Lecture 3

Binary Search Trees (contd.)

Inorder Tree Walk: Proof of Correctness

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof:

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: Basis Step:

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: Basis Step:

Inductive Step:

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: Basis Step: Statement is trivially true when BST contains only one node.

Inductive Step:

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Inductive Hypothesis (IH)

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Inductive Hypothesis (IH)

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Inorder Tree Walk: Proof of Correctness

Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:

Inorder Tree Walk: Proof of Correctness

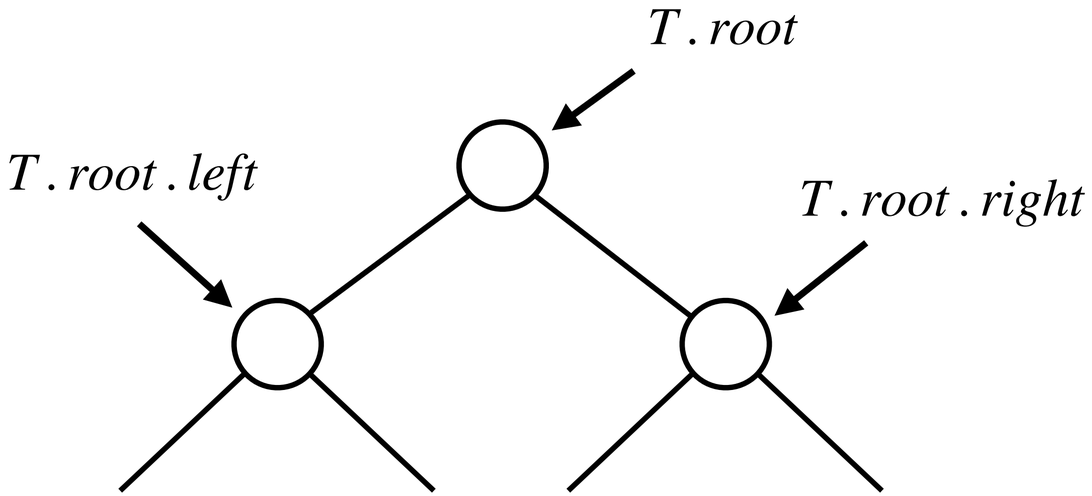
Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:



Inorder Tree Walk: Proof of Correctness

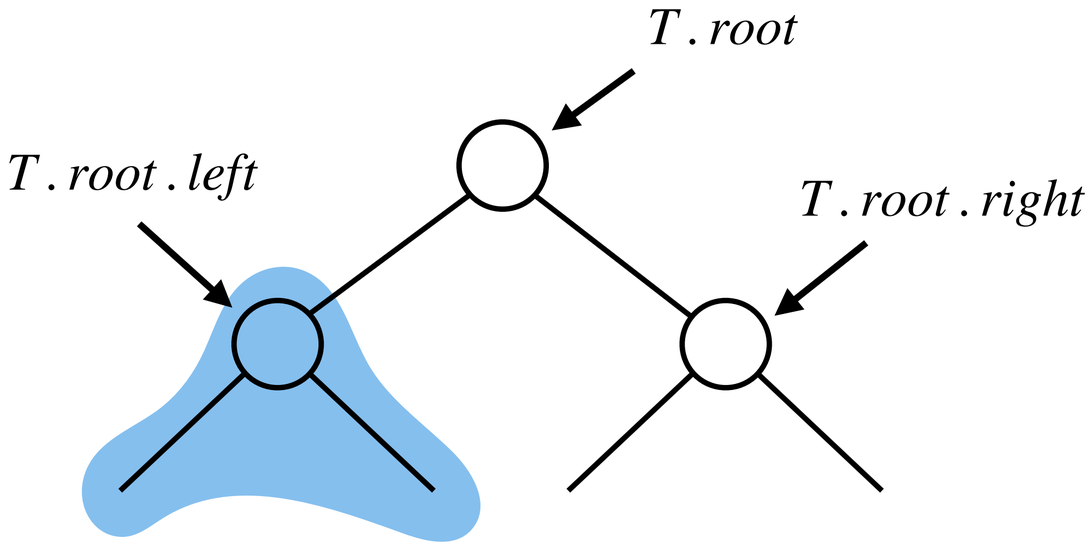
Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:



Inorder Tree Walk: Proof of Correctness

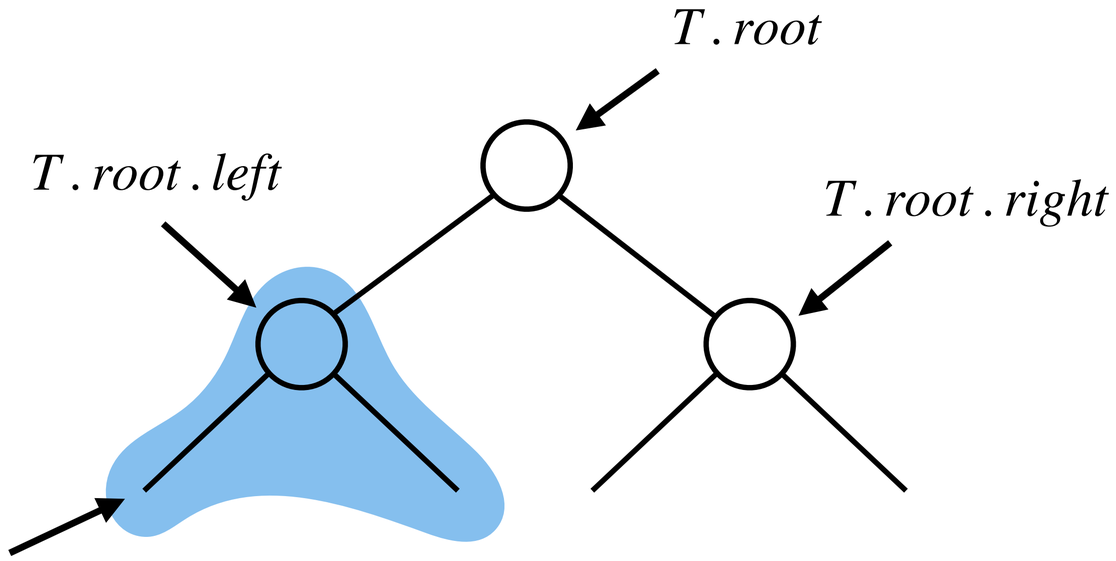
Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:



Inorder Tree Walk: Proof of Correctness

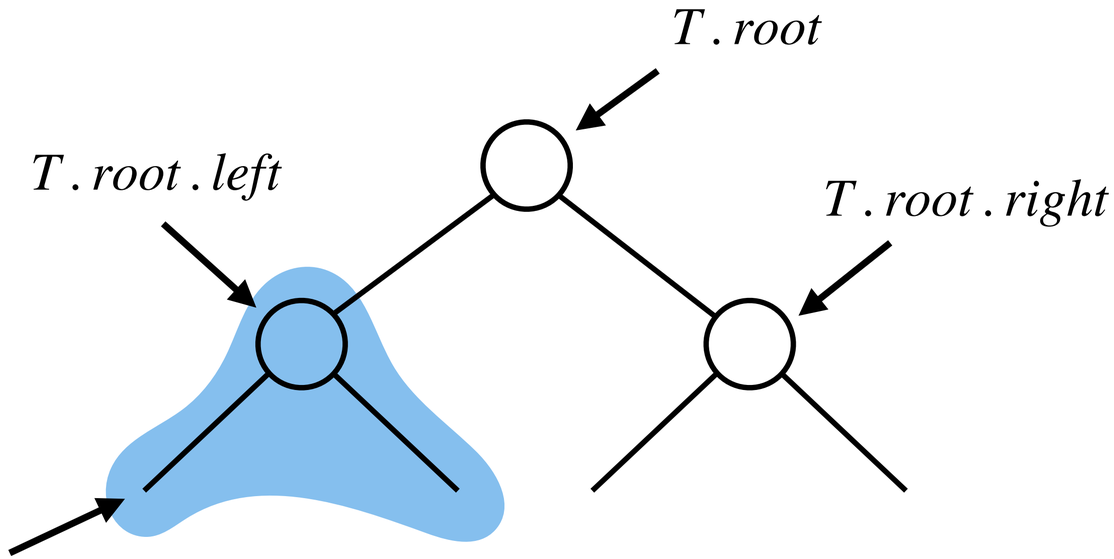
Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:



First print keys here in sorted order
(∵ IH and this subtree has $\leq k$ keys)

Inorder Tree Walk: Proof of Correctness

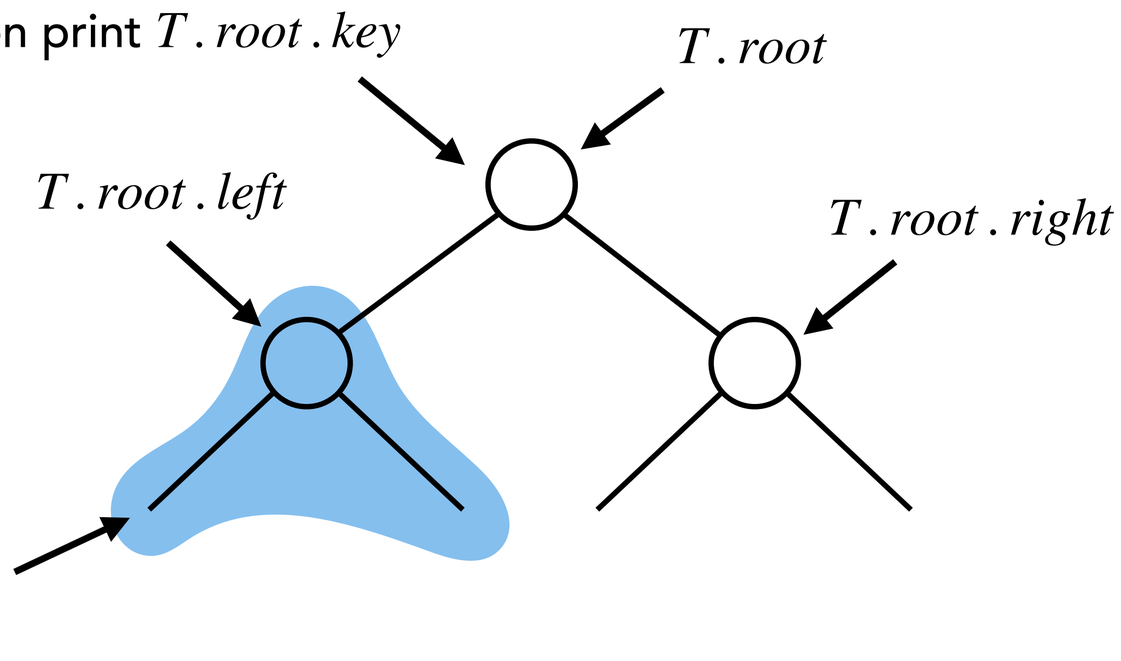
Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:



Inorder Tree Walk: Proof of Correctness

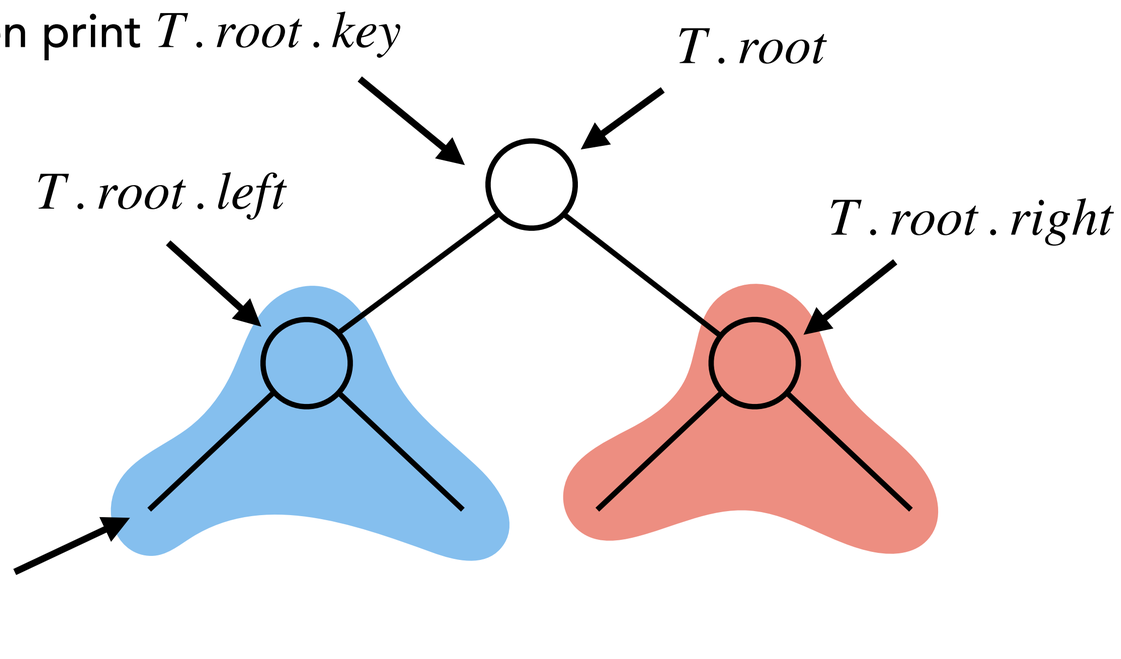
Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:



Inorder Tree Walk: Proof of Correctness

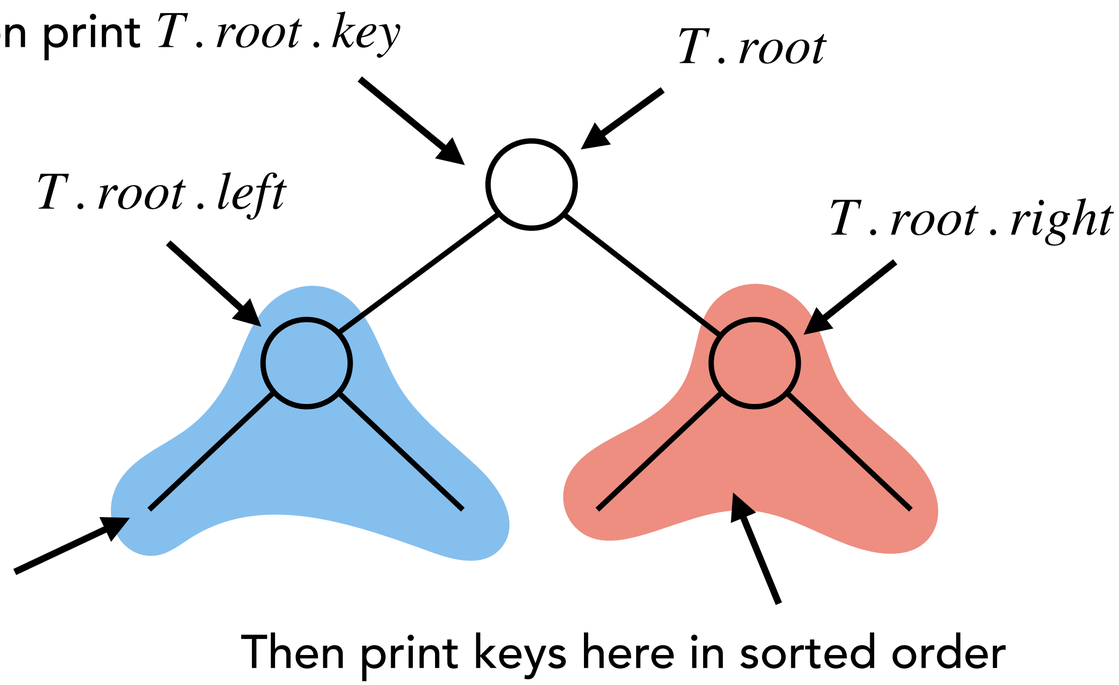
Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:



