

# COMP 250

## INTRODUCTION TO COMPUTER SCIENCE

Assignment Project Exam Help

<https://powcoder.com>

Week 11-2 : Tree Traversals

Giulia Alberini, Fall 2020

Slides adapted from Michael Langer's

# WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Tree traversals
- Depth first VS Breadth first
- Recursive and Non-recursive (with stack or with queue)

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help

TRAVERSALS  
<https://powcoder.com>

Add WeChat powcoder

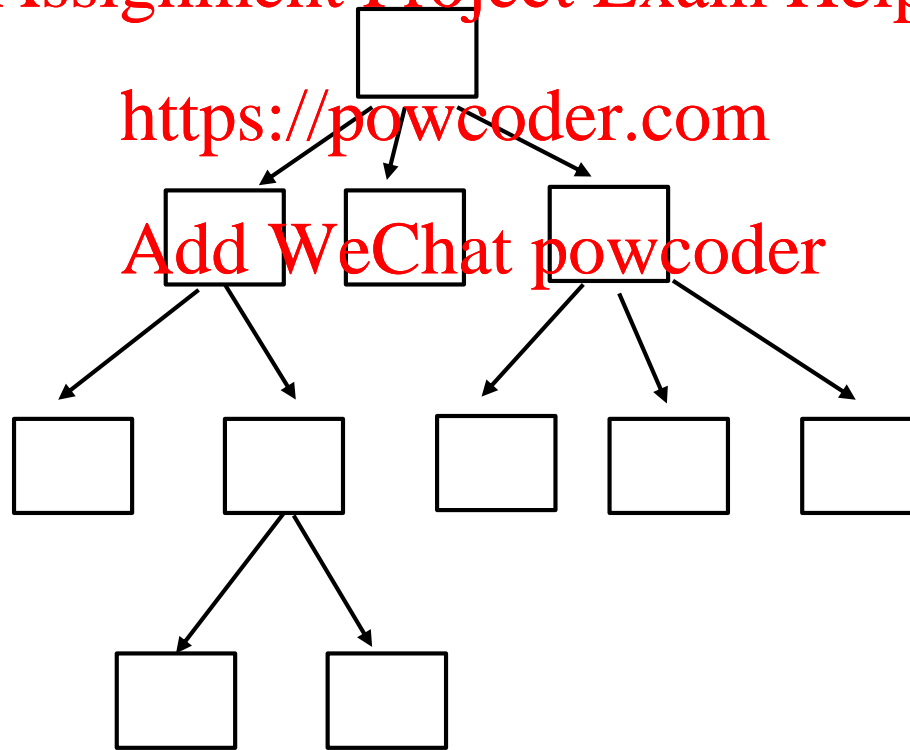
# TREE TRAVERSAL

How to visit (enumerate, iterate through, traverse... ) all the nodes of a tree ?

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



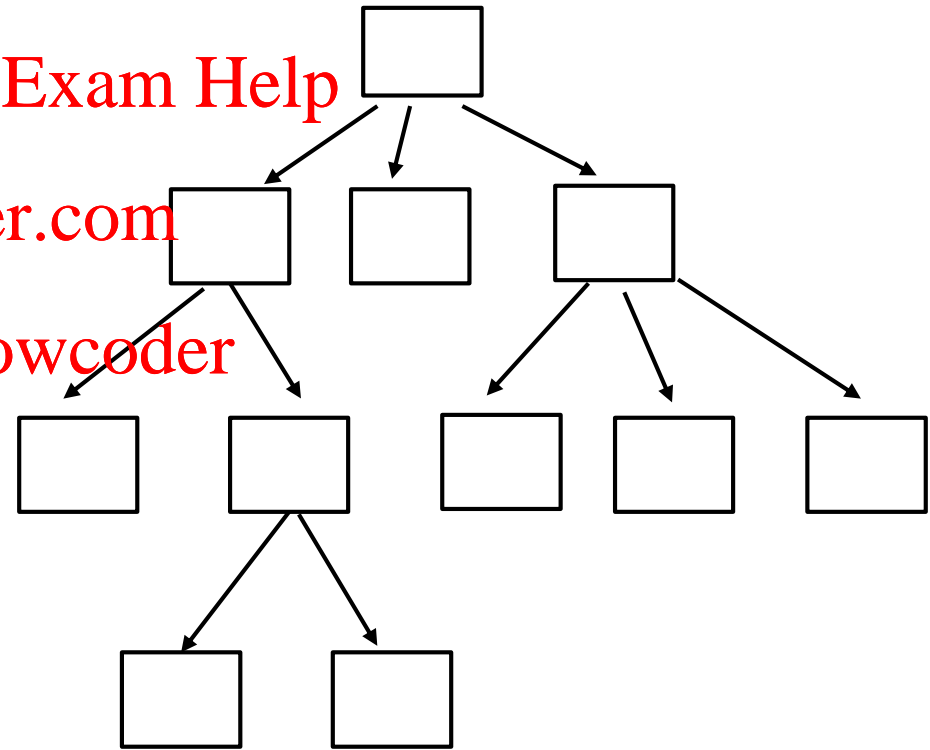
## TREE TRAVERSAL – DEPTH FIRST “PREORDER”

```
depthFirst (root) {  
    if (root is not empty) {  
        visit root  
        for each child of root  
            depthfirst( child )  
    }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



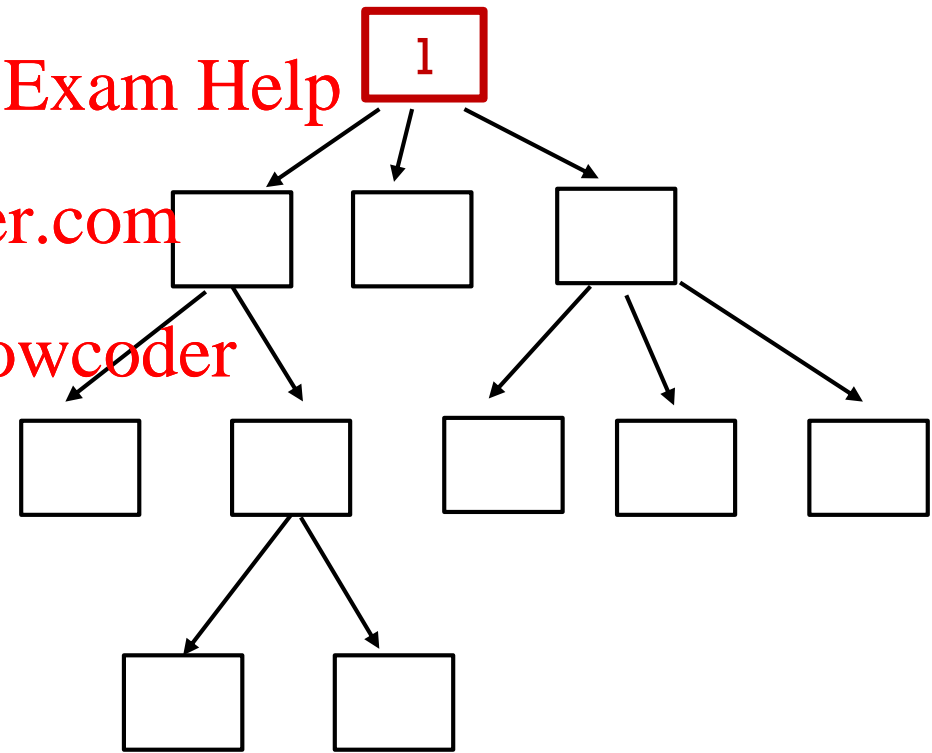
## TREE TRAVERSAL – DEPTH FIRST “PREORDER”

```
depthFirst (root) {  
  if (root is not empty) {  
    visit root  
    for each child of root  
      depthfirst( child )  
  }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



**“preorder” traversal:** visit the root before the children

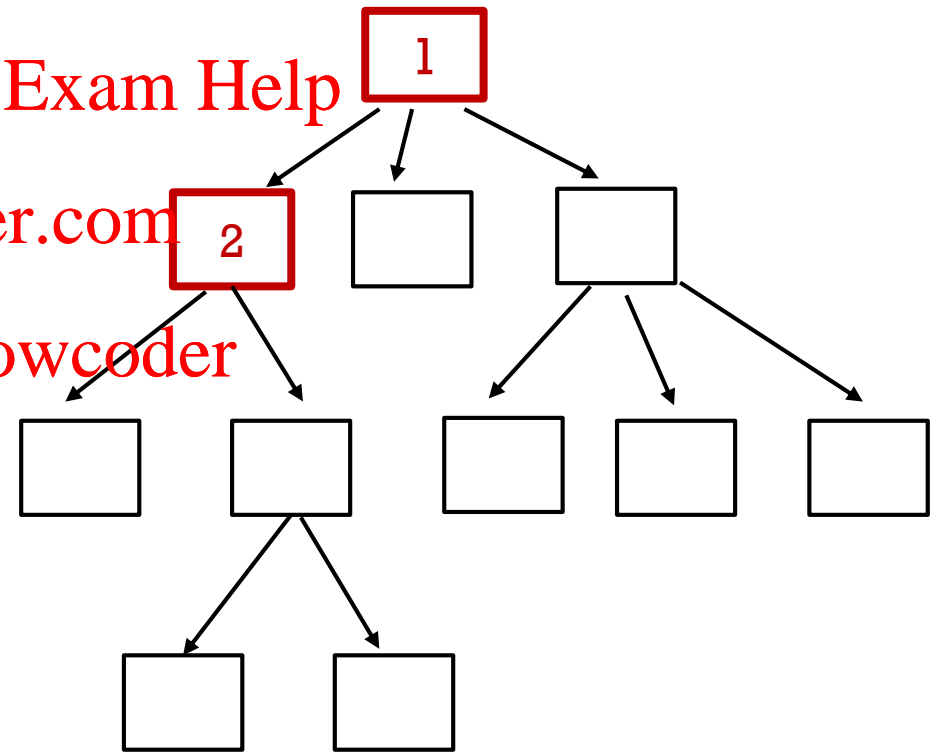
## TREE TRAVERSAL – DEPTH FIRST “PREORDER”

```
depthFirst (root) {  
  if (root is not empty) {  
    visit root  
    for each child of root  
      depthfirst( child )  
  }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



Note that here we are assuming that we iterate through the children nodes from left to right.

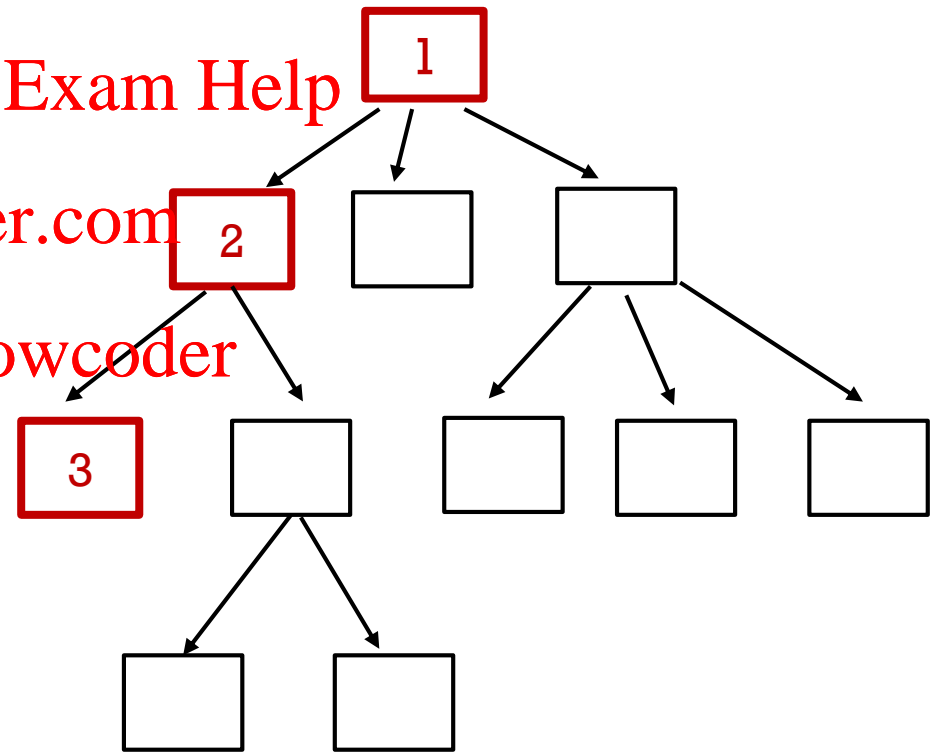
## TREE TRAVERSAL – DEPTH FIRST “PREORDER”

```
depthFirst (root) {  
    if (root is not empty) {  
        visit root  
        for each child of root  
            depthfirst( child )  
    }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder





