DATA STRUCTURES AND ITS APPLICATIONS

Binary Tree Traversal

Shylaja S S & Kusuma K V
Department of Computer Science
& Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Binary Tree Traversal

Shylaja S S
Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Binary Tree Traversals

Important operation: Traversal

Traversal: Moving through all the nodes in a binary tree and visiting each one in turn

Trees: There are many orders possible since it is a nonlinear DS

Tasks: 1. Visiting a node denoted by V

2. Traversing the left subtree denoted by L

3. Traversing the right subtree denoted by R

Six ways to arrange them: VLR, LVR, LRV, VRL, RVL, RLV

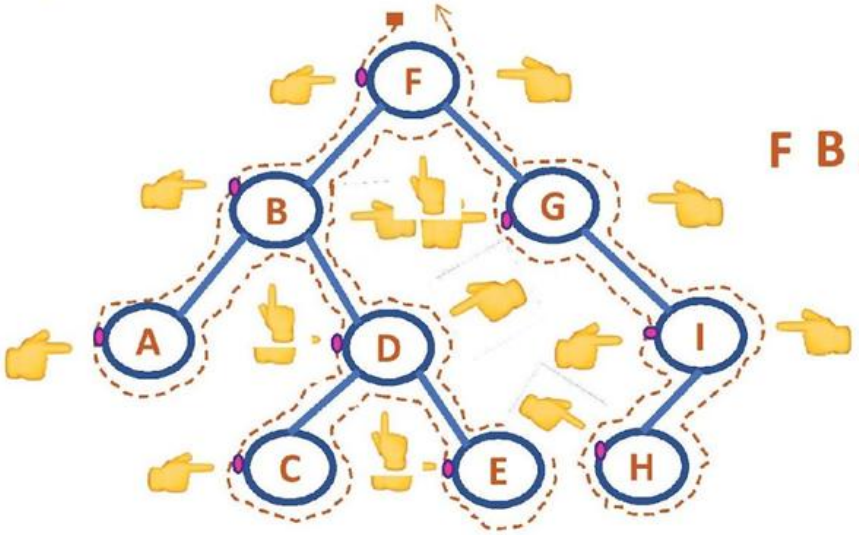
Standard Traversals include: VLR-Preorder, LVR-Inorder, LRV-Postorder

DATA STRUCTURES AND ITS APPLICATIONS

Binary Tree Traversal: Preorder

Steps:

- Root Node is visited before the subtrees
- Left subtree is traversed in preorder
- Right subtree is traversed in preorder



F B A D C E G I H

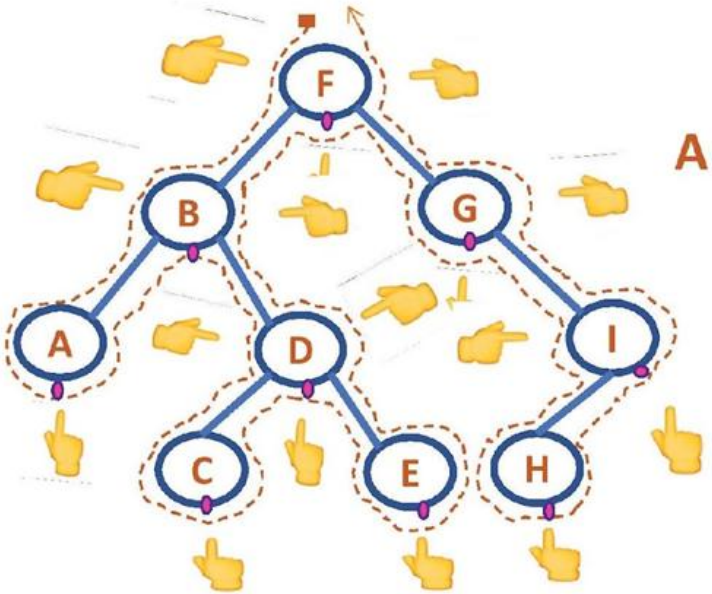
DATA STRUCTURES AND ITS APPLICATIONS

Binary Tree Traversal: Inorder

PES UNIVERSITY

Steps:

- Left subtree is traversed in Inorder
- Root Node is visited
- Right subtree is traversed in Inorder



A B C D E F G H I

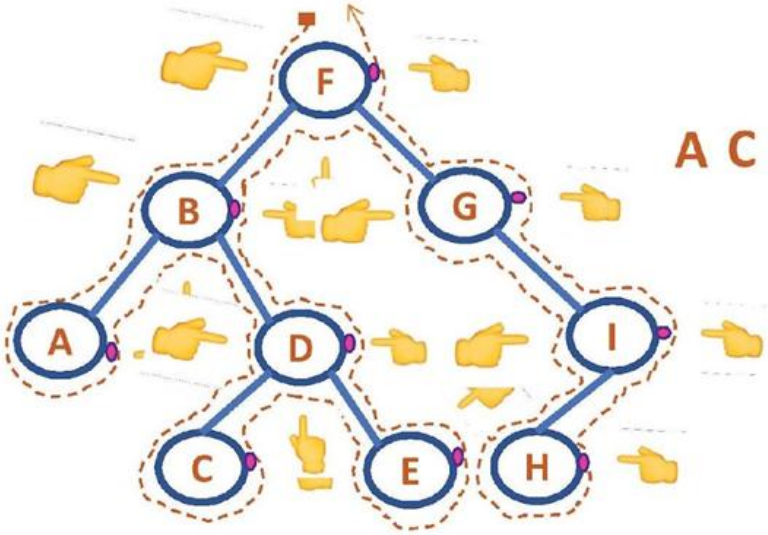
DATA STRUCTURES AND ITS APPLICATIONS

Binary Tree Traversal: Postorder

PES UNIVERSITY

Steps:

- Left subtree is traversed in postorder
- Right subtree is traversed in postorder
- Root Node is visited



A C E D B H I G F

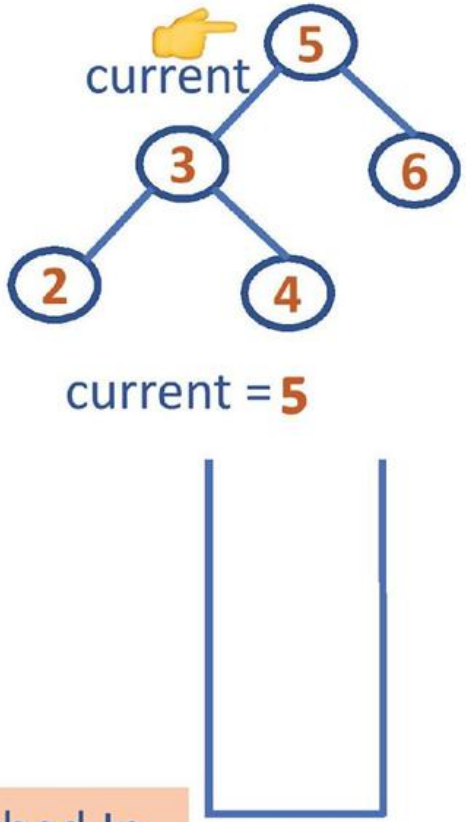
Iterative Inorder Traversal


```
iterativeInorder(root)
 s = emptyStack
current = root
do {
        while(current != null)
        {       /* Travel down left branches as far as possible
                saving pointers to nodes passed in the stack*/
                push(s, current)
                current = current->left
        } //At this point, the left subtree is empty
        poppedNode = pop(s)
        print poppedNode ->info         //visit the node
        current = poppedNode ->right //traverse right subtree
} while(!isEmpty(s) or current != null)
```

