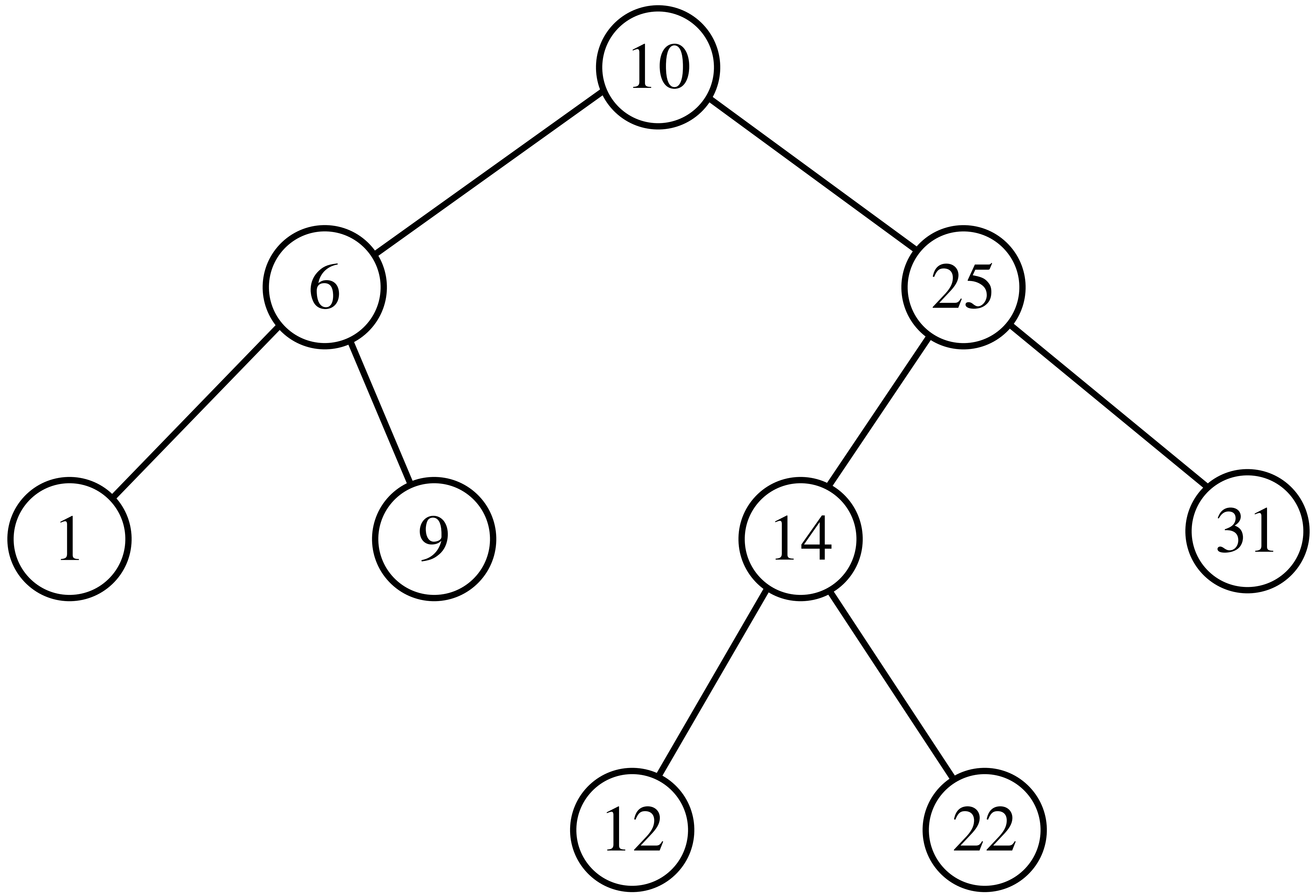
Lecture 6

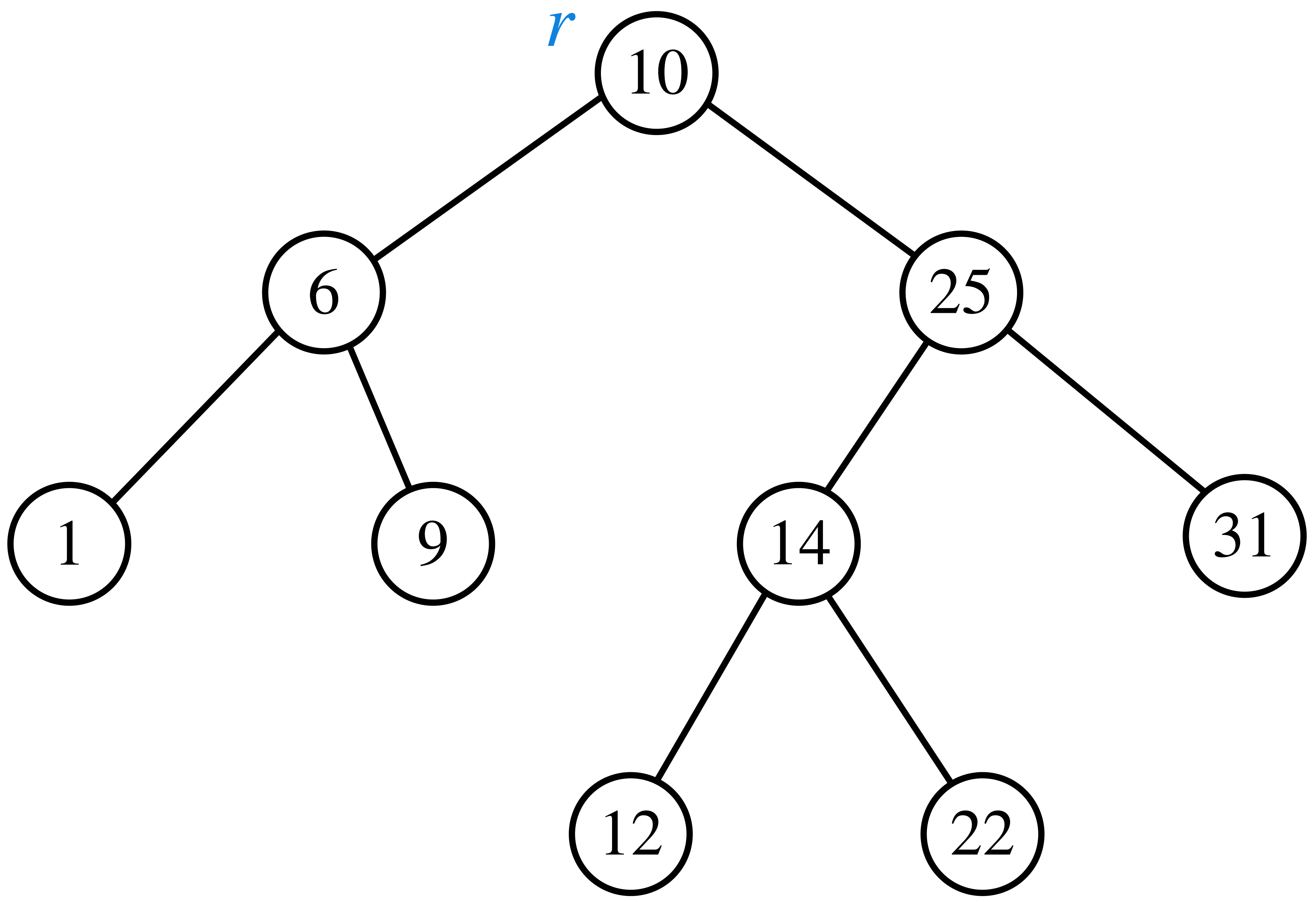
BST Terminology and Operations

BST: Basic Terminology

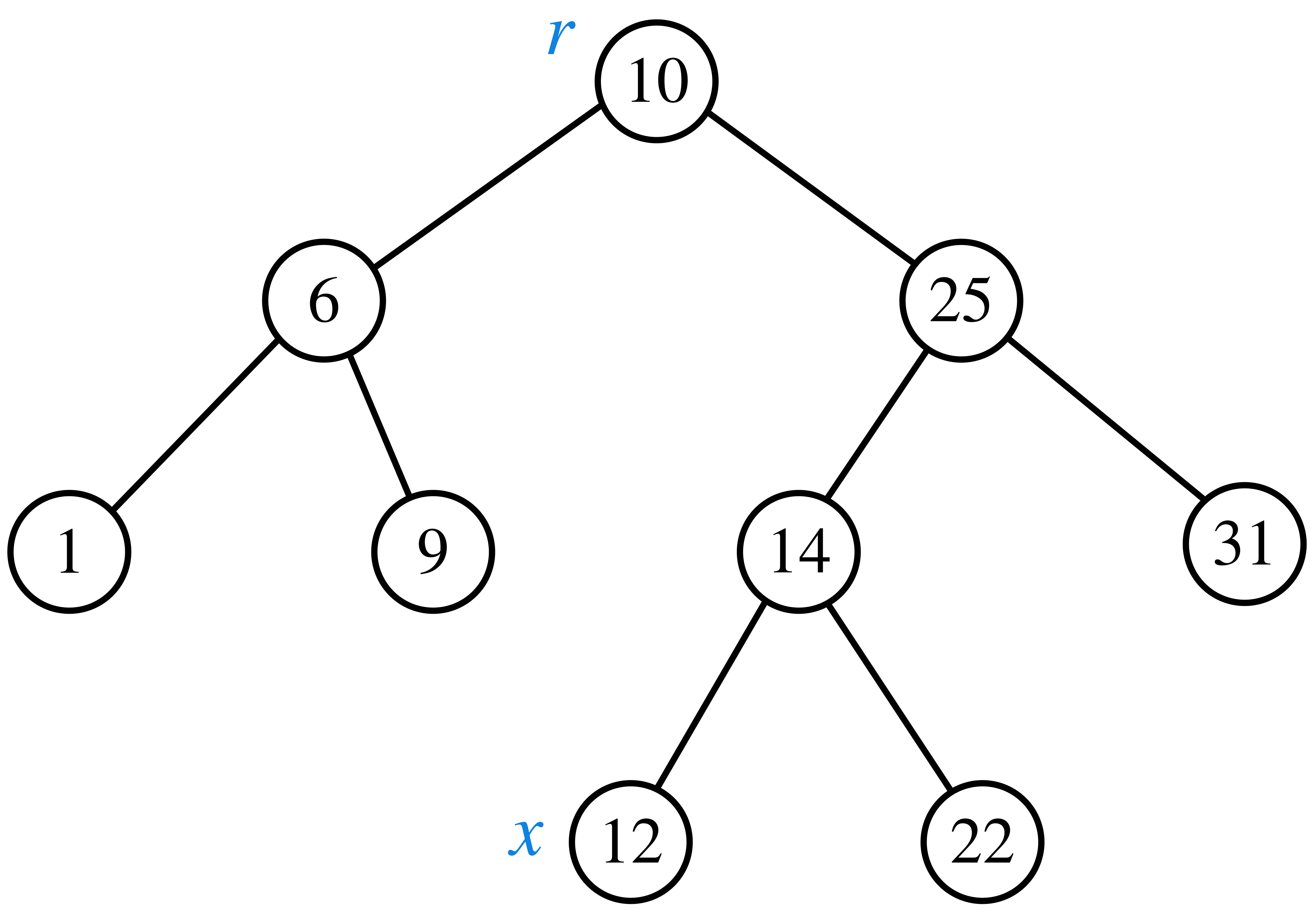
BST: Basic Terminology



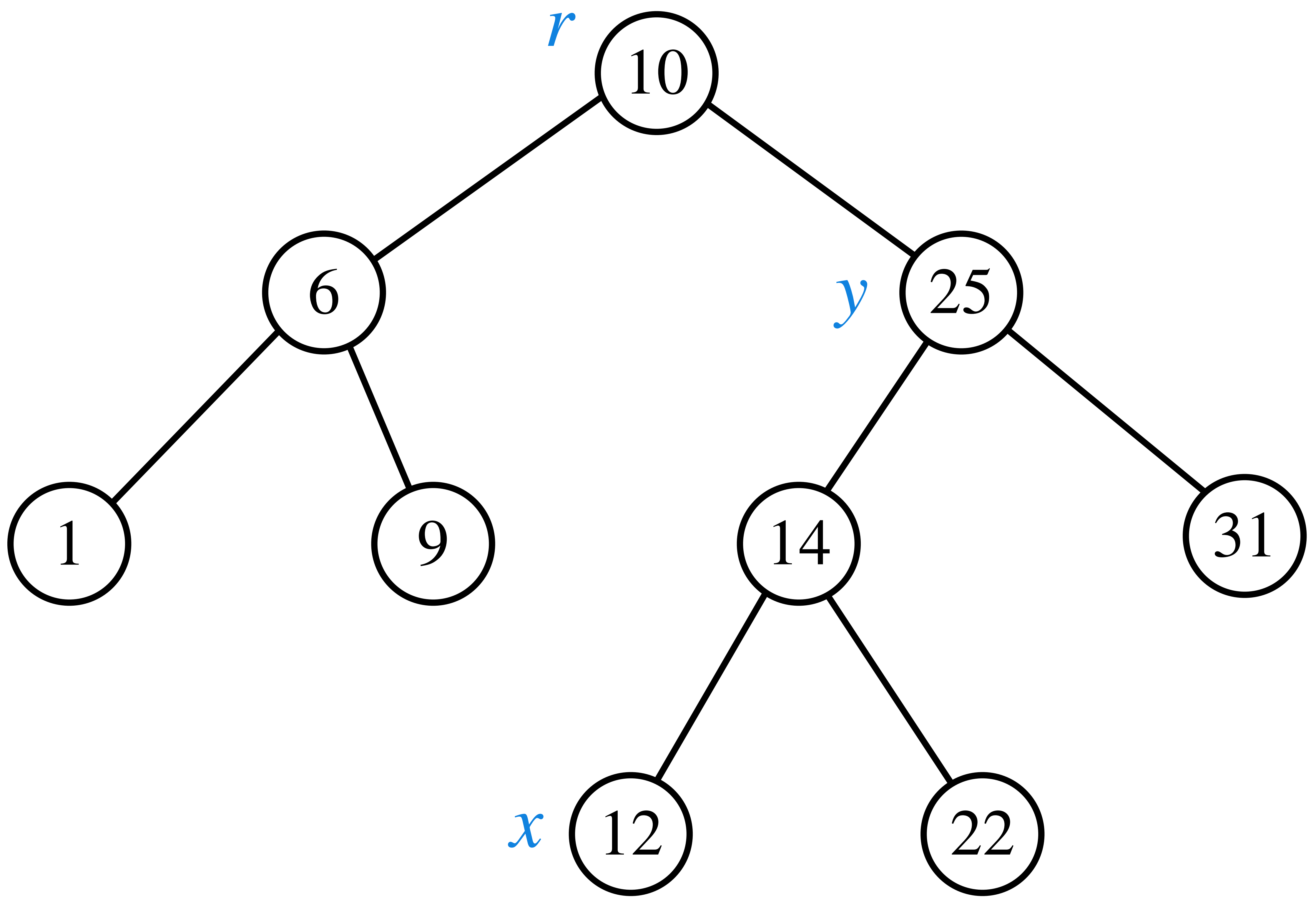
BST: Basic Terminology



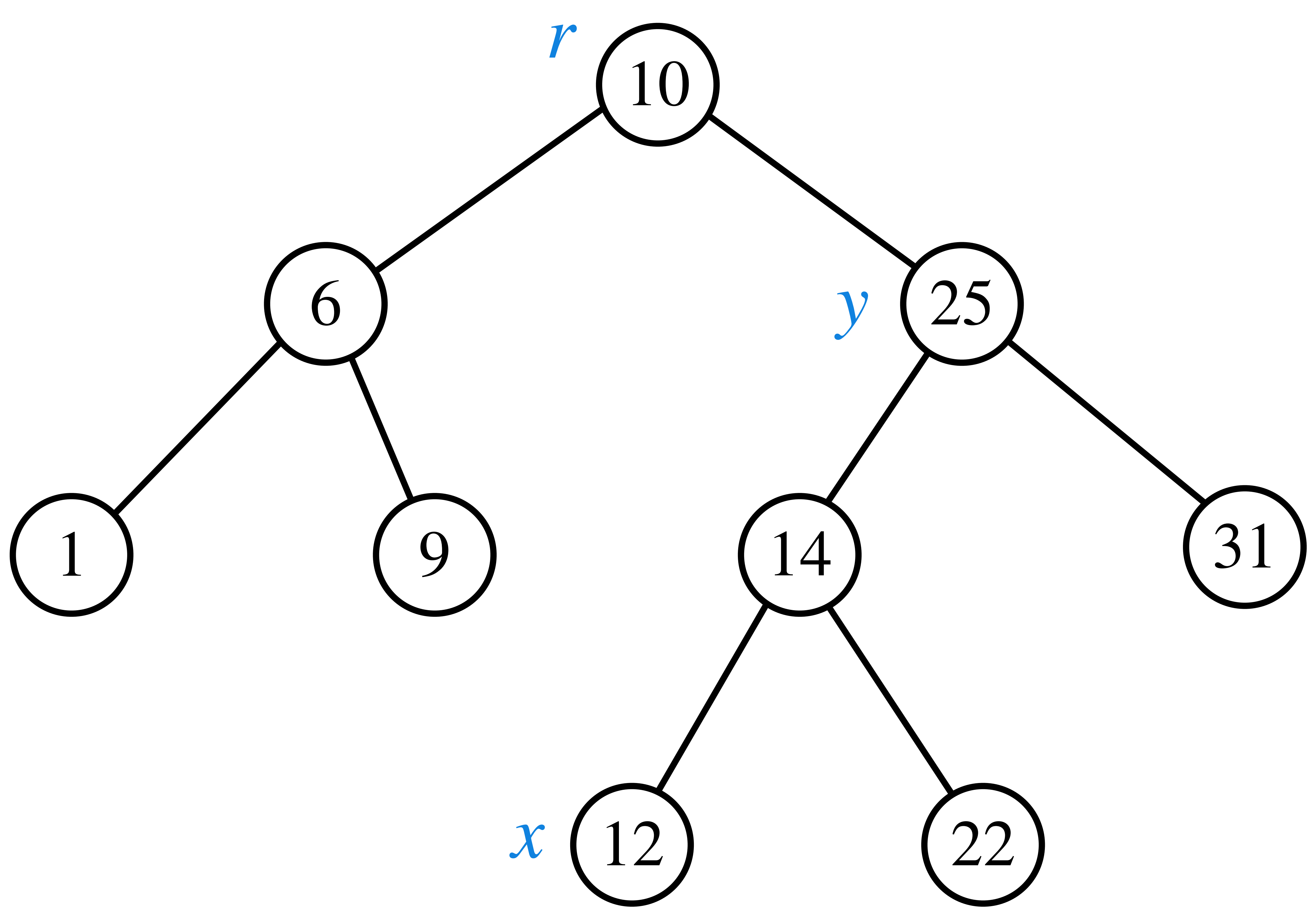
BST: Basic Terminology



BST: Basic Terminology

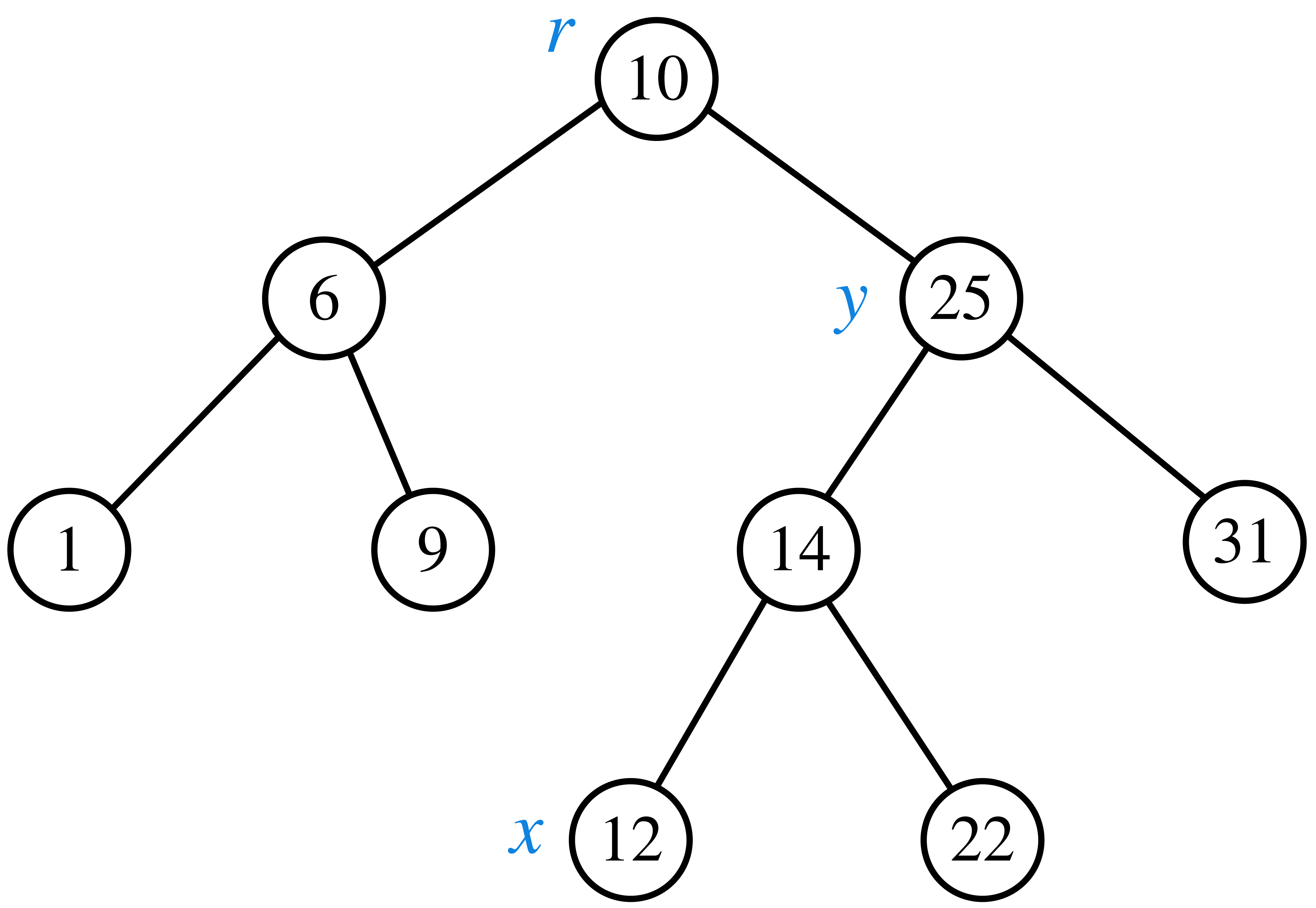


BST: Basic Terminology



y is an ancestor of x,

BST: Basic Terminology

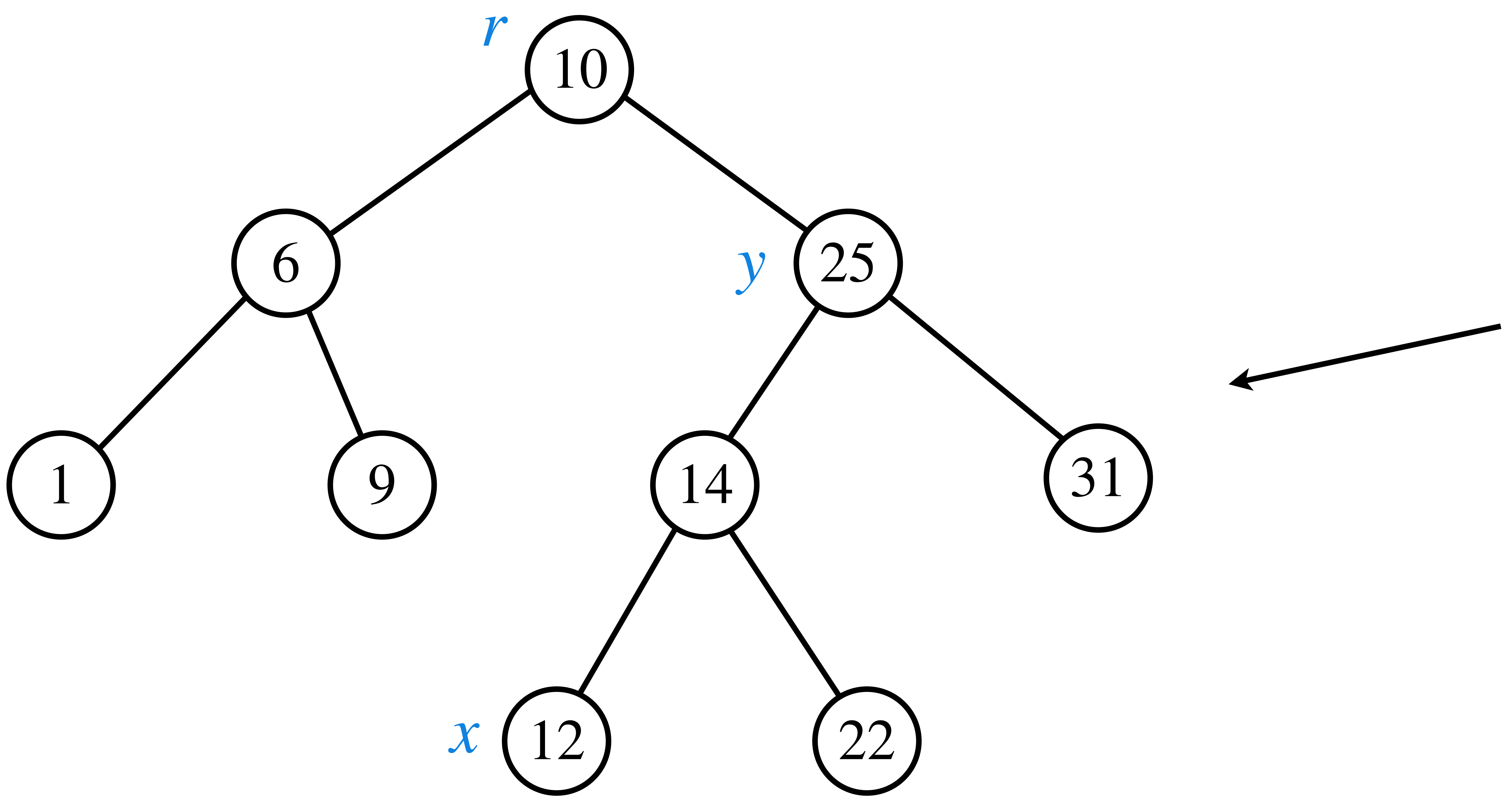


y is an ancestor of x,

x is a descendant of y

BST: Basic Terminology

Defn: Let x be a node in a tree T with root r.

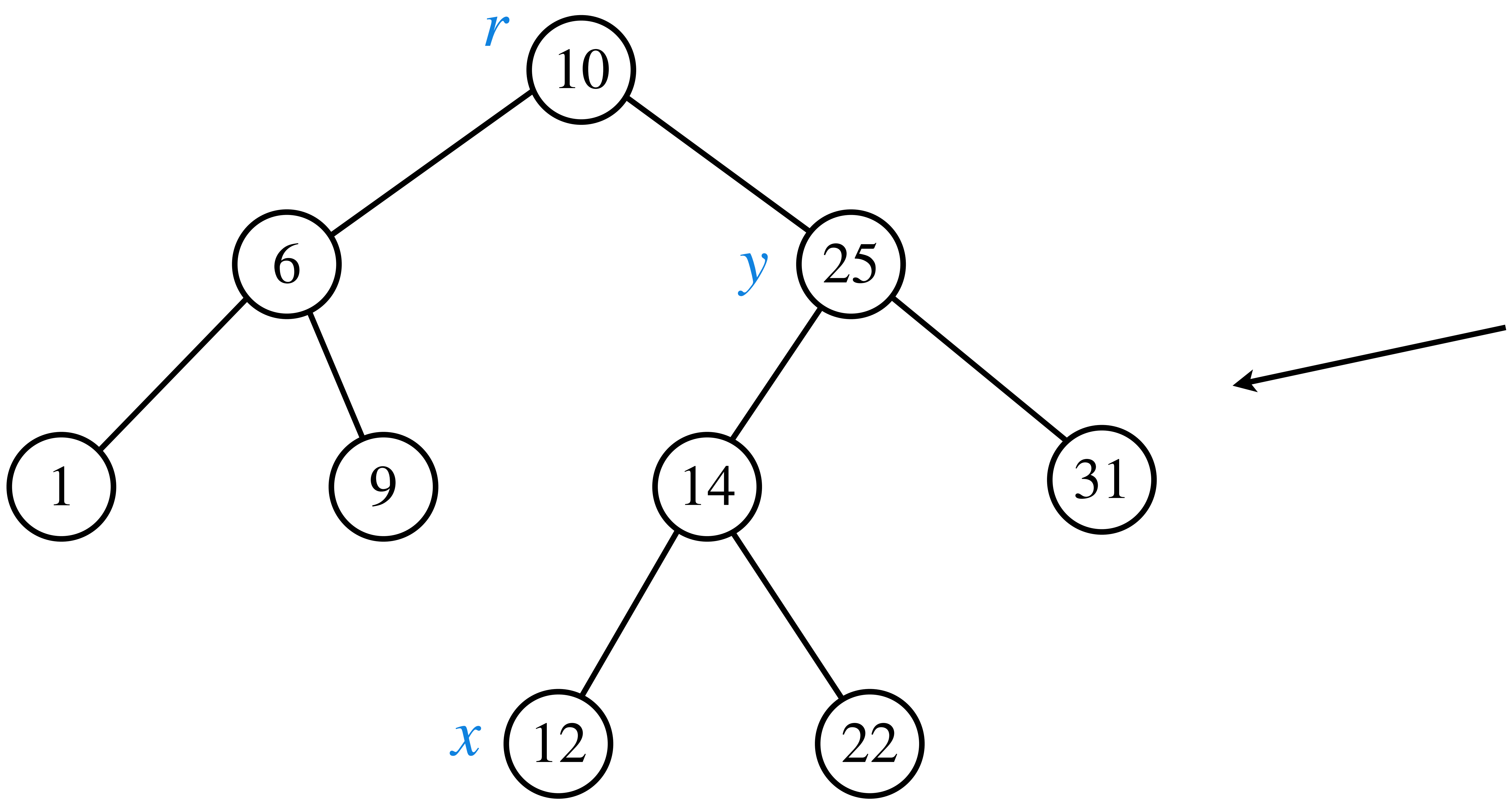


y is an ancestor of x,

x is a descendant of y

BST: Basic Terminology

Defn: Let x be a node in a tree T with root r. Then any node y on the unique path from r to x is