Inorder Tree Walk: Proof of Correctness

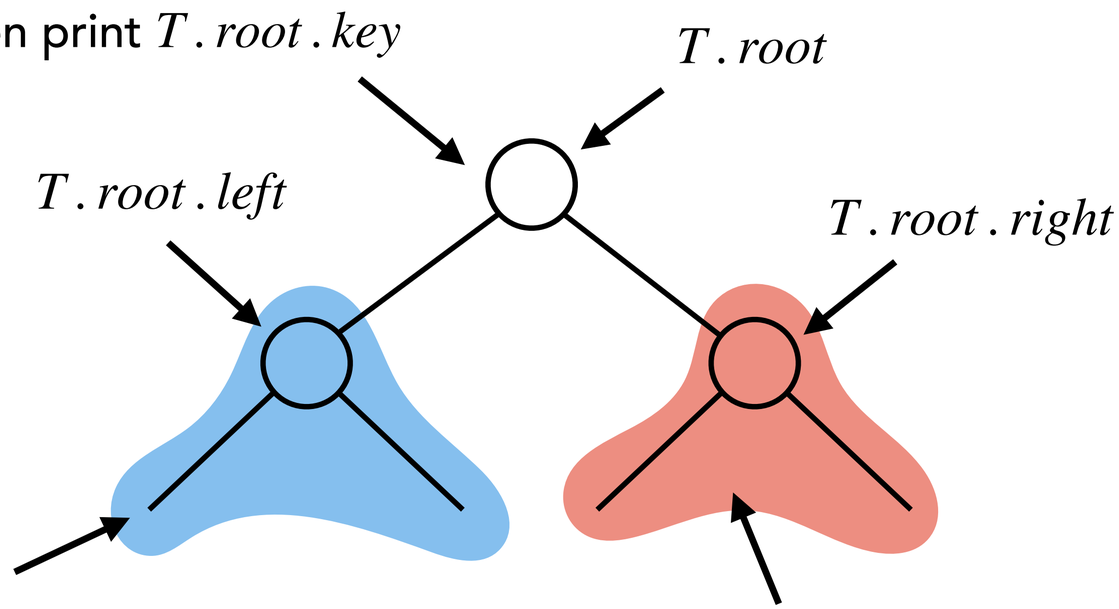
Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:



First print keys here in sorted order
(∵ IH and this subtree has $\leq k$ keys)

Then print keys here in sorted order
(∵ IH and this subtree has $\leq k$ keys)

Inorder Tree Walk: Proof of Correctness

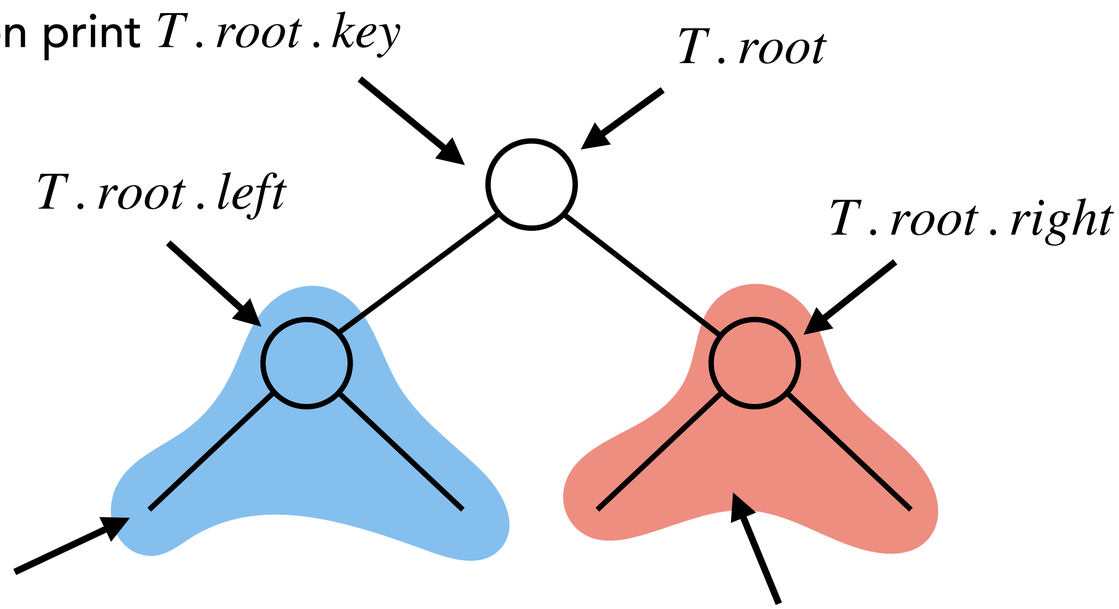
Claim: **Inorder-Tree-Walk**$(T.root)$ will print the keys of the BST T in sorted order.

Proof: **Basis Step:** Statement is trivially true when BST contains only one node.

Inductive Step: Assume the statement is true for all the BSTs with $\leq k$ nodes.

Let T be a BST with $k+1$ nodes.

Then, **Inorder-Tree-Walk**$(T.root)$ will:



First print keys here in sorted order
(∵ IH and this subtree has $\leq k$ keys)

Then print keys here in sorted order
(∵ IH and this subtree has $\leq k$ keys)

■

Inorder Tree Walk: Time Analysis

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

Inorder Tree Walk: Time Analysis

```
Inorder-Tree-Walk(x):
1.   if x ≠ NIL
2.       Inorder-Tree-Walk(x . left)
3.       print x . key
4.       Inorder-Tree-Walk(x . right)
```

Runtime: $\Theta(n)$, where n is the # of nodes in the tree.

Inorder Tree Walk: Time Analysis

```
Inorder-Tree-Walk(x):
1.   if x ≠ NIL
2.       Inorder-Tree-Walk(x . left)
3.       print x . key
4.       Inorder-Tree-Walk(x . right)
```

Runtime: $\Theta(n)$, where n is the # of nodes in the tree. Because each node gets printed only once.

Inorder Tree Walk: Time Analysis

```
Inorder-Tree-Walk(x):
1.   if x ≠ NIL
2.       Inorder-Tree-Walk(x . left)
3.       print x . key
4.       Inorder-Tree-Walk(x . right)
```

Runtime: $\Theta(n)$, where n is the # of nodes in the tree. Because each node gets printed only once.

Can be proven using induction.

Search in a BST

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Illustration:

Search in a BST

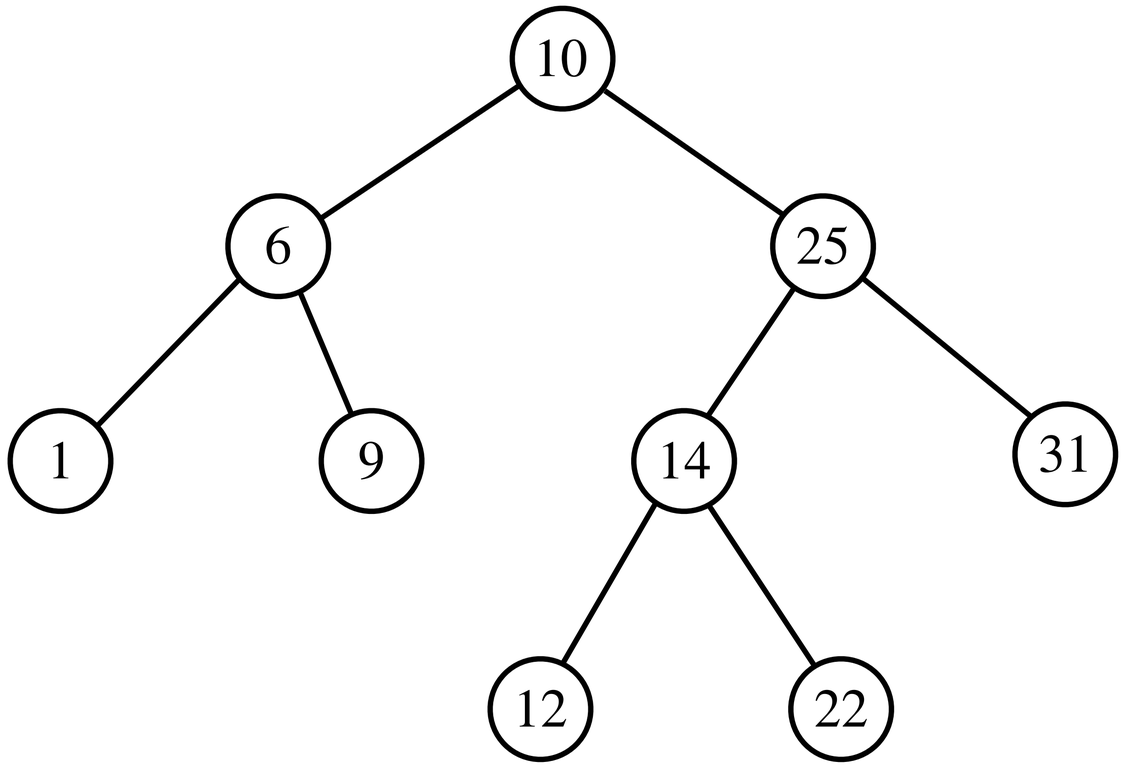
Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Illustration: Searching for 12 in the below BST.

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

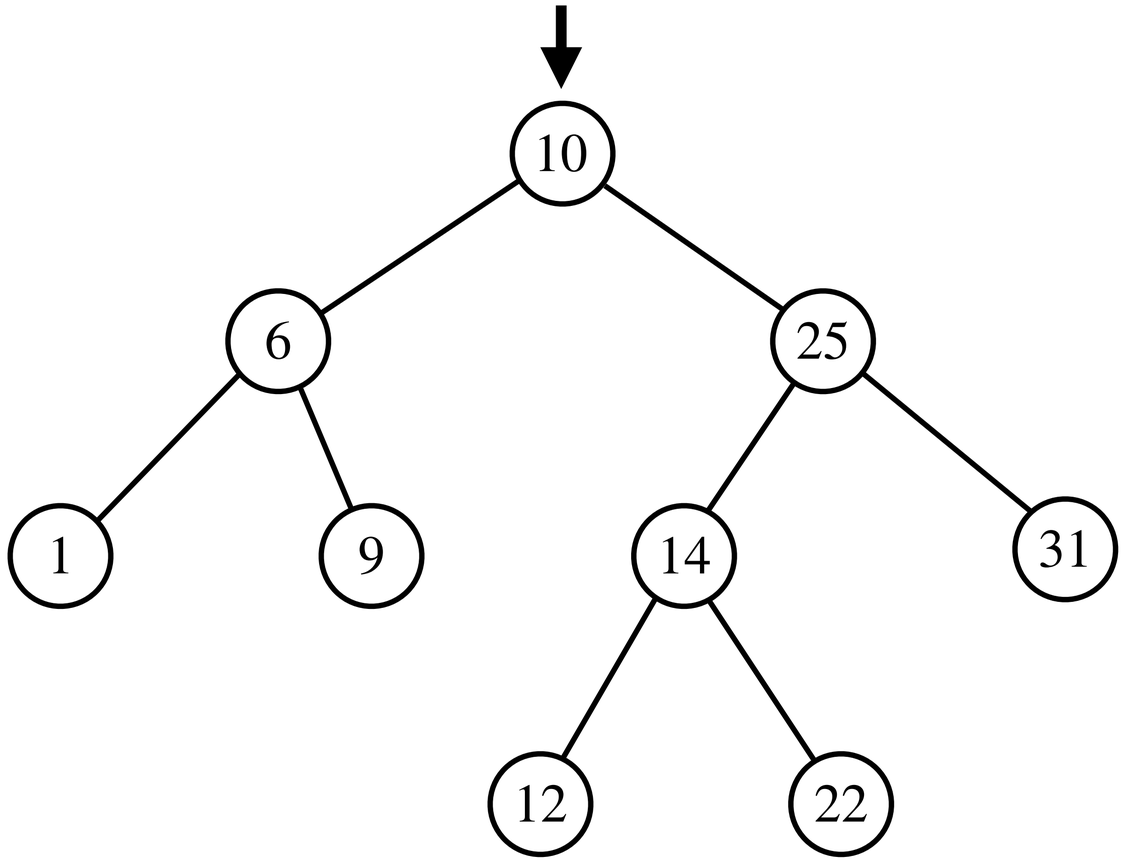
Illustration: Searching for 12 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

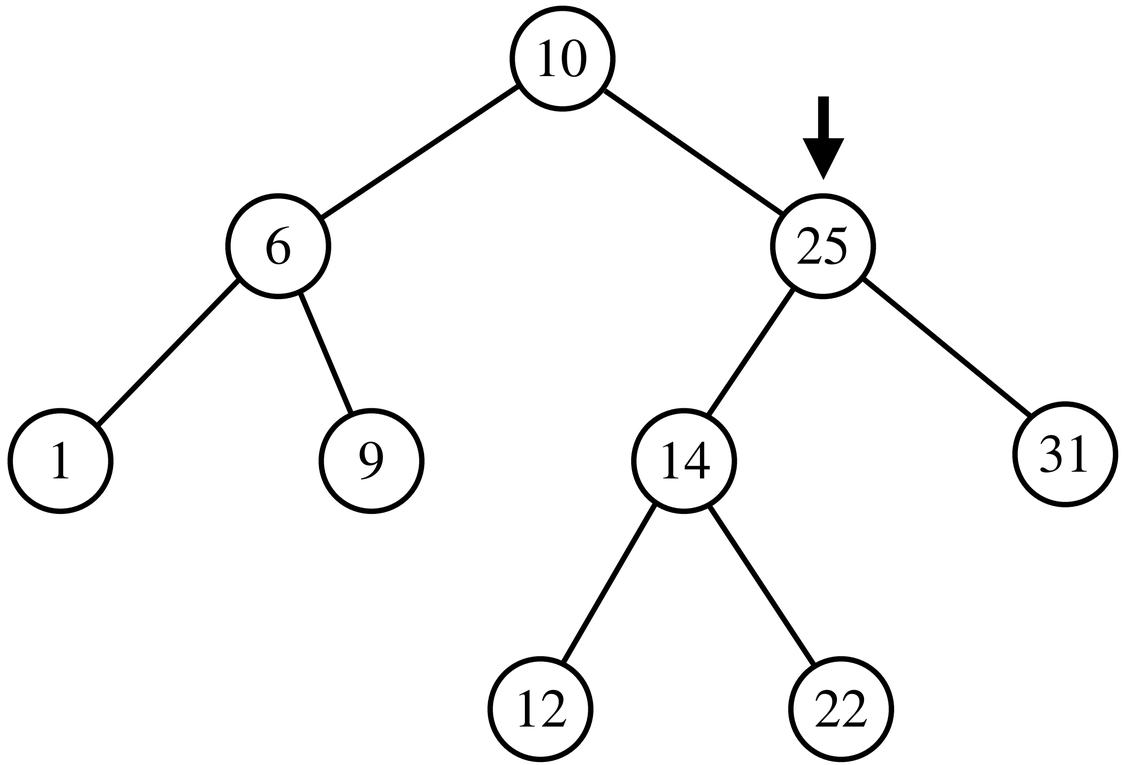
Illustration: Searching for 12 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

