Lecture 2

Binary Search Trees

Binary Search Trees

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

- Insert an element.

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

- Insert an element.
- Search for an element with the key k.

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

- Insert an element.
- Search for an element with the key k.
- Delete an element.

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

- Insert an element.
- Search for an element with the key k.
- Delete an element.
- Minimum or Maximum of the set.

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

- Insert an element.
- Search for an element with the key k.
- Delete an element.
- Minimum or Maximum of the set.
- Successor or Predecessor of an element of the set.

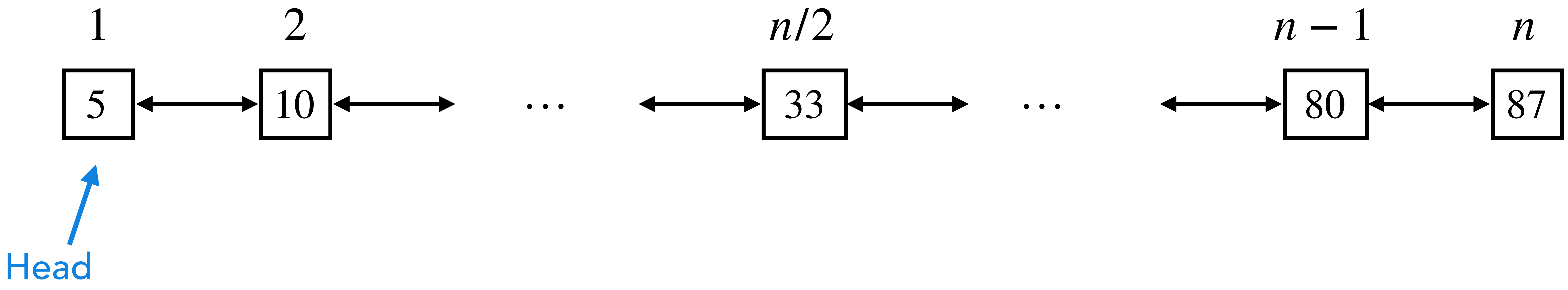
Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

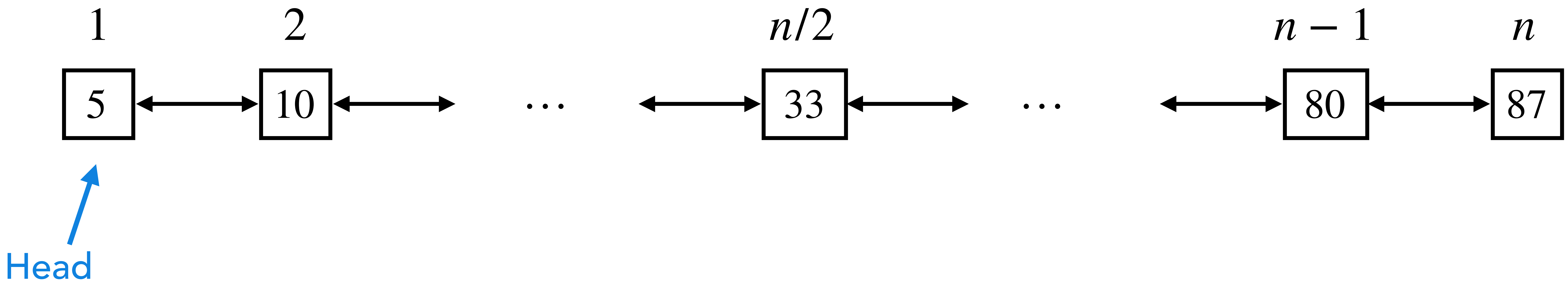
- Insert an element.
- Search for an element with the key k.
- Delete an element.
- Minimum or Maximum of the set.
- Successor or Predecessor of an element of the set.

Let's first implement the operations through sorted linked list

Linked List Implementation

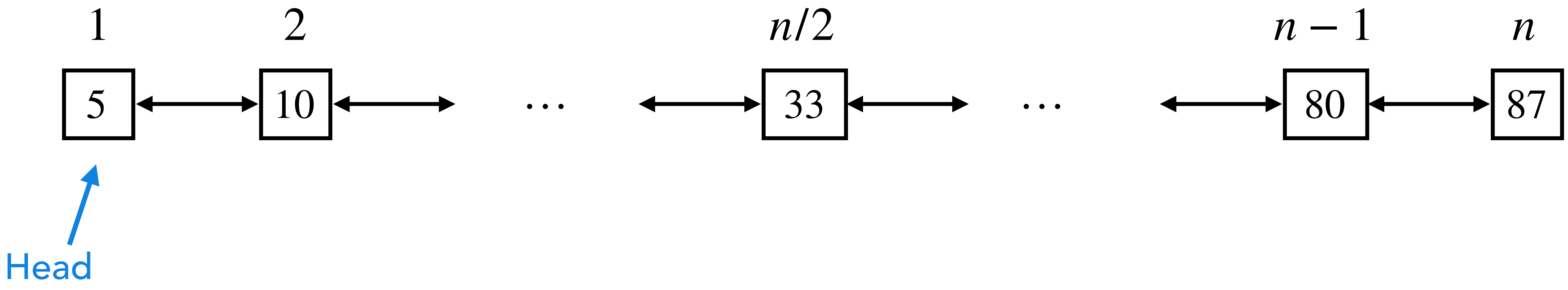


Linked List Implementation



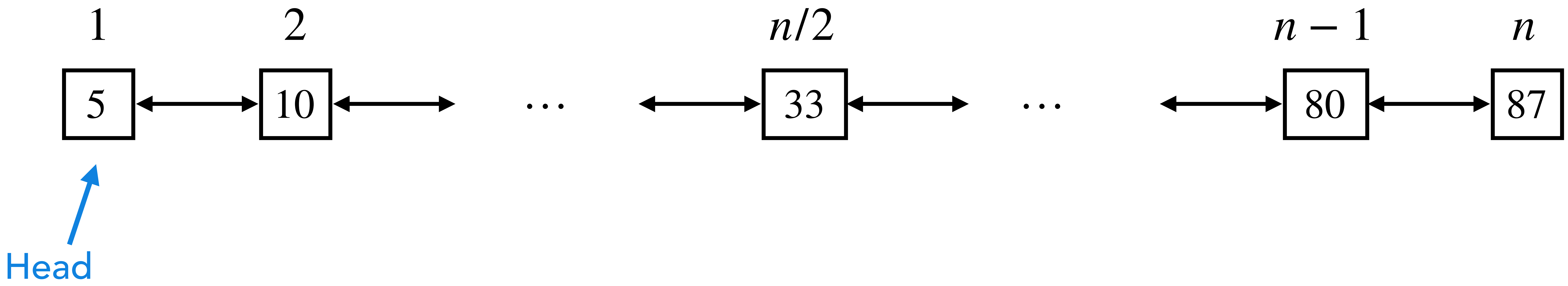
	Time required in Linked list implementation
Insert	
Search	
Delete	
Min/Max	
Succ/Pred	