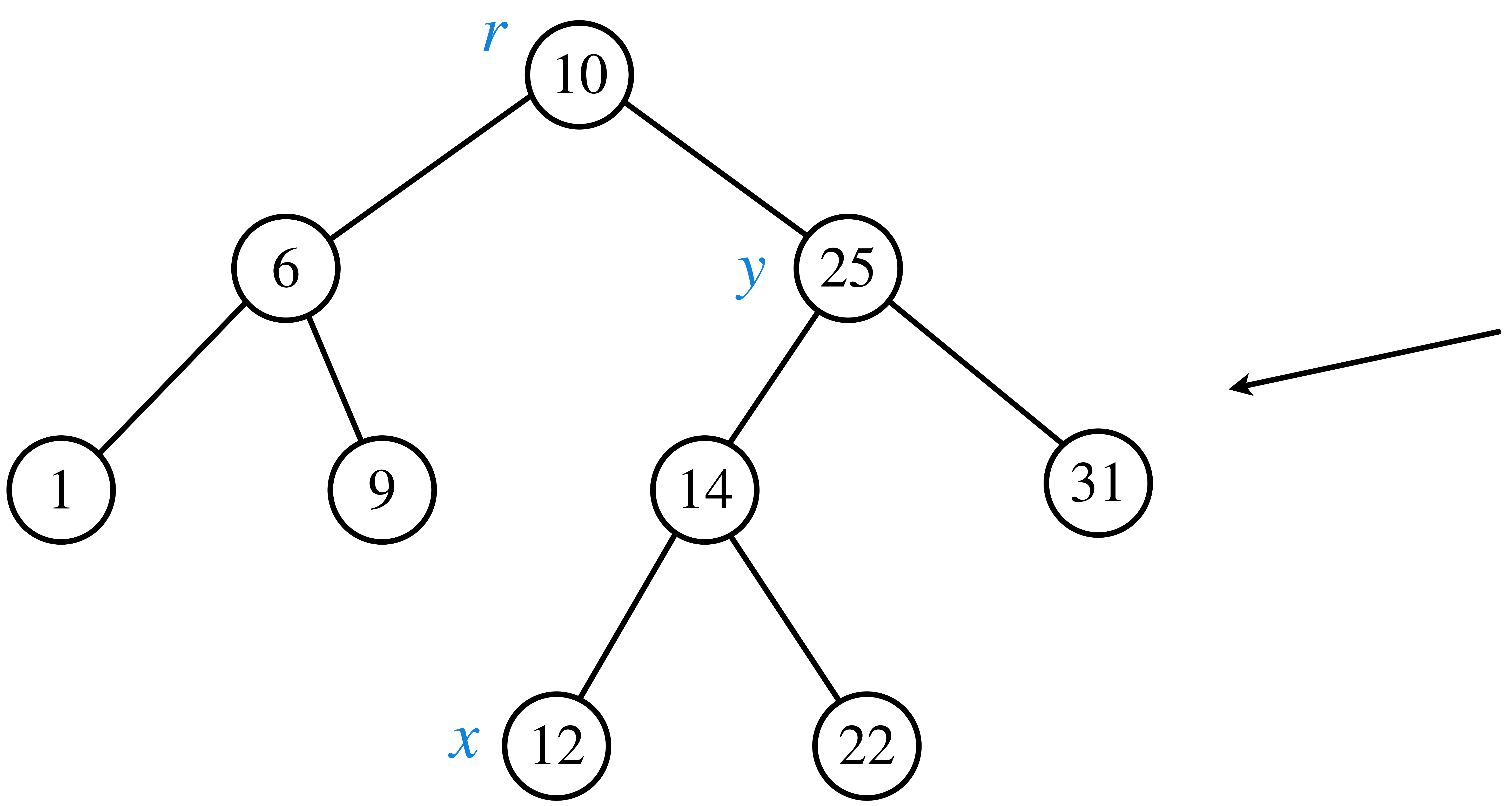


y is an ancestor of x,

x is a descendant of y

BST: Basic Terminology

Defn: Let x be a node in a tree T with root r. Then any node y on the unique path from r to x is called ancestor of x

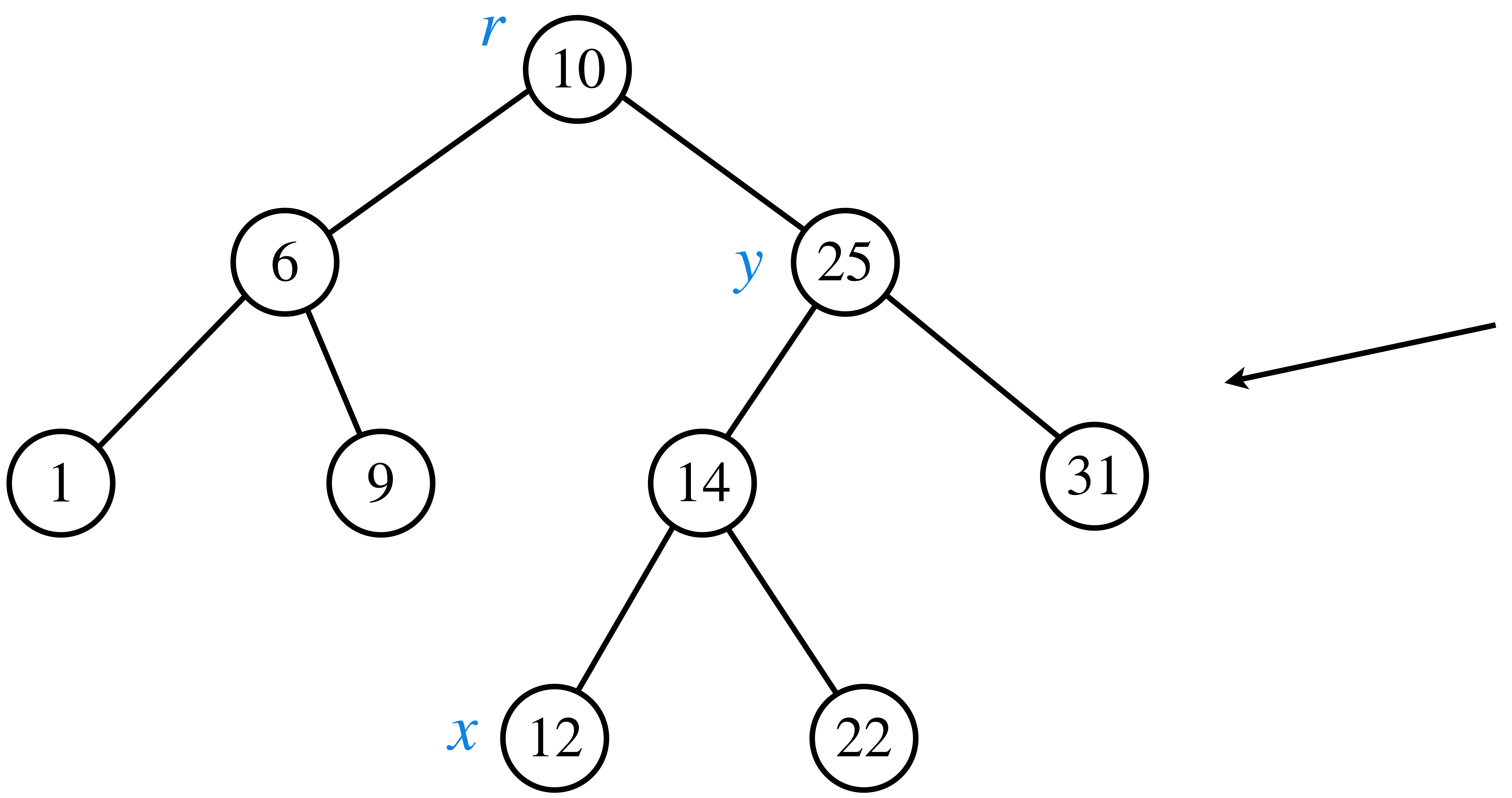


y is an ancestor of x,

x is a descendant of y

BST: Basic Terminology

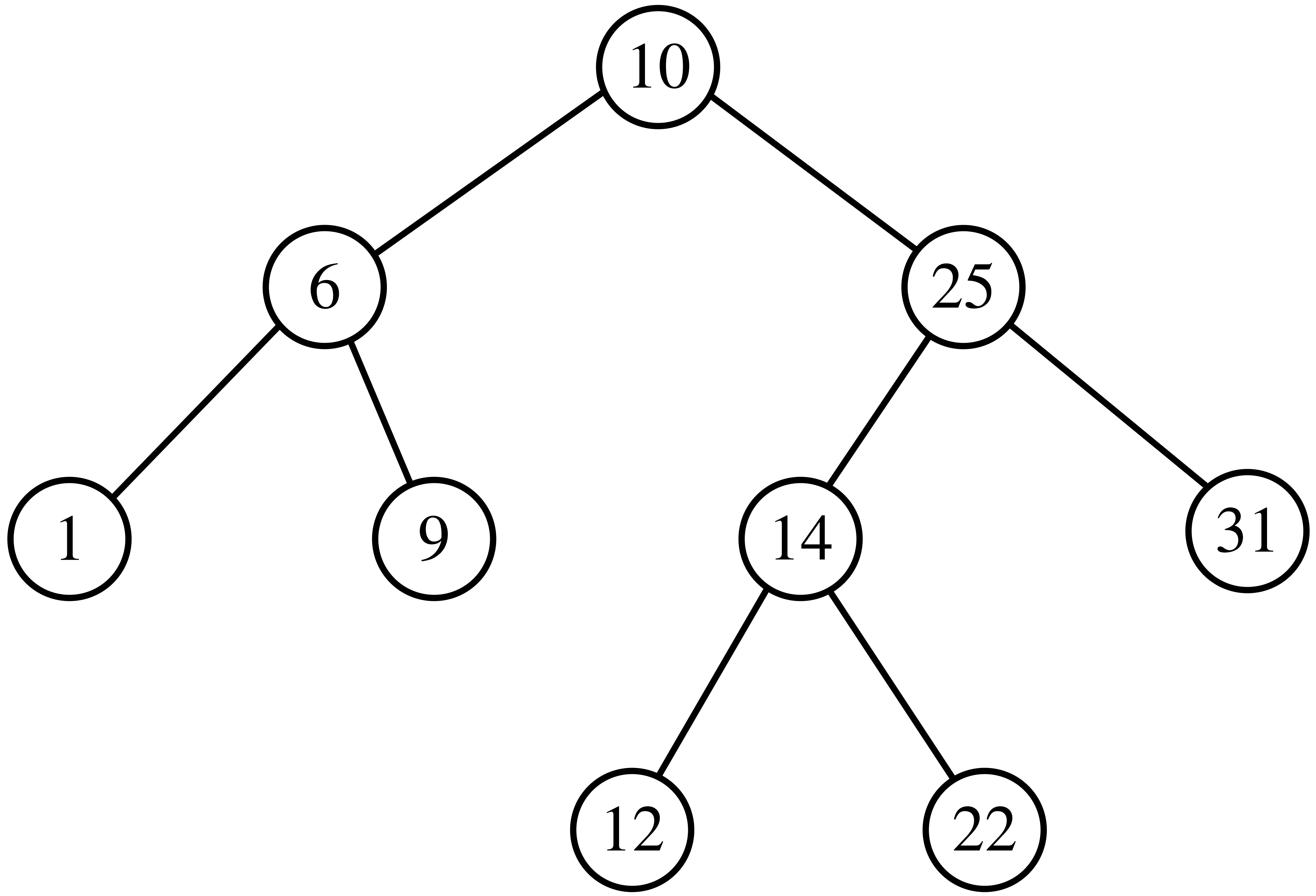
Defn: Let x be a node in a tree T with root r. Then any node y on the unique path from r to x is called ancestor of x and x is called descendant of y.



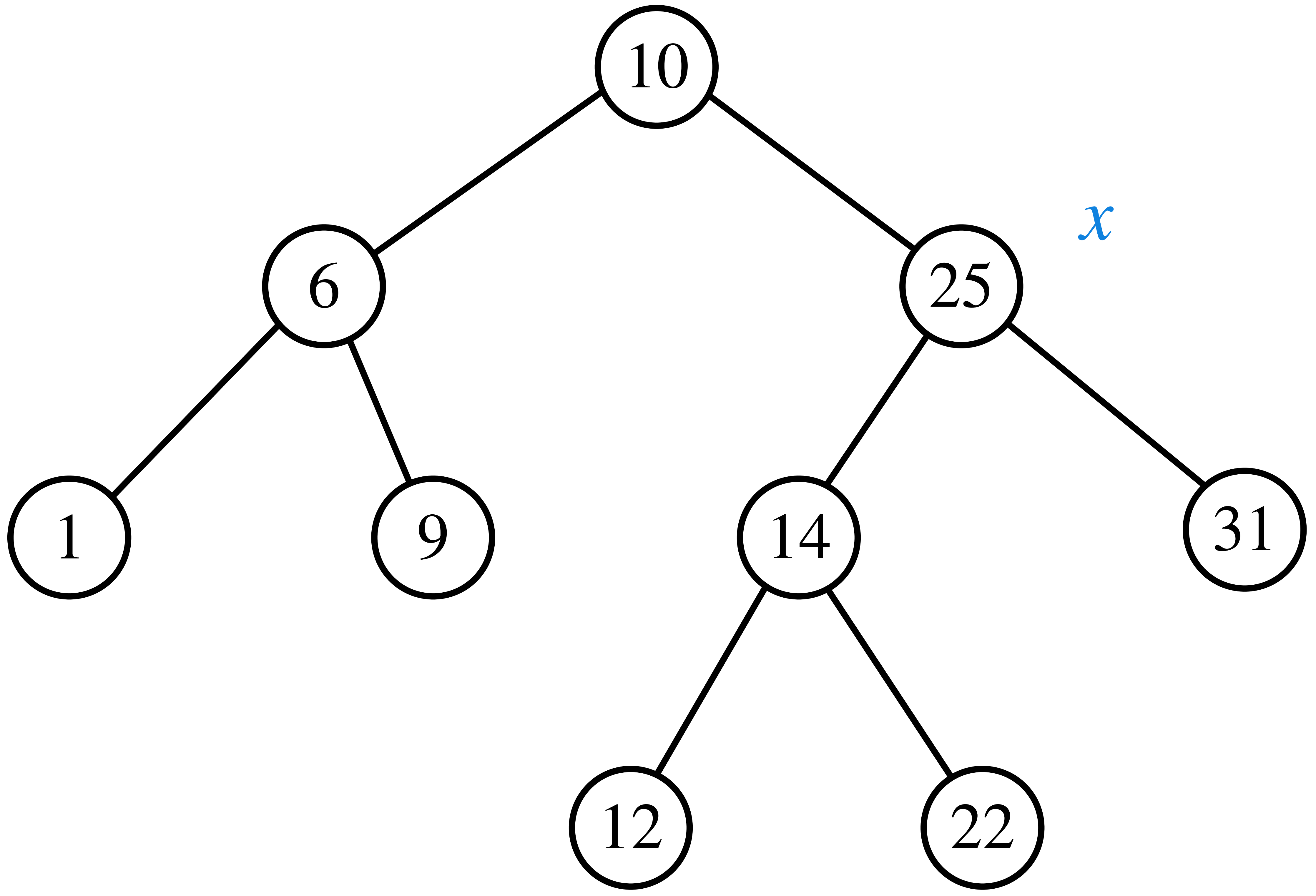
y is an ancestor of x,
x is a descendant of y

BST: Basic Terminology

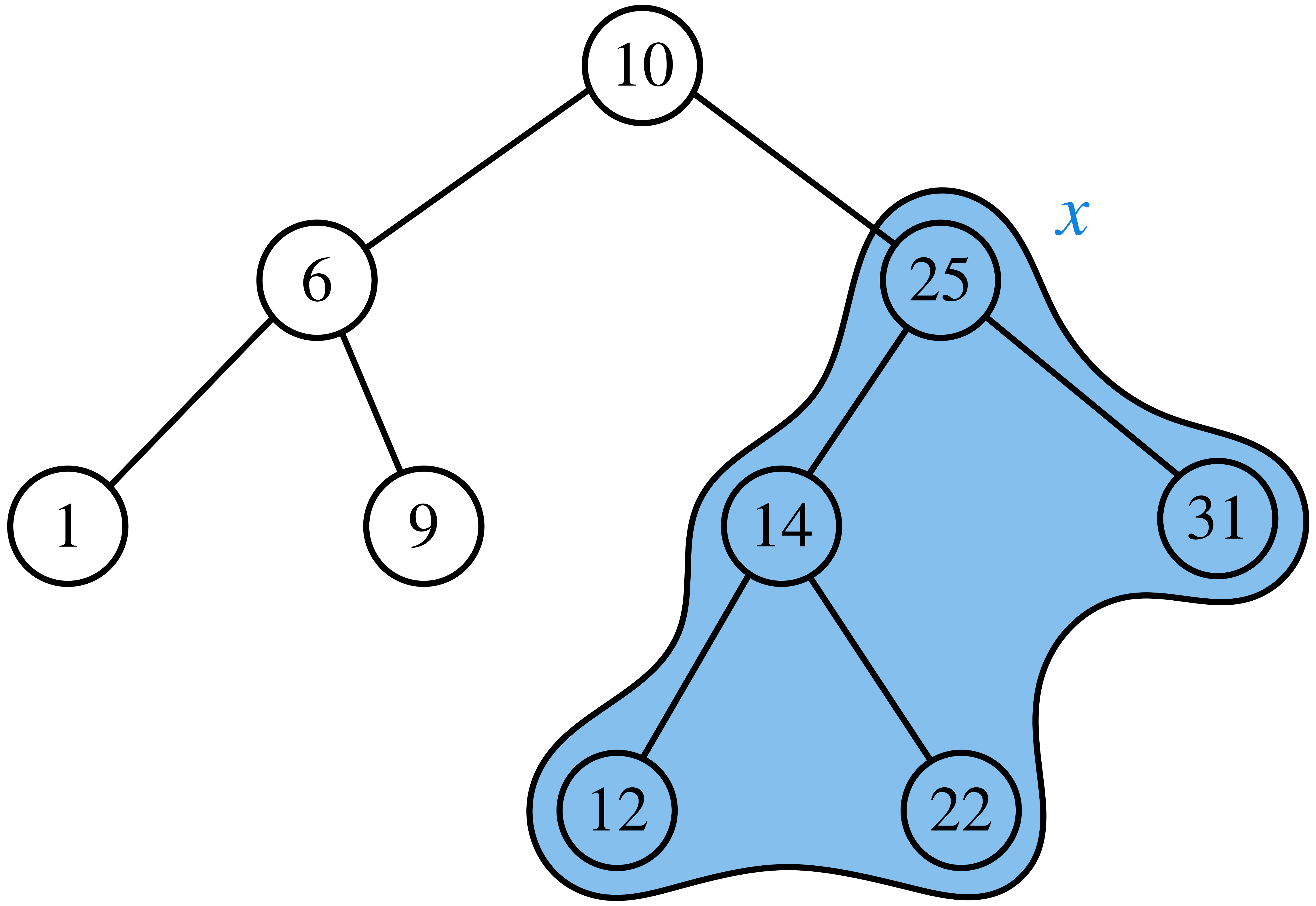
BST: Basic Terminology



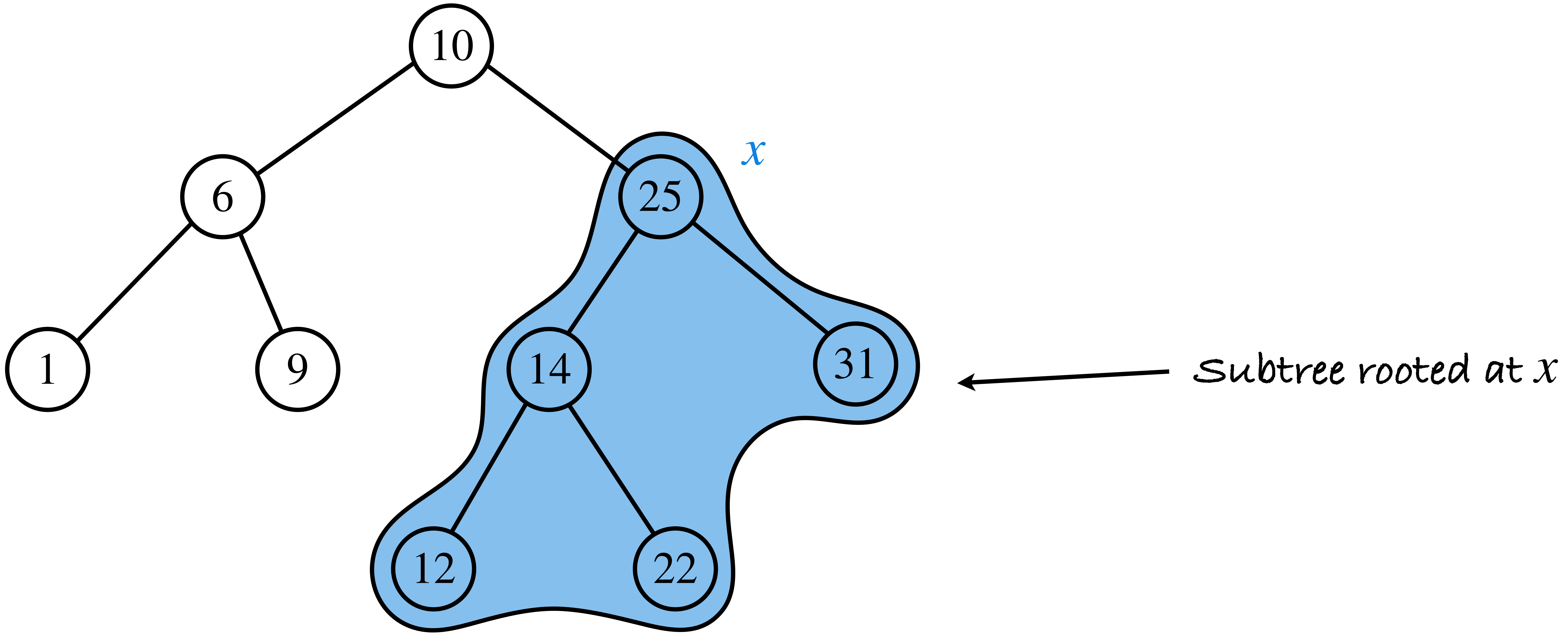
BST: Basic Terminology



BST: Basic Terminology

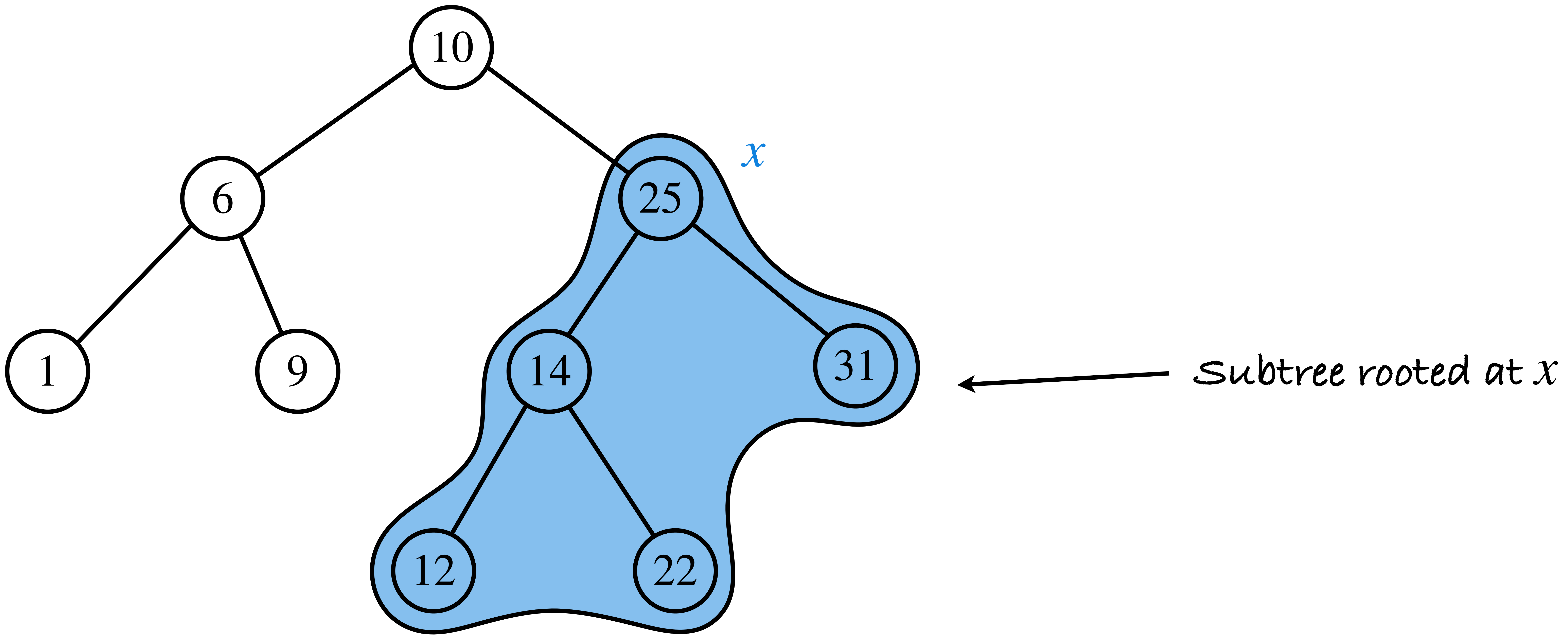


BST: Basic Terminology



BST: Basic Terminology

Defn: Subtree rooted at x is the tree containing only descendants of x.



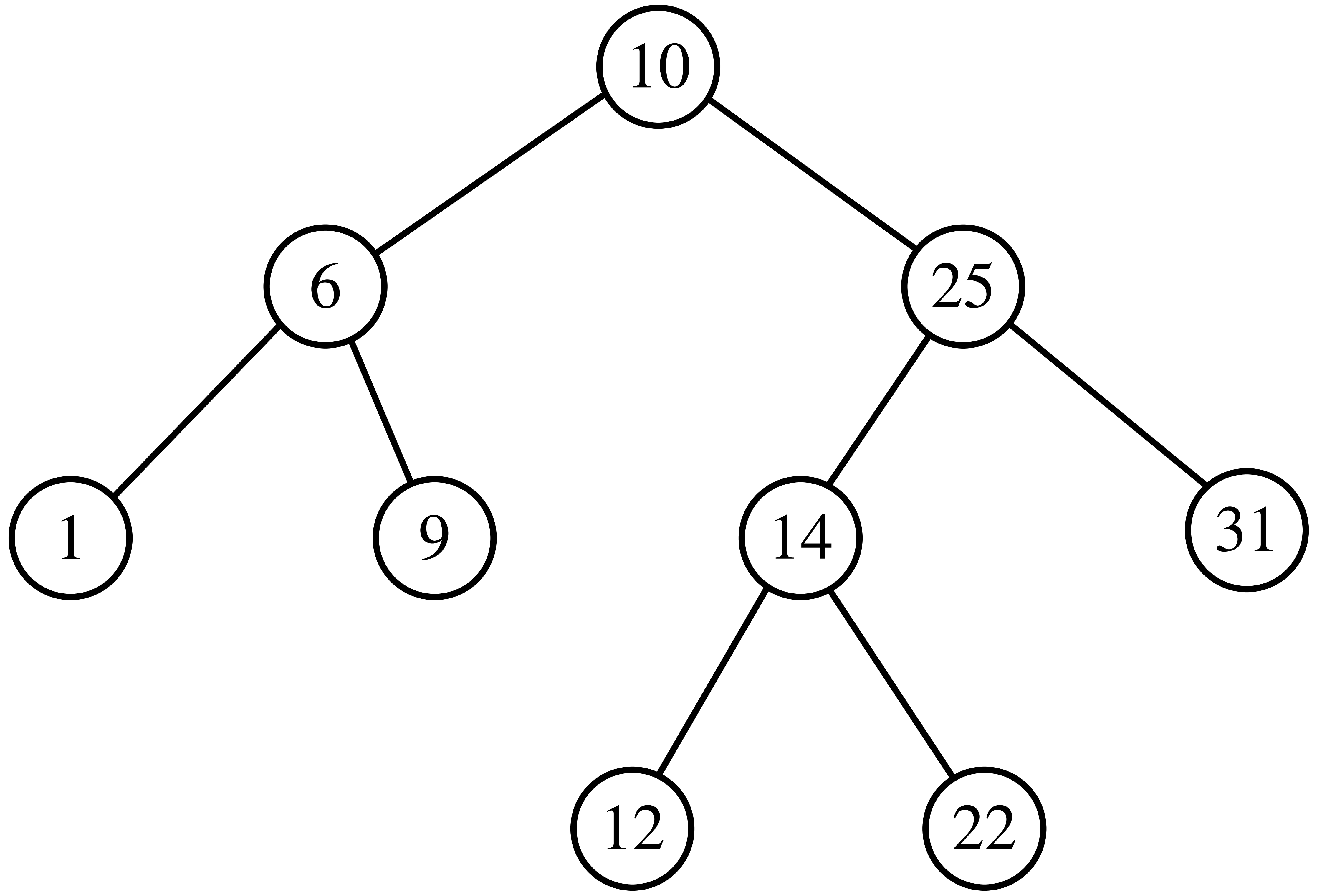
BST: Basic Terminology

BST: Basic Terminology

Defn: Two nodes with the same parent are called siblings.

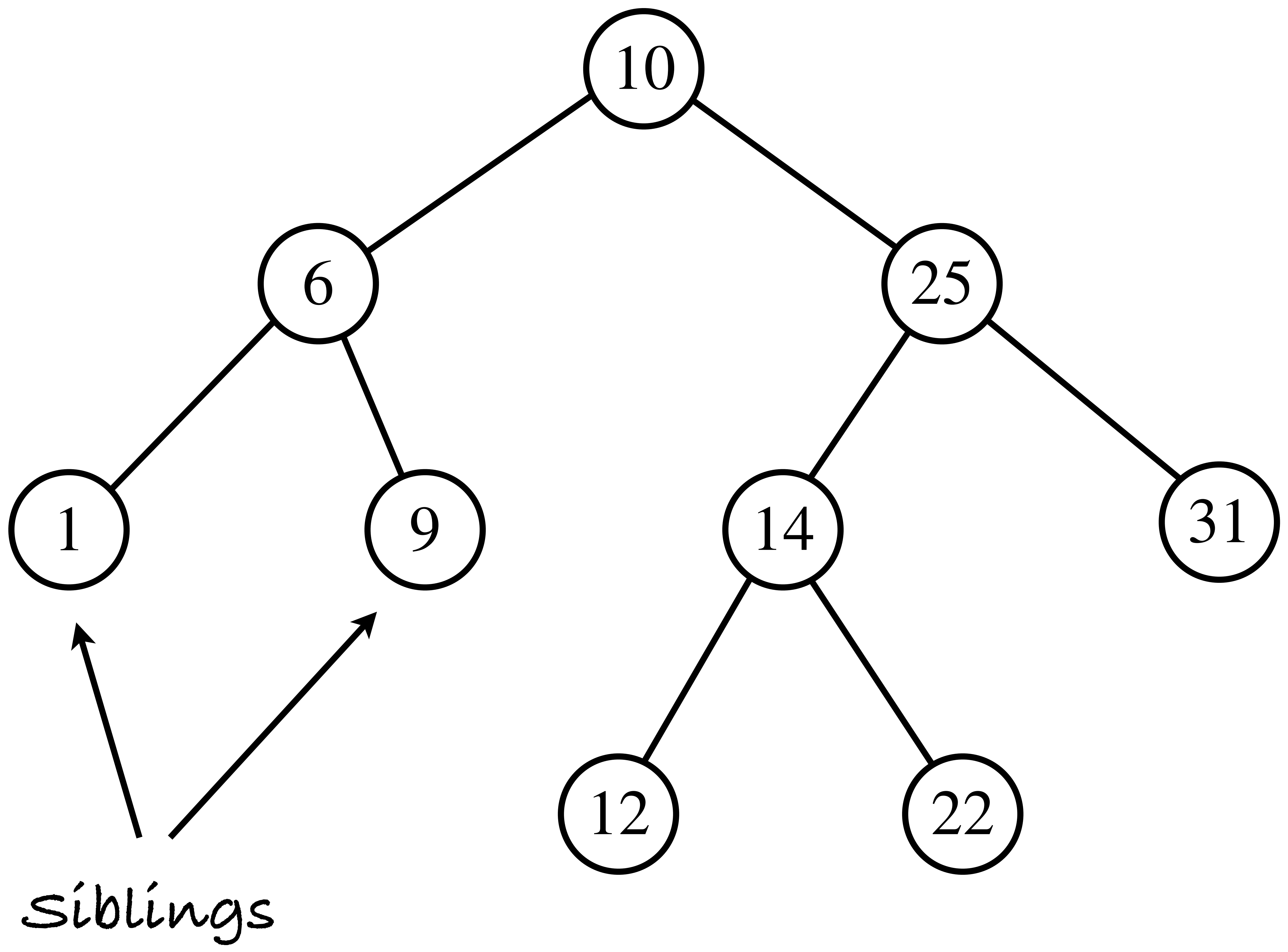
BST: Basic Terminology

Defn: Two nodes with the same parent are called siblings.