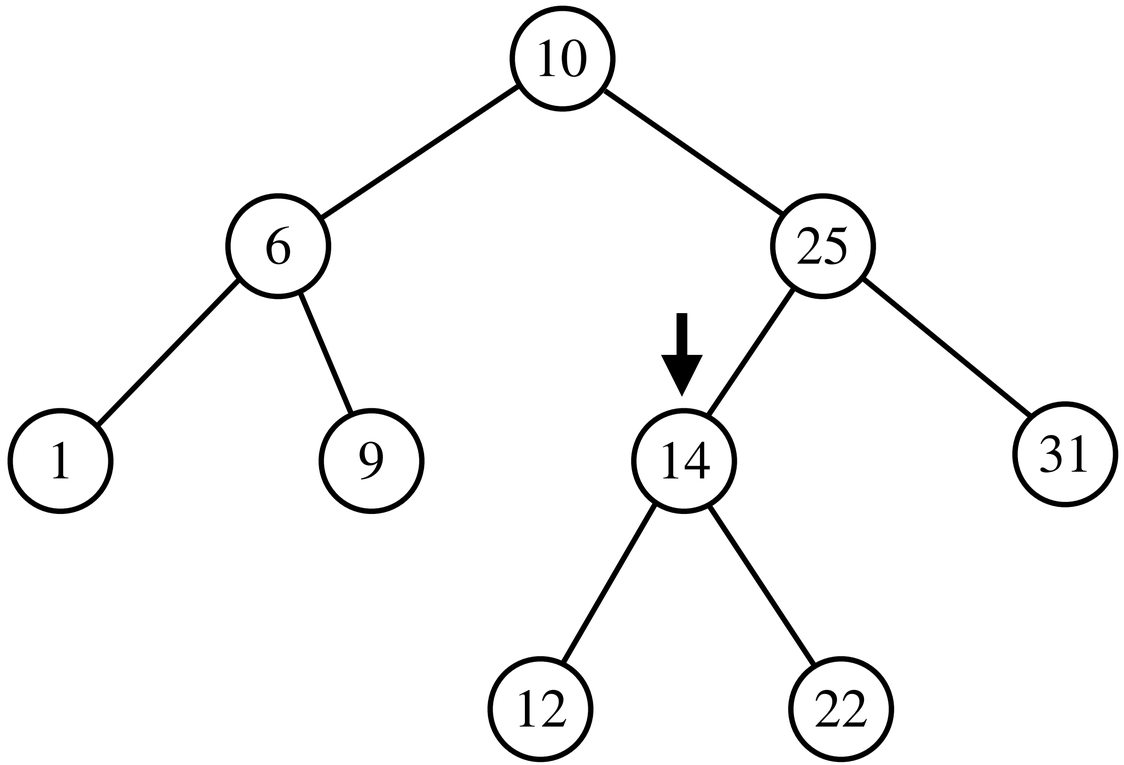
Illustration: Searching for 12 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

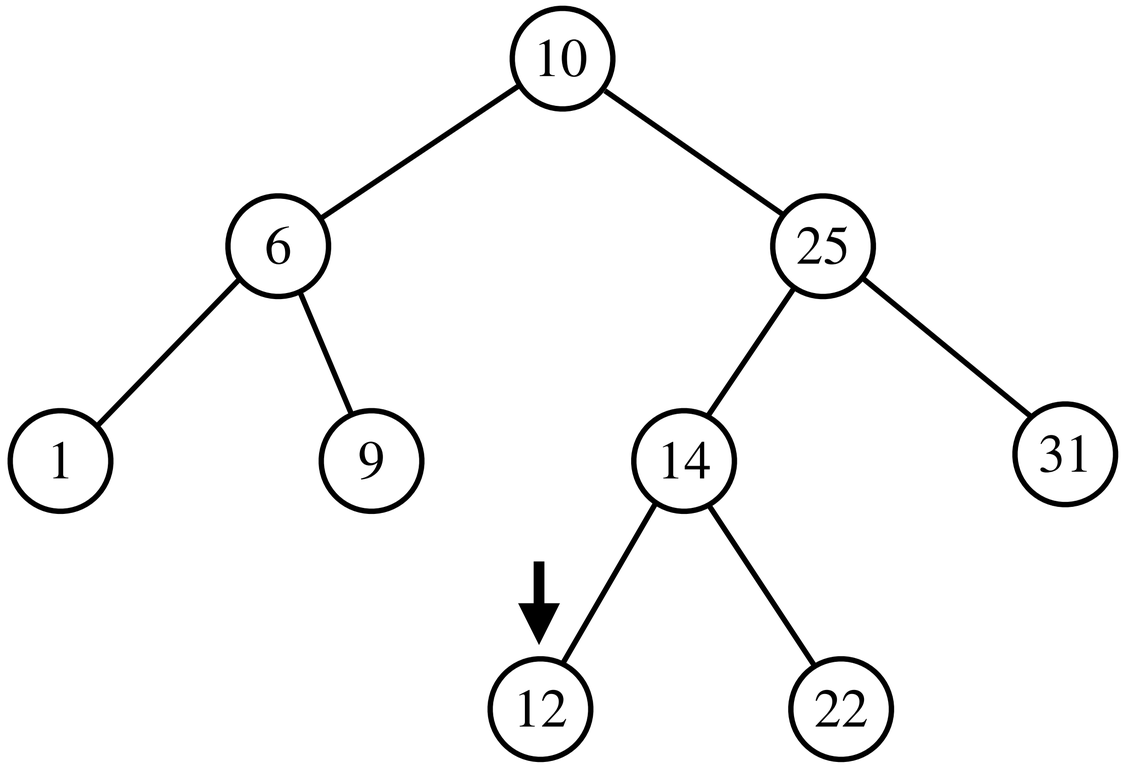
Illustration: Searching for 12 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

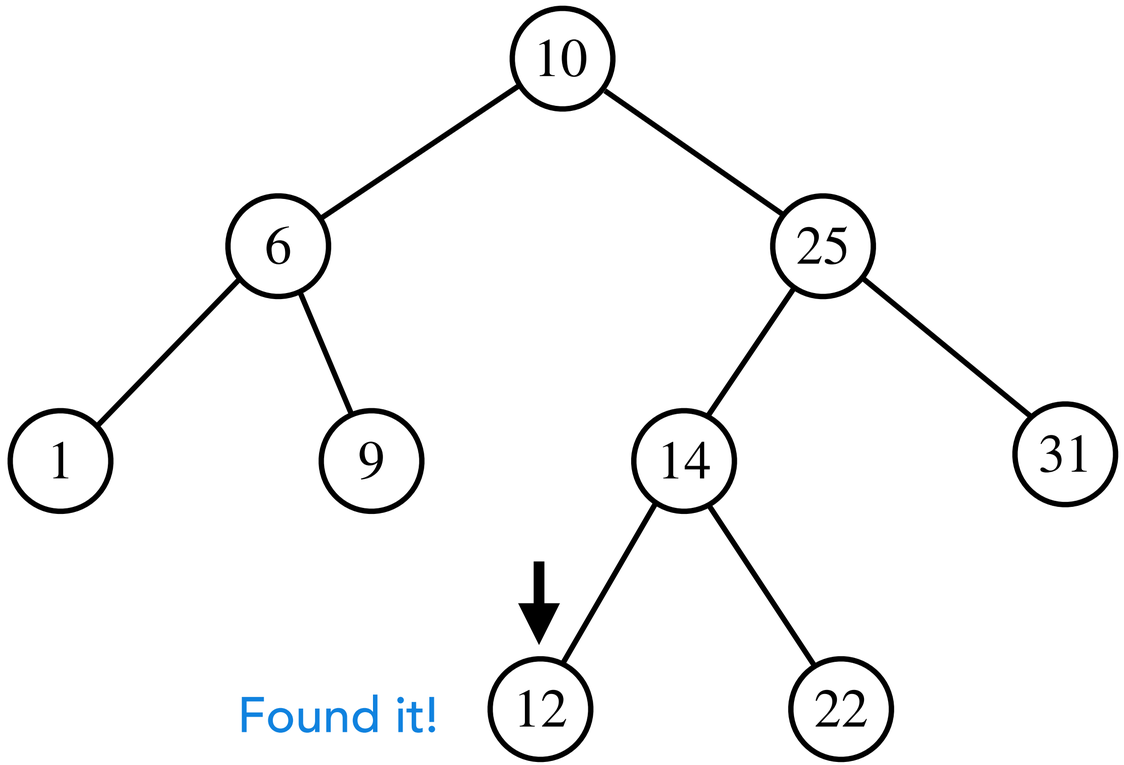
Illustration: Searching for 12 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

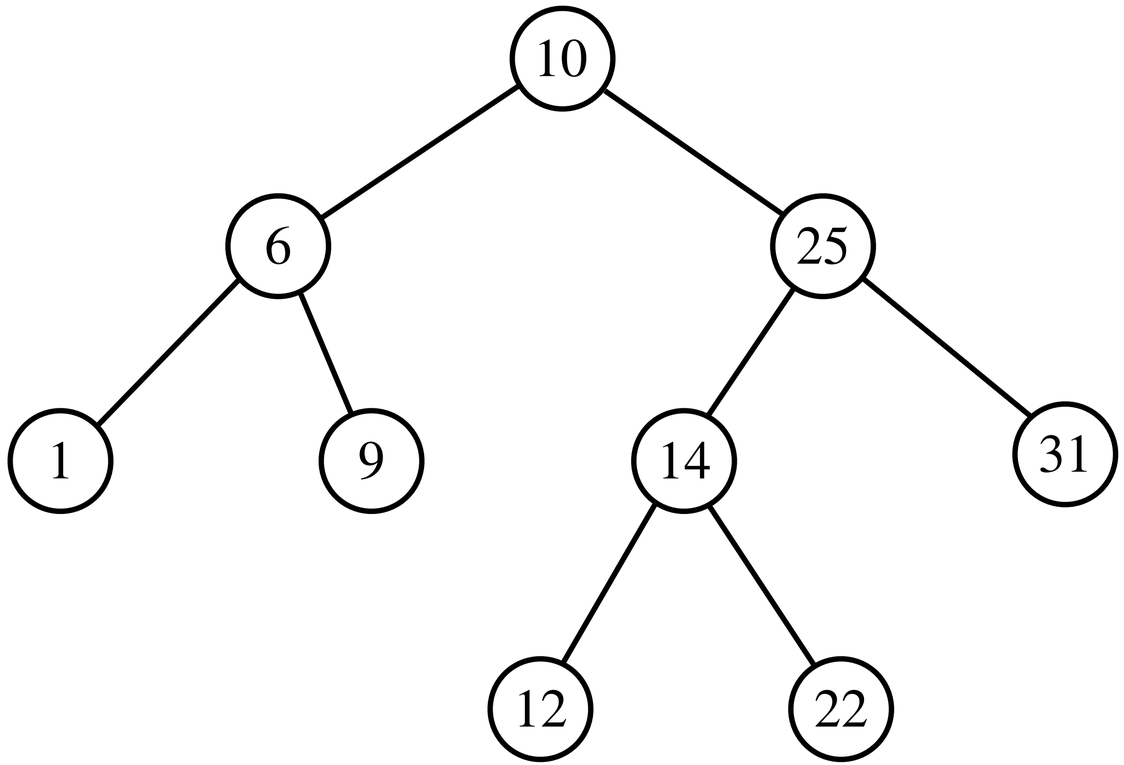
Illustration: Searching for 12 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

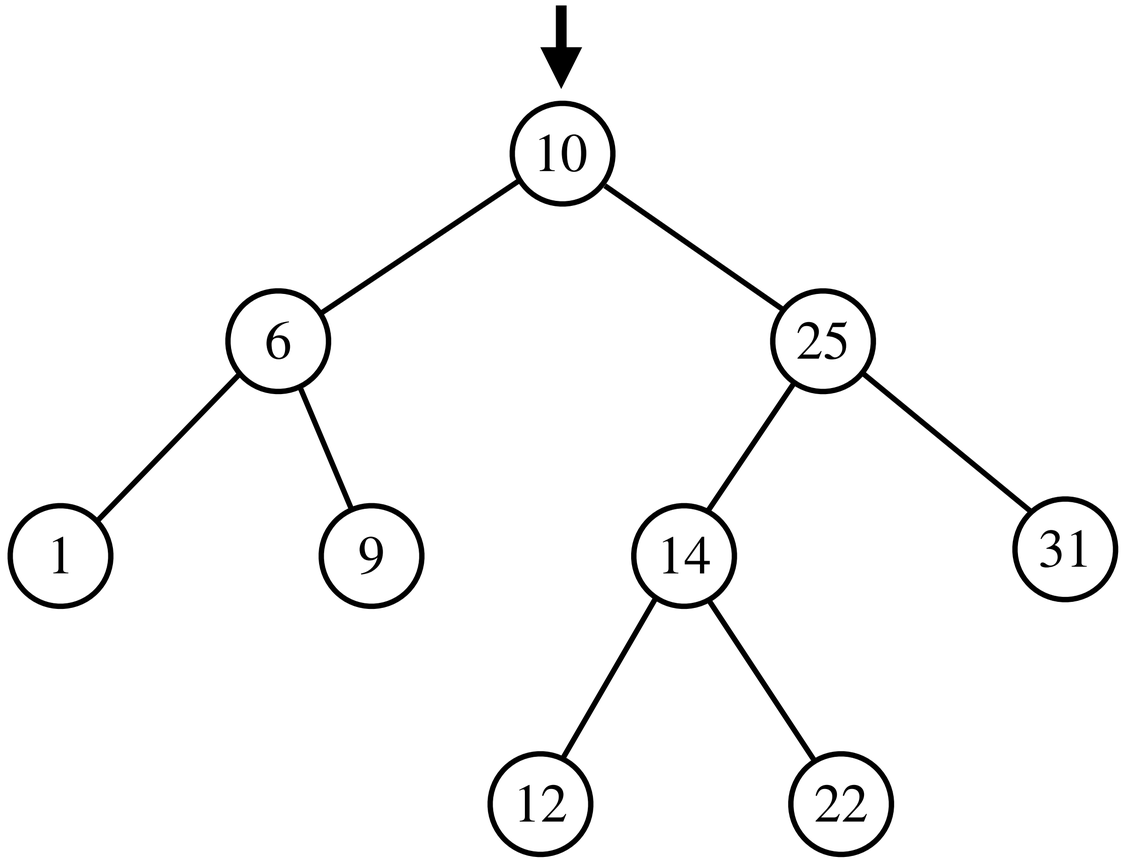
Illustration: Searching for 5 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

