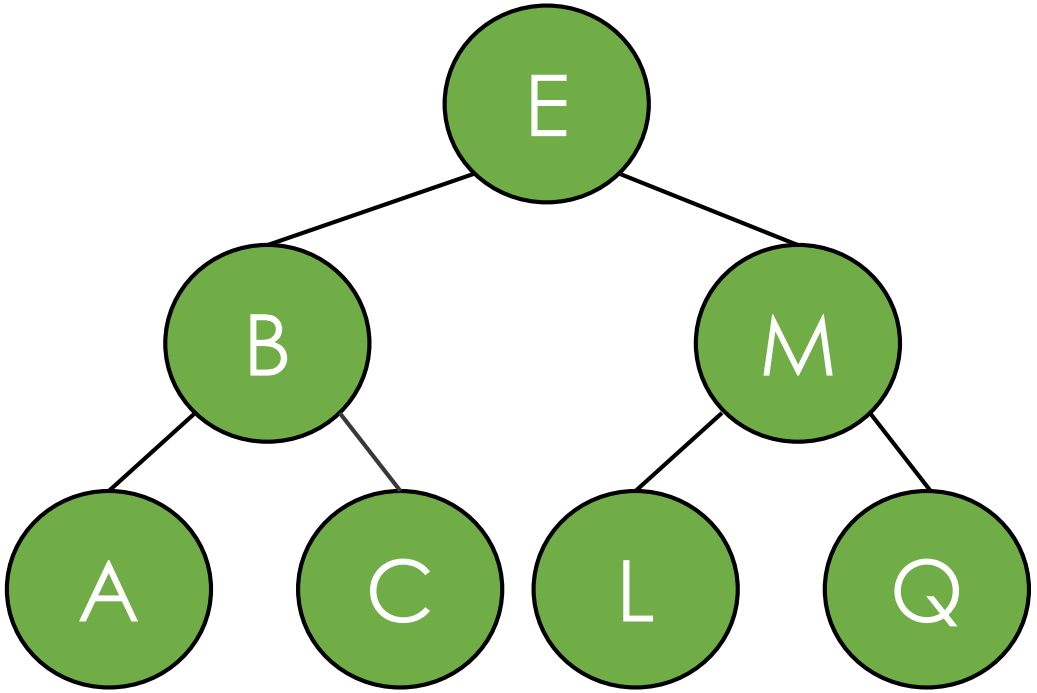
Binary Search Trees: Search

By the end of this video you will be able to…

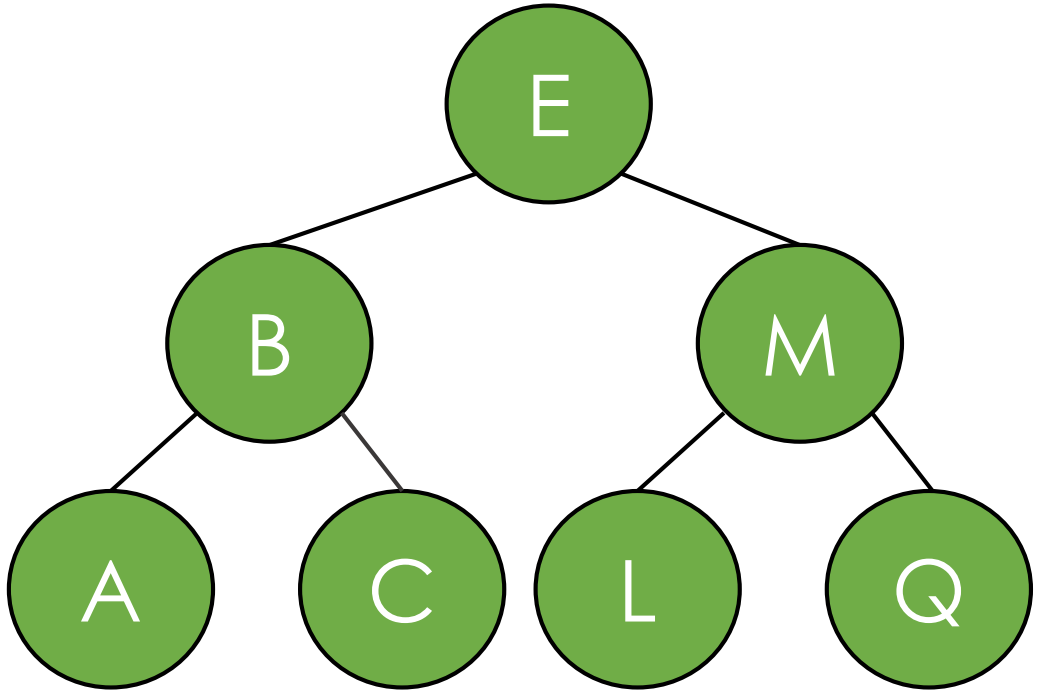
- Perform Search in a Binary Search Tree

Binary Search Tree - Searching



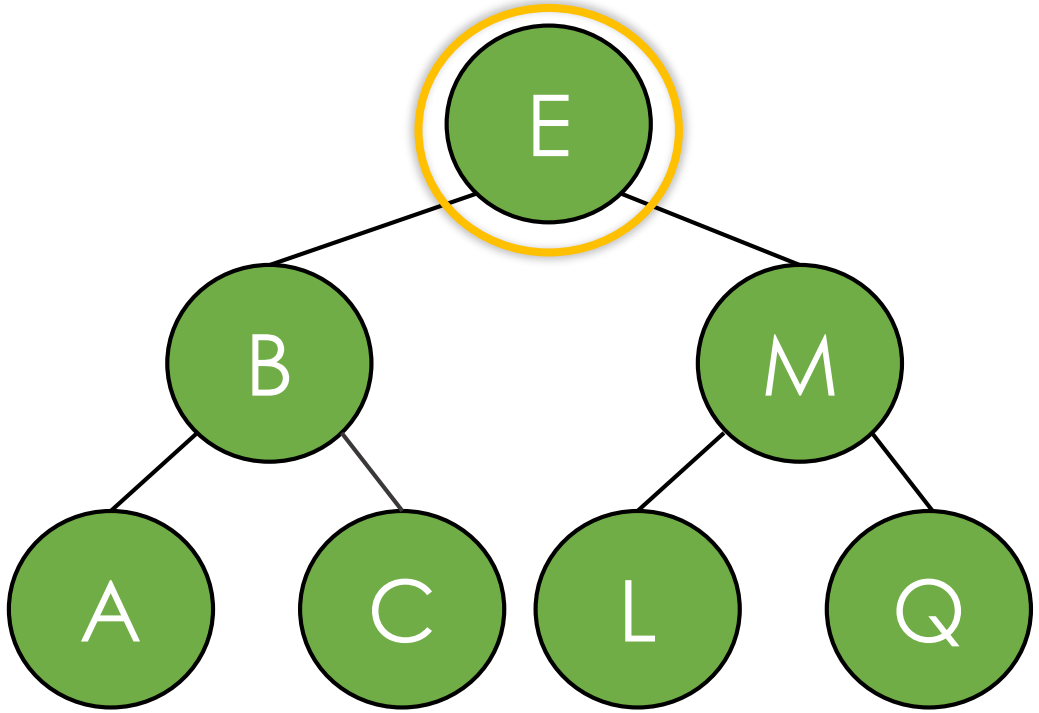
Same fundamental idea as binary search of an array

