

COMP 250

INTRODUCTION TO COMPUTER SCIENCE

Assignment Project Exam Help

<https://powcoder.com>

Week 12-1: Binary Search Trees

Add WeChat: powcoder

Giulia Alberini, Fall 2020

Slides adapted from Michael Langer's

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Binary Search Trees Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help

BINARY SEARCH TREES

<https://powcoder.com>

Add WeChat powcoder

BSTNode

- The keys are “comparable” $<, =, >$
e.g. numbers, strings.

Assignment Project Exam Help

<https://powcoder.com>

```
class BSTNode<K>{  
    K key;  
    BSTNode<K> leftchild;  
    BSTNode<K> rightchild;  
    :  
}
```

Add WeChat powcoder

BINARY SEARCH TREE DEFINITION

- binary tree

Assignment Project Exam Help

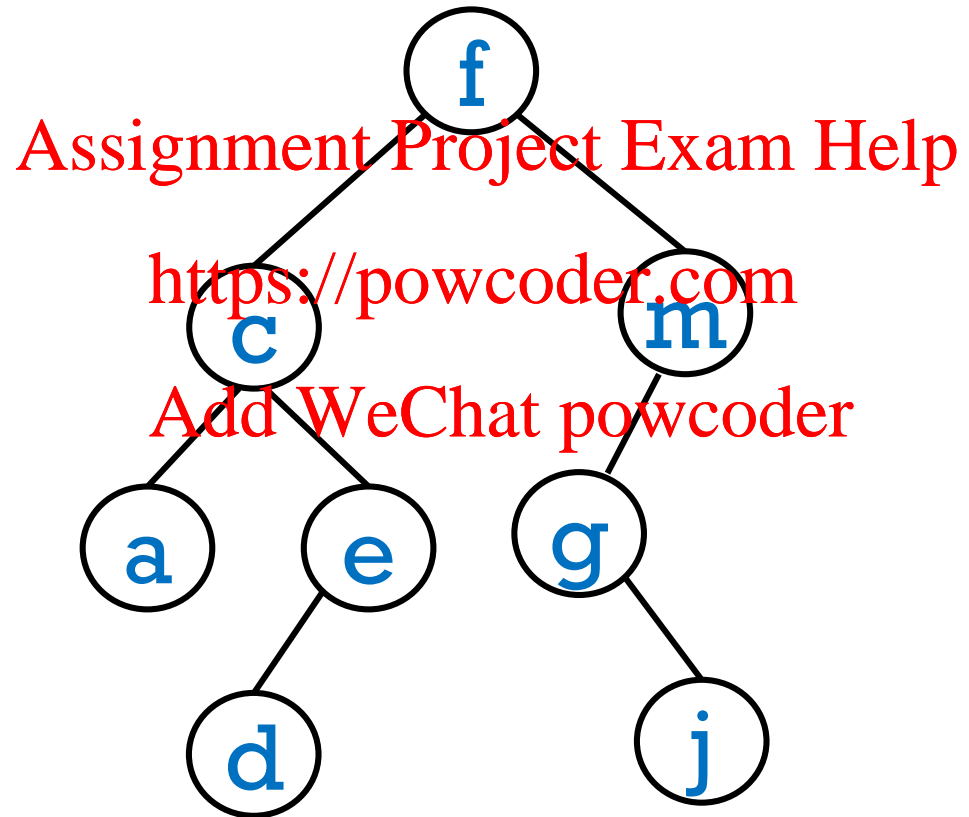
- keys are comparable, and unique (no duplicates)

<https://powcoder.com>

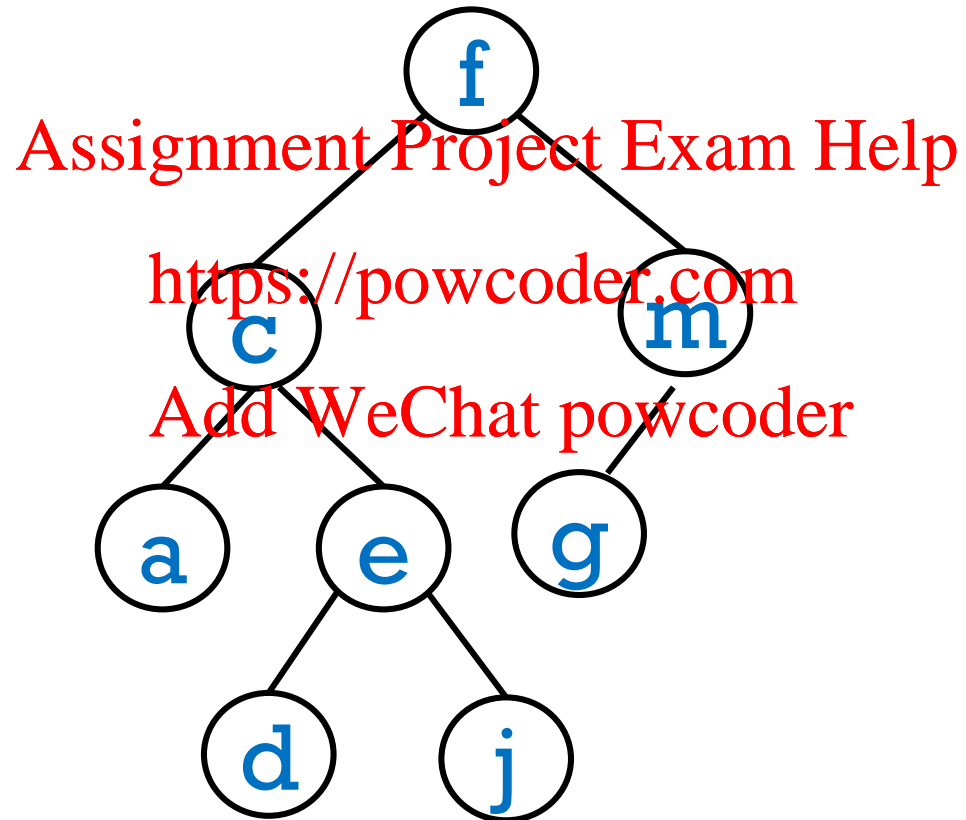
Add WeChat powcoder

- for each node, all descendents in left subtree are less than the node, and all descendents in the node's right subtree are greater than the node
(comparison is based on node key)

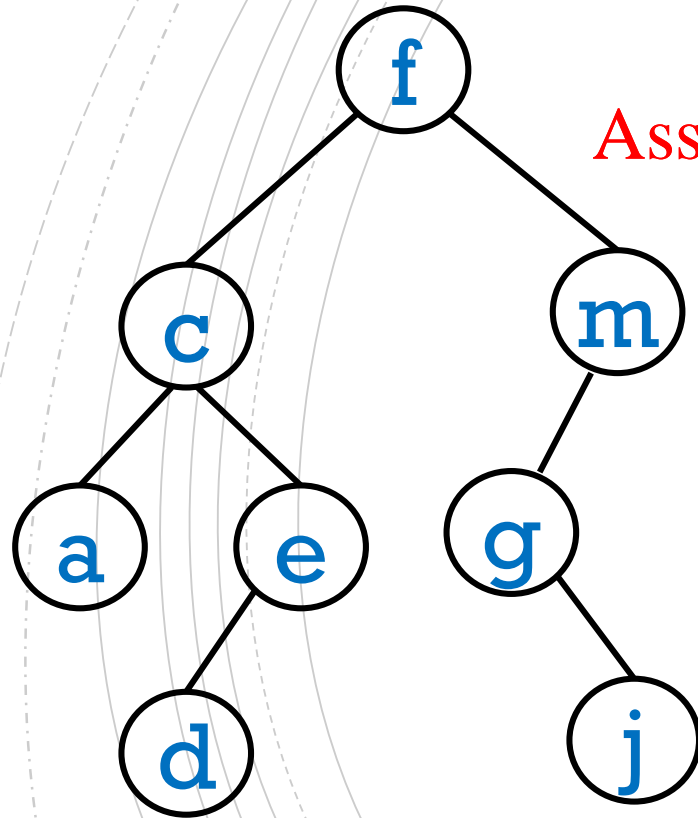
EXAMPLE



THIS IS NOT A BST. WHY NOT?



BST - TRAVERSALS



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

An in-order traversal on a BST visits the nodes in the natural order defined by the key.

a c d e f g j m

BINARY SEARCH TREE ADT

- find(key)
- findMin()
- findMax()
- add(key)
- remove(key)

Assignment Project Exam Help

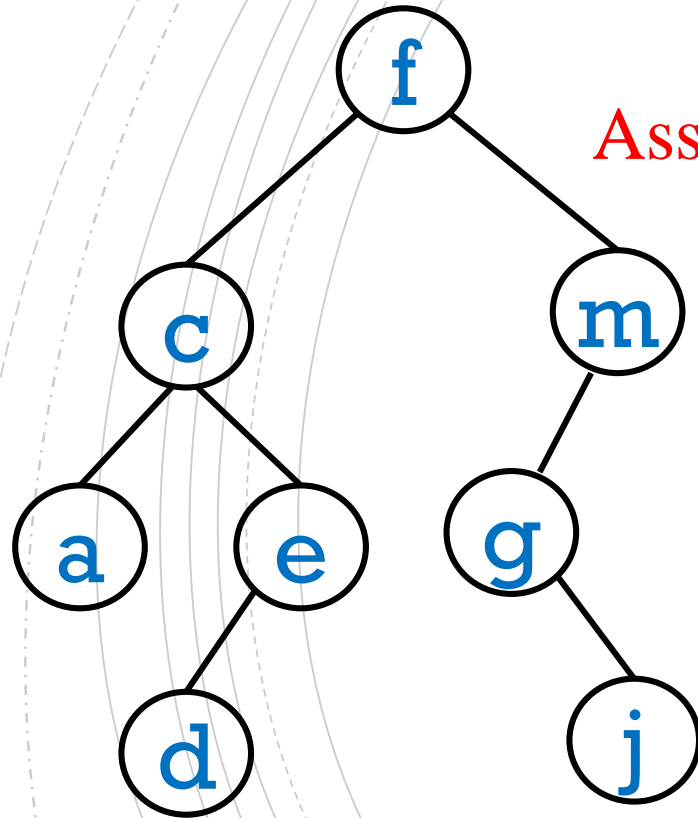
<https://powcoder.com>

Add WeChat powcoder

We can define the operations of a BST without knowing how they are implemented. (ADT)

Let's next look at some recursive algorithms for implementing them.