Linked List Implementation



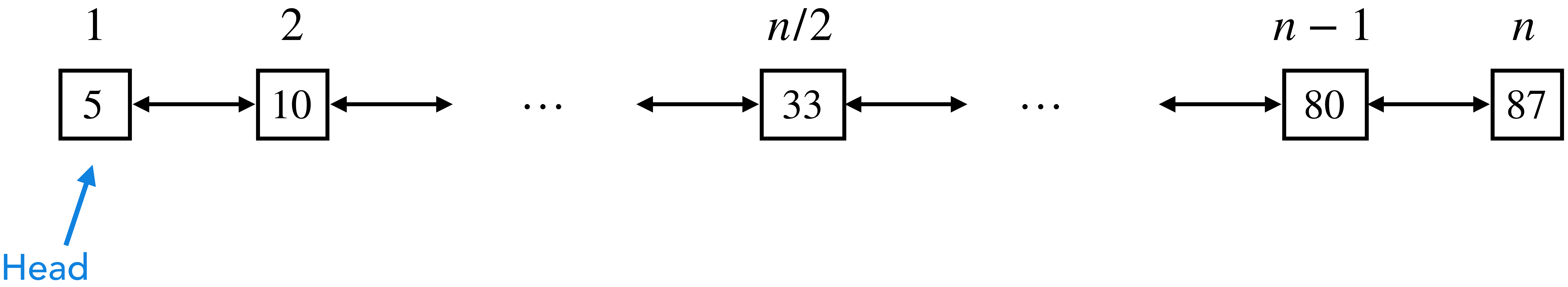
	Time required in Linked list implementation
Insert	$\Theta(n)$
Search	
Delete	
Min/Max	
Succ/Pred	

Linked List Implementation



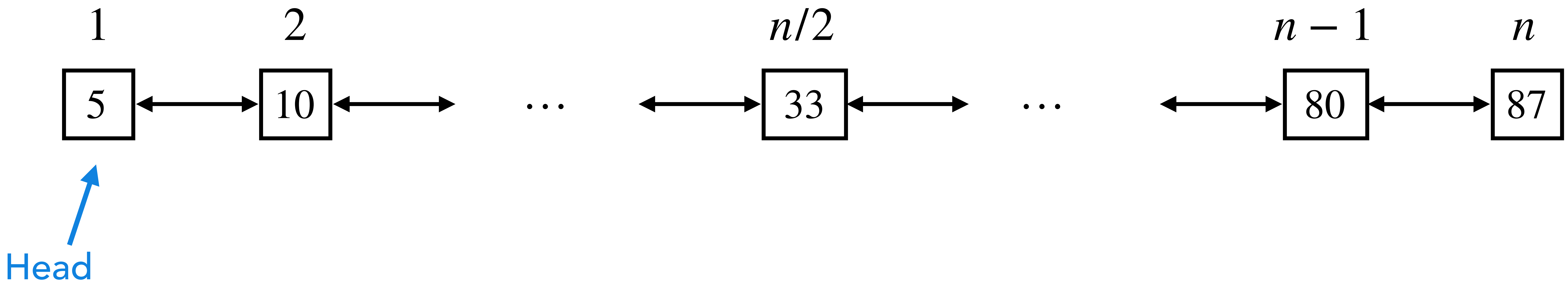
	Time required in Linked list implementation
Insert	$\Theta(n)$
Search	$\Theta(n)$
Delete	
Min/Max	
Succ/Pred	

Linked List Implementation



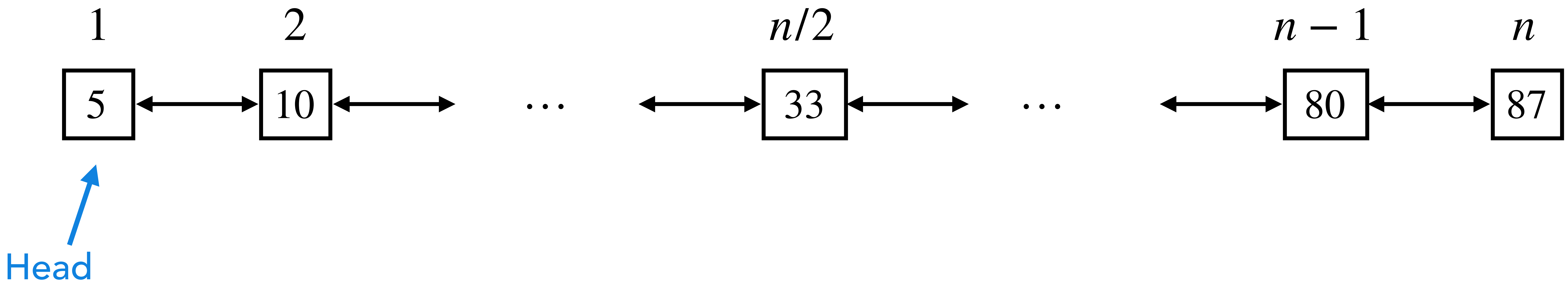
	Time required in Linked list implementation
Insert	$\Theta(n)$
Search	$\Theta(n)$
Delete	$\Theta(1)$
Min/Max	
Succ/Pred	