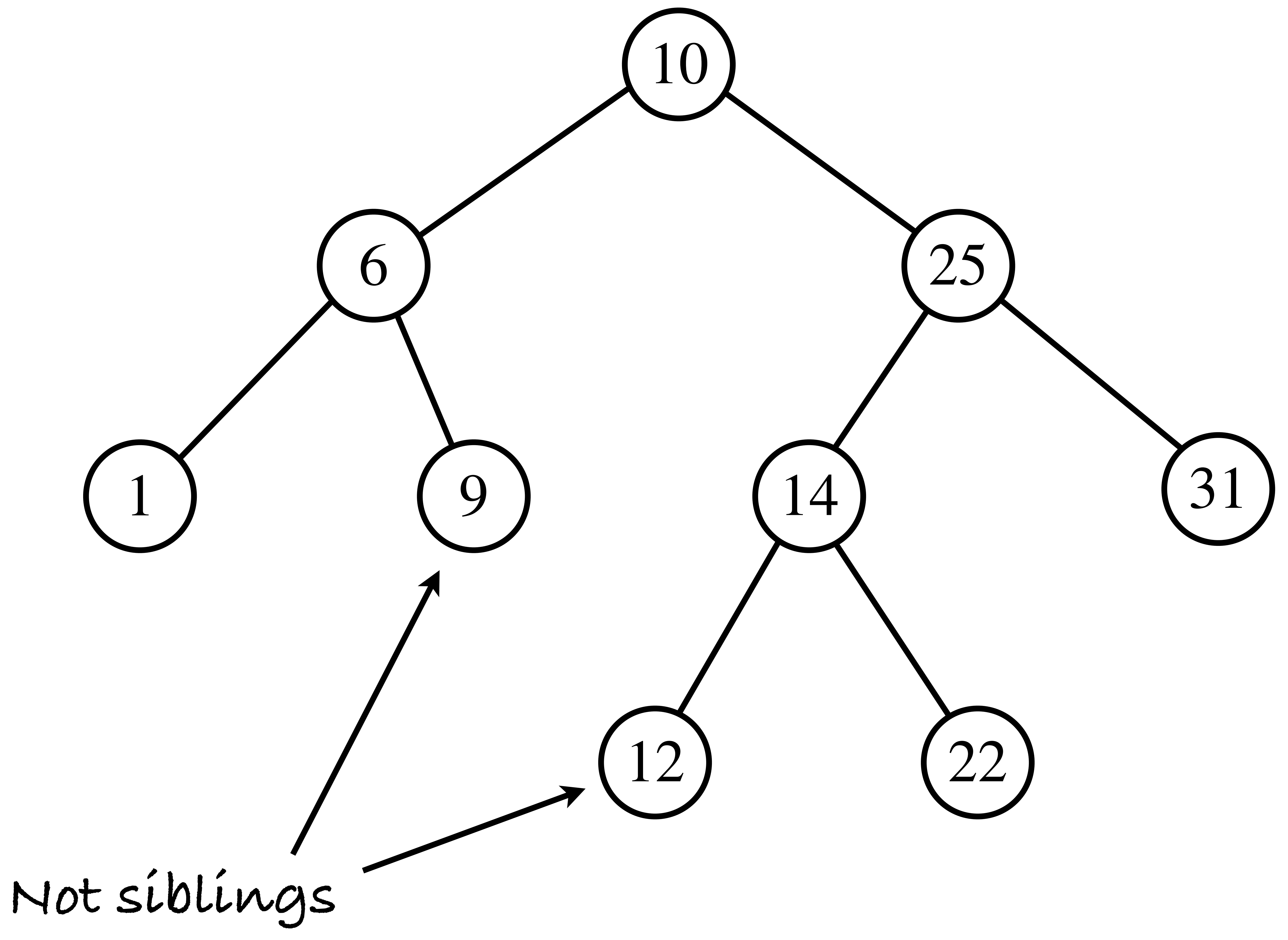
BST: Basic Terminology

Defn: Two nodes with the same parent are called siblings.



BST: Basic Terminology

Defn: Two nodes with the same parent are called siblings.



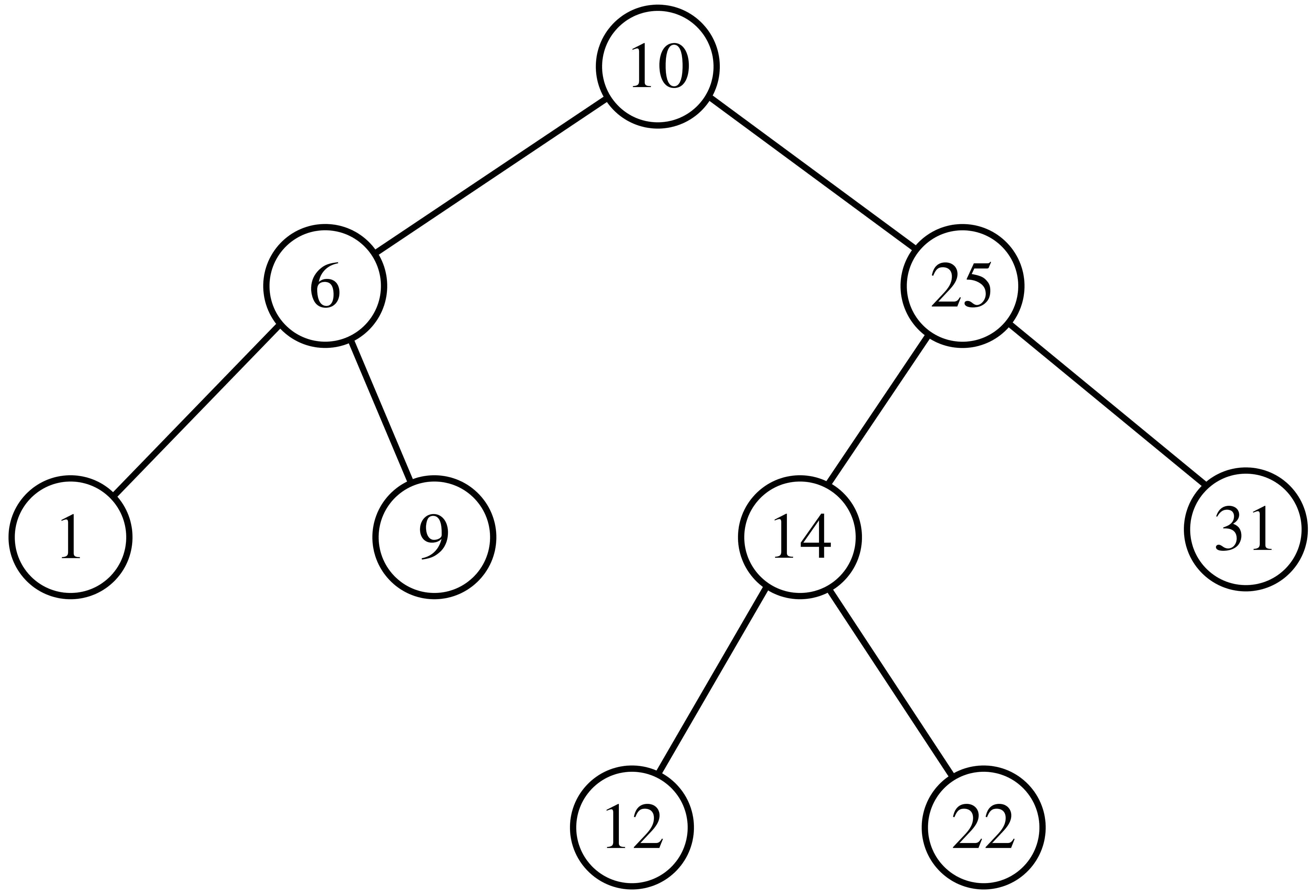
BST: Basic Terminology

BST: Basic Terminology

Defn: Nodes with no children are called leaves.

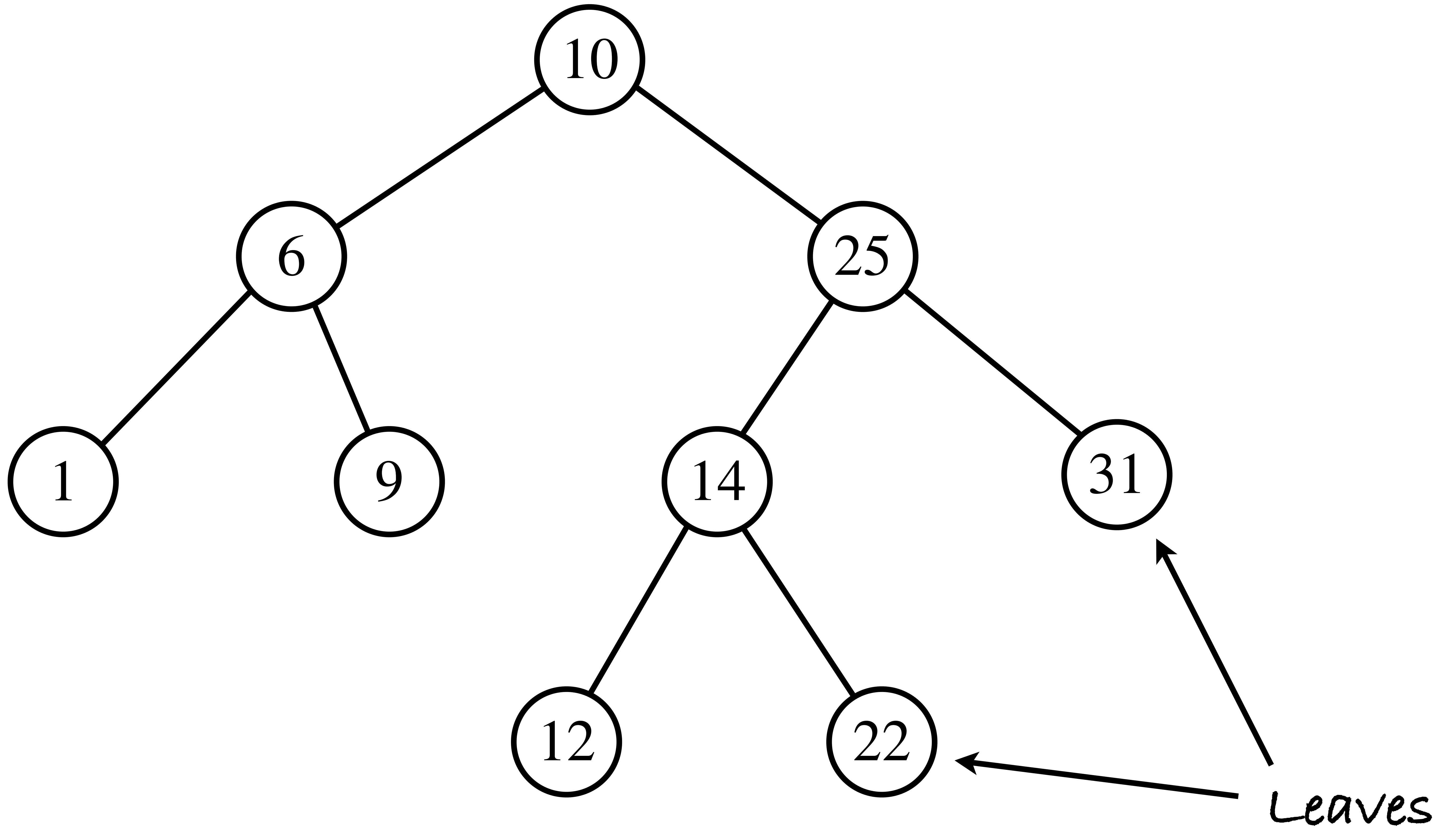
BST: Basic Terminology

Defn: Nodes with no children are called leaves.



BST: Basic Terminology

Defn: Nodes with no children are called leaves.



BST: Basic Terminology

BST: Basic Terminology

Defn: The height of a node in a tree

BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path

BST: Basic Terminology

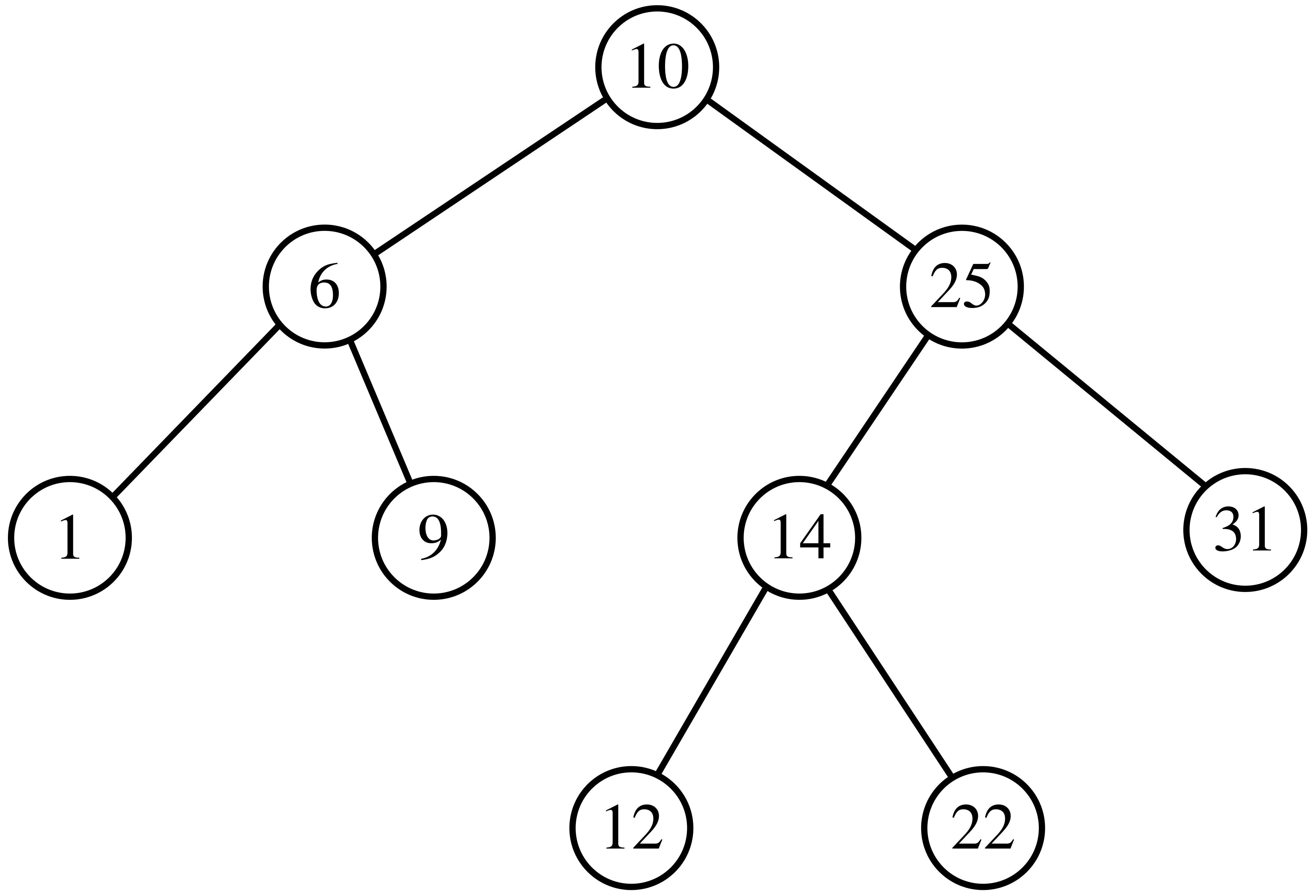
Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf.

BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.

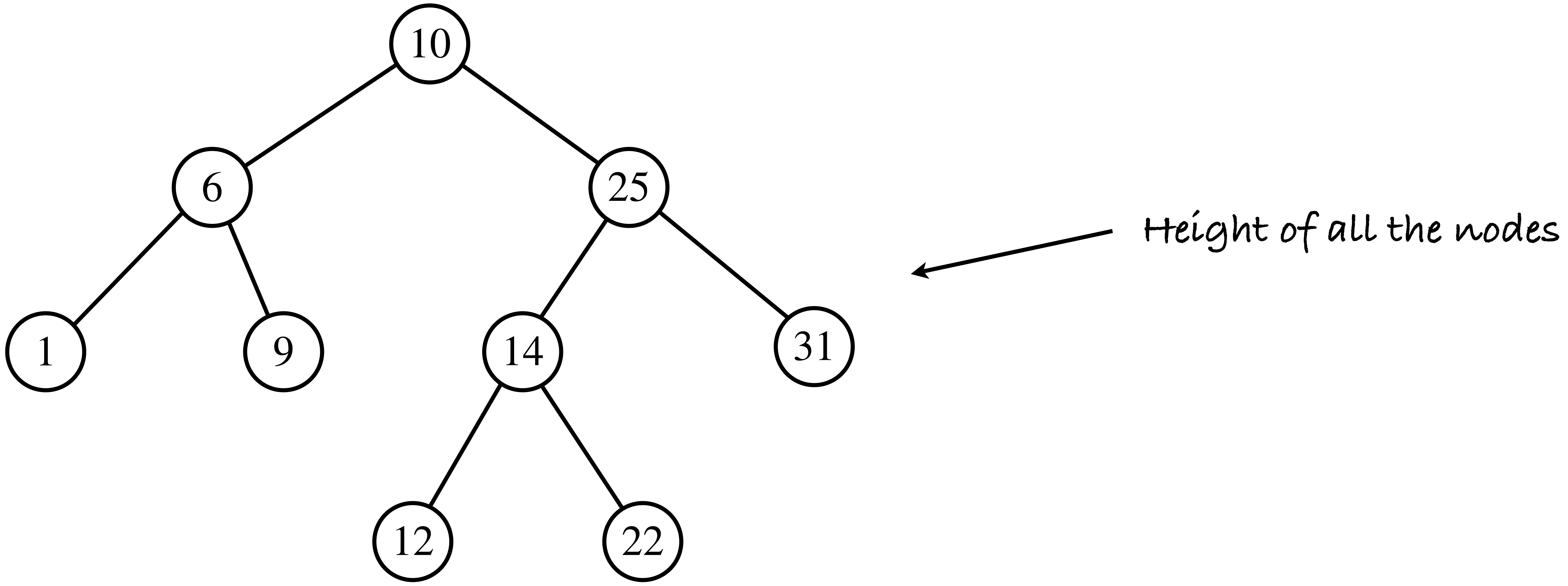
BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



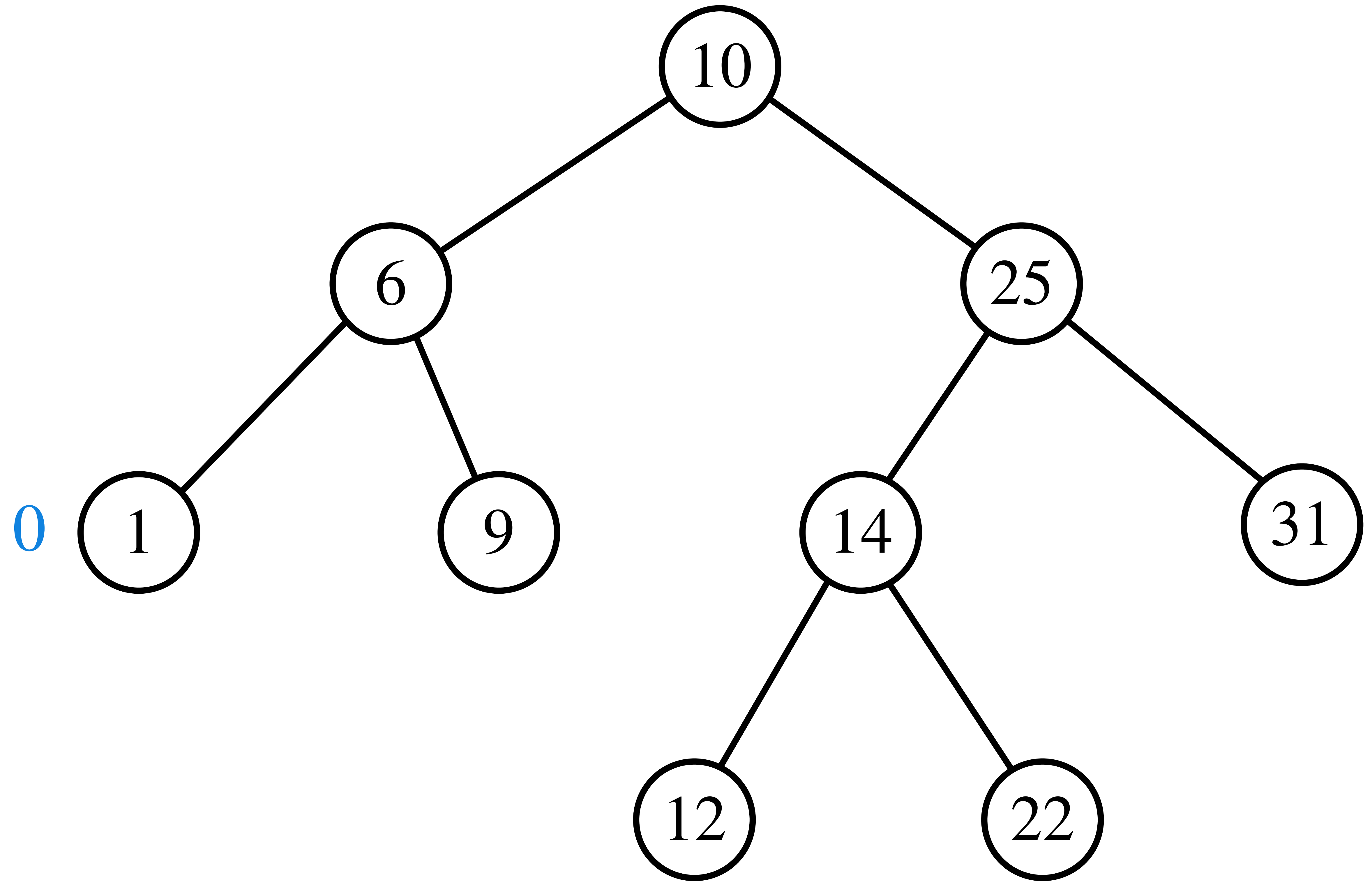
BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



BST: Basic Terminology

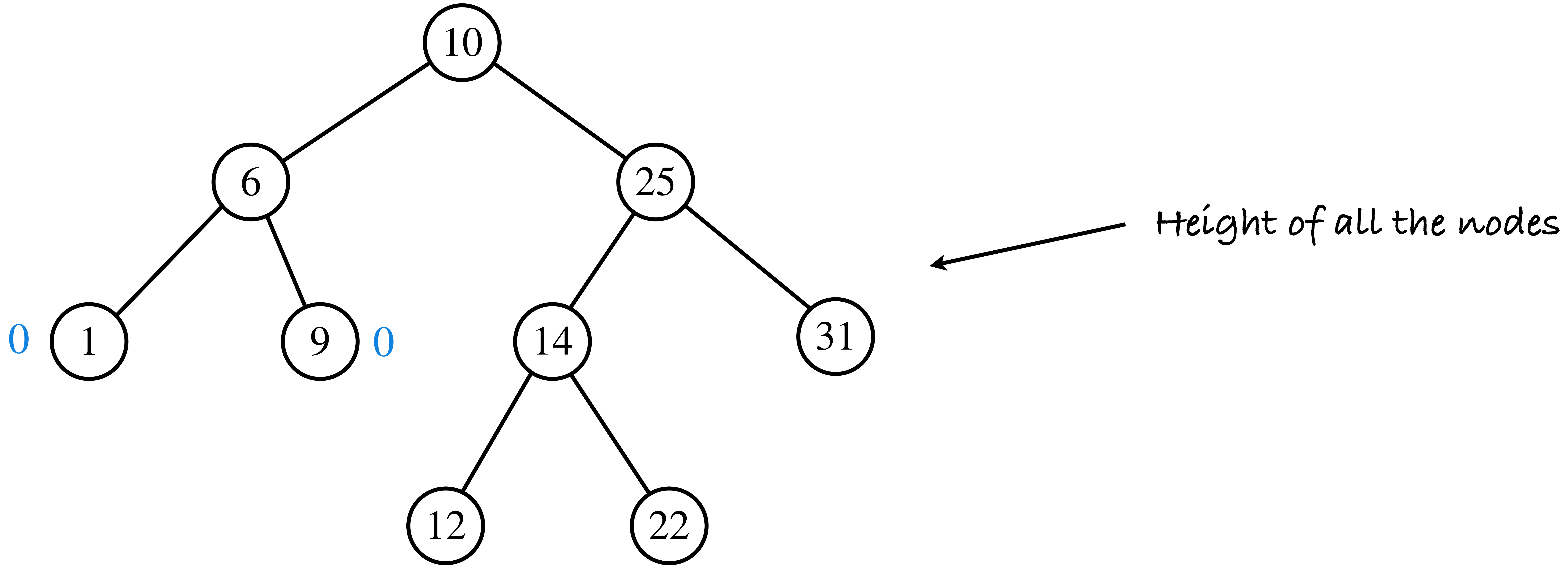
Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



Height of all the nodes

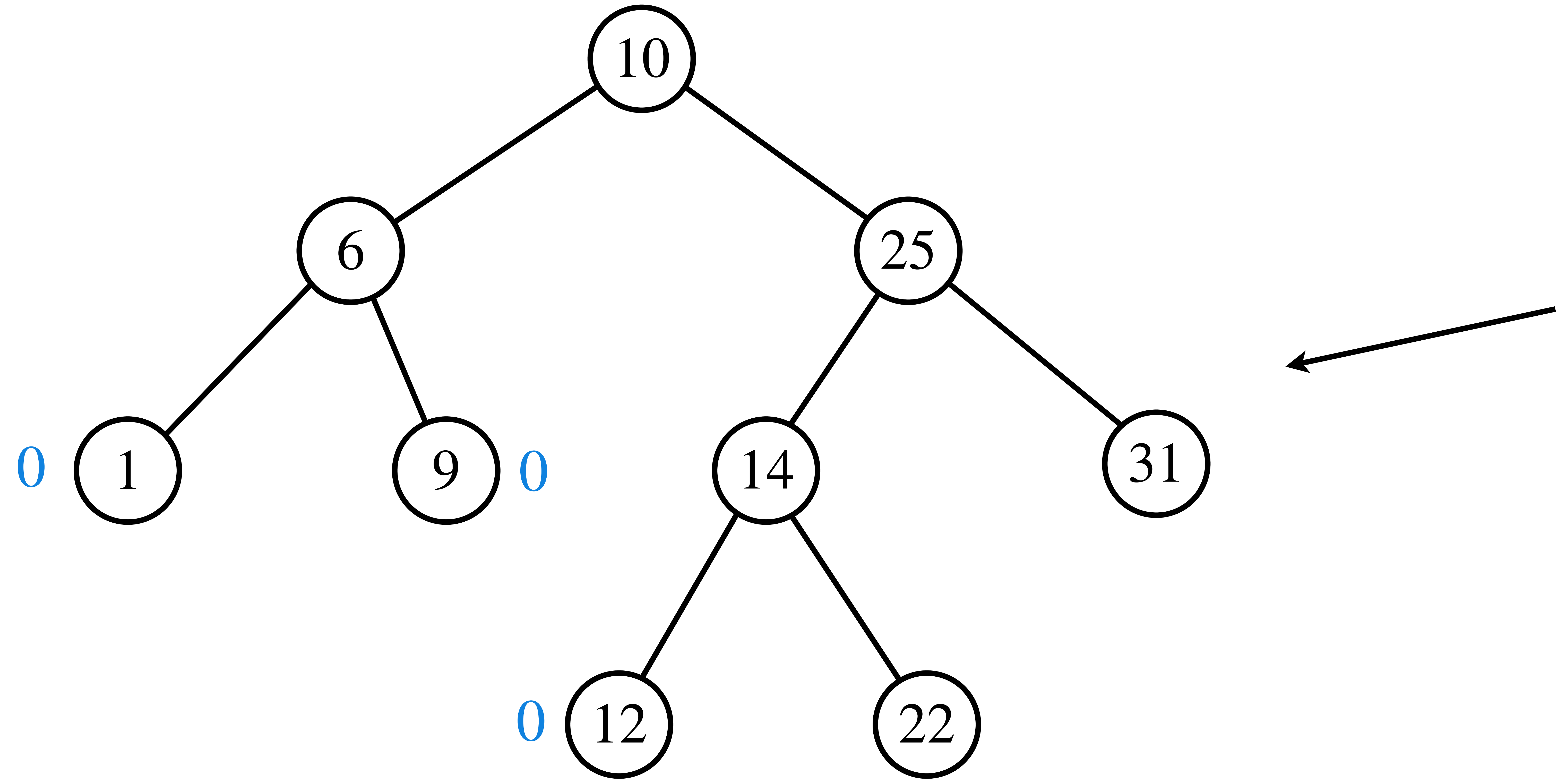
BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



BST: Basic Terminology

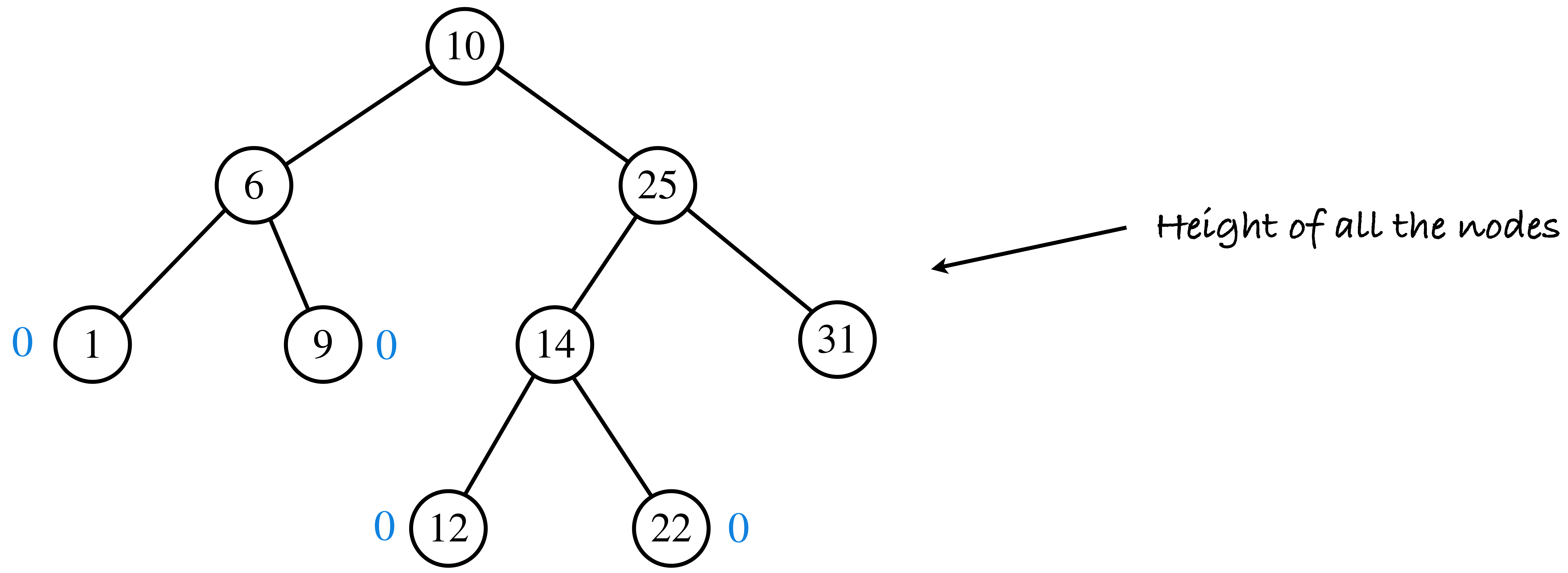
Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