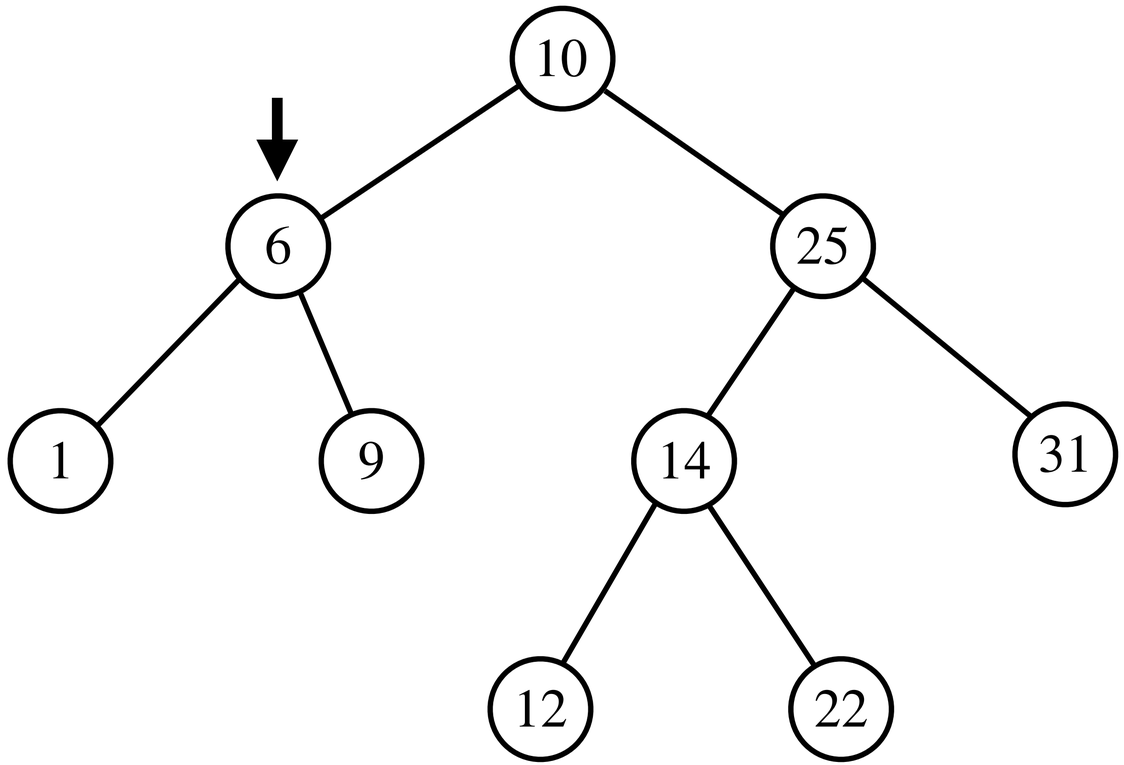
Illustration: Searching for 5 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

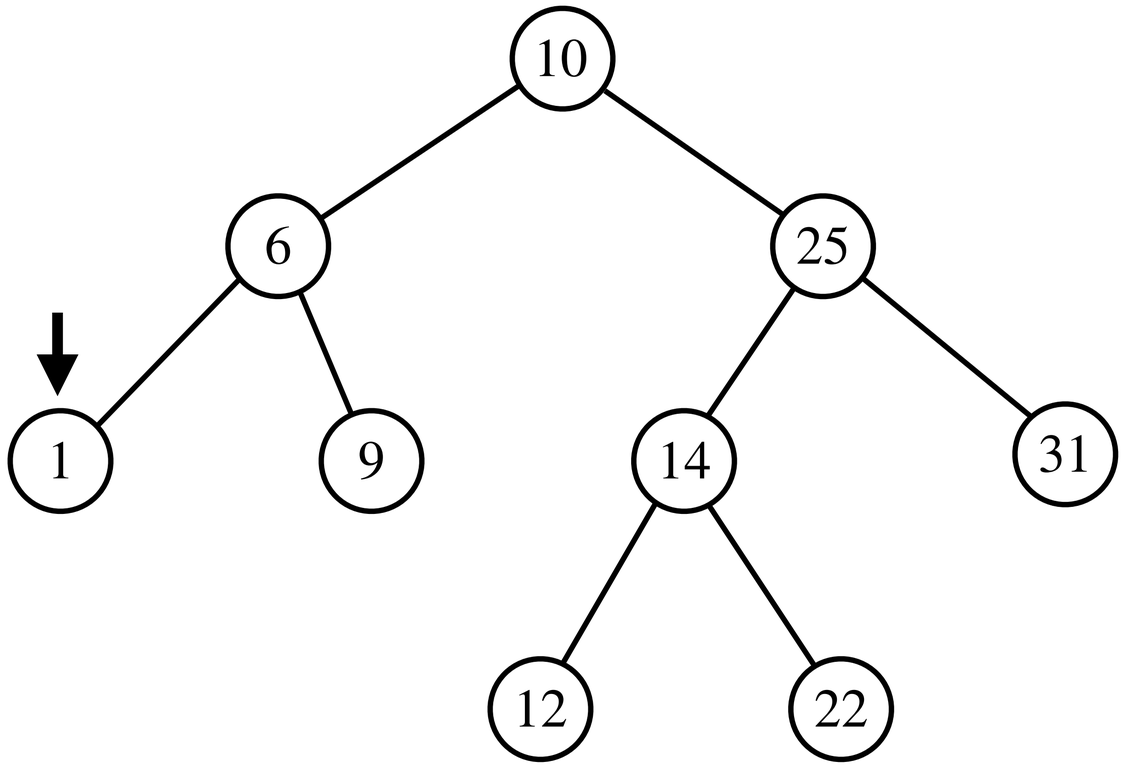
Illustration: Searching for 5 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

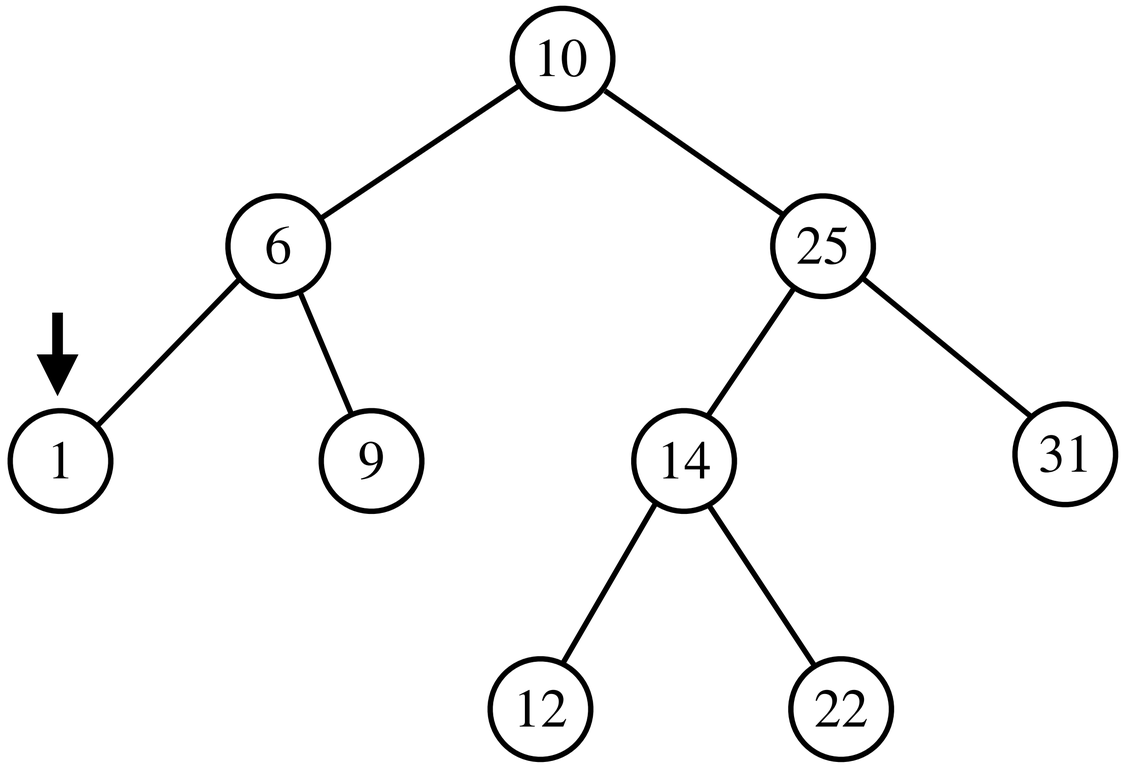
Illustration: Searching for 5 in the below BST.



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Illustration: Searching for 5 in the below BST.



5 is not present!

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T.root, k)$ to search for element with key k in tree T.

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T.root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

1. **while** $x \neq$ NIL **and** $k \neq x\,.\,key$

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

1. **while** $x \neq$ NIL **and** $k \neq x\,.\,key$
2. **if** $k < x\,.\,key$

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

1. **while** $x \neq$ NIL **and** $k \neq x\,.\,key$
2. **if** $k < x\,.\,key$
3. $x = x\,.\,left$

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

1. **while** $x \neq$ NIL **and** $k \neq x\,.\,key$
2. **if** $k < x\,.\,key$
3. $x = x\,.\,left$
4. **else**

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

```
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
```

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

```
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

```
Tree-Search (x, k):
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```

$k = 8$

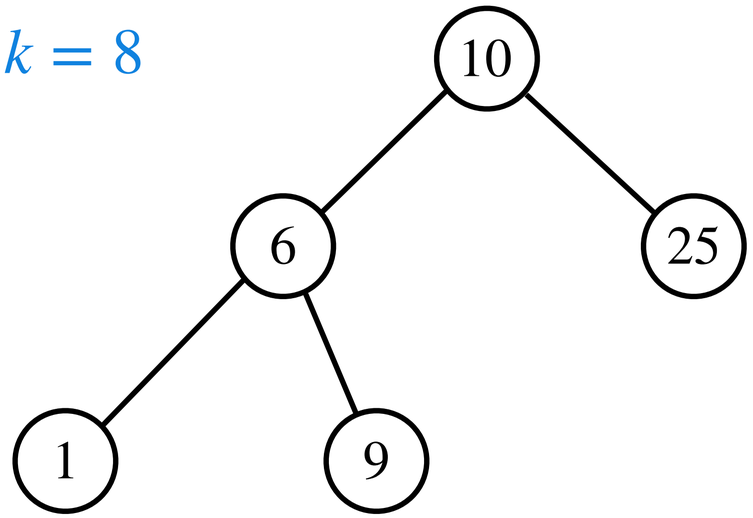
Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

```
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```



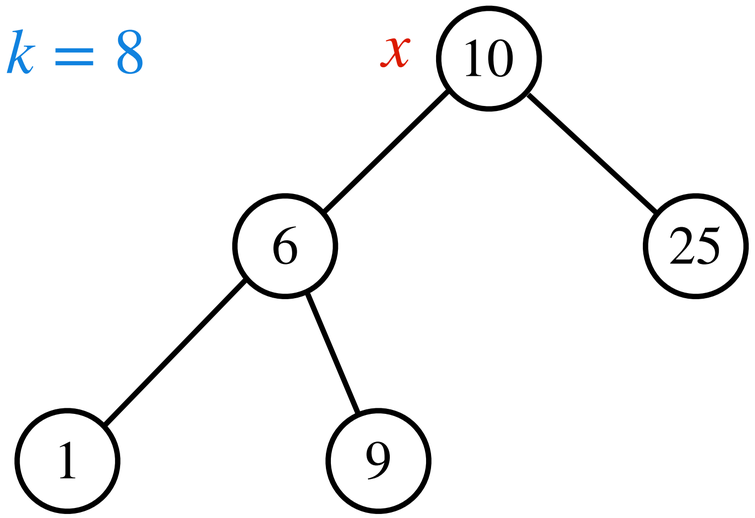
Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

```
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```



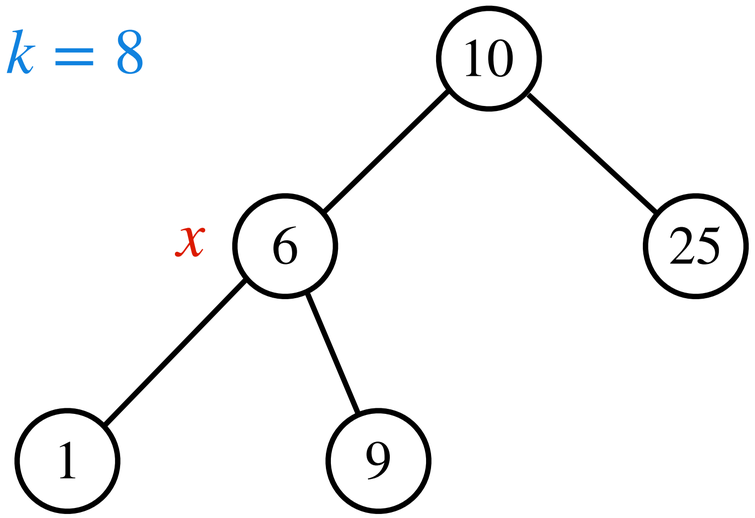
Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

```
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```



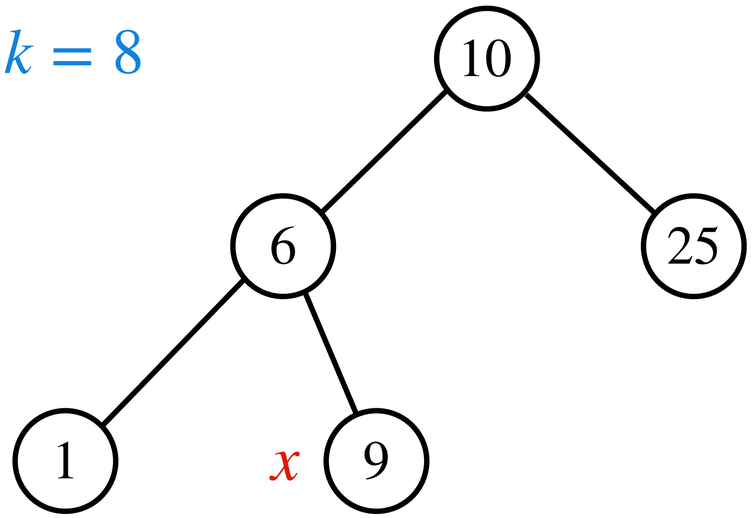
Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

```
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```



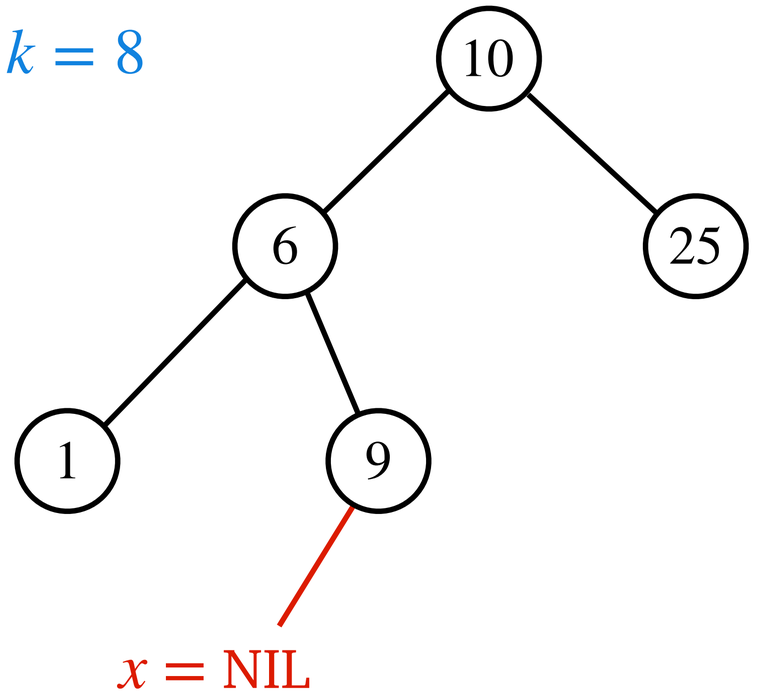
Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

```
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```



Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T\,.\,root, k)$ to search for element with key k in tree T.