FIND()



Assignment Project Exam Help

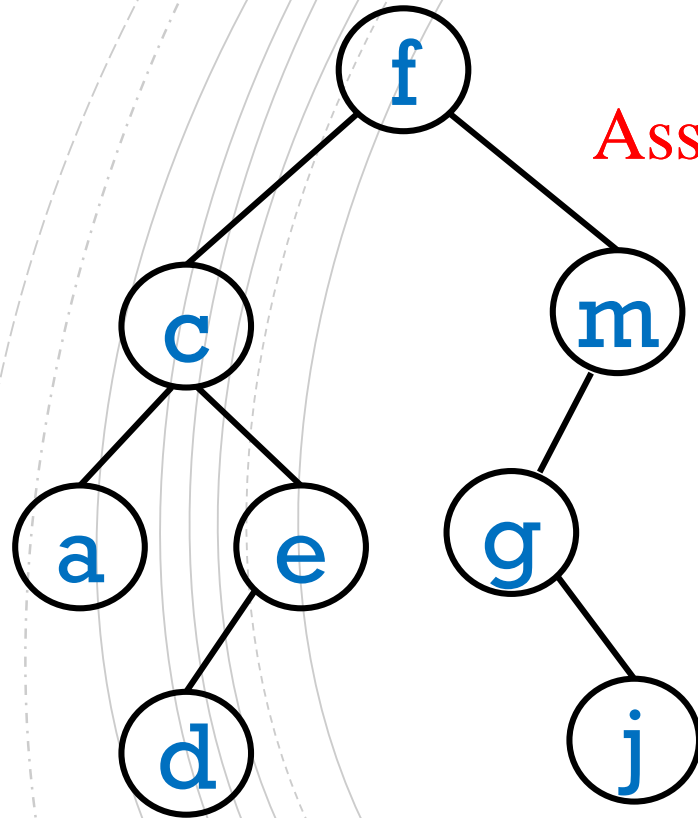
Desired behaviour:

<https://powcoder.com>

- `find(root, g)` returns the `g` node
- `find(root, s)` returns null.

Add WeChat powcoder

FIND() – IMPLEMENTATION



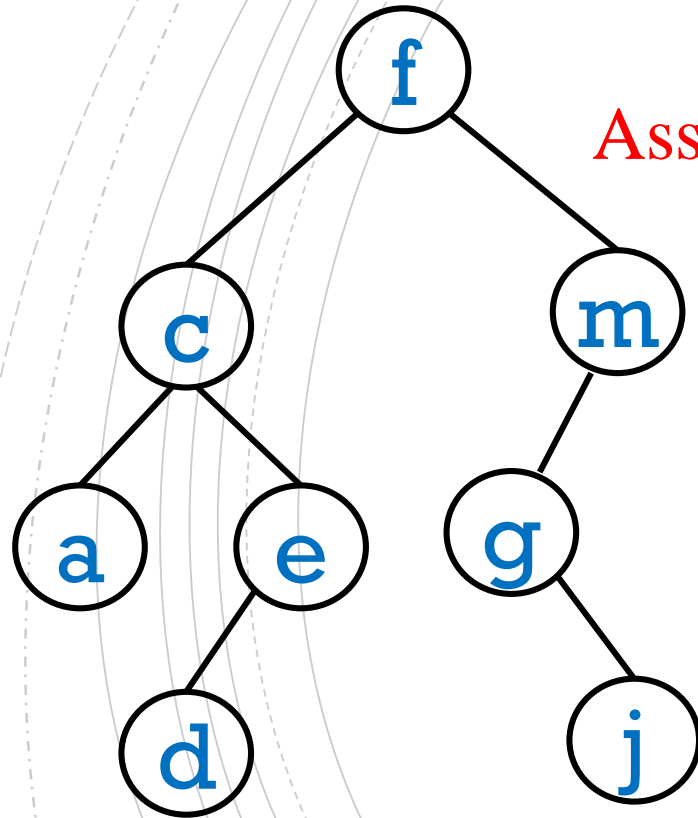
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
find(root, key) { // returns a node
    if (root == null)
        return null
    else if (root.key == key)
        return root
}
```

FIND() – IMPLEMENTATION



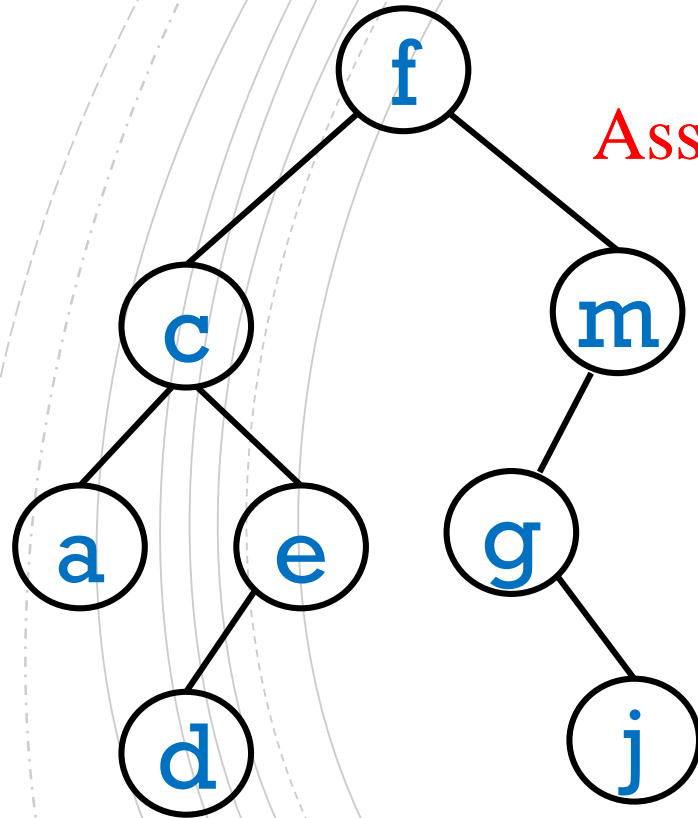
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
find(root, key) { // returns a node
    if (root == null)
        return null
    else if (root.key == key)
        return root
    else if (key < root.key)
        return find(root.left, key)
    else
        return find(root.right, key)
}
```

FINDMIN()



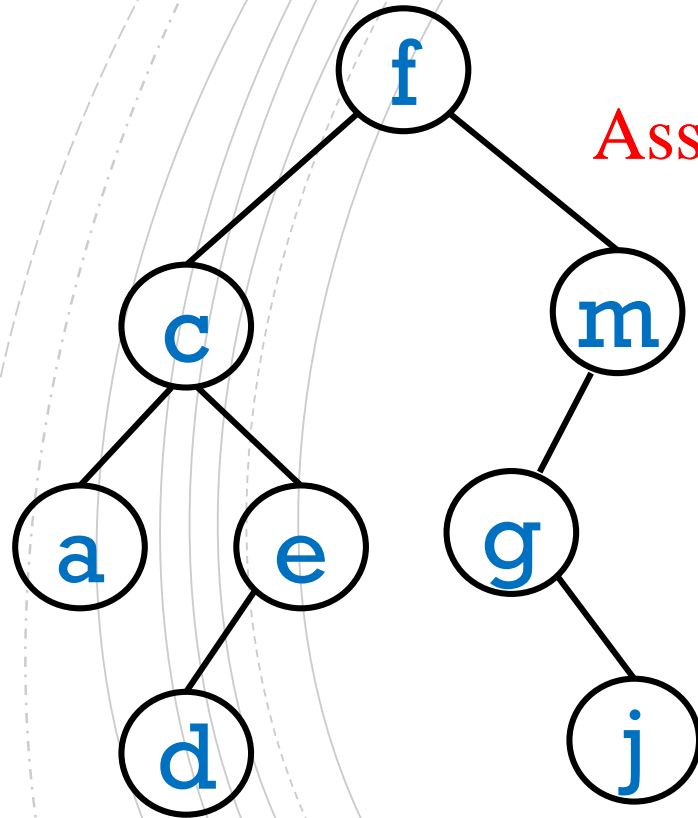
Assignment Project Exam Help

<https://powcoder.com>
findMin() should return the node with the smallest key. So, for example given

Add WeChat [powcoder](https://powcoder.com)
the BST on the left,

- findMin(root) returns ... ?

FINDMIN()



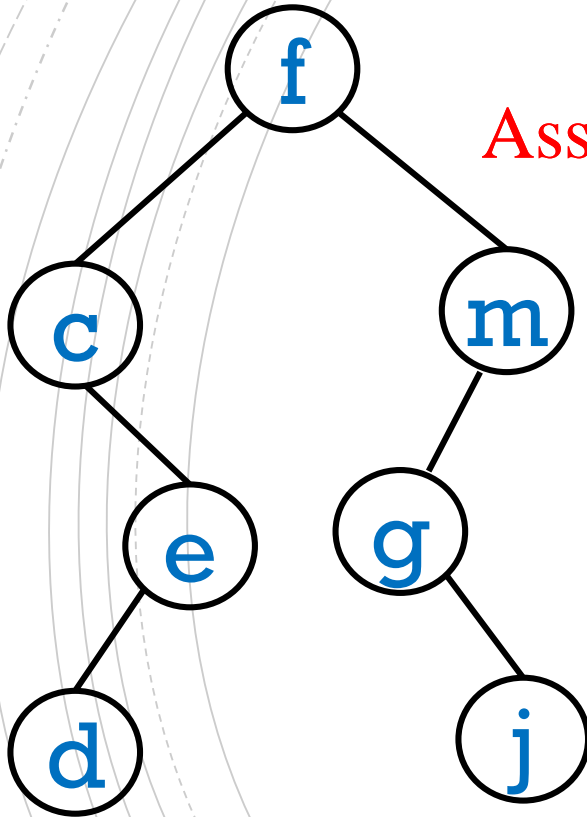
Assignment Project Exam Help

<https://powcoder.com>
findMin() should return the node with the smallest key. So, for example given

Add WeChat [powcoder](https://powcoder.com)
the BST on the left,

- `findMin(root)` returns the **a** node

FINDMIN()



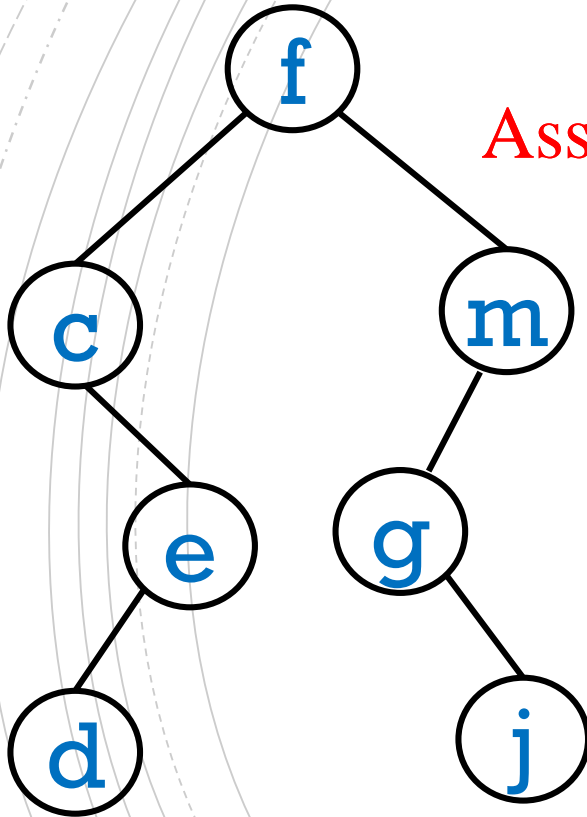
Assignment Project Exam Help

<https://powcoder.com>
findMin() should return the node with the smallest key. So, for example given

Add WeChat [powcoder](https://powcoder.com)
the BST on the left,

- `findMin(root)` returns ... ?

