Binary trees

Traversal-Navigation

D.E ZEGOUR

Ecole Supérieure d'Informatique

ESI

Binary trees : Traversal & Navigation

Traversal

- Visit all the nodes in a given order
  - Recursive algorithms
  - Iterative algorithms
    - With a stack
    - Without a stack
      - With the Parent operation
      - With node threading

Navigation

- How to go from one node to another ?
  - With a stack
  - With node threading
  - With the Parent operation

Binary trees : Traversal

Recursive traversal

Inorder traversal

Formula : T1 n T2

⇩

Inorder(n) :

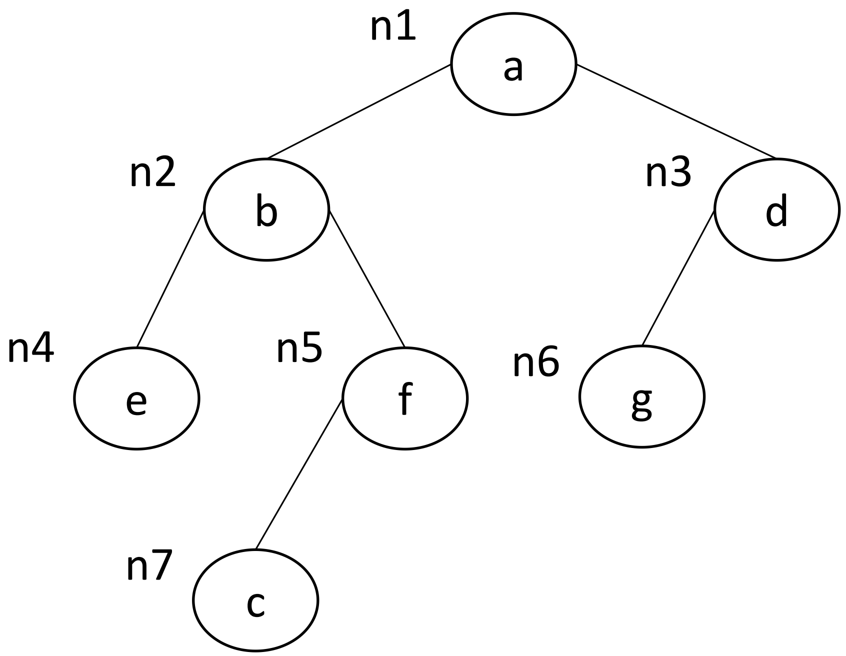
IF n <> nil

Inorder (Lc(n))

Write (Node_value(n))

Inorder(Rc(n))

ENDIF



In(n1) = In(n2), a, In(n3)

In(1) = In(n4), b, In(n5), a, In(n3)

In(1) = e, b, In(n5), a, In(n3)

In(1) = e, b, In(7),f, a, In(n3)

In(1) = e, b, c, f, a, In(n3)