```
Tree-Search (x, k):
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T.root, k)$ to search for element with key k in tree T.

```
Tree-Search (x, k):
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```

Runtime: $\Theta(h)$, where h = height of T,

Search in a BST

Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Algorithm: Call **Tree-Search**$(T.root, k)$ to search for element with key k in tree T.

Tree-Search (x, k)**:**

```
1.  while x ≠ NIL and k ≠ x.key
2.      if k < x.key
3.          x = x.left
4.      else
5.          x = x.right
6.  return x
```

Runtime: $\Theta(h)$, where h = height of T, as while loop goes one level down with every iteration.

Finding Successor in a BST

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Node printed after x in inorder-walk.

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration:

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

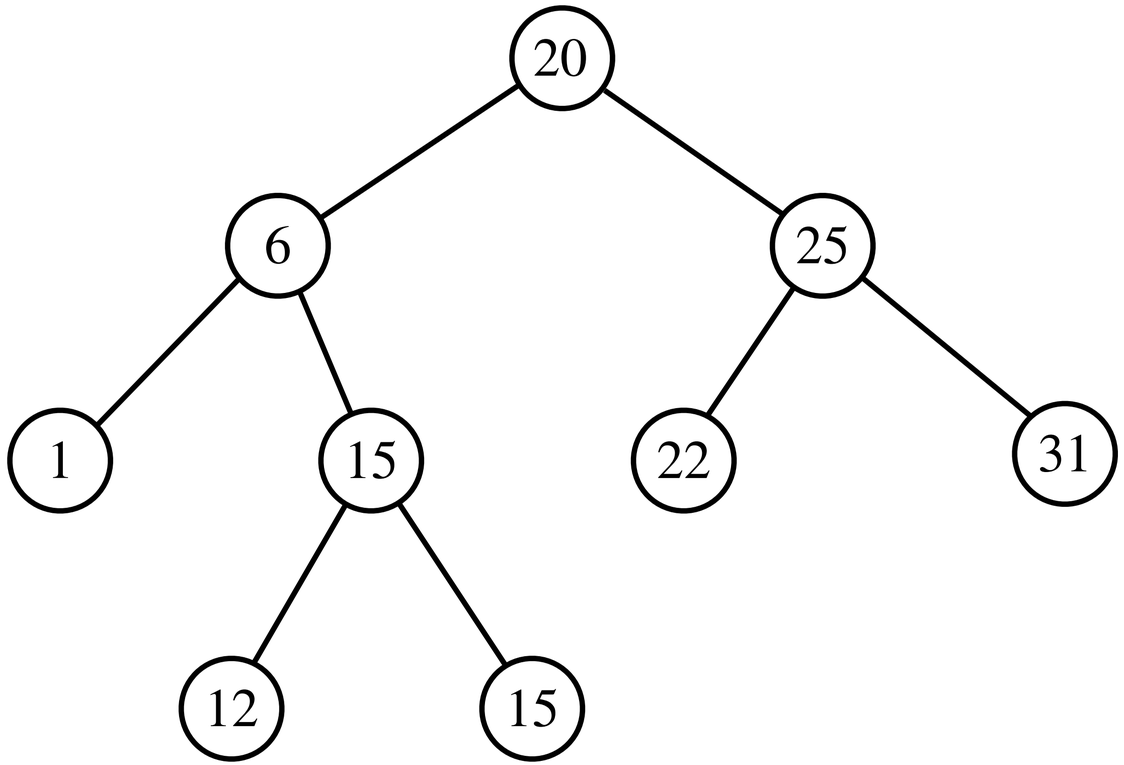
Illustration: Find the successor of 6 in the below BST.

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of 6 in the below BST.

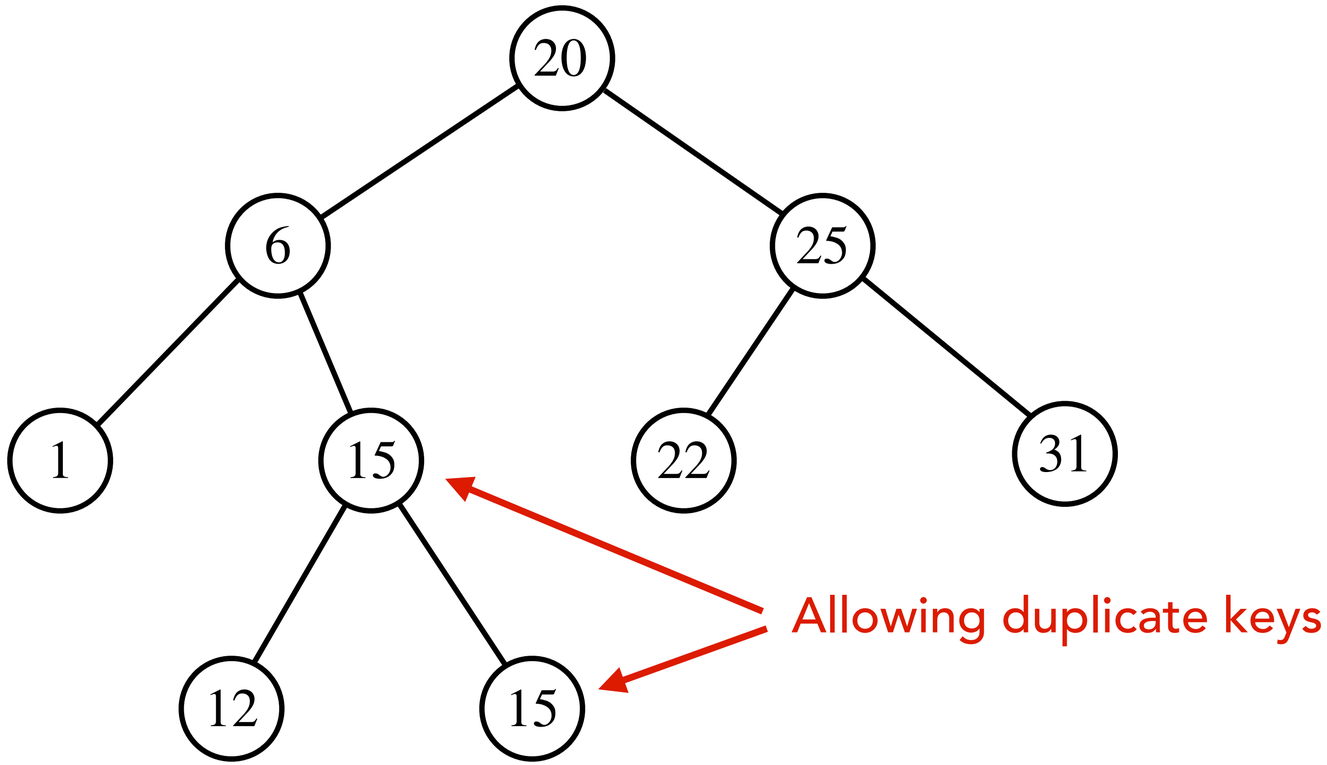


Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of 6 in the below BST.

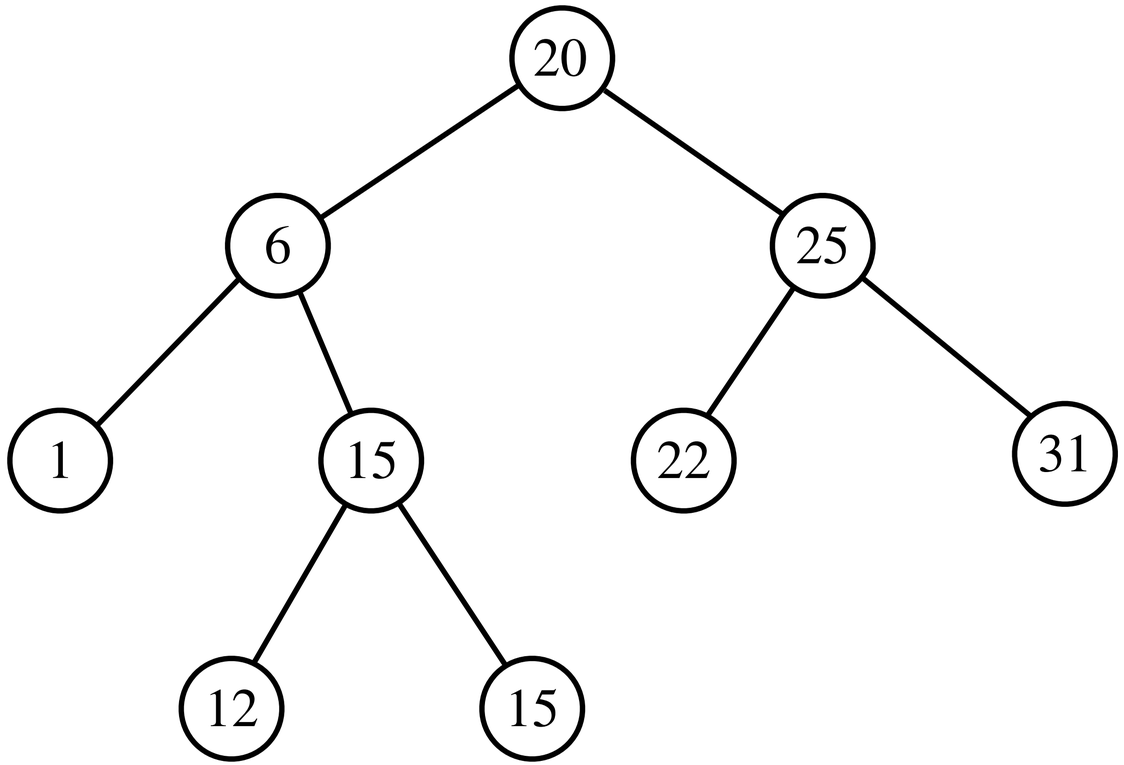


Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of 6 in the below BST.

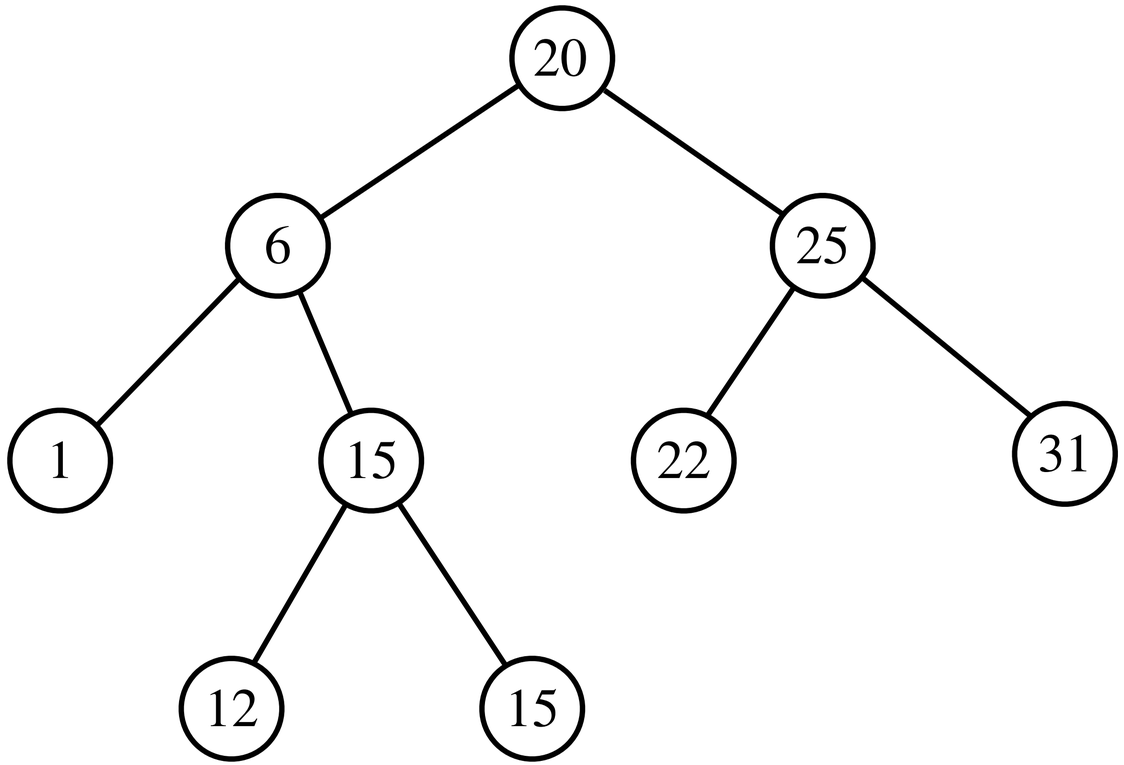


Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of 6 in the below BST.



Recall:

Inorder-Tree-Walk(x):

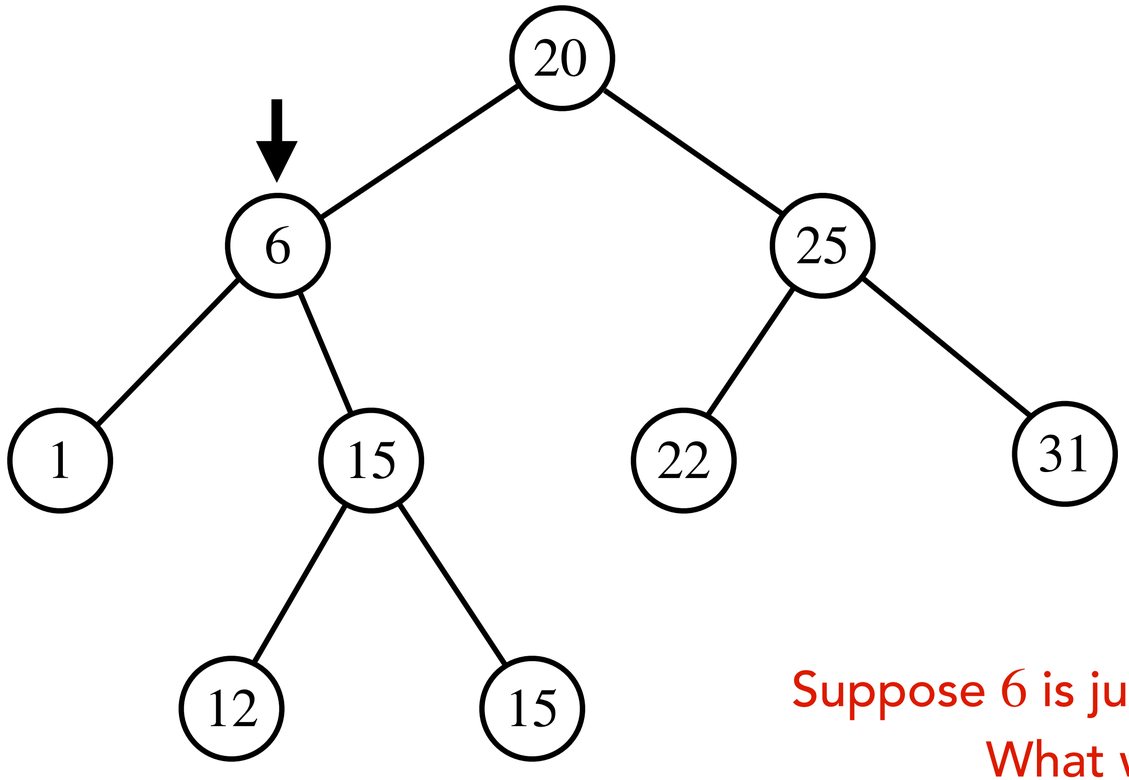
1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of 6 in the below BST.



Recall:

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