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
 s = emptyStack
current = root
do {
      while(current != null)
     {
          push(s, current)
          current = current->left
      }
      poppedNode = pop(s)
      print poppedNode ->info
      current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```

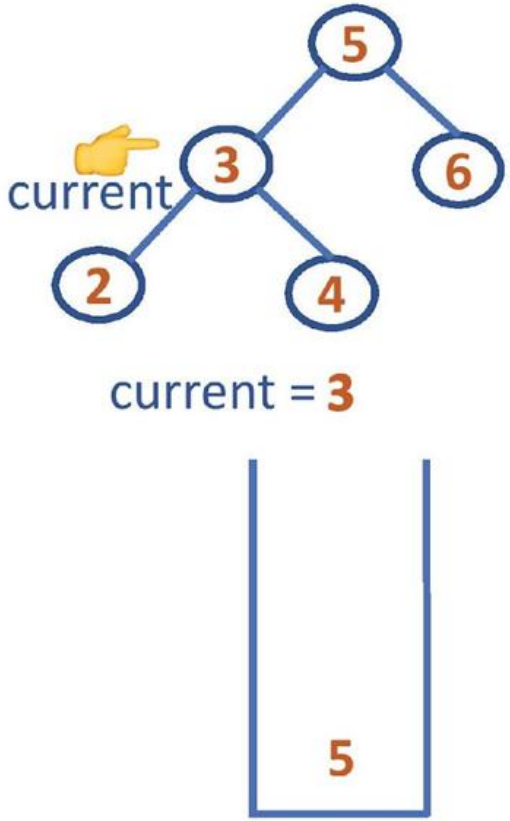


Note: Stack has Address of Nodes Pushed In

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

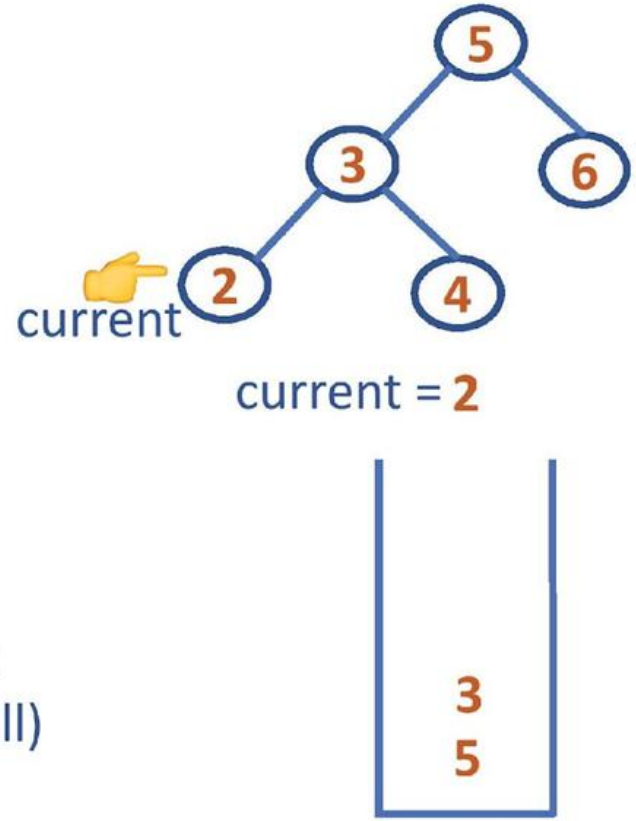
```
iterativeInorder(root)
 s = emptyStack
current = root
do {
       while(current != null)
      {
           push(s, current)
           current = current->left
       }
       poppedNode = pop(s)
       print poppedNode ->info
       current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

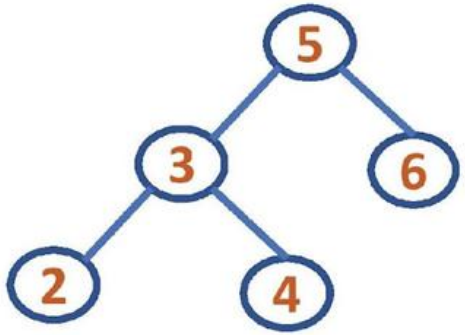
```
iterativeInorder(root)
 s = emptyStack
current = root
do {
       while(current != null)
      {
            push(s, current)
            current = current->left
       }
       poppedNode = pop(s)
       print poppedNode ->info
       current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
 s = emptyStack
current = root
do {
       while(current != null)
      {
           push(s, current)
           current = current->left
       }
       poppedNode = pop(s)
       print poppedNode ->info
       current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```

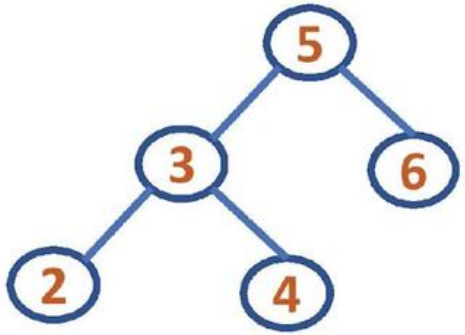


current = N

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
 s = emptyStack
current = root
do {
       while(current != null)
      {
            push(s, current)
            current = current->left
       }
       poppedNode = pop(s)
       print poppedNode ->info
       current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



current = **N**

poppedNode =

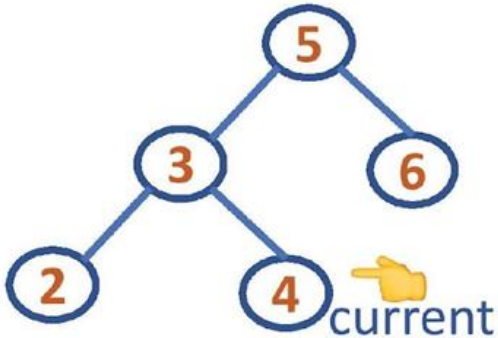
Inorder Traversal:

2

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
 s = emptyStack
current = root
do {
      while(current != null)
     {
          push(s, current)
          current = current->left
      }
      poppedNode = pop(s)
      print poppedNode ->info
      current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



current = NULL

poppedNode = 2

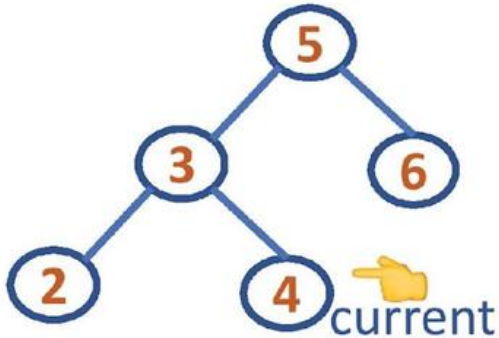
Inorder Traversal:

2 3

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
 s = emptyStack
current = root
do {
      while(current != null)
     {
          push(s, current)
          current = current->left
      }
      poppedNode = pop(s)
      print poppedNode ->info
      current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



current = **4**

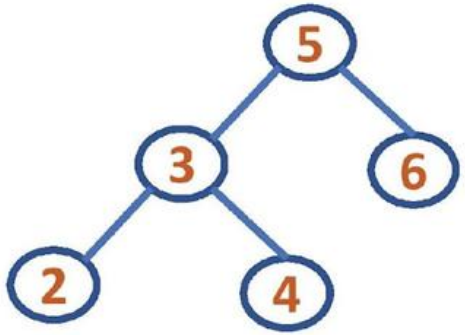
Inorder Traversal:

