? *s = {}*

output: j e

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

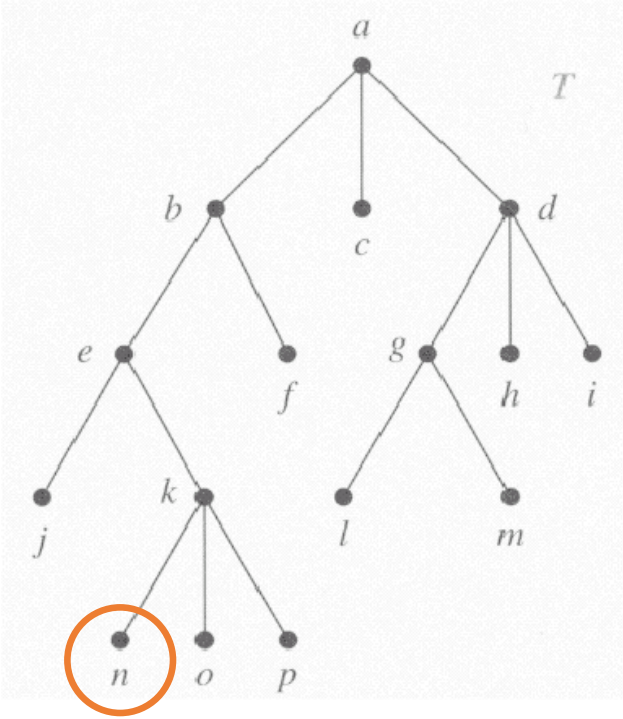



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left to right
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}
r = k, l = n s = {}

output: j e

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

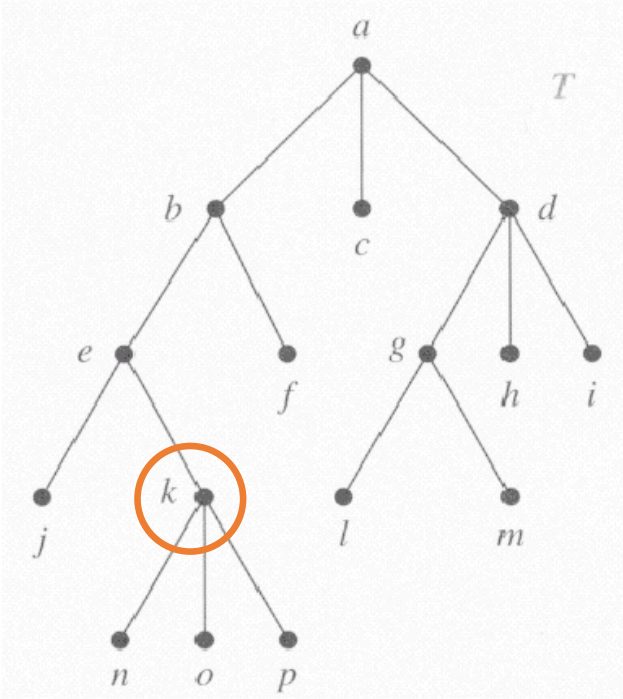



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}
r = k, l = n *s = {}*
r = n, l = ? *s = {}*