FINDMIN()



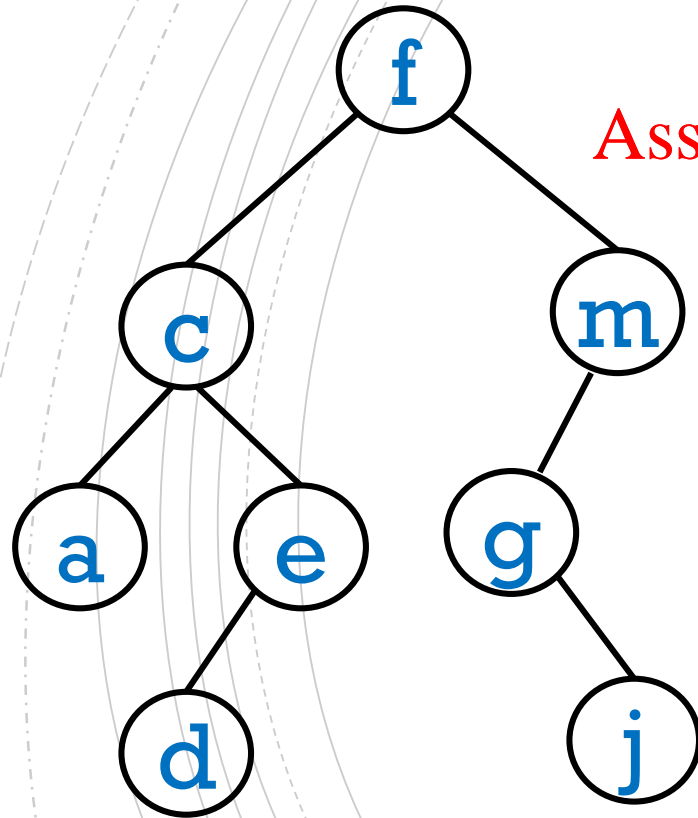
Assignment Project Exam Help

<https://powcoder.com>
findMin() should return the node with the smallest key. So, for example given

Add WeChat [powcoder](https://powcoder.com)
the BST on the left,

- `findMin(root)` returns the **c** node

FINDMIN() - IMPLEMENTATION



Assignment Project Exam Help

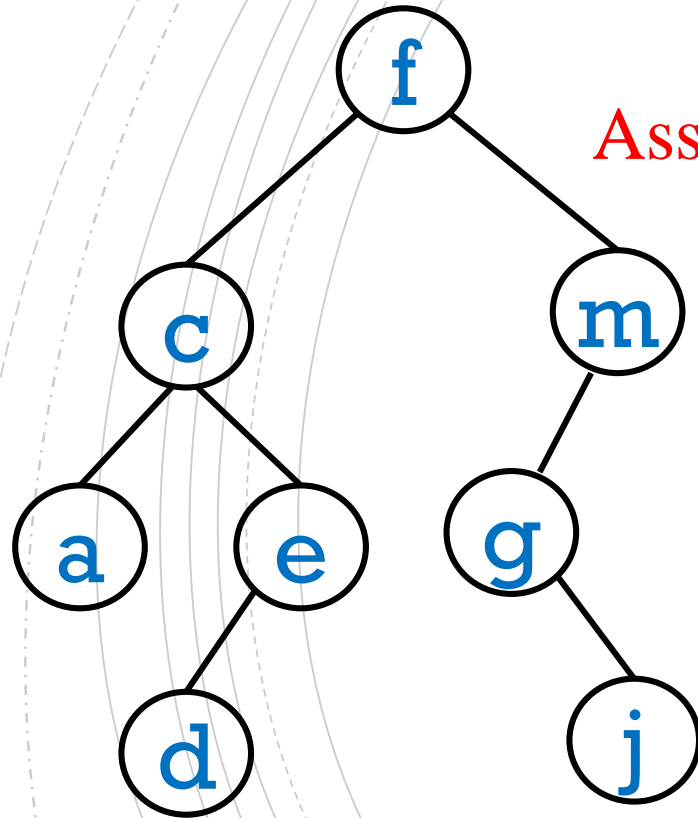
```
findMin(root){ // returns a node  
    if (root == null)
```

<https://powcoder.com>
 return null

Add WeChat powcoder

```
}
```

FINDMIN() - IMPLEMENTATION



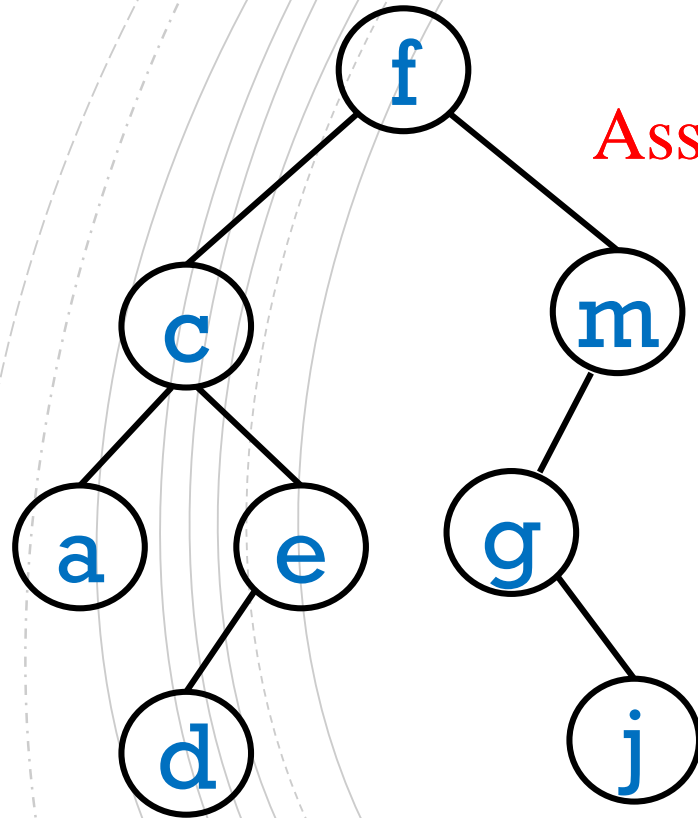
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
findMin(root){ // returns a node
    if (root == null)
        return null
    else if (root.left == null)
        return root
    else
        return findMin( root.left )
}
```

FINDMAX()



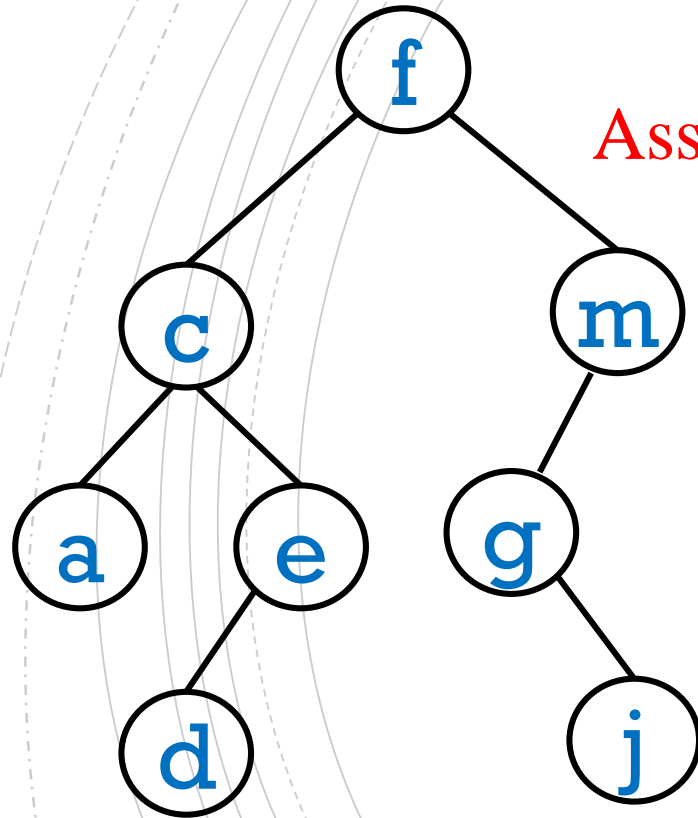
Assignment Project Exam Help

<https://powcoder.com>
findMax() should return the node with the greatest key. So, for example given

Add WeChat [powcoder](https://powcoder.com) the BST on the left,

- findMax(root) returns ... ?

FINDMAX()



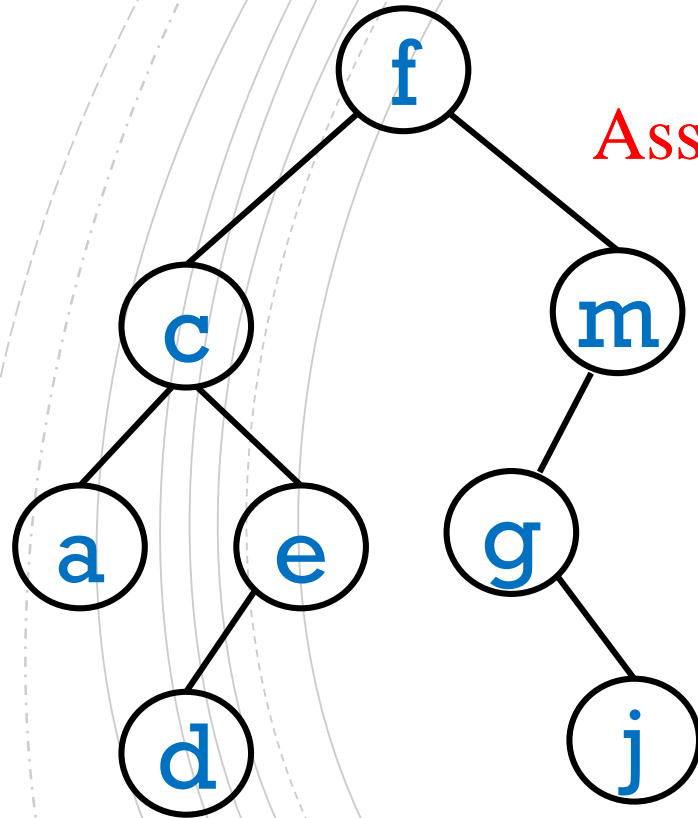
Assignment Project Exam Help

<https://powcoder.com>
findMax() should return the node with the greatest key. So, for example given

Add WeChat [powcoder](https://powcoder.com)
the BST on the left,

- `findMax(root)` returns the **m** node.

FINDMAX() – IMPLEMENTATION



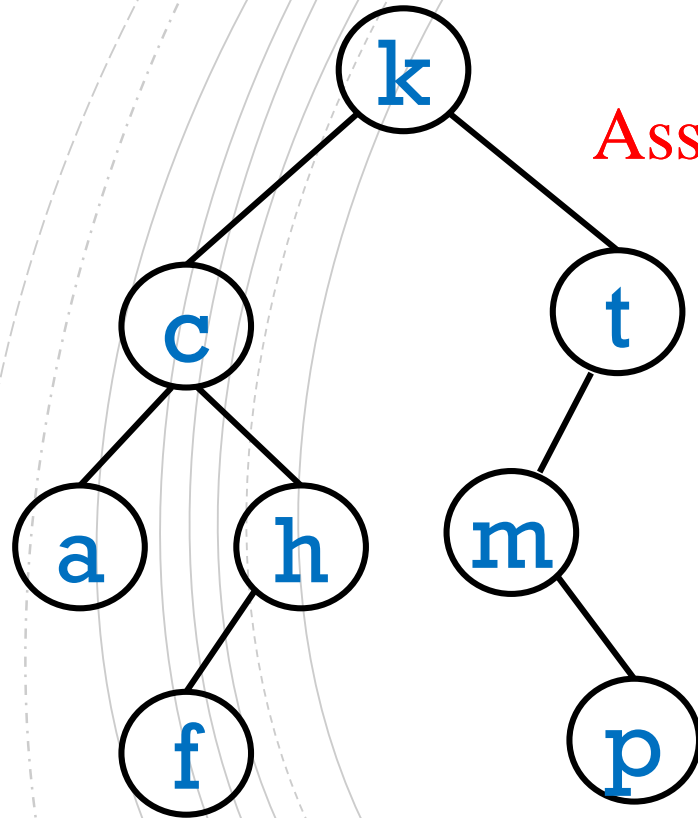
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
findMax(root) { // returns a node
    if (root == null)
        return null
    else if (root.right == null)
        return root
    else
        return findMax (root.right)
}
```