2 3

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
 s = emptyStack
current = root
do {
       while(current != null)
      {
           push(s, current)
           current = current->left
       }
       poppedNode = pop(s)
       print poppedNode ->info
       current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



current = N

poppedNode = 3

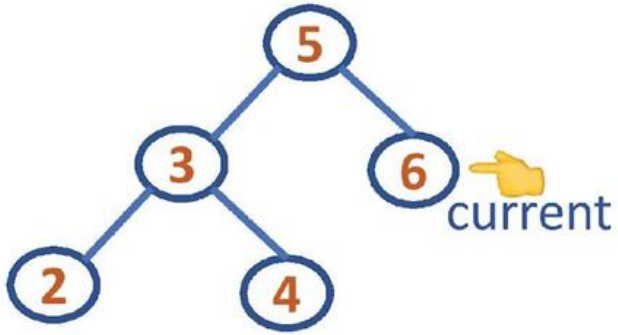
Inorder Traversal:

2 3 4

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
 s = emptyStack
current = root
do {
      while(current != null)
     {
          push(s, current)
          current = current->left
      }
      poppedNode = pop(s)
      print poppedNode ->info
      current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



current =

poppedNode = 4

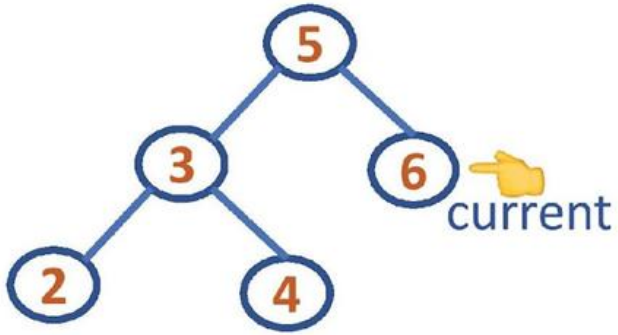
Inorder Traversal:

2 3 4 5

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
 s = emptyStack
current = root
do {
      while(current != null)
     {
          push(s, current)
          current = current->left
      }
      poppedNode = pop(s)
      print poppedNode ->info
      current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



current = **6**

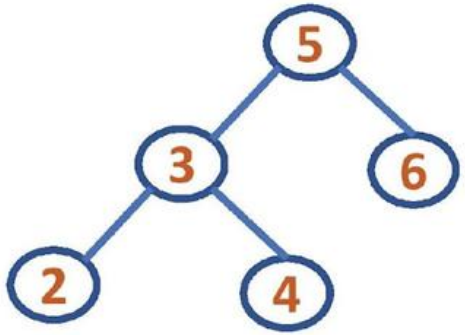
Inorder Traversal:

2 3 4 5

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
 s = emptyStack
current = root
do {
       while(current != null)
      {
           push(s, current)
           current = current->left
       }
       poppedNode = pop(s)
       print poppedNode ->info
       current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



current = N

poppedNode =

PES UNIVERSITY

Inorder Traversal:

2 3 4 5 6

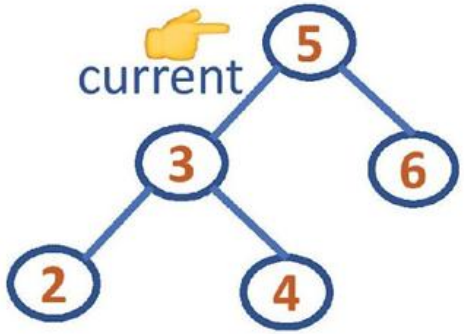
Iterative Preorder Traversal


```
iterativePreorder(root)
current=root
if (current == null)
   return
s = emptyStack
push(s, current)
while(!isEmpty(s)) {
      current = pop(s)
      print current->info
      //right child is pushed first so that left is processed first
      if(current->right !=NULL)
            push(s, current->right)
      if(current->left !=NULL)
            push(s, current->left)
}
```

