

# COMP 250

## INTRODUCTORY SCIENCE

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Week 12-1: Binary

Add WeChat: edu\_assist\_pro

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://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

Assignment Project Exam Help

B

H

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

## BSTNode

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

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

```
class  
    K Key;  
    BSTNode<K> leftchild;  
    BSTNode<K> rightchild;  
    :  
}
```

## BINARY SEARCH TREE DEFINITION

- binary tree

Assignment Project Exam Help

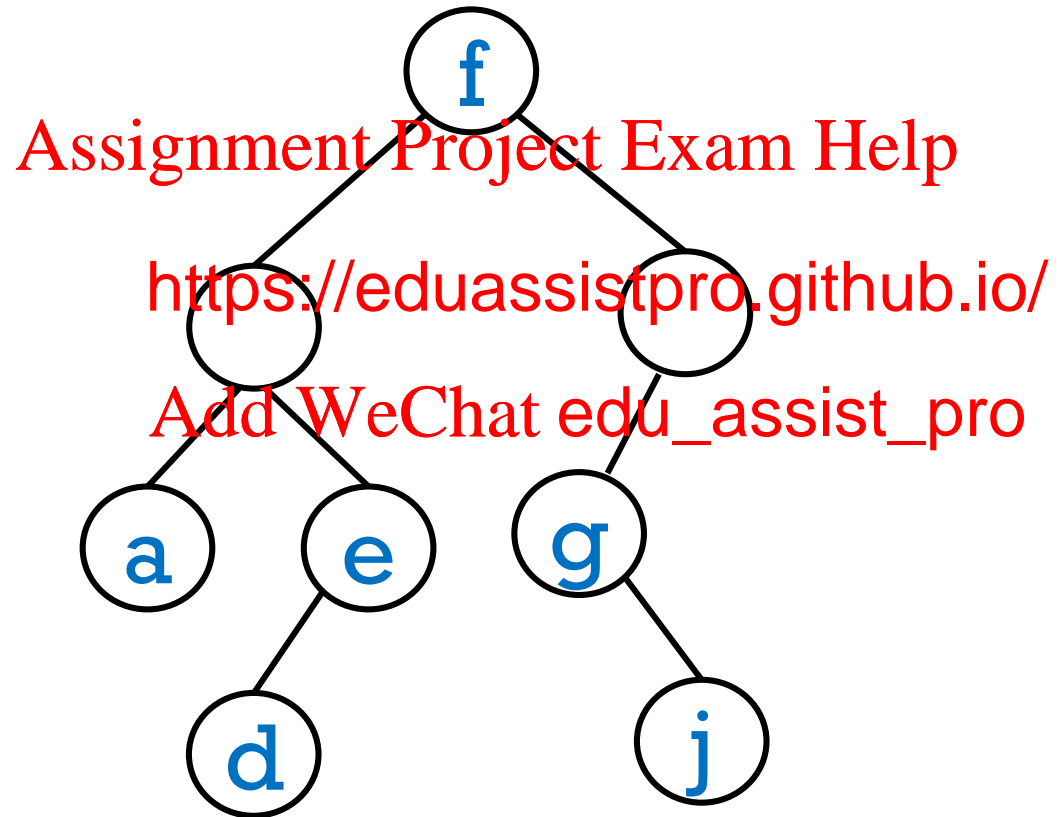
- keys are comparable, a

<https://eduassistpro.github.io/>

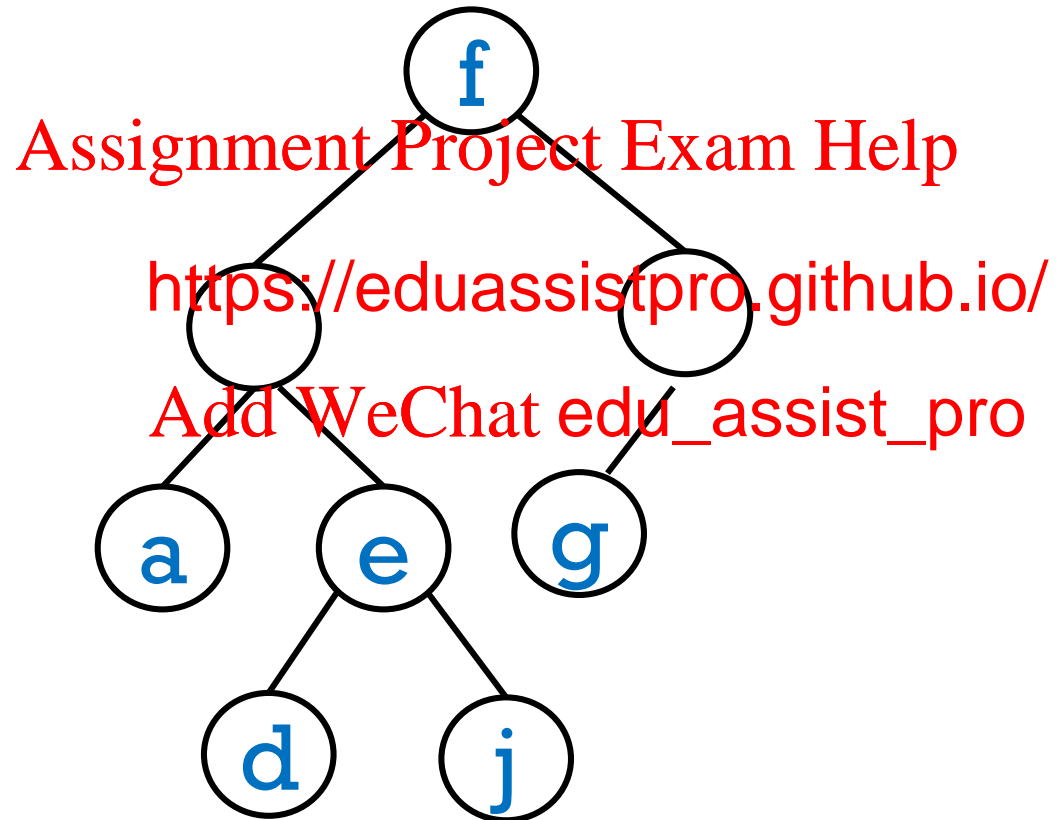
Add WeChat edu\_assist\_pro

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

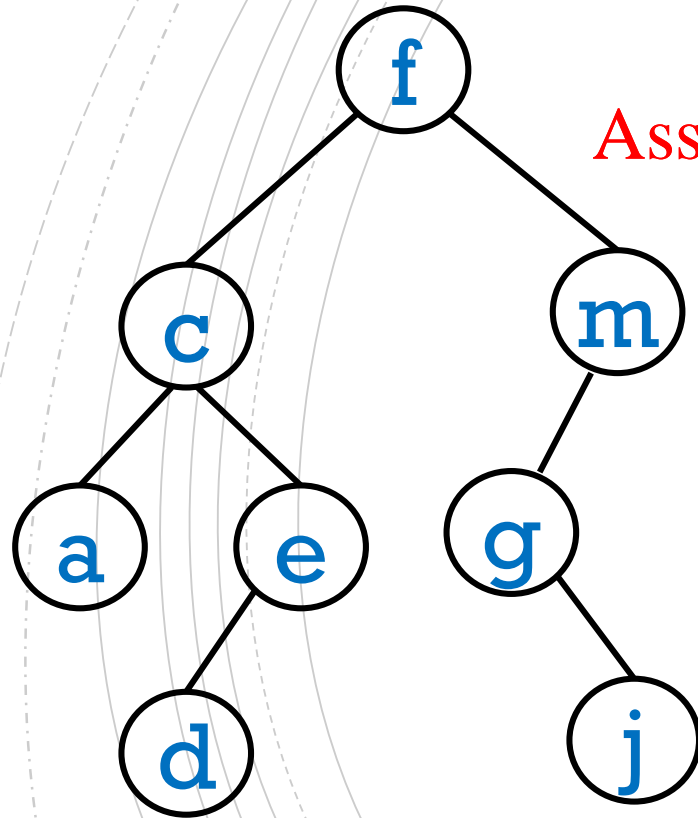
## EXAMPLE



THIS IS NOT A BST. WHY NOT?



## BST - TRAVERSALS



Assignment Project Exam Help  
An in-order traversal on a BST visits the natural order

<https://eduassistpro.github.io/>

Add WeChat [edu\\_assist\\_pro](#)



## BINARY SEARCH TREE ADT

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

Assignment Project Exam Help

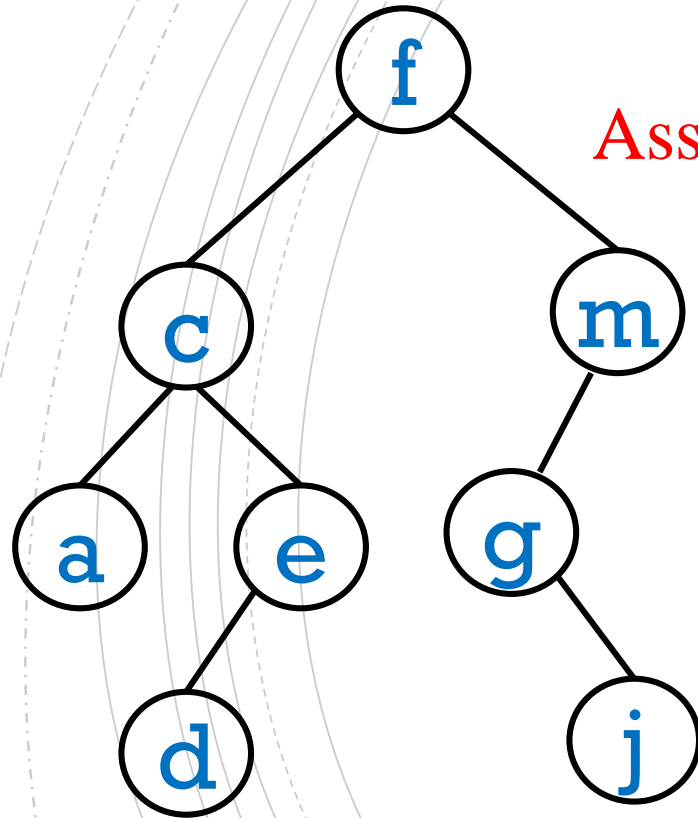
<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

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

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

FIND()