In(1) = e, b, c, f, a, In(6), d

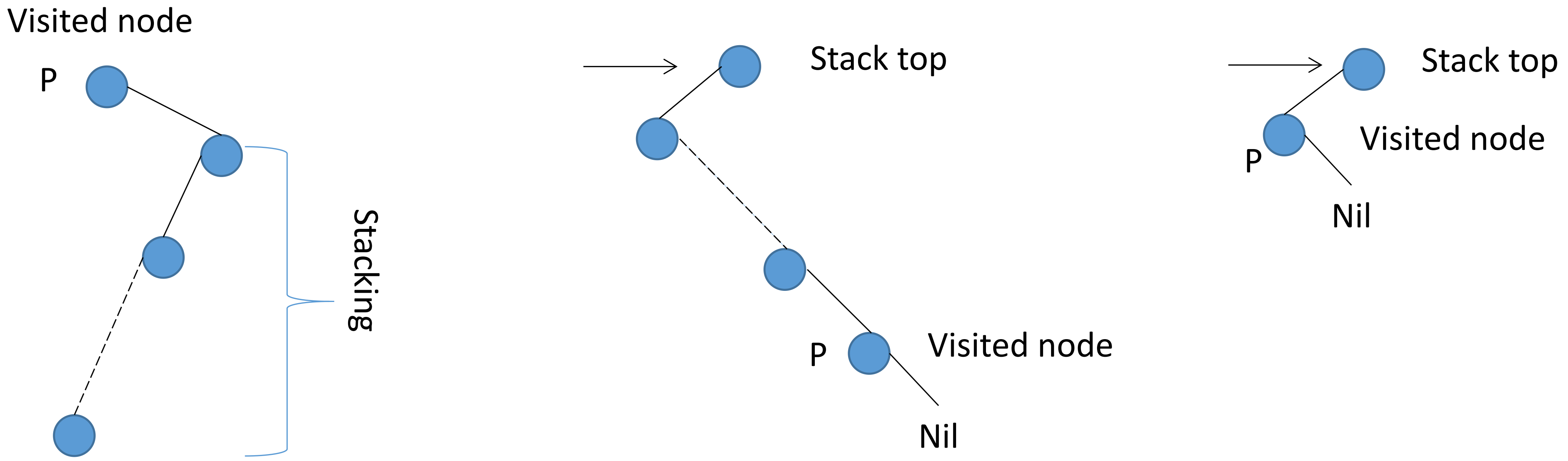
In(1) = e, b, c, f, a, g, d

Binary trees : Traversal

Iterative traversal with a stack

Inorder traversal

Consequence: The stack contains all the nodes not yet visited through which we exit on the left.



With each visit to a node,
If it has a right child, we continue to descend always to the left of this node by stacking all the nodes.

With each visit to a node,
- If it does not have a right child, the next one to visit is at the top of the stack.

Binary trees : Traversal

Iterative traversal with a stack

Inorder traversal

```
P := A ; Createstack(Pil)
Possible := TRUE
WHILE Possible
    WH P <> NIL
        Push( Pil , P )
        P := LC( P )
    EWH
    IF NOT Empty_stack( Pil )
        Pop( Pil , P )
        Write( INFO ( P ) )
        P := RC( P )
    ELSE
        Possible := FALSE
    ENDIF
ENDWHILE
```

Pushing nodes

Pop, visit and go to the right

Binary trees : Traversal

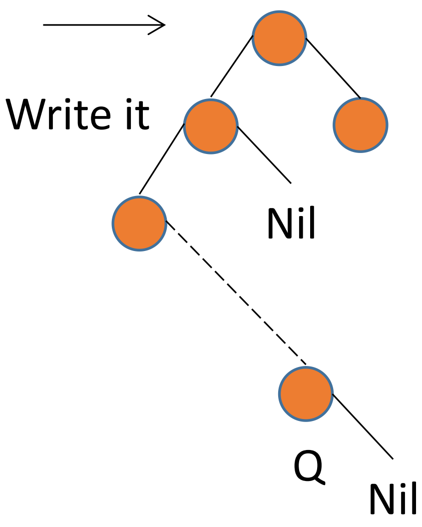
Iterative traversal with the Parent operation

Inorder traversal

```
P := Root
Possible := TRUE
WHILE Possible
      WHP # NIL
           Q := P
           P := Lc( P )
      EWH
      Write( Node_value( Q ) )
      IF Rc( Q ) <> NIL
           P := Rc( Q )
      ELSE
           Go back
      ENDIF
ENDWHILE
```

Go back

Go back to the first node through which we ascend on the left that has a right child.




```
P := Parent( Q )
Continue := TRUE
WHILE( P <> NIL ) AND Continue
      IF Q = Rc( P )
            Q := P
            P := Parent( P )
      ELSE
            IF Rc( P ) = NIL
                  Write( Node_value( P ) )
                  Q := P
                  P := Parent( P )
            ELSE
                  Continue := FALSE
            ENDIF
      ENDIF
ENDWHILE
IF P <> NIL
      Write( Node_value( P ) )
      P := Rc( P )
ELSE
      Possible := FALSE
ENDIF
```