ADD()



Assignment Project Exam Help

add(key) should add a BSTNode to the tree.

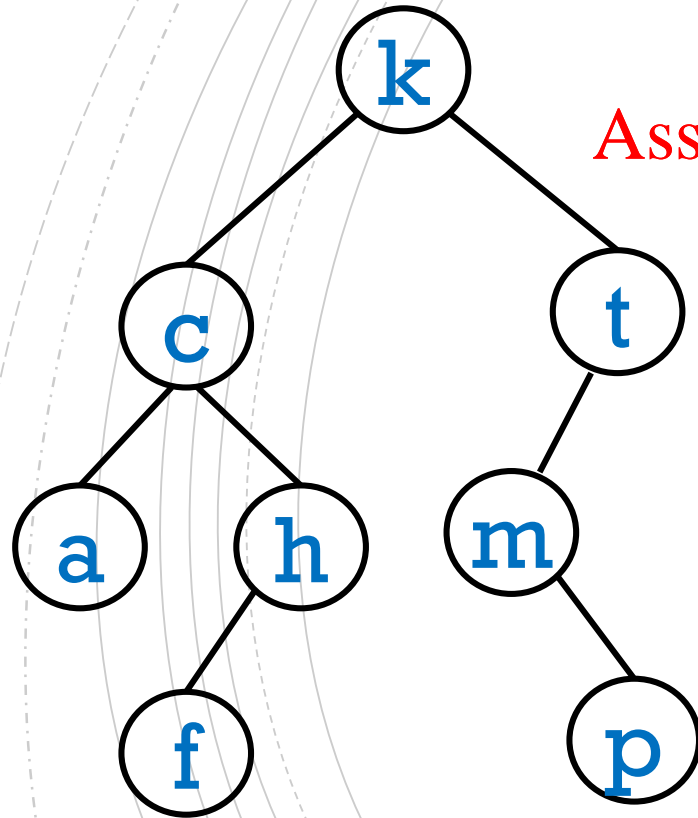
<https://powcoder.com>

- Where?

Add WeChat powcoder

- How?

ADD()



Assignment Project Exam Help

add(key) should add a BSTNode to the tree.

<https://powcoder.com>

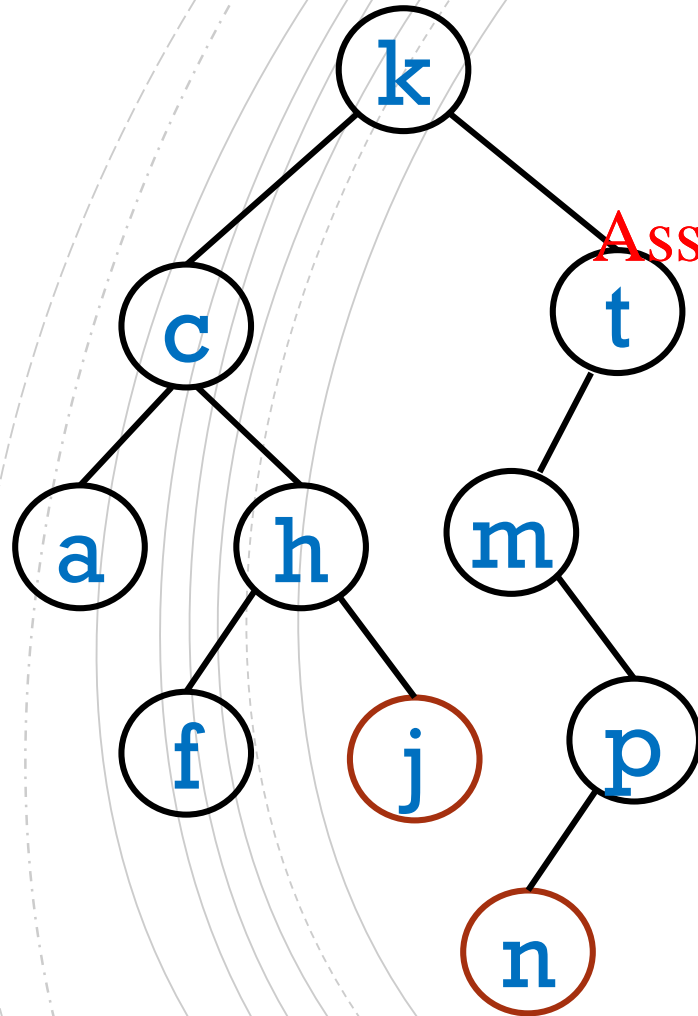
- add(j) ?

Add WeChat powcoder

- add(w) ?

A new node is always a leaf.

ADD()



Assignment Project Exam Help

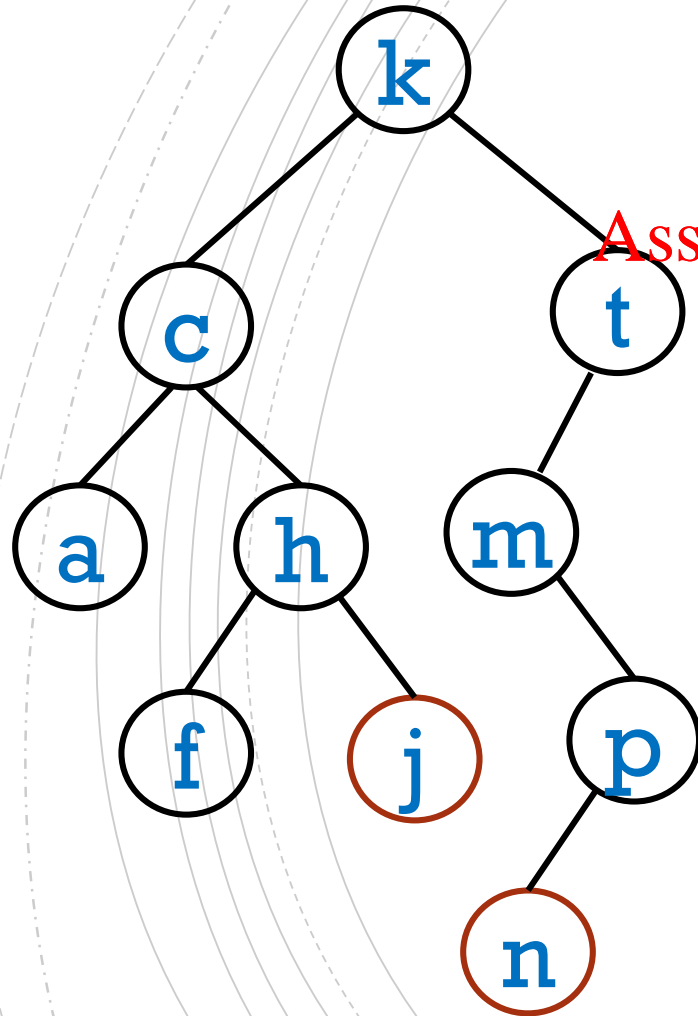
add(key) should add a BSTNode to the
<https://powcoder.com>
tree.

Add WeChat powcoder

- add(**n**) ?

A new node is always a leaf.