Assignment Project Exam Help

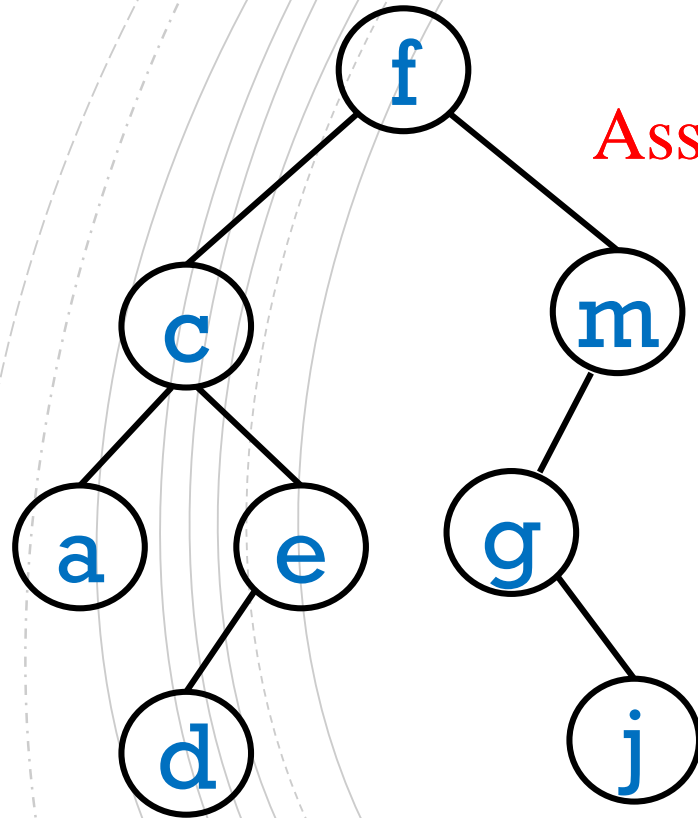
viour:

<https://eduassistpro.github.io/>

, **g**) returns the **g** node

Add WeChat **edu\_assist\_pro** returns null.

## FIND() – IMPLEMENTATION



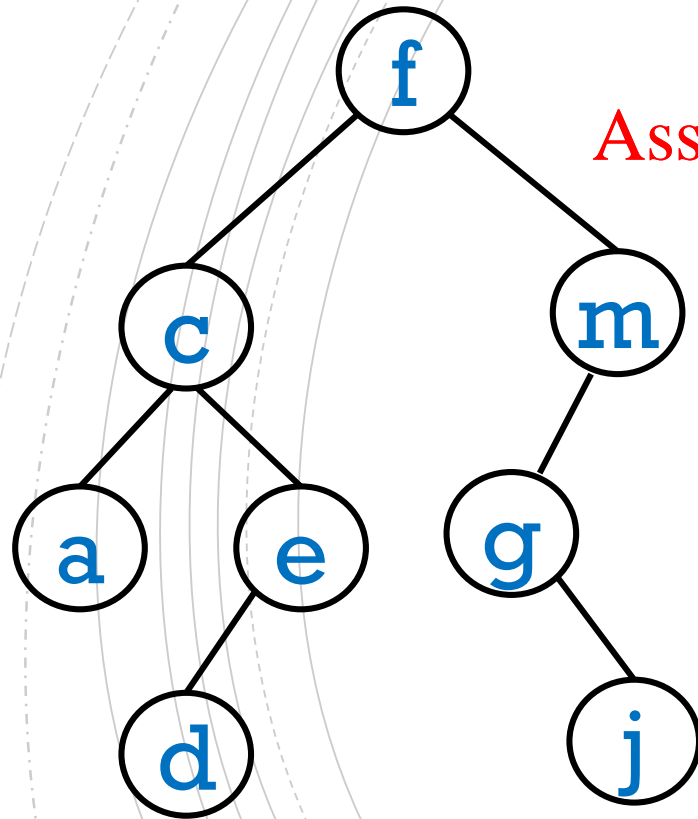
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

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

## FIND() – IMPLEMENTATION



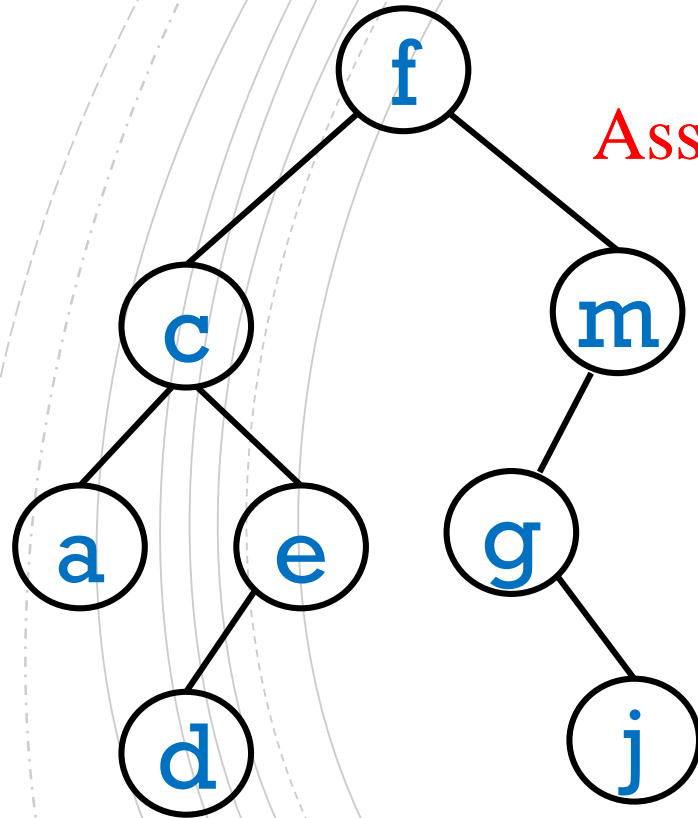
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

```
find(root, key) { // returns a node
    if (root == null)
        return null
    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://eduassistpro.github.io/>

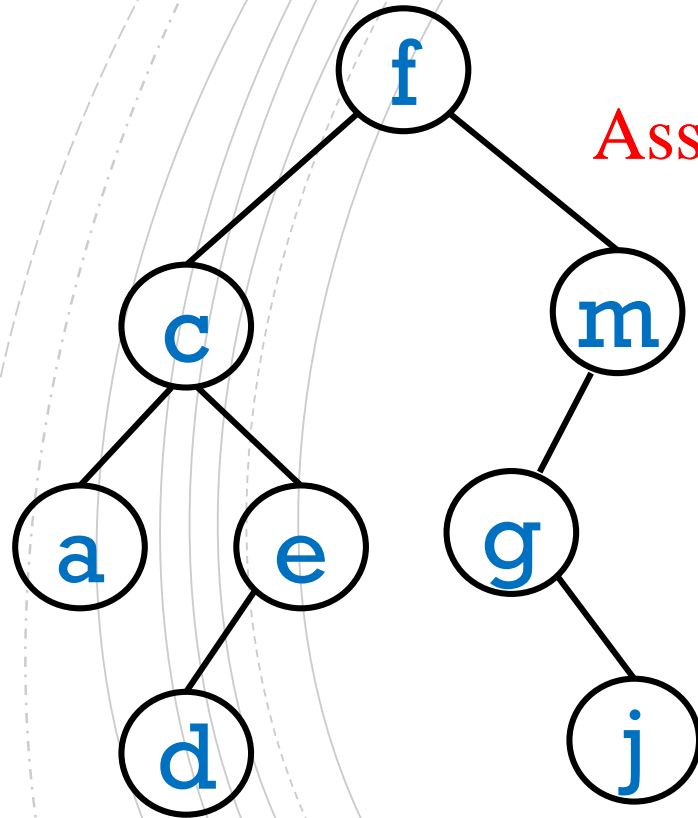
Add WeChat edu\_assist\_pro

uld return the node with

ey. So, for example given

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

## FINDMIN()



Assignment Project Exam Help

<https://eduassistpro.github.io/>

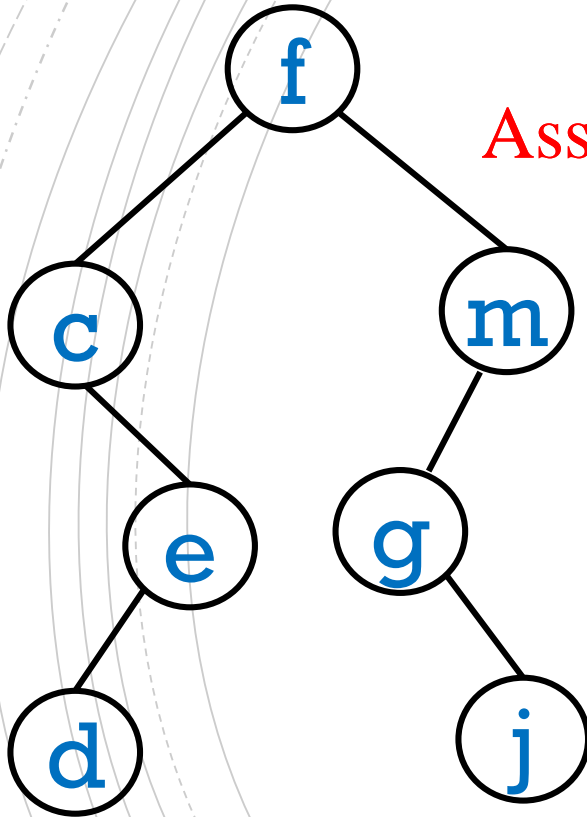
Add WeChat edu\_assist\_pro

uld return the node with

ey. So, for example given

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

## FINDMIN()



Assignment Project Exam Help

<https://eduassistpro.github.io/>

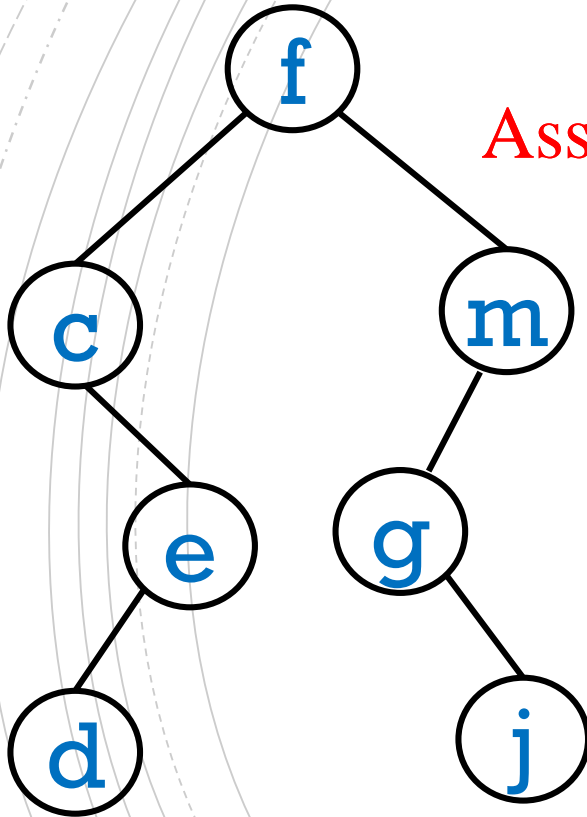
Add WeChat edu\_assist\_pro

uld return the node with

ey. So, for example given

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

## FINDMIN()



Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

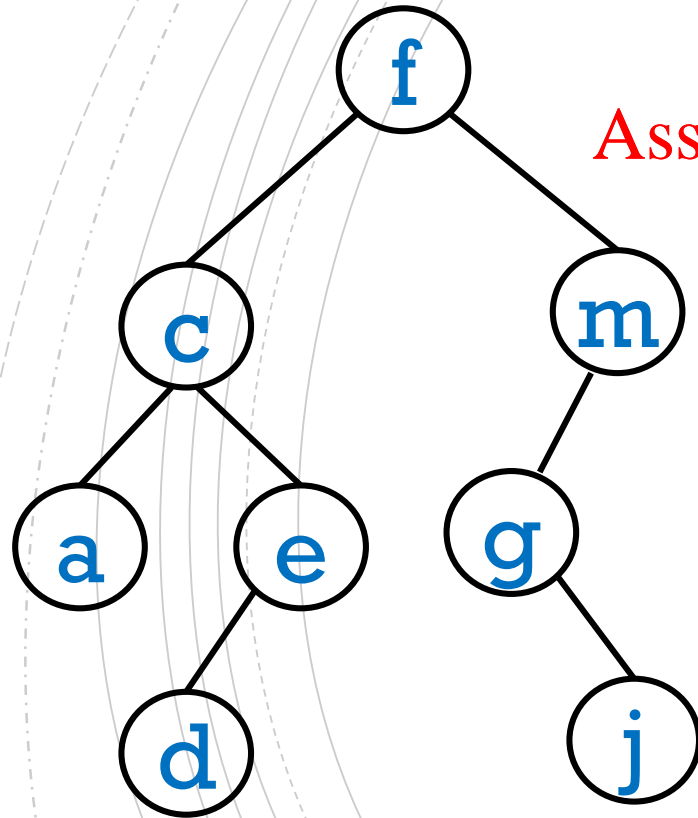
uld return the node with

ey. So, for example given

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



## FINDMIN() - IMPLEMENTATION



**Assignment Project Exam Help**

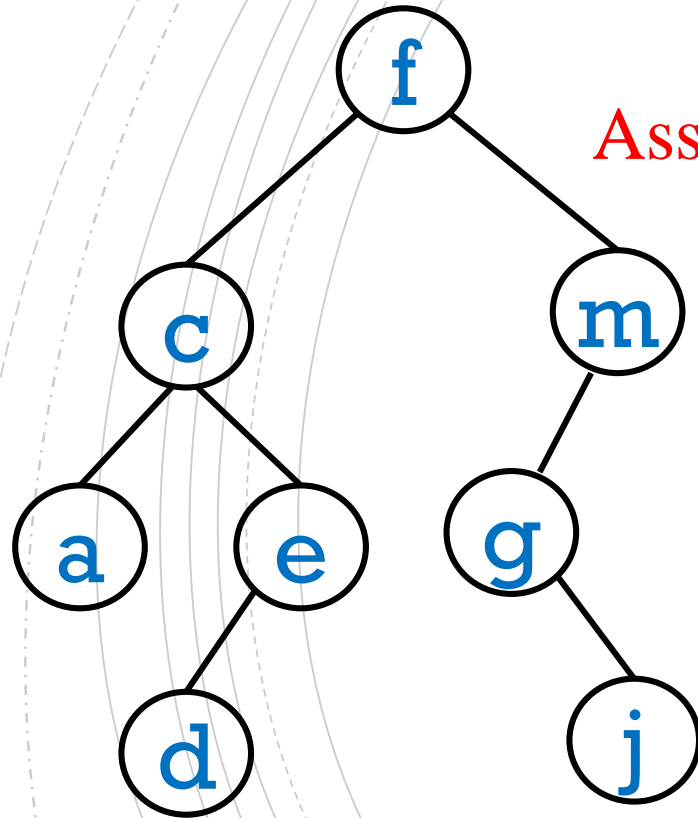
```
findMin(root){ // returns a node  
    if (root == null)
```

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