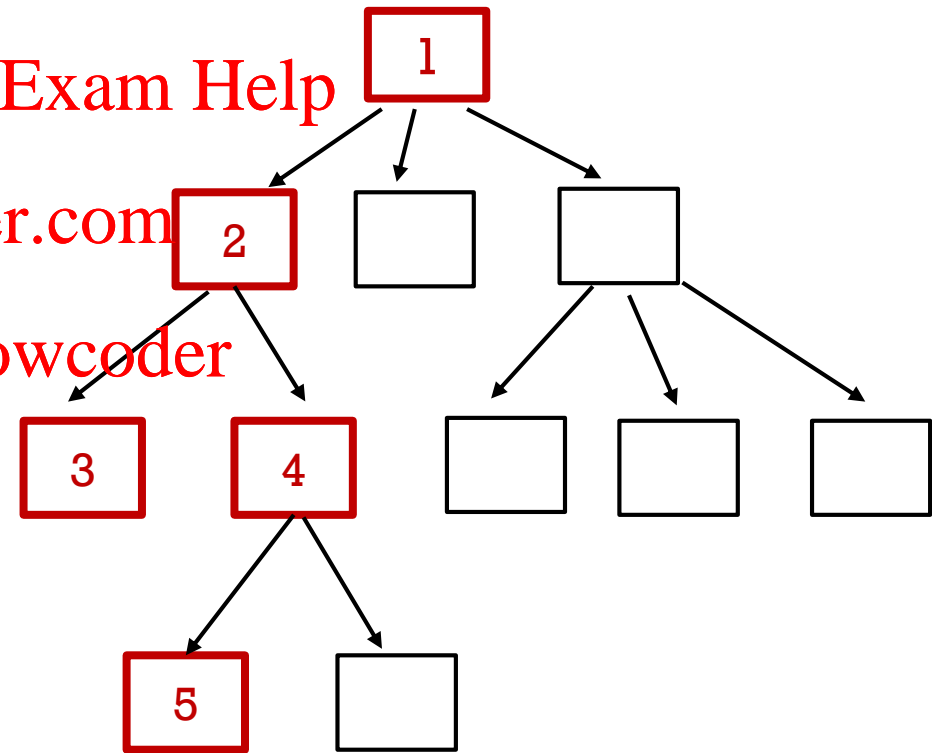
## TREE TRAVERSAL – DEPTH FIRST “PREORDER”

```
depthFirst (root) {  
  if (root is not empty) {  
    visit root  
    for each child of root  
      depthfirst( child )  
  }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



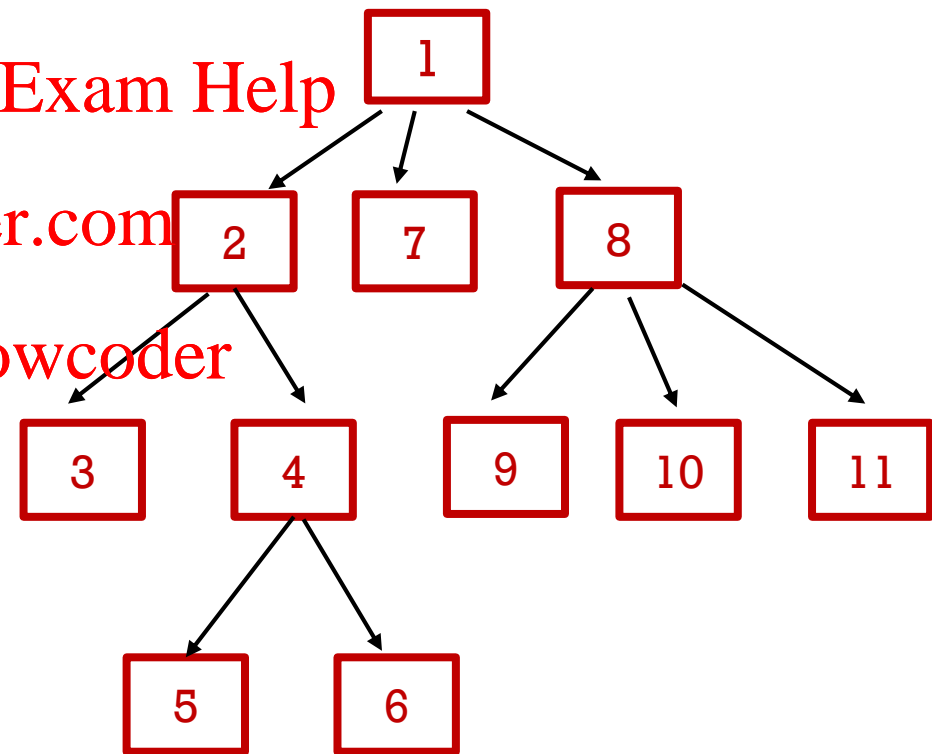
## TREE TRAVERSAL – DEPTH FIRST “PREORDER”

```
depthFirst (root) {  
    if (root is not empty) {  
        visit root  
        for each child of root  
            depthfirst( child )  
    }  
}
```

Assignment Project Exam Help

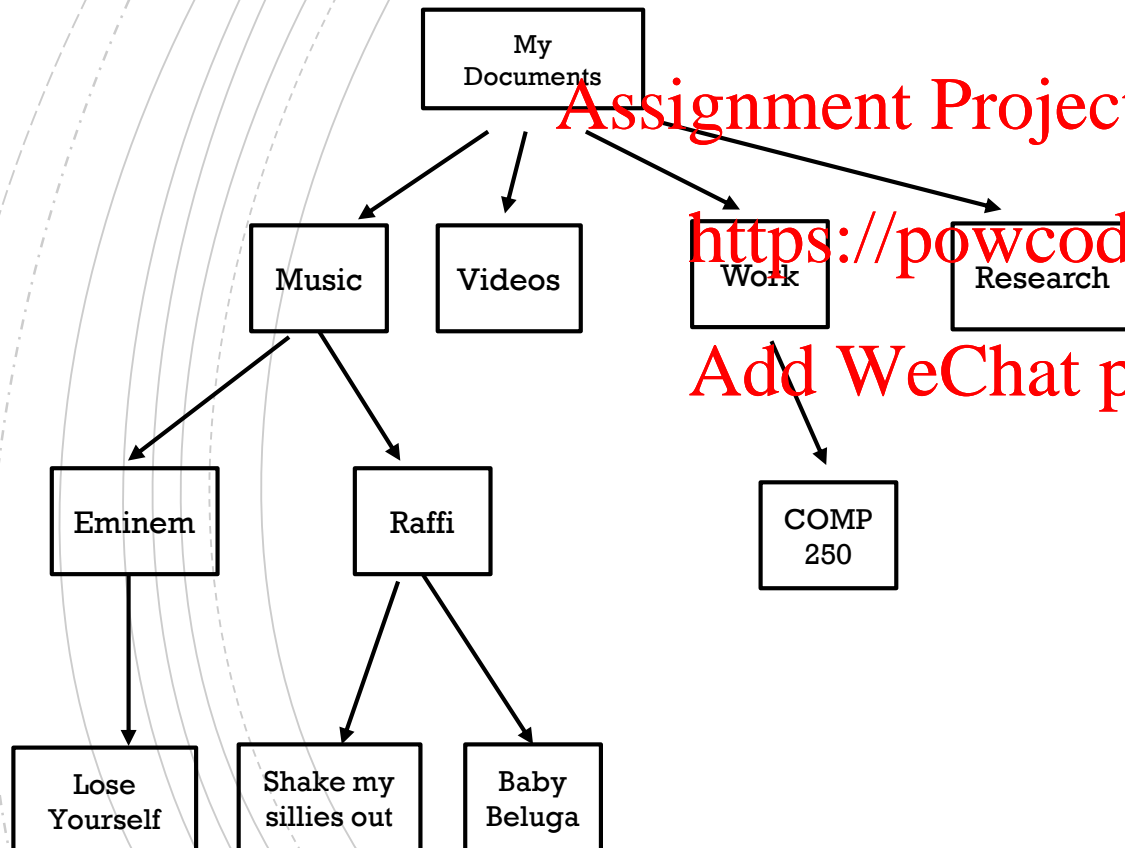
<https://powcoder.com>

Add WeChat powcoder



## EXAMPLE OF USING A PREORDER TRAVERSAL

We would like to print a hierarchical file system  
(visit = print directory or file name)



Documents (directory)  
Music (directory)  
Eminem (directory)  
Lose Yourself (file)  
Raffi (directory)  
Shake My Sillies Out (file)  
Baby Beluga (file)  
Videos (directory)  
:  
Work (directory)  
COMP250 (directory)  
:  
Research (directory)  
:

## “VISIT” A NODE

---

“Visit” implies that you do something at that node.

<https://powcoder.com>

Analogy: you aren't visiting London UK if you just fly through Heathrow.

## TREE TRAVERSAL – DEPTH FIRST “POSTORDER”

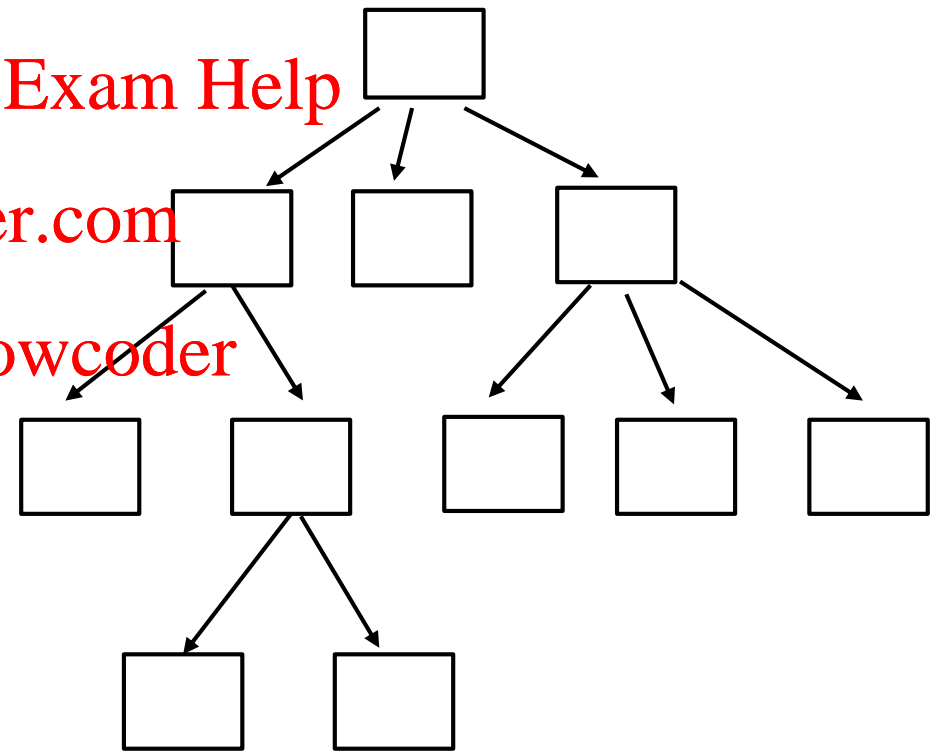
Q: Which node is visited first?

```
depthFirst (root) {  
  if (root is not empty) {  
    for each child of root {  
      depthfirst ( child )  
    }  
    visit root  
  }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



**“postorder” traversal:** visit  
the root after the children

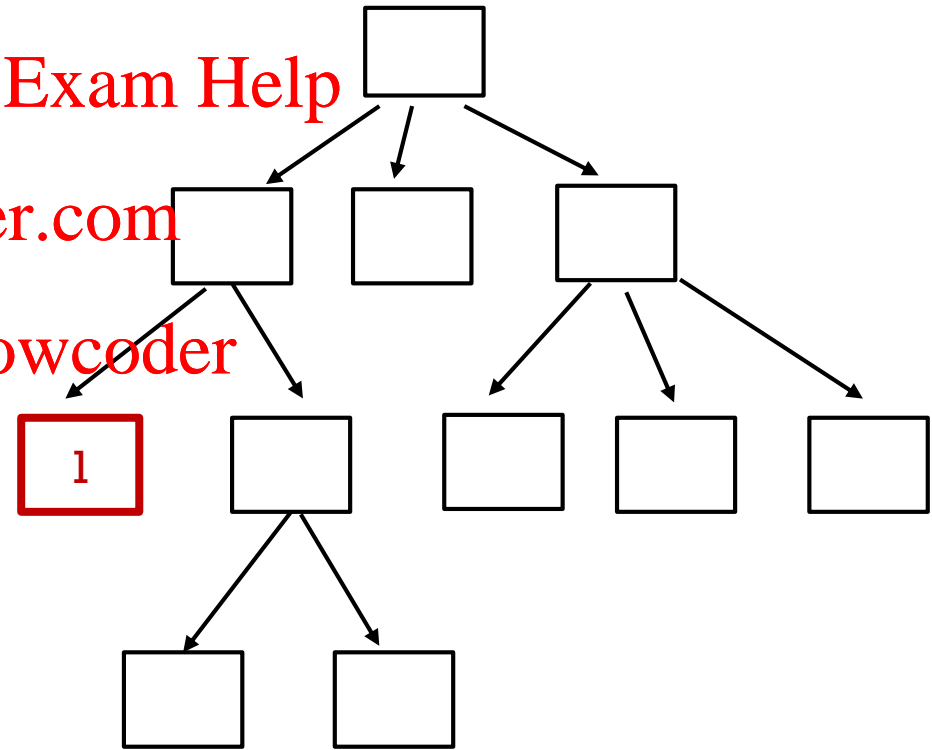
## TREE TRAVERSAL – DEPTH FIRST “POSTORDER”

```
depthFirst (root) {  
  if (root is not empty) {  
    for each child of root  
      depthfirst ( child )  
    visit root  
  }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



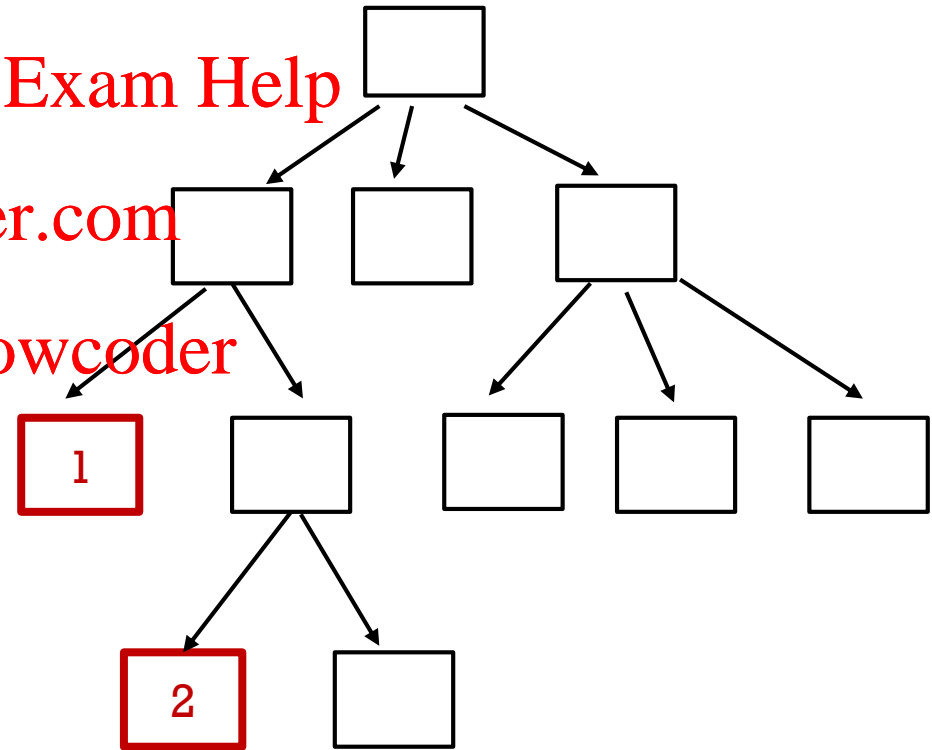
## TREE TRAVERSAL – DEPTH FIRST “POSTORDER”