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
current=root
if (current == null)
    return
s = emptyStack
push(s, current)
    ...
```



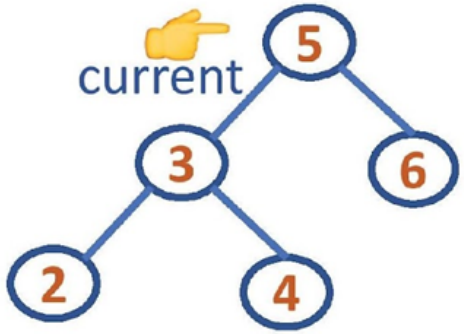
current = **5**

Note: Stack has Address of Nodes Pushed In

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
    ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 5

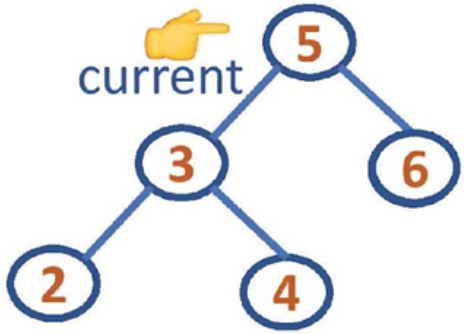
Preorder Traversal:

5

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
    ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 5

current->right = 6

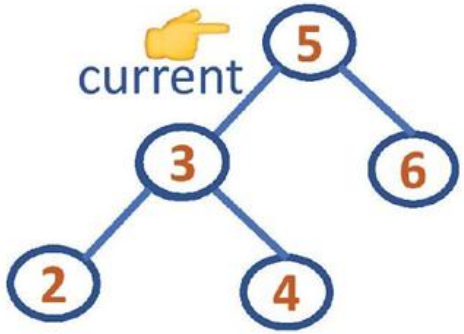
Preorder Traversal:

5

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
    ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 5

current->left = 3

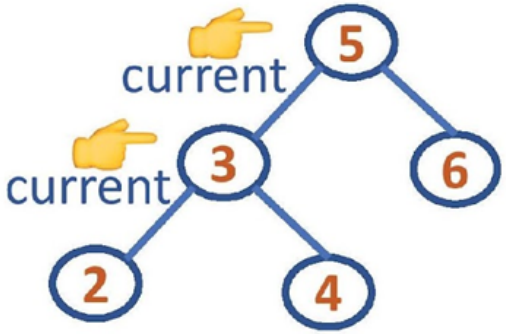
Preorder Traversal:

5

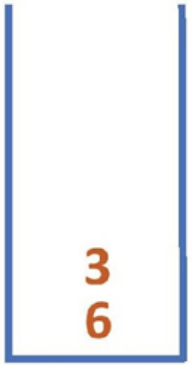
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
    ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 3



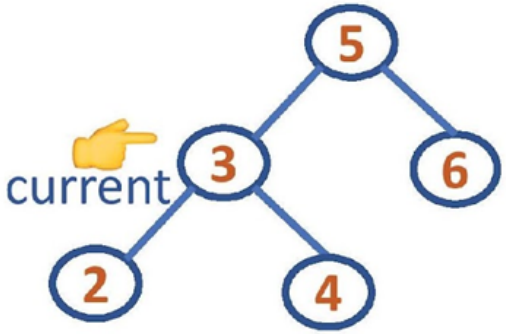
Preorder Traversal:

5 3

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
      ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 3

current->right = 4

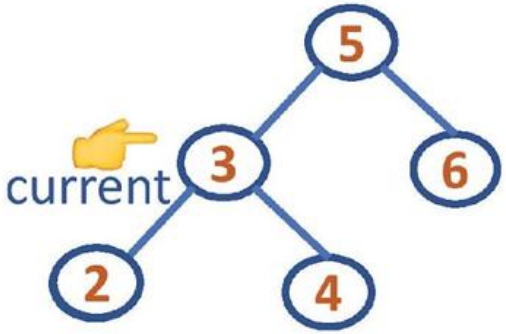
Preorder Traversal:

5 3

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
    ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = **3**

current->left = **2**

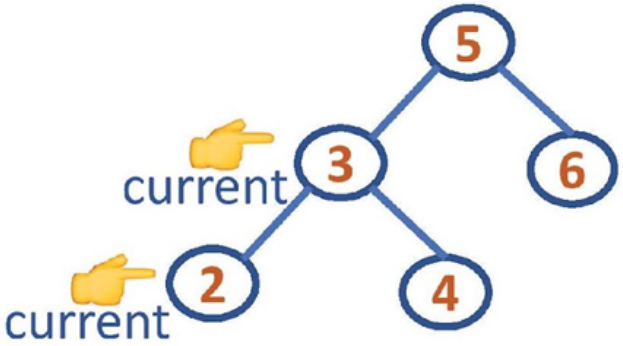
Preorder Traversal:

5 3

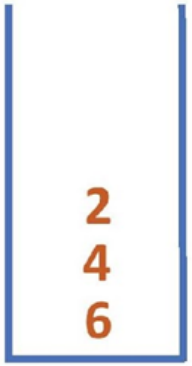
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
      ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 3



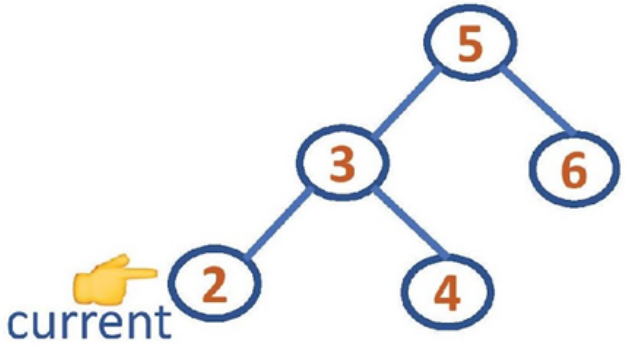
Preorder Traversal:

5 3 2

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
      ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 2

current->right = N

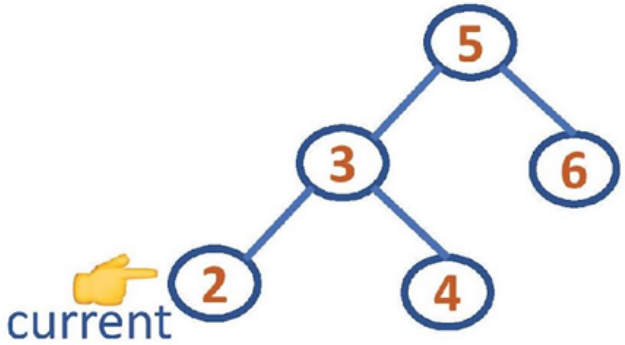
Preorder Traversal:

5 3 2

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
      ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 2

current->left = N

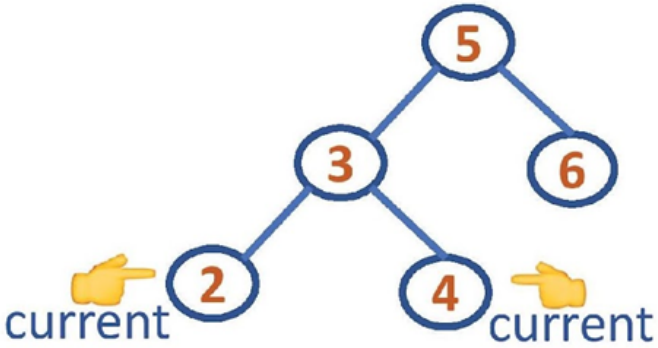
Preorder Traversal:

5 3 2

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
    ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 2

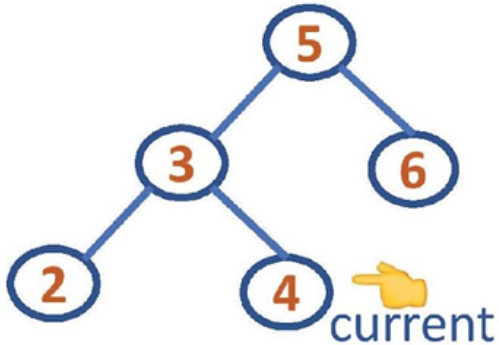
Preorder Traversal:

5 3 2 4

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
      ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



current = 4

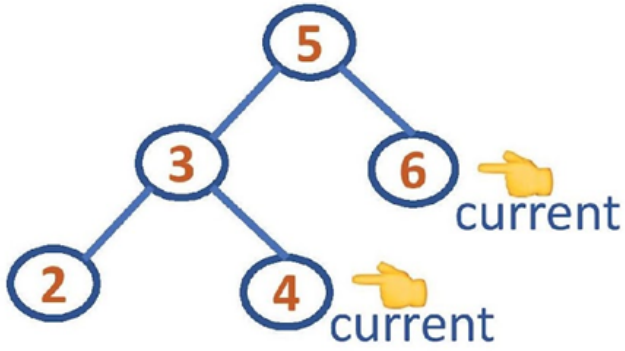
current->right = N
current->left = N

Preorder Traversal:
5 3 2 4

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
    ::::
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



Preorder Traversal:
5 3 2 4 6

current = 4

current->right = N
current->left = N

Iterative Postorder Traversal

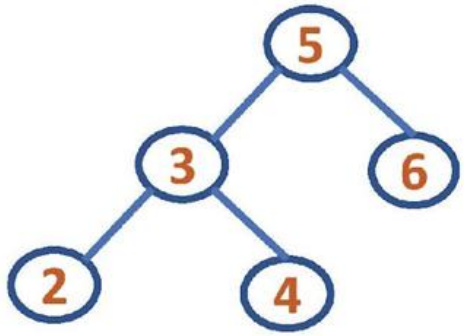