output: j e n

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

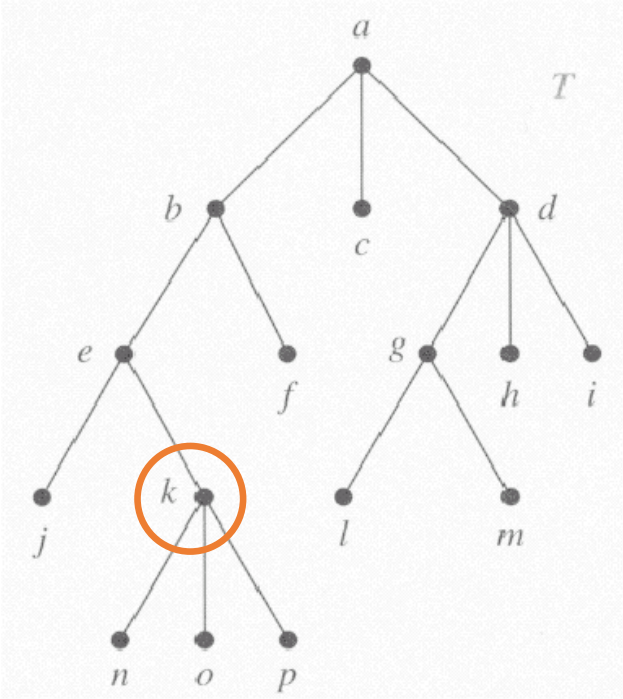



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}
r = k, l = n s = {}

output: j e n k

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

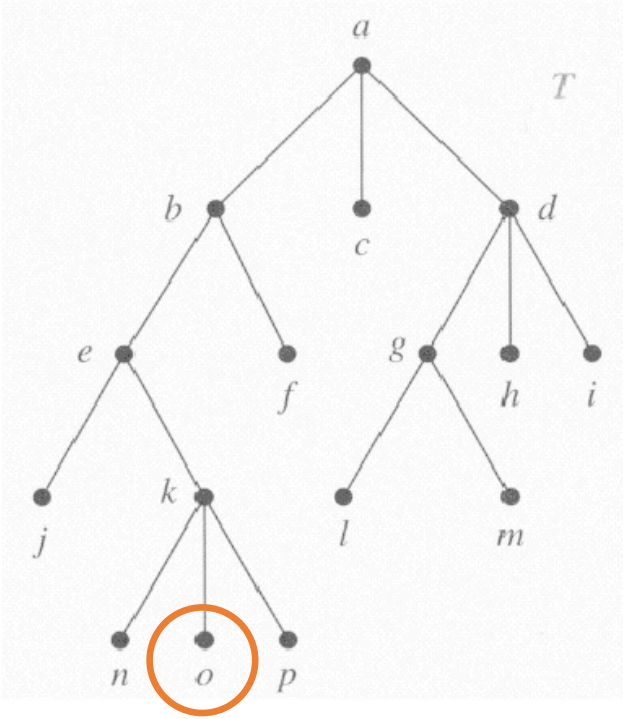



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}
r = k, l = n s = {o,p}

output: j e n k

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

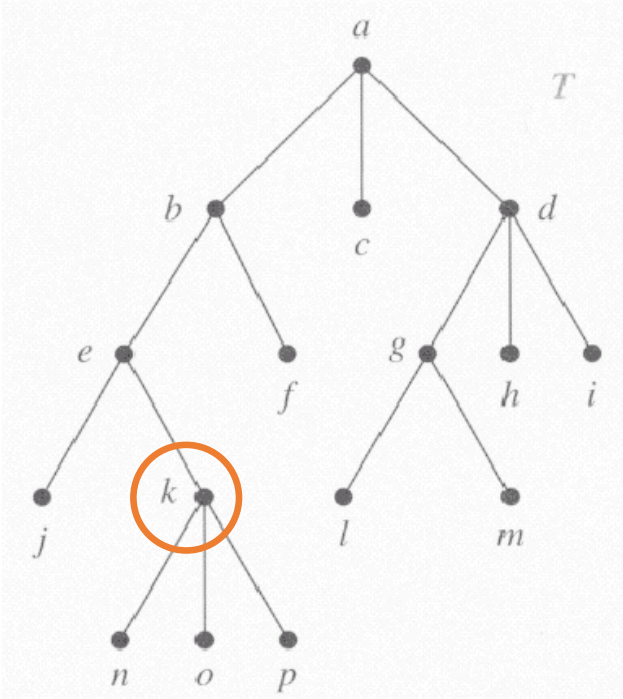



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}
r = k, l = n *s = {o,p}*
r = o, l = ? *s = {}*