```
depthFirst (root) {  
    if (root is not empty) {  
        for each child of root  
            depthfirst ( child )  
        visit root  
    }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



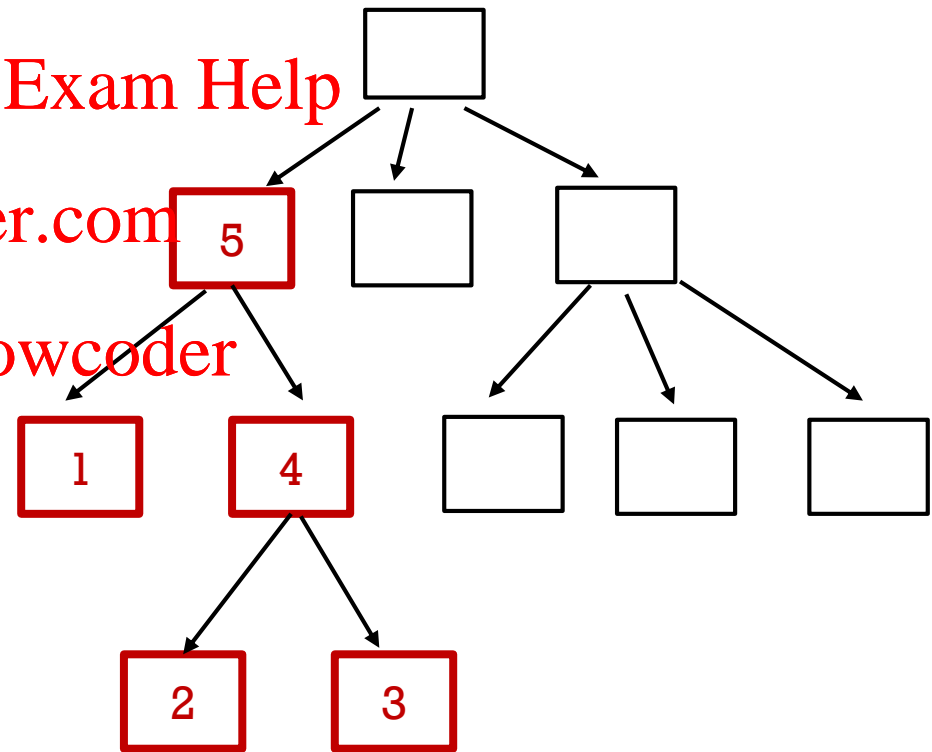
## TREE TRAVERSAL – DEPTH FIRST “POSTORDER”

```
depthFirst (root) {  
  if (root is not empty) {  
    for each child of root  
      depthfirst ( child )  
    visit root  
  }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder





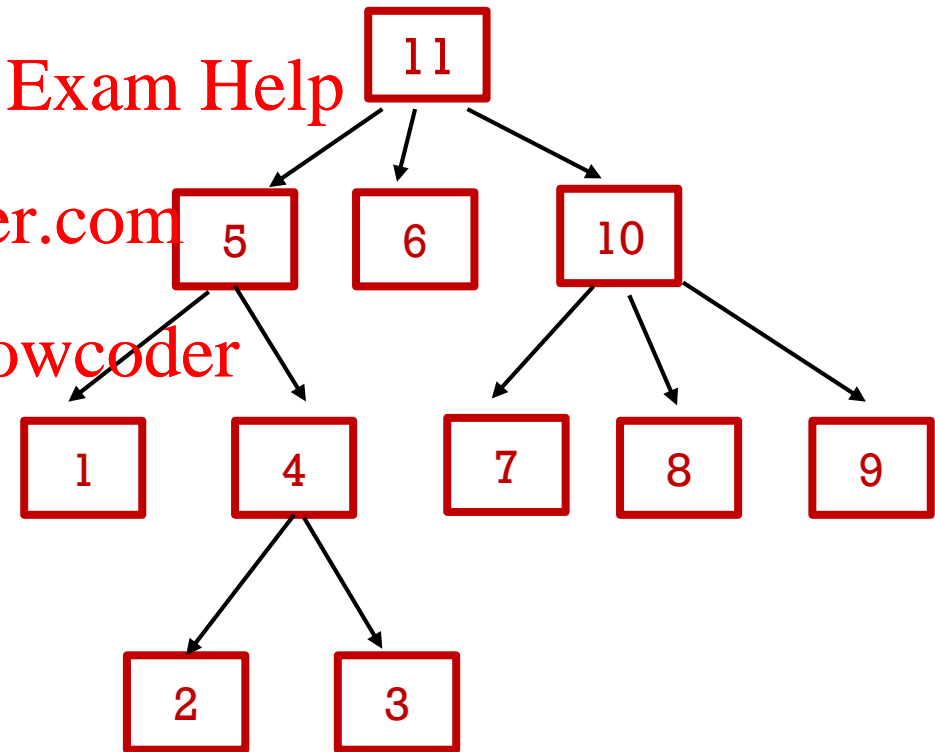
## TREE TRAVERSAL – DEPTH FIRST “POSTORDER”

```
depthFirst (root) {  
    if (root is not empty) {  
        for each child of root  
            depthfirst ( child )  
        visit root  
    }  
}
```

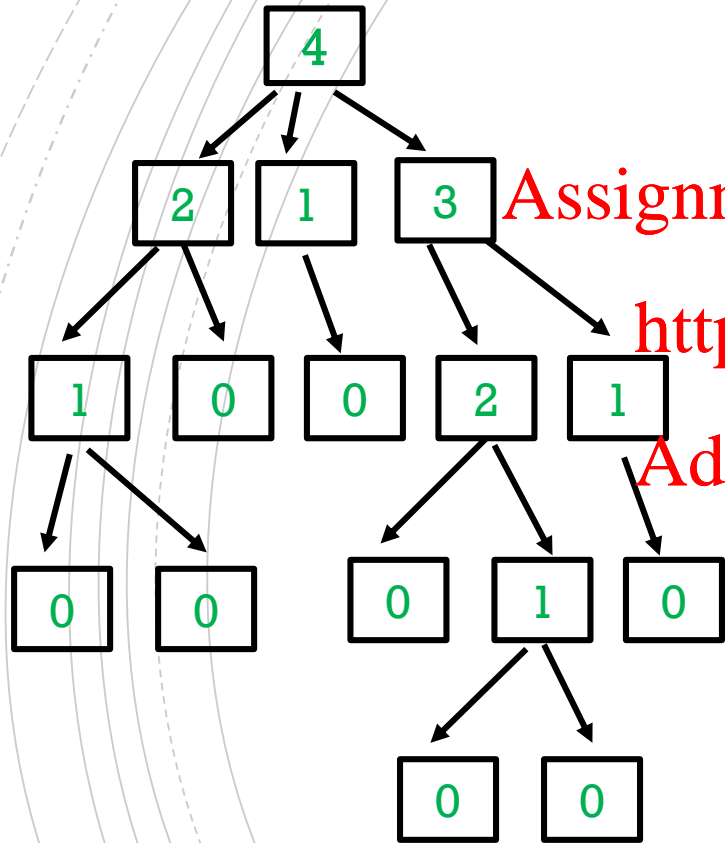
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



## EXAMPLE 1 OF USING A POSTORDER TRAVERSAL: height(v)

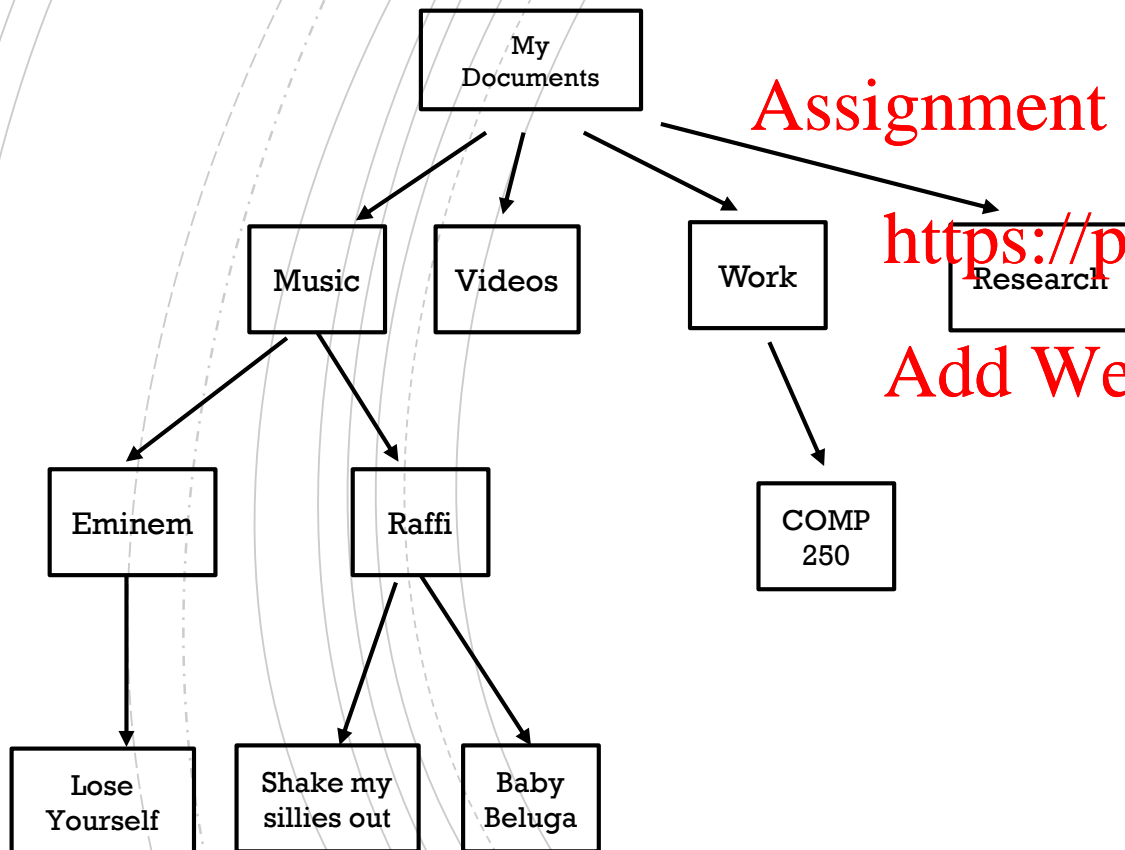


```
height(v) {  
    if (v is a leaf)  
        return 0  
    else  
        h = 0  
        for each child w of v  
            h = max(h, height(w))  
        return 1 + h  
    }  
}
```

visit = return value of height

## EXAMPLE 2 OF USING A POSTORDER TRAVERSAL

What is the total number of bytes in all files in a directory?



Assignment Project Exam Help  
<https://powcoder.com>  
Add WeChat powcoder

```
numBytes(v) {  
    if (v is a leaf)  
        return number of bytes at v  
    else {  
        sum = 0  
        for each child w of v  
            sum += numBytes(w)  
        return sum  
    }  
}
```

visit = determining the number of bytes for a node, e.g. If we were to store 'sum' at the node.

## depthFirst () – PREORDER VS POSTORDER TRAVERSAL

```
depthFirst (root) {  
    if (root is not empty) {  
        visit root  
        for each child of root  
            depthfirst( child )  
    }  
}
```