Linked List Implementation



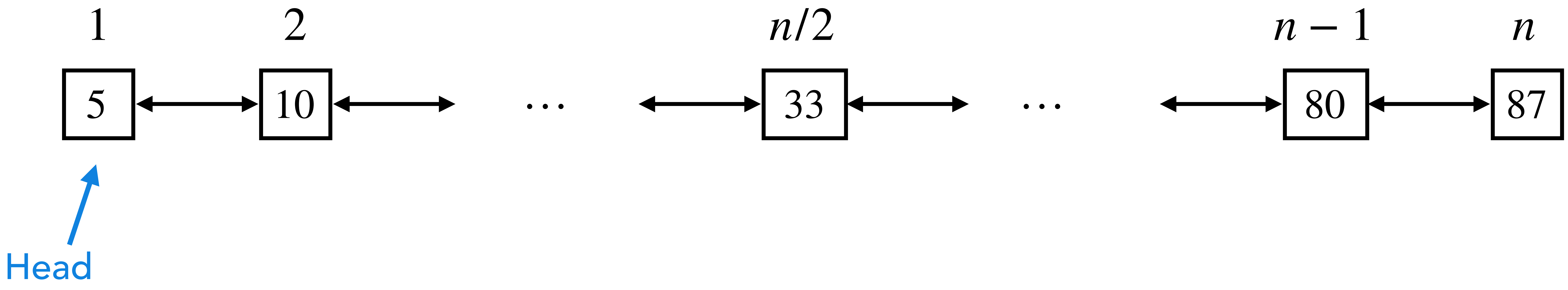
	Time required in Linked list implementation
Insert	$\Theta(n)$
Search	$\Theta(n)$
Delete	$\Theta(1)$
Min/Max	$\Theta(1), \Theta(n)$
Succ/Pred	

Linked List Implementation



	Time required in Linked list implementation
Insert	$\Theta(n)$
Search	$\Theta(n)$
Delete	$\Theta(1)$
Min/Max	$\Theta(1), \Theta(n)$
Succ/Pred	$\Theta(1)$

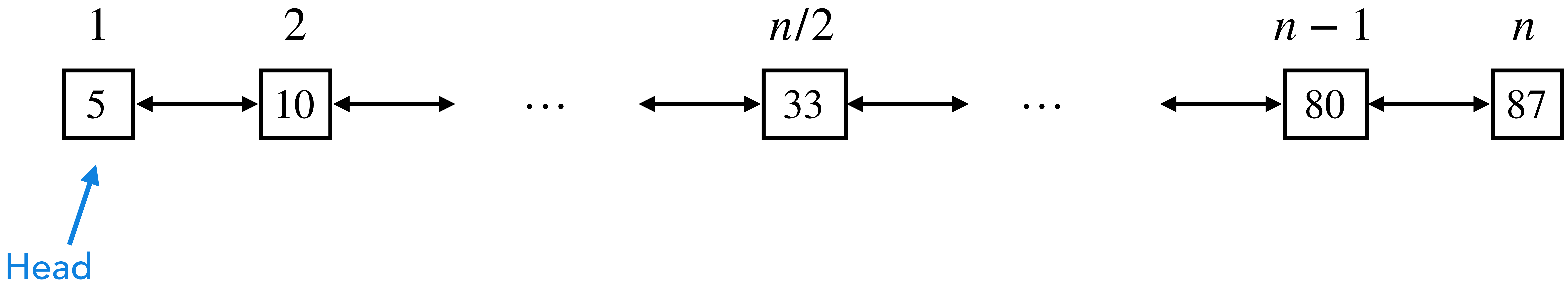
Linked List Implementation



	Time required in Linked list implementation
Insert	$\Theta(n)$
Search	$\Theta(n)$
Delete	$\Theta(1)$
Min/Max	$\Theta(1), \Theta(n)$
Succ/Pred	$\Theta(1)$