ADD() - IMPLEMENTATION



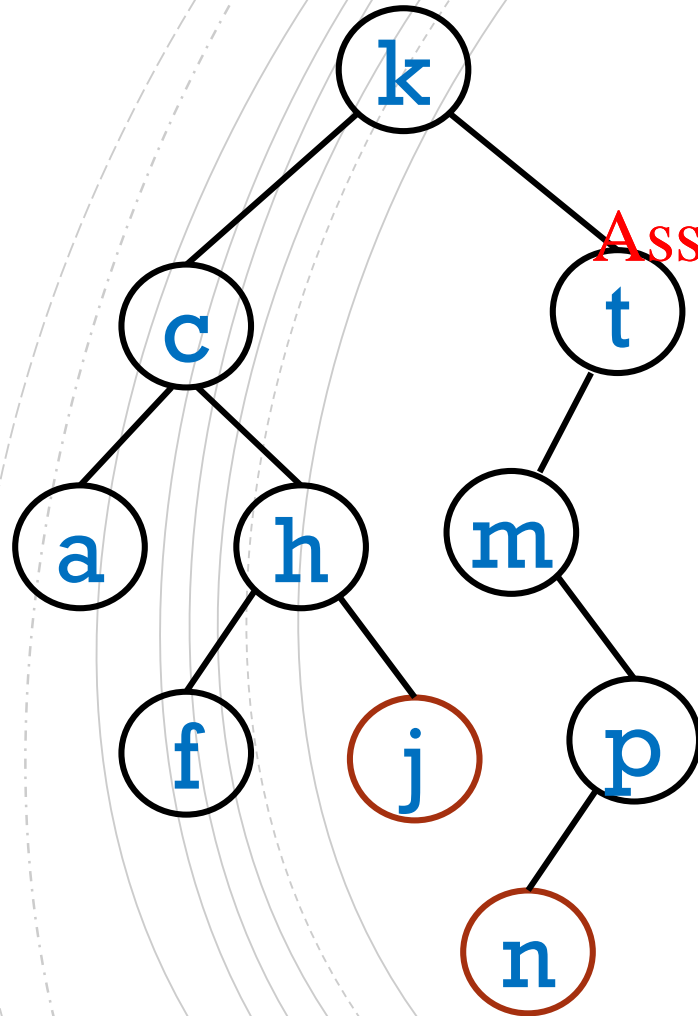
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
add(root, key) { // returns root node  
  
    return root  
}
```

ADD() - IMPLEMENTATION



Assignment Project Exam Help

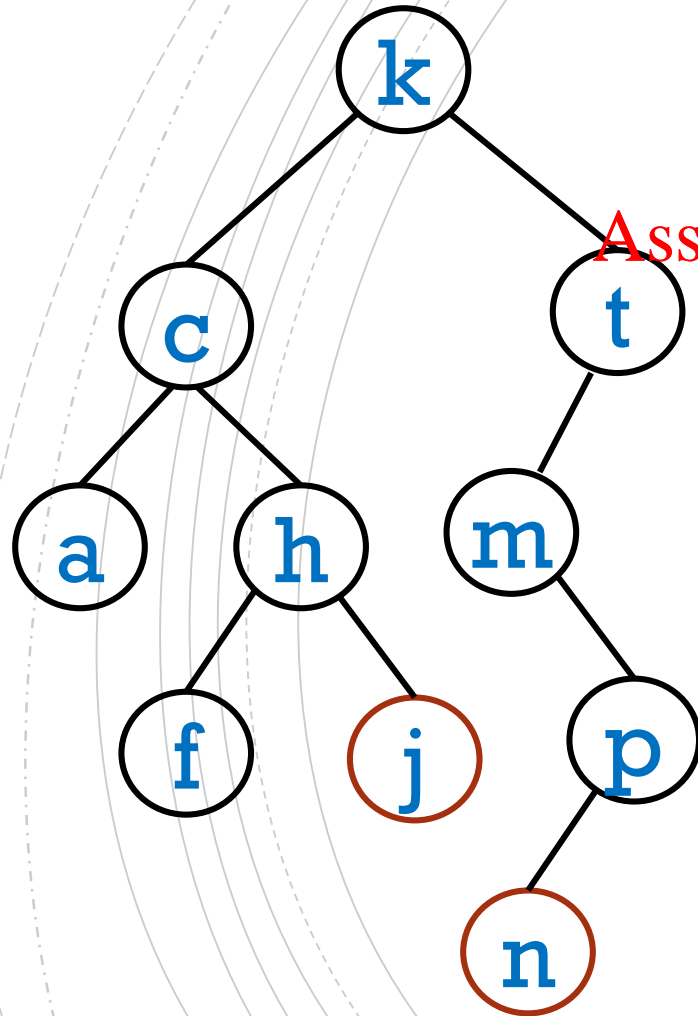
<https://powcoder.com>

Add WeChat powcoder

```
add(root, key) { // returns root node
    if (root == null)
        root = new BSTnode(key)

    return root
}
```

ADD() - IMPLEMENTATION



Assignment Project Exam Help

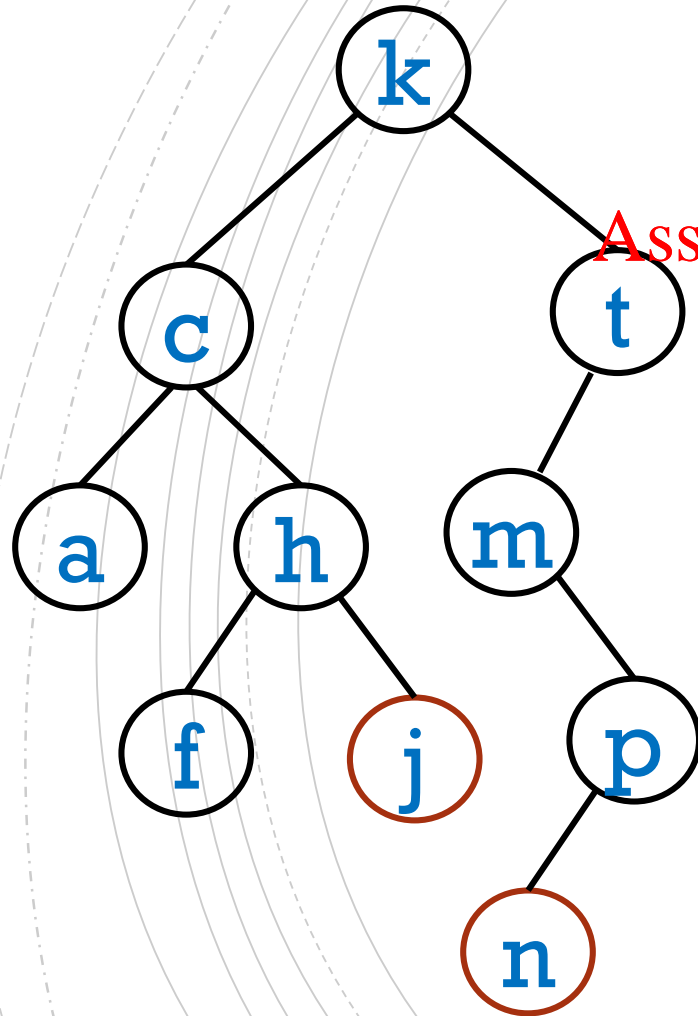
<https://powcoder.com>

Add WeChat powcoder

```
add(root, key) { // returns root node
    if (root == null)
        root = new BSTnode(key)
    else if (key < root.key) {
        root.left = add(root.left, key)
    }