output: j e n k o

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

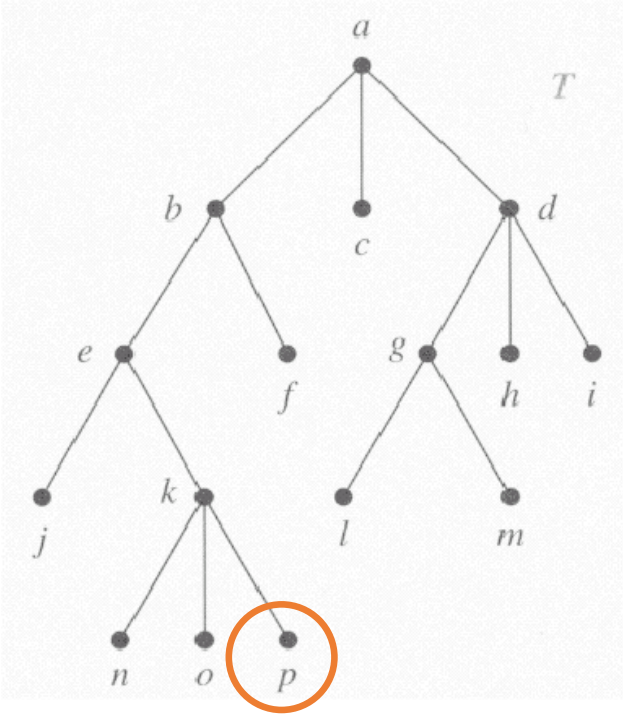



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}
r = k, l = n s = {o,p}

output: j e n k o

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

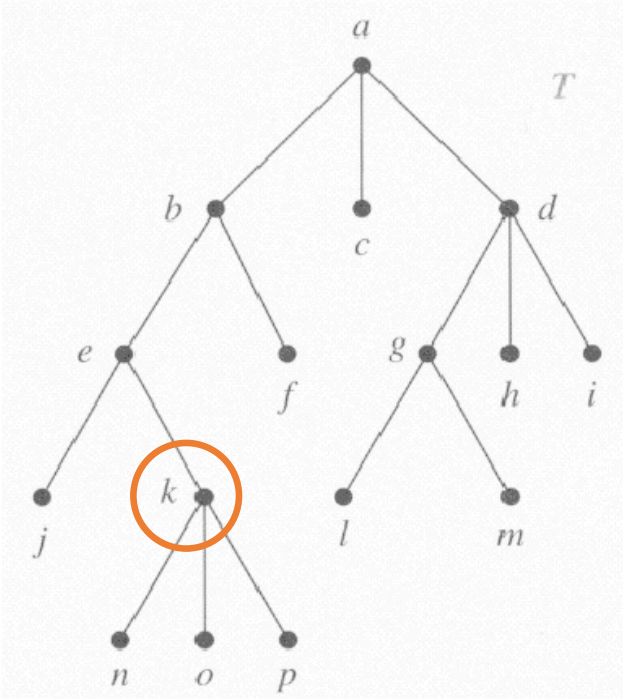



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}
r = k, l = n *s = {o,p}*
r = k, l = ? *s = {}*