```
}
```

## FINDMIN() - IMPLEMENTATION



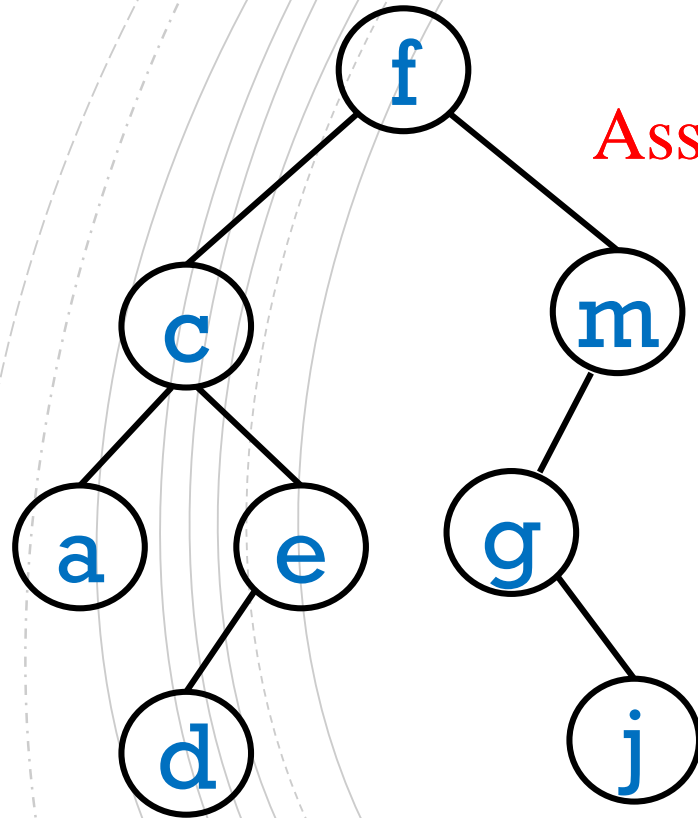
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

```
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://eduassistpro.github.io/>