Binary trees : Traversal

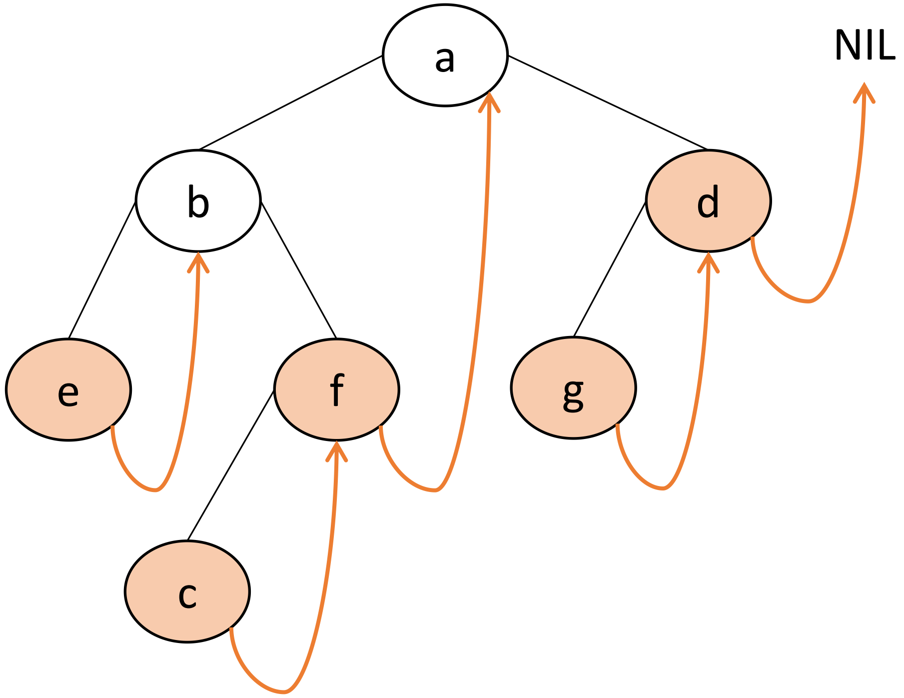
Threaded Binary Search trees

Exploit the right child field of nodes if it is equal to Nil.

Instead of pointing to NIL, it will point to the Inorder successor.

Requires an additional field to distinguish between threaded nodes and non-threaded nodes.

Add to the abstract machine : Threaded(P)
Ass_Threaded(P, Bool)

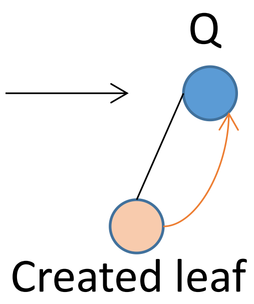


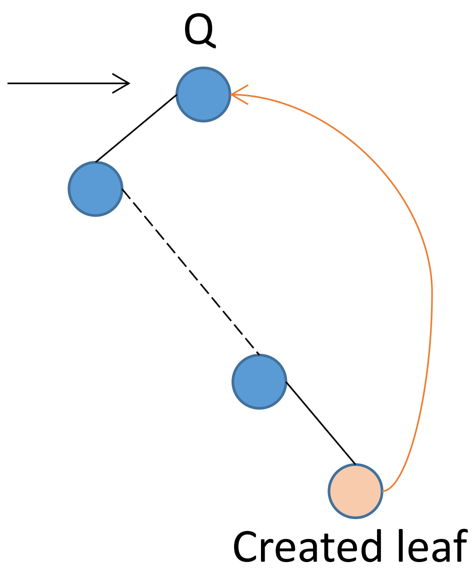
Binary trees : Traversal

Threaded Binary Search trees

In the search phase, save the last node (let's call it Q) through which we exit on the left.

The created leaf will point to its right to the node Q.