```
depthFirst (root) {  
    if (root is not empty) {  
        for each child of root  
            depthfirst( child )  
        visit root  
    }  
}
```

Assignment Project Exam Help

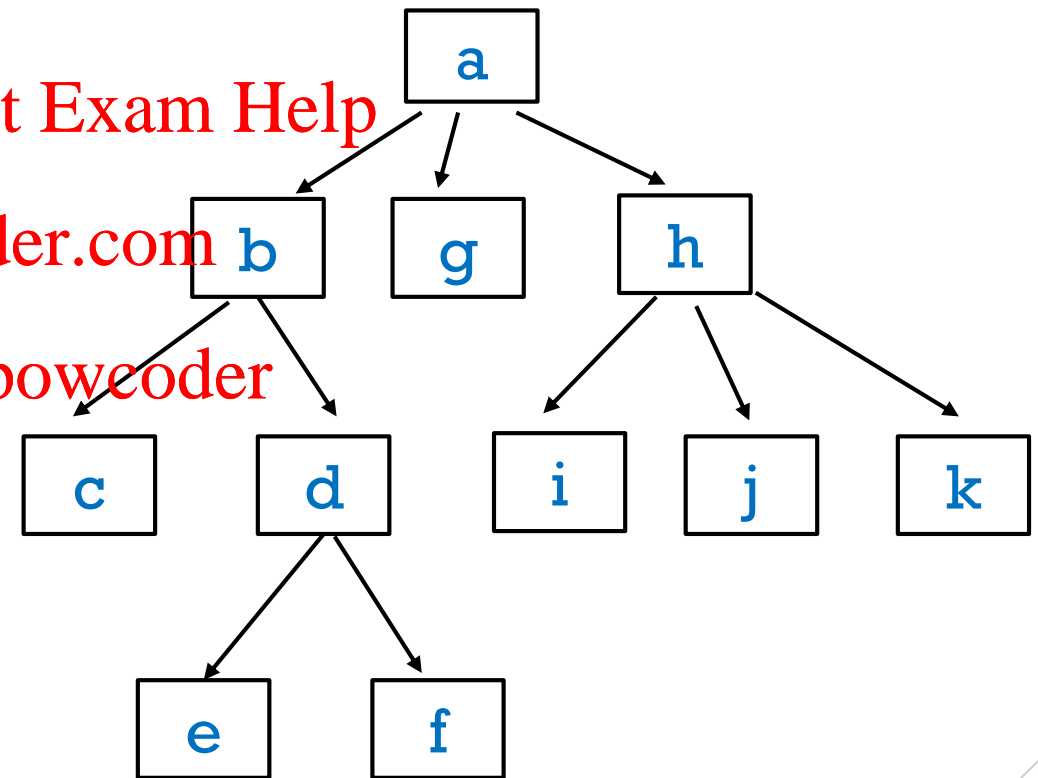
<https://powcoder.com>

Add WeChat powcoder

## CALL SEQUENCE OF `depthFirst()`

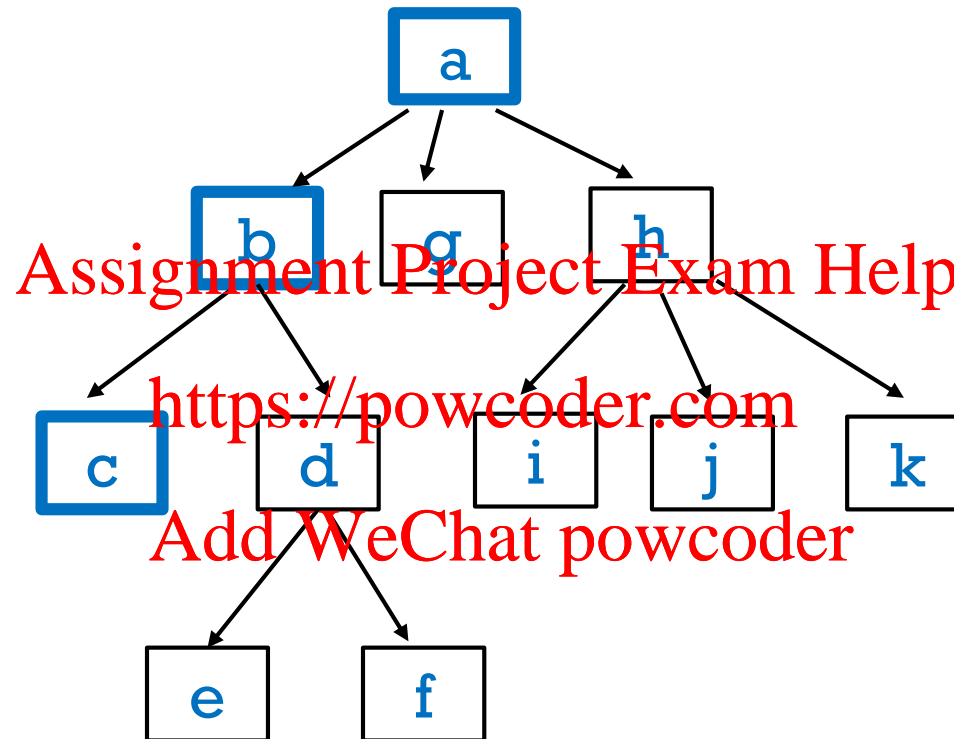
When we call `depthFirst(root)`, the same call sequence occurs for preorder vs postorder implementation.

In the example on the right, the letter order corresponds to `depthFirst(root)` call order.



Note that the call stack stores information about the active method calls in a program.

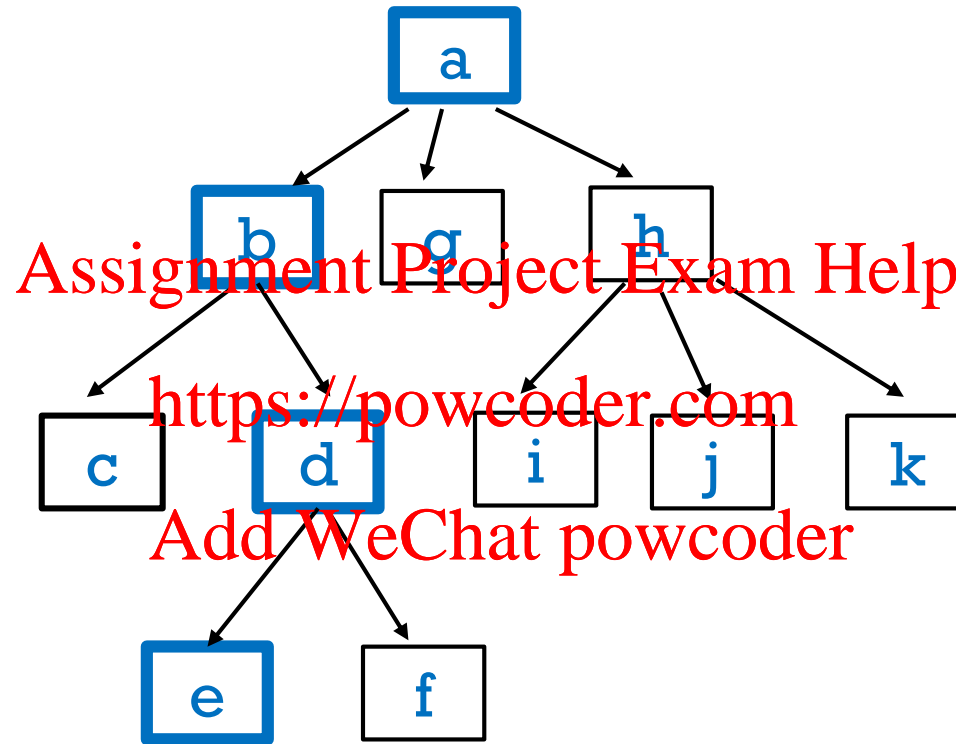
## CALL STACK FOR depthFirst(root)



a a c  
b a b  
c

Note that the call stack stores information about the active method calls in a program.

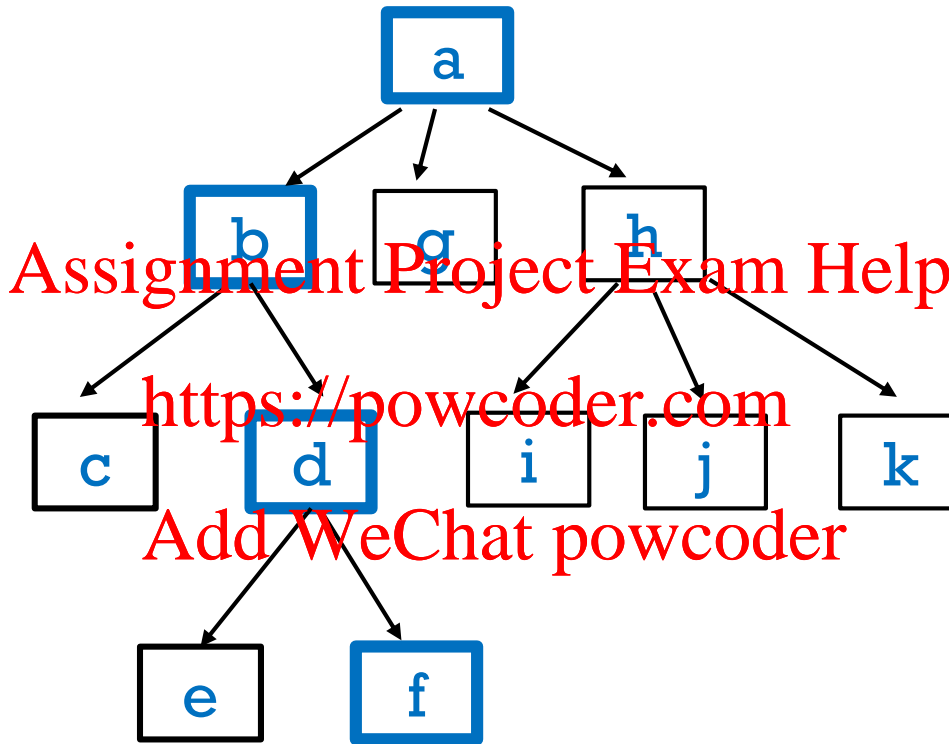
## CALL STACK FOR depthFirst(root)



a a b b c b d b e d  
a a a a a a

Note that the call stack stores information about the active method calls in a program.

## CALL STACK FOR depthFirst(root)

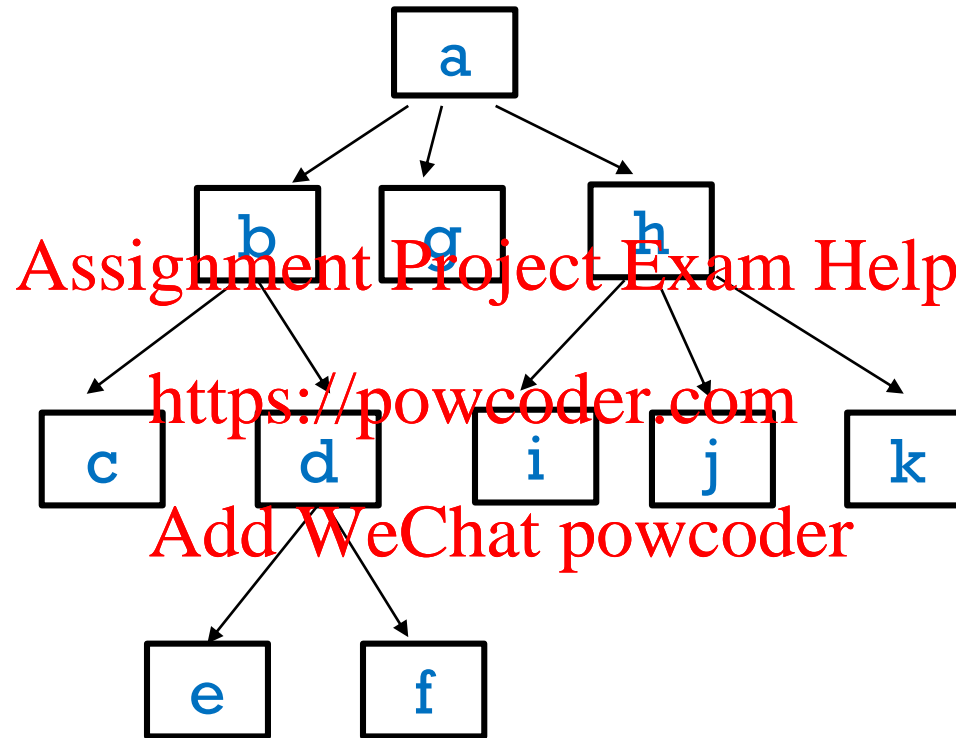


				e	f	
		c	d	d	d	
	b	b	b	b	b	
a	a	a	a	a	a	a



Note that the call stack stores information about the active method calls in a program.

## CALL STACK FOR depthFirst(root)



Notation: the letters indicate the call order of `depthFirst(root)`

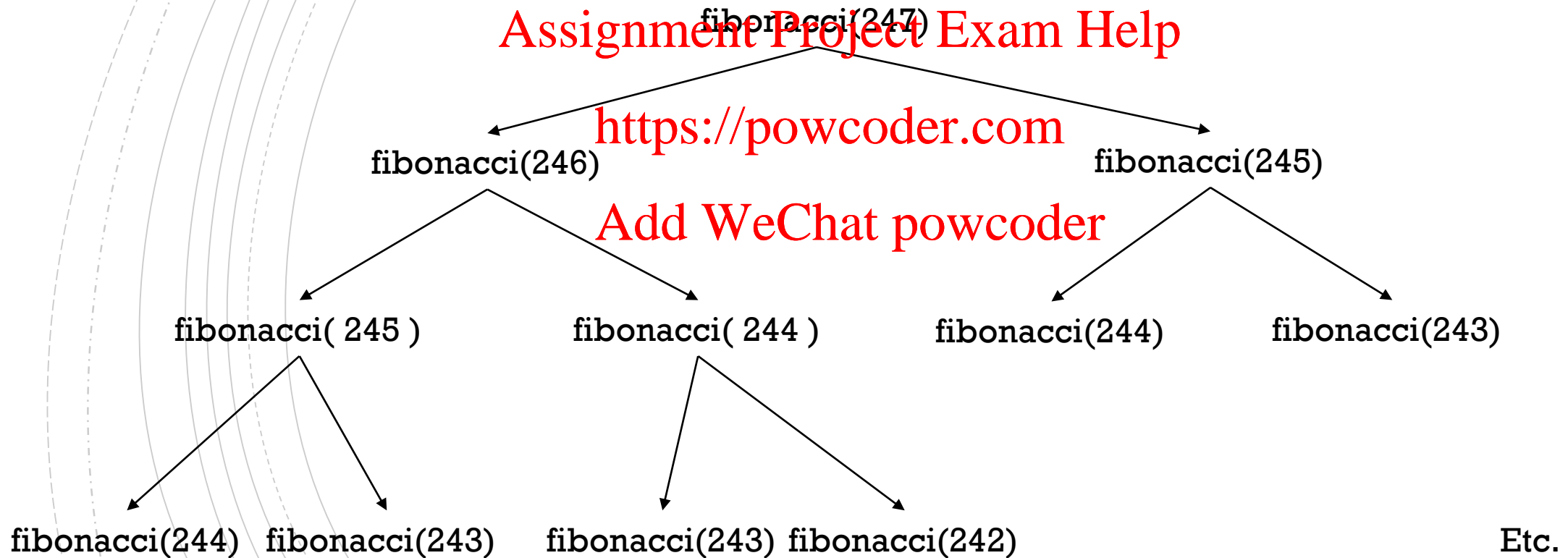
a a b a c b b a d b b e d d f d d i h h j h h k h a

## EXAMPLE

We used a tree to represent the call stack of the recursive Fibonacci method.