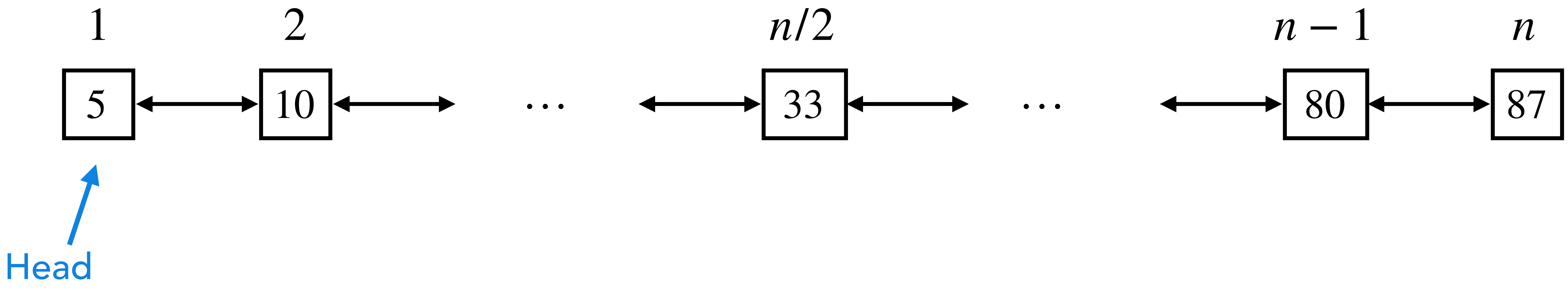
Let's try to reduce this first

Reducing Search Time in Linked Lists



Source: *DSA Slides of Surender Baswana*

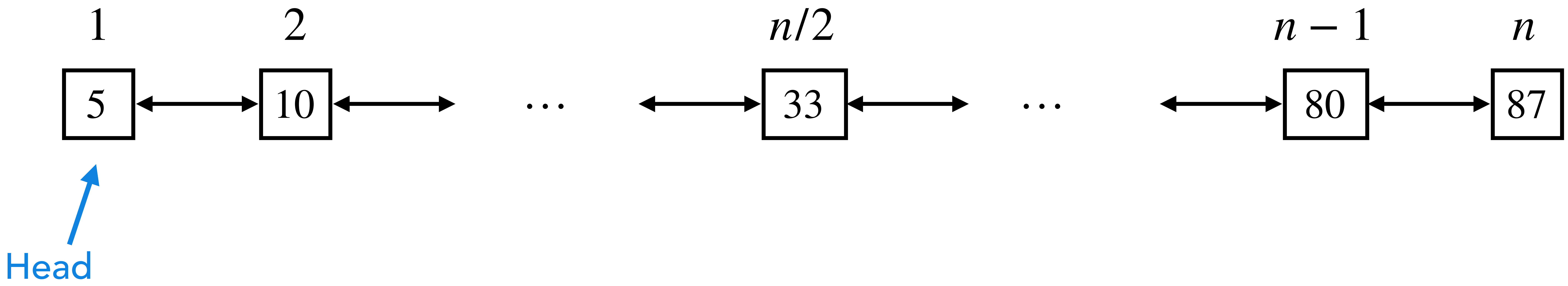
Reducing Search Time in Linked Lists



Searching for a node may take at most n comparisons.

Source: *DSA Slides of Surender Baswana*

Reducing Search Time in Linked Lists

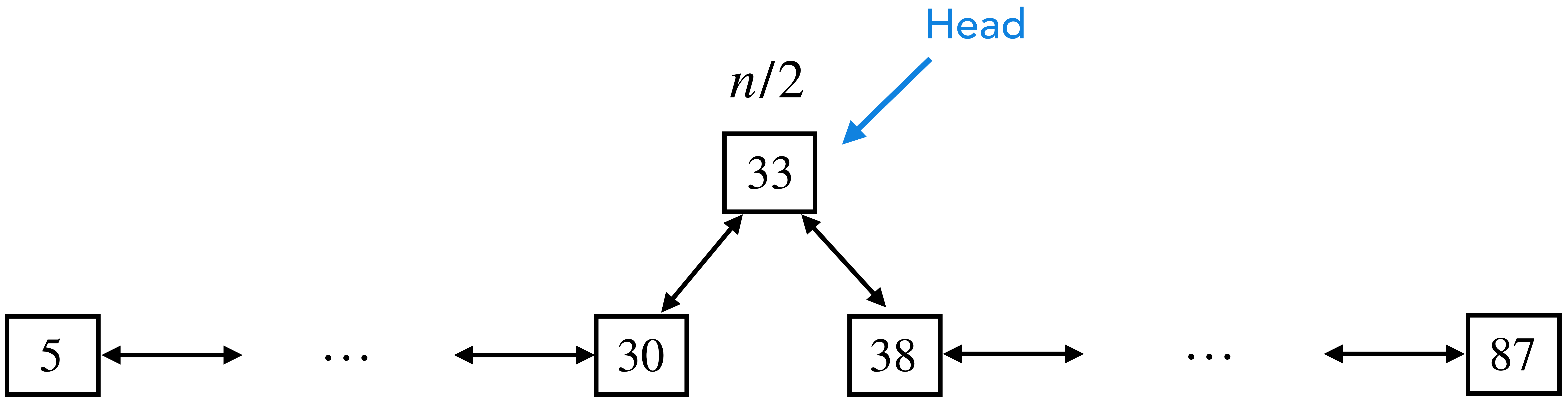


Searching for a node may take at most n comparisons.

What can be done to find a node with around $n/2$ comparisons?

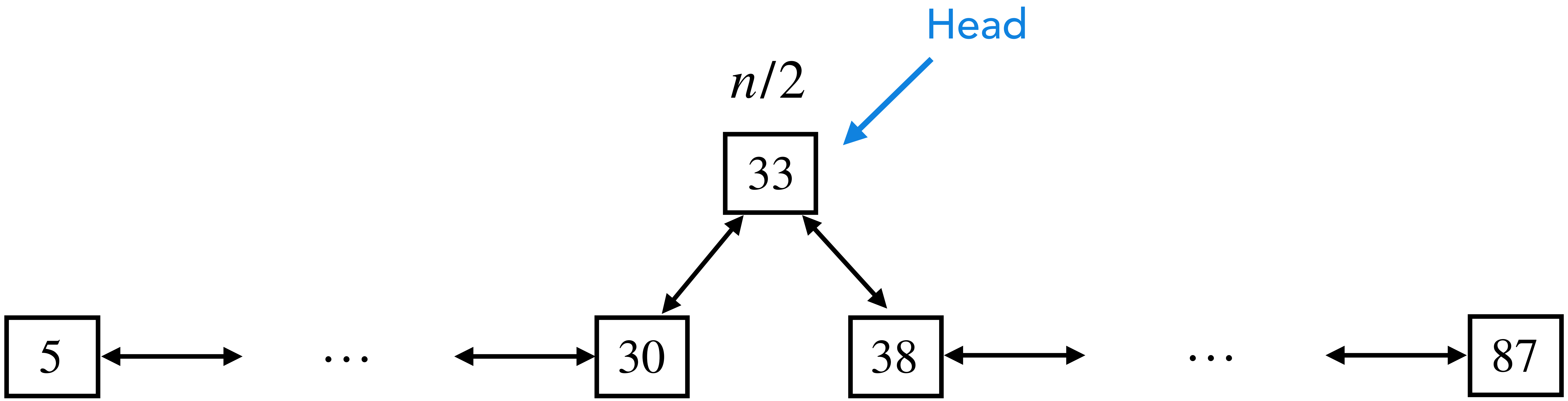
Source: *DSA Slides of Surender Baswana*