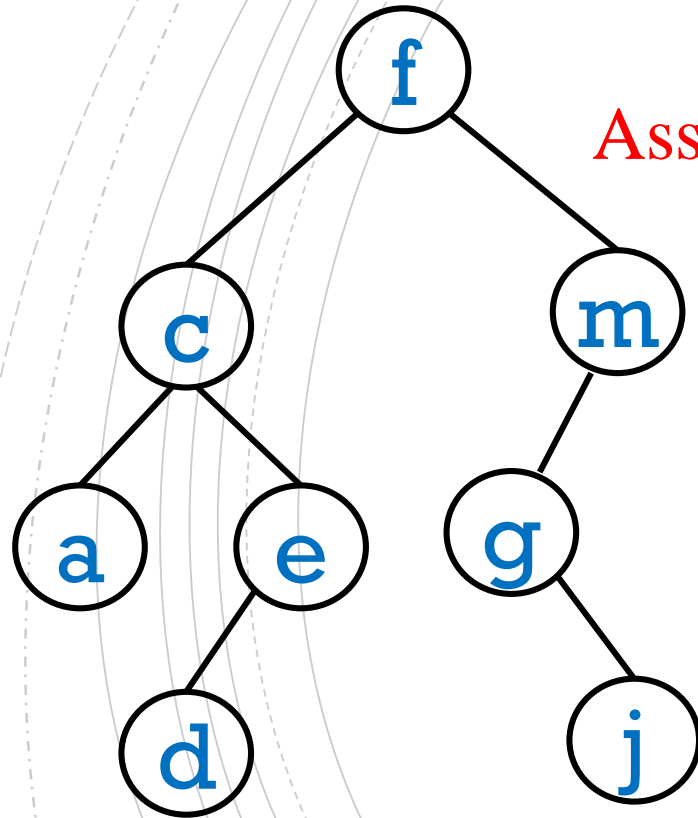
Add WeChat edu\_assist\_pro

uld return the node with

ey. So, for example given

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

## FINDMAX()



Assignment Project Exam Help

<https://eduassistpro.github.io/>

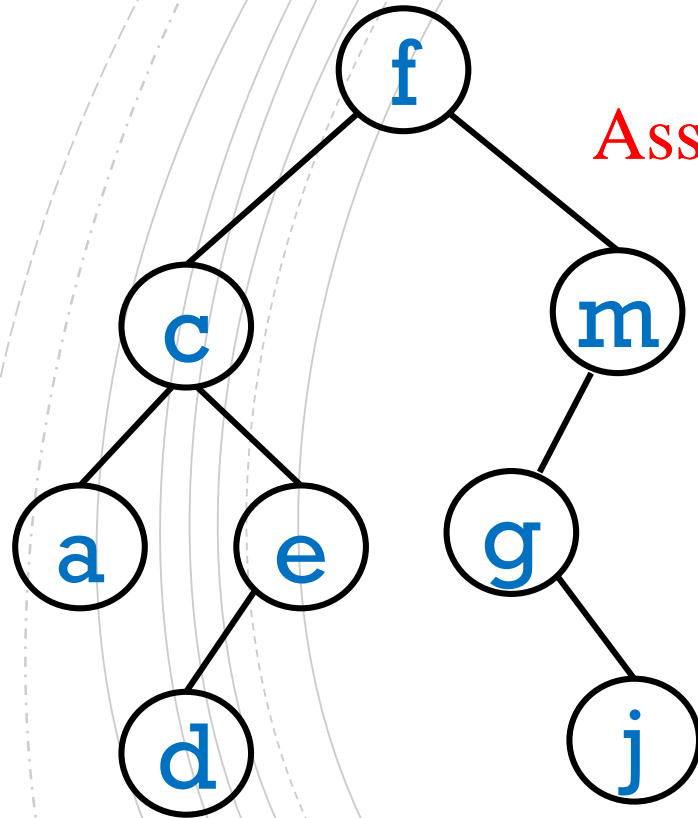
Add WeChat edu\_assist\_pro

uld return the node with

ey. So, for example given

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

## FINDMAX() – IMPLEMENTATION



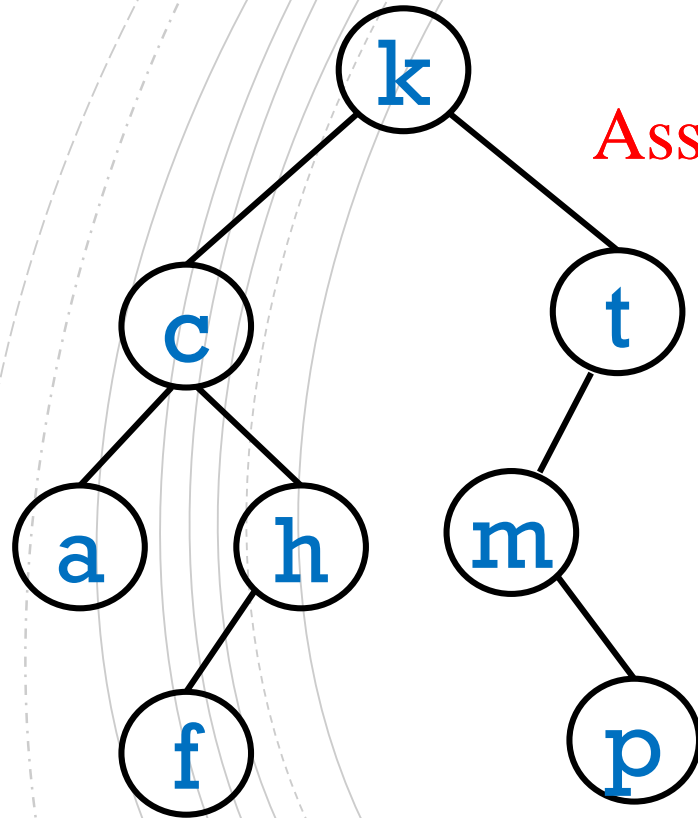
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

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

ADD()



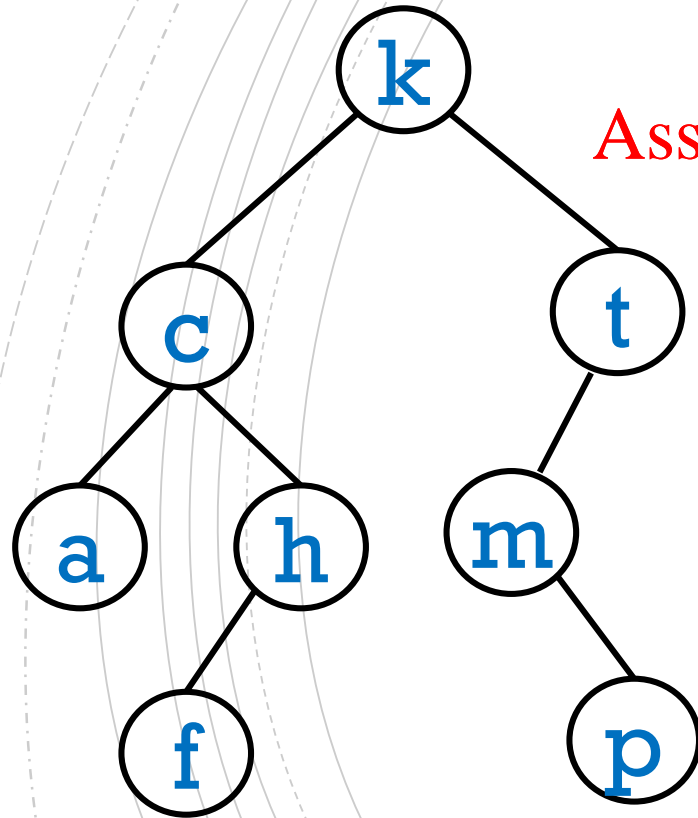
Assignment Project Exam Help

Id add a BSTNode to the tree.

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

ADD()



Assignment Project Exam Help

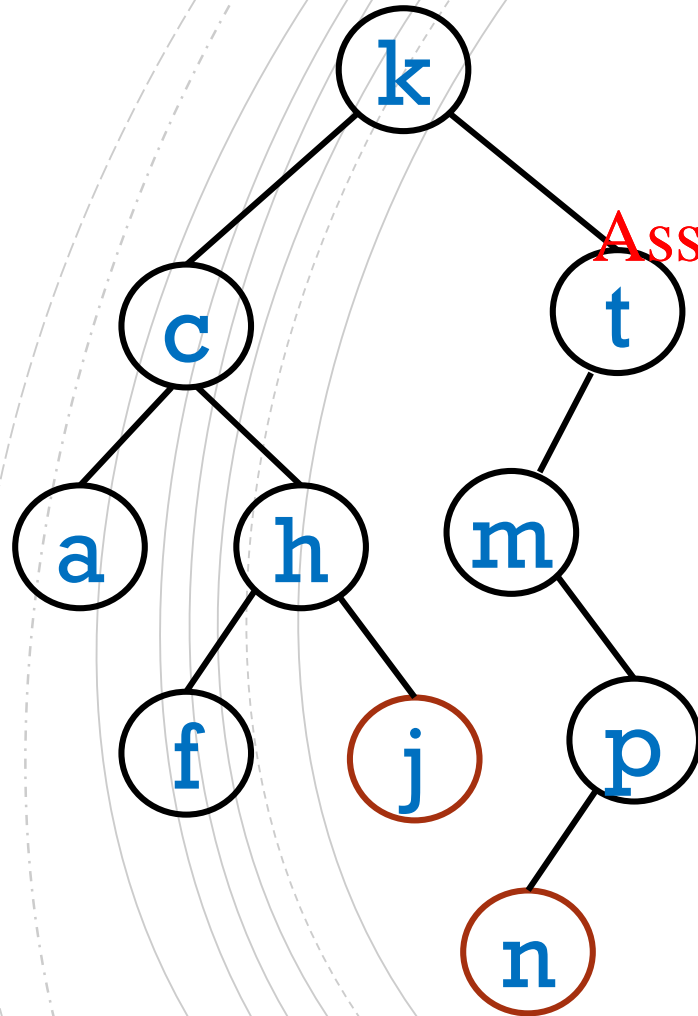
Id add a BSTNode to the tree.

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

A new node is always a leaf.

ADD()



Assignment Project Exam Help

uld add a BSTNode to the

<https://eduassistpro.github.io/>

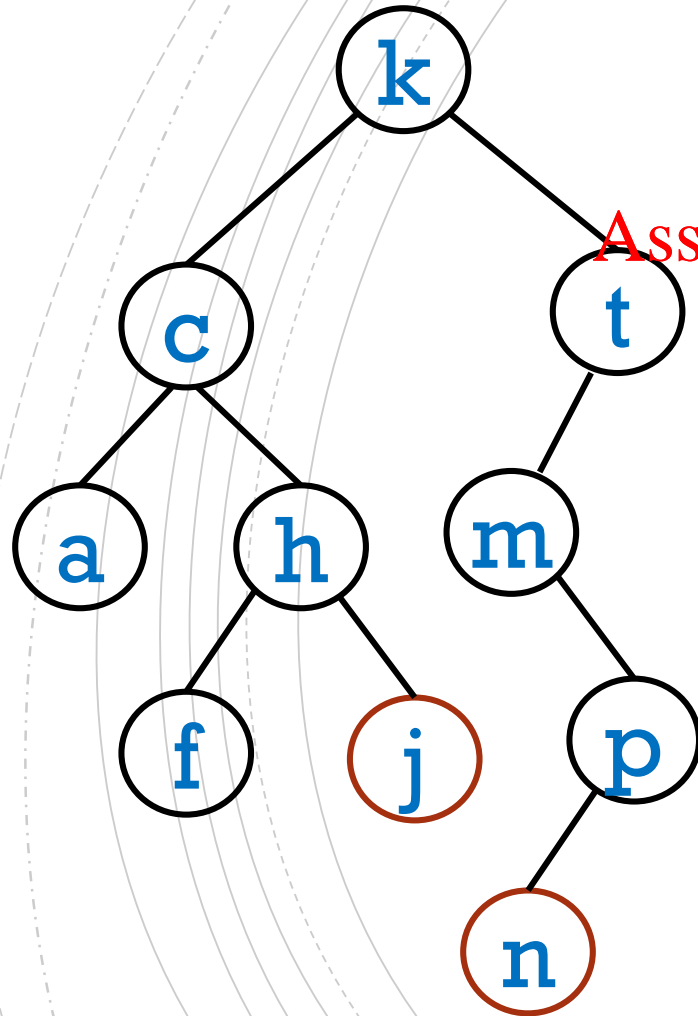
Add WeChat edu\_assist\_pro

- add(**n**) ?

A new node is always a leaf.