    return root
}
```

ADD() - IMPLEMENTATION



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

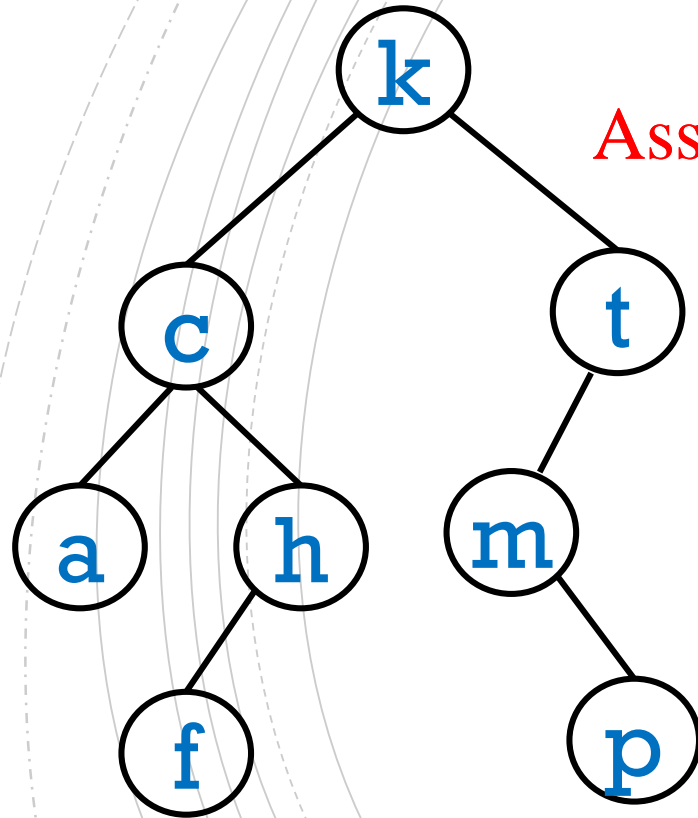
```
add(root, key) { // returns root node
    if (root == null)
        root = new BSTnode(key)
    else if (key < root.key) {
        root.left = add(root.left, key)
    }
    else if (key > root.key) {
        root.right = add(root.right, key)
    }
    return root
}
```

Q: What happens if root.key == key?

A: Nothing!

REMOVE()

remove(**c**) →



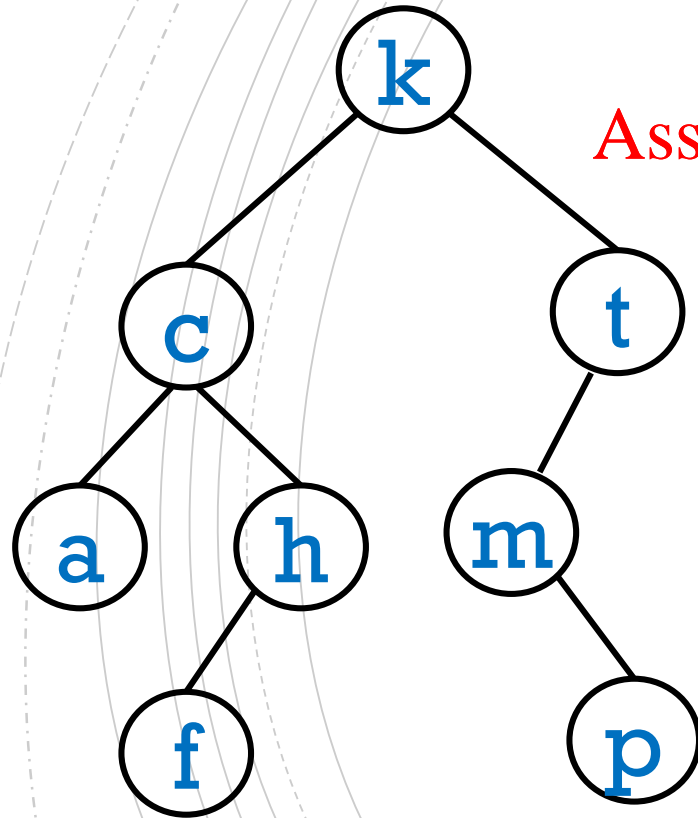
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

REMOVE()

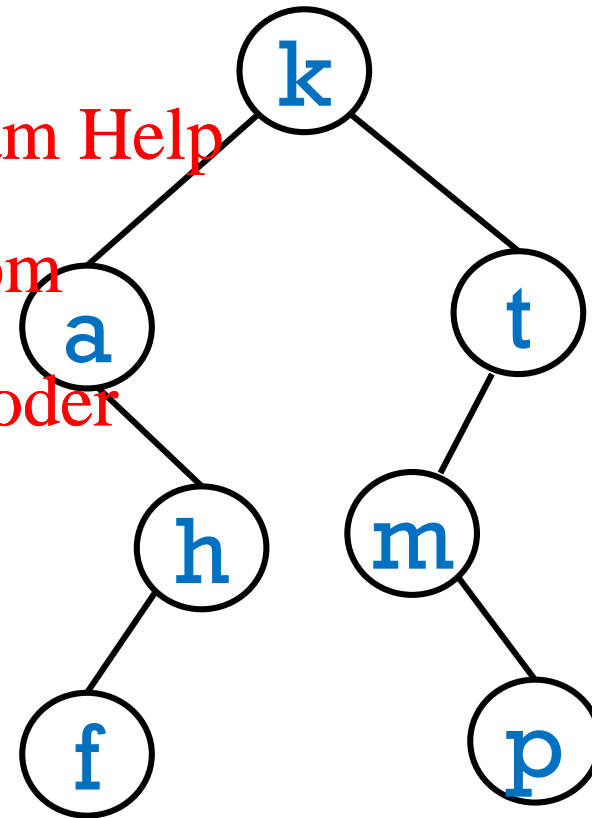
remove(**c**) → this is one way to do it



Assignment Project Exam Help

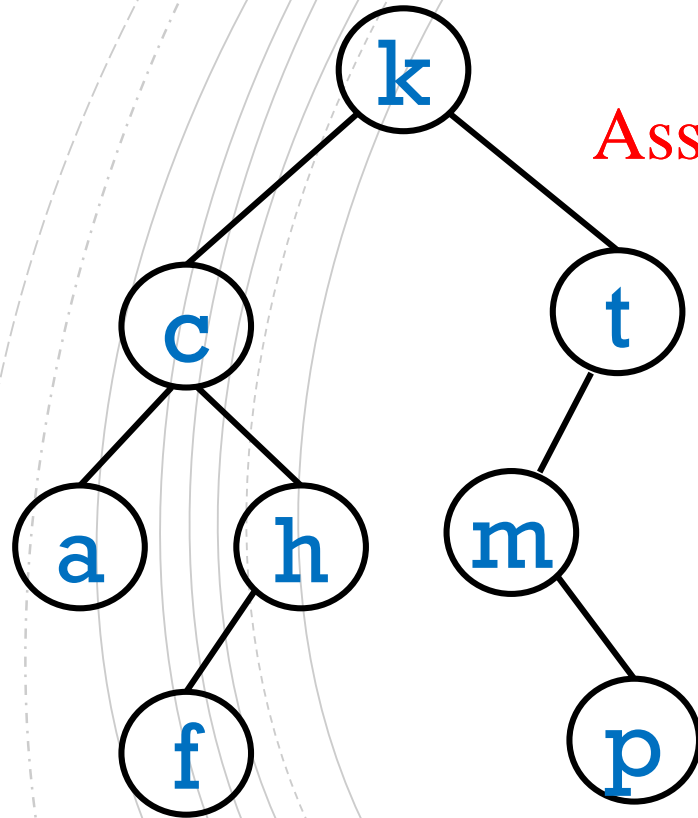
<https://powcoder.com>

Add WeChat powcoder



REMOVE()

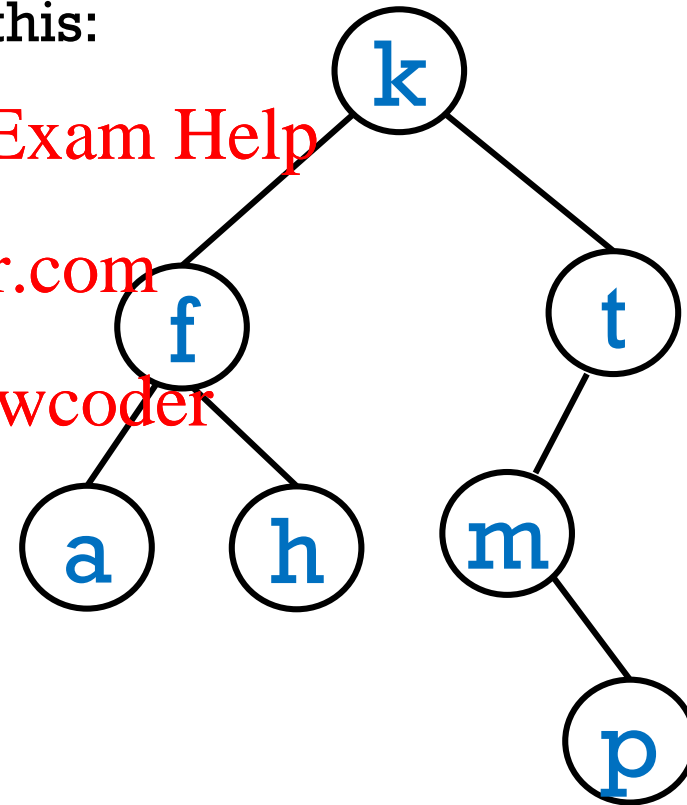
remove(**c**) → the following algorithm
does this:



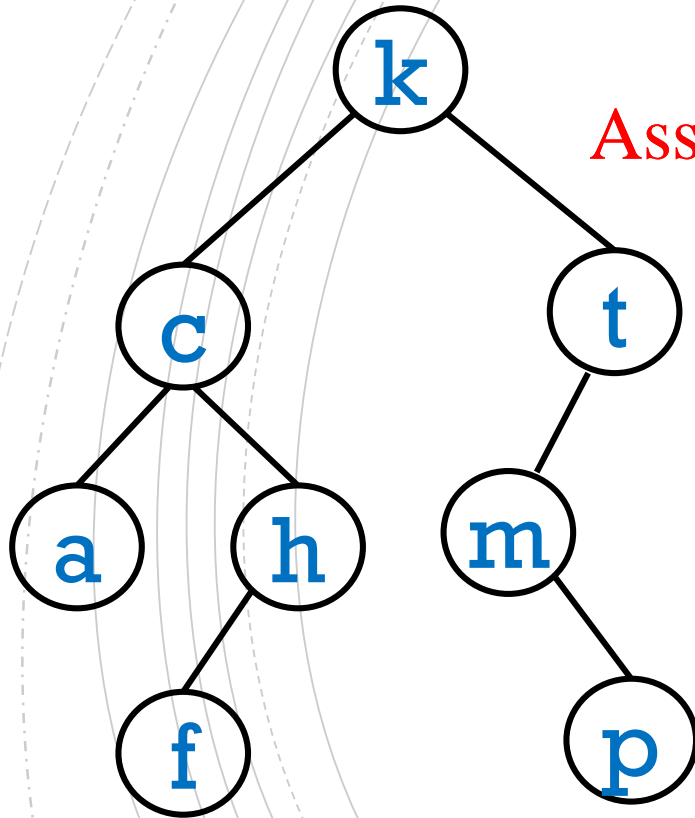
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



REMOVE() - IMPLEMENTATION



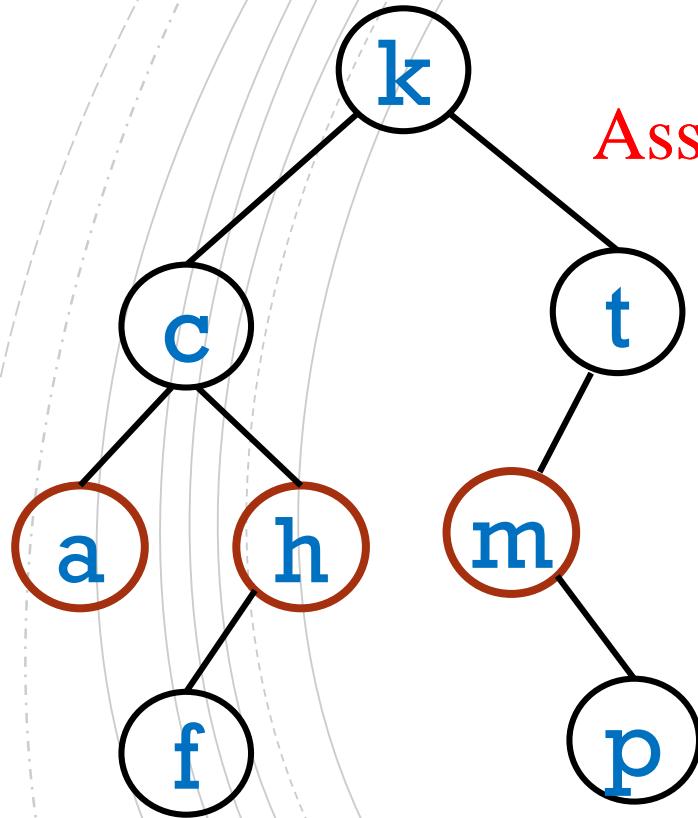
```
remove(root, key){ // returns root node
    if( root == null )
        return null
    else if ( key < root.key )
    else if ( key > root.key )
    else
    }
    return root
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