Reducing Search Time in Linked Lists



Source: *DSA Slides of Surender Baswana*

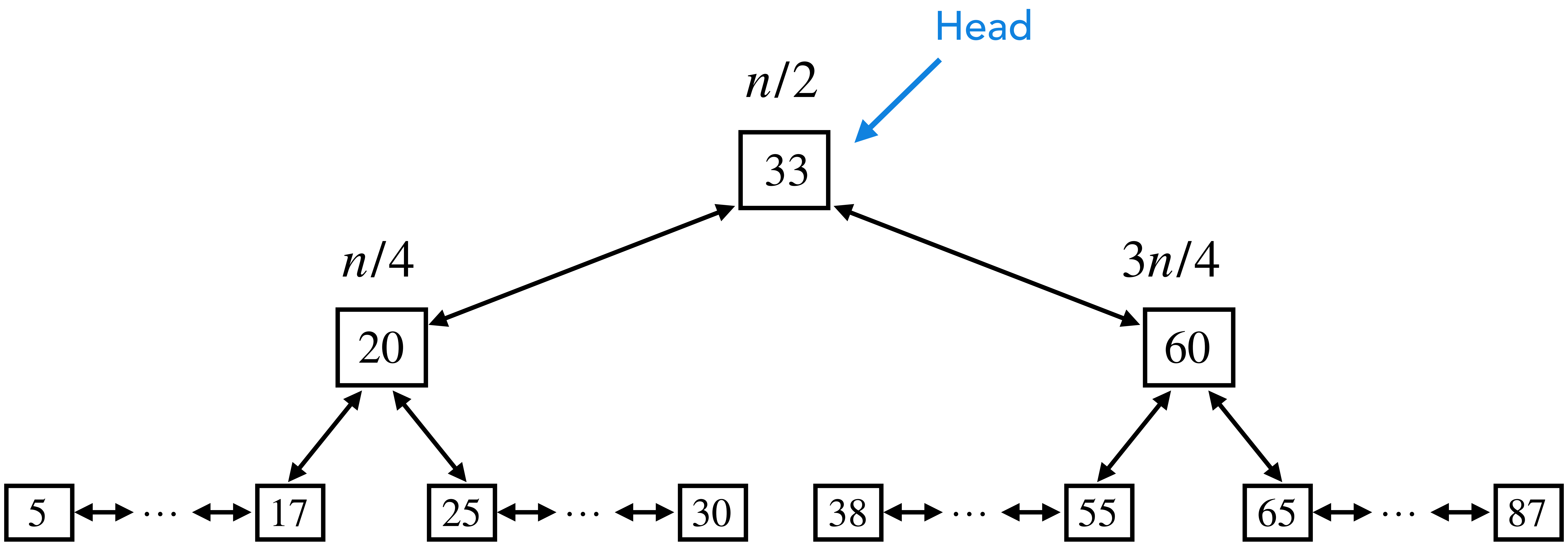
Reducing Search Time in Linked Lists



What can be done to find a node with around $n/4$ comparisons?

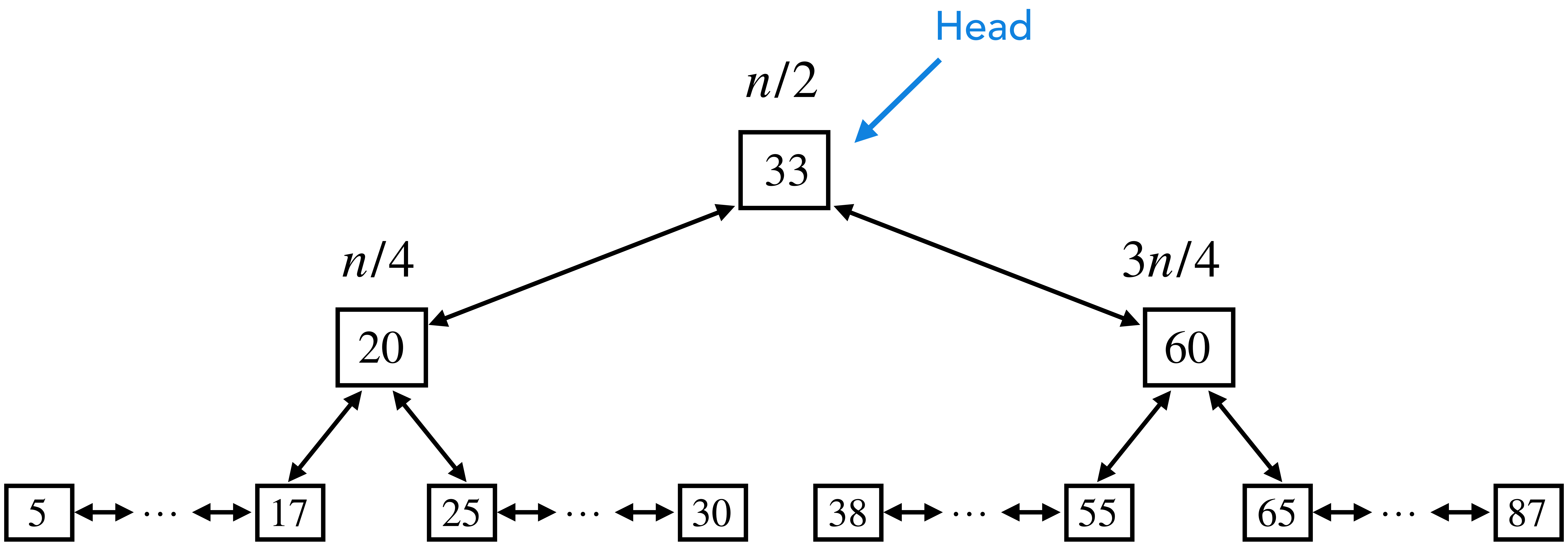
Source: *DSA Slides of Surender Baswana*