## ADD() - IMPLEMENTATION



Assignment Project Exam Help

<https://eduassistpro.github.io/>

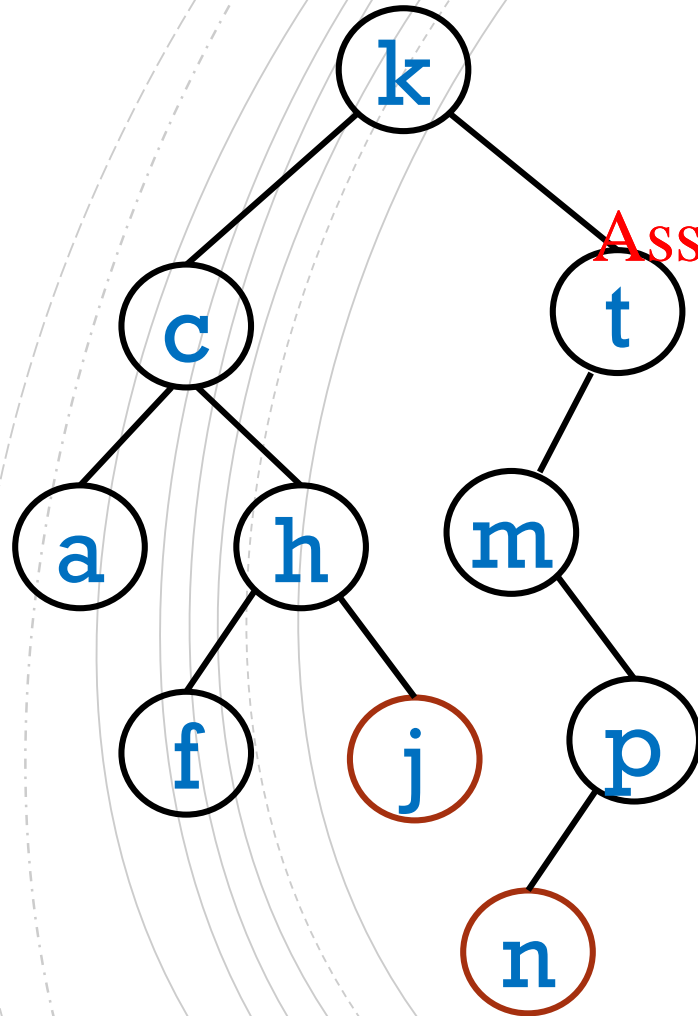
Add WeChat edu\_assist\_pro

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

```
    return root
```

```
}
```

## ADD() - IMPLEMENTATION



Assignment Project Exam Help

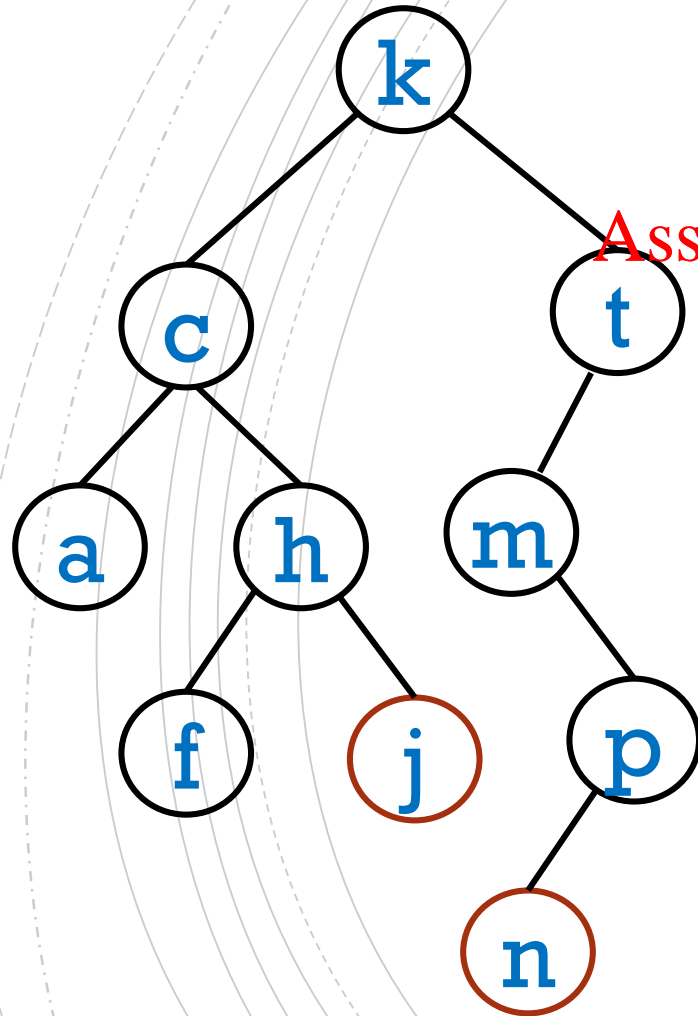
<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

```
add(root, key) { // returns root node
    null)
    BSTNode(key)

    return root
}
```

## ADD() - IMPLEMENTATION



Assignment Project Exam Help

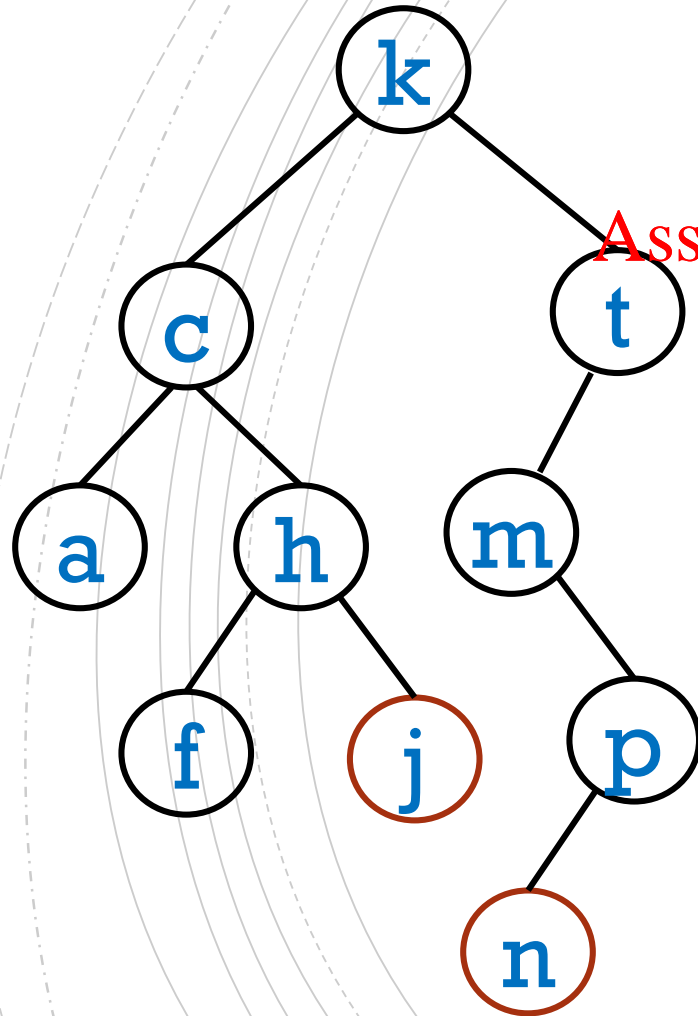
<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

```
add(root, key) { // returns root node
    null)
    BSTNode(key)
    root.key) {
        add(root.left, key)

    return root
}
```

## ADD() - IMPLEMENTATION



Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

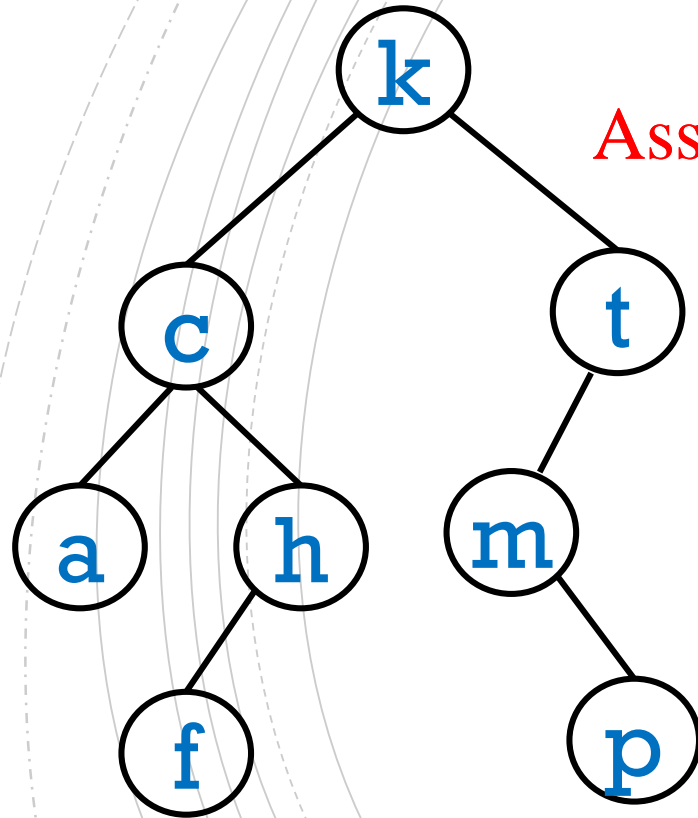
```
add(root, key) { // returns root node
    null)
    BSTNode(key)
    else if (key < root.key) {
        root.left = add(root.left, key)
    }
    else if (key > root.key) {
        root.right = add(root.right, key)
    }
    return root
}
```

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

A: Nothing!

REMOVE()

remove(**c**) →



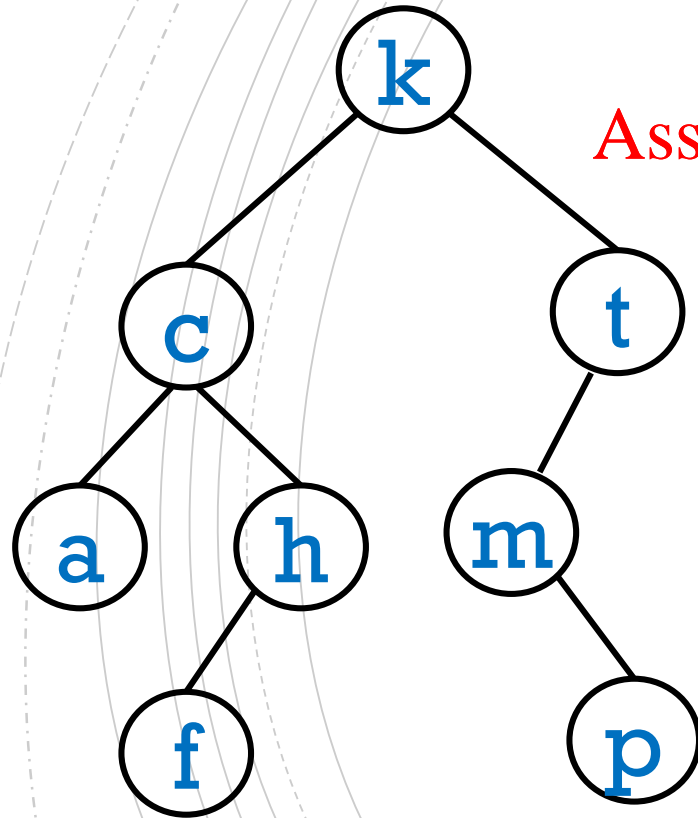
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