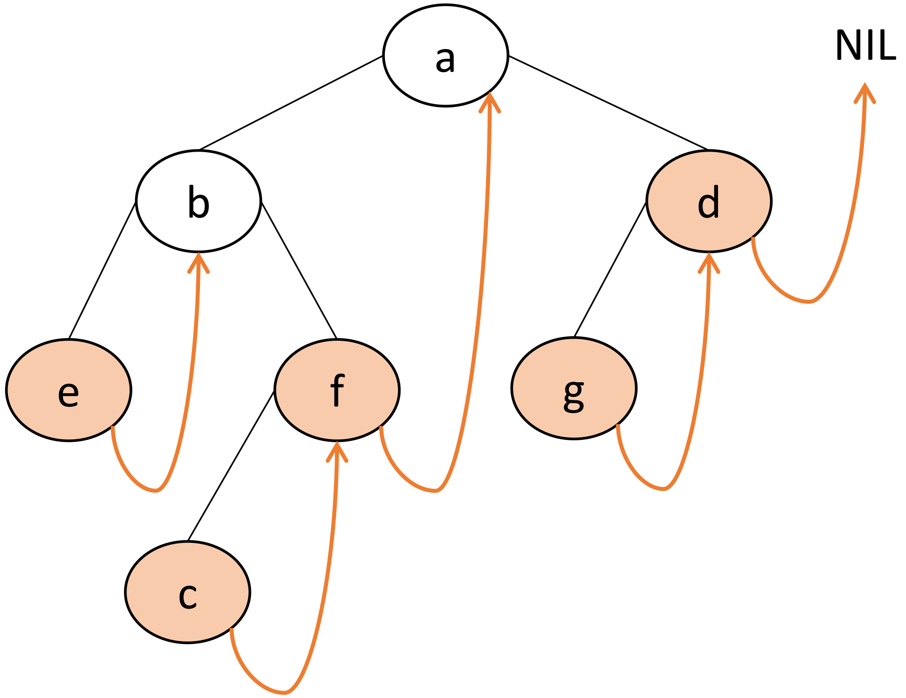


Binary trees : Traversal

Iterative traversal with node threading

Inorder traversal

```
P := Root
WHILE P <> NIL
   WH P # NIL
      Q := P
      P := Lc( P )
   EWH;
   Write( Node_value ( Q ) )
   P := Rc( Q )
   WH ( Threaded( Q ) ) AND ( P # NIL )
      Q := P
      Write( Node_value( Q ) )
      P := Rc( Q )
   EWH
ENDWHILE
```

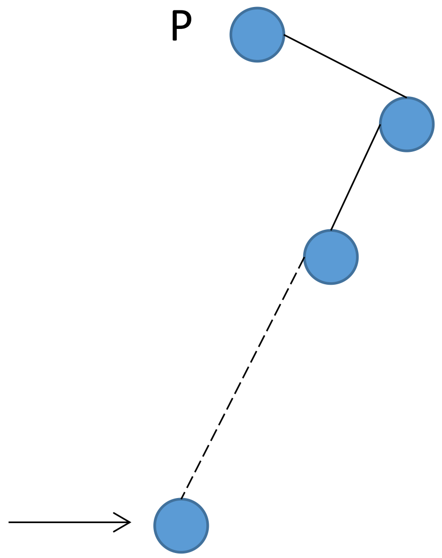


Binary trees : Navigation

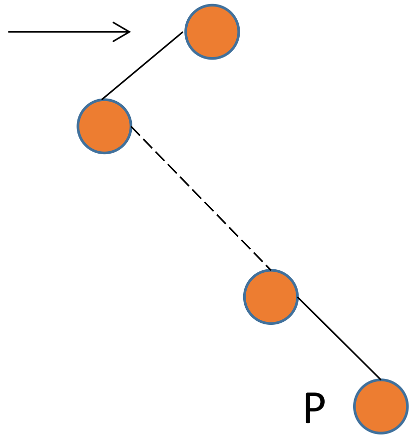
Navigation using a stack

Next inorder

P has a right child



The stack contains the path from the root to the parent of P.
If the stack is empty, possible = False.