```
graph TD; f247["fibonacci(247)"] --> f246["fibonacci(246)"]; f247 --> f245["fibonacci(245)"]; f246 --> f245_246["fibonacci(245)"]; f246 --> f244_246["fibonacci(244)"]; f245_246 --> f244_245["fibonacci(244)"]; f245_246 --> f243_245["fibonacci(243)"]; f244_246 --> f243_246["fibonacci(243)"]; f244_246 --> f242_246["fibonacci(242)"]; f245 --> f244_245["fibonacci(244)"]; f245 --> f243_245["fibonacci(243)"]; f244_245 --> f243_245_2["fibonacci(243)"]; f244_245 --> f242_245["fibonacci(242)"]; f243_245 --> f242_245_2["fibonacci(242)"]; f243_245 --> f241_245["fibonacci(241)"]; f243_245_2 --> f242_245_3["fibonacci(242)"]; f243_245_2 --> f241_245_2["fibonacci(241)"]; f242_245_2 --> f241_245_3["fibonacci(241)"]; f242_245_2 --> f240_245["fibonacci(240)"]; f241_245_2 --> f240_245_2["fibonacci(240)"]; f241_245_3 --> f240_245_3["fibonacci(240)"]; f241_245_3 --> f239_245["fibonacci(239)"]; f240_245_2 --> f239_245_2["fibonacci(239)"]; f240_245_3 --> f239_245_3["fibonacci(239)"]; f239_245_2 --> f238_245["fibonacci(238)"]; f239_245_3 --> f238_245_2["fibonacci(238)"]; f238_245_2 --> f237_245["fibonacci(237)"]; f238_245_3 --> f237_245_2["fibonacci(237)"]; f237_245_2 --> f236_245["fibonacci(236)"]; f237_245_3 --> f236_245_2["fibonacci(236)"]; f236_245_2 --> f235_245["fibonacci(235)"]; f236_245_3 --> f235_245_2["fibonacci(235)"]; f235_245_2 --> f234_245["fibonacci(234)"]; f235_245_3 --> f234_245_2["fibonacci(234)"]; f234_245_2 --> f233_245["fibonacci(233)"]; f234_245_3 --> f233_245_2["fibonacci(233)"]; f233_245_2 --> f232_245["fibonacci(232)"]; f233_245_3 --> f232_245_2["fibonacci(232)"]; f232_245_2 --> f231_245["fibonacci(231)"]; f232_245_3 --> f231_245_2["fibonacci(231)"]; f231_245_2 --> f230_245["fibonacci(230)"]; f231_245_3 --> f230_245_2["fibonacci(230)"]; f230_245_2 --> f229_245["fibonacci(229)"]; f230_245_3 --> f229_245_2["fibonacci(229)"]; f229_245_2 --> f228_245["fibonacci(228)"]; f229_245_3 --> f228_245_2["fibonacci(228)"]; f228_245_2 --> f227_245["fibonacci(227)"]; f228_245_3 --> f227_245_2["fibonacci(227)"]; f227_245_2 --> f226_245["fibonacci(226)"]; f227_245_3 --> f226_245_2["fibonacci(226)"]; f226_245_2 --> f225_245["fibonacci(225)"]; f226_245_3 --> f225_245_2["fibonacci(225)"]; f225_245_2 --> f224_245["fibonacci(224)"]; f225_245_3 --> f224_245_2["fibonacci(224)"]; f224_245_2 --> f223_245["fibonacci(223)"]; f224_245_3 --> f223_245_2["fibonacci(223)"]; f223_245_2 --> f222_245["fibonacci(222)"]; f223_245_3 --> f222_245_2["fibonacci(222)"]; f222_245_2 --> f221_245["fibonacci(221)"]; f222_245_3 --> f221_245_2["fibonacci(221)"]; f221_245_2 --> f220_245["fibonacci(220)"]; f221_245_3 --> f220_245_2["fibonacci(220)"]; f220_245_2 --> f219_245["fibonacci(219)"]; f220_245_3 --> f219_245_2["fibonacci(219)"]; f219_245_2 --> f218_245["fibonacci(218)"]; f219_245_3 --> f218_245_2["fibonacci(218)"]; f218_245_2 --> f217_245["fibonacci(217)"]; f218_245_3 --> f217_245_2["fibonacci(217)"]; f217_245_2 --> f216_245["fibonacci(216)"]; f217_245_3 --> f216_245_2["fibonacci(216)"]; f216_245_2 --> f215_245["fibonacci(215)"]; f216_245_3 --> f215_245_2["fibonacci(215)"]; f215_245_2 --> f214_245["fibonacci(214)"]; f215_245_3 --> f214_245_2["fibonacci(214)"]; f214_245_2 --> f213_245["fibonacci(213)"]; f214_245_3 --> f213_245_2["fibonacci(213)"]; f213_245_2 --> f212_245["fibonacci(212)"]; f213_245_3 --> f212_245_2["fibonacci(212)"]; f212_245_2 --> f211_245["fibonacci(211)"]; f212_245_3 --> f211_245_2["fibonacci(211)"]; f211_245_2 --> f210_245["fibonacci(210)"]; f211_245_3 --> f210_245_2["fibonacci(210)"]; f210_245_2 --> f209_245["fibonacci(209)"]; f210_245_3 --> f209_245_2["fibonacci(209)"]; f209_245_2 --> f208_245["fibonacci(208)"]; f209_245_3 --> f208_245_2["fibonacci(208)"]; f208_245_2 --> f207_245["fibonacci(207)"]; f208_245_3 --> f207_245_2["fibonacci(207)"]; f207_245_2 --> f206_245["fibonacci(206)"]; f207_245_3 --> f206_245_2["fibonacci(206)"]; f206_245_2 --> f205_245["fibonacci(205)"]; f206_245_3 --> f205_245_2["fibonacci(205)"]; f205_245_2 --> f204_245["fibonacci(204)"]; f205_245_3 --> f204_245_2["fibonacci(204)"]; f204_245_2 --> f203_245["fibonacci(203)"]; f204_245_3 --> f203_245_2["fibonacci(203)"]; f203_245_2 --> f202_245["fibonacci(202)"]; f203_245_3 --> f202_245_2["fibonacci(202)"]; f202_245_2 --> f201_245["fibonacci(201)"]; f202_245_3 --> f201_245_2["fibonacci(201)"]; f201_245_2 --> f200_245["fibonacci(200)"]; f201_245_3 --> f200_245_2["fibonacci(200)"]; f200_245_2 --> f199_245["fibonacci(199)"]; f200_245_3 --> f199_245_2["fibonacci(199)"]; f199_245_2 --> f198_245["fibonacci(198)"]; f199_245_3 --> f198_245_2["fibonacci(198)"]; f198_245_2 --> f197_245["fibonacci(197)"]; f198_245_3 --> f197_245_2["fibonacci(197)"]; f197_245_2 --> f196_245["fibonacci(196)"]; f197_245_3 --> f196_245_2["fibonacci(196)"]; f196_245_2 --> f195_245["fibonacci(195)"]; f196_245_3 --> f195_245_2["fibonacci(195)"]; f195_245_2 --> f194_245["fibonacci(194)"]; f195_245_3 --> f194_245_2["fibonacci(194)"]; f194_245_2 --> f193_245["fibonacci(193)"]; f194_245_3 --> f193_245_2["fibonacci(193)"]; f193_245_2 --> f192_245["fibonacci(192)"]; f193_245_3 --> f192_245_2["fibonacci(192)"]; f192_245_2 --> f191_245["fibonacci(191)"]; f192_245_3 --> f191_245_2["fibonacci(191)"]; f191_245_2 --> f190_245["fibonacci(190)"]; f191_245_3 --> f190_245_2["fibonacci(190)"]; f190_245_2 --> f189_245["fibonacci(189)"]; f190_245_3 --> f189_2
```

We used a tree to represent the call stack of the recursive Fibonacci method.



# TREE TRAVERSAL IMPLEMENTATIONS

## Recursive

- depth first (pre- versus post-order)

Assignment Project Exam Help

<https://powcoder.com>

## Non-Recursive

- using a stack
- using a queue

Add WeChat powcoder

## TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
  
    while s is not empty  
    {  
        node = s.pop()  
        visit node  
        if node has left child  
            s.push(left child)  
        if node has right child  
            s.push(right child)  
    }  
}
```

# Assignment Project Exam Help

<https://powcoder.com>

## Add WeChat powcoder

## TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
    while s is not empty {  
        cur = s.pop()  
        visit cur  
    }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

## TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
    while s is not empty {  
        cur = s.pop()  
        visit cur  
        for each child of cur  
            s.push(child)  
    }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

## TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
    while s is not empty {  
        cur = s.pop()  
        visit cur  
        for each child of cur  
            s.push(child)  
    }  
}
```

Assignment Project Exam Help

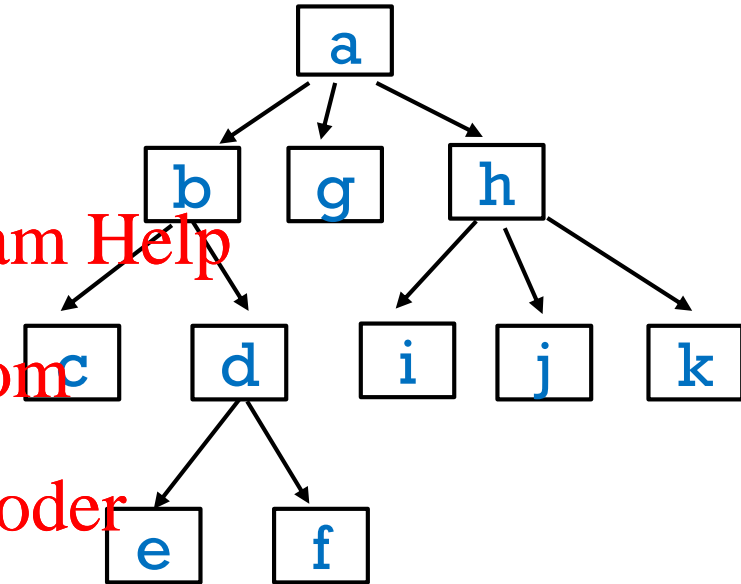
<https://powcoder.com>

Add WeChat powcoder

What is the order in which  
the nodes are visited?

# TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
    while s is not empty {  
        cur = s.pop()  
        visit cur  
        for each child of cur  
            s.push(child)  
    }  
}
```