## REMOVE()

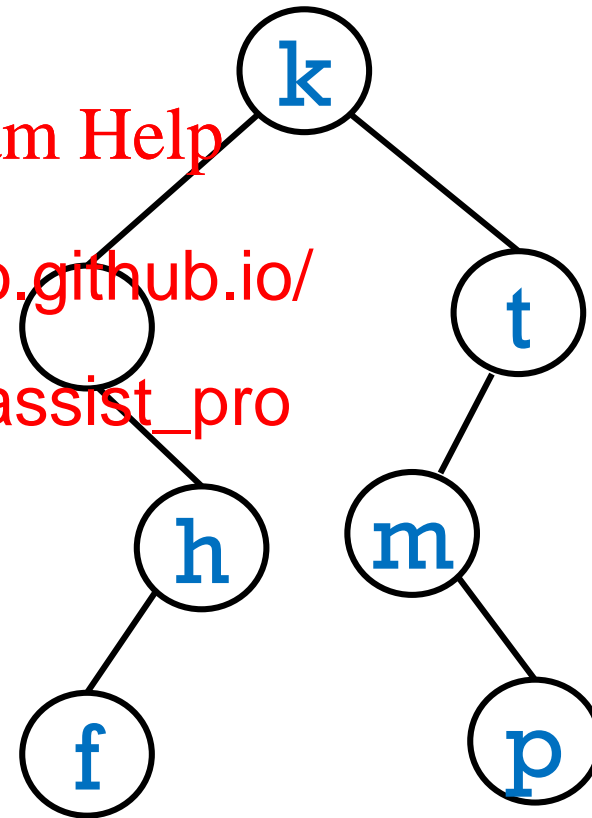
remove(**c**) → this is one way to do it



Assignment Project Exam Help

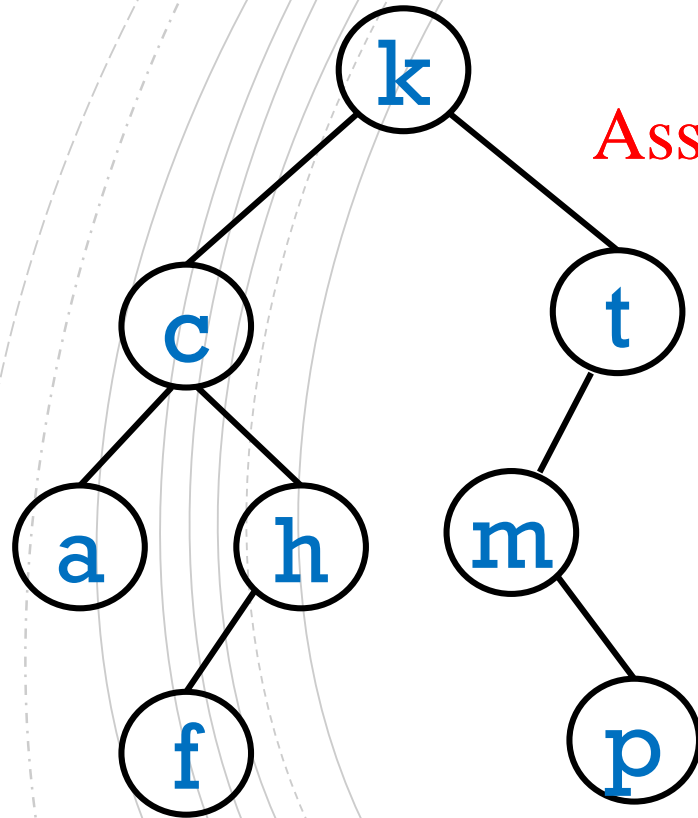
<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro



## REMOVE()

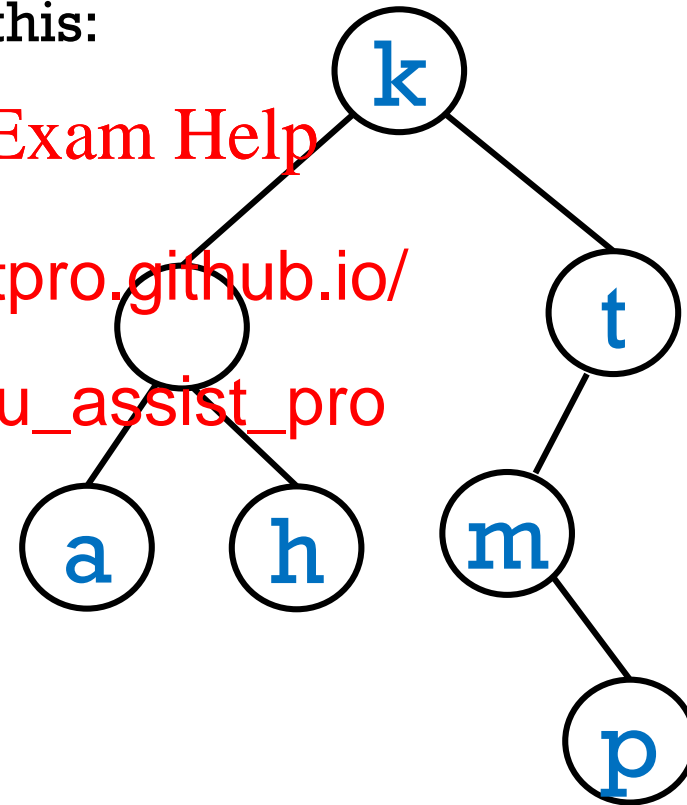
remove(**c**) → the following algorithm  
does this:



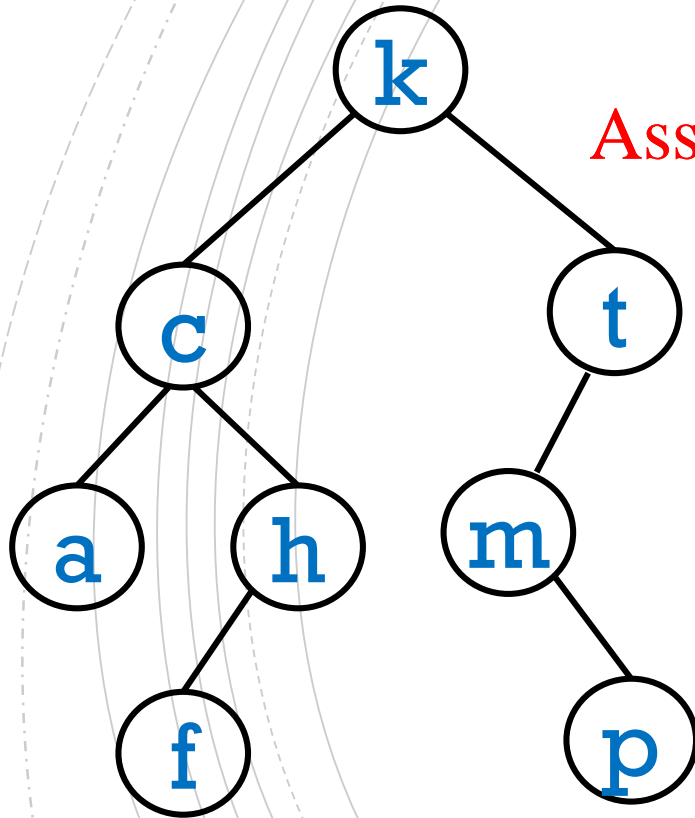
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro



## REMOVE() - IMPLEMENTATION



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

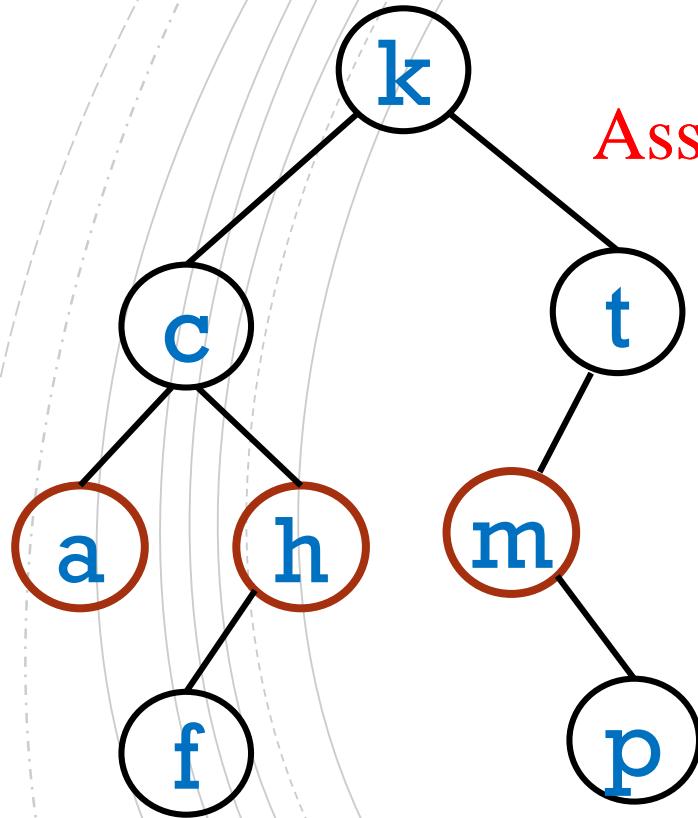
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro