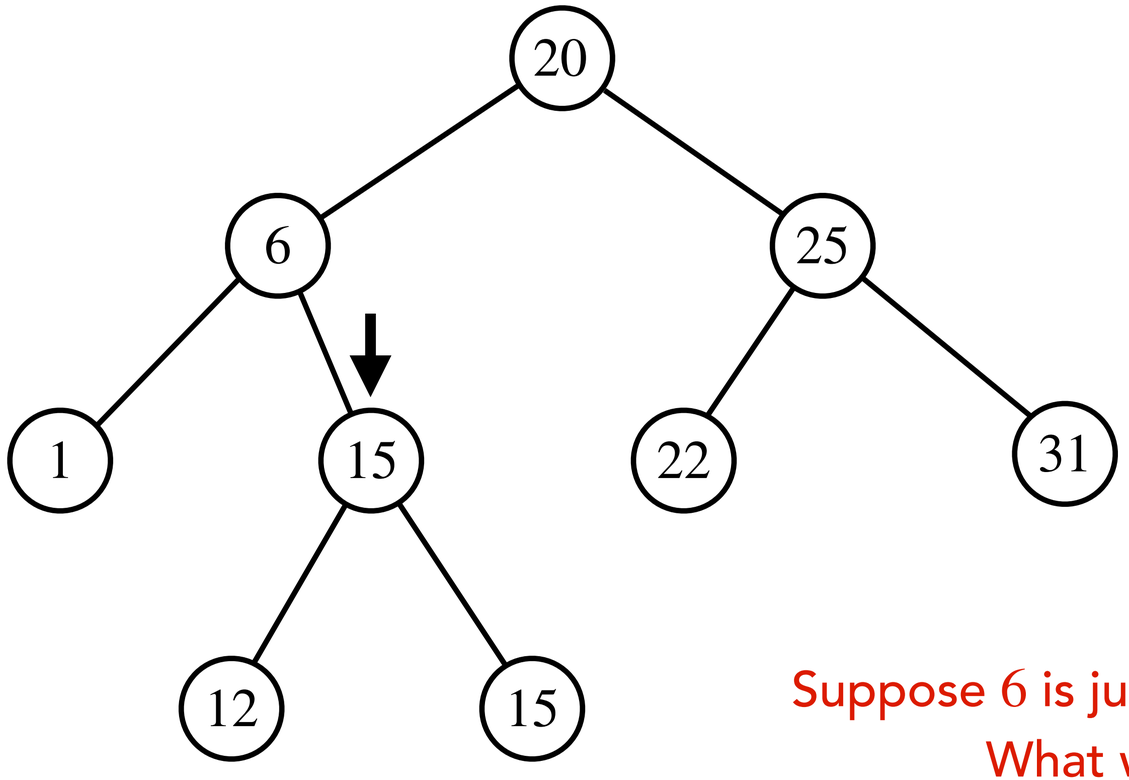
Suppose 6 is just printed in Inorder walk.
What will happen next?

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of 6 in the below BST.



Recall:

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

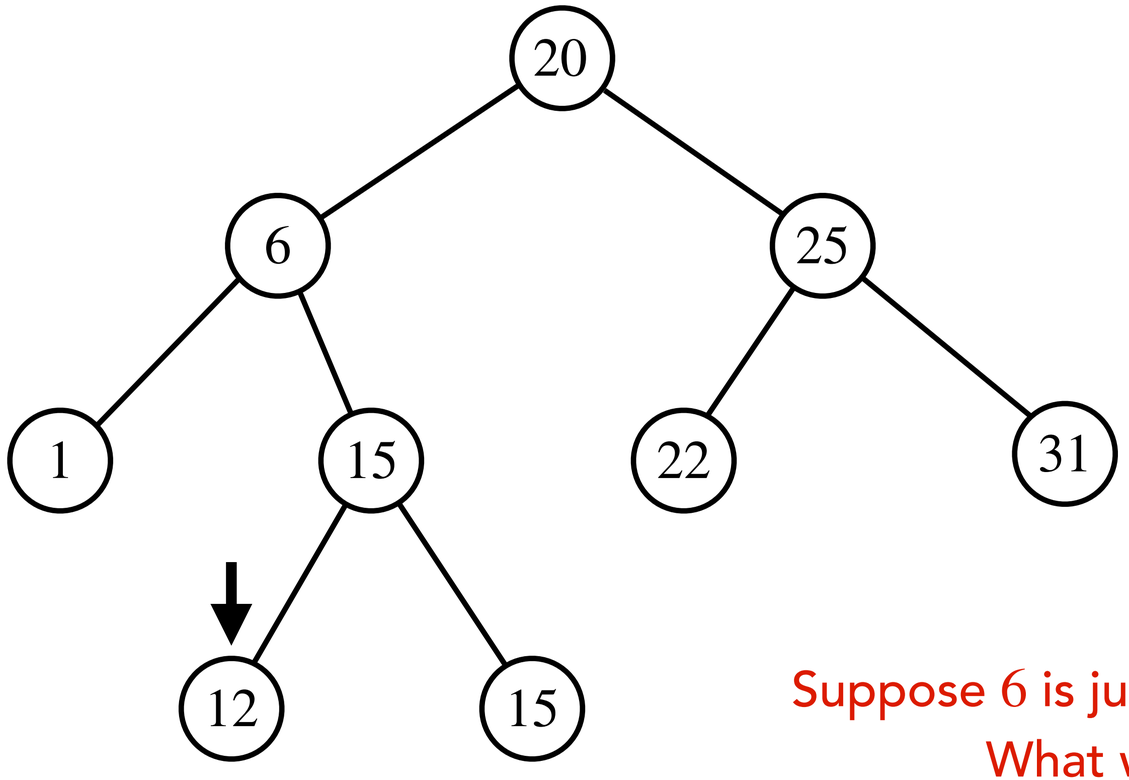
Suppose 6 is just printed in Inorder walk.
What will happen next?

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of 6 in the below BST.



Recall:

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

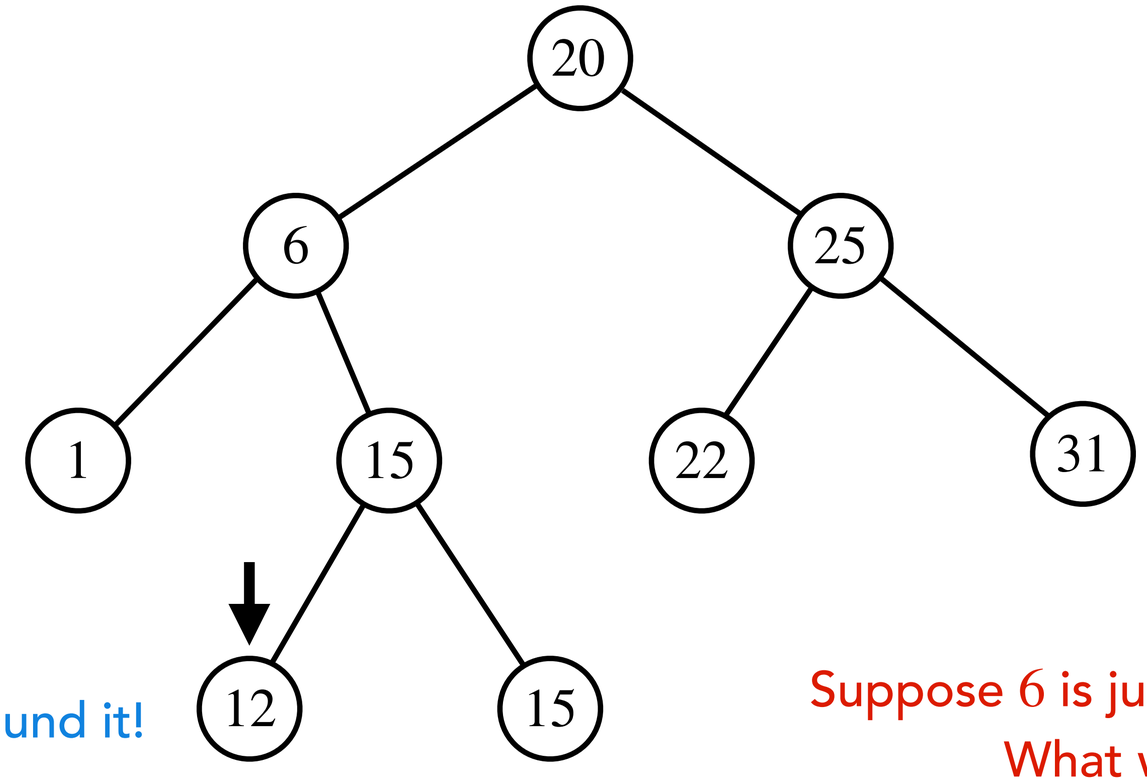
Suppose 6 is just printed in Inorder walk.
What will happen next?

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of 6 in the below BST.



Recall:

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

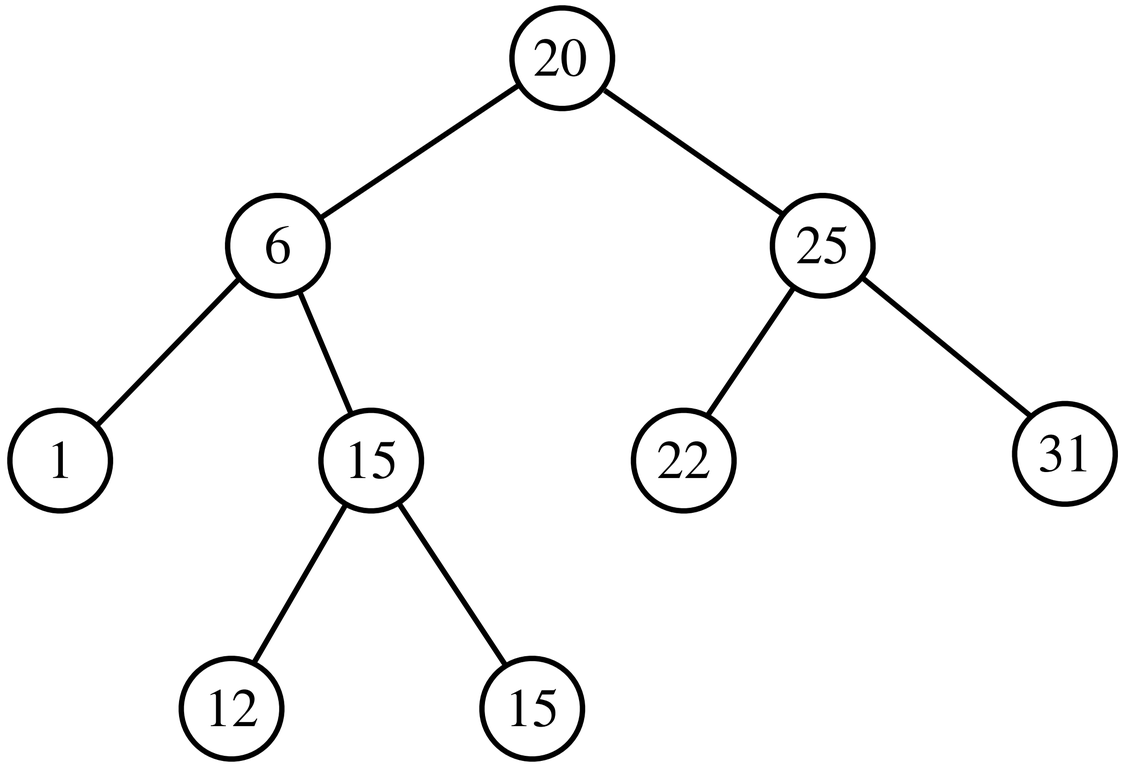
Suppose 6 is just printed in Inorder walk.
What will happen next?

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration:



Recall:

Inorder-Tree-Walk(x):

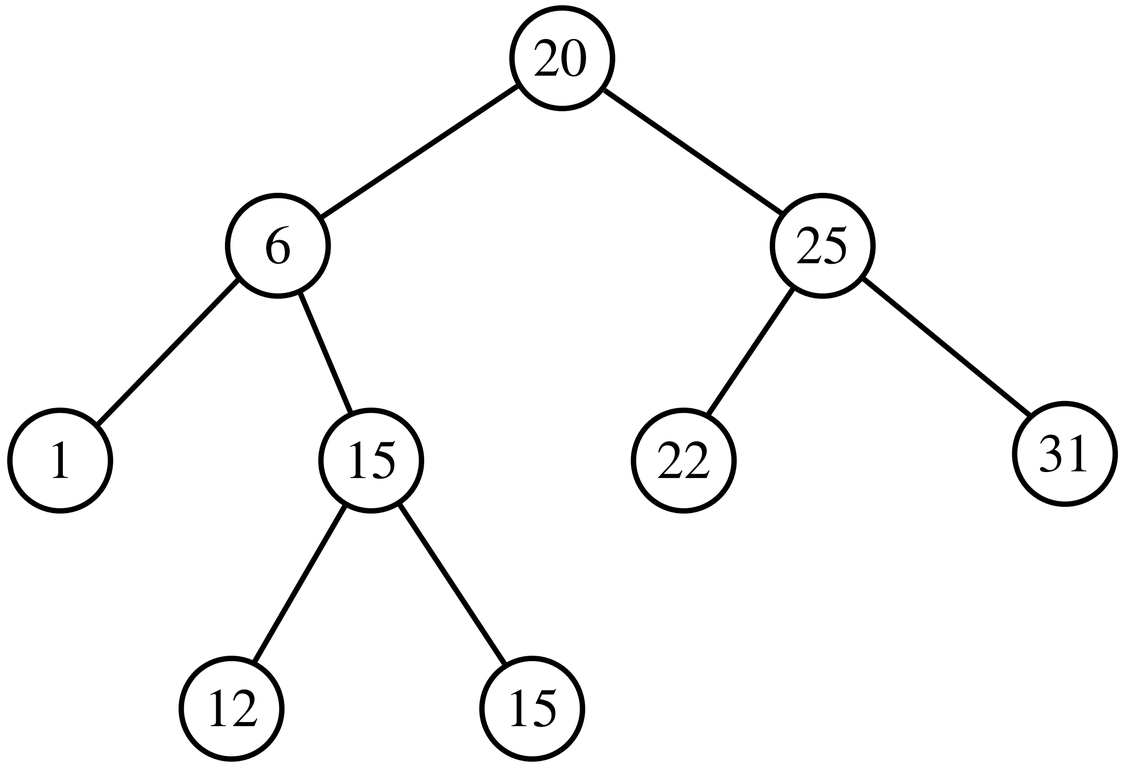
1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of the leaf 15 in the below BST.



Recall:

Inorder-Tree-Walk(x):

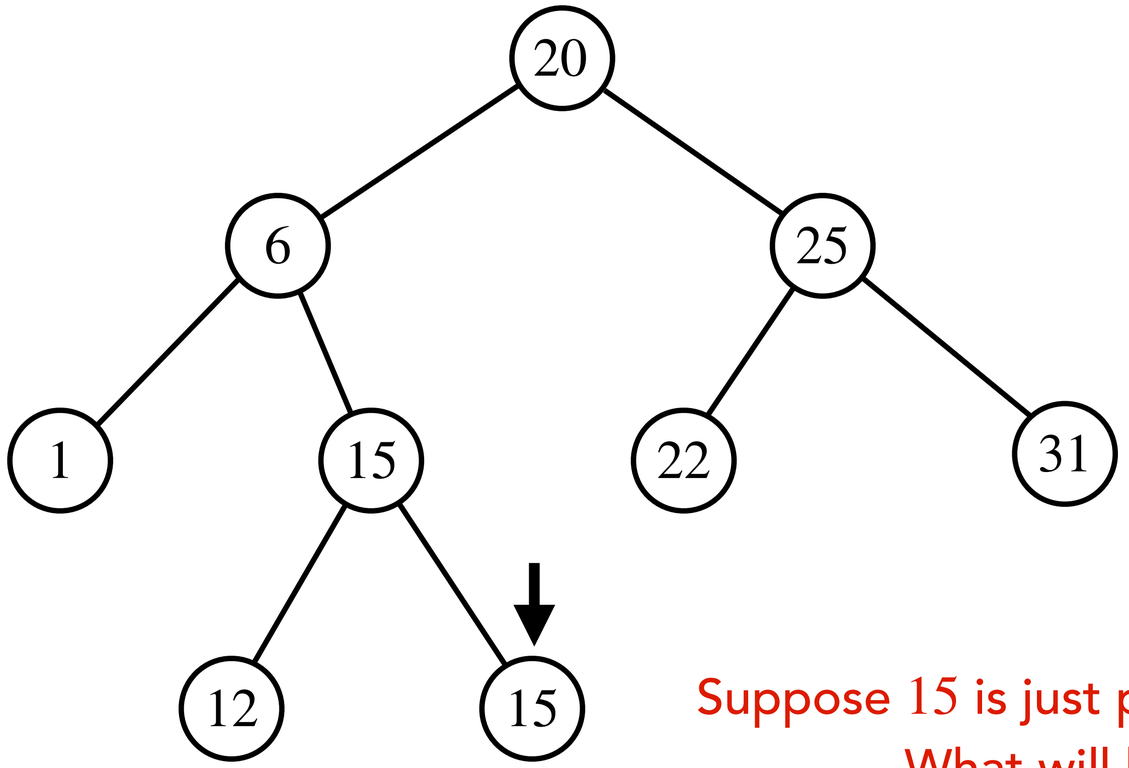
1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of the leaf 15 in the below BST.



Recall:

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

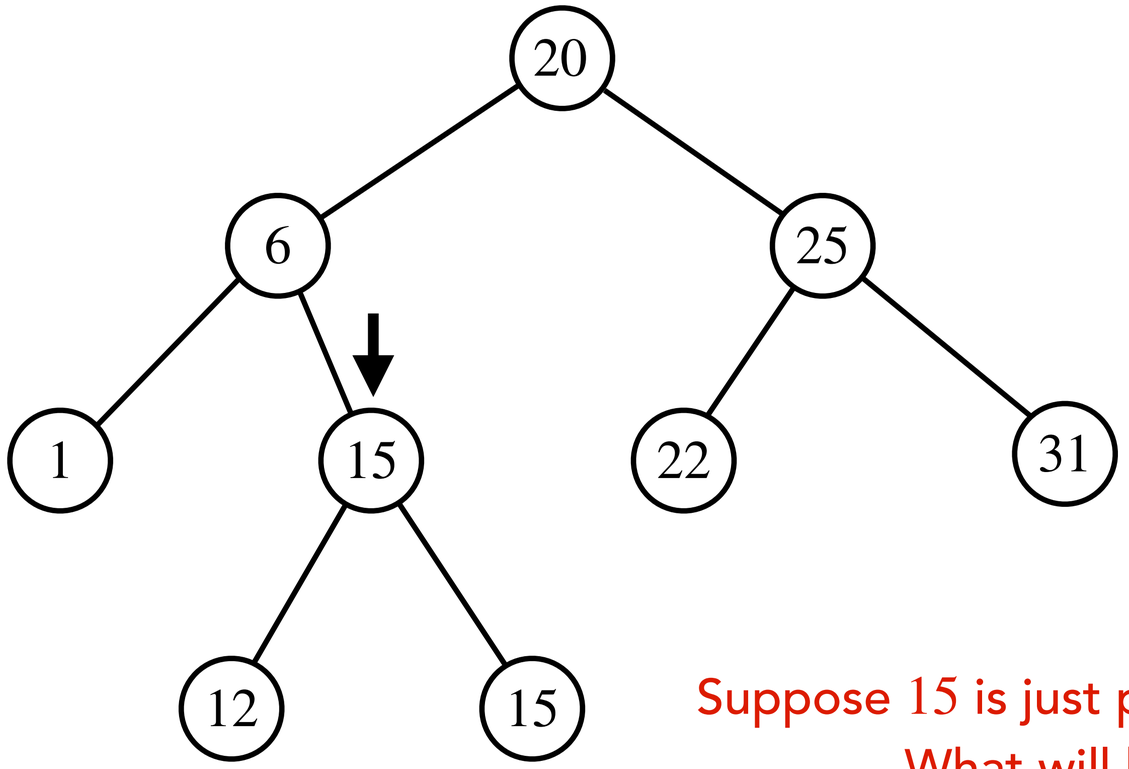
Suppose 15 is just printed in Inorder walk.
What will happen next?

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of the leaf 15 in the below BST.



Recall:

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

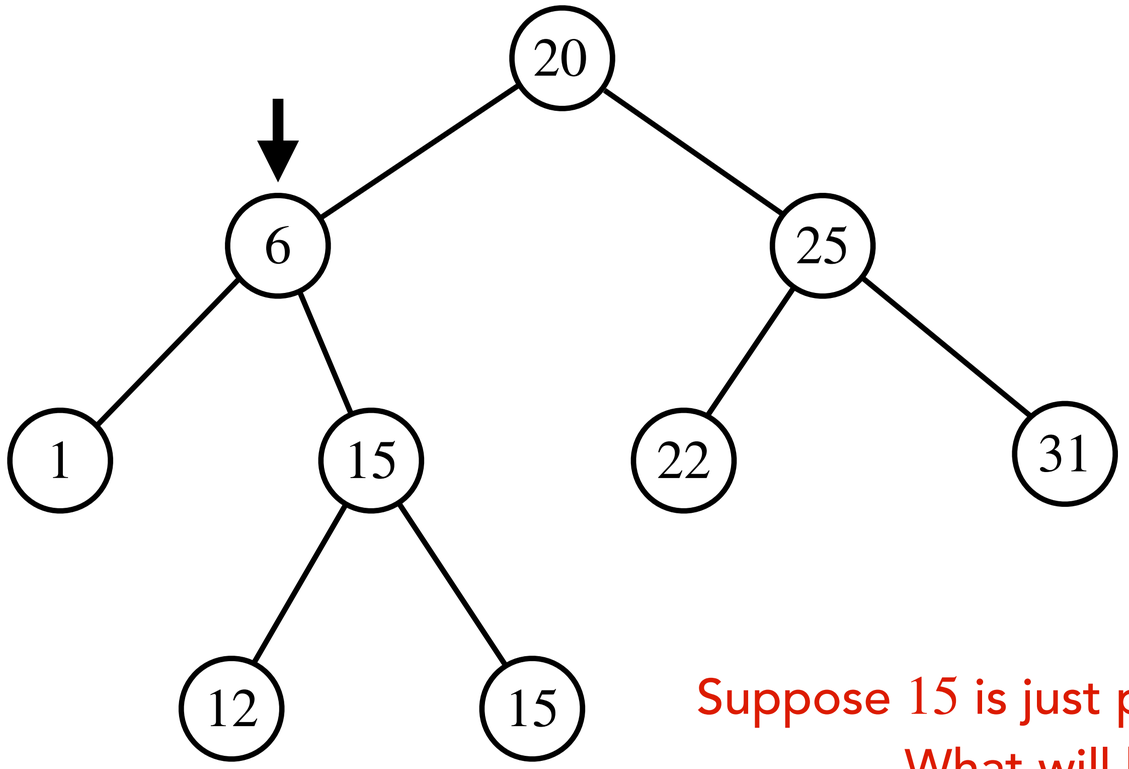
Suppose 15 is just printed in Inorder walk.
What will happen next?