Reducing Search Time in Linked Lists



Source: DSA Slides of Surender Baswana

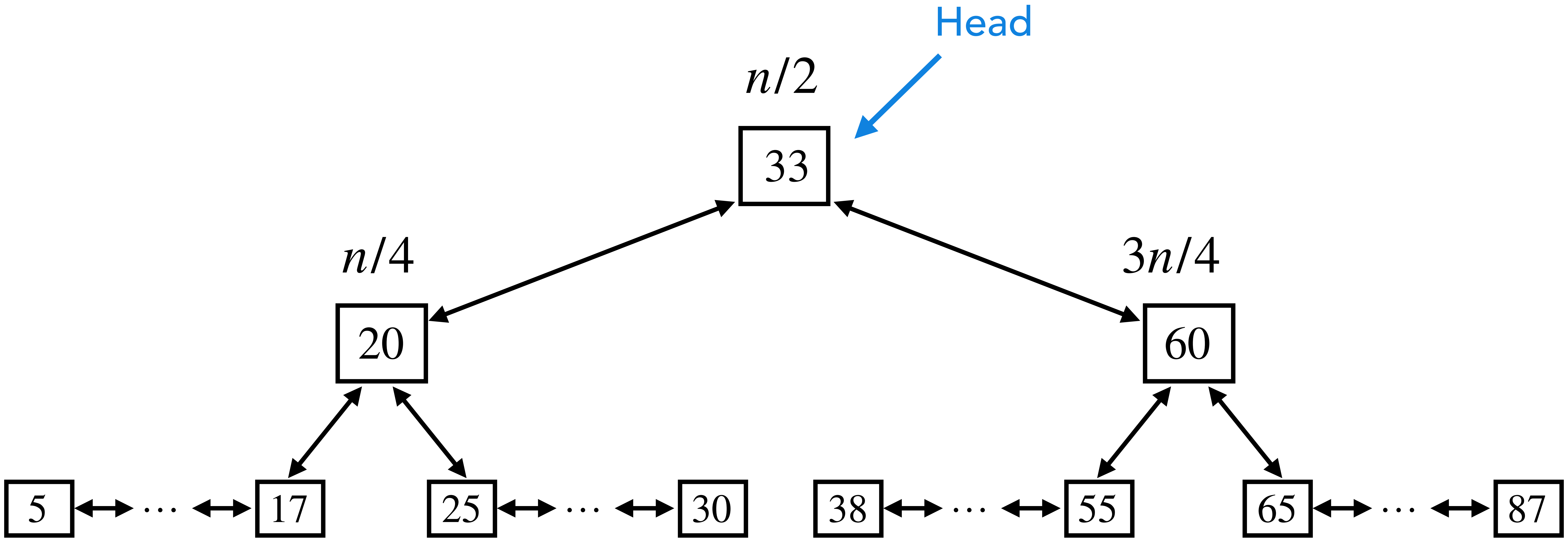
Reducing Search Time in Linked Lists



What will happen if we continue like this?

Source: *DSA Slides of Surender Baswana*

Reducing Search Time in Linked Lists

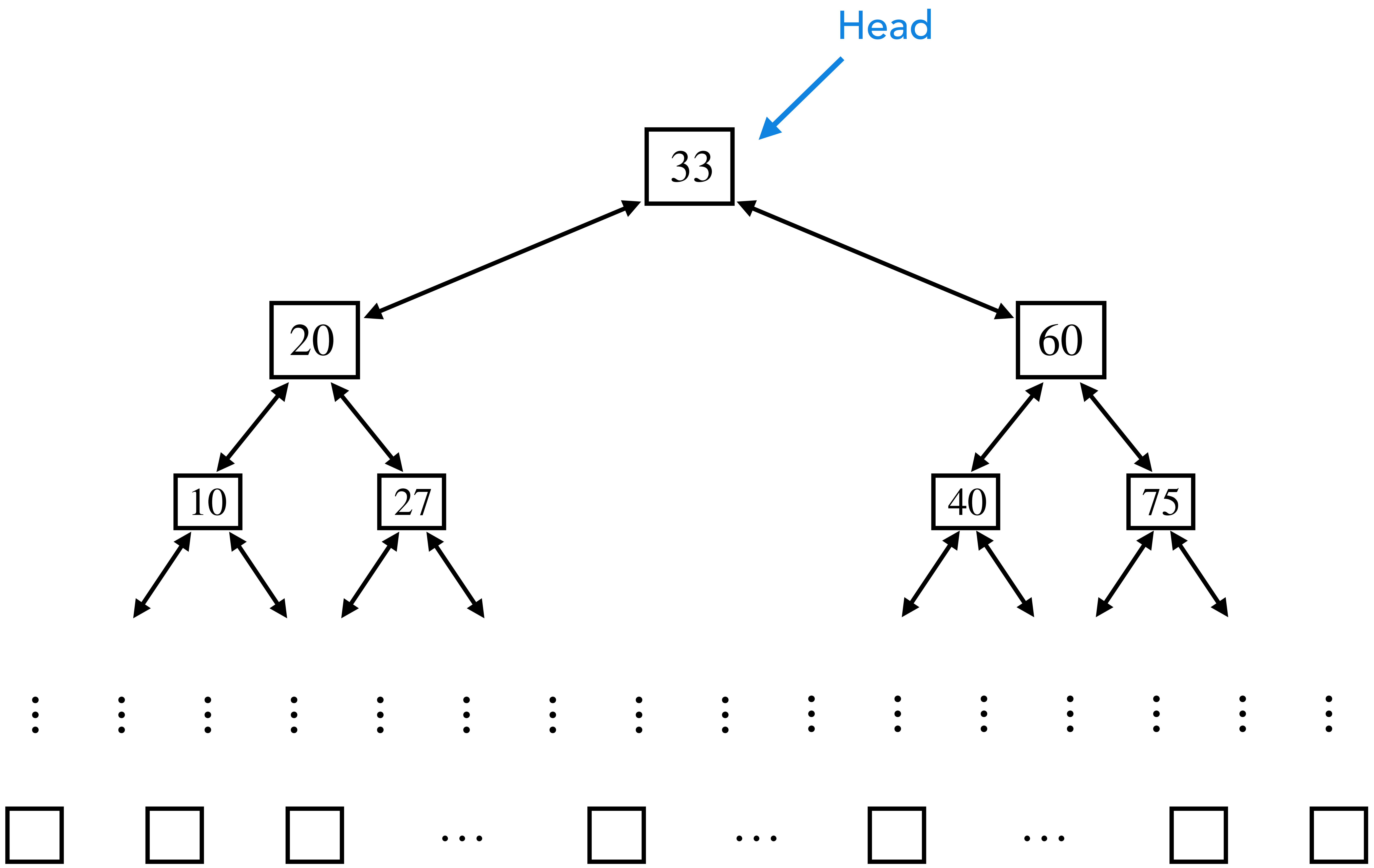


What will happen if we continue like this?