## REMOVE() - IMPLEMENTATION



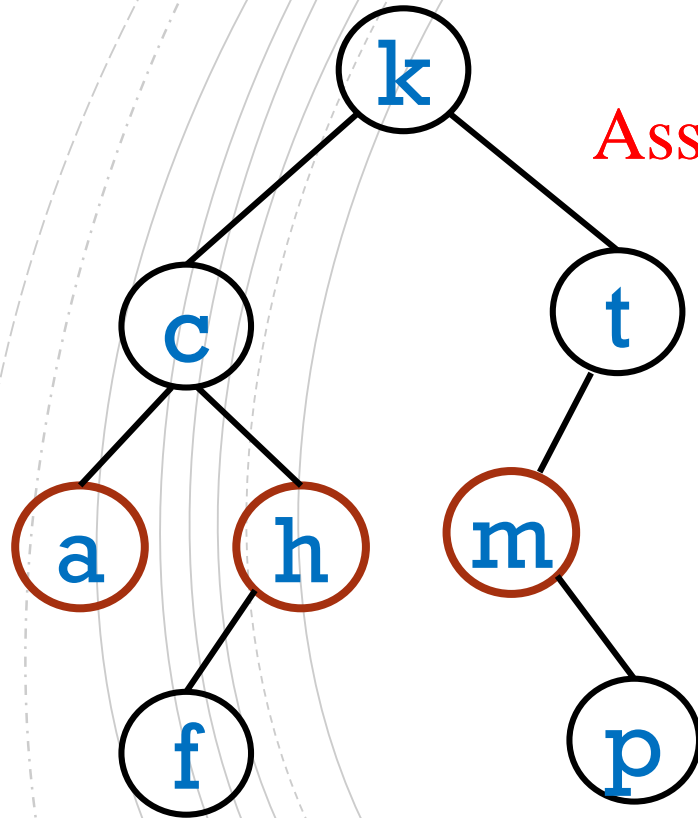
```
remove(root, key){ // returns root node
    if( root == null )
        return null
    else if ( key < root.key )
        root.left = remove(root.left, key)
    else if ( key > root.key )
        root.right = remove(root.right, key)
    else
        // Node to be removed is root
        // Implement logic to replace root with its in-order successor
    }
    return root
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

## REMOVE() - IMPLEMENTATION



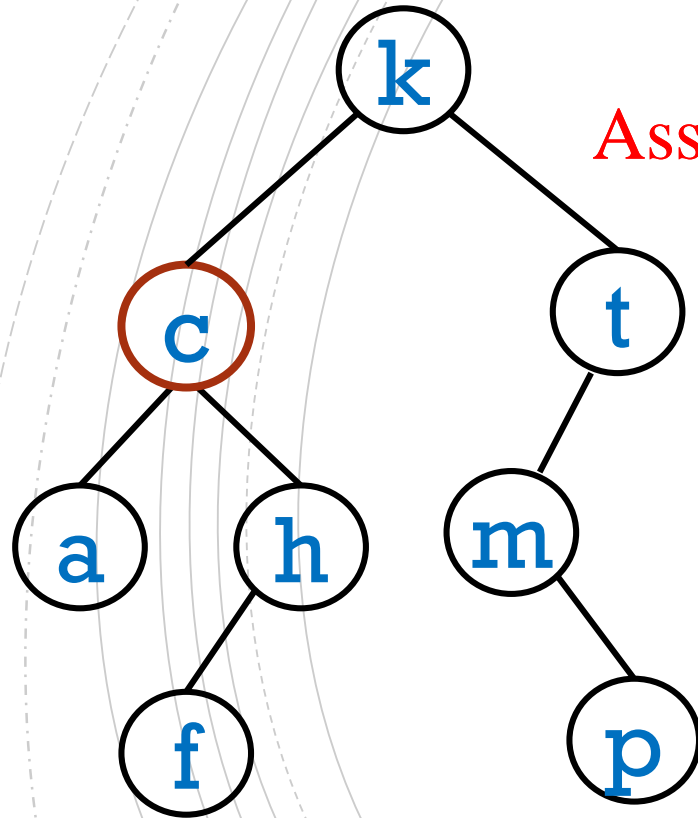
```
remove(root, key){ // returns root node
    if( root == null )
        return null
    else if ( key < root.key )
        root.left = remove(root.left, key)
    else if ( key > root.key )
        root.right = remove(root.right, key)
    else if ( key == root.key )
        if ( root.left == null )
            return root.right
        else if ( root.right == null )
            return root.left
        else
            // find the in-order successor (smallest node in the right subtree)
            t = root.right
            while ( t.left != null )
                t = t.left
            root.key = t.key
            root.left = remove(root.left, key)
            root.right = remove(t, key)
    }
    return root
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

## REMOVE() - IMPLEMENTATION



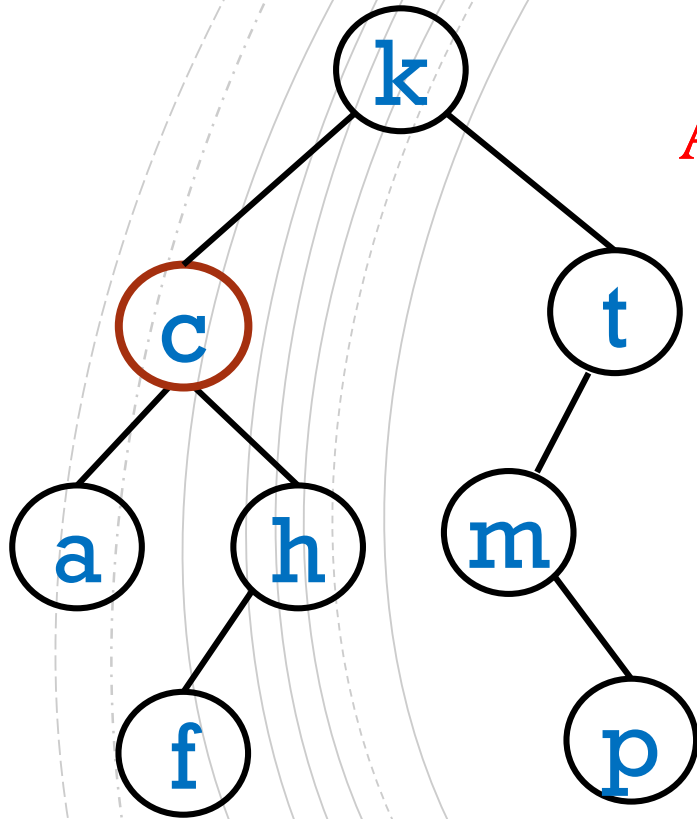
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

```
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 ( key == root.key )
        if ( root.left == null )
            return root.right
        else if ( root.right == null )
            return root.left
        else
            // Node has both left and right children
            // Find the in-order successor (smallest node in the right subtree)
            let temp = root.right
            while (temp.left != null)
                temp = temp.left
            root.key = temp.key
            root.right = remove (temp, temp.key)
    }
    return root
}
```