Binary Search Tree - Searching



`toFind` | C

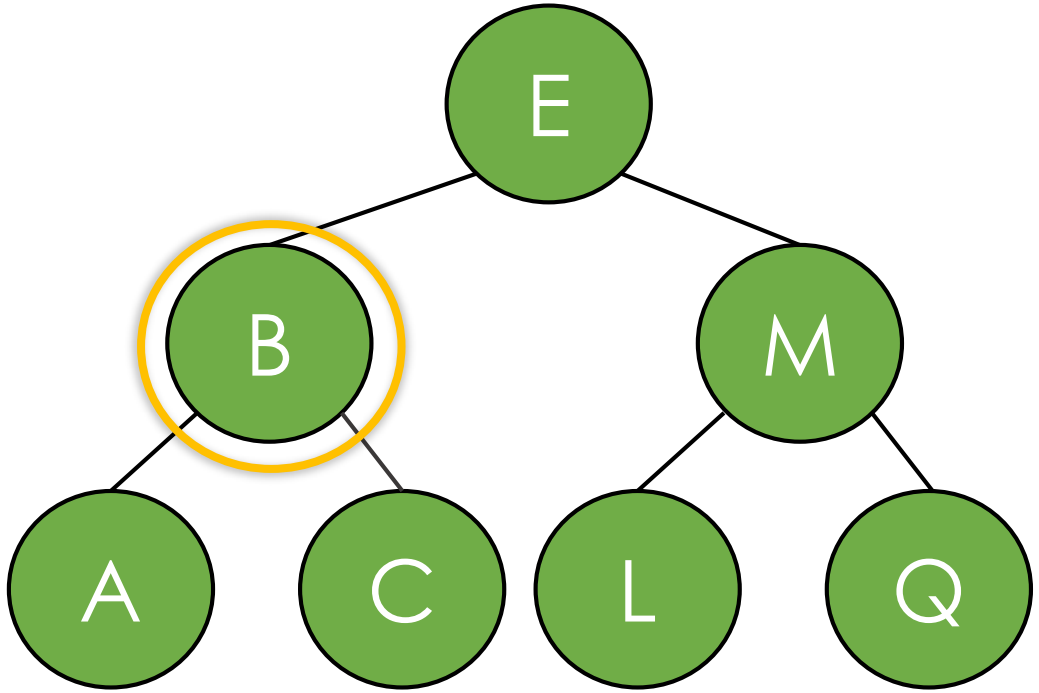
Binary Search Tree - Searching



toFind C

Compare: E and C

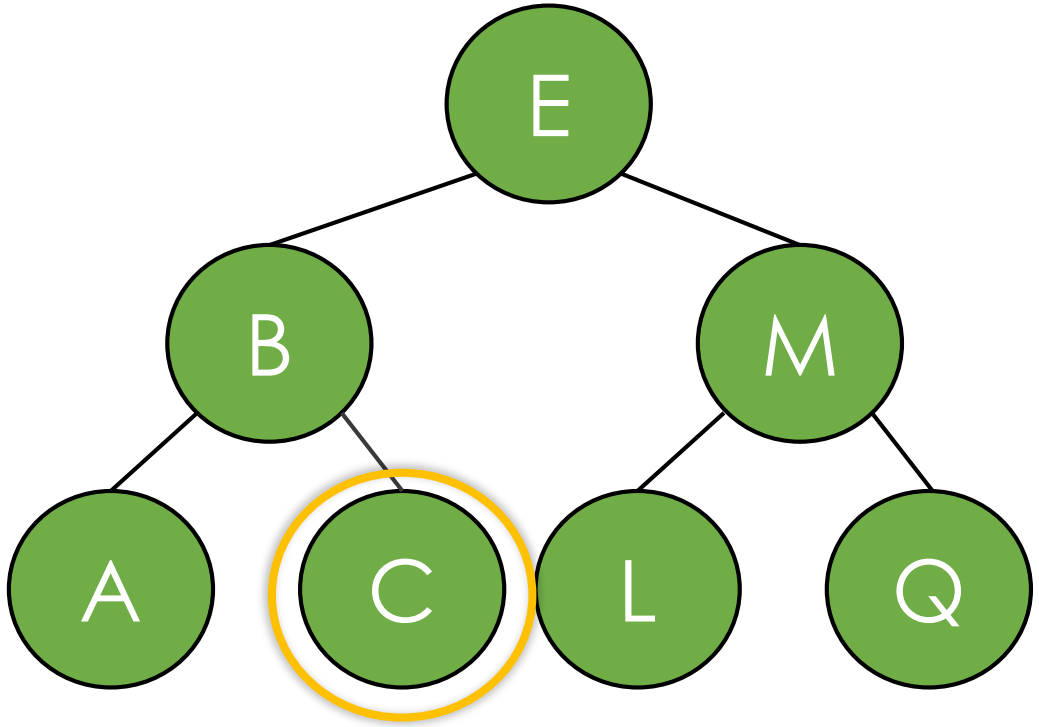
Binary Search Tree - Searching



toFind C