We get a binary search tree

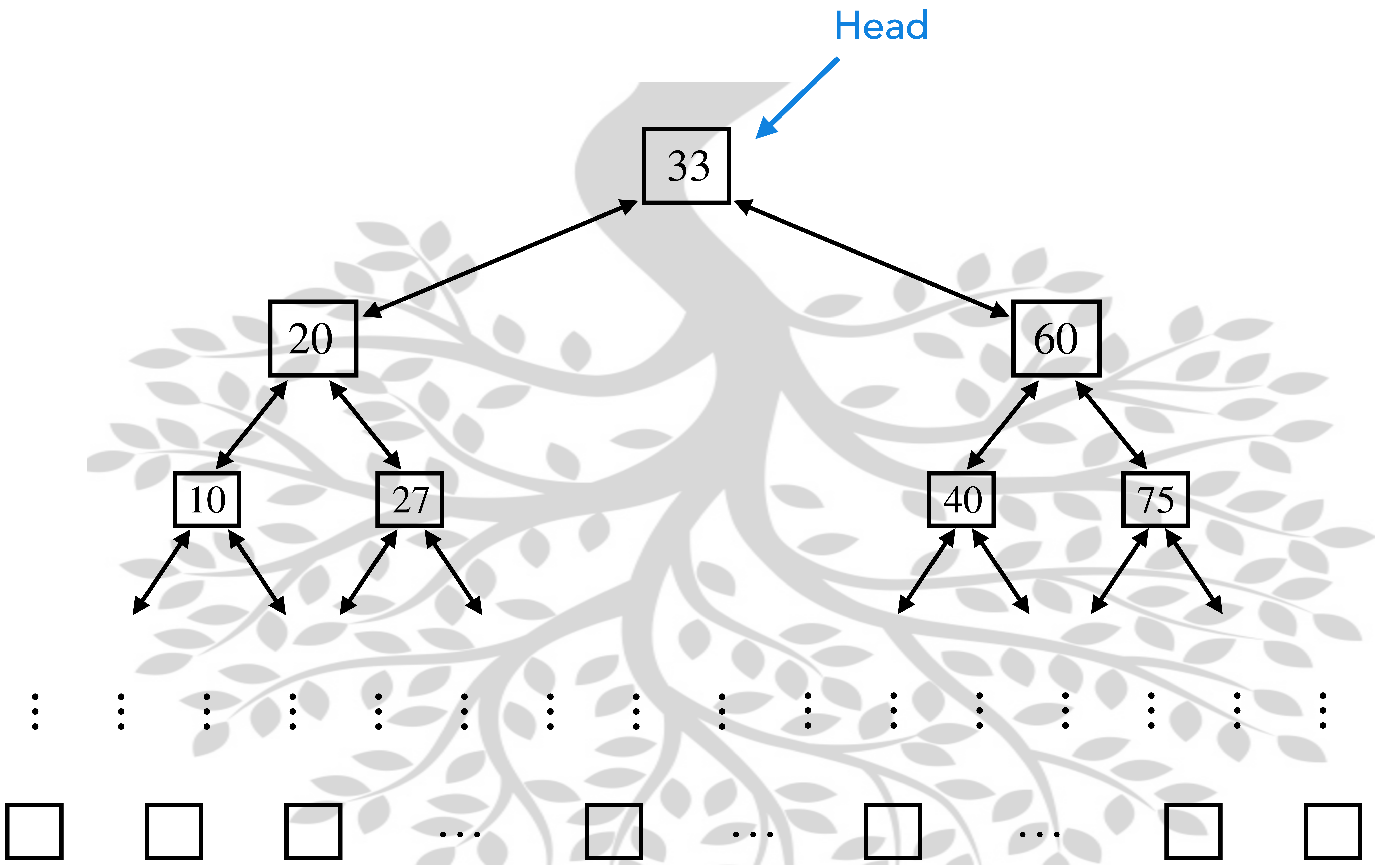
Source: *DSA Slides of Surender Baswana*

Binary Search Tree



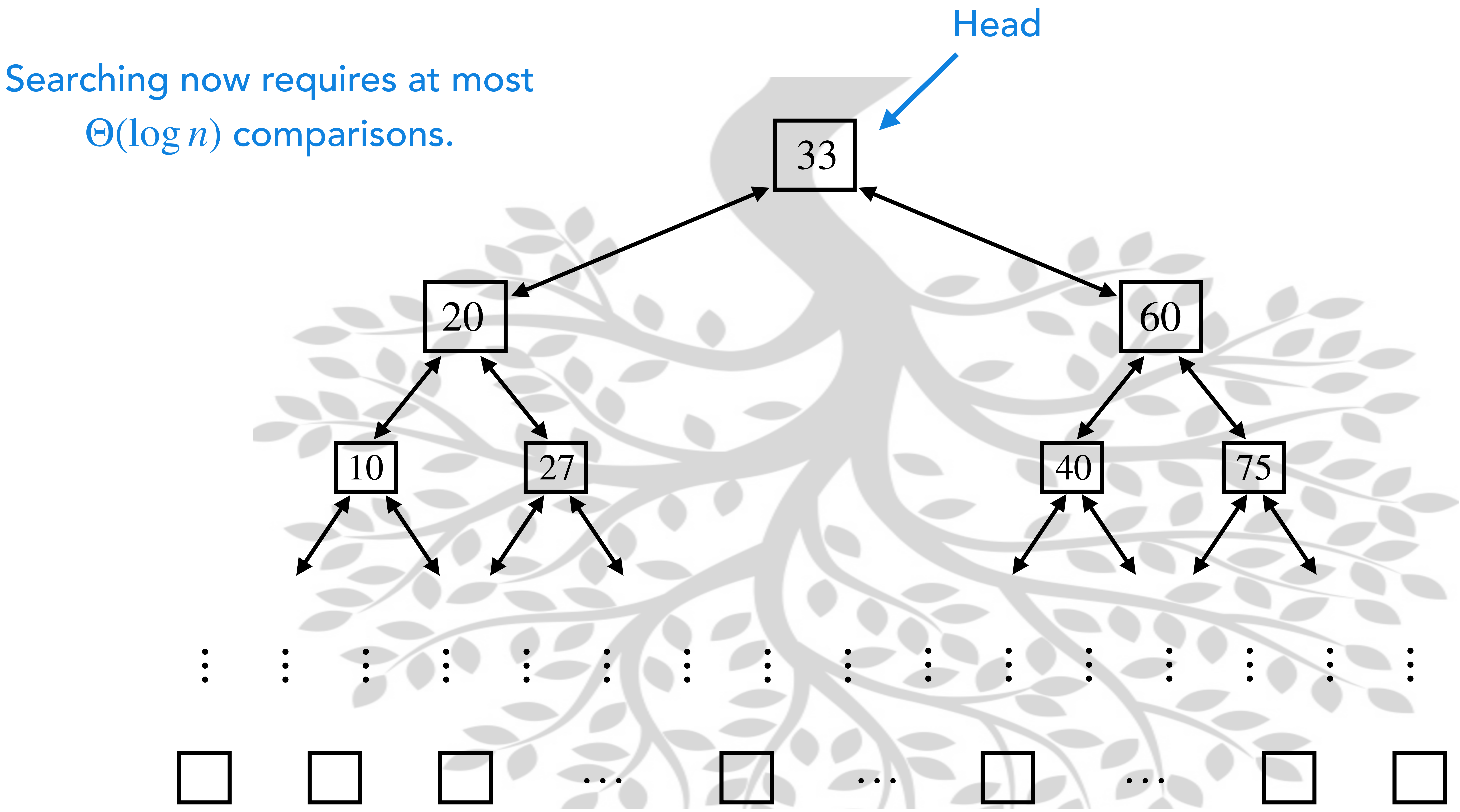
Source: *DSA Slides of Surender Baswana*