P does not have a right child

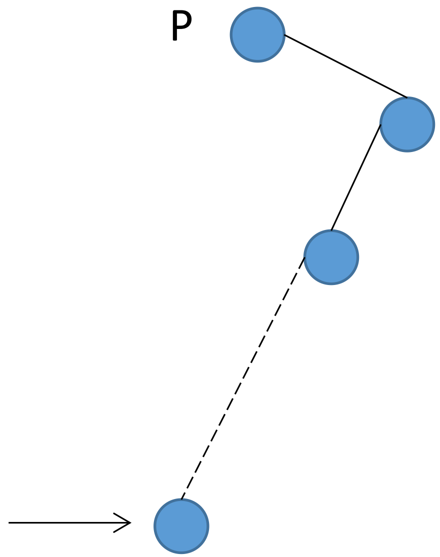
```
IF Rc(P) <> NIL
    P := Rc(P)
    WH Lc(P )<> NIL
       P := Lc(P)
    EWH;
    Next_inorder := P
ELSE
      Pop(a_stack, N,Possible)
      Stop := False;
       WH NOT Stop AND Possible
          IF P = Lc(N)
             Stop := True
          ELSE
             P:= N
             Pop(a_stack, N, Possible)
          ENDIF
       EWH
       IF Stop
            Next_inorder := N
       ELSE
            Next_inorder := NIL
       ENDIF
 ENDIF
```