Height of all the nodes

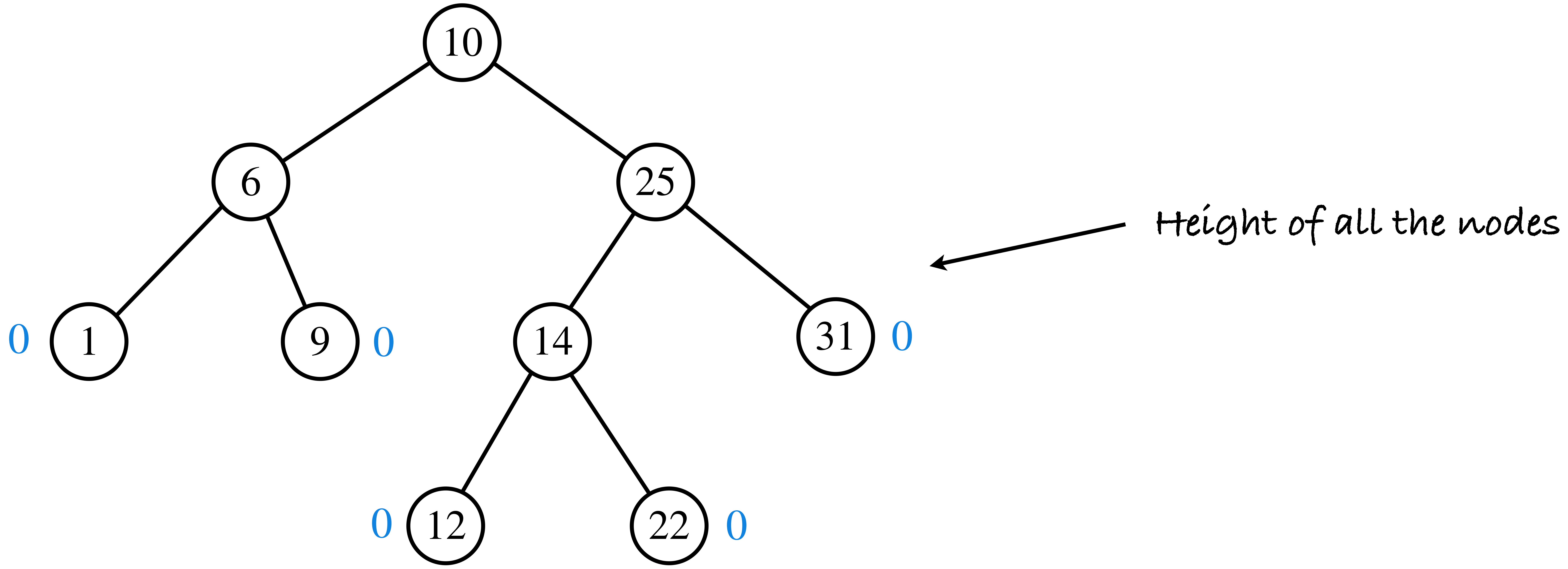
BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



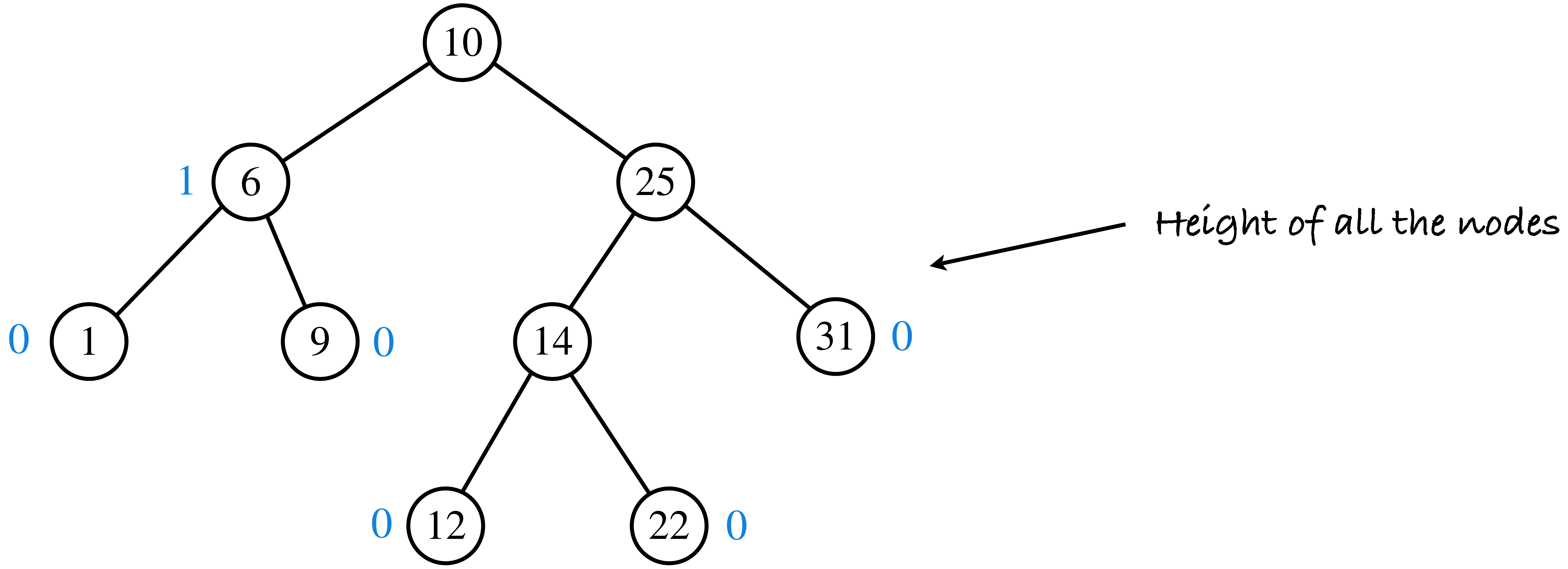
BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



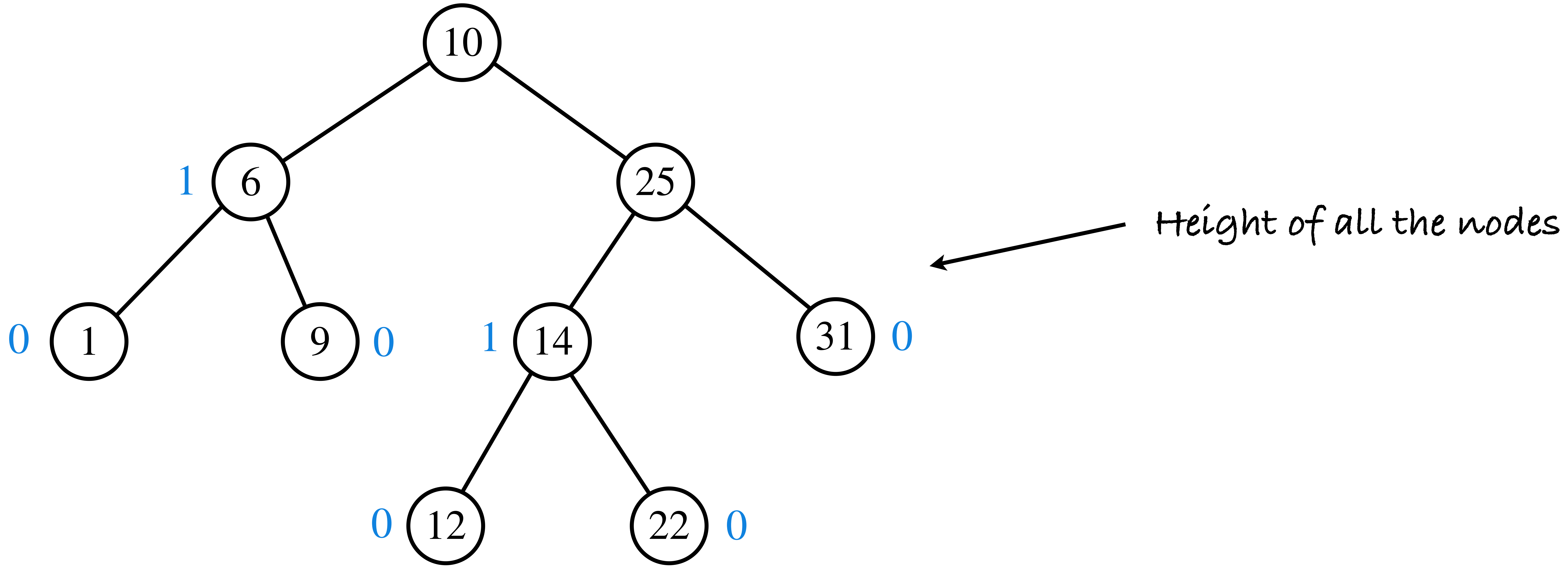
BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



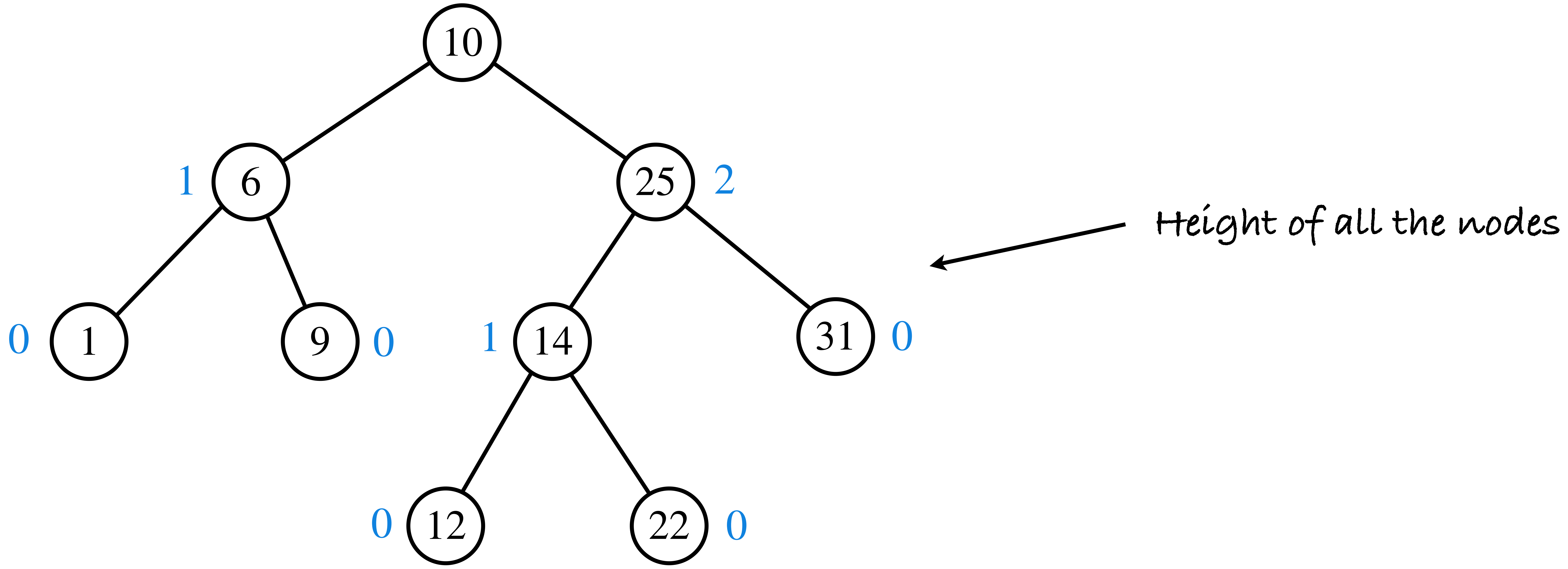
BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



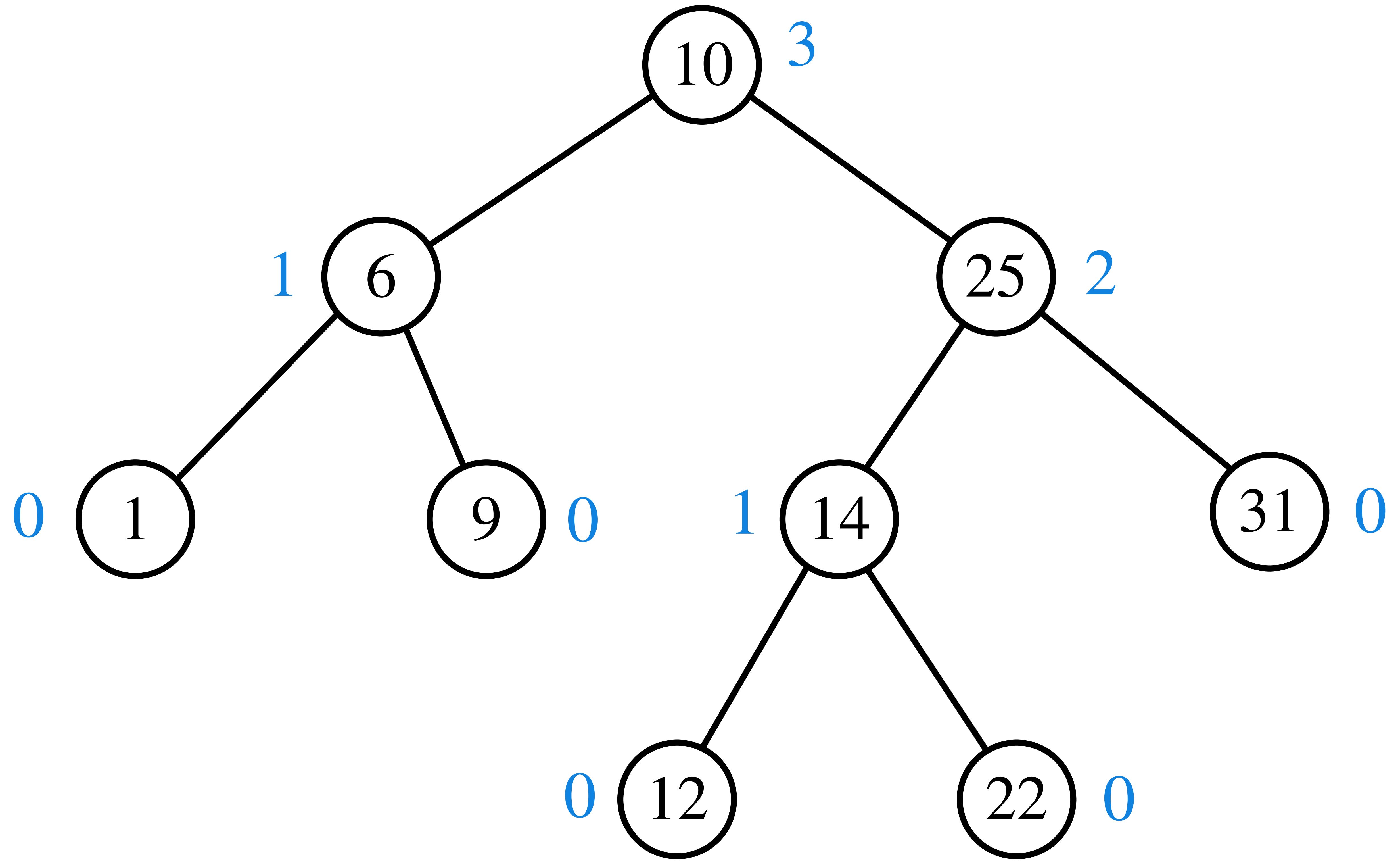
BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



BST: Basic Terminology

Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



Height of all the nodes

Inorder Tree Walk

Inorder Tree Walk

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

Inorder Tree Walk

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

```
Inorder-Tree-Walk($x$):
```

Inorder Tree Walk

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

```
Inorder-Tree-Walk($x$):
1.    if $x \neq$ NIL
```

Inorder Tree Walk

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

```
Inorder-Tree-Walk($x$):
1.  if $x \neq$ NIL
2.      Inorder-Tree-Walk($x\,.\,left$)
```

Inorder Tree Walk

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

```
Inorder-Tree-Walk($x$):
1.  if $x \neq$ NIL
2.      Inorder-Tree-Walk($x\,.\,left$)
3.      print $x\,.\,key$
```

Inorder Tree Walk

Calling **Inorder-Tree-Walk**($T.\mathit{root}$) will print the keys of a BST in sorted order:

```
Inorder-Tree-Walk($x$):
1.  if $x \neq$ NIL
2.      Inorder-Tree-Walk($x.\mathit{left}$)
3.      print $x.\mathit{key}$
4.      Inorder-Tree-Walk($x.\mathit{right}$)
```

Inorder Tree Walk

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

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

Proof of Correctness: Can be proven using induction on the number of nodes in the tree.