Compare: B and C

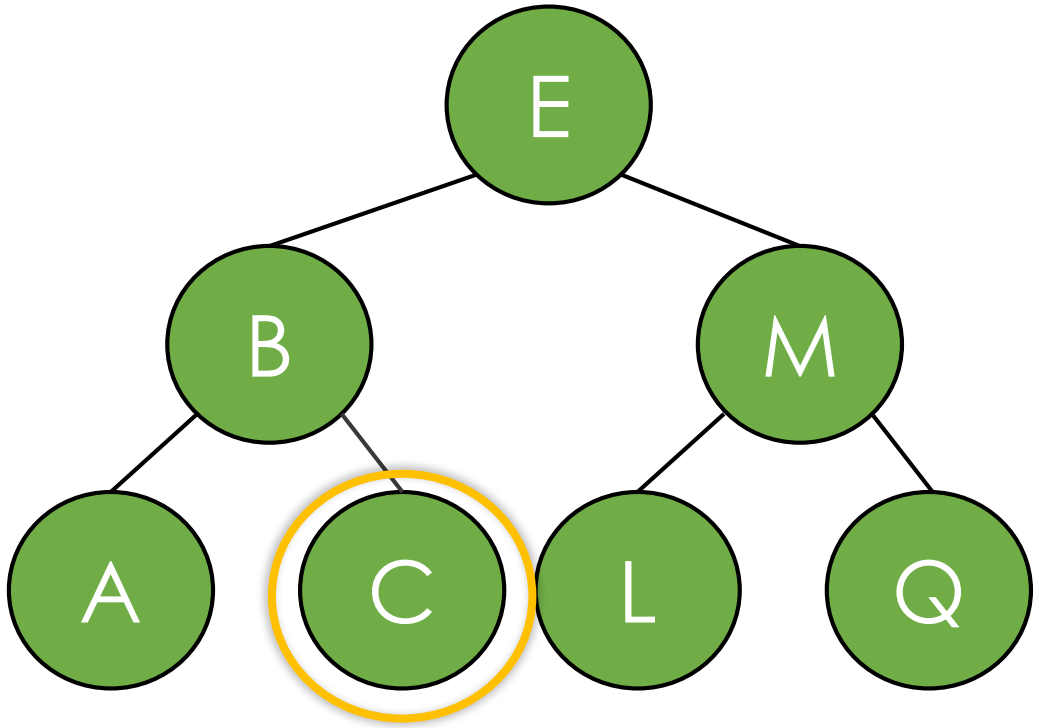
Binary Search Tree - Searching



toFind C

Compare: C and C

Binary Search Tree - Searching

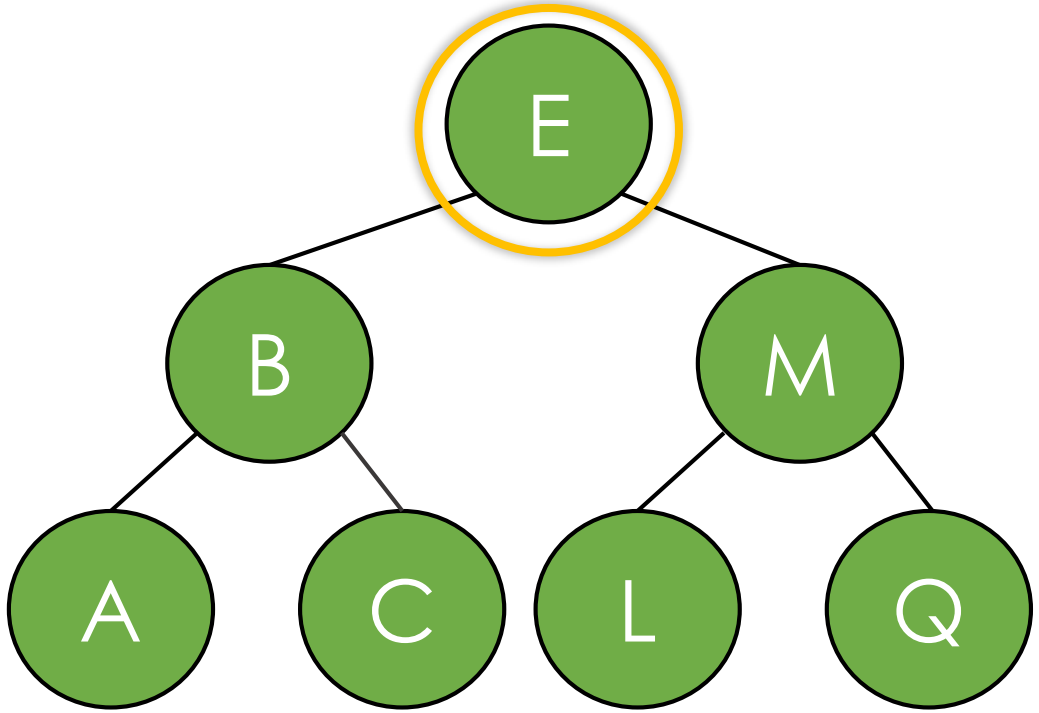


`toFind` C

`Compare: C and C`

Found it!