## 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 if ( root.key == key )
        if ( root.left == null )
            return root.right
        else if ( root.right == null )
            return root.left
        else {
            root.key = findMin(root.right).key
            root.right = remove(root.right, root.key)
        }
    return root
}
```

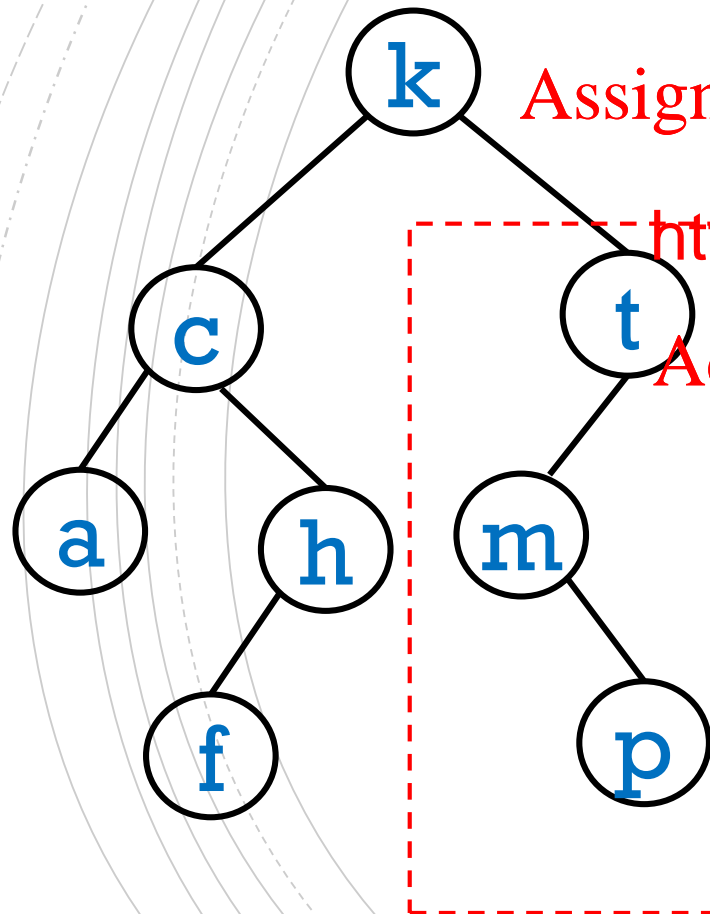
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

## REMOVE() - EXAMPLE

remove(**k**) →



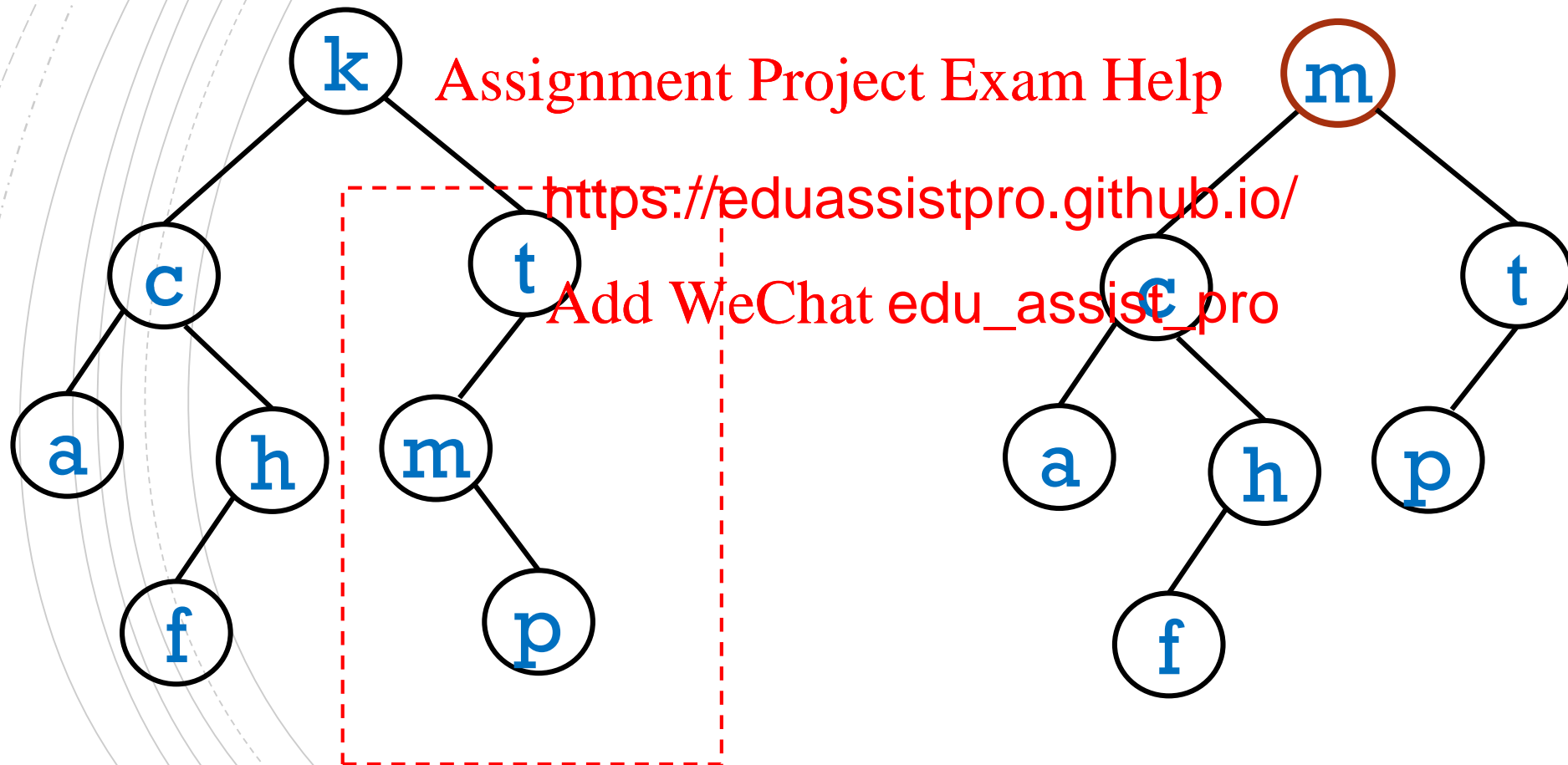
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

## REMOVE() - EXAMPLE

remove(**k**) →

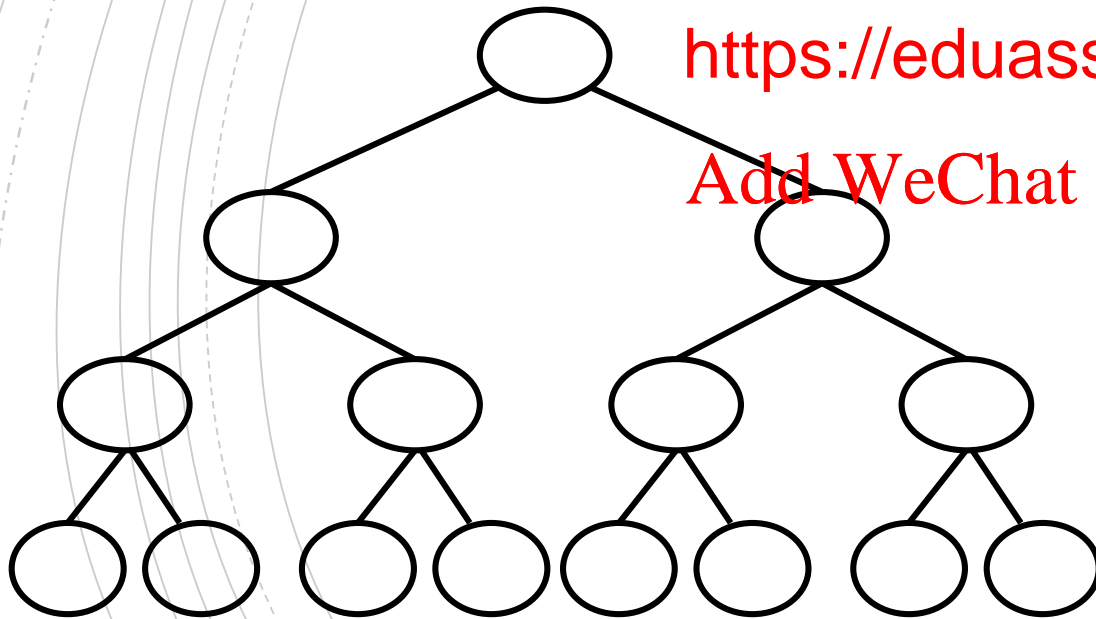


## BALANCED VS UNBALANCED

balanced

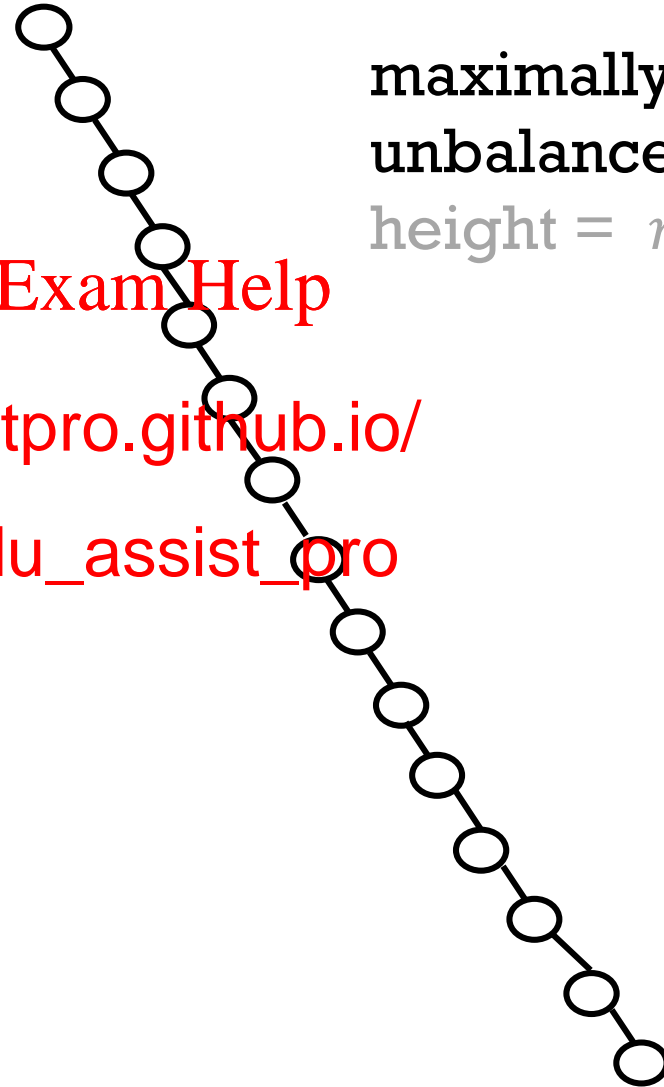
$$\text{height} = \log(n + 1) - 1$$

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



maximally  
unbalanced

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



Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

## BEST VS WORST CASE SCENARIO

best case

worst case

**findMin()** Assignment Project Exam Help

**findMax()**

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

**find( key )**

**add(key)**

**remove(key)**

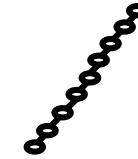


# BEST VS WORST CASE SCENARIO

best case

worst case

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



findMax()

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

find( key )

add(key)

remove(key)

## BEST VS WORST CASE SCENARIO

best case

worst case

findMin()  $O(1)$

findMax()

find( key )

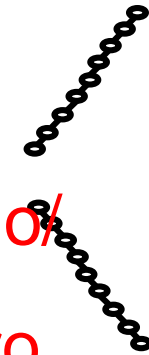
add(key)

remove(key)

[Assignment Project Exam Help](https://eduassistpro.github.io/)

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro



## BEST VS WORST CASE SCENARIO

best case

worst case

findMin()

Assignment Project Exam Help

$O(1)$

$O(n)$

findMax()

<https://eduassistpro.github.io/>

$O(n)$

Add WeChat edu\_assist\_pro

find( key )

$O(1)$

$O(n)$  could be zigzag

add(key)

remove(key)



## BEST VS WORST CASE SCENARIO

best case

worst case

findMin()

[Assignment Project Exam Help](https://eduassistpro.github.io/)

$O(1)$

$O(n)$

findMax()

<https://eduassistpro.github.io/>

$(n)$

Add WeChat edu\_assist\_pro

find( key )

$O(1)$

$O(n)$

add(key)

$O(1)$

$O(n)$

remove(key)

$O(1)$

$O(n)$

} Could  
be  
zigzag

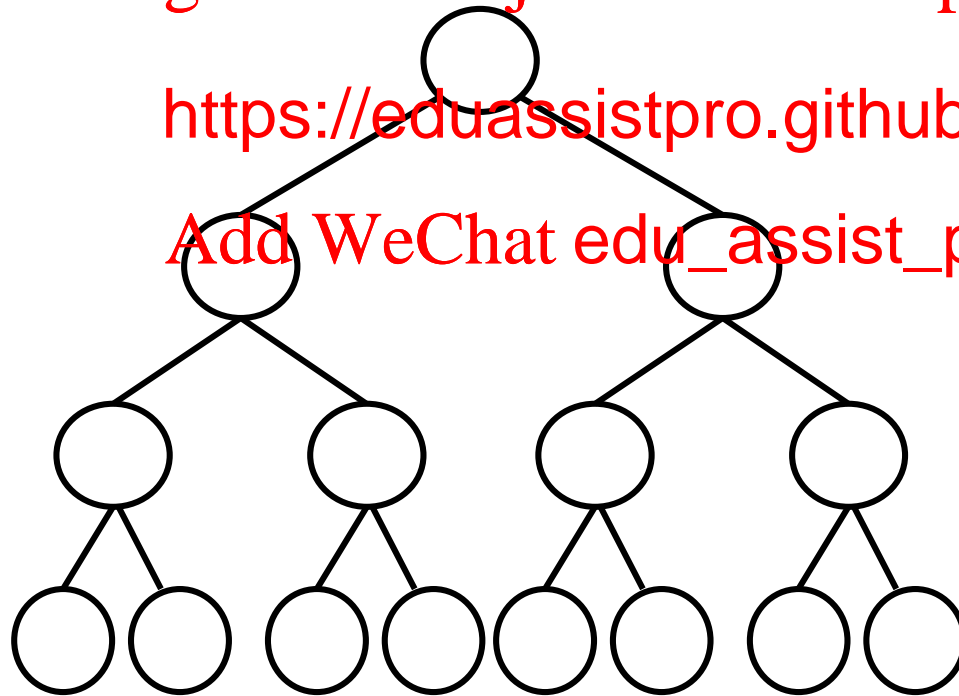
## BINARY SEARCH (TREES)

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

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro



# BALANCED BINARY SEARCH TREES

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

best case

worst case

**findMin()**  $O(\log n)$  **Assignment Project Exam Help**

**findMax()**  $O(\log n)$

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

**find( key )**  $O(1)$   $O(\log n)$

**add(key)**  $O(\log n)$   $O(\log n)$

**remove(key)**  $O(\log n)$   $O(\log n)$

An orange paint roller with a red handle, positioned horizontally. The roller is partially filled with orange paint, and there are orange paint splatters and drips around it. The text "Coming Soon" is written in white on the orange part of the roller.

# Coming Soon

## Assignment Project Exam Help

In the next

- Heaps <https://eduassistpro.github.io/>  
Add WeChat edu\_assist\_pro