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of the leaf 15 in the below BST.



Recall:

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

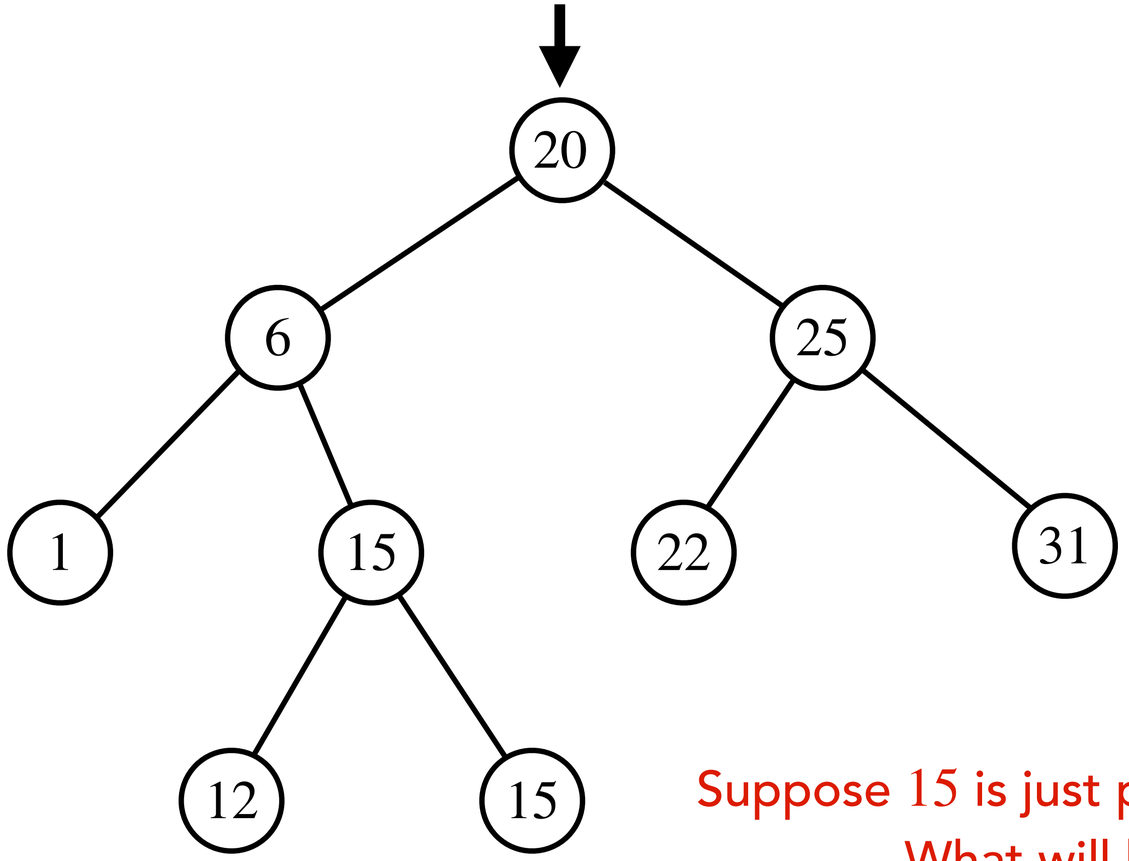
Suppose 15 is just printed in Inorder walk.
What will happen next?

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of the leaf 15 in the below BST.



Recall:

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

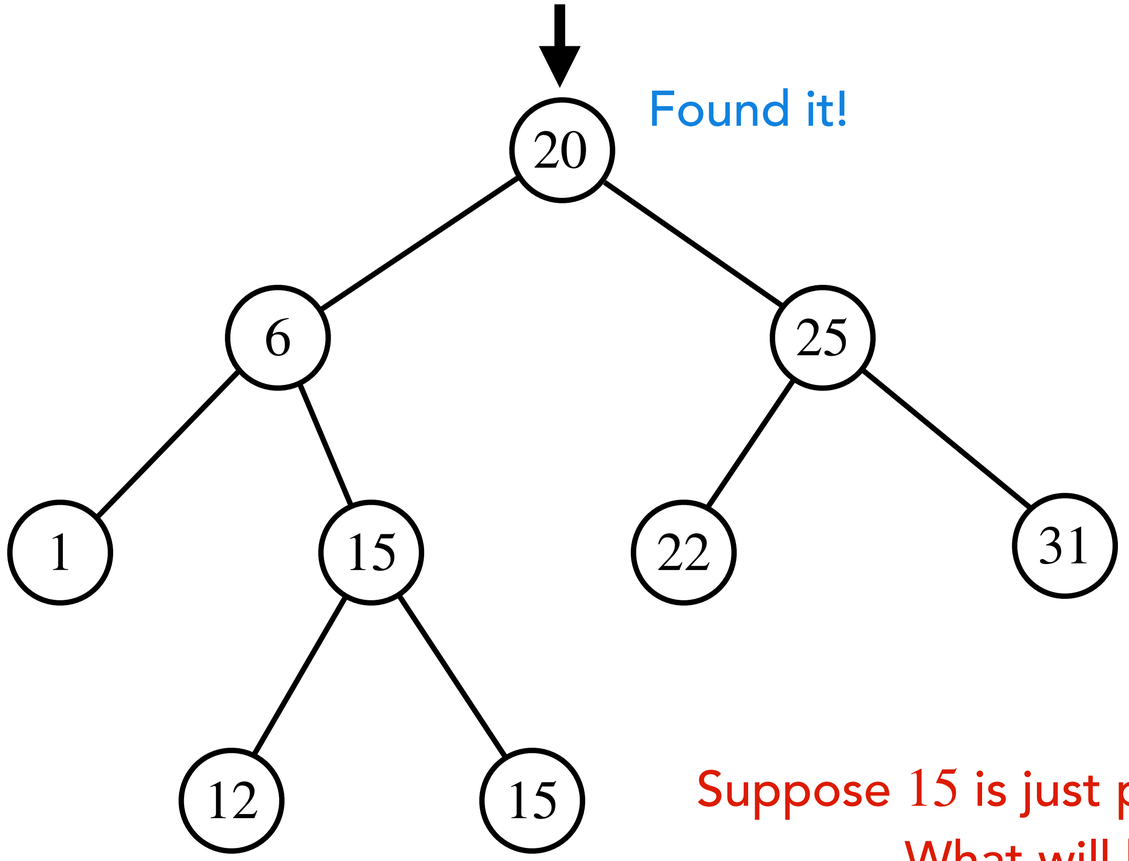
Suppose 15 is just printed in Inorder walk.
What will happen next?

Finding Successor in a BST

Node printed after x in inorder-walk.

Goal: Given a node x of a BST find its successor.

Illustration: Find the successor of the leaf 15 in the below BST.



Recall:

Inorder-Tree-Walk(x):

1. **if** $x \neq$ NIL
2. **Inorder-Tree-Walk**(x . *left*)
3. **print** x . *key*
4. **Inorder-Tree-Walk**(x . *right*)

Suppose 15 is just printed in Inorder walk.
What will happen next?

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

1. **if** $x\,.\,right \neq$ NIL

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

1. **if** $x\,.\,right \neq$ NIL
2. **return Tree-Minimum**$(x\,.\,right)$

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

1. **if** $x\,.\,right \neq$ NIL
2. **return Tree-Minimum**$(x\,.\,right)$
3. **else**

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

1. **if** $x\,.\,right \neq$ NIL
2. **return Tree-Minimum**$(x\,.\,right)$
3. **else**
4. $y = x\,.\,p$

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

1. **if** $x\,.\,right \neq$ NIL
2. **return Tree-Minimum**$(x\,.\,right)$
3. **else**
4. $y = x\,.\,p$
5. **while** $y \neq$ NIL and $x \neq y\,.\,left$

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