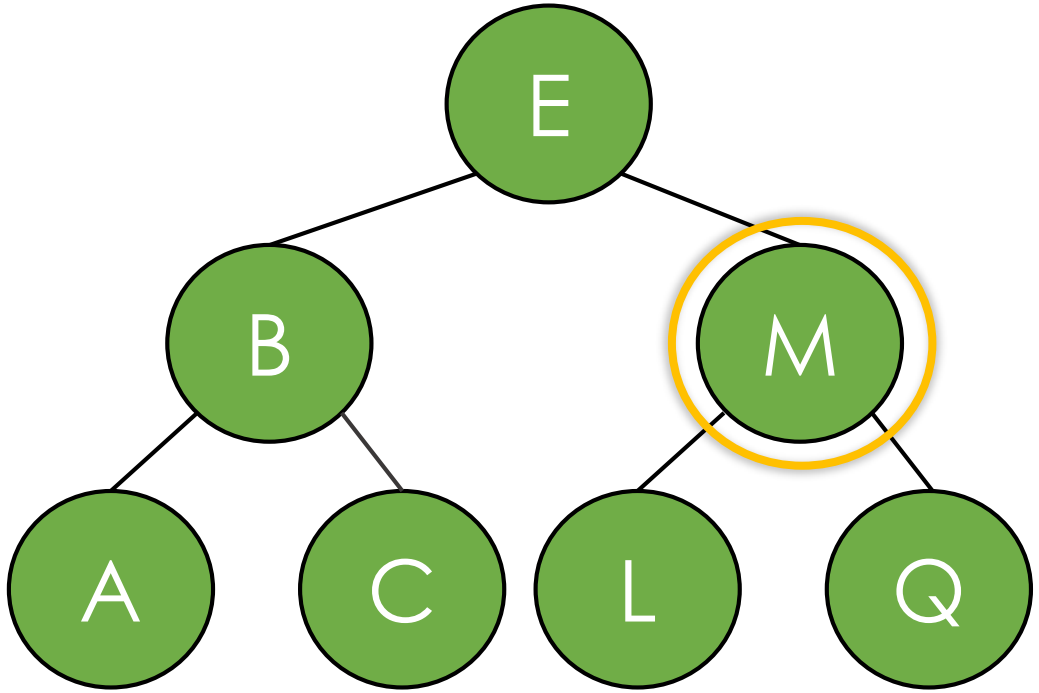
Binary Search Tree - Searching



toFind | P

Compare: E and P

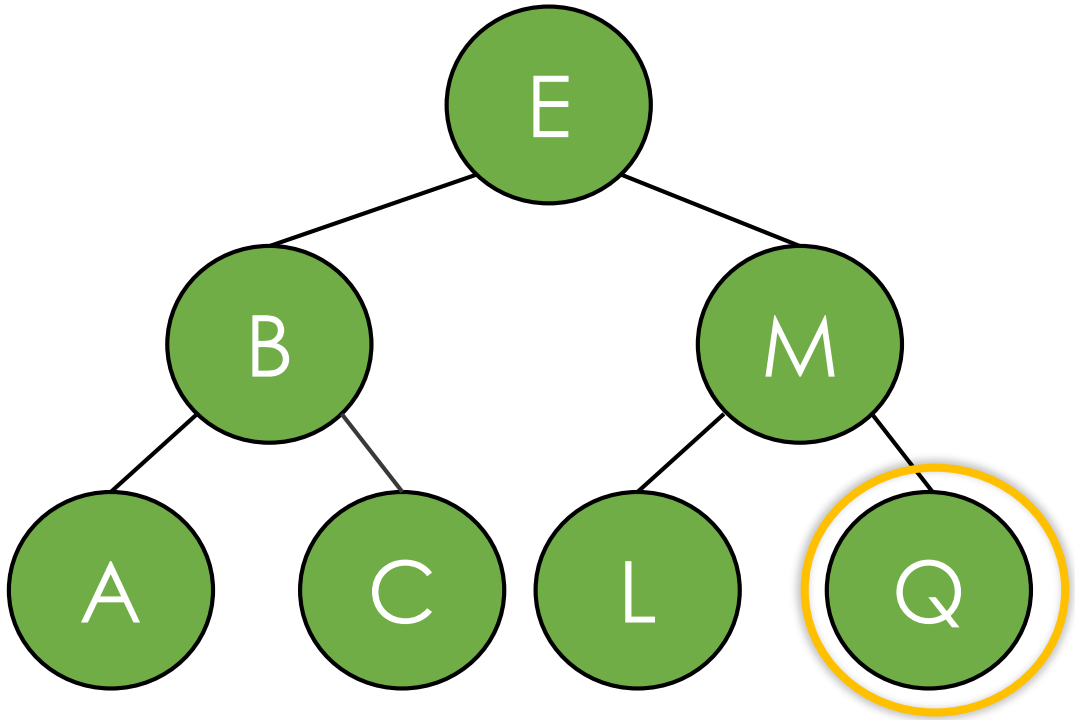
Binary Search Tree - Searching



toFind P

Compare: M and P

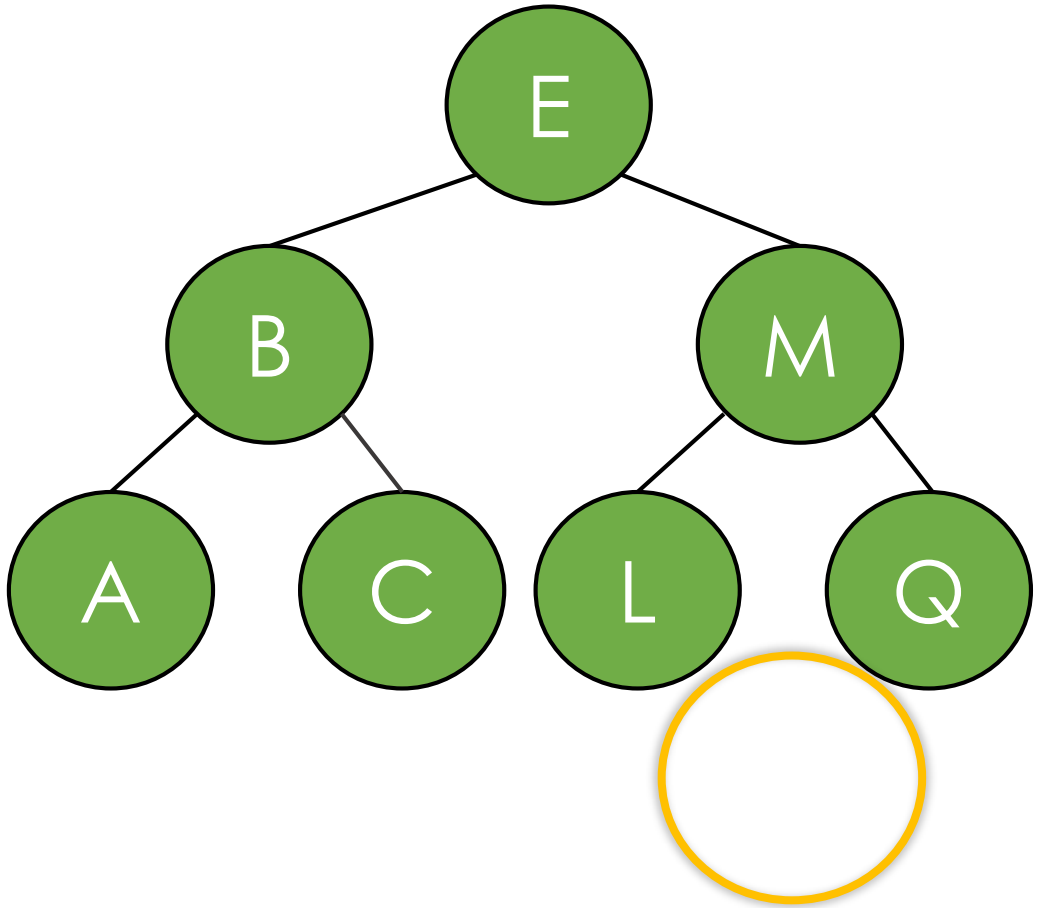
Binary Search Tree - Searching



toFind P

Compare: P and Q

Binary Search Tree - Searching