output: j e n k o p

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

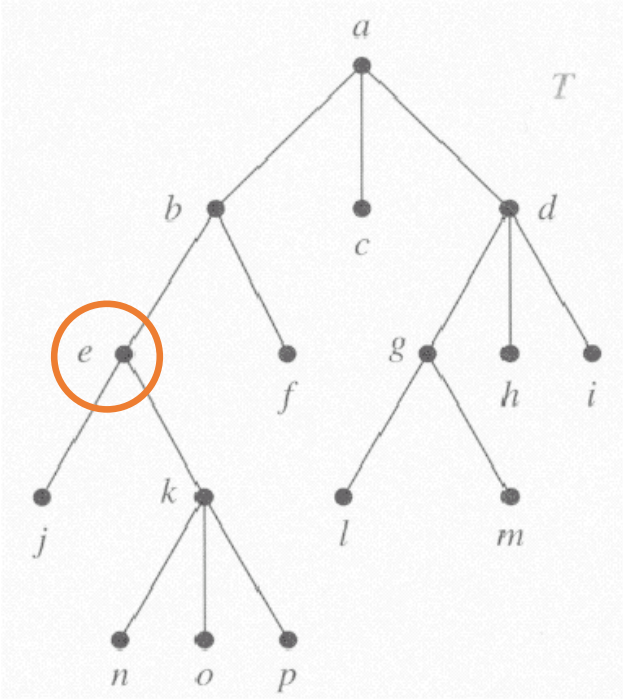



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}
r = k, l = n s = {o,p}

output: j e n k o p

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

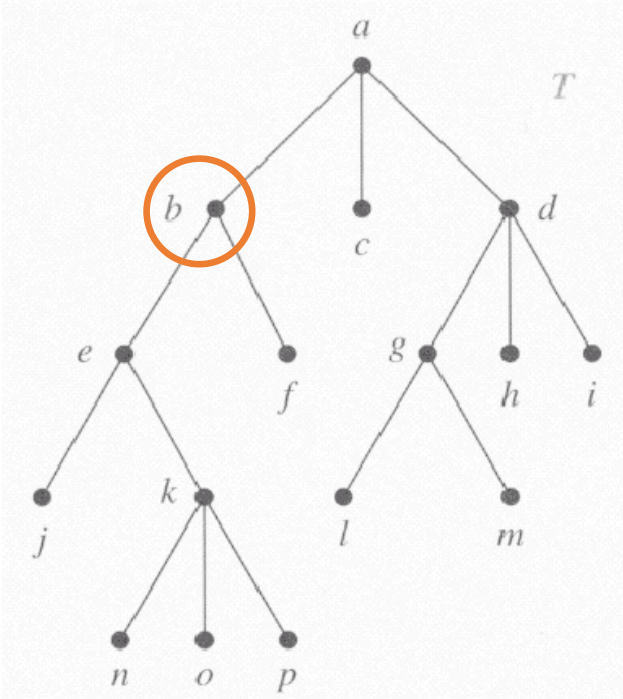



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}
r = e, l = j s = {k}

output: j e n k o p

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

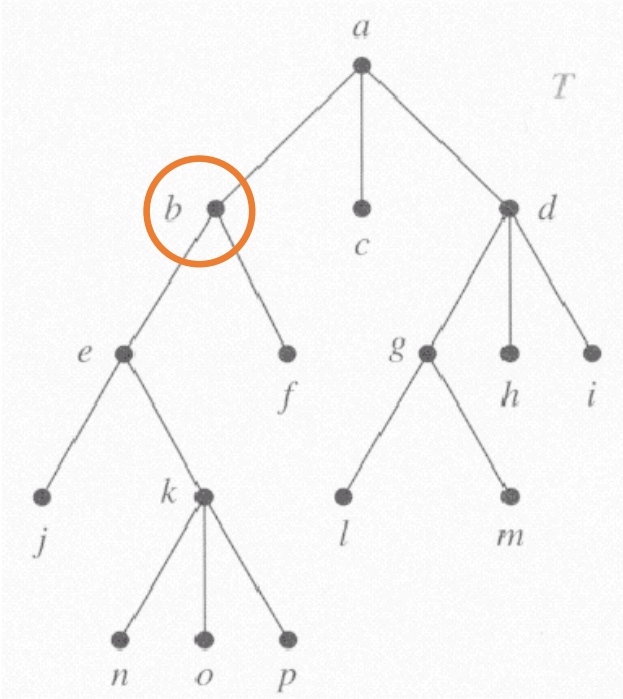



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {}

output: j e n k o p b