```
1.  if x . right ≠ NIL
2.      return Tree-Minimum(x . right)
3.  else
4.      y = x . p
5.      while y ≠ NIL and x ≠ y . left
6.          x = y, y = y . p
```

Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

```
1.  if x . right ≠ NIL
2.      return Tree-Minimum(x . right)
3.  else
4.      y = x . p
5.      while y ≠ NIL and x ≠ y . left
6.          x = y, y = y . p
7.      return y
```

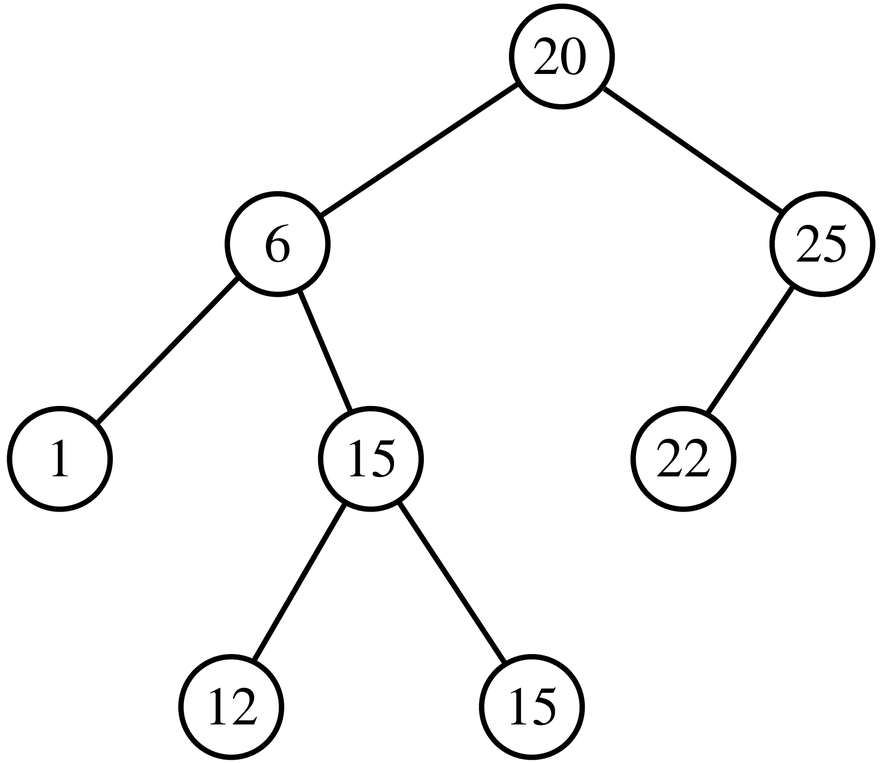
Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

```
1.  if x . right ≠ NIL
2.      return Tree-Minimum(x . right)
3.  else
4.      y = x . p
5.      while y ≠ NIL and x ≠ y . left
6.          x = y, y = y . p
7.      return y
```



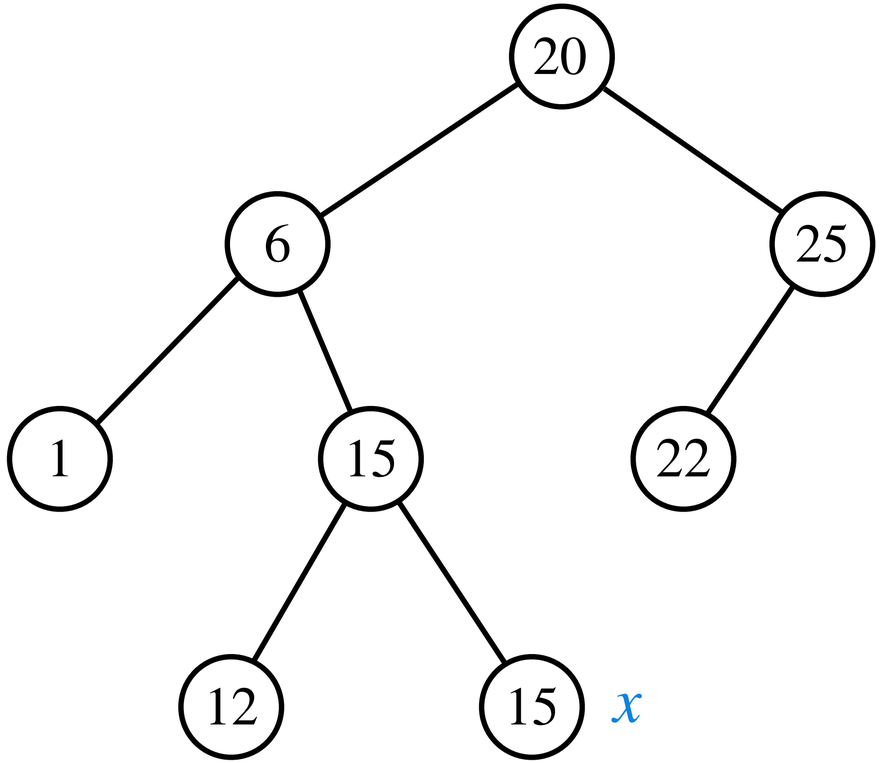
Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

```
1.  if x . right ≠ NIL
2.      return Tree-Minimum(x . right)
3.  else
4.      y = x . p
5.      while y ≠ NIL and x ≠ y . left
6.          x = y, y = y . p
7.      return y
```



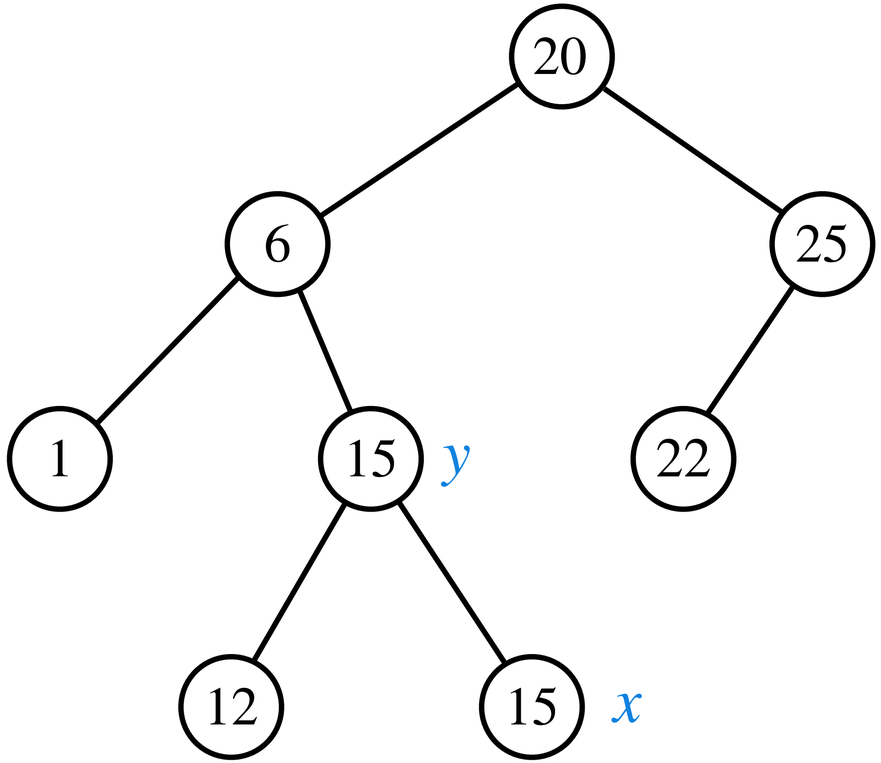
Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

```
1.  if x.right ≠ NIL
2.      return Tree-Minimum(x.right)
3.  else
4.      y = x.p
5.      while y ≠ NIL and x ≠ y.left
6.          x = y, y = y.p
7.      return y
```



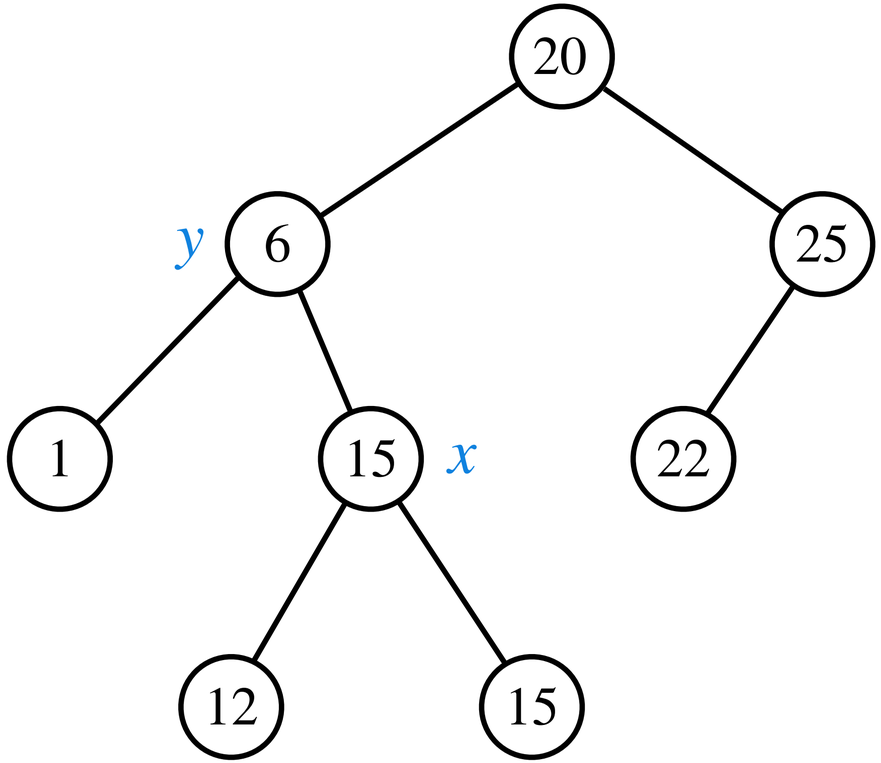
Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

```
1.  if x . right ≠ NIL
2.      return Tree-Minimum(x . right)
3.  else
4.      y = x . p
5.      while y ≠ NIL and x ≠ y . left
6.          x = y, y = y . p
7.      return y
```



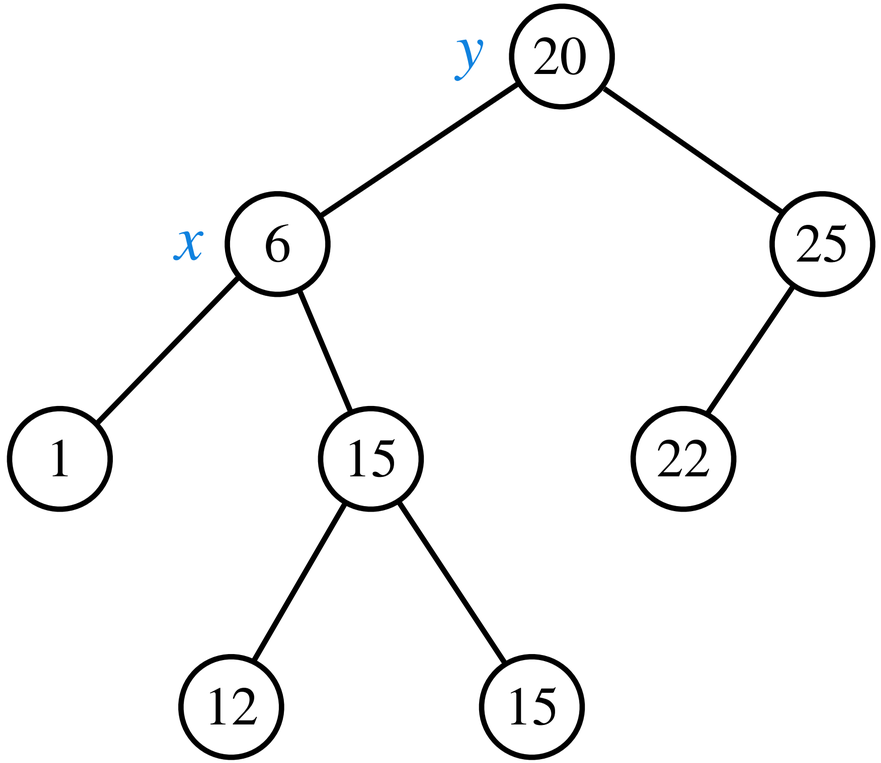
Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

Tree-Successor(x)**:**

```
1.  if x . right ≠ NIL
2.      return Tree-Minimum(x . right)
3.  else
4.      y = x . p
5.      while y ≠ NIL and x ≠ y . left
6.          x = y, y = y . p
7.      return y
```

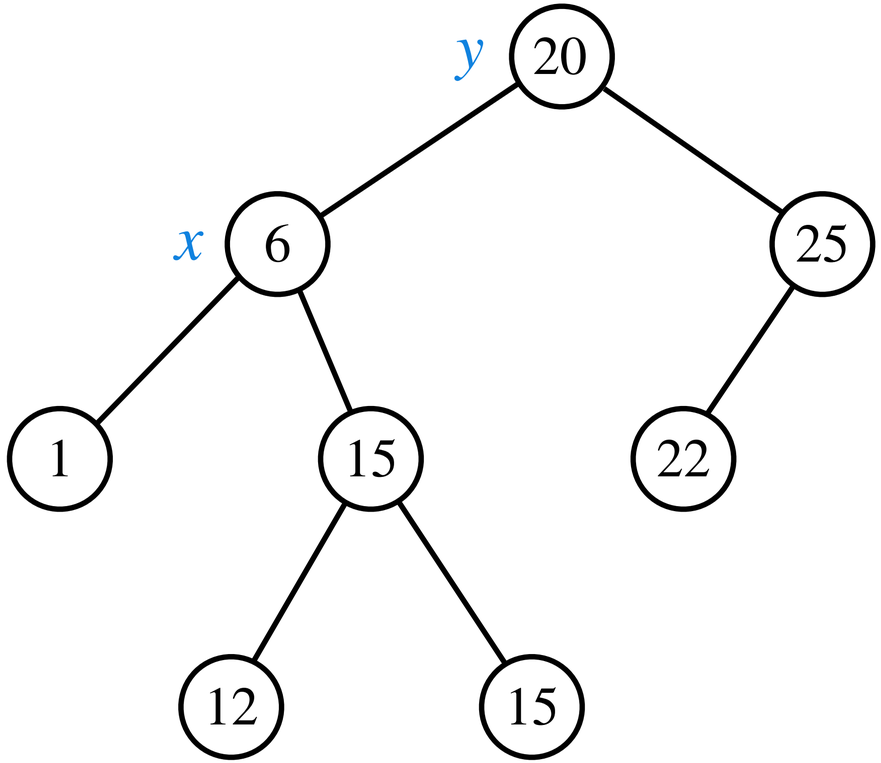


Finding Successor in a BST

Goal: Given a node x of a BST find its successor.

Algorithm: Call **Tree-Successor**(x) to find x's successor in T.

```
Tree-Successor(x):
1.   if x . right ≠ NIL
2.       return Tree-Minimum(x . right)
3.   else
4.       y = x . p
5.       while y ≠ NIL and x ≠ y . left
6.           x = y, y = y . p
7.       return y
```



Runtime: $\Theta(h)$, where h = height of T.