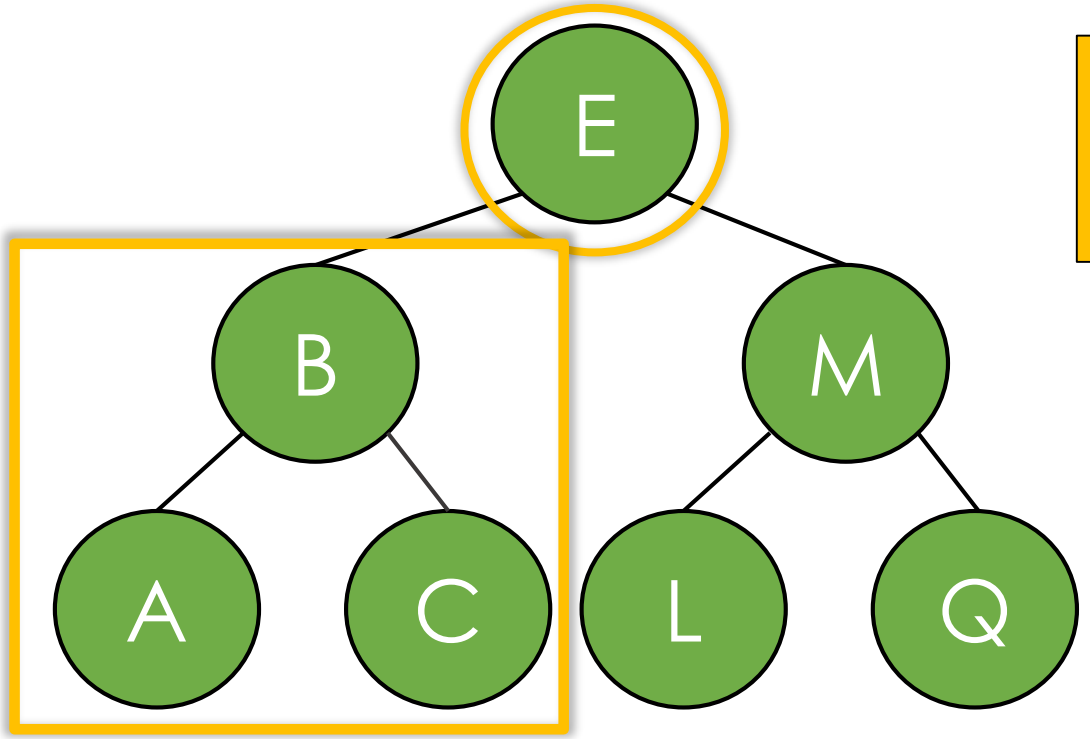


`toFind` `P`

`Node is null`

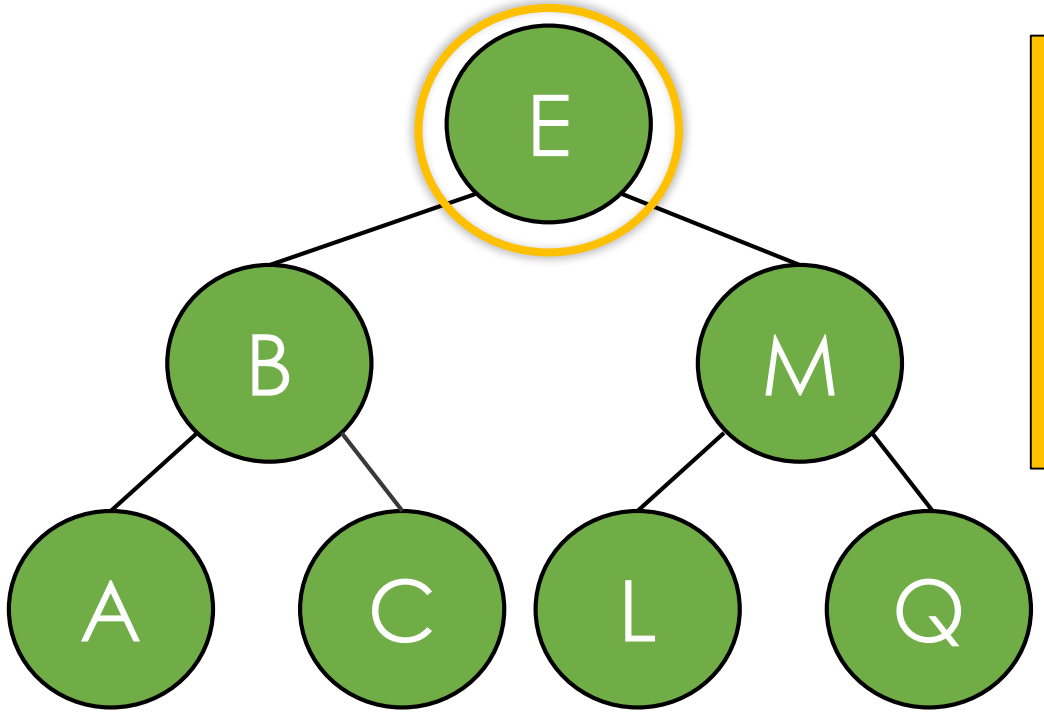
Not Found!

Binary Search Tree - Searching



You could solve this with recursion.

Binary Search Tree - Searching



You could also solve it with iteration by keeping track of your current node.

If you want more help, we walk through this in a support video.