REMOVE() - IMPLEMENTATION



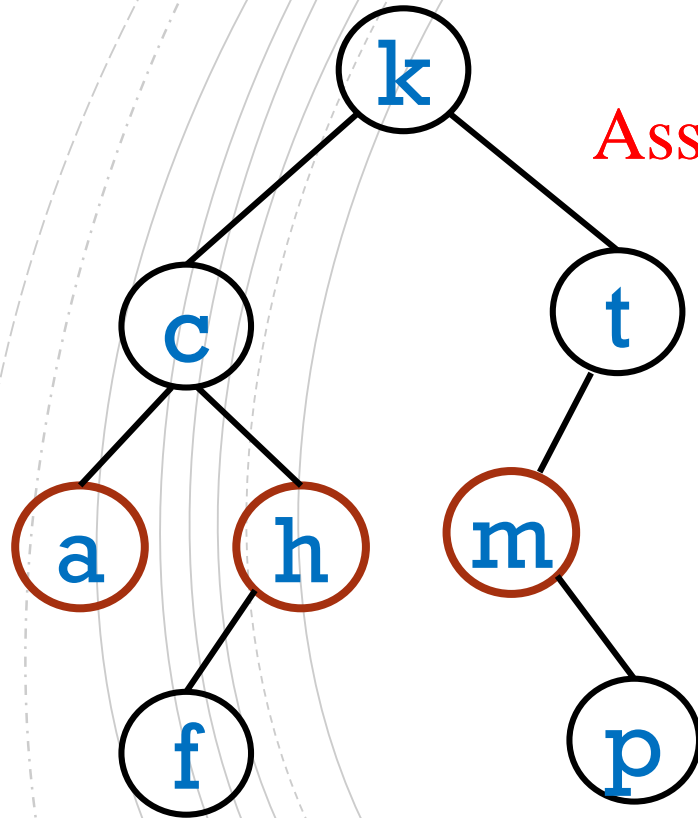
```
remove (root, key) { // returns root node
    if( root == null )
        return null
    else if ( key < root.key )
        root.left = remove (root.left, key)
    else if ( key > root.key )
        root.right = remove (root.right, key)
    else
        // Node to be removed found
        // If node has no children, simply return null
        // If node has one child, return the child
        // If node has two children, find the in-order successor (smallest node in the right subtree)
        // and replace the node's value with the successor's value, then remove the successor node
        // This implementation assumes a standard BST structure where the in-order successor is the leftmost node in the right subtree
        // For simplicity, we'll assume the node to be removed has no children or one child, as the diagram shows.
        // In a full implementation, the logic for two children would be more complex.
        // For the purpose of this diagram, we'll assume the node to be removed is 'h' or 'm'.
        // If removing 'h', its left child 'f' would become the new right child of 'c'.
        // If removing 'm', its right child 'p' would become the new left child of 't'.
        // The diagram shows 'a', 'h', 'm', and 'p' with red borders, suggesting they are the nodes being removed or the result of the removal process.
        // The code above is a simplified version of the remove function.
    }
    return root
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

REMOVE() - IMPLEMENTATION

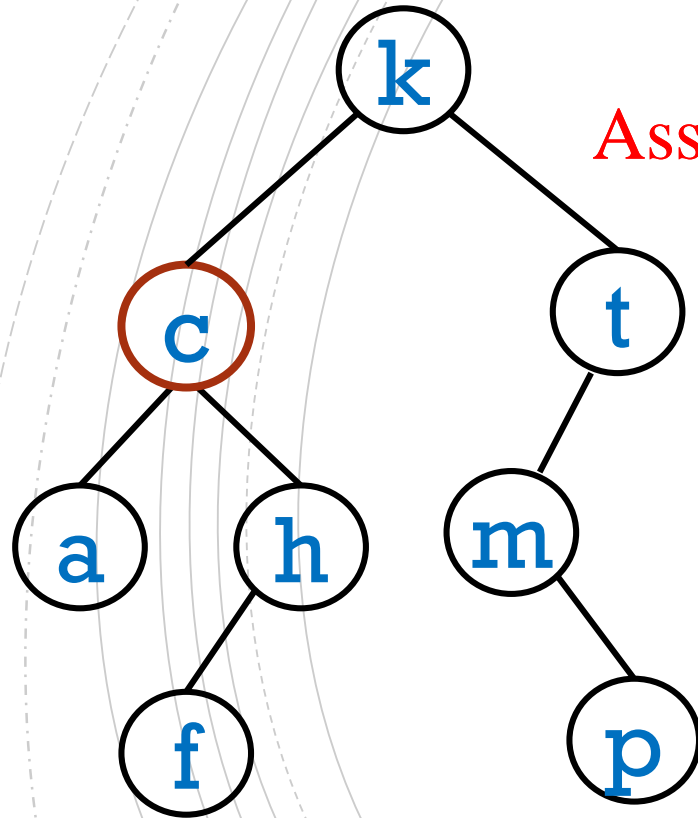


Assignment Project Exam Help
<https://powcoder.com>
Add WeChat powcoder

```
remove (root, key) { // returns root node
    if ( root == null )
        return null
    else if ( key < root.key )
        root.left = remove (root.left, key)
    else if ( key > root.key )
        root.right = remove (root.right, key)
    else if (root.left == null)
        root = root.right
    else if (root.right == null)
        root = root.left

    }
    return root
}
```

REMOVE() - IMPLEMENTATION

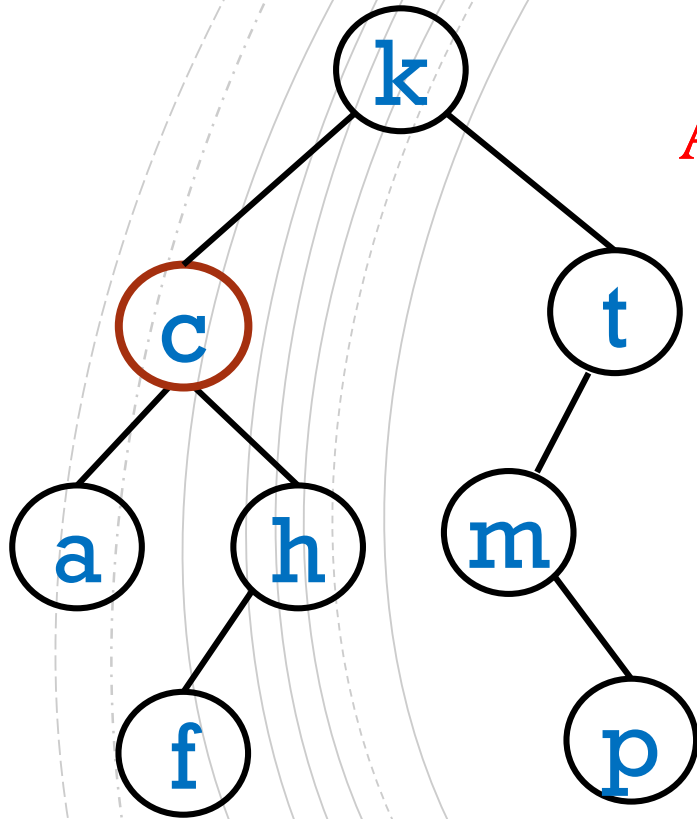


Assignment Project Exam Help
<https://powcoder.com>
Add WeChat powcoder

```
remove (root, key) { // returns root node
    if ( root == null )
        return null
    else if ( key < root.key )
        root.left = remove (root.left, key)
    else if ( key > root.key )
        root.right = remove (root.right, key)
    else if (root.left == null)
        root = root.right
    else if (root.right == null)
        root = root.left

    }
    return root
}
```

REMOVE() - IMPLEMENTATION

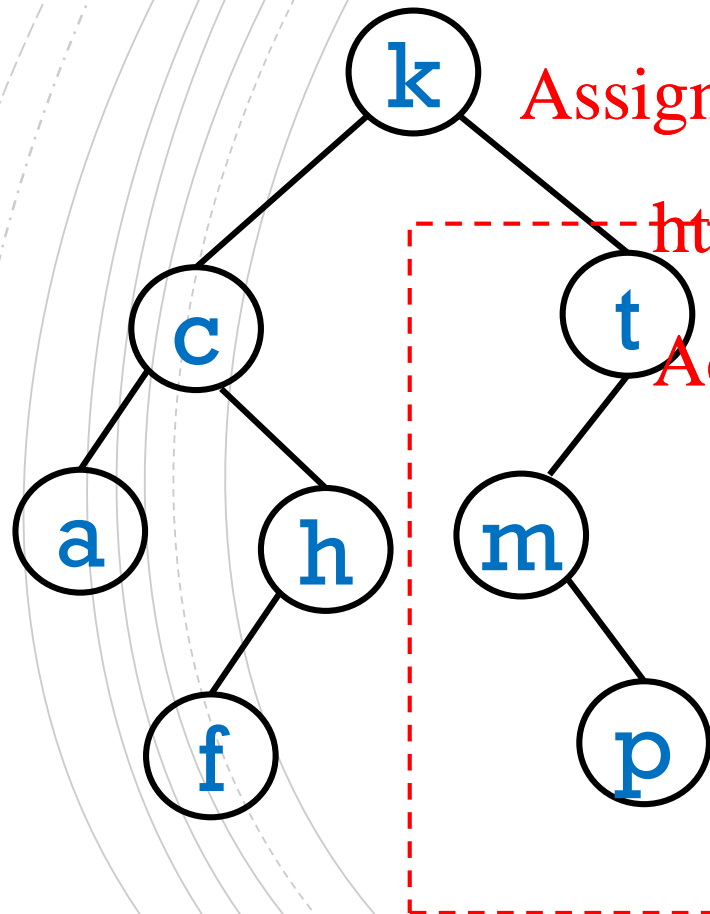


Assignment Project Exam Help
<https://powcoder.com>
Add WeChat powcoder