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

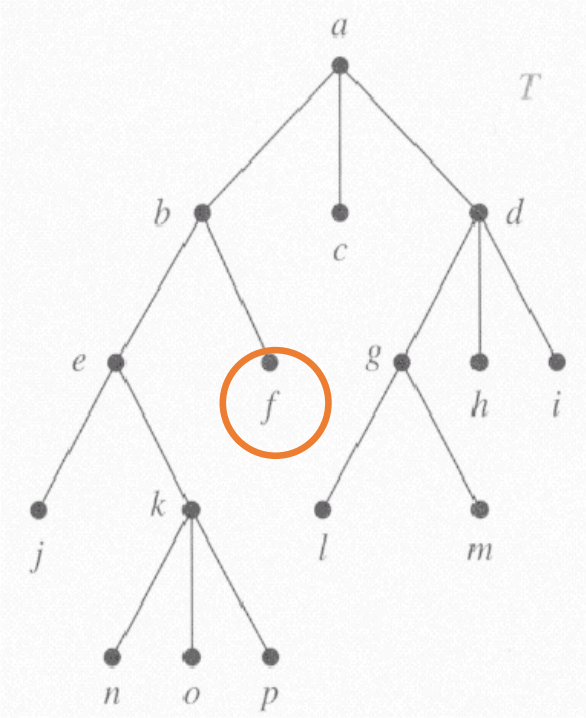



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {f}

output: j e n k o p b

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

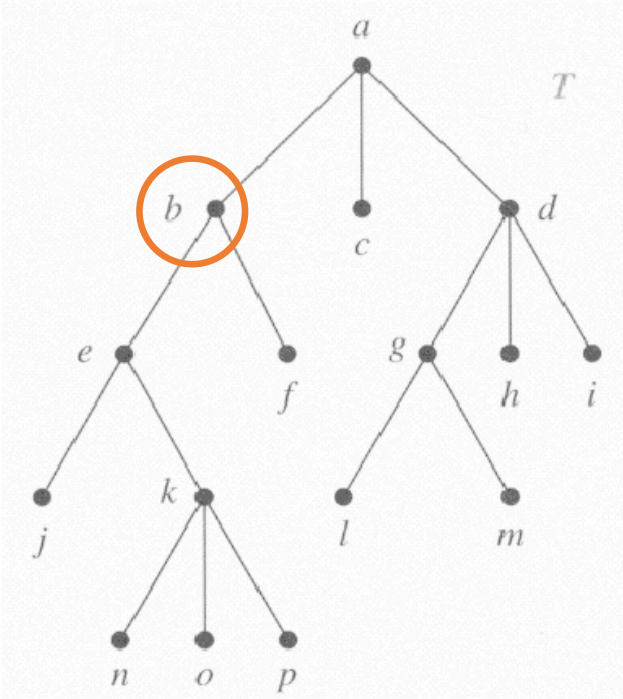



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {}
r = b, l = e s = {f}
r = f, l = e s = {}

output: j e n k o p b f

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

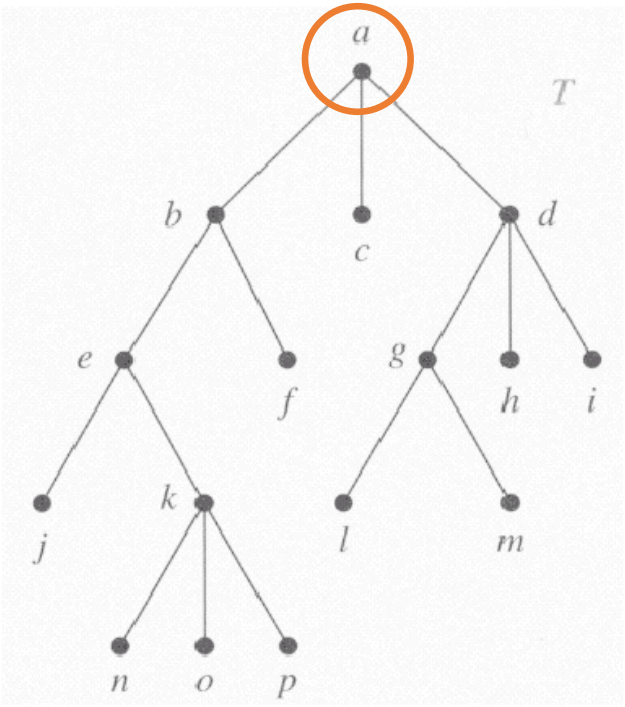



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

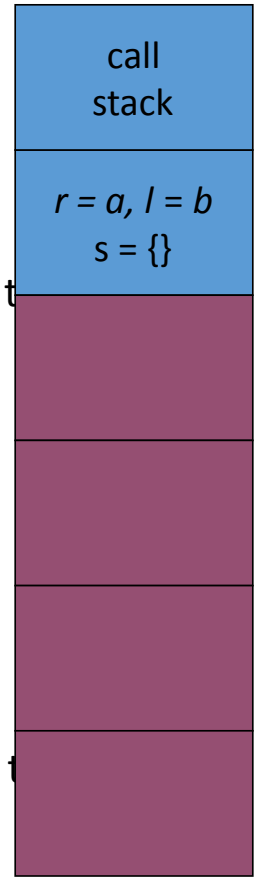
call stack
r = a, l = b s = {}
r = b, l = e s = {f}