```
iterativePostorder(root)
s1 = emptyStack ; s2 = emptyStack ; push(s1, root)
while(!isEmpty(s1)) {
      current = pop(s1)
      push(s2,current)
      if(current->left !=NULL)
            push(s1, current->left)
      if(current->right !=NULL)
            push(s1, current->right)
}
while(!isEmpty(s2)) {   //Print all the elements of stack2
      current = pop(s2)
      print current->info
}
```

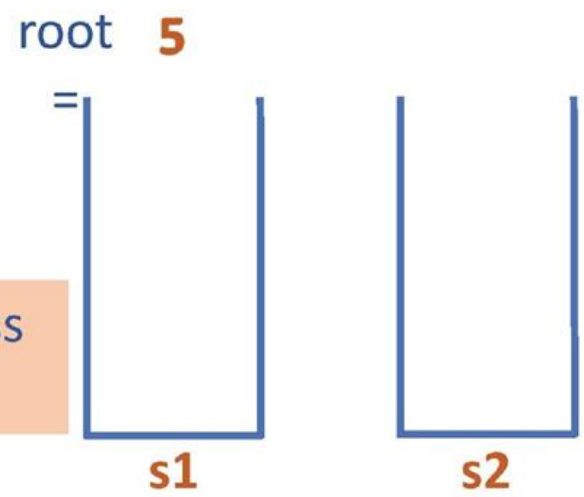
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

PES UNIVERSITY

iterativePostorder(root)
s1 = emptyStack 👈
s2 = emptyStack 👈
push(s1, root) 👈
⋮⋮



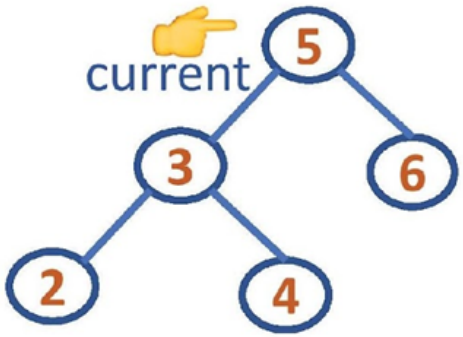


Note: Stacks have Address of Nodes Pushed In

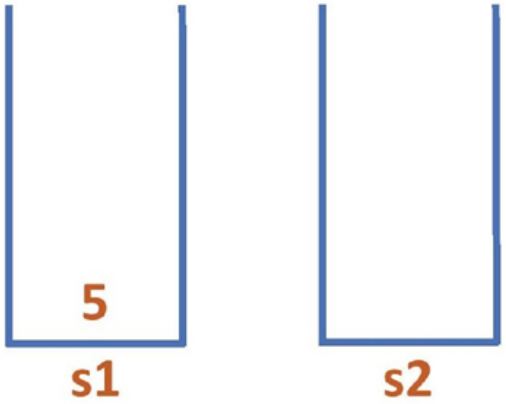
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



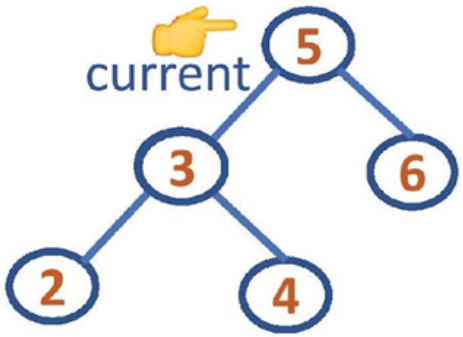
current =



DATA STRUCTURES AND ITS APPLICATIONS

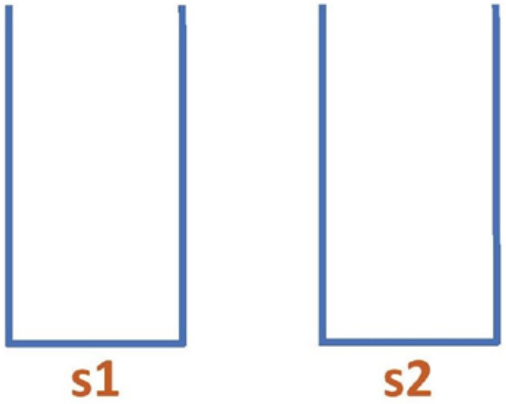
Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



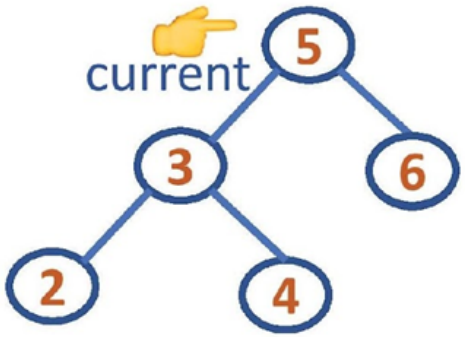
current = **5**

current->left = **3**



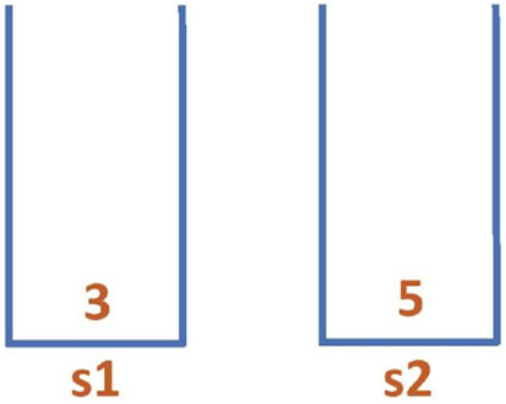
Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



current = 5

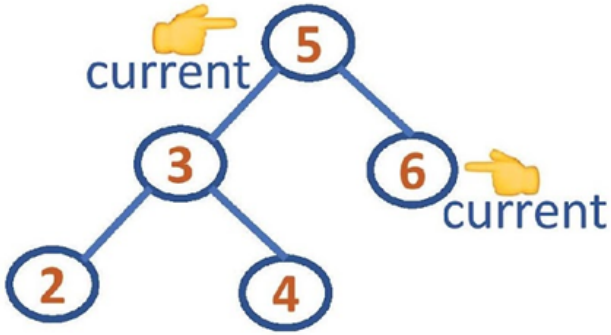
current->right = 6



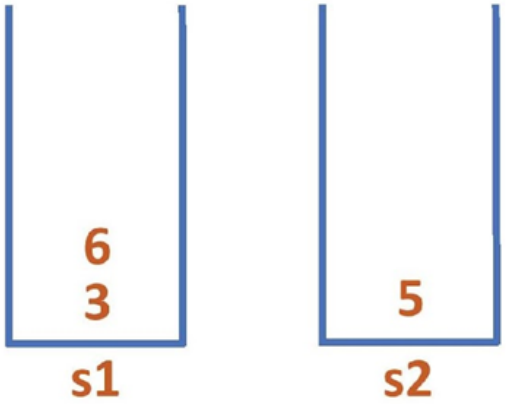
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      :::
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      :::
```