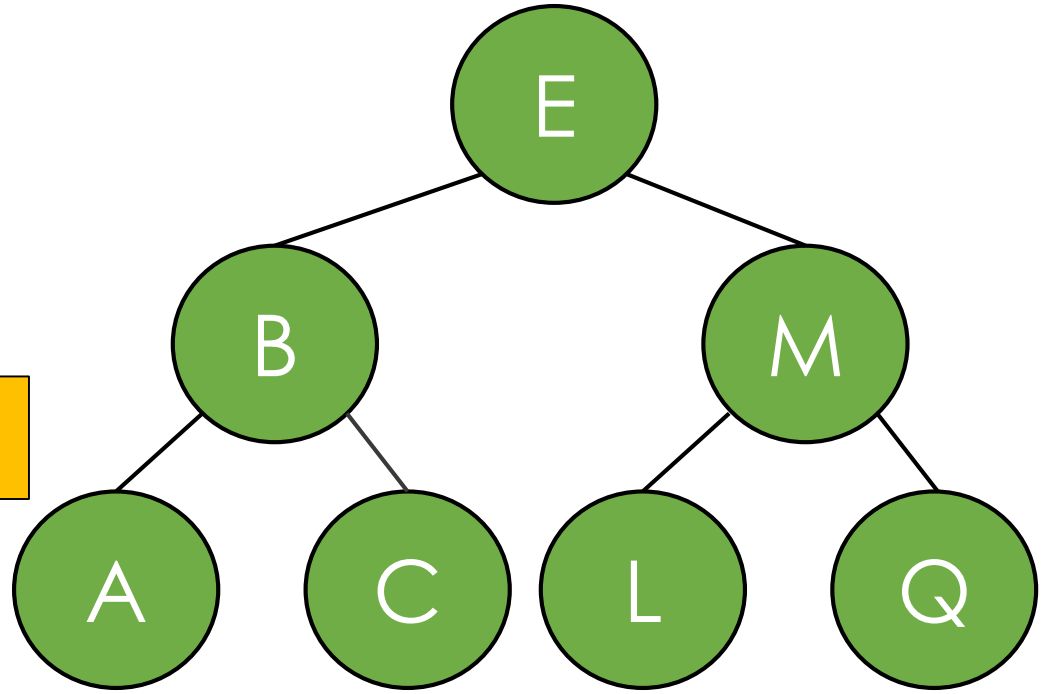
Next step

- Insertion and deletion in a BST

BST: Contains Algorithm

```
// Return if item is in the tree
// rooted at node
contains(item):
  curr = root
  while (curr != null) {
    if (node.val == item)
        return true
    else if (node.val < item)
        curr = node.right
    else // node.val > item
        curr = node.left
```

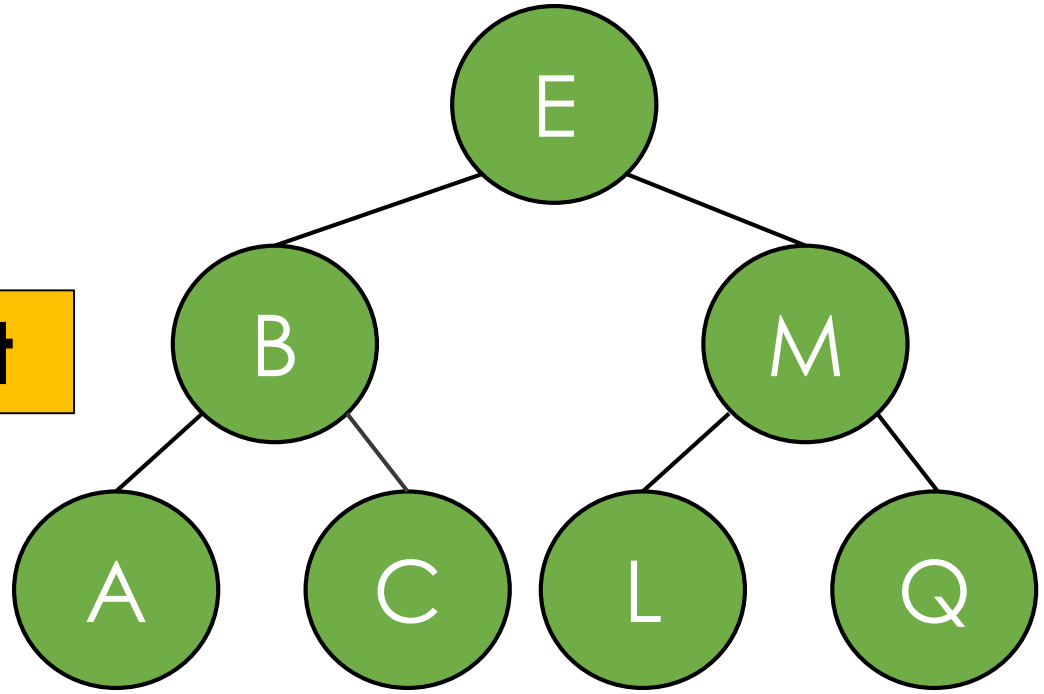
Start at root



BST: Contains Algorithm

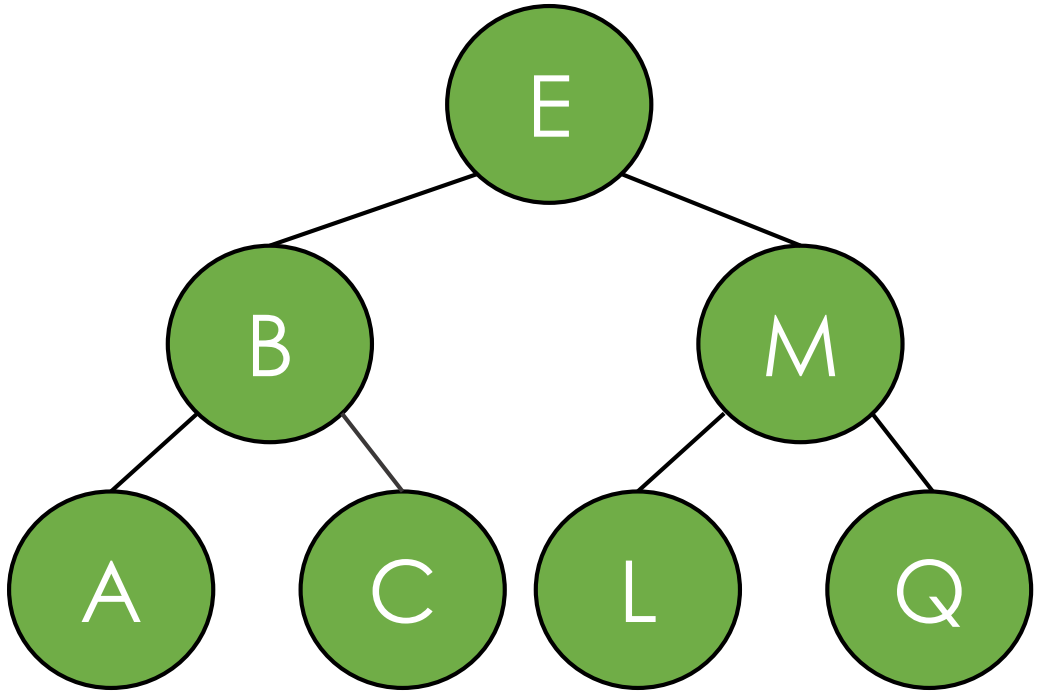
```
// Return if item is in the tree
// rooted at node
contains(item):
  curr = root
  while (curr != null) {
    if (node.val == item)
        return true
    else if (node.val < item)
        curr = node.right
    else // node.val > item
        curr = node.left
  }
  return false
```