Inorder Tree Walk

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

```
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

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

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

Runtime: $O(n)$, where n is the number of nodes in the tree.

Inorder Tree Walk

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

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

Runtime: $O(n)$, where n is the number of nodes in the tree. Each node gets printed once.

Inorder Tree Walk

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

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

Runtime: $O(n)$, where n is the number of nodes in the tree. Each node gets printed once.

Can be formally proven by solving recurrence using substitution method.

Querying a BST: Search

Querying a BST: Search

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

Querying a BST: Search

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

Illustration:

Querying a BST: Search

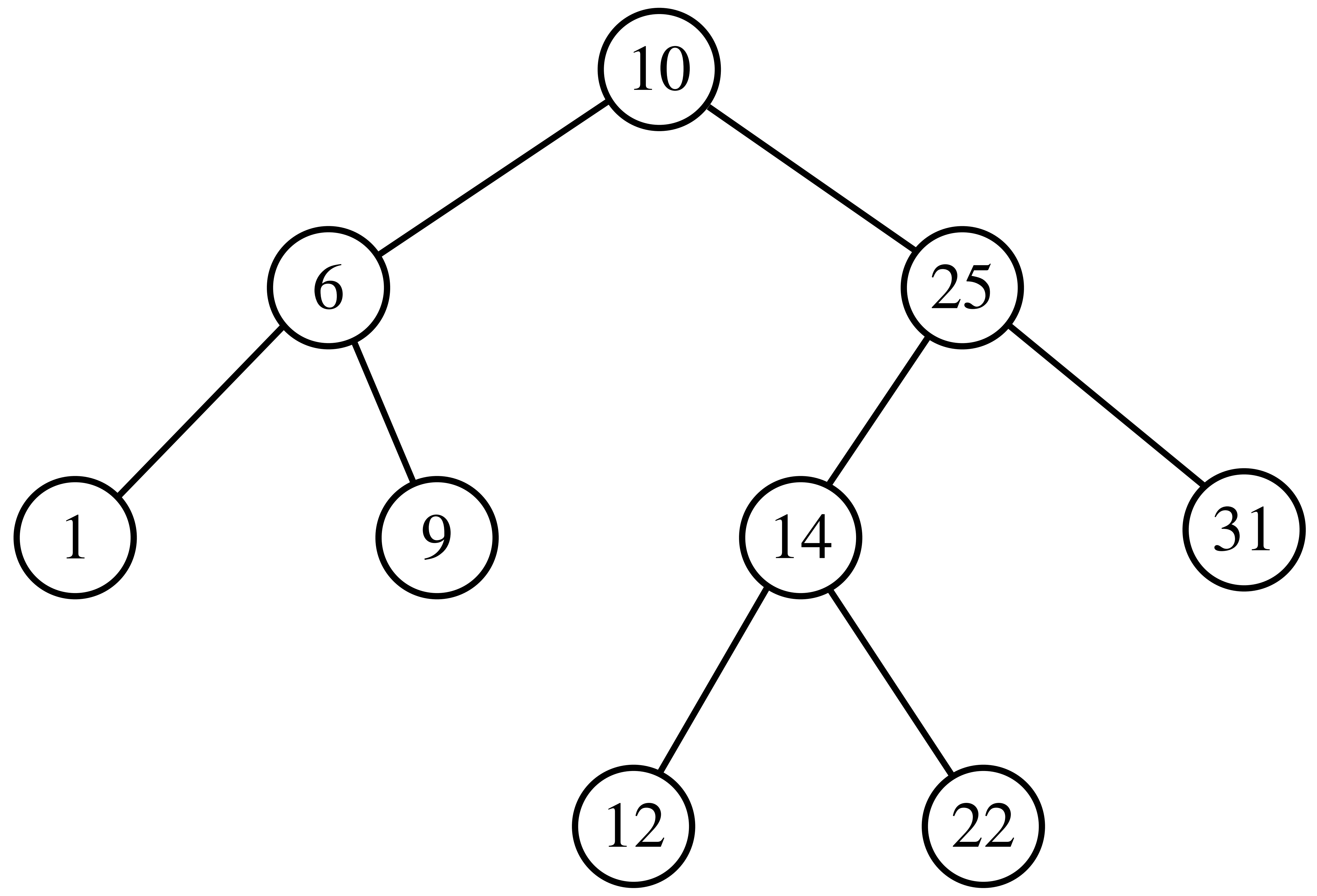
Goal: Given a pointer to the root of a BST, search for an element with the key k in it.

Illustration: Searching for 12 in below BST.

Querying a BST: Search

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

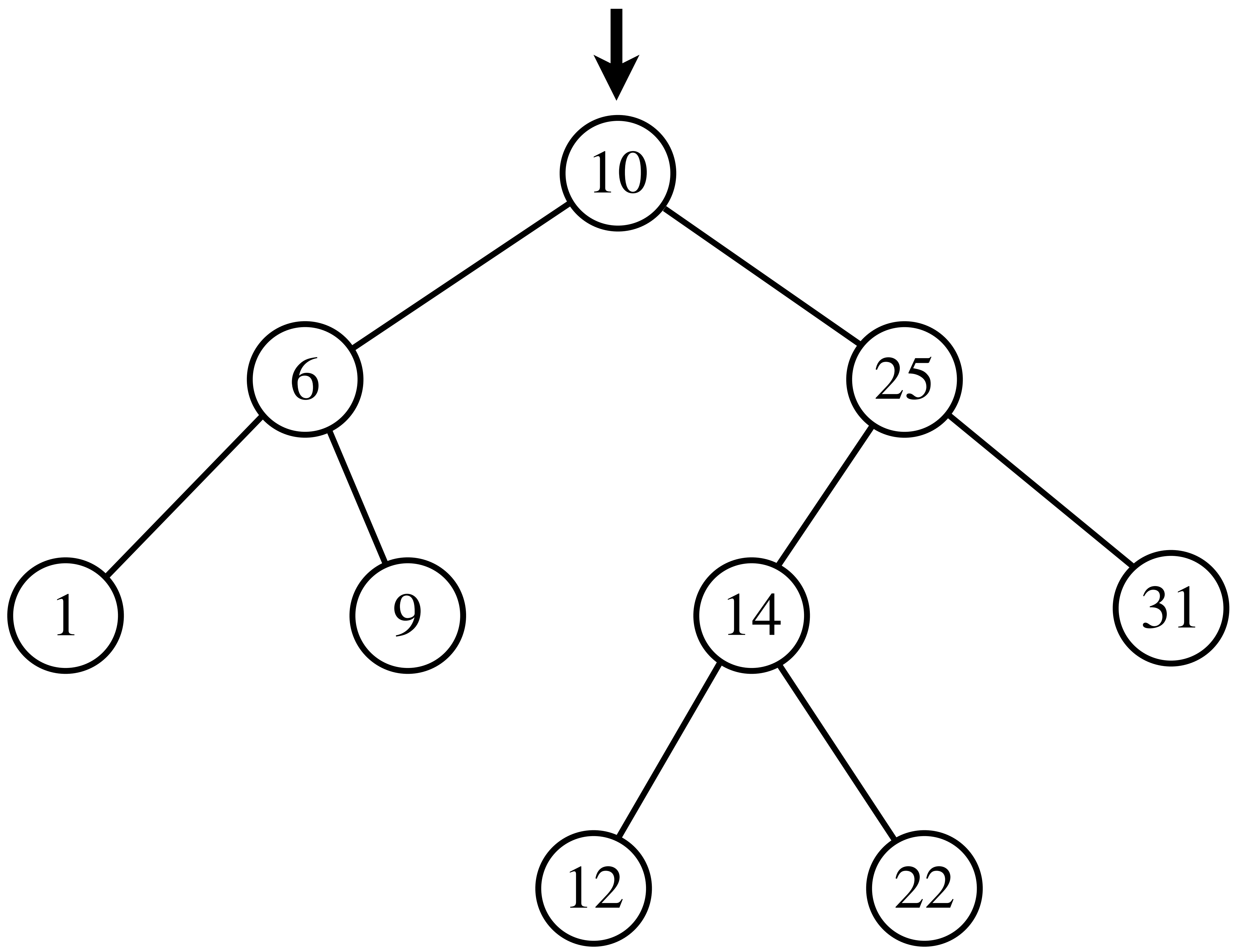
Illustration: Searching for 12 in below BST.



Querying a BST: Search

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

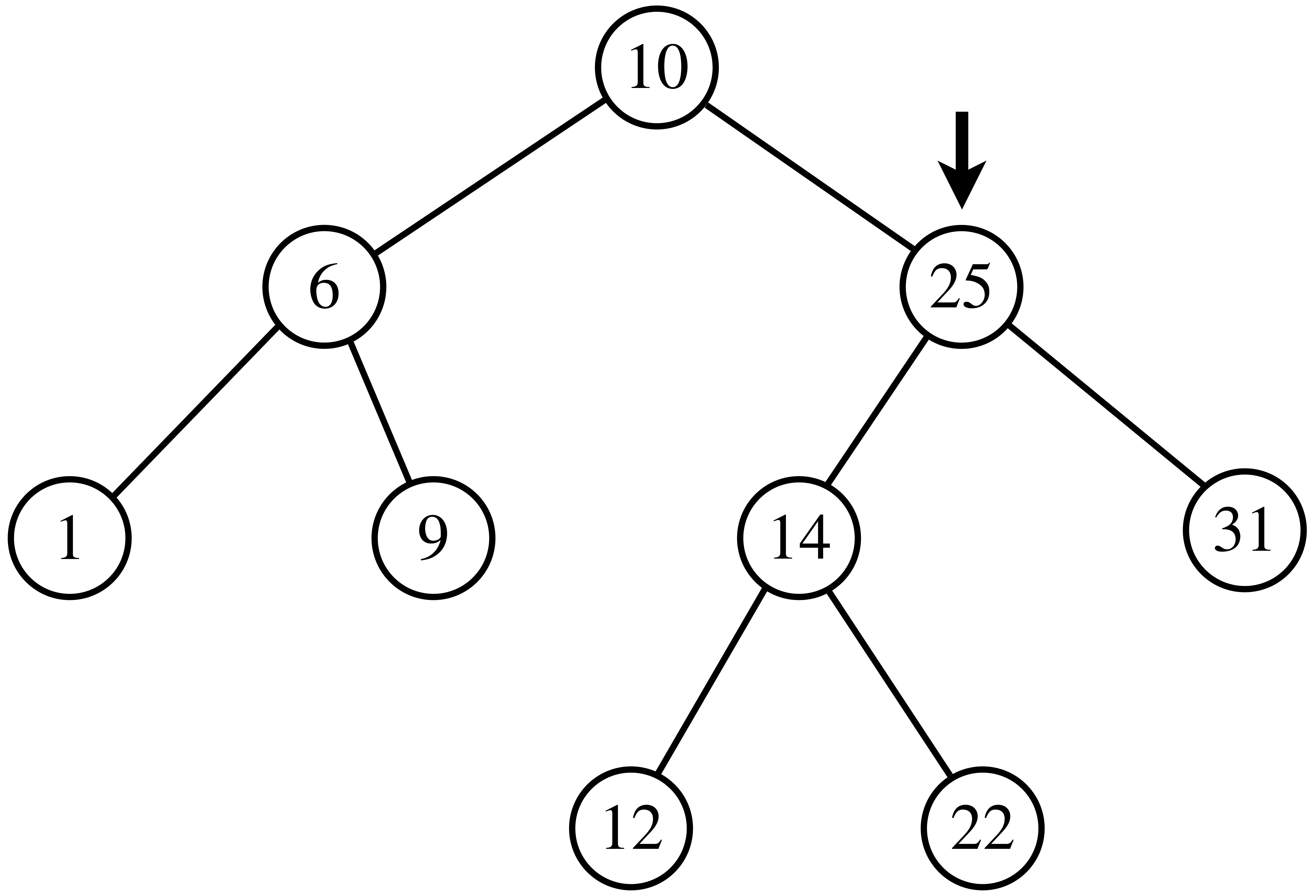
Illustration: Searching for 12 in below BST.



Querying a BST: Search

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

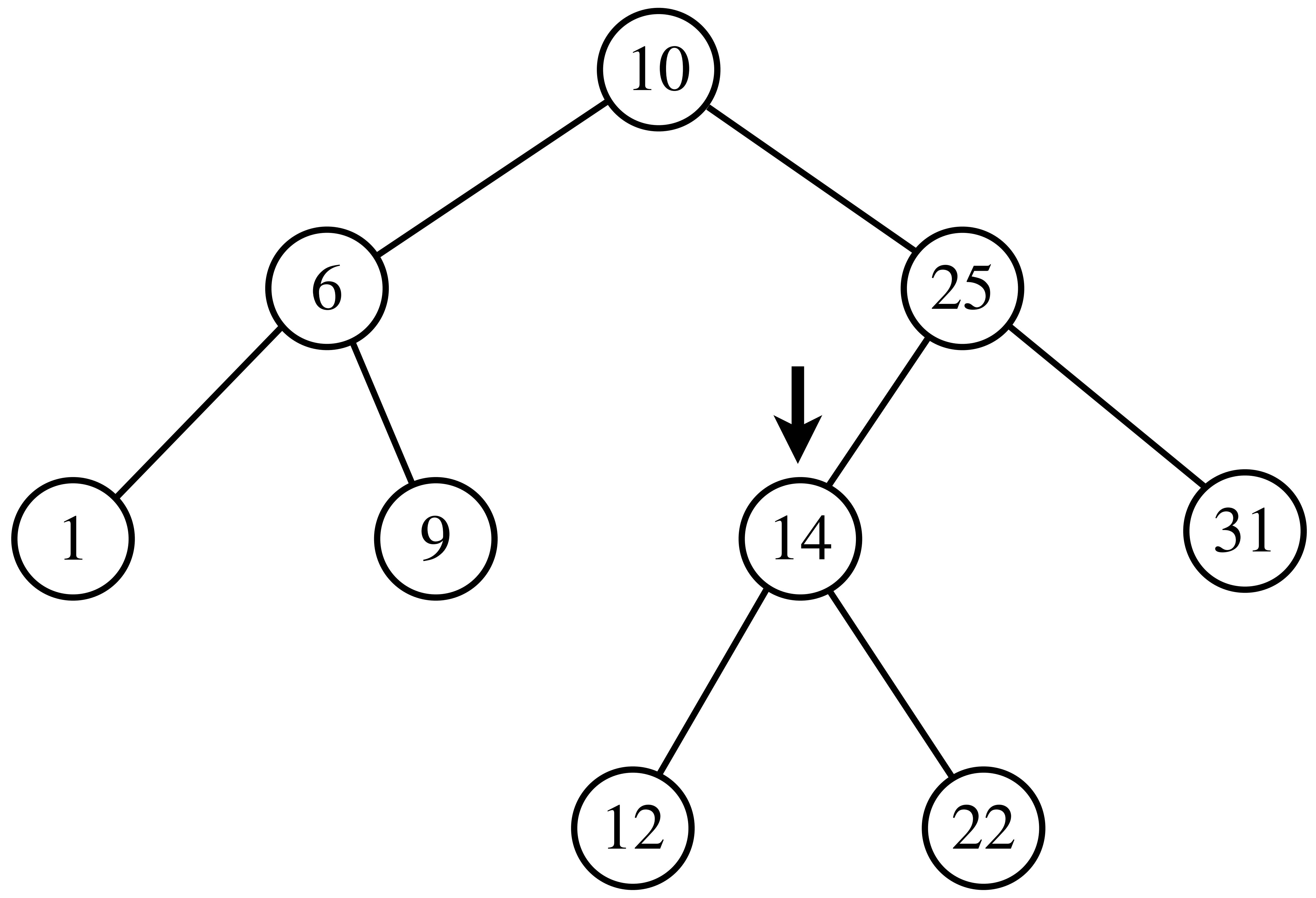
Illustration: Searching for 12 in below BST.



Querying a BST: Search

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

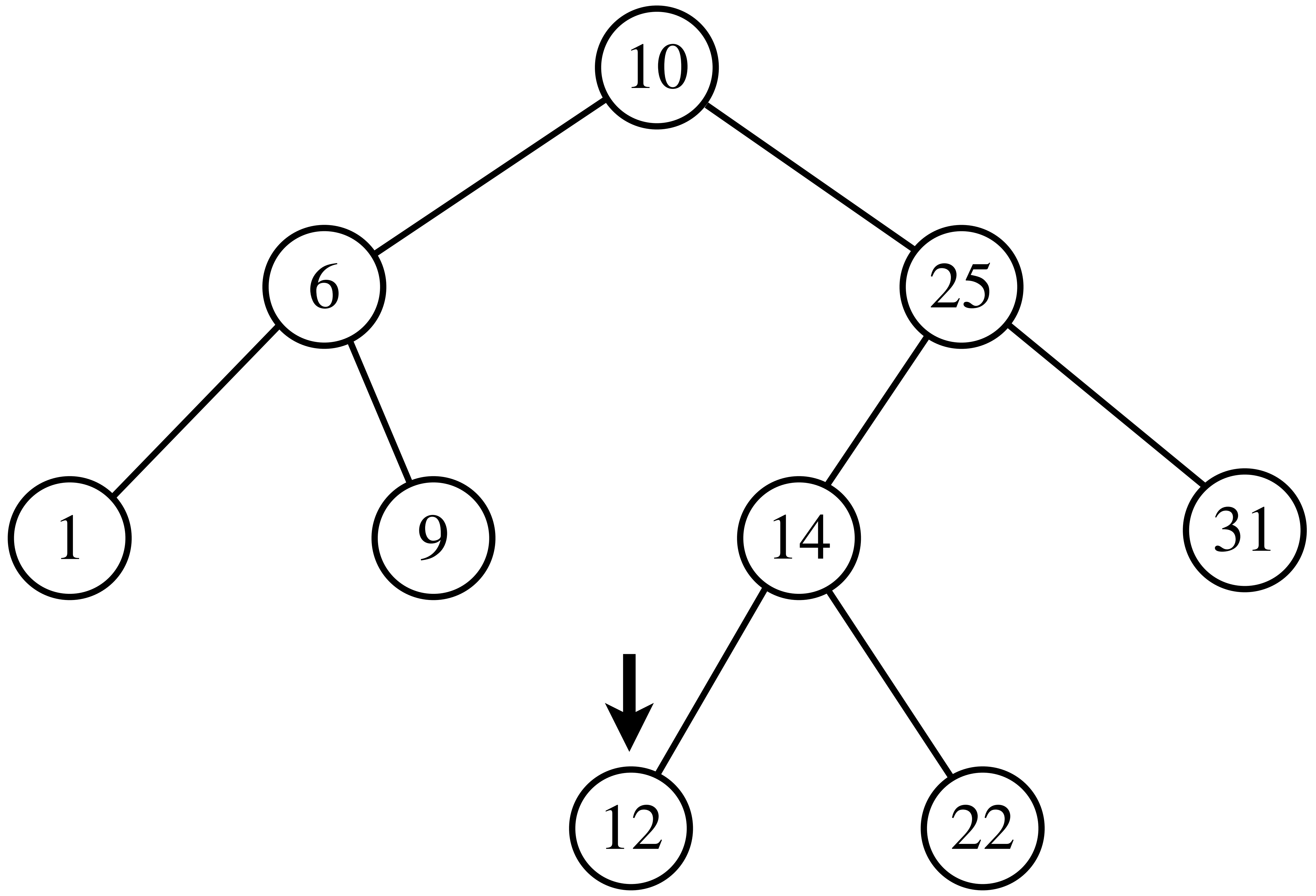
Illustration: Searching for 12 in below BST.



Querying a BST: Search

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

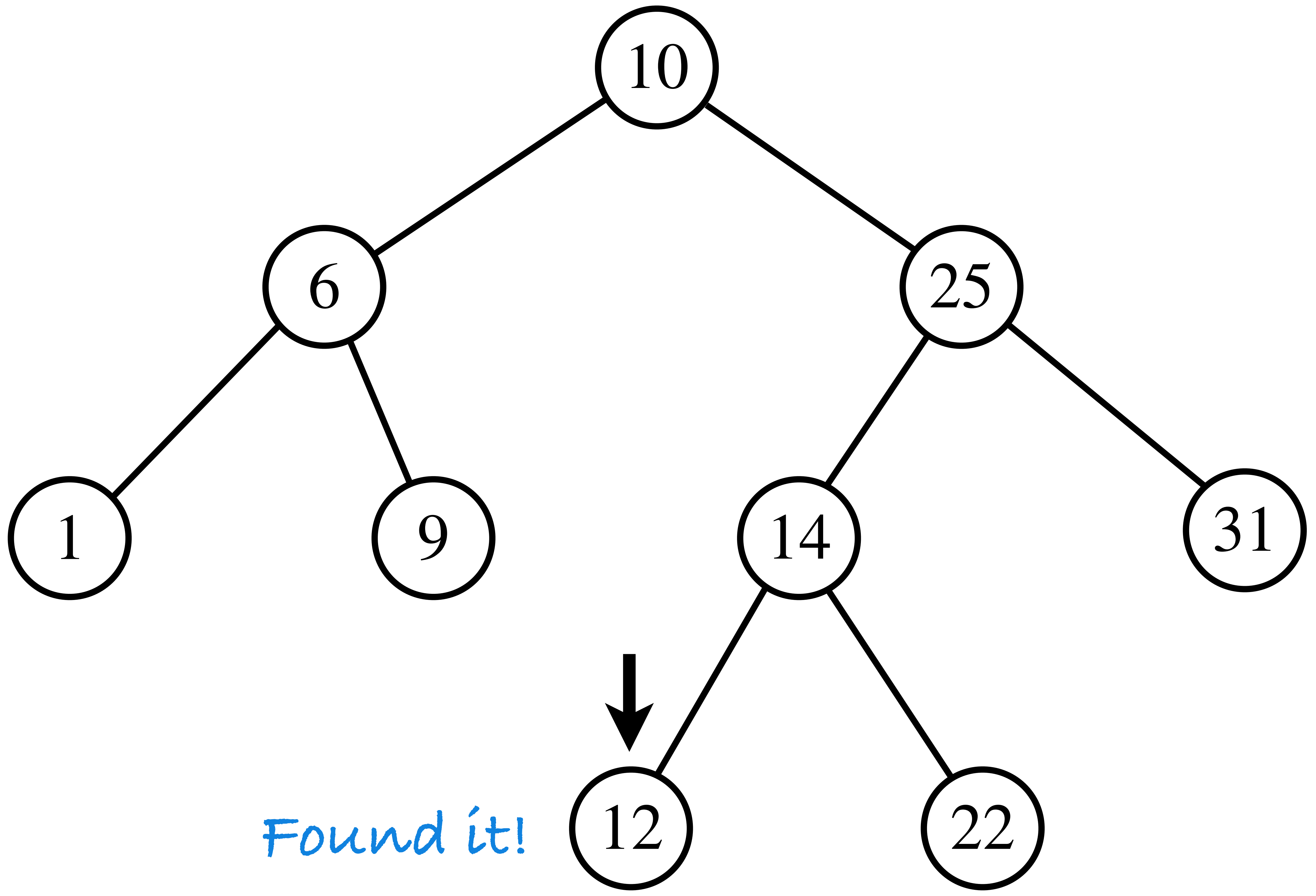
Illustration: Searching for 12 in below BST.



Querying a BST: Search

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

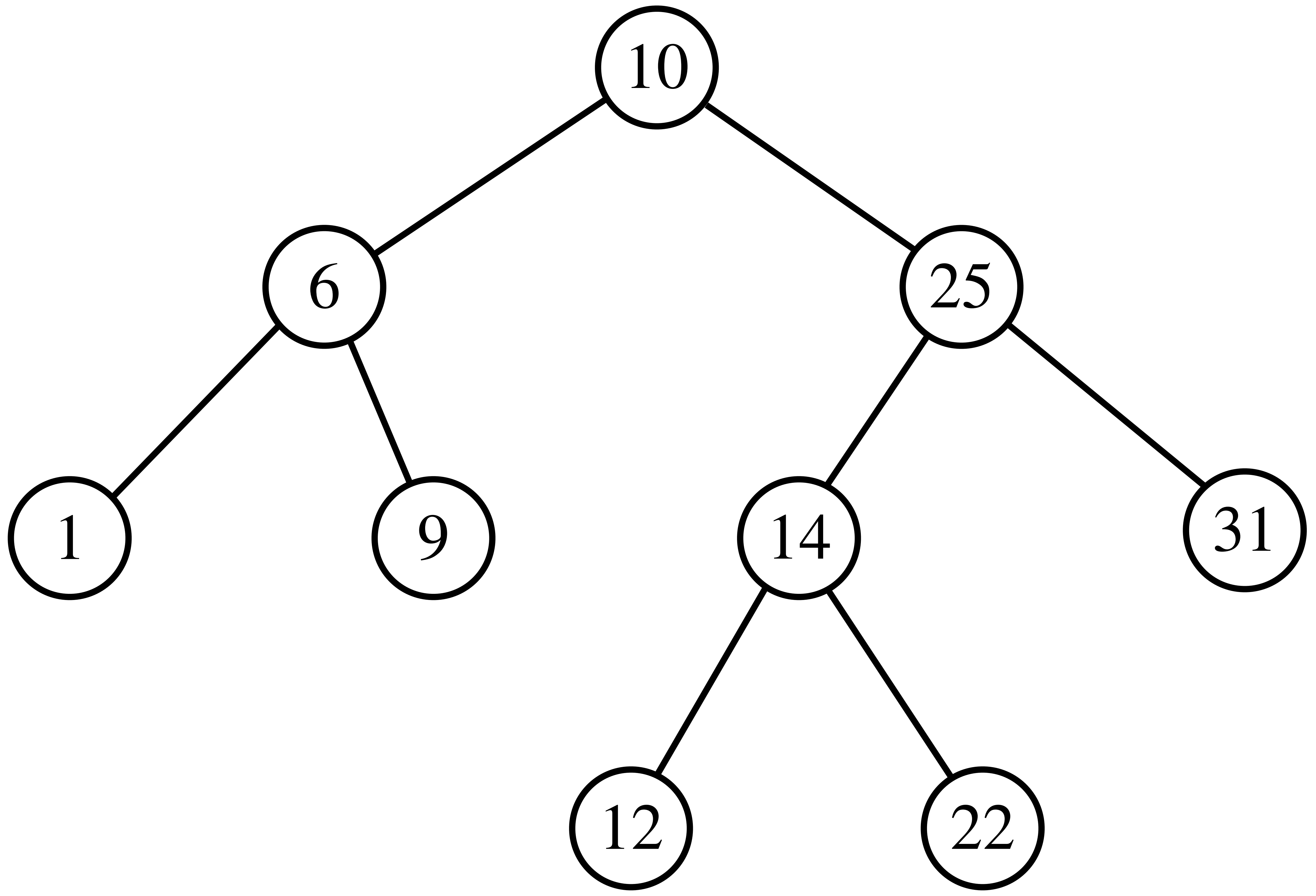
Illustration: Searching for 12 in below BST.



Querying a BST: Search

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

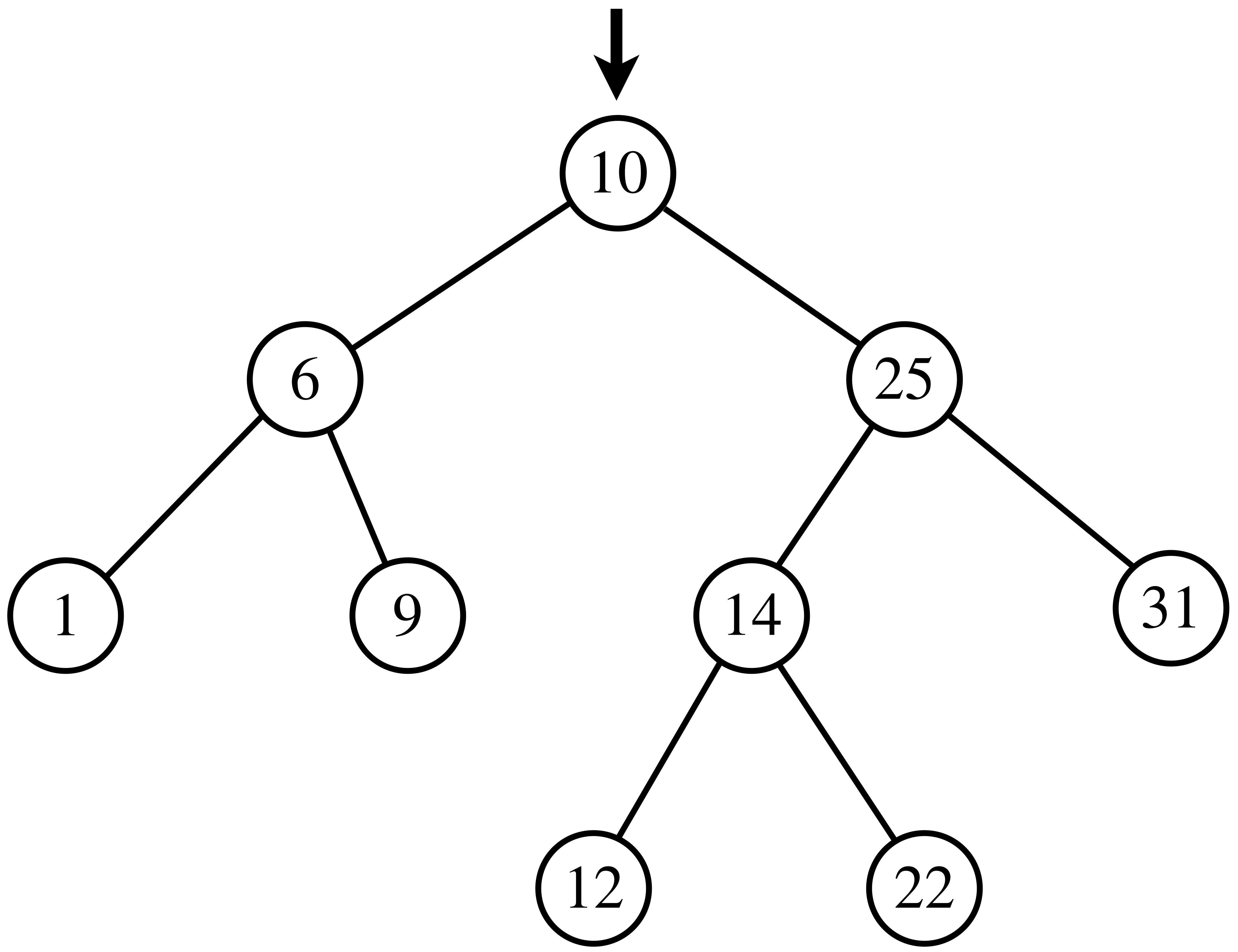
Illustration: Searching for 5 in below BST.



Querying a BST: Search

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

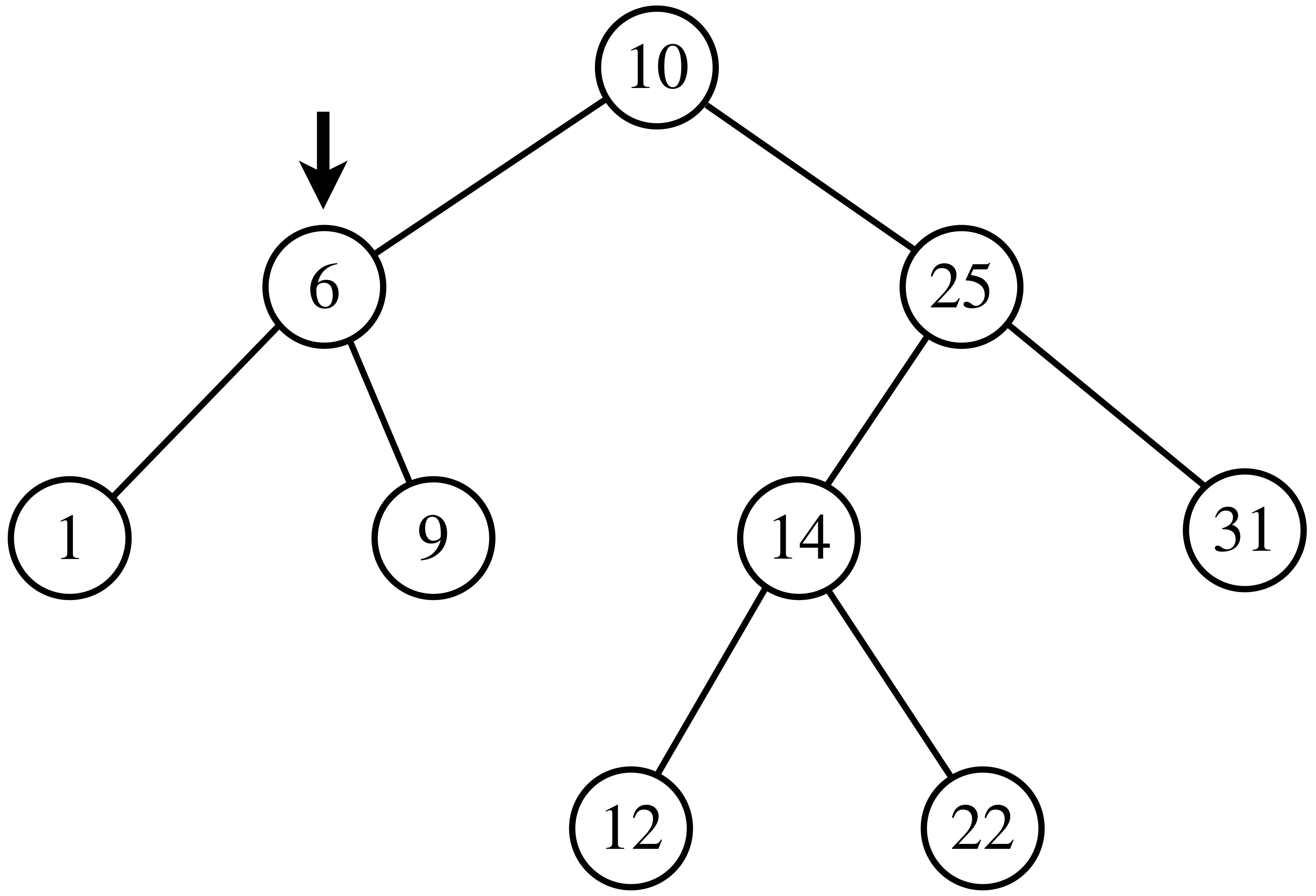
Illustration: Searching for 5 in below BST.



Querying a BST: Search

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

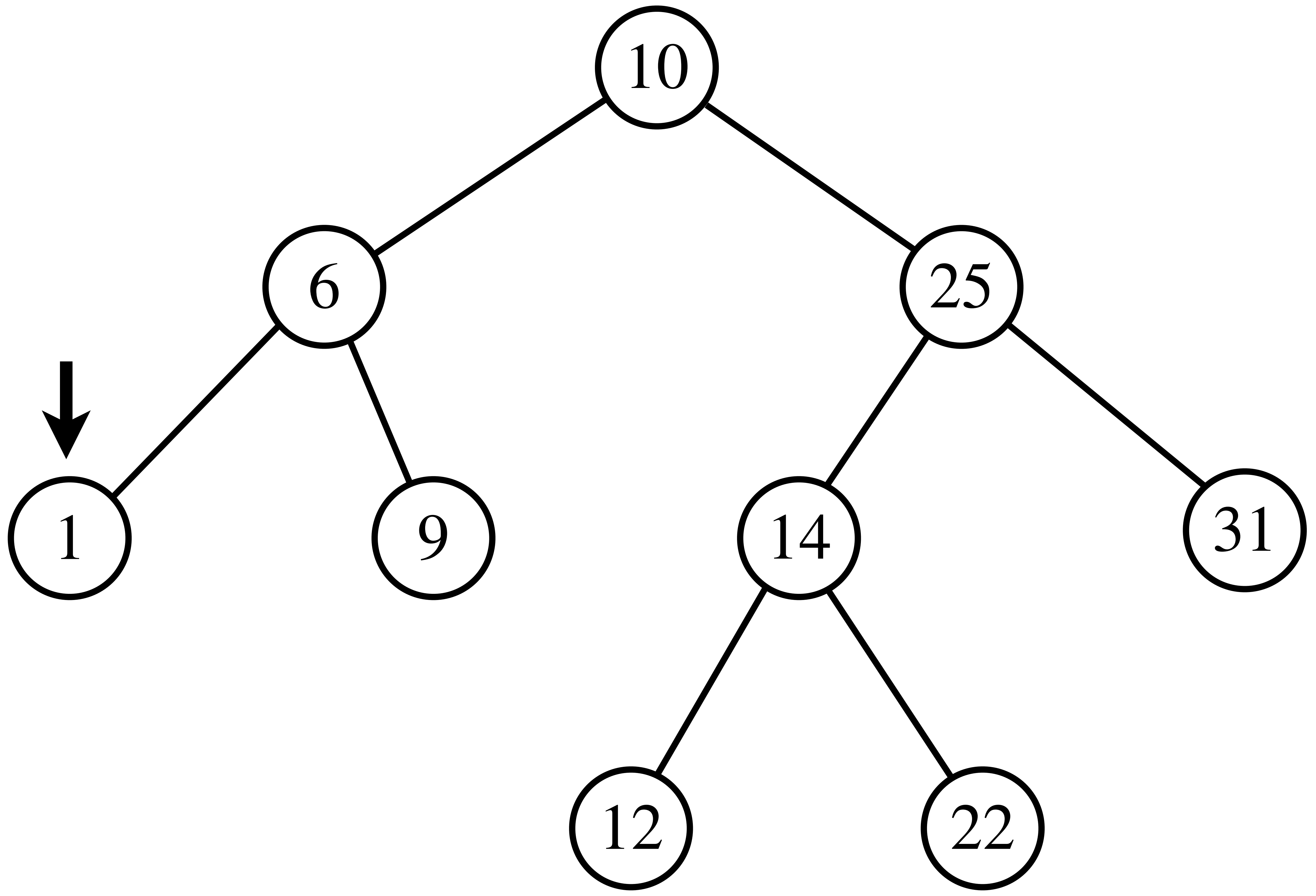
Illustration: Searching for 5 in below BST.



Querying a BST: Search

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

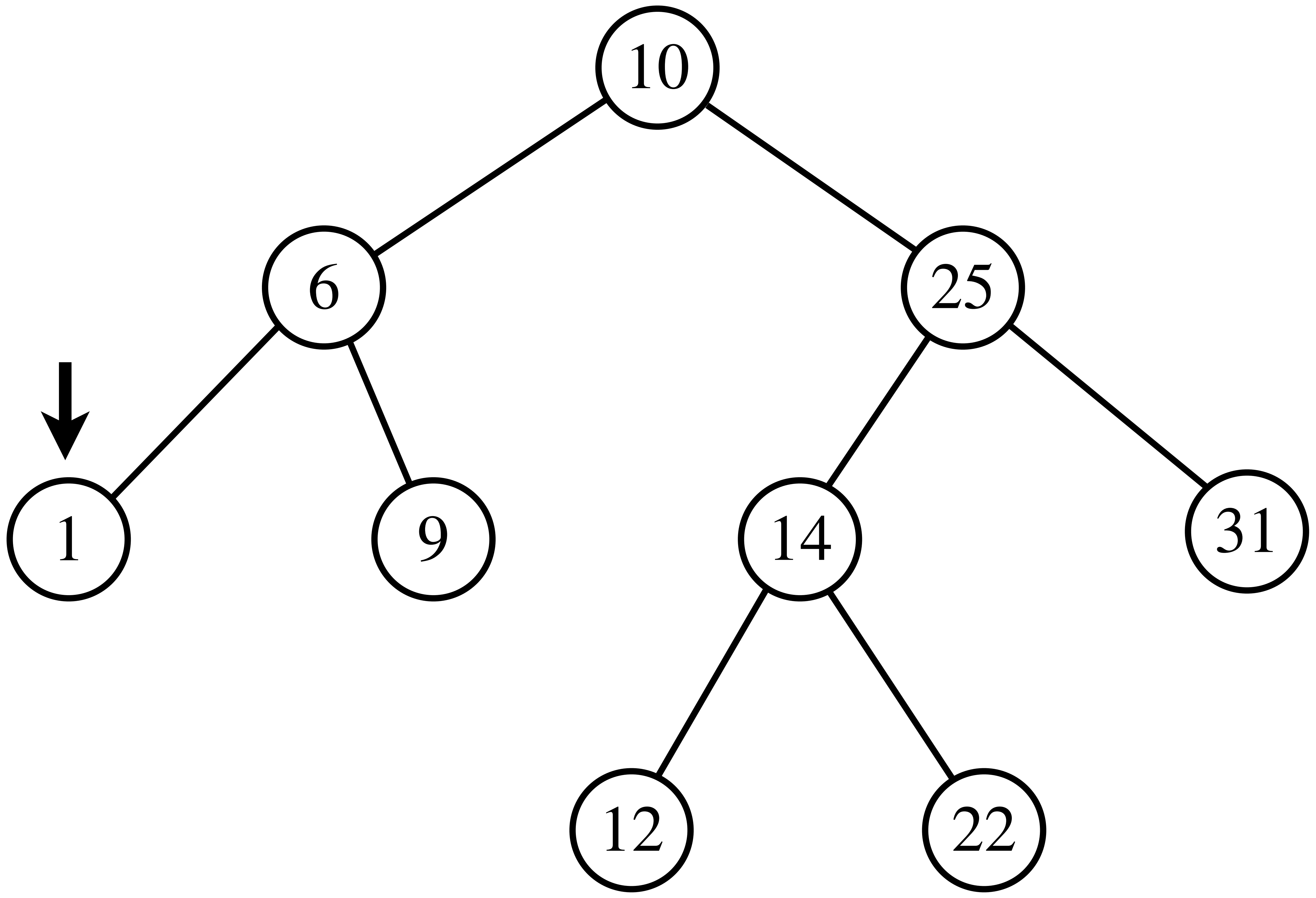
Illustration: Searching for 5 in below BST.



Querying a BST: Search

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 below BST.



5 is not present.

Querying a BST: Search

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

Querying a BST: Search

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.

Querying a BST: Search

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):
```

Querying a BST: Search

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 $k \neq x\,.\,key$
```

Querying a BST: Search

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 $k < x.key$
```

Querying a BST: Search

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.          $x = x.left$
```

Querying a BST: Search

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
```

Querying a BST: Search

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
5.          $x = x.right$
```

Querying a BST: Search

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
5.          $x = x\,.\,right$
6.  return $x$
```

Querying a BST: Search

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
5.          $x = x.right$
6.  return $x$
```

$k = 8$

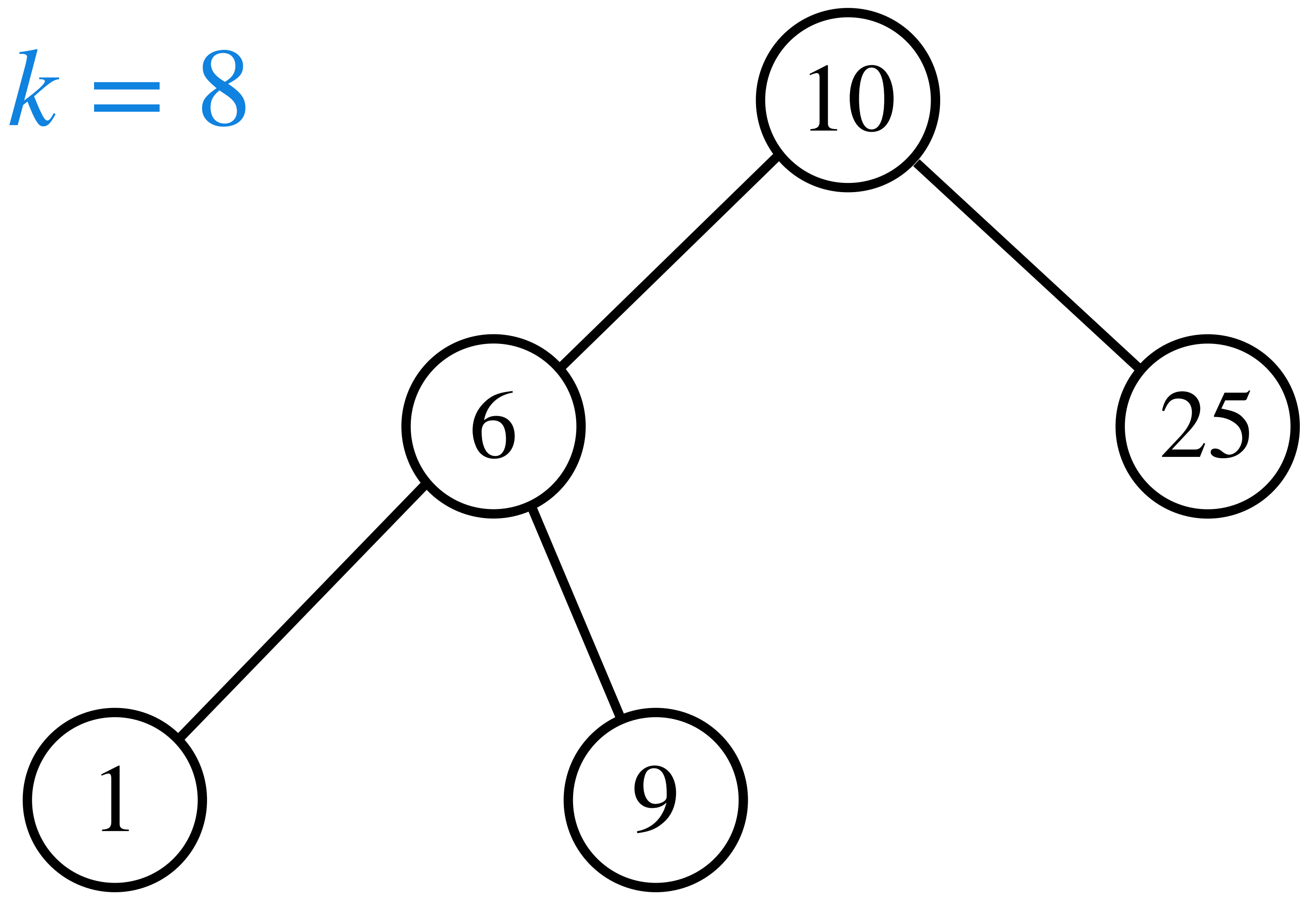
Querying a BST: Search

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
5.          $x = x.right$
6.  return $x$
```