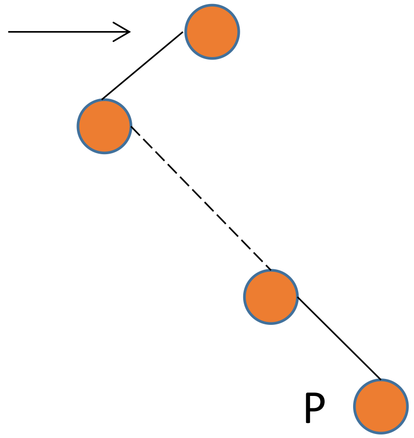
Binary trees : Navigation

Navigation using the Parent operation

Next inorder

P has a right child





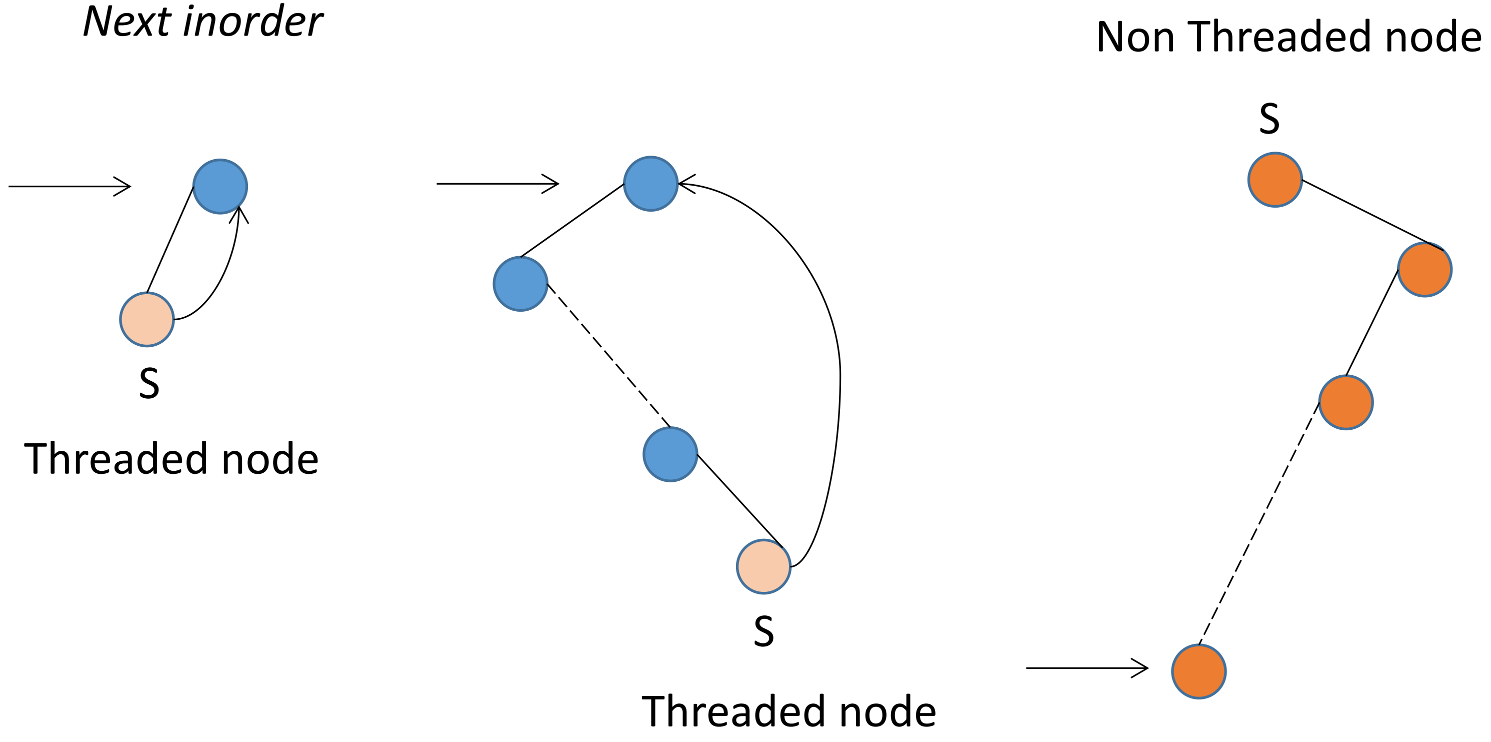
P does not have a right child

```
IF Rc( P ) <> NIL
   P := Rc( P ) ;
   WH Lc( P ) <> NIL
      P := Lc( P )
   EWH;
   Next_inorder := P
ELSE
   Q := Parent( P )
   Continue := TRUE
   WH ( Q <> NIL ) AND Continue
      IF P = Rc( Q )
         P := Q
         Q := Parent( P )
      ELSE
         Continue := FALSE
      ENDIF
   EWH;
   IF Q <> NIL
      Next_inorder :=  Q
   ELSE
      Next_inorder :=  Nil
   ENDIF
ENDIF
```

Binary trees : Navigation

Navigation using node threading

Next inorder




```
IF Threaded( S )
    Next_inorder := Rc( S )
ELSE
    P := Rc( S )
    WH Lc( P ) # NIL
        P := Lc( P )
    EWH
    Next_inorder := P
ENDIF
```

Binary trees : Traversal & Navigation

Synthesis

For the traversal, we considered the inorder; we can redo everything with the preorder and the postorder.

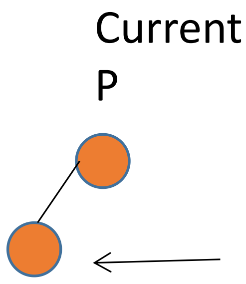
We considered right-threaded binary trees; we can consider left-threaded binary trees.

For navigation, we considered the next inorder; we can redo everything with the next preorder and the next postorder.