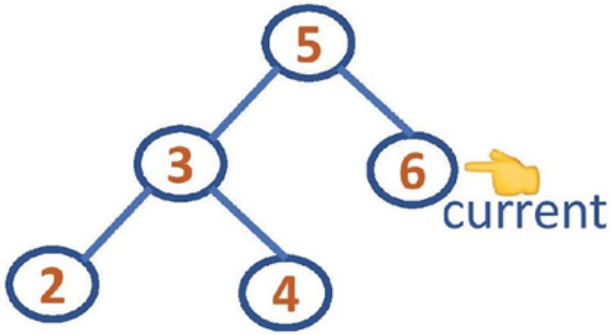
current = 5



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

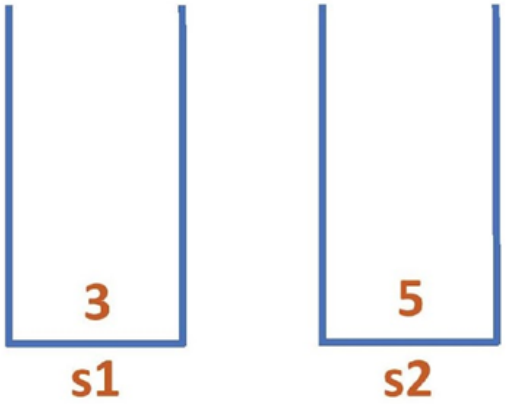
```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



current = **6**

current->left = **N**

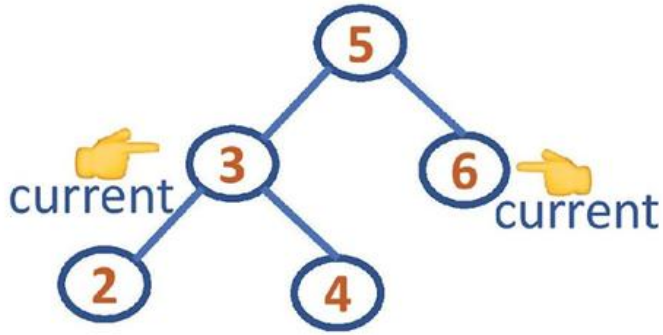
current->right = **N**



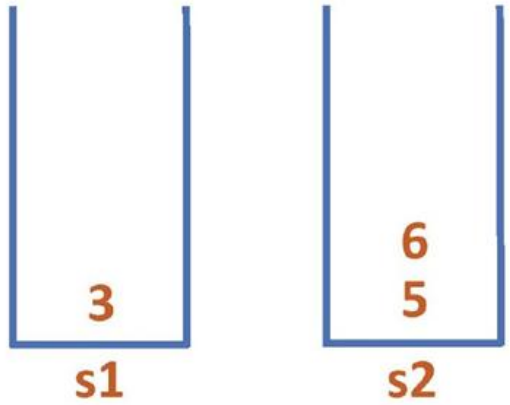
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



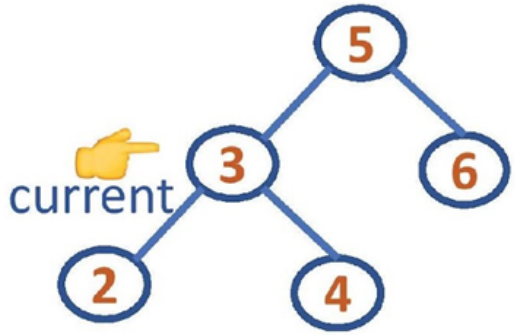
current = 6



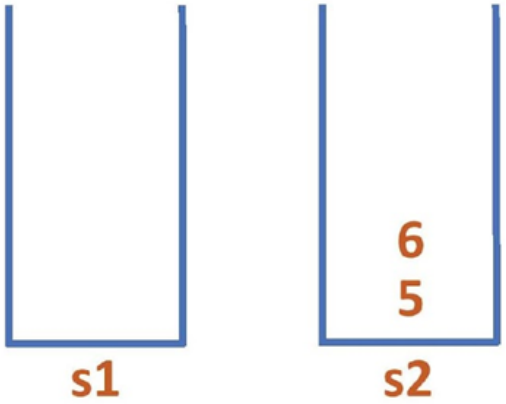
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```

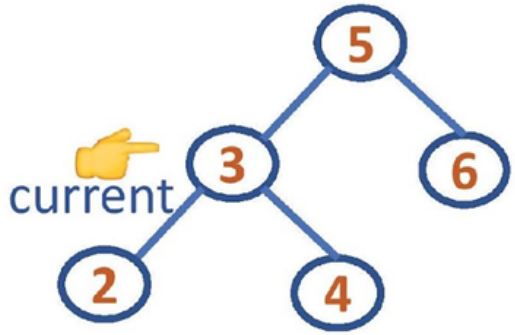


current = **3**



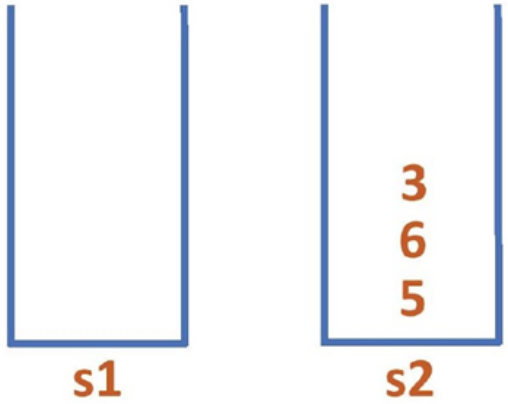
Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



current = 3

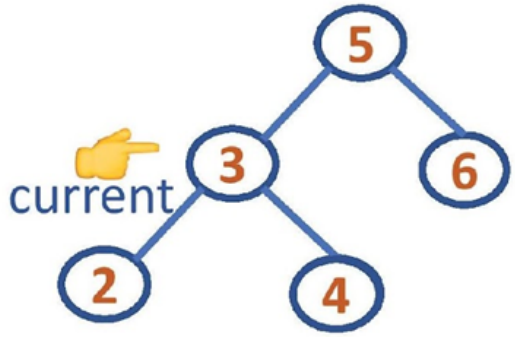
current->left = 2



DATA STRUCTURES AND ITS APPLICATIONS

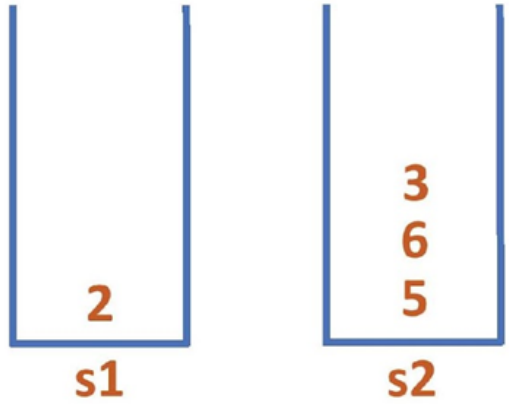
Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



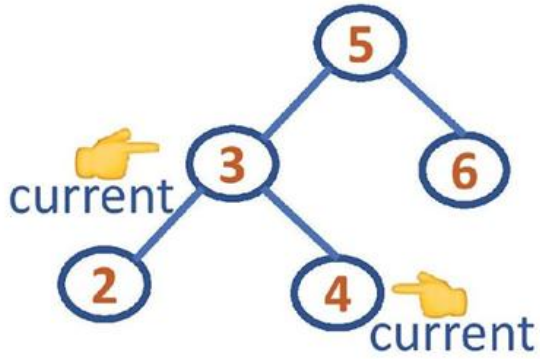
current = 3

current->right = 4

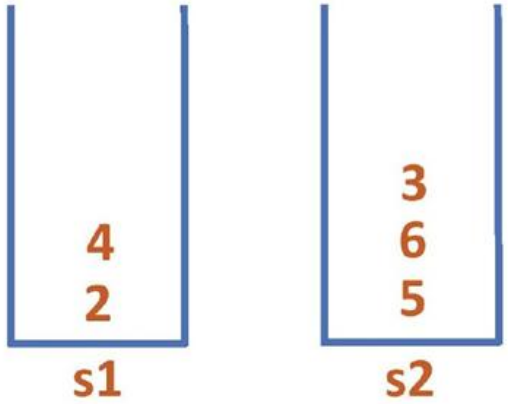


Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



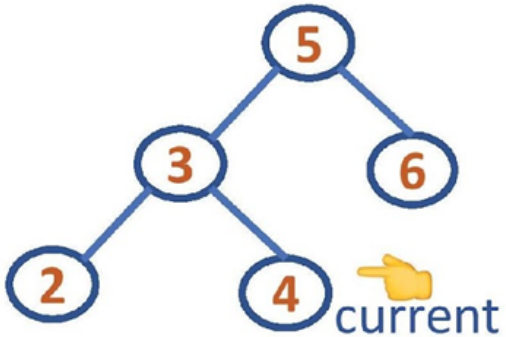
current = 3



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

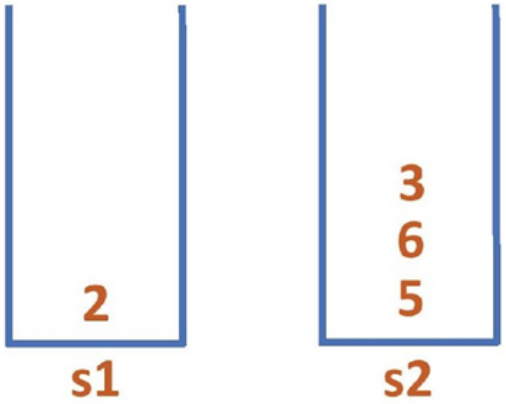
```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