$k = 8$

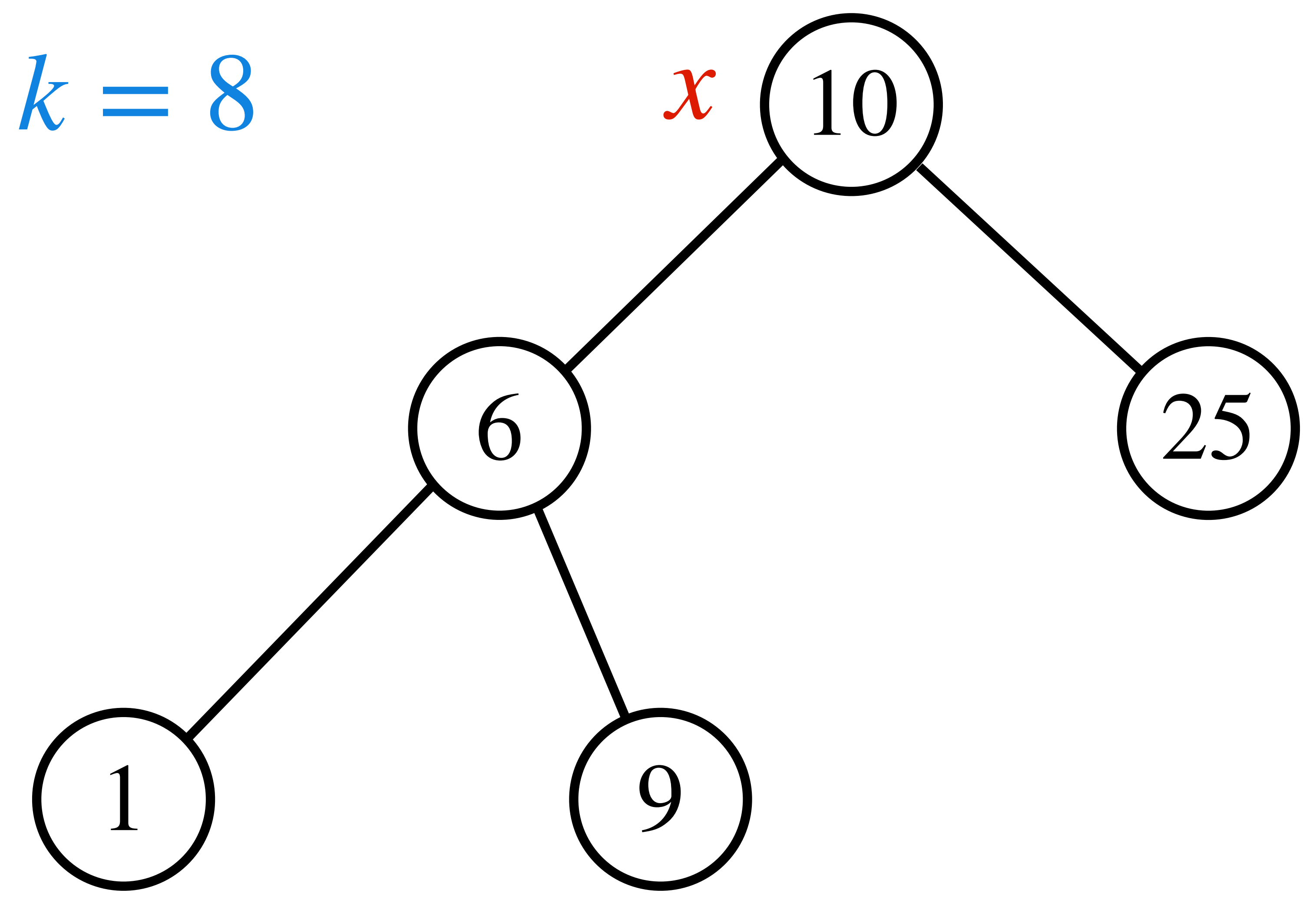


Querying a BST: Search

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
5.          $x = x.right$
6.  return $x$
```

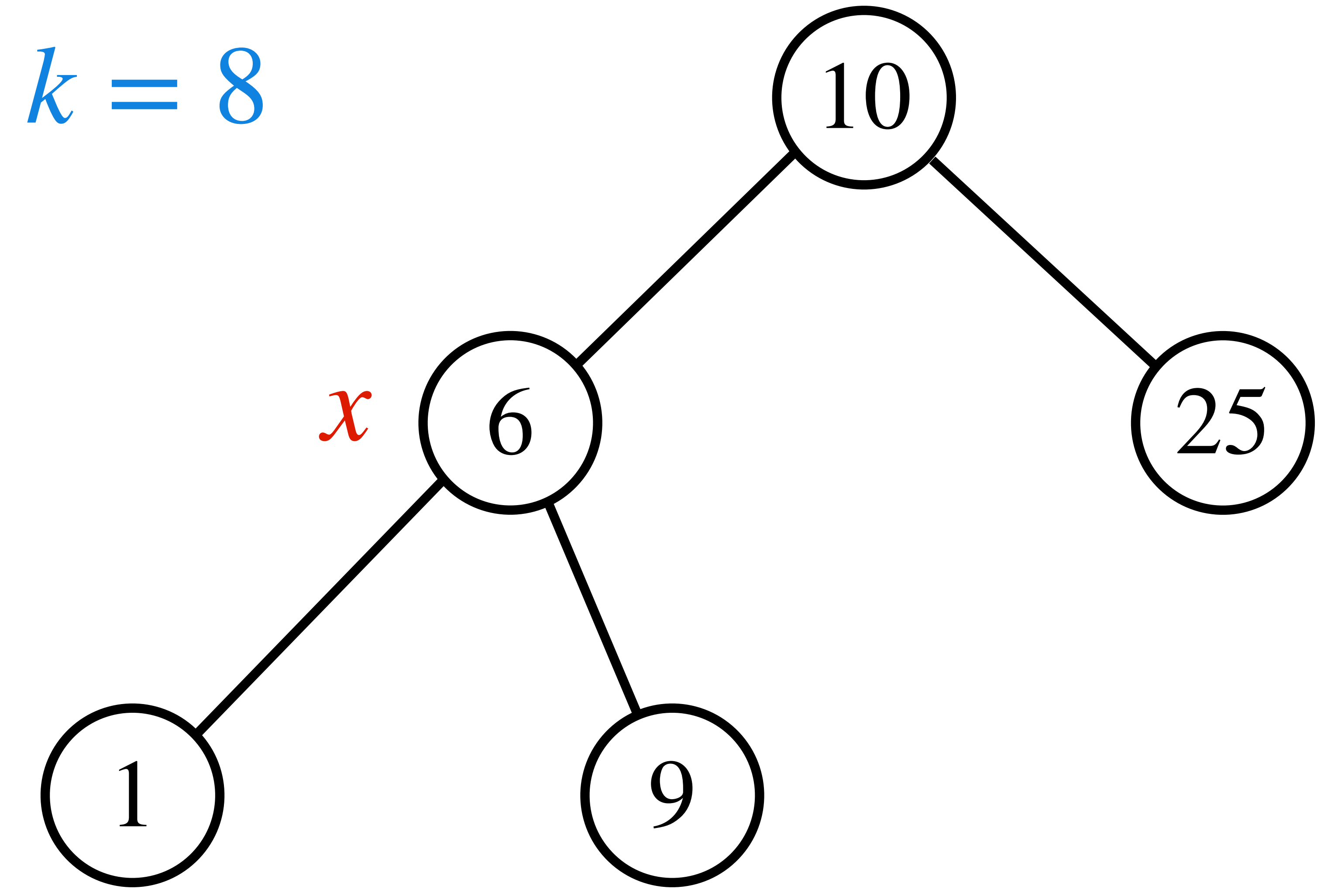


Querying a BST: Search

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
5.          $x = x.right$
6.  return $x$
```

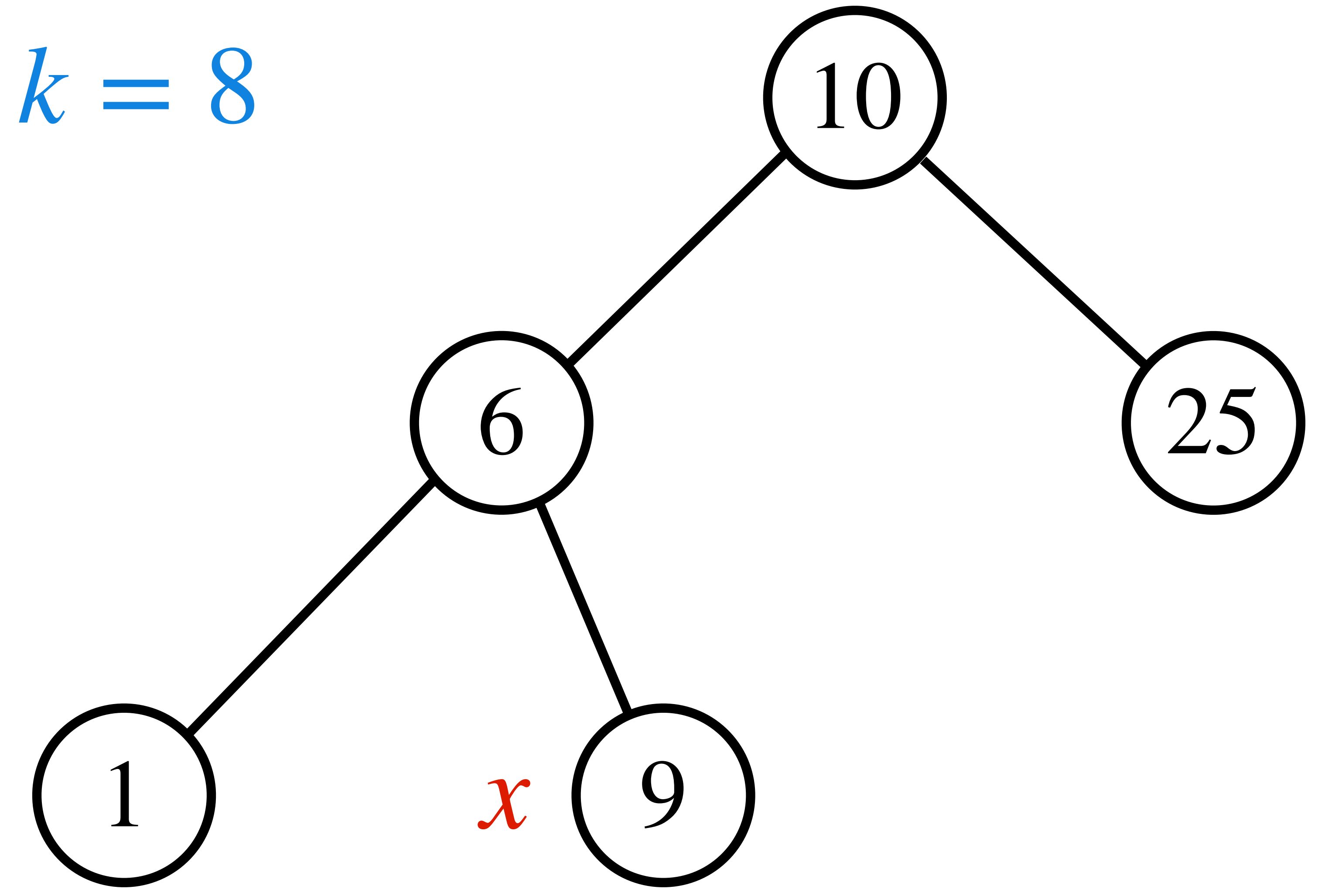


Querying a BST: Search

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
5.          $x = x\,.\,right$
6.  return $x$
```

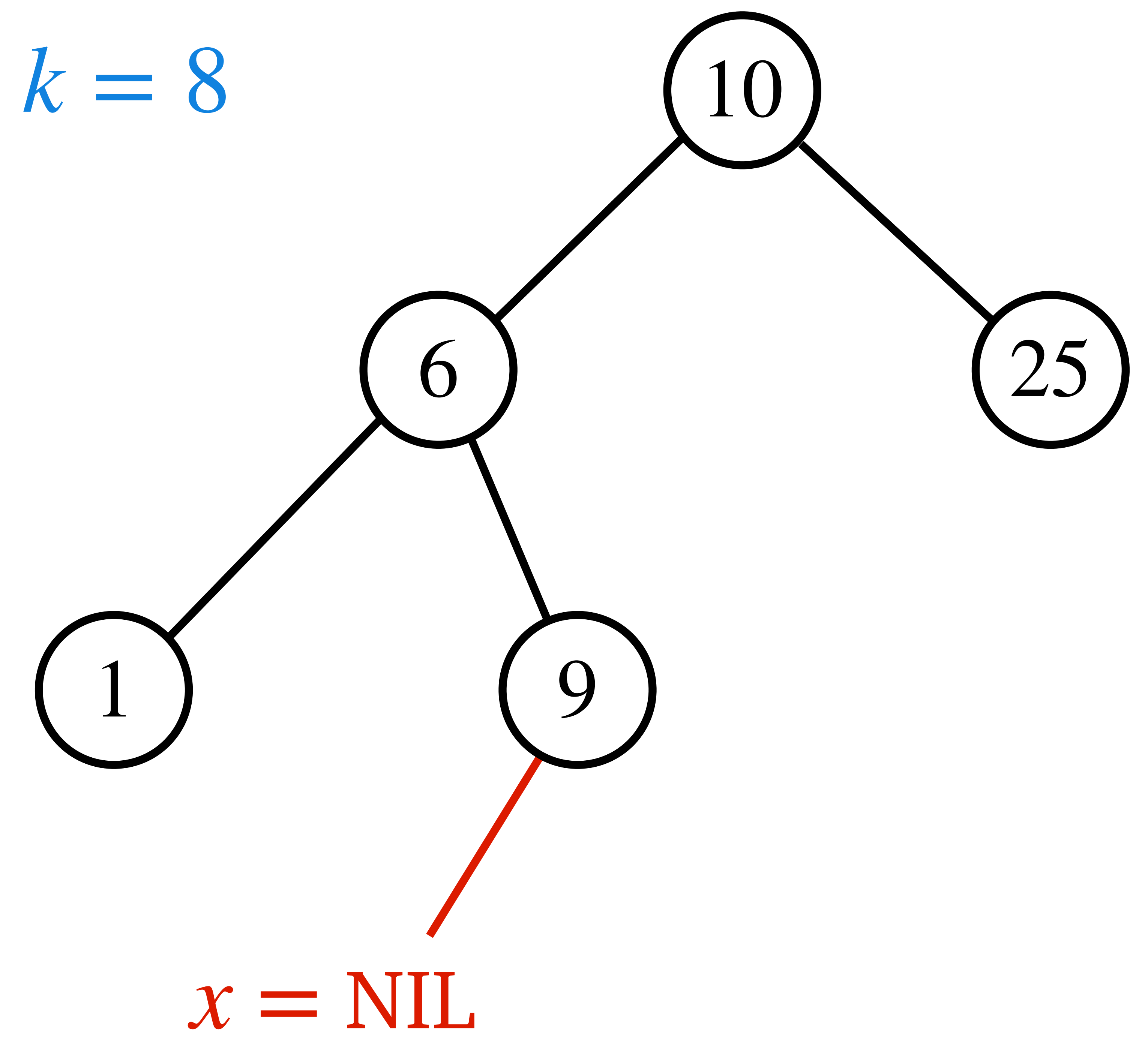


Querying a BST: Search

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
5.          $x = x.right$
6.  return $x$
```



Querying a BST: Search

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
5.          $x = x.right$
6.  return $x$
```

Querying a BST: Search

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
5.          $x = x\,.\,right$
6.  return $x$
```

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

Querying a BST: Search

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
5.          $x = x.right$
6.  return $x$
```

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

Querying a BST: Minimum

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Illustration:

Querying a BST: Minimum

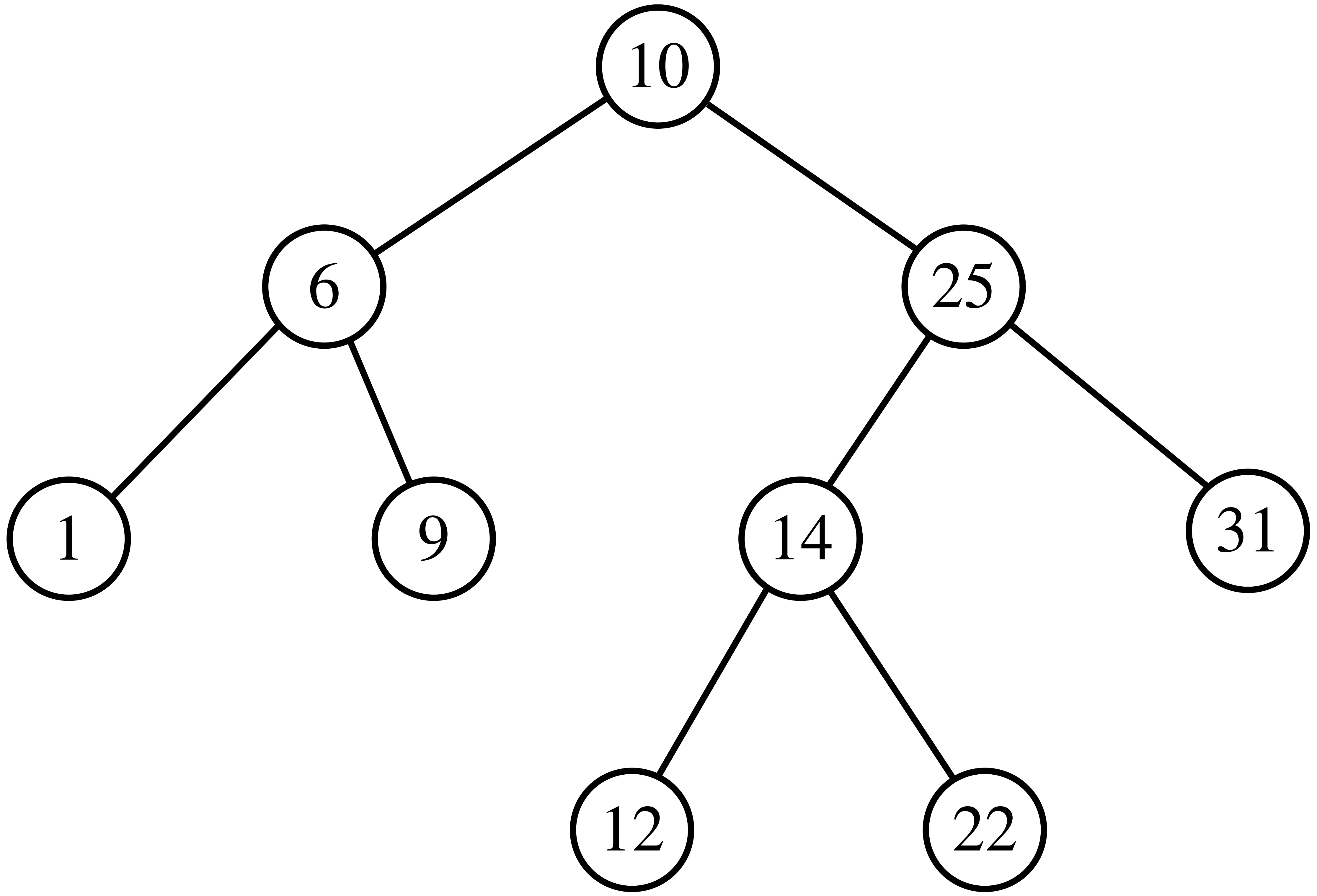
Goal: Given a pointer to the root of a BST to find the element with minimum key.

Illustration: Find the minimum key in the below BST.

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

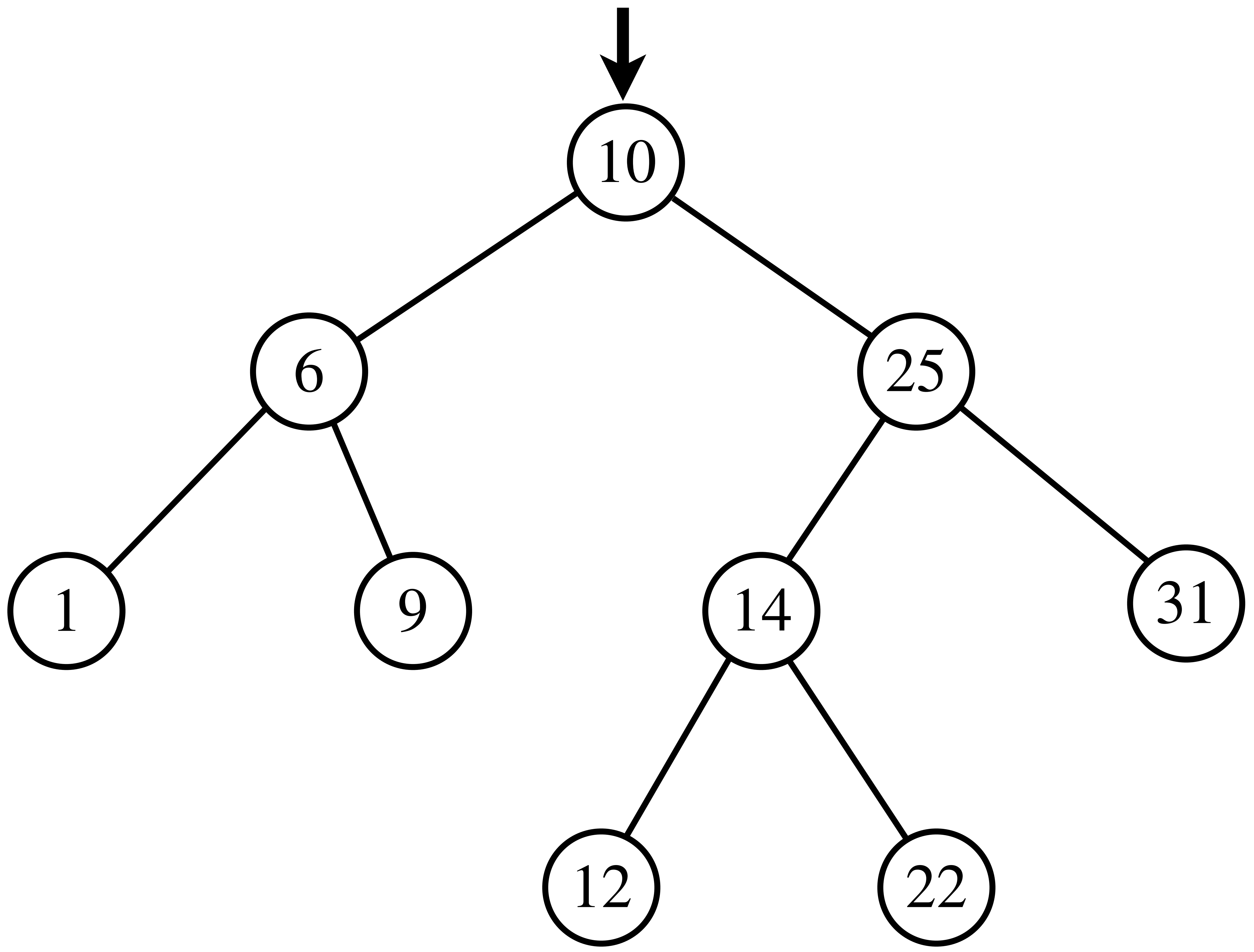
Illustration: Find the minimum key in the below BST.



Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

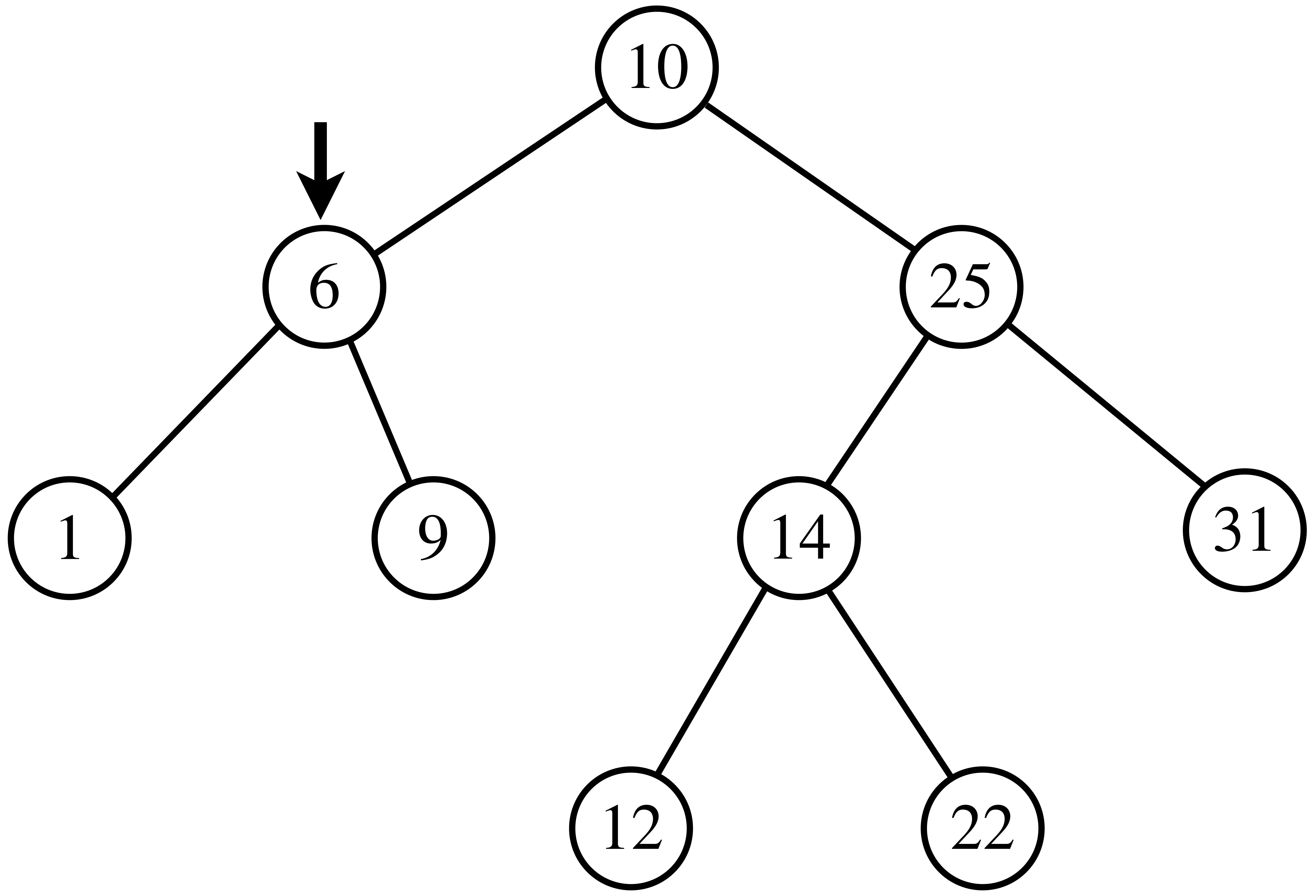
Illustration: Find the minimum key in the below BST.



Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

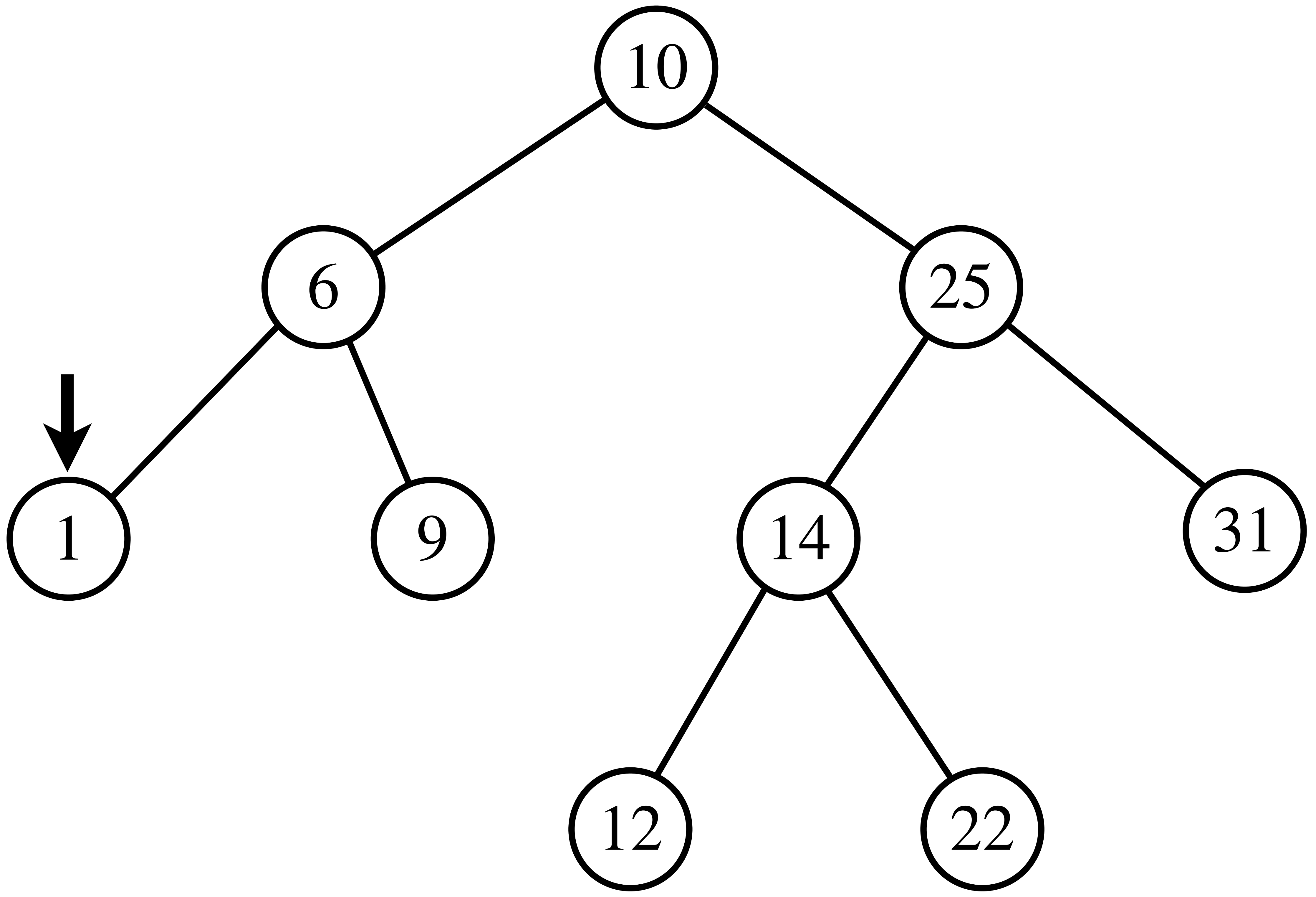
Illustration: Find the minimum key in the below BST.



Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

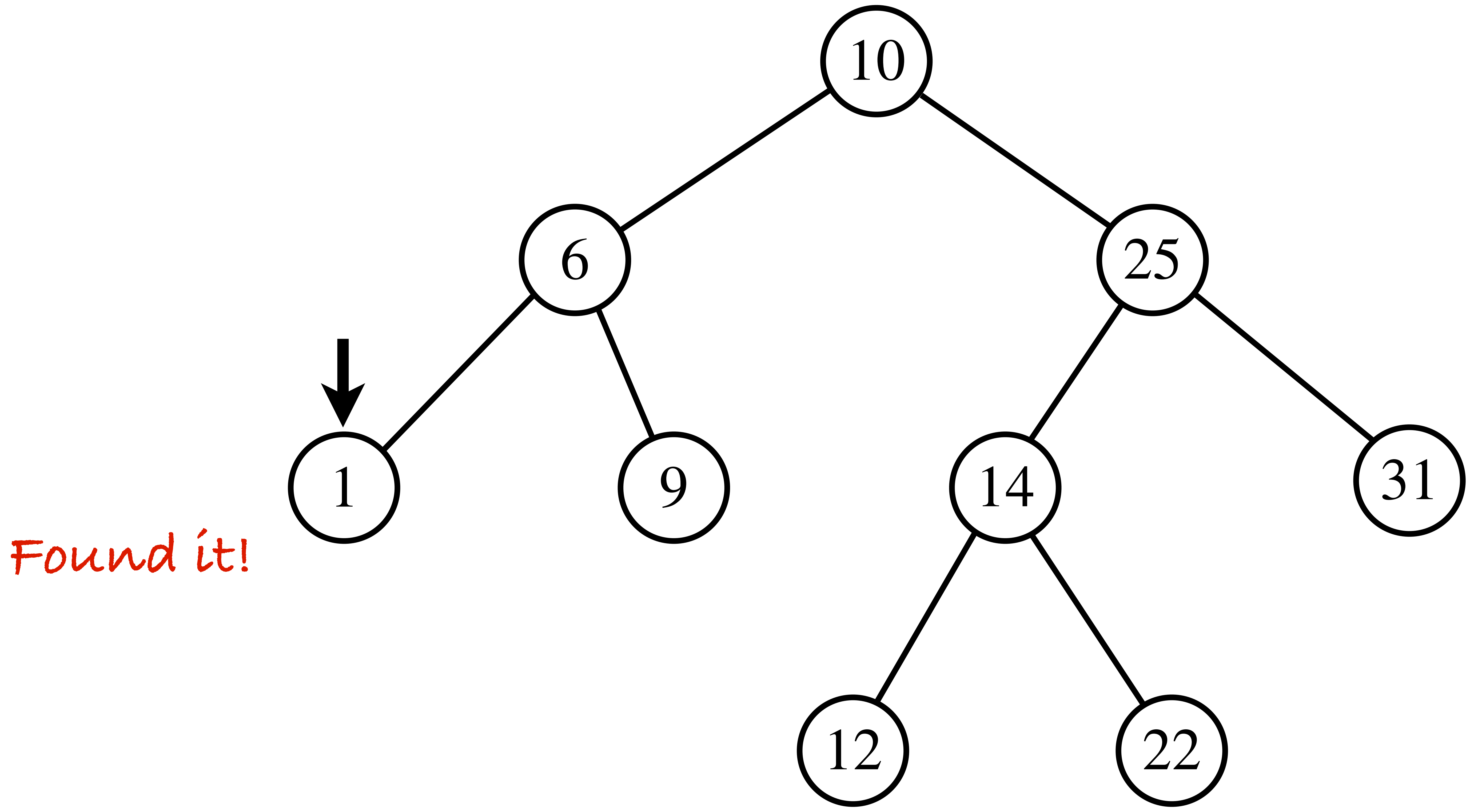
Illustration: Find the minimum key in the below BST.



Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Illustration: Find the minimum key in the below BST.



Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Algorithm: Call **Tree-Minimum**$(T.root)$ to find the element with minimum key in tree T.

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Algorithm: Call **Tree-Minimum**$(T.root)$ to find the element with minimum key in tree T.