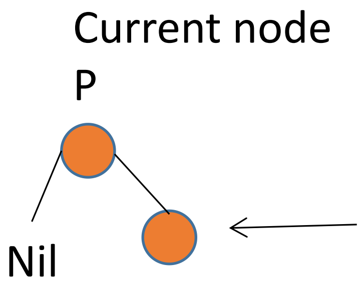
Binary trees : Navigation

Navigation : additional information

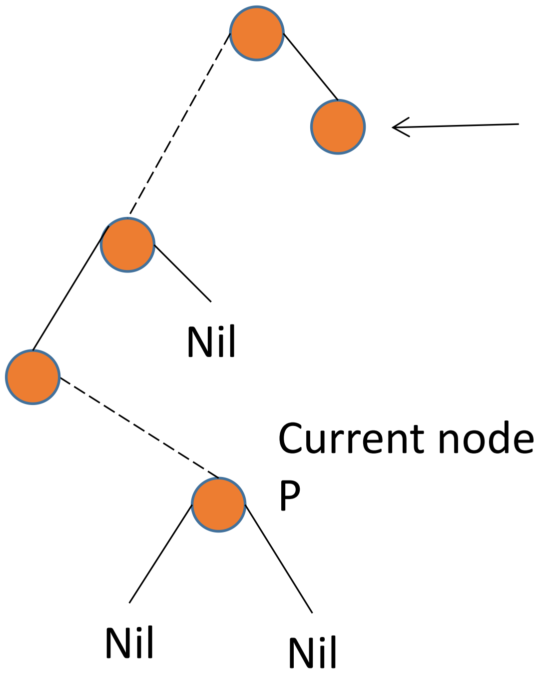
Next preorder



It’s the left child



It’s the right child

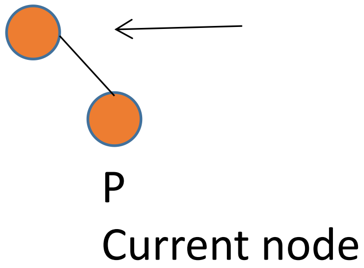


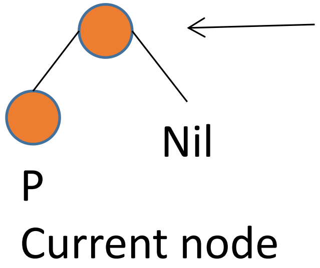
It refers to the right child, if it exists, of the first node encountered as we ascend to the left.

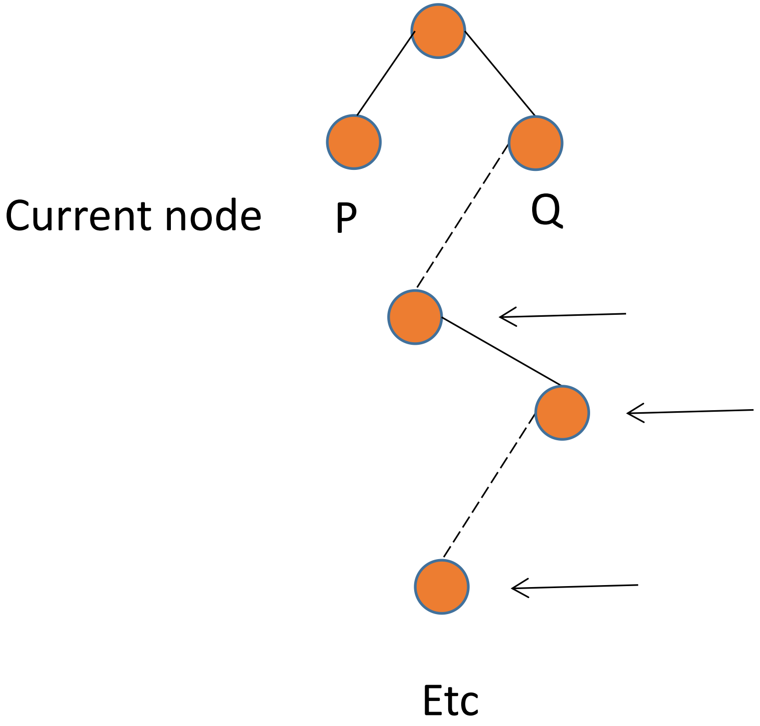
Binary trees : Navigation

Navigation : additional information

Next postorder







It’s the parent of P

It’s the leftmost leaf of the right sub tree of node Q http://zegour.esi.dz/Cours/Animation%20arbres.htm

Binary trees : Traversal & Navigation

Educational software : Accrobaties on binary search trees

3 types of trees : Binary search tree, AVL tree, threaded tree

Presentation of abstract machines

Construction, Traversal, Navigation

More than thirty Java programs

With and without animation