Assignment Project Exam Help

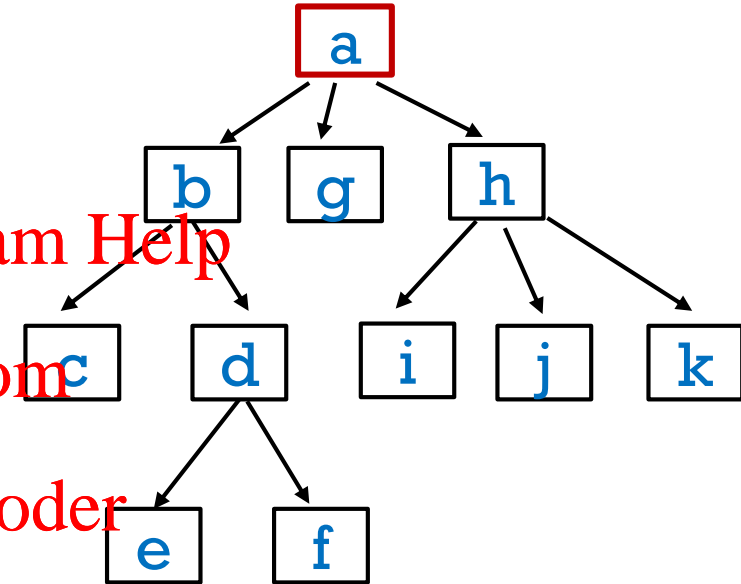
<https://powcoder.com>

Add WeChat powcoder



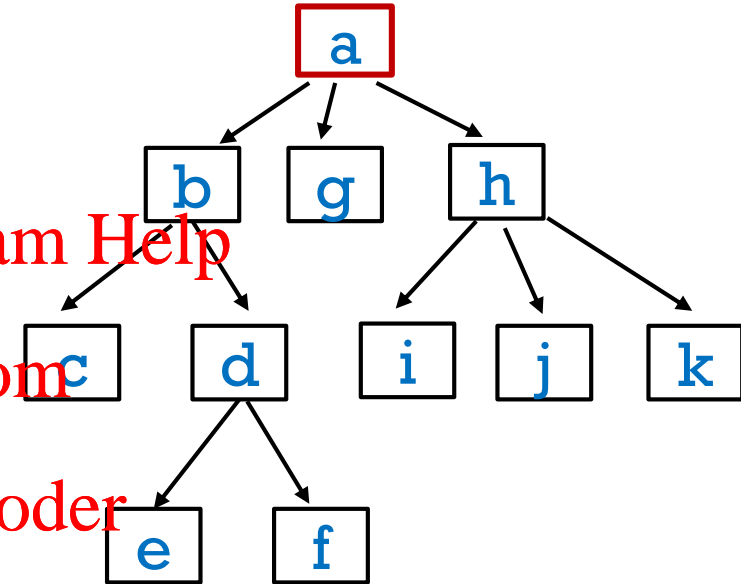
## TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
  initialize empty stack s  
  s.push(root)  
  while s is not empty {  
    cur = s.pop()  
    visit cur  
    for each child of cur  
      s.push(child)  
  }  
}
```



# TREE TRAVERSAL – WITH A STACK

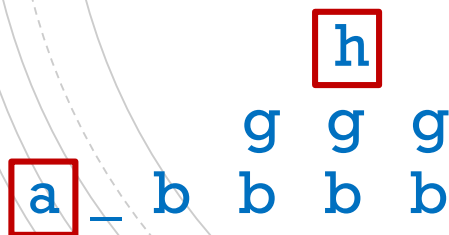
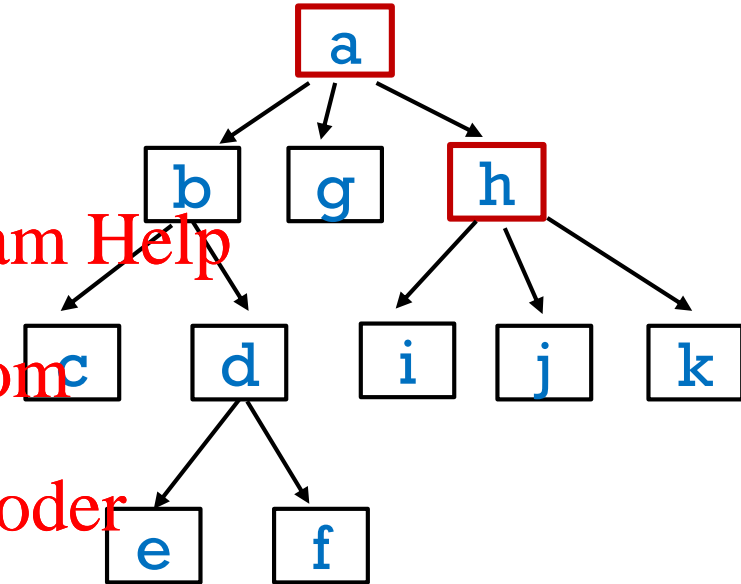
```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
    while s is not empty {  
        cur = s.pop()  
        visit cur  
        for each child of cur  
            s.push(child)  
    }  
}
```



a \_ b b b h g g

# TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
    while s is not empty {  
        cur = s.pop()  
        visit cur  
        for each child of cur  
            s.push(child)  
    }  
}
```



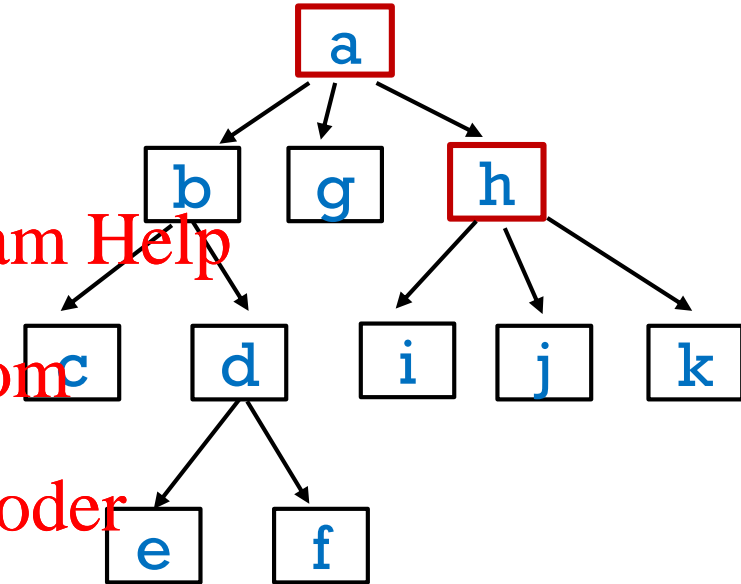
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
    while s is not empty {  
        cur = s.pop()  
        visit cur  
        for each child of cur  
            s.push(child)  
    }  
}
```

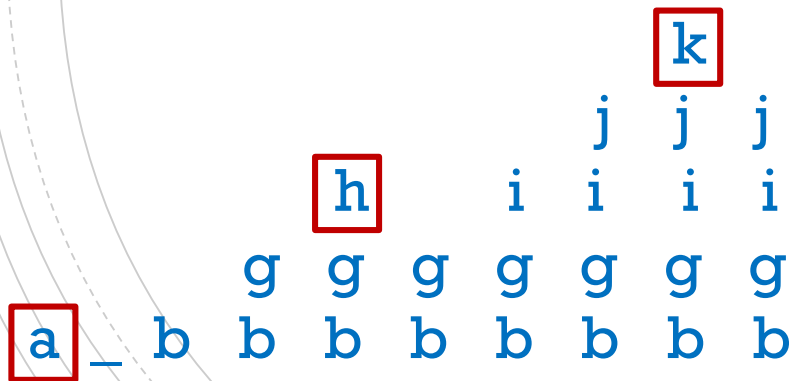
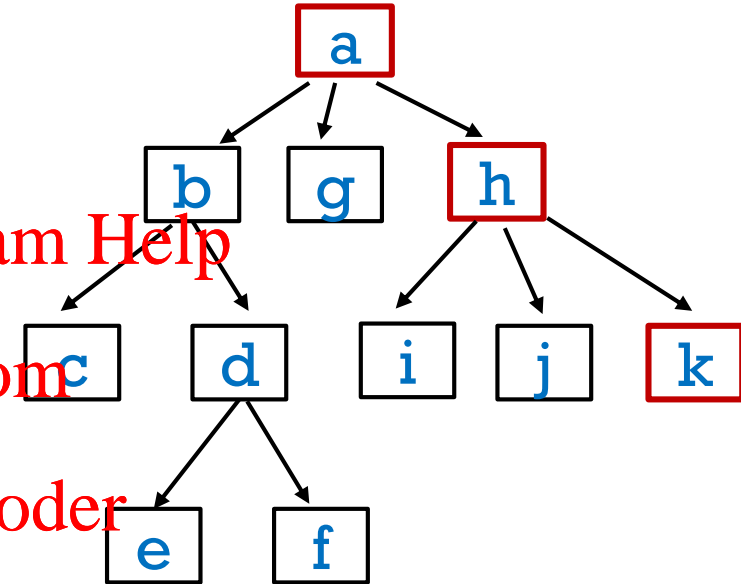


Sequence of nodes visited (root to leaf):

a	b	b	b	b	b	b	k
		g	g	g	g	g	j
			h	i	i	i	j

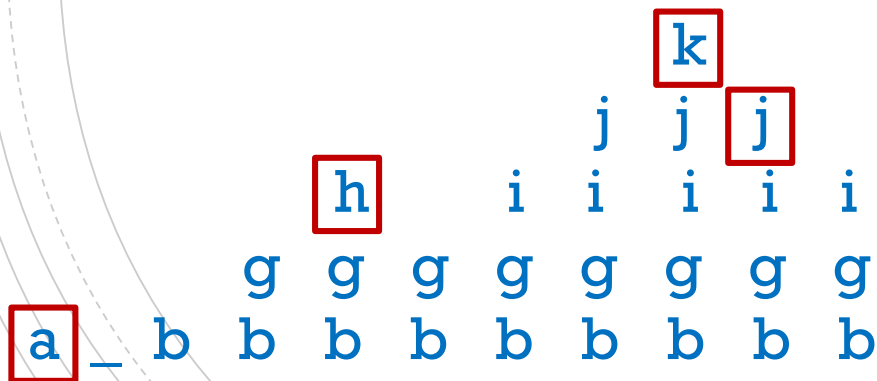
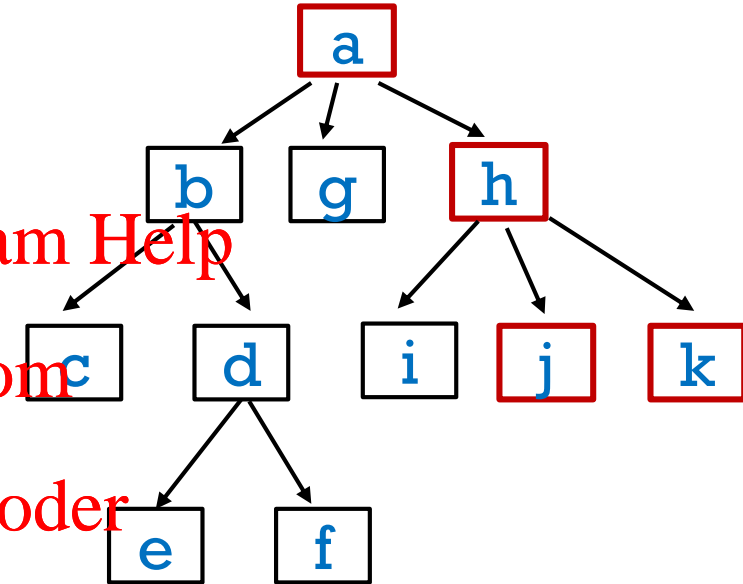
# TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
  initialize empty stack s  
  s.push(root)  
  while s is not empty {  
    cur = s.pop()  
    visit cur  
    for each child of cur  
      s.push(child)  
  }  
}
```



# TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
  initialize empty stack s  
  s.push(root)  
  while s is not empty {  
    cur = s.pop()  
    visit cur  
    for each child of cur  
      s.push(child)  
  }  
}
```



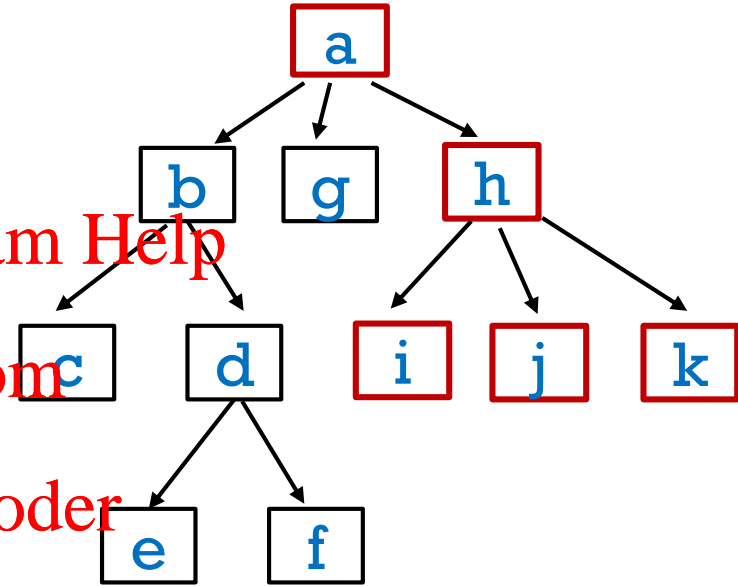
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

## TREE TRAVERSAL – WITH A STACK

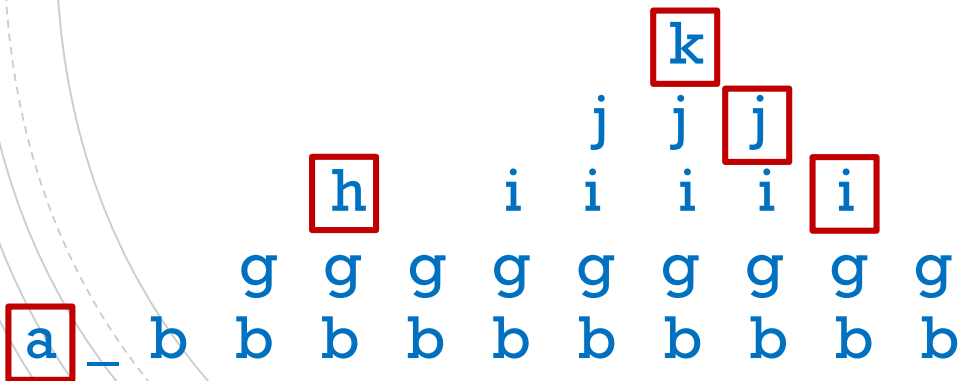
```
treeTraversalUsingStack(root) {
    initialize empty stack s
    s.push(root)
    while s is not empty {
        cur = s.pop()
        visit cur
        for each child of cur
            s.push(child)
    }
}
```



# Assignment Project Exam Help

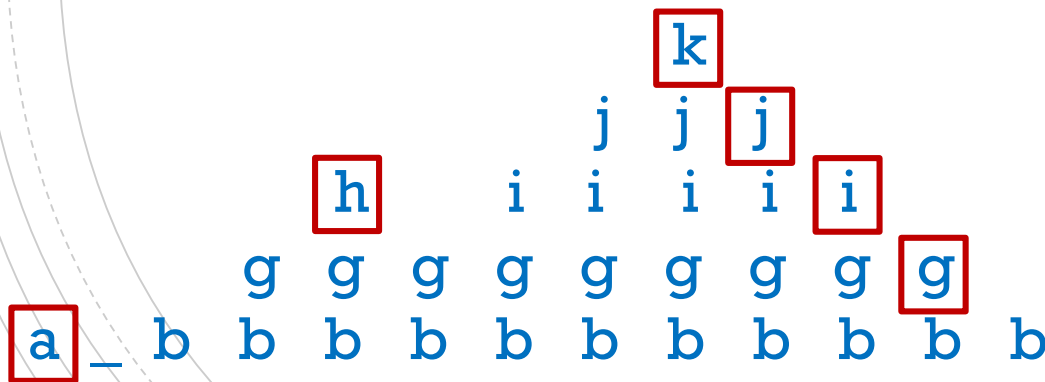
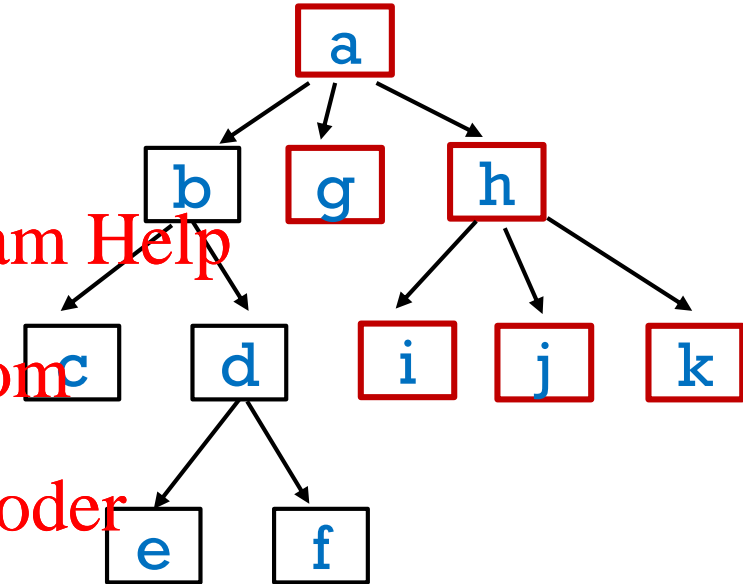
<https://powcoder.com>

## Add WeChat powcoder



# TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
  initialize empty stack s  
  s.push(root)  
  while s is not empty {  
    cur = s.pop()  
    visit cur  
    for each child of cur  
      s.push(child)  
  }  
}
```