current = **4**

current->left = **N**

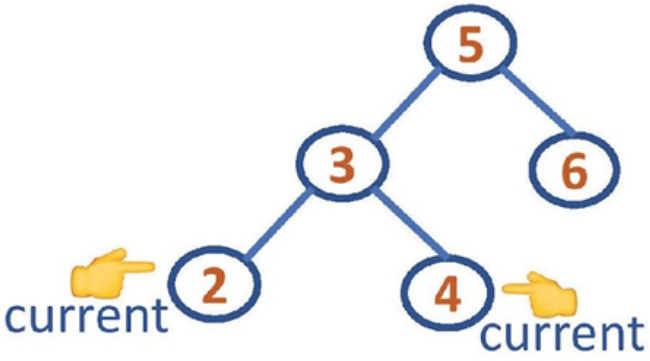
current->right = **N**



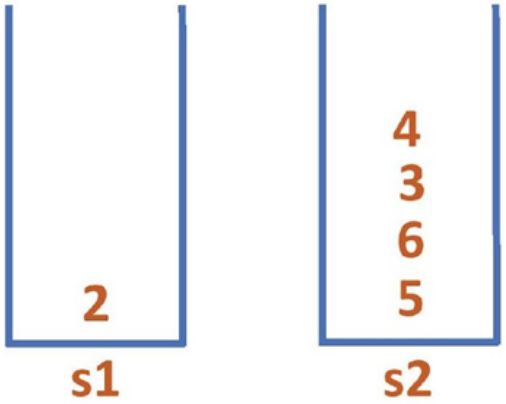
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



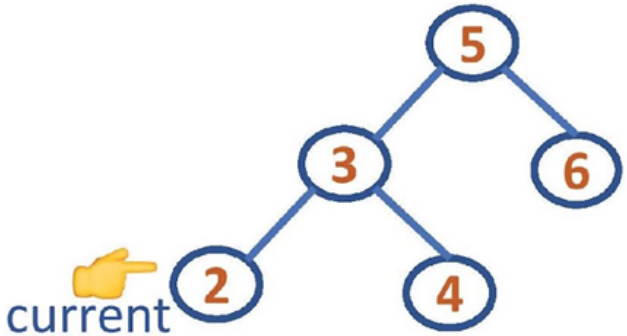
current = 4



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

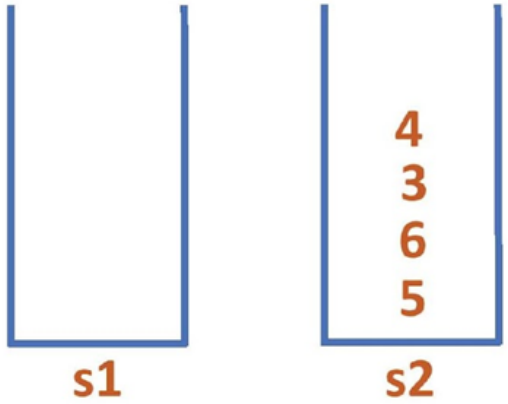
```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



current = **2**

current->left = **N**

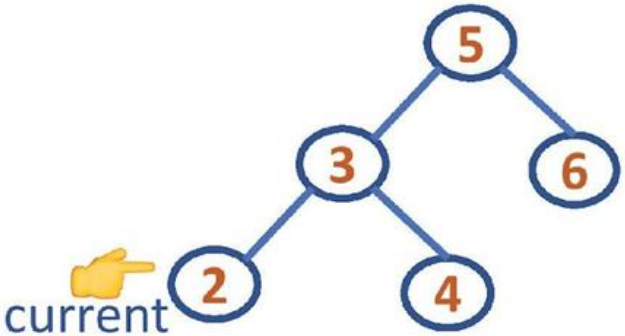
current->right = **N**



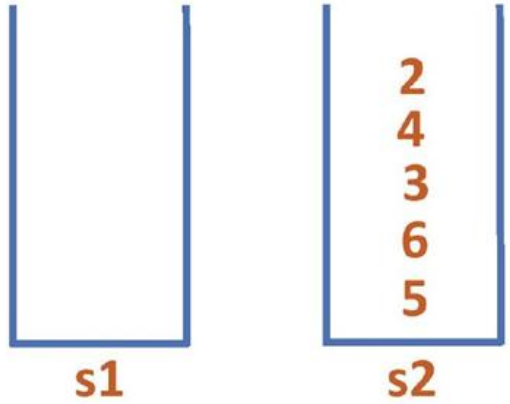
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
while(!isEmpty(s2)) {
     current = pop(s2)
     print current->info
}
```

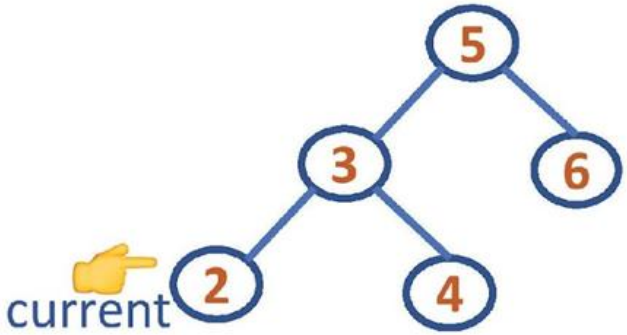


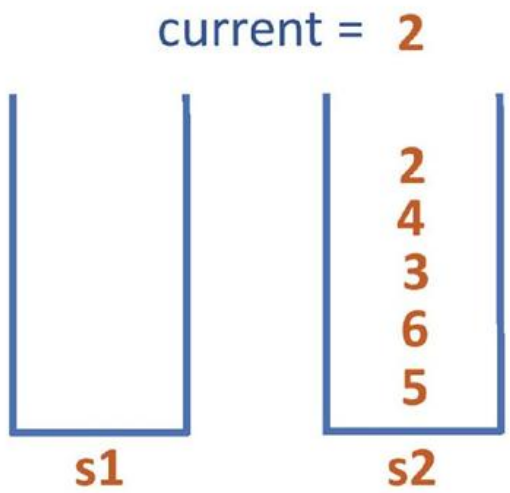
current = 2



Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
while(!isEmpty(s2)) {
     current = pop(s2)
     print current->info
}
```





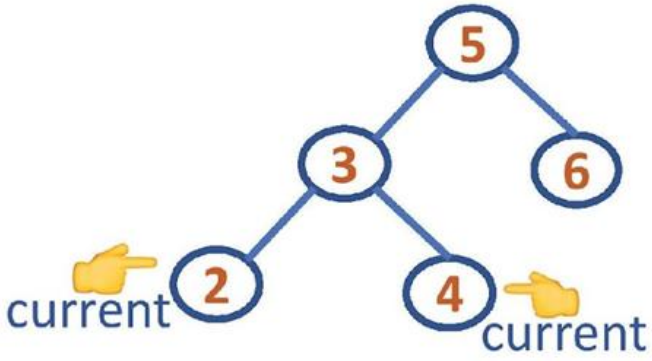
Postorder Traversal:

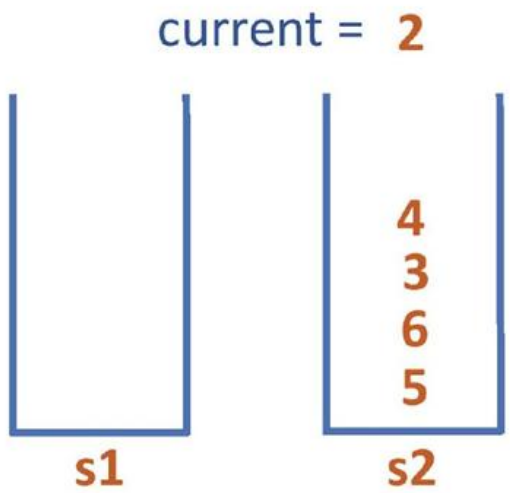
2

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
while(!isEmpty(s2)) {
     current = pop(s2)
     print current->info
}
```