Start at root



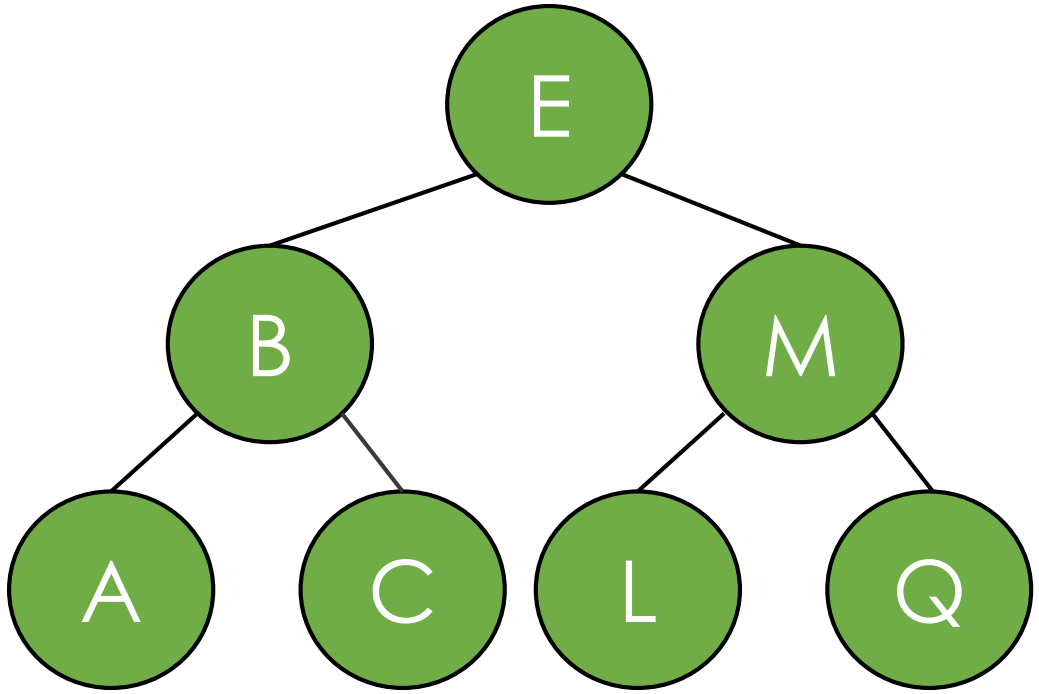
BST: Contains Algorithm

```
// Return if item is in the tree
// rooted at node
contains(item):
  curr = root
  while (curr != null) {
    if (node.val == item)
        return true
    else if (node.val < item)
        curr = node.right
    else // node.val > item
        curr = node.left
```



BST: Contains Algorithm

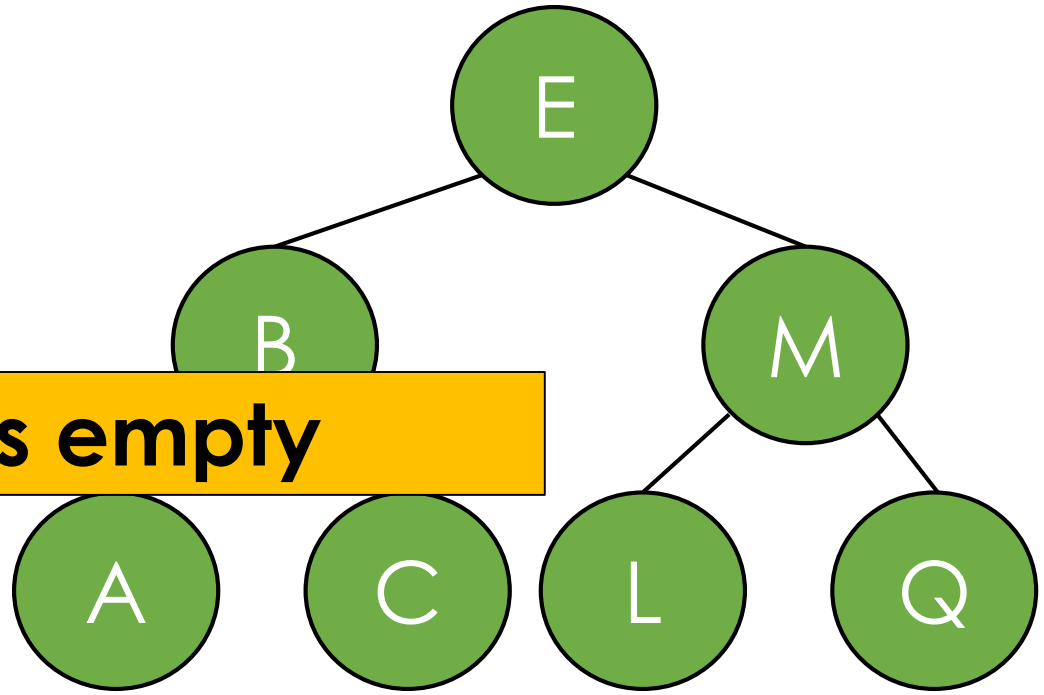
```
// Return if item is in the tree
// rooted at node
contains(item):
  curr = root
  while (curr != null) {
    if (node.val == item)
        return true
    else if (node.val < item)
        curr = node.right
    else // node.val > item
        curr = node.left
```



BST: Contains Algorithm

```
// Return if item is in the tree
// rooted at node
contains(item, node):
  if (node == null)
      return false
  else if (node.val == item)
      return true
  else if (node.val < item)
      return contains(item, node.right)
  else // node.val > item
      return contains(item, node.left)
```