Assignment Project Exam Help

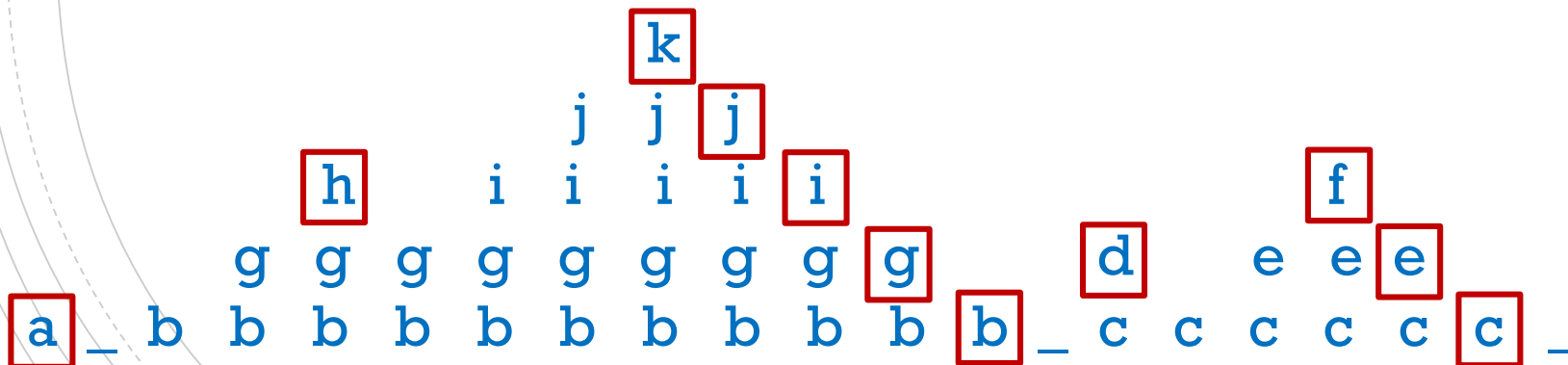
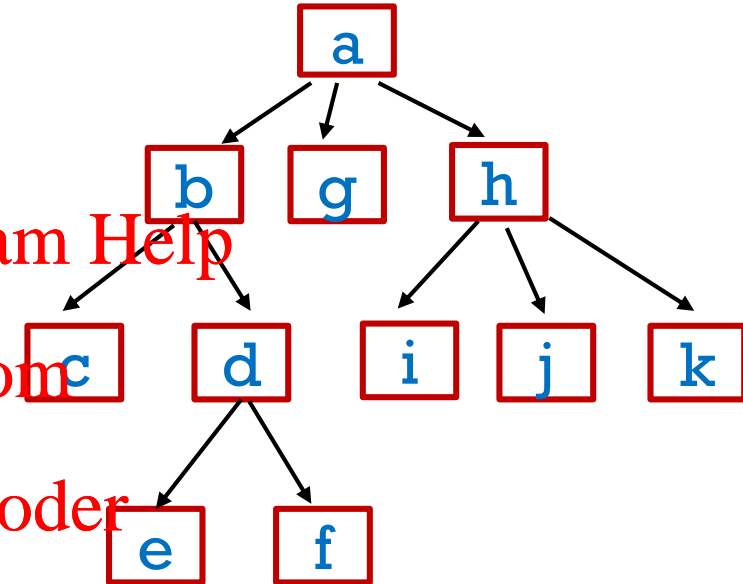
<https://powcoder.com>

Add WeChat powcoder



# TREE TRAVERSAL – WITH A STACK

```
treeTraversalUsingStack(root) {  
  initialize empty stack s  
  s.push(root)  
  while s is not empty {  
    cur = s.pop()  
    visit cur  
    for each child of cur  
      s.push(child)  
  }  
}
```



## TREE TRAVERSAL – WITH A STACK

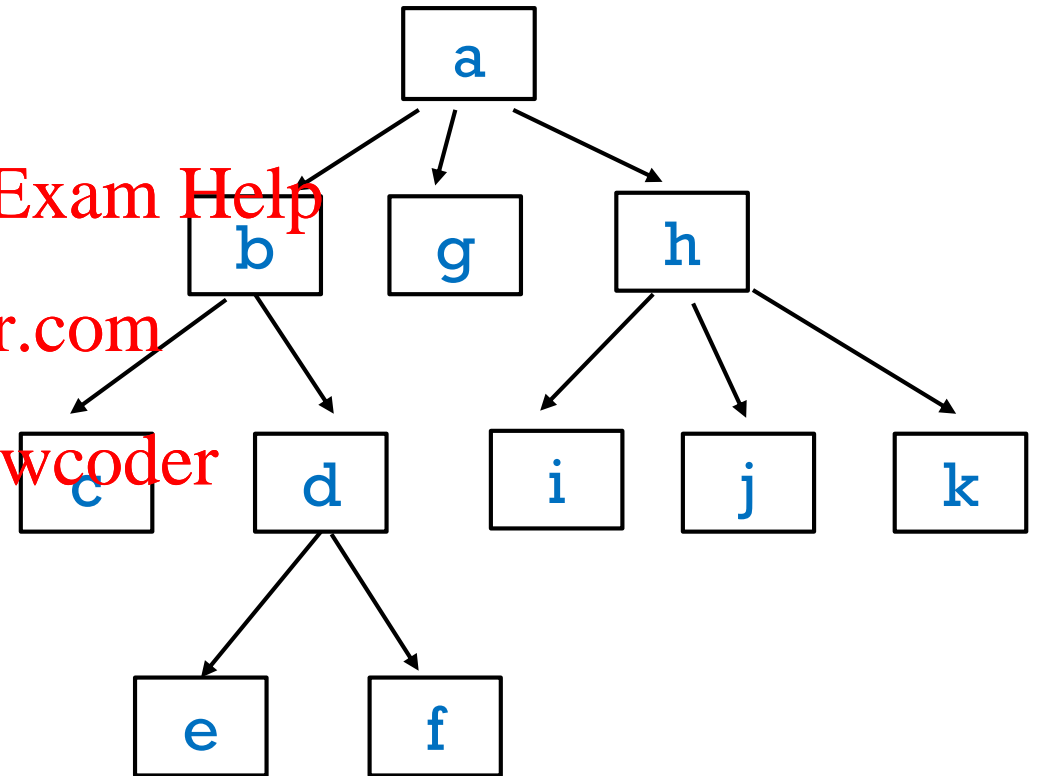
Q: Is it depth first?

A: Yes, but it visits the children “from right to left”

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



Recursive preorder: abcdefghijk

Recursive postorder: cefdbgijkha

Non-recursive (stack): **ahkjigbdfec**

## TREE TRAVERSAL – WITH A STACK

Q: Is it preorder or postorder?

A: It's preorder.

Q: Would move the visit change that?


A: No... why?

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
    while s is not empty {  
        cur = s.pop()  
        visit cur  
        for each child of cur  
            s.push(child)  
    }  
}
```



## WHAT IF WE USED A QUEUE INSTEAD?

```
treeTraversalUsingStack(root) {  
    initialize empty stack s  
    s.push(root)  
    while s is not empty {  
        cur = s.pop()  
        visit cur  
        for each child of cur  
            s.push(child)  
    }  
}
```

```
treeTraversalUsingQueue(root) {  
    initialize empty queue q  
    q.enqueue(root)  
    while q is not empty {  
        cur = q.dequeue()  
        visit cur  
        for each child of cur  
            q.enqueue(child)  
    }  
}
```

Assignment Project Exam Help

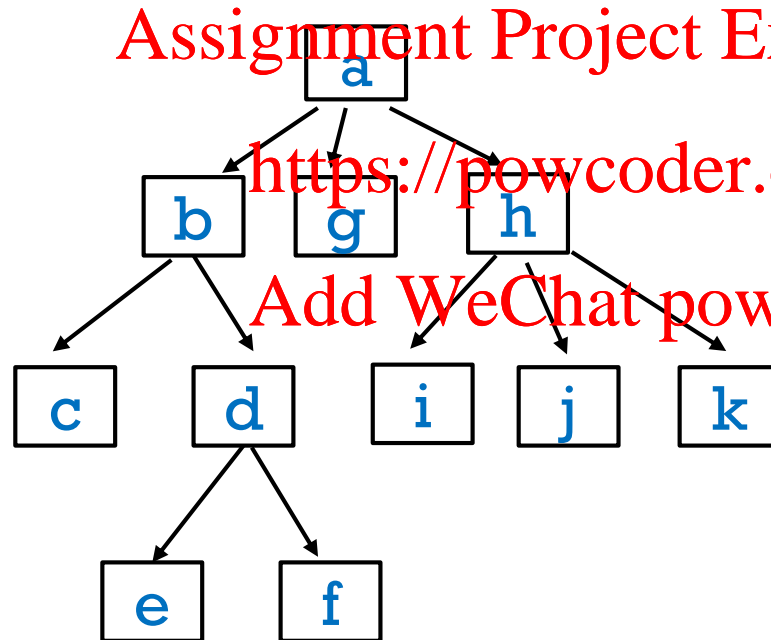
<https://powcoder.com>

Add WeChat powcoder

## WHAT IF WE USED A QUEUE INSTEAD?

Queue state at start of the while loop

a



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

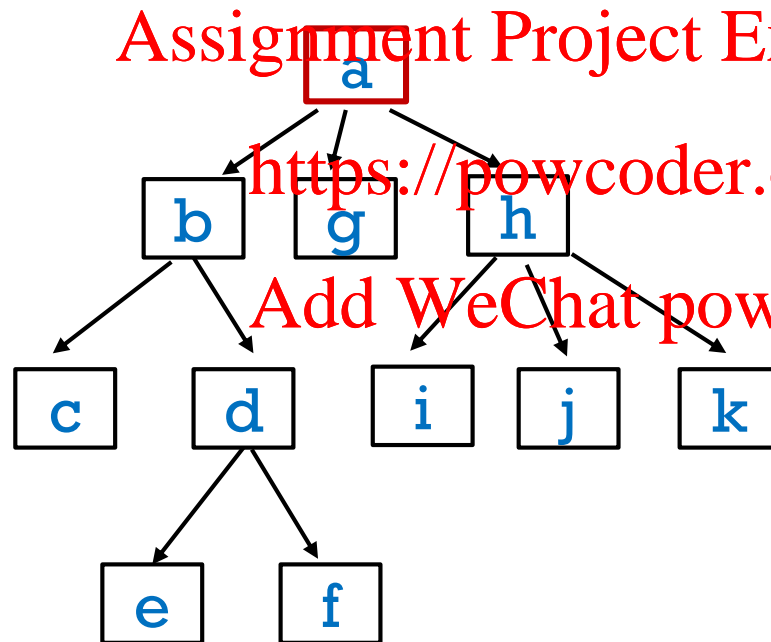
```
preorderTraversalUsingQueue(root) {  
    initialize empty queue q  
    q.enqueue(root)  
    while s is not empty {  
        cur = q.dequeue()  
        visit cur  
        for each child of cur  
            q.enqueue(child)  
    }  
}
```

## WHAT IF WE USED A QUEUE INSTEAD?

Queue state at start of the while loop

a

b g h



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
preOrderTraversalUsingQueue(root) {  
    initialize empty queue q  
    q.enqueue(root)  
    while s is not empty {  
        cur = q.dequeue()  
        visit cur  
        for each child of cur  
            q.enqueue(child)  
    }  
}
```