```
Tree-Minimum(x):
```

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Algorithm: Call **Tree-Minimum**$(T.root)$ to find the element with minimum key in tree T.

```
Tree-Minimum(x):
1.    while $x.left \neq$ NIL
```

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Algorithm: Call **Tree-Minimum**$(T\,.\,root)$ to find the element with minimum key in tree T.

```
Tree-Minimum($x$):
1.  while $x\,.\,left \neq$ NIL
2.      $x = x\,.\,left$
```

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Algorithm: Call **Tree-Minimum**($T\,.\,root$) to find the element with minimum key in tree T.

```
Tree-Minimum($x$):
1.  while $x\,.\,left \neq \text{NIL}$
2.      $x = x\,.\,left$
3.  return $x$
```

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Algorithm: Call **Tree-Minimum**$(T\,.\,root)$ to find the element with minimum key in tree T.

```
Tree-Minimum($x$):
1.  while $x\,.\,left \neq$ NIL
2.      $x = x\,.\,left$
3.  return $x$
```

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

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Algorithm: Call **Tree-Minimum**($T\,.\,root$) to find the element with minimum key in tree T.

```
Tree-Minimum($x$):
1.  while $x\,.\,left \neq$ NIL
2.          $x = x\,.\,left$
3.  return $x$
```

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

Querying a BST: Minimum

Goal: Given a pointer to the root of a BST to find the element with minimum key.

Algorithm: Call **Tree-Minimum**$(T.root)$ to find the element with minimum key in tree T.

```
Tree-Minimum(x):
1.  while $x.left \neq$ NIL
2.      $x = x.left$
3.  return $x$
```

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

Finding Maximum: Symmetrically opposite to finding minimum.

Querying a BST: Successor

Querying a BST: Successor

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. $\leftarrow$ node printed after x in inorder-walk.

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← node printed after x in inorder-walk.

Illustration:

Querying a BST: Successor

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 below BST.

Querying a BST: Successor

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 below BST.

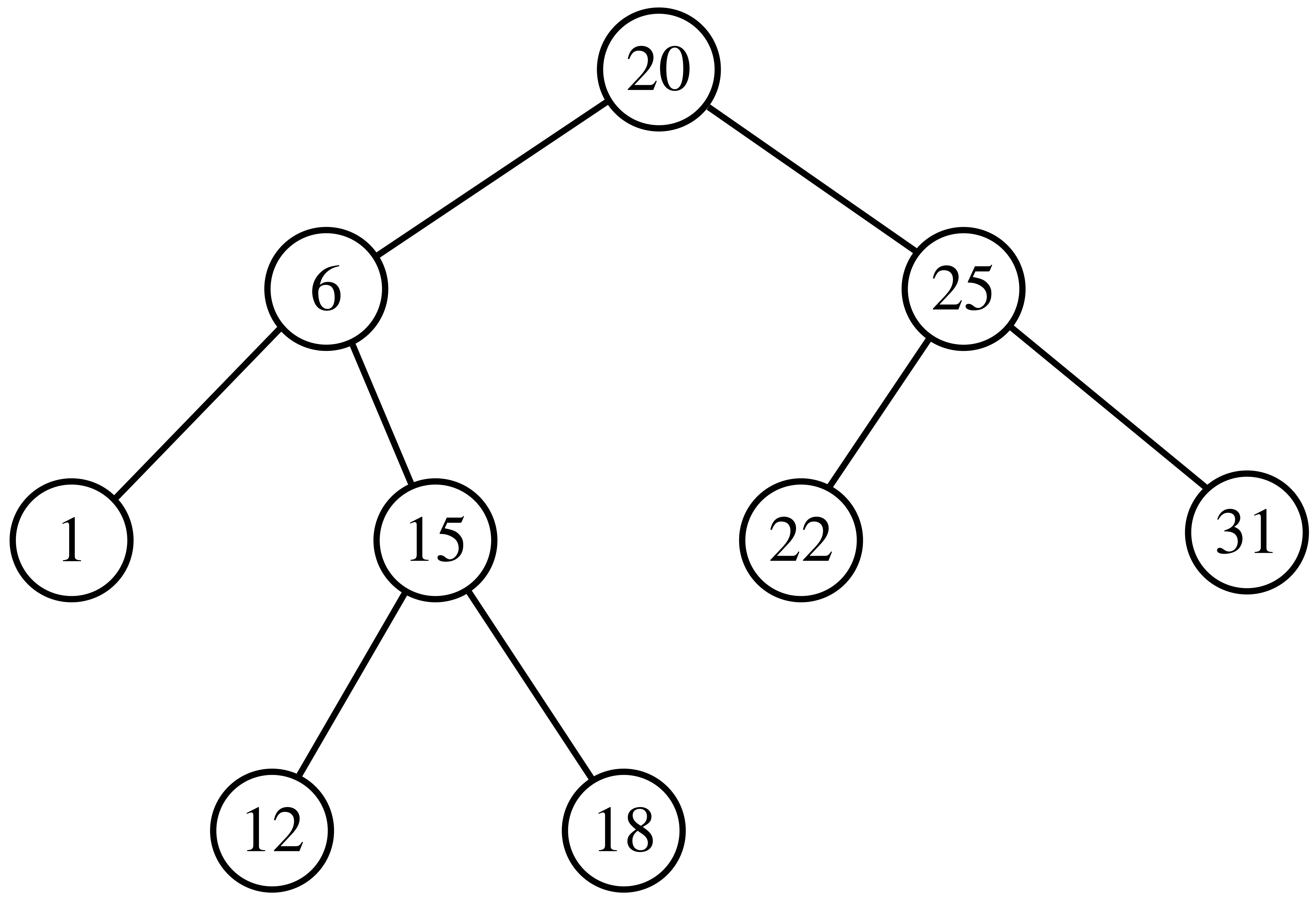
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← node printed after x in inorder-walk.

Illustration: Find the successor of 6 in below BST.



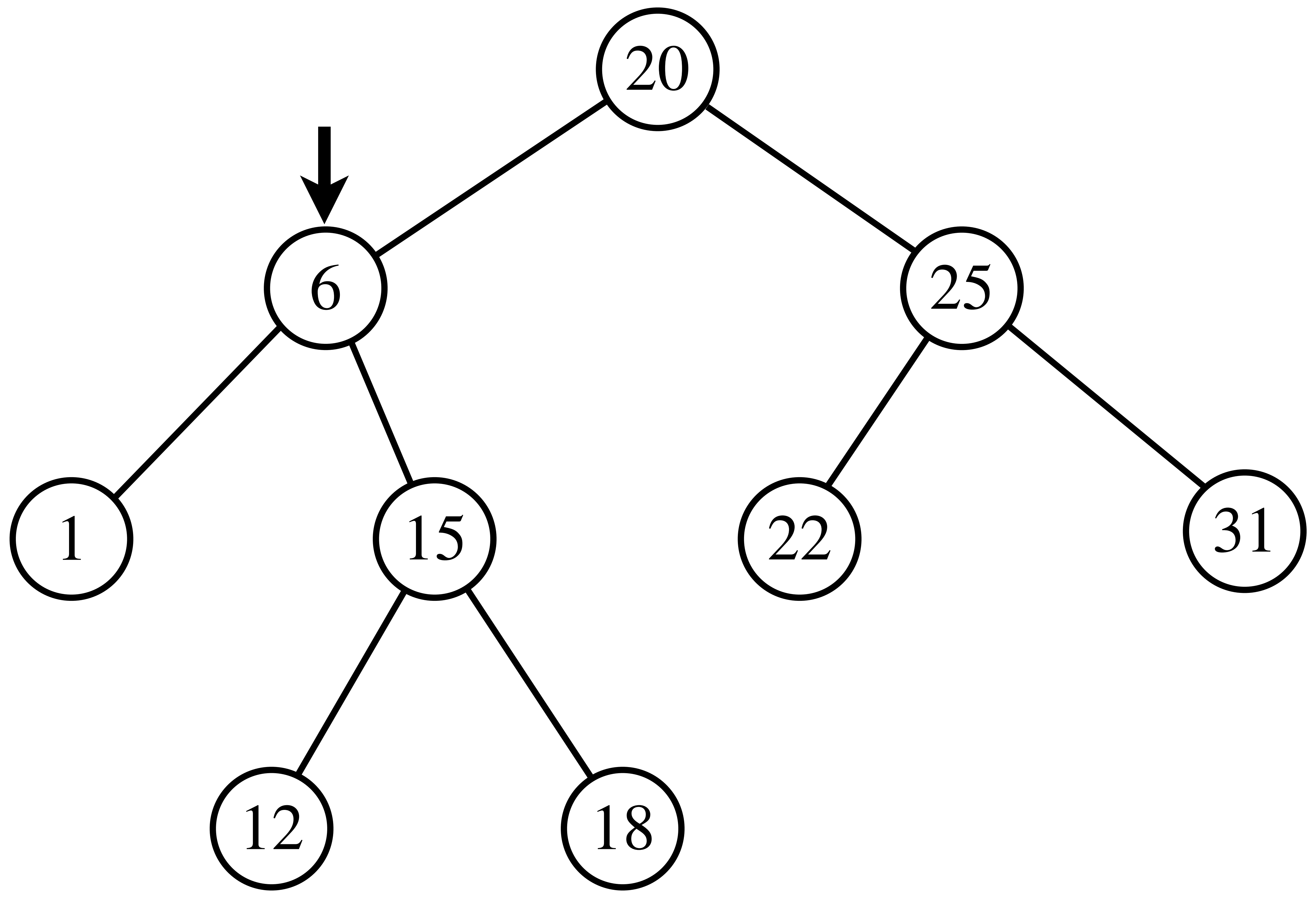
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← node printed after x in inorder-walk.

Illustration: Find the successor of 6 in below BST.



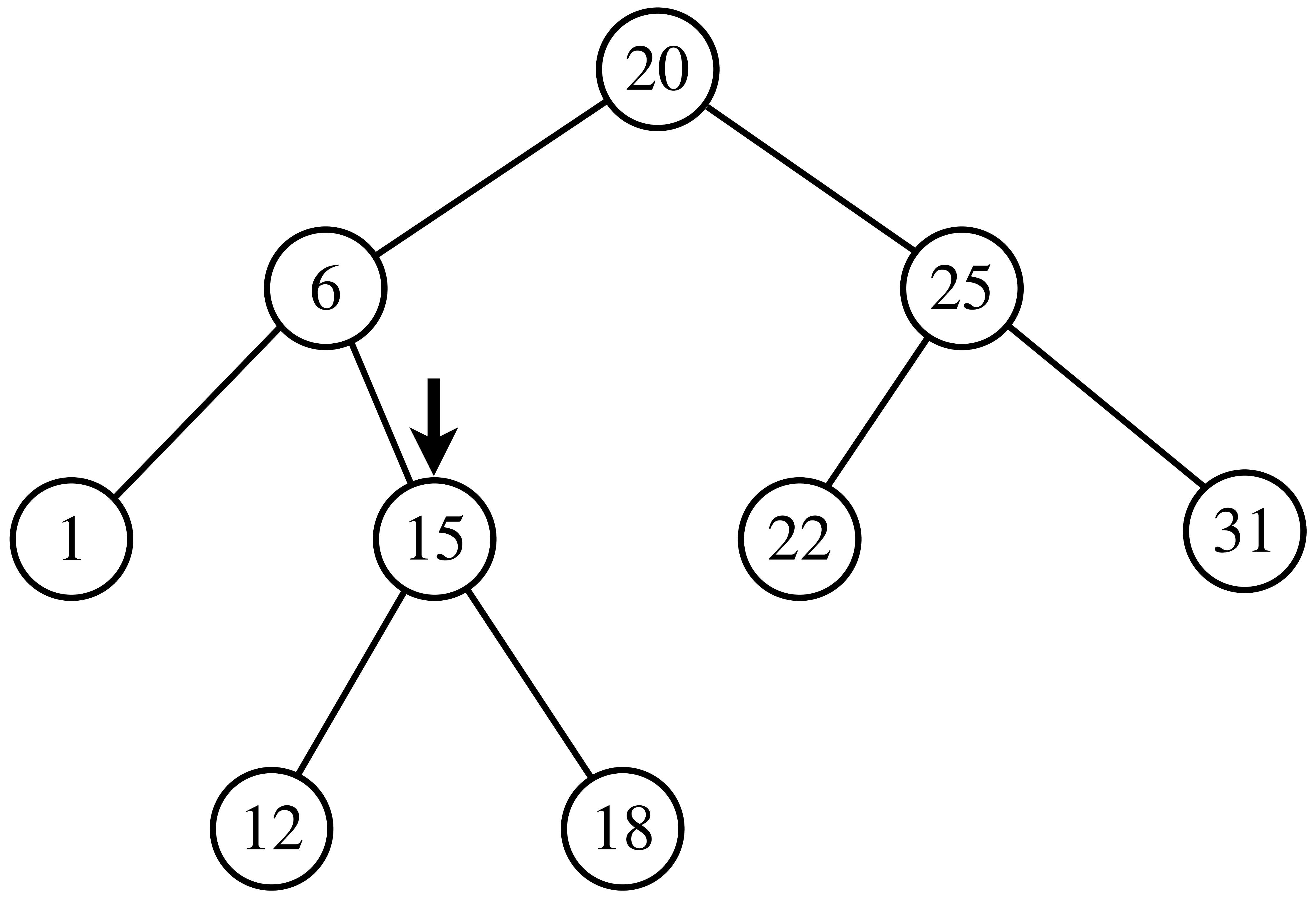
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← node printed after x in inorder-walk.

Illustration: Find the successor of 6 in below BST.



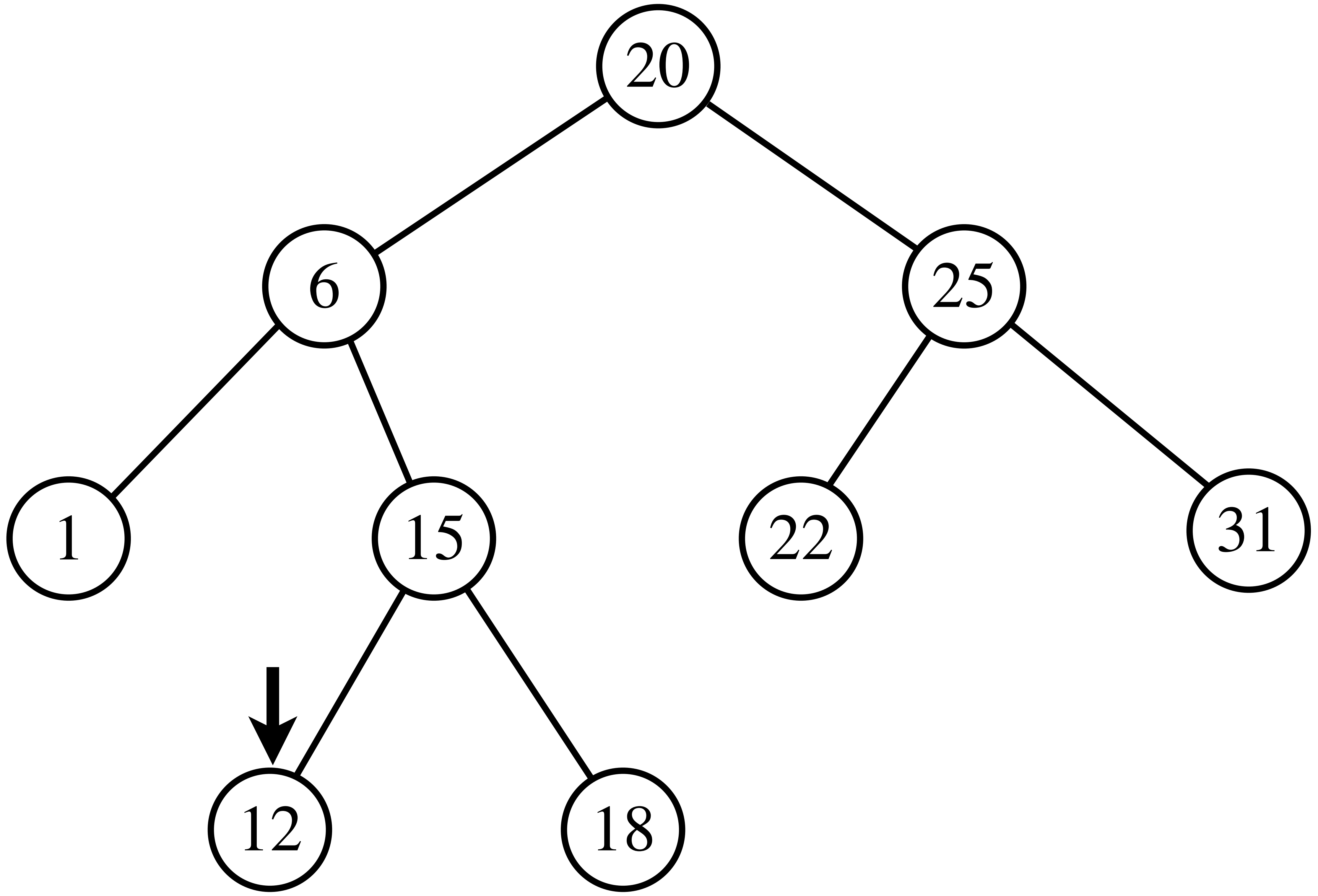
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← node printed after x in inorder-walk.

Illustration: Find the successor of 6 in below BST.



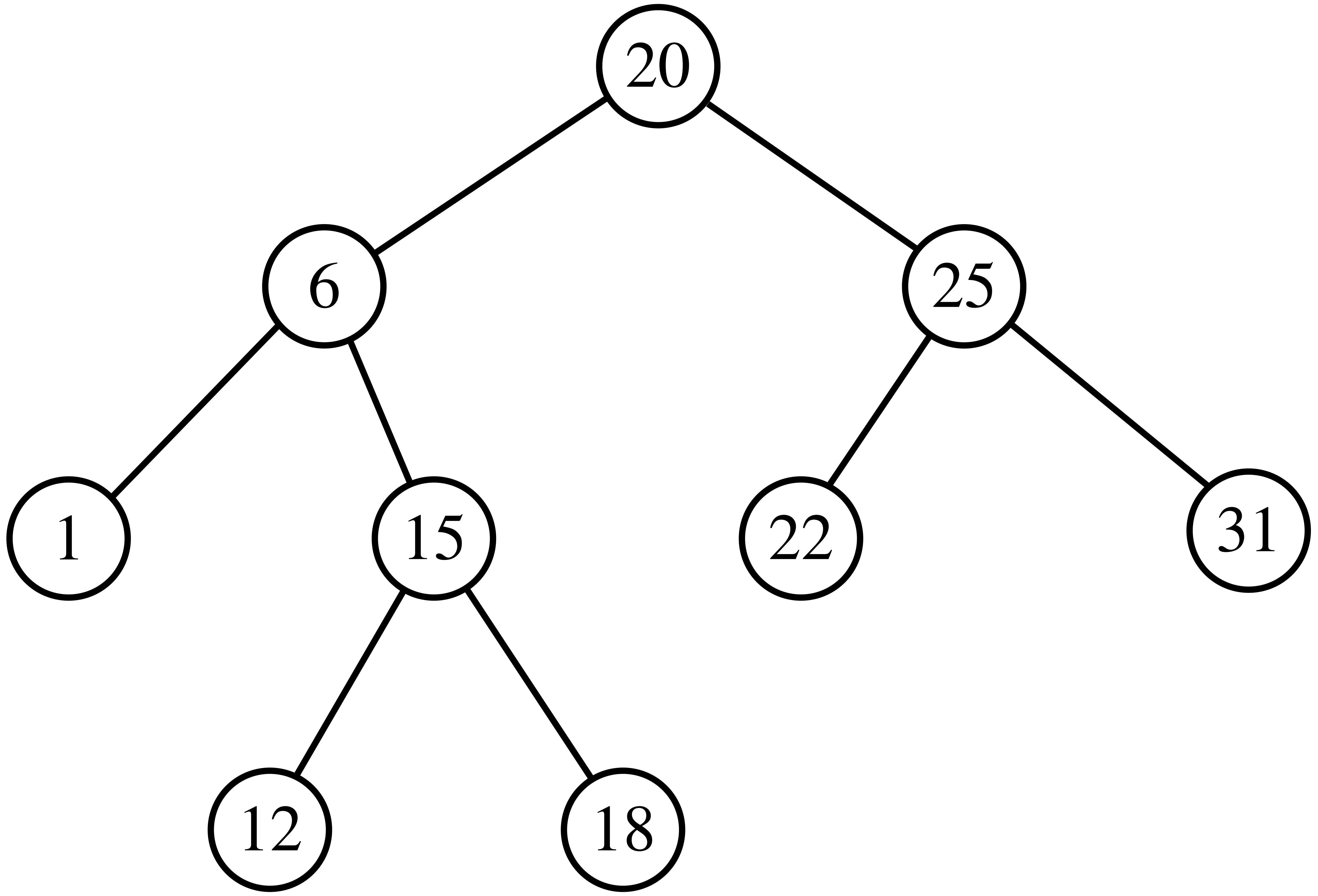
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← node printed after x in inorder-walk.

Illustration:



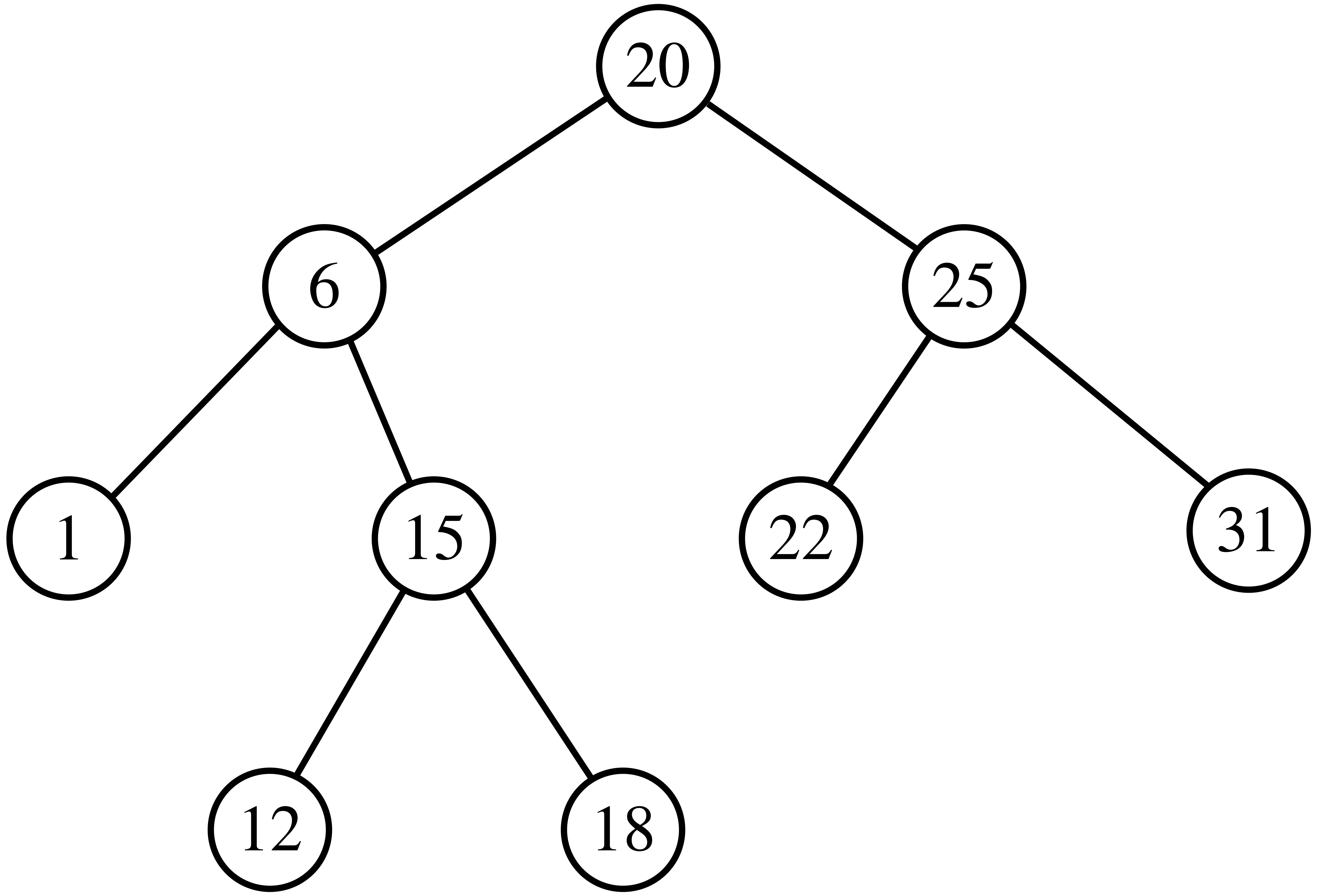
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← node printed after x in inorder-walk.

Illustration: Find the successor of 18 in below BST.



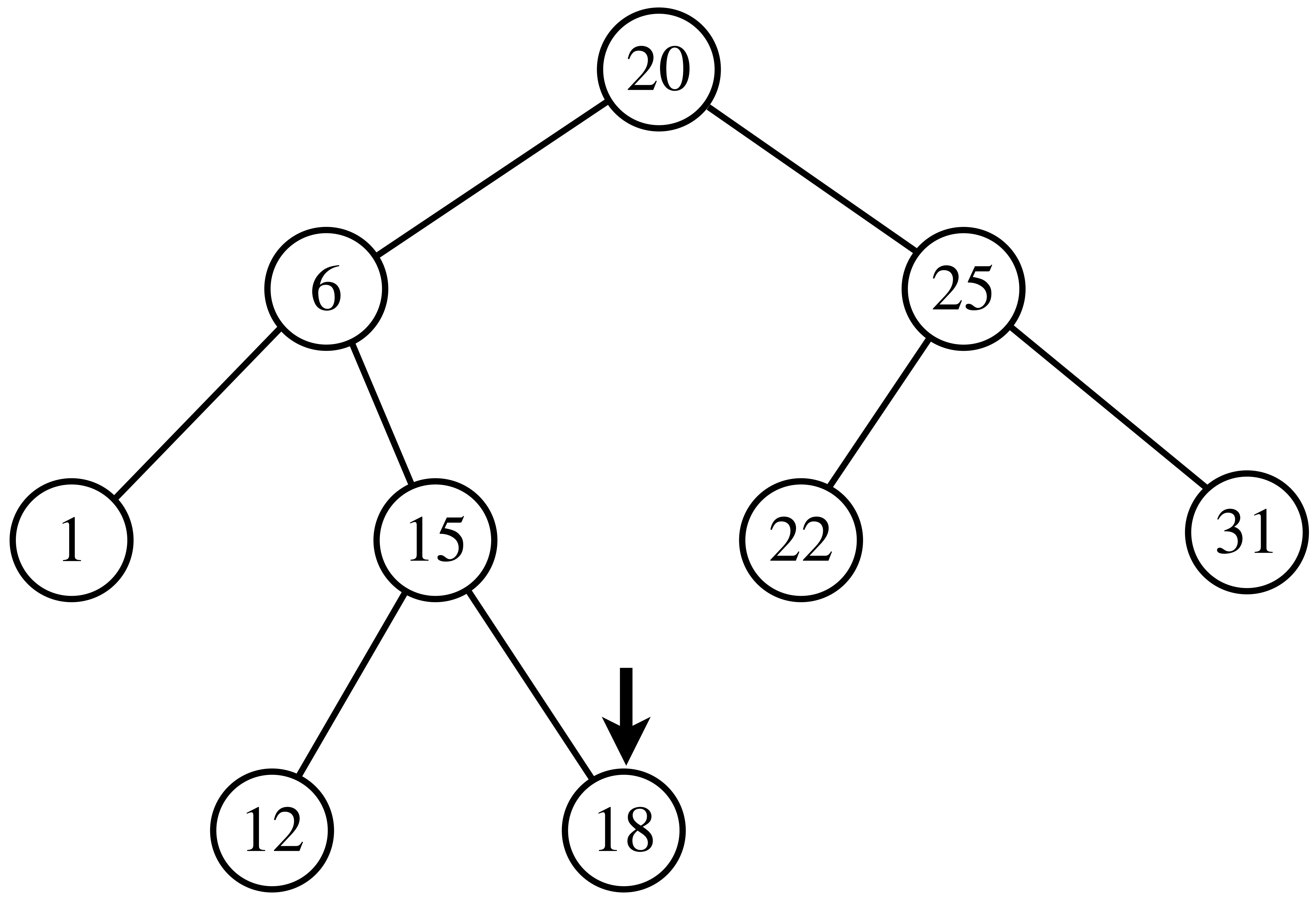
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← *node printed after x in inorder-walk.*

Illustration: Find the successor of 18 in below BST.



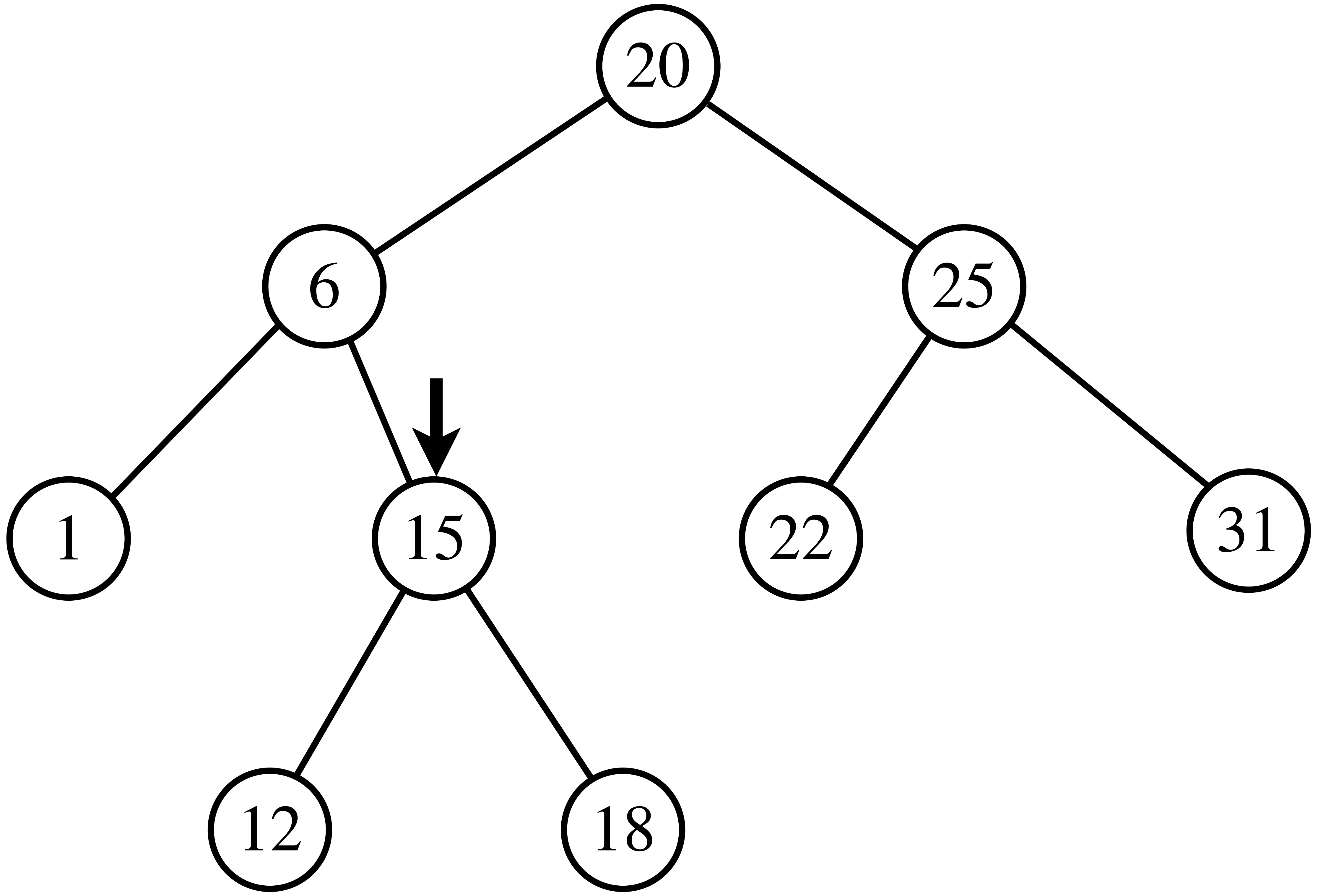
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← *node printed after x in inorder-walk.*

Illustration: Find the successor of 18 in below BST.



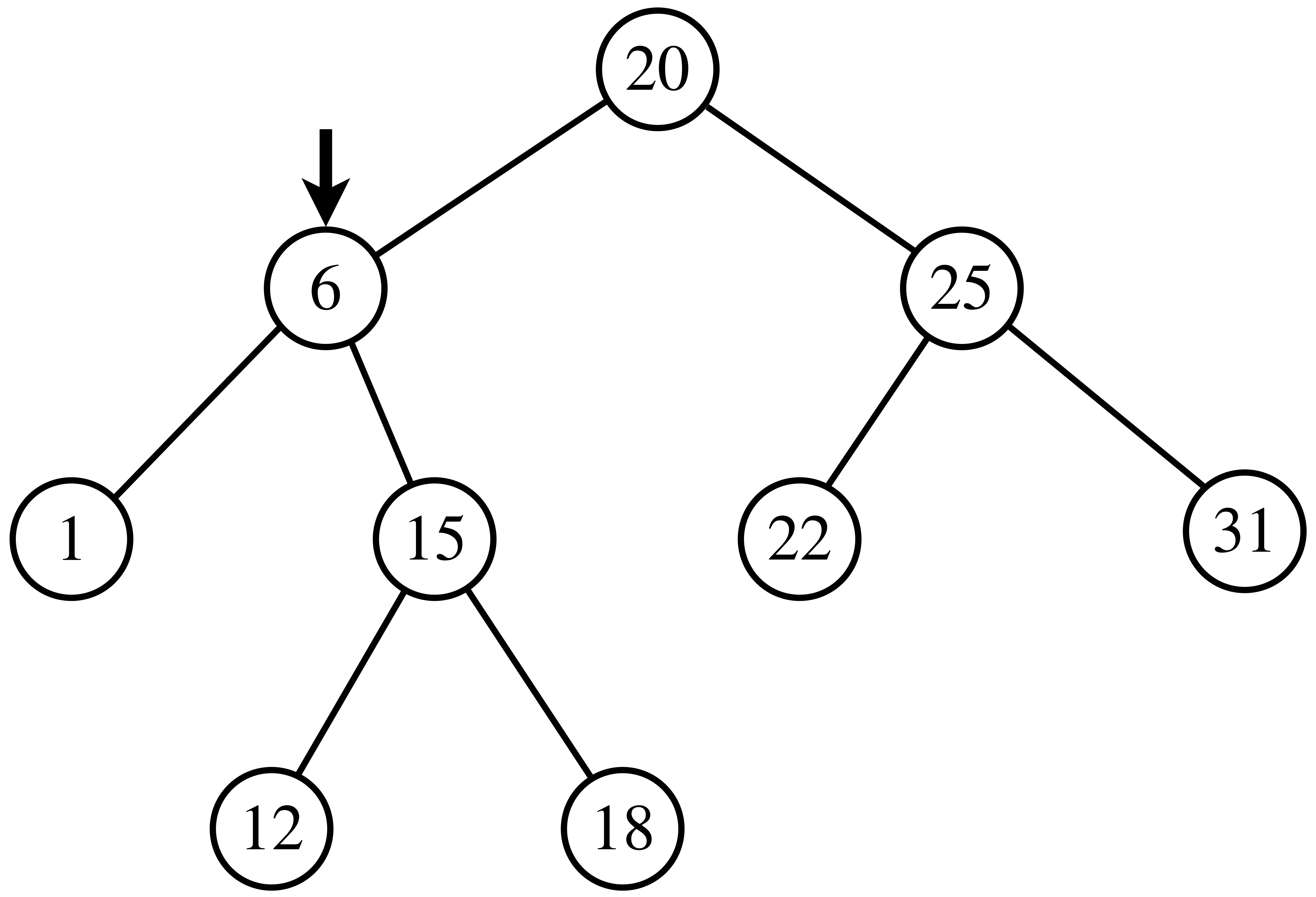
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← node printed after x in inorder-walk.

Illustration: Find the successor of 18 in below BST.



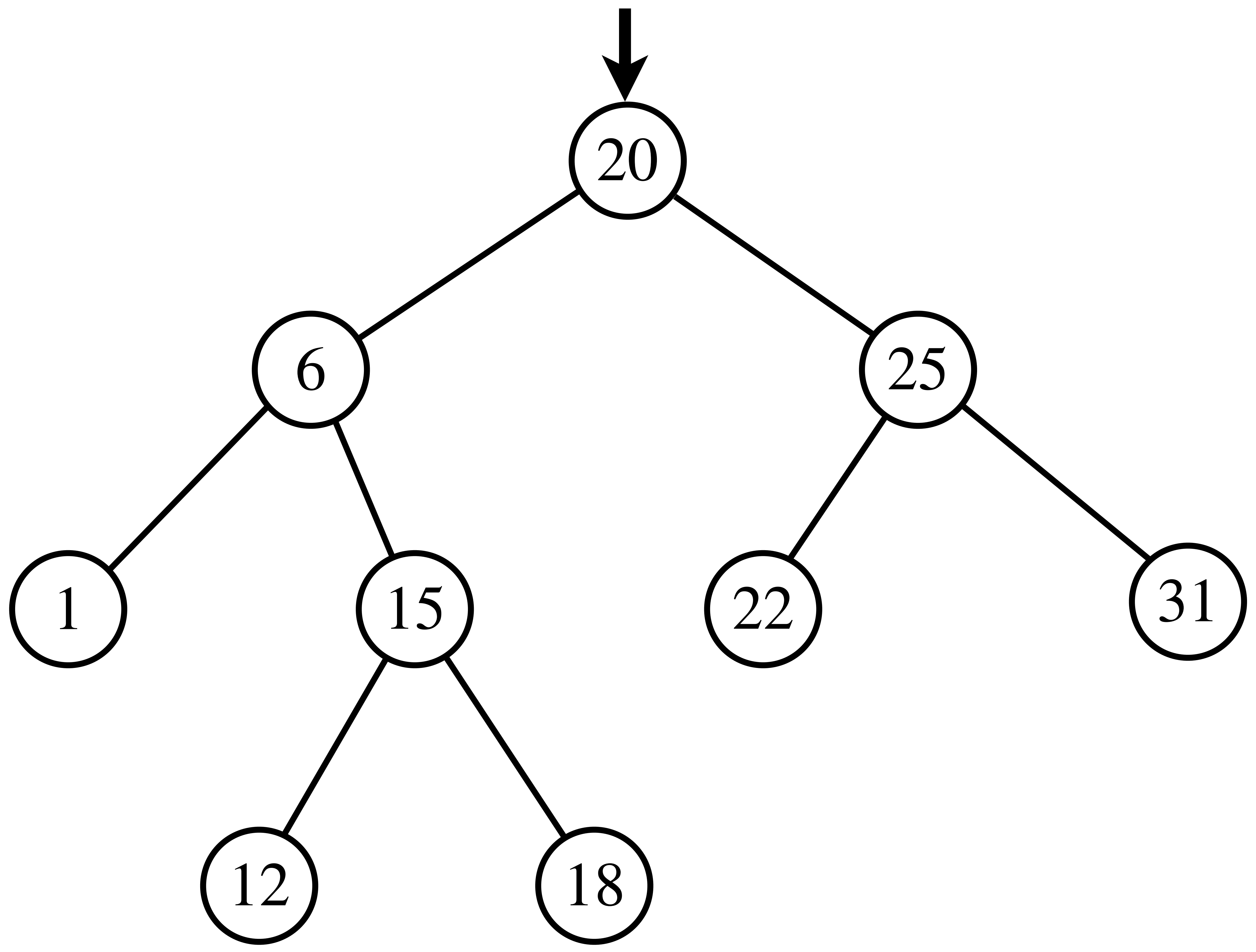
Recall:

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

Querying a BST: Successor

Goal: Given a node x of a BST find its successor. ← node printed after x in inorder-walk.

Illustration: Find the successor of 18 in 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($x.right$)
```