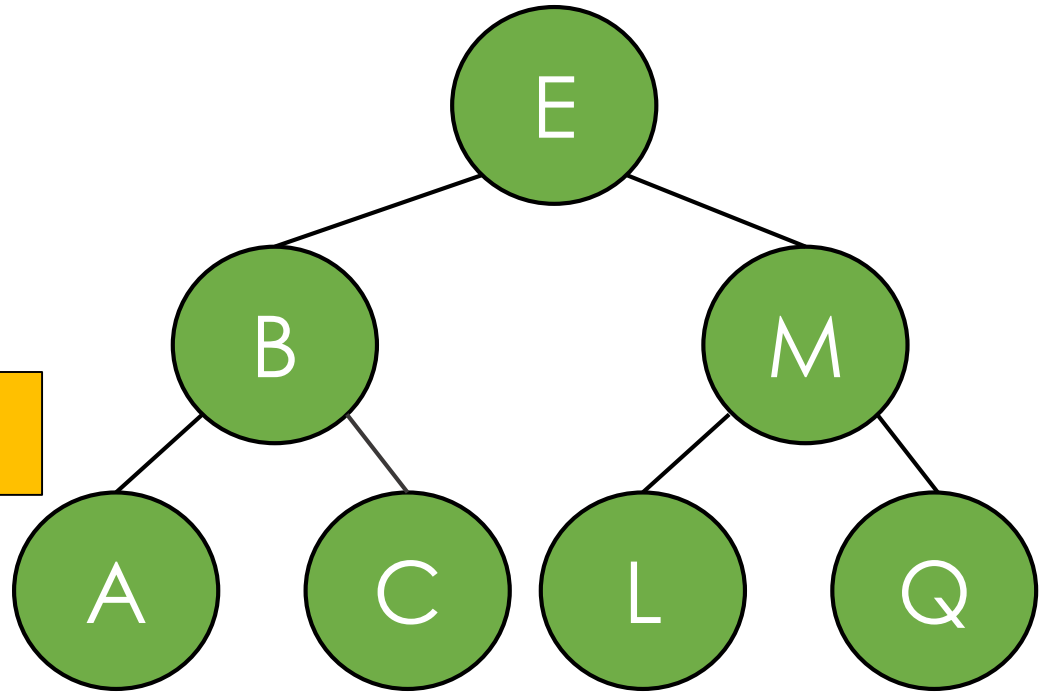
Tree's empty



BST: Contains Algorithm

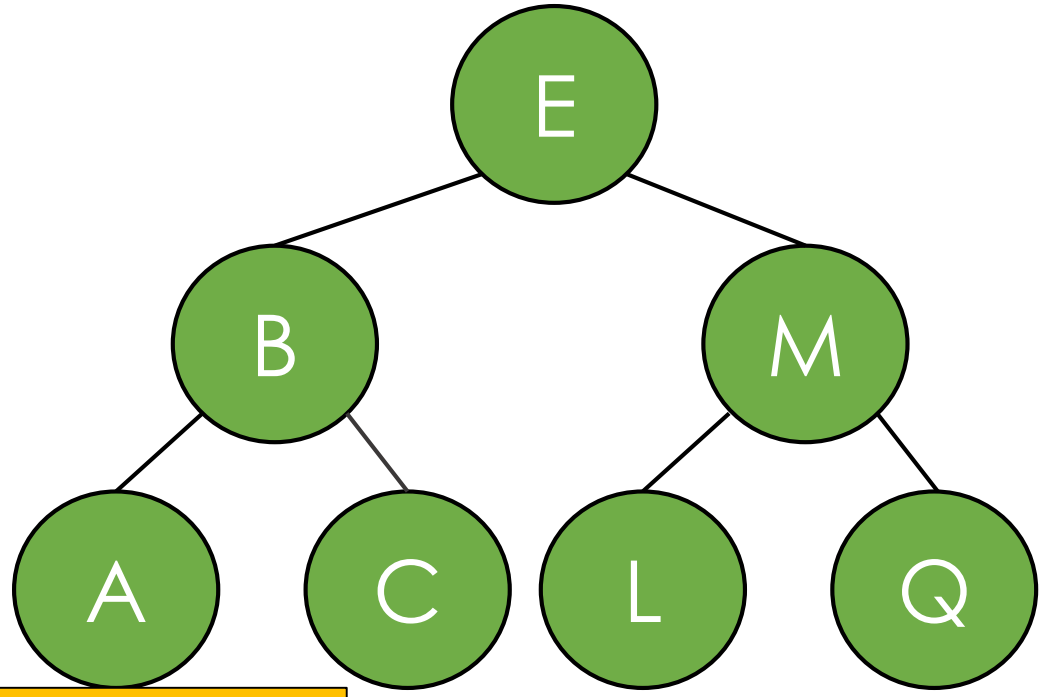
```
// Return if item is in the tree
// rooted at node
contains(item, node):
  if (node == null)
      return false
  else if (node.val == item)
      return true
  else if (node.val < item)
      return contains(item, node.right)
  else // node.val > item
      return contains(item, node.left)
```

Found it!



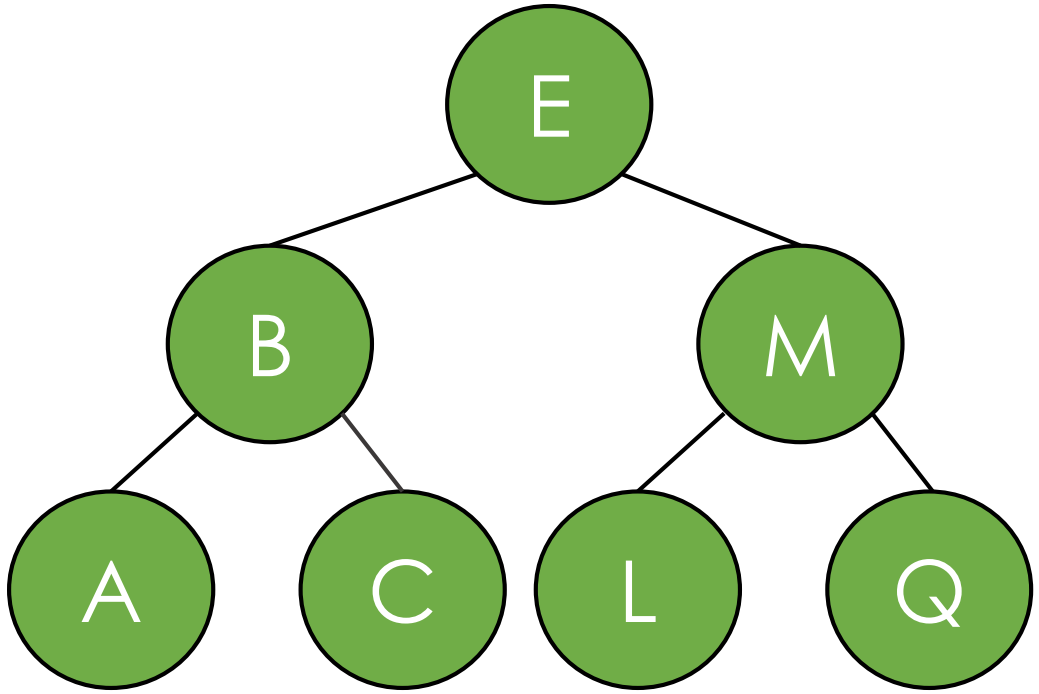
BST: Contains Algorithm

```
// Return if item is in the tree
// rooted at node
contains(item, node):
  if (node == null)
      return false
  else if (node.val == item)
      return true
  else if (node.val < item)
      return contains(item, node.right)
  else // node.val > item
      return contains(item, node.left)
```



BST: Contains Algorithm

```
// Return if item is in the tree
// rooted at node
contains(item, node):
  if (node == null)
      return false
  else if (node.val == item)
      return true
  else if (node.val < item)
      return contains(item, node.right)
  else // node.val > item
      return contains(item, node.left)
```



Look left