output: j e n k o p b f

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

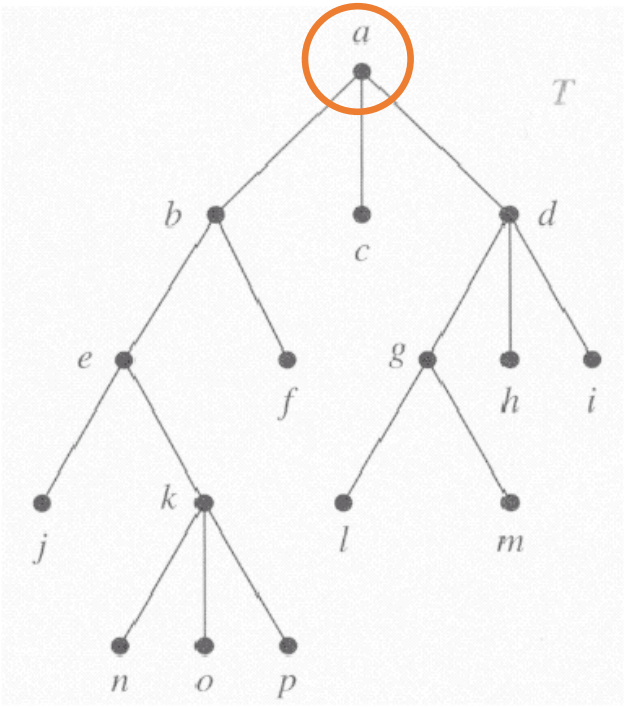



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```



output: j e n k o p b f

In which order does a inorder traversal visit the vertices in this ordered rooted tree?

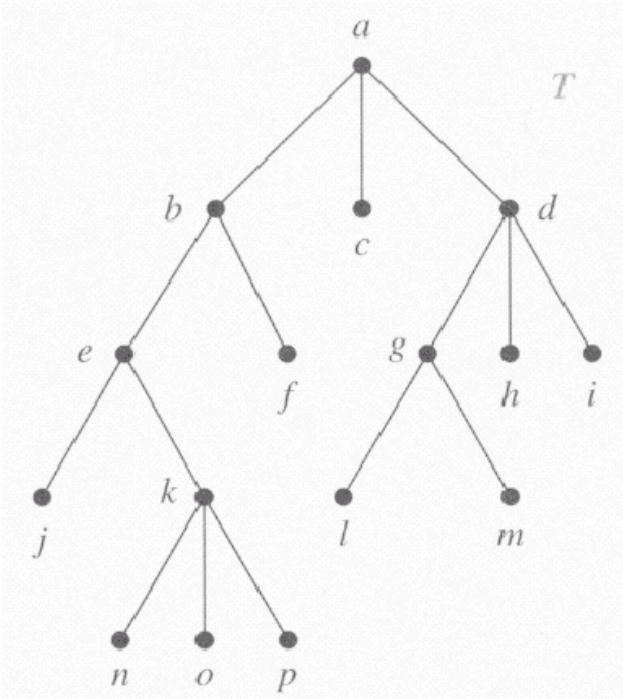



```
procedure inorder(T: ordered rooted t
r := root of T
if r is a leaf then list r
else
begin
   l:= first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l left t
     T(c) := subtree with c as its root
     preorder(T(c))
end
```

call stack
r = a, l = b s = {c,d}

output: j e n k o p b f

In which order does a inorder traversal visit the vertices in this ordered rooted tree?



procedure *inorder*(*T*: ordered rooted tree)
r := root of *T*
if r is a leaf **then** list *r*
else
begin
  l:= first child of *r* from left to right
  T(l) := subtree with *l* as its root
  inorder(T(l))
  list *r*
  for each child *c* of *r* except for *l* left to right
    T(*c*) := subtree with *c* as its root
    preorder(*T*(*c*))
end

output: j e n k o p b f a c l g m d h i

In which order does a postorder traversal visit the vertices in this ordered rooted tree?