Querying a BST: Successor

Querying a BST: Successor

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

Querying a BST: Successor

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

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

Querying a BST: Successor

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($x$):
```

Querying a BST: Successor

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
```

Querying a BST: Successor

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($x\,.\,right$)
```

Querying a BST: Successor

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
```

Querying a BST: Successor

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.      $y = x.p$
```

Querying a BST: Successor

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 $x \neq y.left$
```

Querying a BST: Successor

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 $x \neq y.left$
6.          $x = y, y = y.p$
```

Querying a BST: Successor

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

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

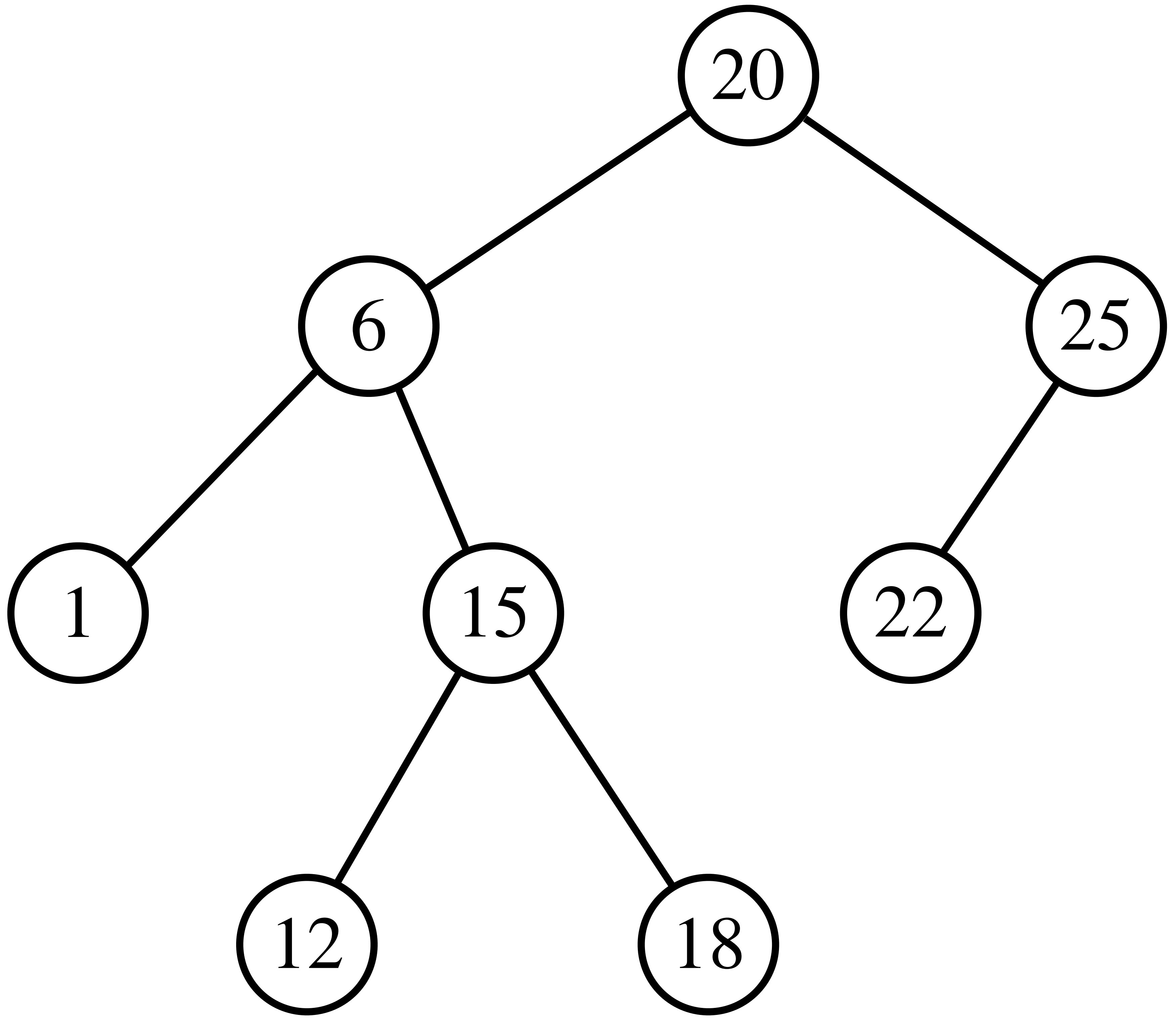
```
Tree-Successor($x$):
1.  if $x.right \neq$ NIL
2.      return Tree-Minimum($x.right$)
3.  else
4.      $y = x.p$
5.      while $y \neq$ NIL and $x \neq y.left$
6.          $x = y, y = y.p$
7.      return $y$
```

Querying a BST: Successor

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 $x \neq y.left$
6.          $x = y, y = y.p$
7.      return $y$
```

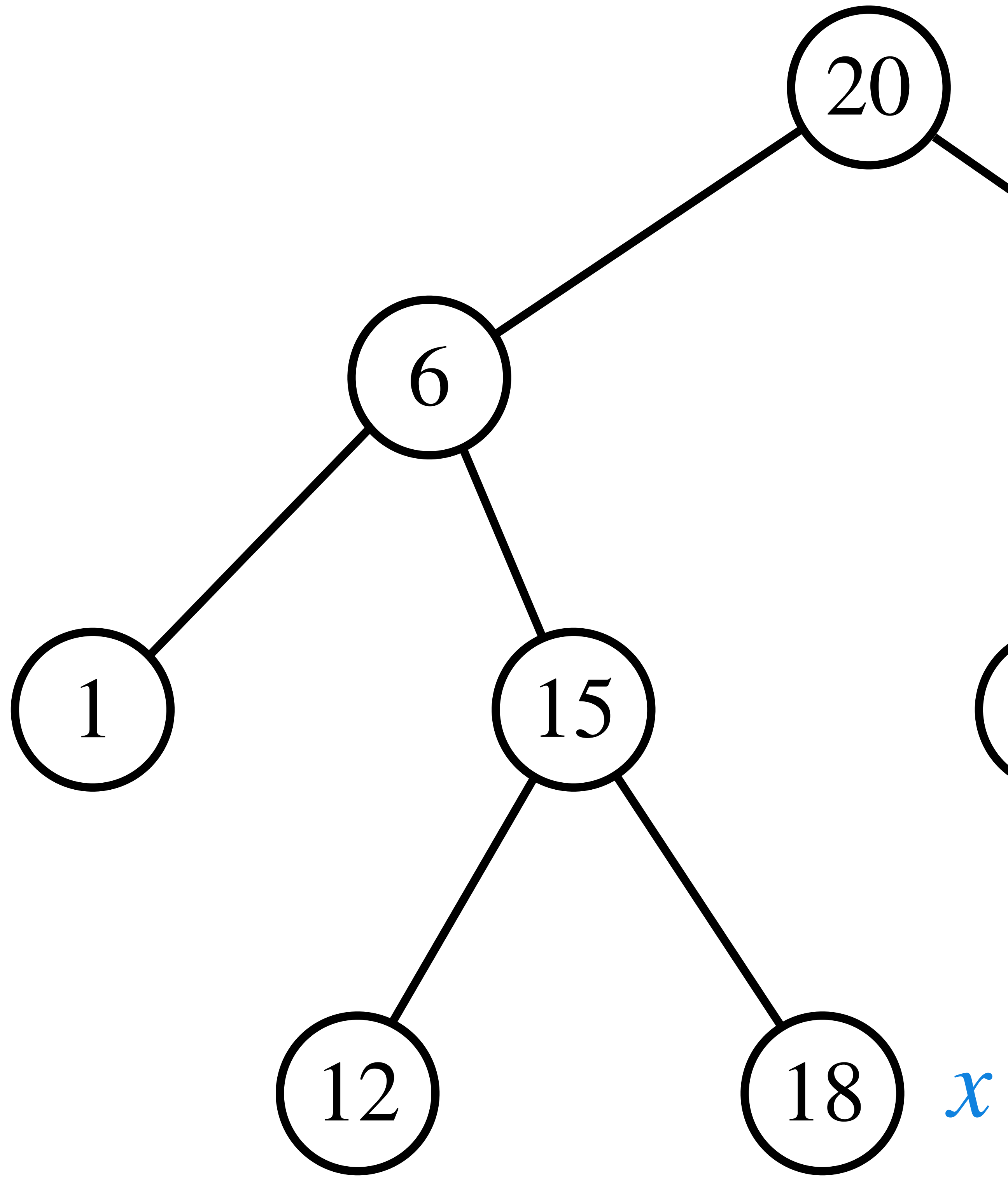


Querying a BST: Successor

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 $x \neq y.left$
6.          $x = y, y = y.p$
7.      return $y$
```

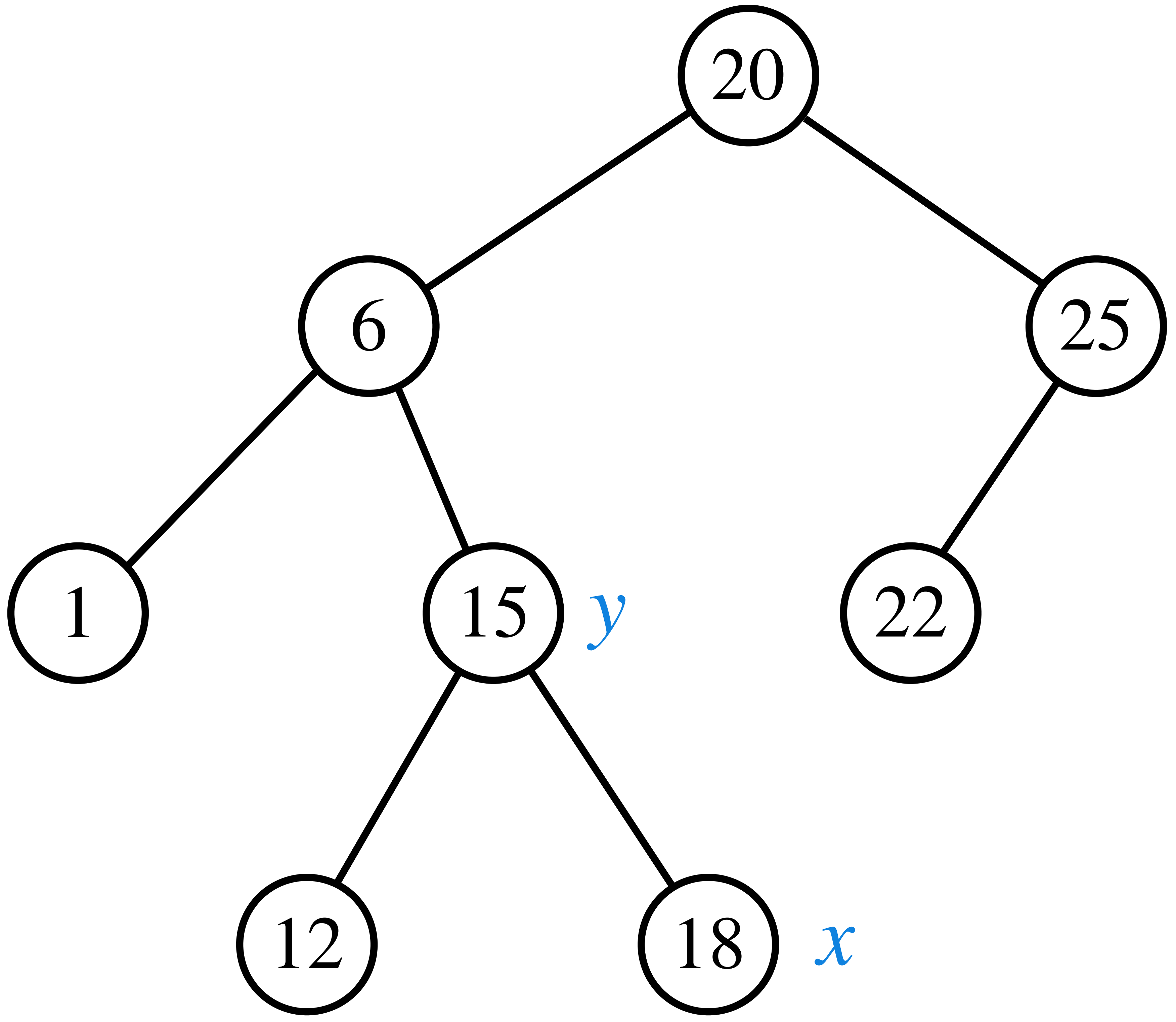


Querying a BST: Successor

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 $x \neq y.left$
6.          $x = y, y = y.p$
7.      return $y$
```

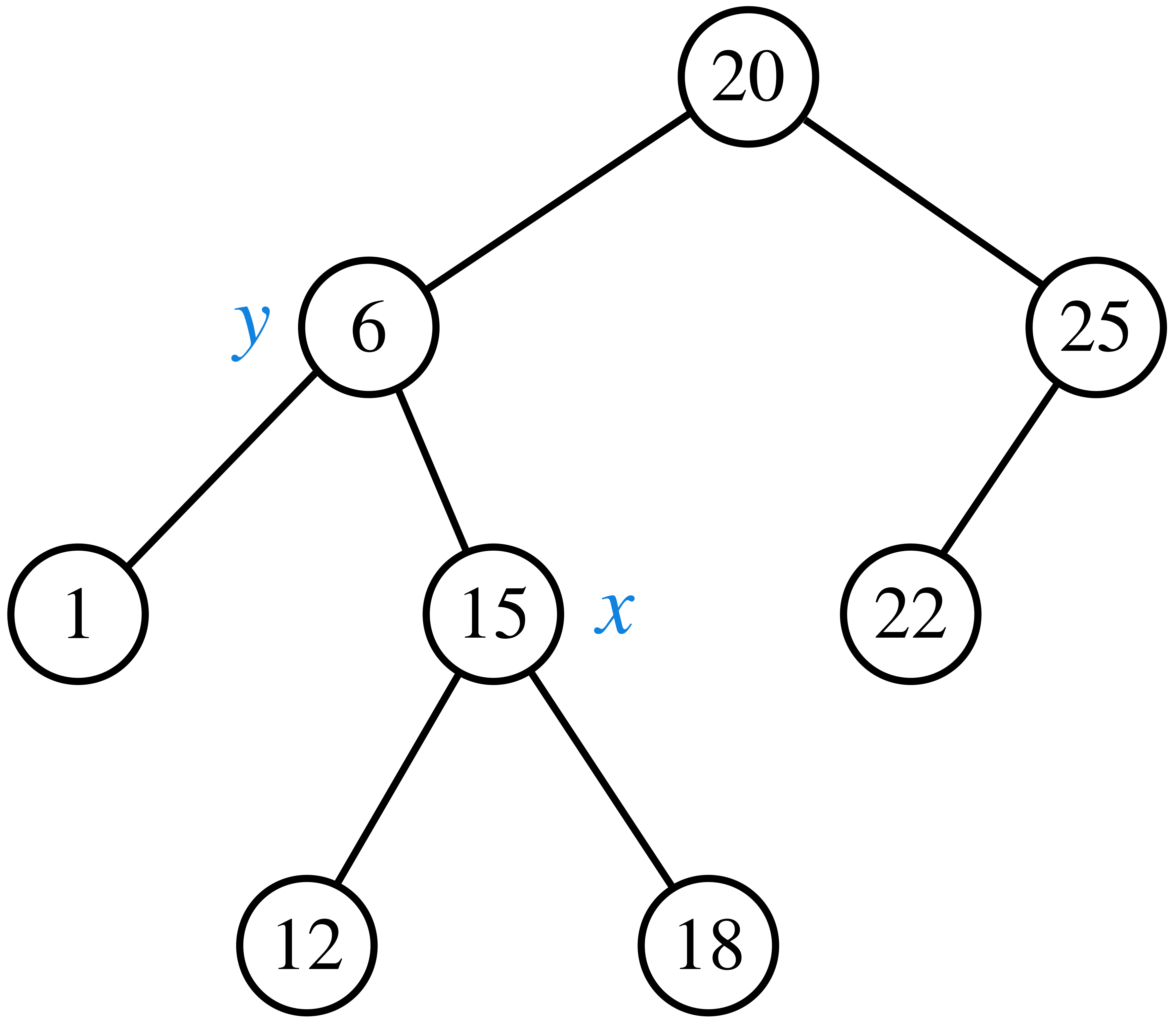


Querying a BST: Successor

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 $x \neq y.left$
6.          $x = y, y = y.p$
7.      return $y$
```

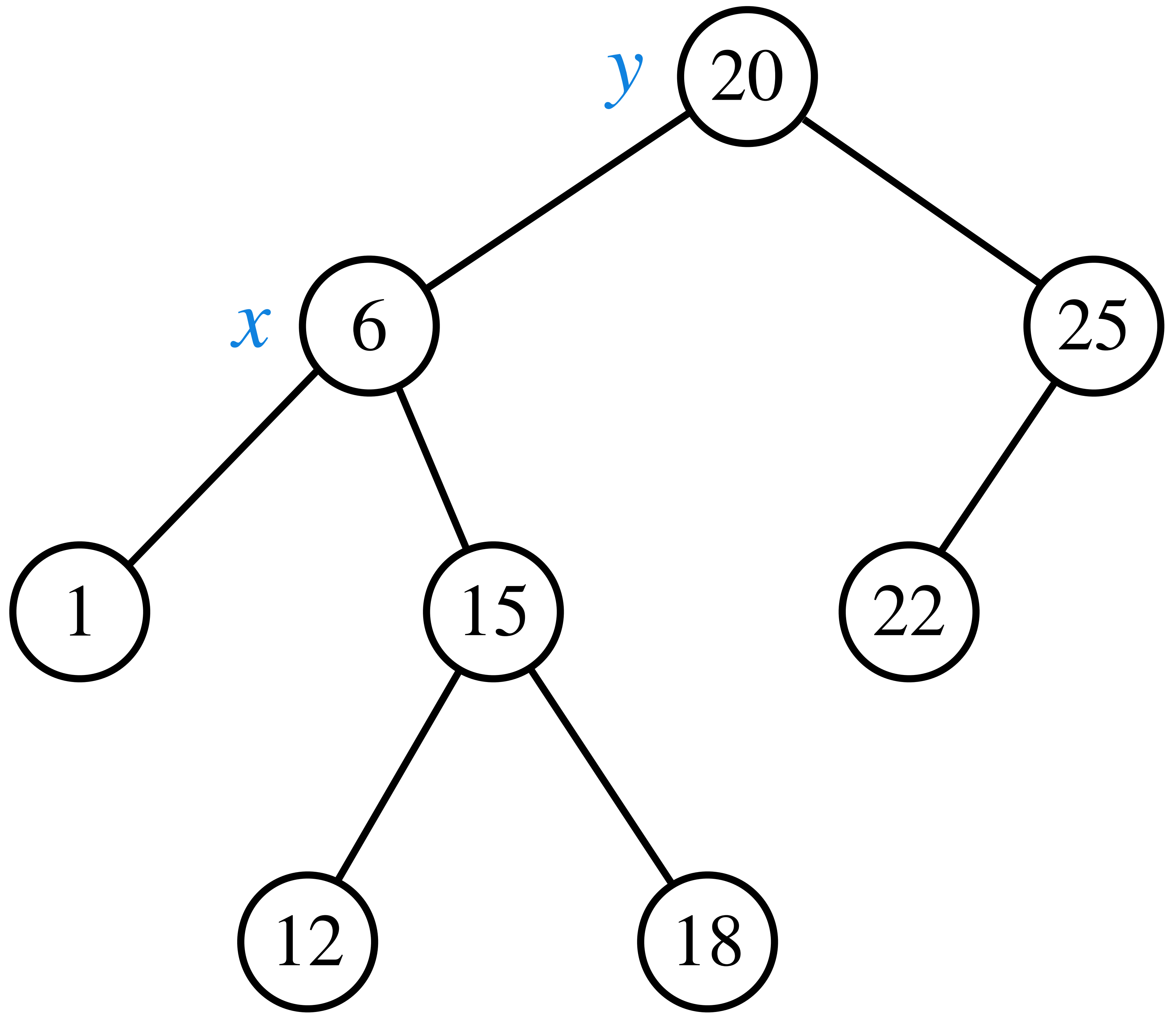


Querying a BST: Successor

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 $x \neq y.left$
6.          $x = y, y = y.p$
7.      return $y$
```

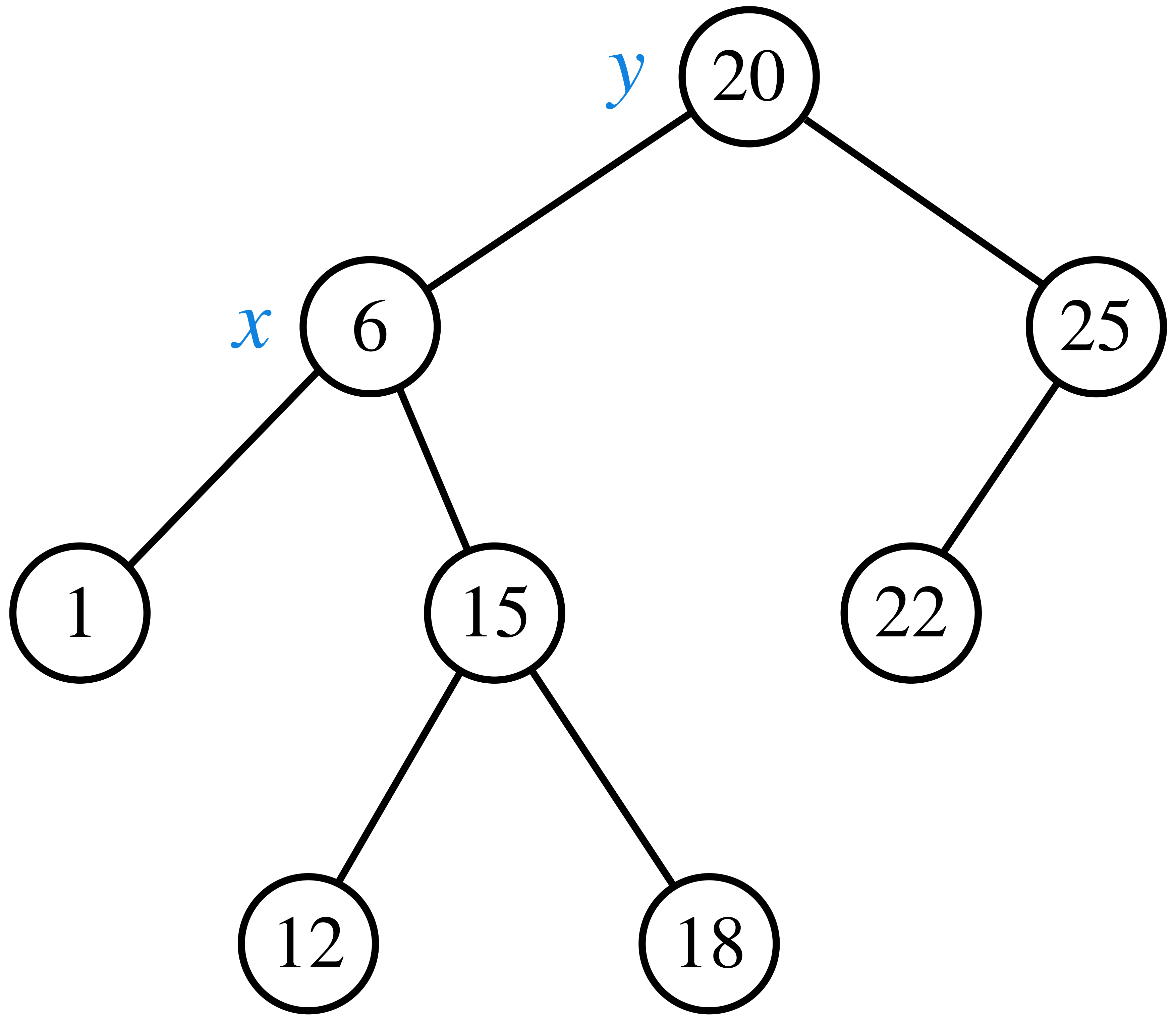


Querying a BST: Successor

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 $x \neq y.left$
6.          $x = y, y = y.p$
7.      return $y$
```



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