## WHAT IF WE USED A QUEUE INSTEAD?

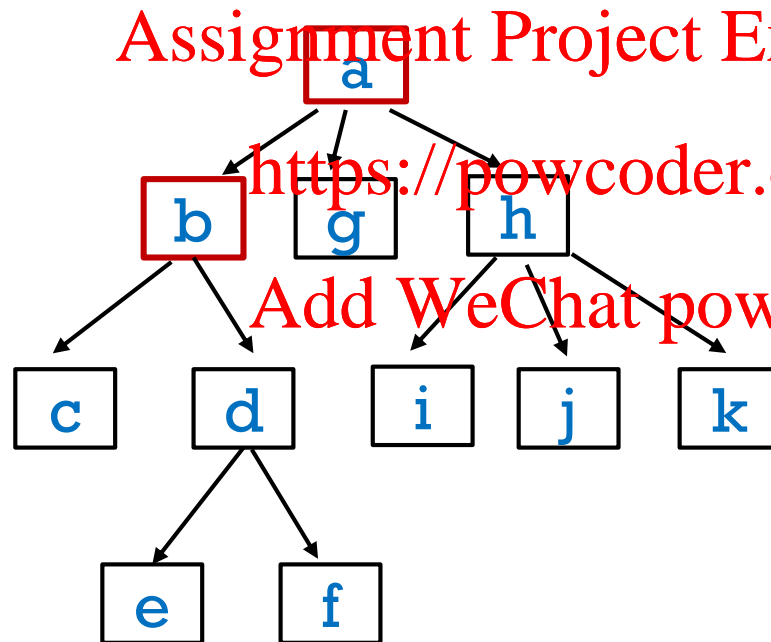
Queue state at start of the while loop

a

b g h

g h

g h c d



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
preOrderTraversalUsingQueue(root) {  
    initialize empty queue q  
    q.enqueue(root)  
    while s is not empty {  
        cur = q.dequeue()  
        visit cur  
        for each child of cur  
            q.enqueue(child)  
    }  
}
```

## WHAT IF WE USED A QUEUE INSTEAD?

Queue state at start of the while loop

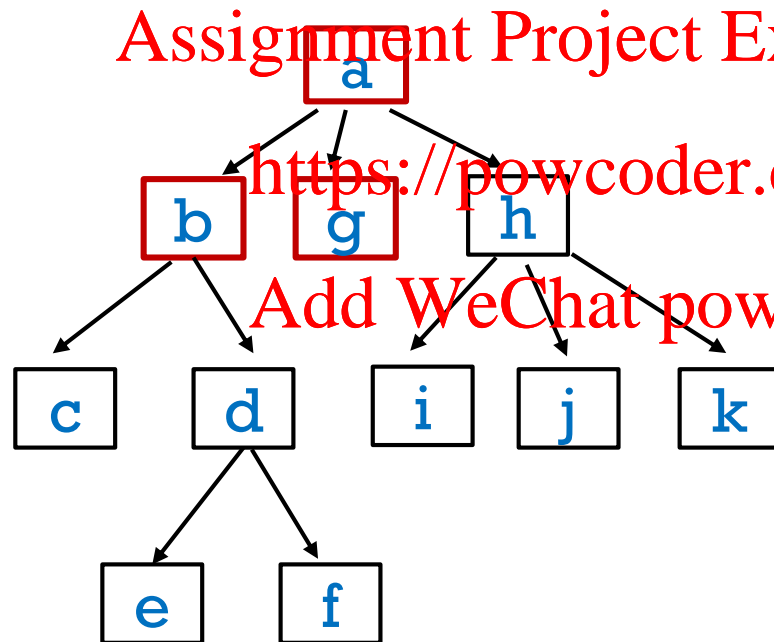
a

b g h

g h

g h c d

h c d



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
preOrderTraversalUsingQueue(root) {  
    initialize empty queue q  
    q.enqueue(root)  
    while q is not empty {  
        cur = q.dequeue()  
        visit cur  
        for each child of cur  
            q.enqueue(child)  
    }  
}
```



## WHAT IF WE USED A QUEUE INSTEAD?

Queue state at start of the while loop

a

b g h

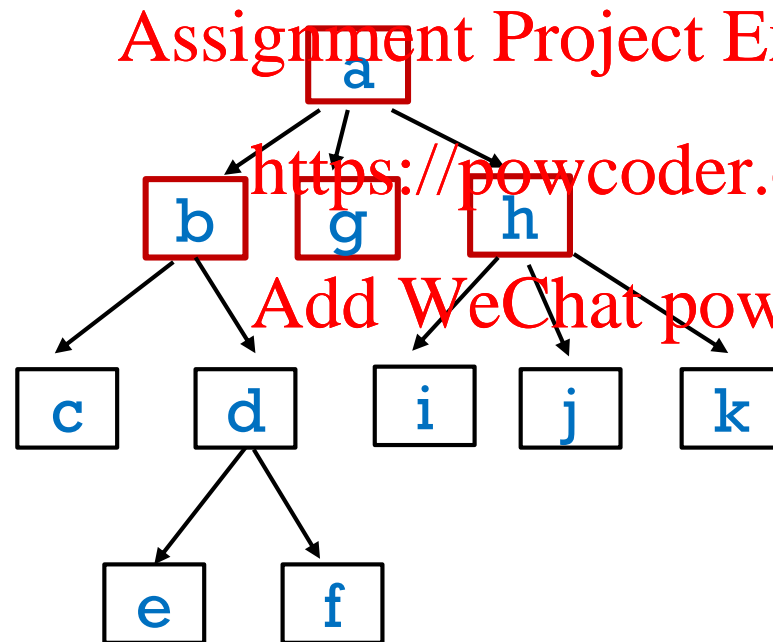
g h

g h c d

h c d

c d

c d i j k



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
preOrderTraversalUsingQueue(root) {  
    initialize empty queue q  
    q.enqueue(root)  
    while s is not empty {  
        cur = q.dequeue()  
        visit cur  
        for each child of cur  
            q.enqueue(child)  
    }  
}
```

## WHAT IF WE USED A QUEUE INSTEAD?

Queue state at start of the while loop

a

b g h

g h

g h c d

h c d

c d

c d i j k

d i j k

i j k

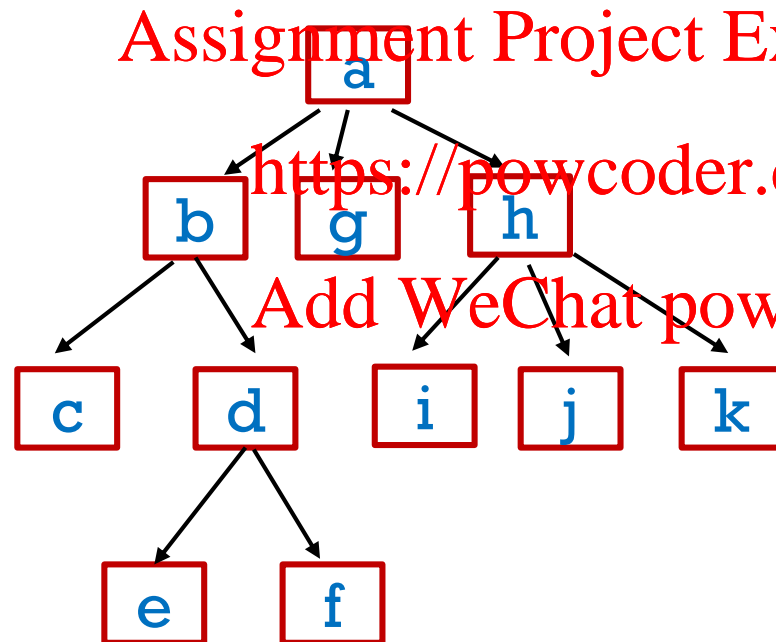
i j k e f

j k e f

k e f

e f

f



```
preOrderTraversalUsingQueue(root) {  
    initialize empty queue q  
    q.enqueue(root)  
    while s is not empty {  
        cur = q.dequeue()  
        visit cur  
        for each child of cur  
            q.enqueue(child)  
    }  
}
```

# BREADTH FIRST TRAVERSAL

*For each level  $i$*

*visit all nodes at level  $i$*

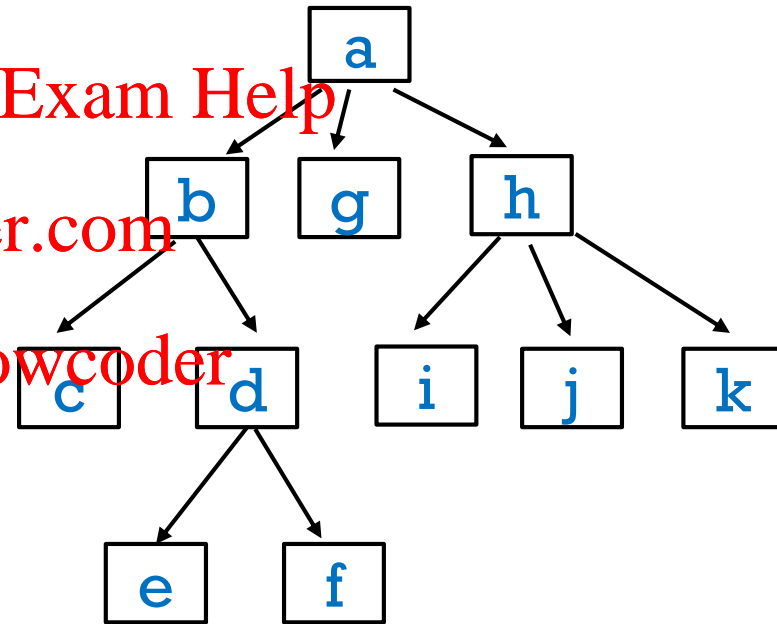
Order visited:

abghcdijkef

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



## IMPLEMENTATION DETAILS

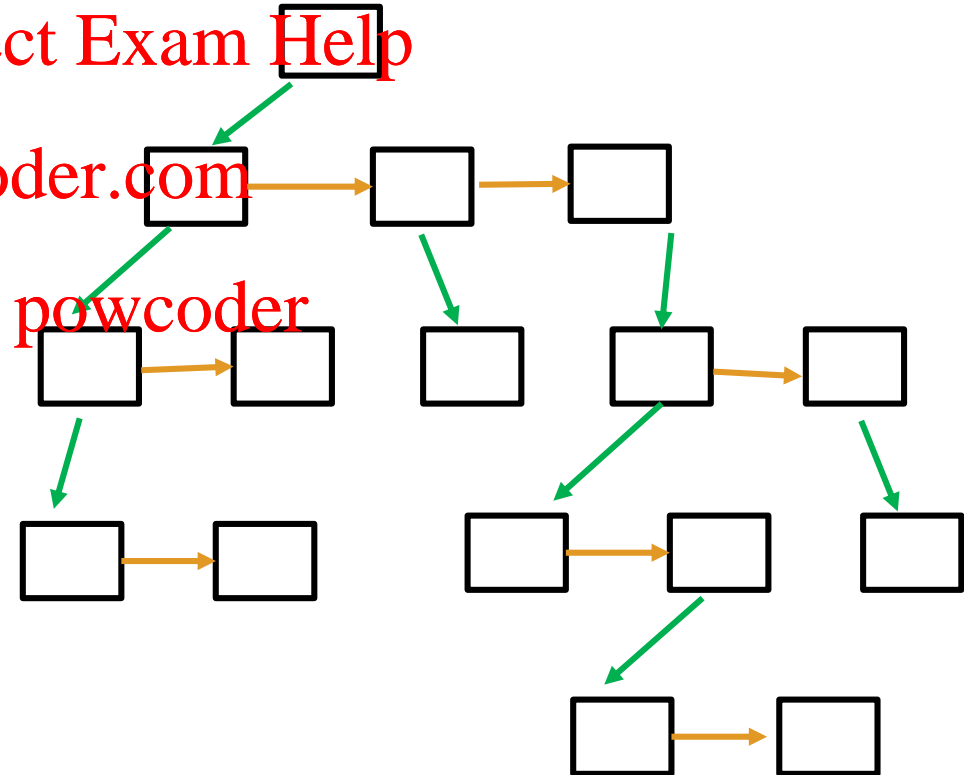
Recall the “**first child**, **next sibling**” implementation

```
class Tree<T>{  
    TreeNode<T> root;  
    :  
  
    class TreeNode<T>{  
        T element;  
        TreeNode<T> firstChild;  
        TreeNode<T> nextSibling;  
        :  
    }  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



## IMPLEMENTATION DETAILS

Recall the “**first child**, **next sibling**” implementation

Then when we write

```
for each child {  
    :  
}
```

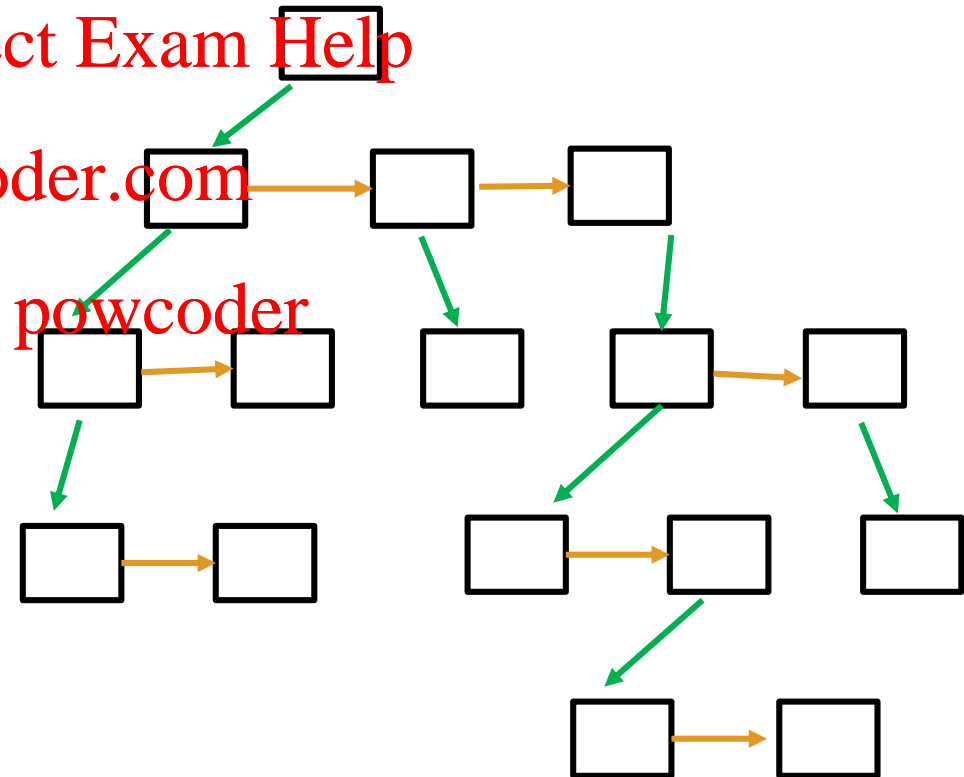
*it means*

```
child = cur.firstChild  
while(child !=null) {  
    :  
    child = child.nextSibling  
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder





# Coming Soon

**Assignment Project Exam Help**

**In the next video:**

■ **Binary Trees**

<https://powcoder.com>

**Add WeChat powcoder**