```
remove(root, key){ // returns root node
    if( root == null )
        return null
    else if ( key < root.key )
        root.left = remove(root.left, key)
    else if ( key > root.key )
        root.right = remove(root.right, key)
    else if (root.left == null)
        root = root.right
    else if (root.right == null)
        root = root.left
    else {
        root.key = findMin(root.right).key
        root.right = remove(root.right, root.key)
    }
    return root
}
```

REMOVE() - EXAMPLE

remove(**k**) →



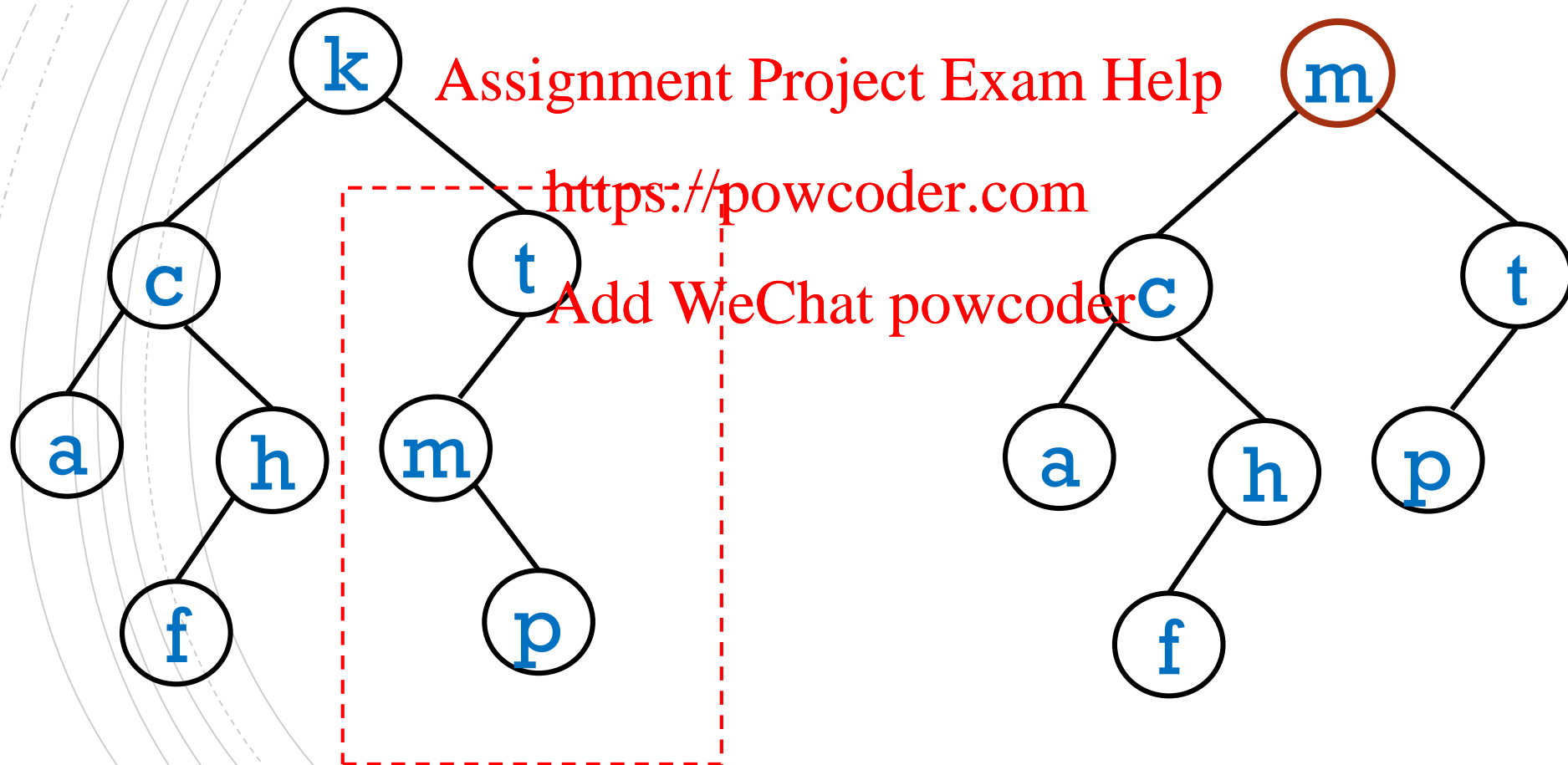
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

REMOVE() - EXAMPLE

remove(**k**) →

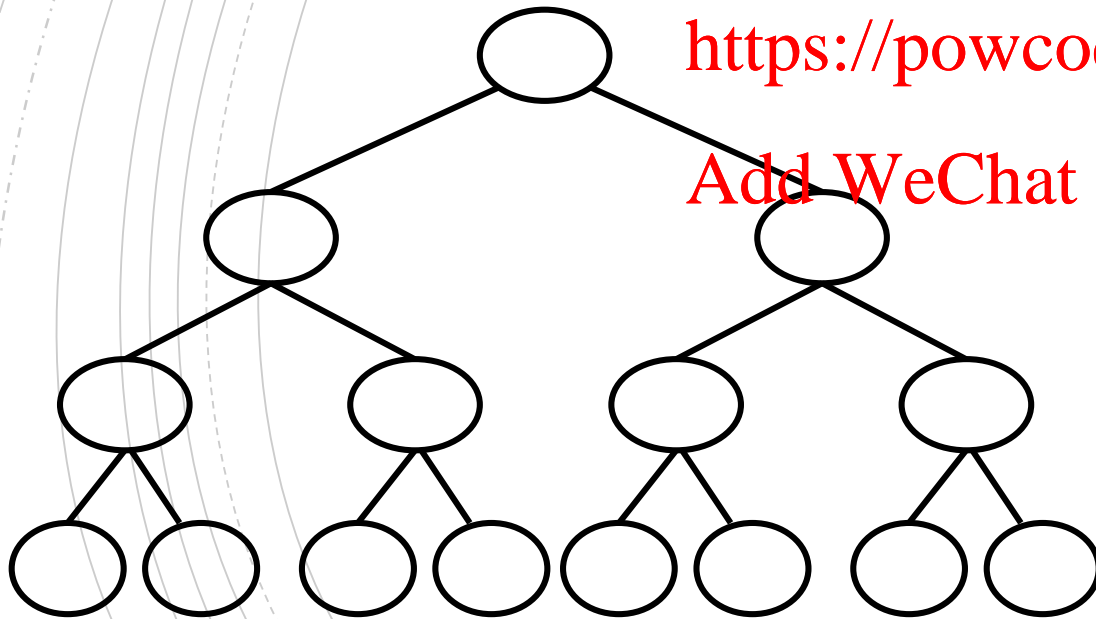


BALANCED VS UNBALANCED

balanced

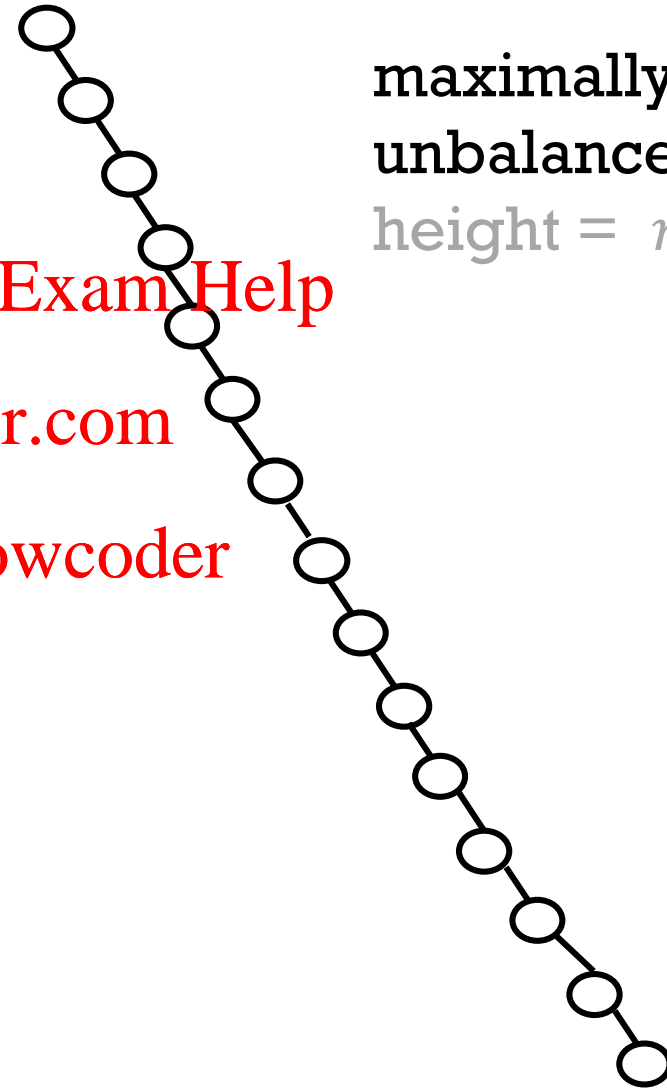
$$\text{height} = \log(n + 1) - 1$$

$$n = 2^{h+1} - 1$$



maximally
unbalanced

$$\text{height} = n - 1$$



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

BEST VS WORST CASE SCENARIO

best case

worst case

findMin() Assignment Project Exam Help

findMax()

<https://powcoder.com>

Add WeChat powcoder

find(key)

add(key)

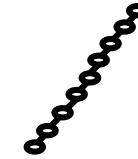
remove(key)

BEST VS WORST CASE SCENARIO

best case

worst case

findMin() $O(1)$ findMax() $O(n)$



findMax()

<https://powcoder.com>

Add WeChat powcoder

find(key)

add(key)

remove(key)

BEST VS WORST CASE SCENARIO

best case

worst case

findMin() $O(1)$ $O(n)$

findMax() $O(1)$ $O(n)$

find(key)

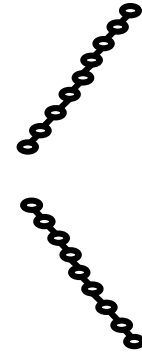
add(key)

remove(key)

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



BEST VS WORST CASE SCENARIO

best case

worst case

findMin() $O(1)$ $O(n)$

findMax() $O(1)$ $O(n)$

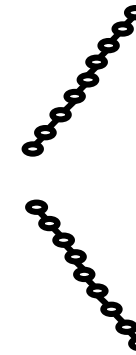
find(key) $O(1)$ $O(n)$

add(key)

remove(key)

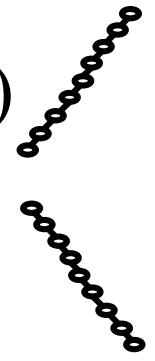
<https://powcoder.com>

Add WeChat powcoder



could be zigzag

BEST VS WORST CASE SCENARIO

	best case	worst case	
findMin()	$O(1)$	$O(n)$	
findMax()	$O(1)$	$O(n)$	
find(key)	$O(1)$	$O(n)$	} Could be zigzag
add(key)	$O(1)$	$O(n)$	
remove(key)	$O(1)$	$O(n)$	

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

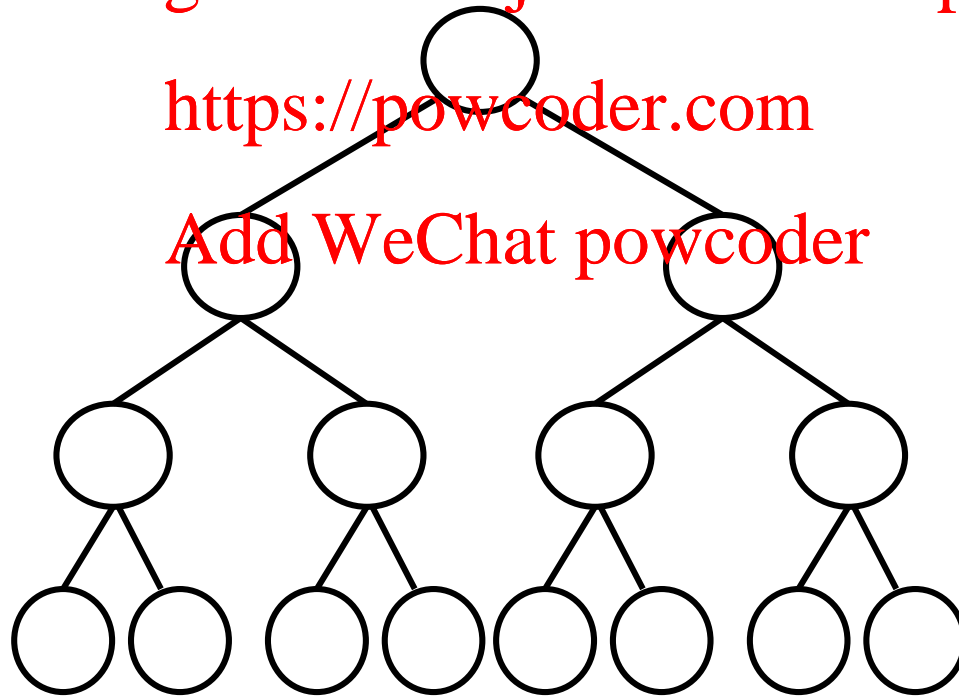
BINARY SEARCH (TREES)

When a binary search tree is balanced, then finding a key is very similar to a binary search.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



BALANCED BINARY SEARCH TREES

(COMP 251: AVL TREES, RED-BLACK TREES)

	best case	worst case
findMin()	$O(\log n)$	$O(\log n)$
findMax()	$O(\log n)$	$O(\log n)$
find(key)	$O(1)$	$O(\log n)$
add(key)	$O(\log n)$	$O(\log n)$
remove(key)	$O(\log n)$	$O(\log n)$

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



Coming Soon

Assignment Project Exam Help

In the next videos:

■ Heaps <https://powcoder.com>

Add WeChat powcoder