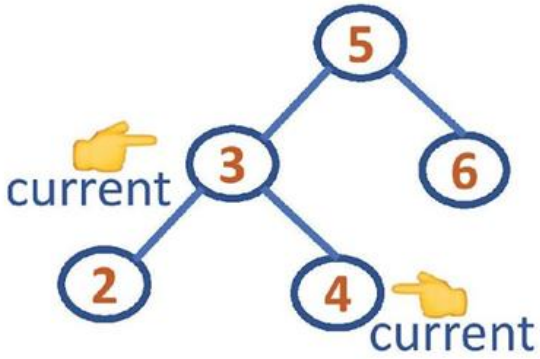
Postorder Traversal:

2 4

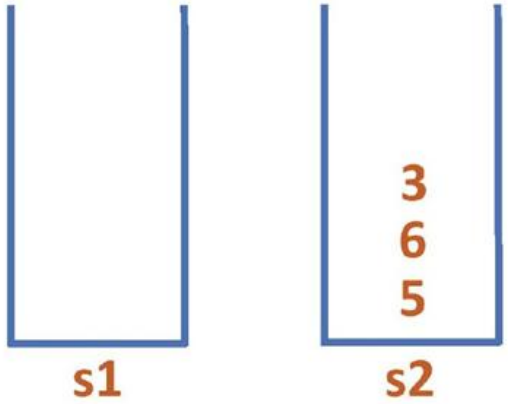
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
while(!isEmpty(s2)) {
     current = pop(s2)
     print current->info
}
```



current = 4



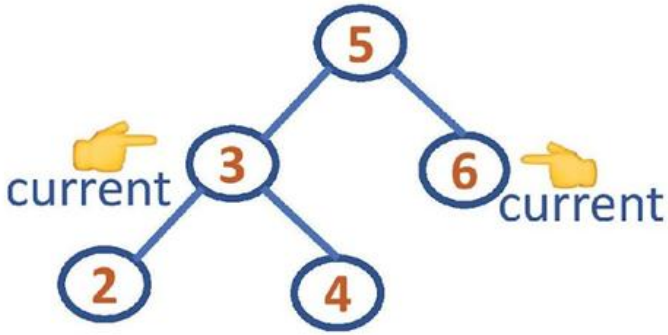
Postorder Traversal:

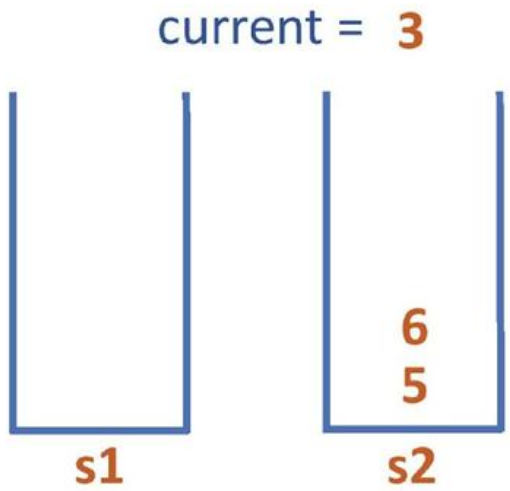
2 4 3

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
while(!isEmpty(s2)) {
     current = pop(s2)
     print current->info
}
```





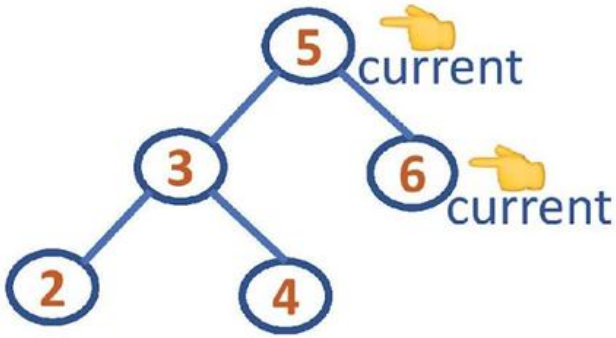
Postorder Traversal:

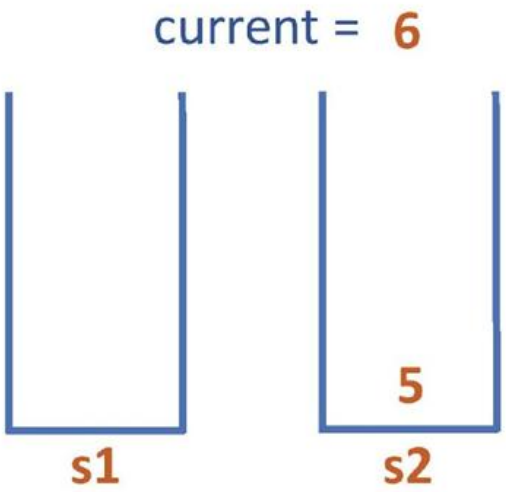
2 4 3 6

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
while(!isEmpty(s2)) {
     current = pop(s2)
     print current->info
}
```





Postorder Traversal:

2 4 3 6 5

1. Which is the correct method for iterative postorder traversal using two stacks?

A) Push root to stack1, pop from stack1, push children to stack2, then output stack2.

B) Push root to stack1, pop from stack1, push to stack2, and push its children into stack1.

C) Perform inorder traversal and reverse the order.

D) Always push left child only into stack2.

1. Which is the correct method for iterative postorder traversal using two stacks?

A) Push root to stack1, pop from stack1, push children to stack2, then output stack2.

B) Push root to stack1, pop from stack1, push to stack2, and push its children into stack1.

C) Perform inorder traversal and reverse the order.

D) Always push left child only into stack2.

2. Which traversal is most suitable for evaluating an expression tree using a stack?

A) Inorder
B) Preorder
C) Postorder
D) Level-order

2. Which traversal is most suitable for evaluating an expression tree using a stack?

A) Inorder
B) Preorder
C) Postorder
D) Level-order

3. Suppose you want to print all values of a BST in ascending order without recursion. Which traversal and data structure would you use?

A) Preorder traversal with stack
B) Inorder traversal with stack
C) Postorder traversal with queue
D) Level-order traversal with queue

3. Suppose you want to print all values of a BST in ascending order without recursion. Which traversal and data structure would you use?

A) Preorder traversal with stack

B) Inorder traversal with stack

C) Postorder traversal with queue

D) Level-order traversal with queue

4. Which of the following is true about level-order traversal?

A) It uses recursion only
B) It visits nodes depth-first
C) It visits nodes breadth-first
D) It always requires a stack

4. Which of the following is true about level-order traversal?

A) It uses recursion only

B) It visits nodes depth-first

C) It visits nodes breadth-first

D) It always requires a stack

5. You want to print all paths from root to leaves. Which traversal is generally preferred?

A) Pre-order
B) In-order
C) Post-order
D) Level-order

5. You want to print all paths from root to leaves. Which traversal is generally preferred?

A) Pre-order
B) In-order
C) Post-order
D) Level-order

(Explanation: Pre-order allows keeping track of the current path from root to node.)

THANK YOU

Shylaja S S

Department of Computer Science
& Engineering

shylaja.sharath@pes.edu

+91 9449867804