Binary Search Tree



Source: *DSA Slides of Surender Baswana*

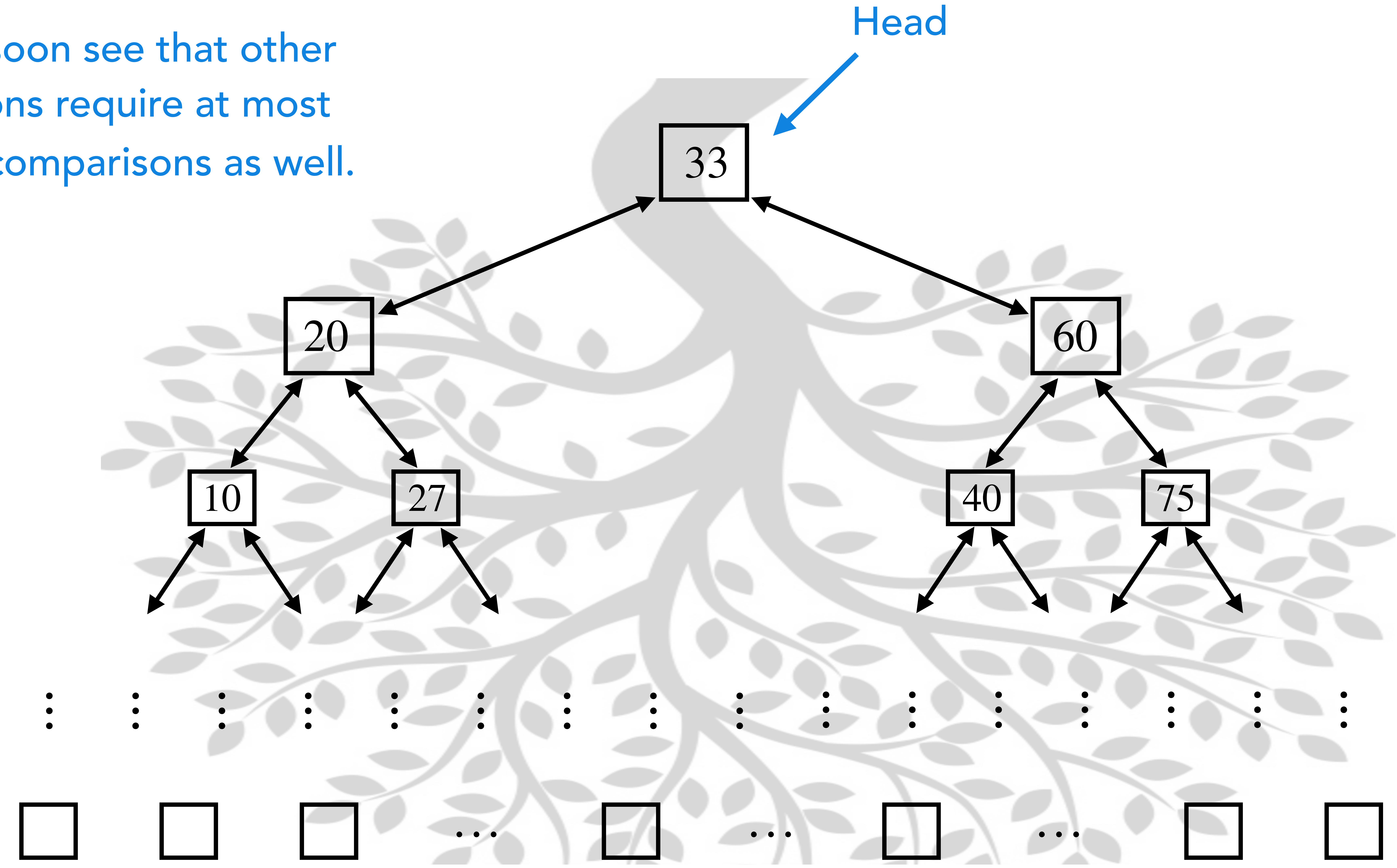
Binary Search Tree



Source: *DSA Slides of Surender Baswana*

Binary Search Tree

We will soon see that other operations require at most $\Theta(\log n)$ comparisons as well.



Source: *DSA Slides of Surender Baswana*

Comparison of Different Data Structures

Comparison of Different Data Structures

	Time required in Linked list implementation	Time required in BST implementation	Time required in Array implementation
Insert	$\Theta(n)$		
Search	$\Theta(n)$		
Delete	$\Theta(1)$		
Min/Max	$\Theta(1), \Theta(n)$		
Succ/Pred	$\Theta(1)$		

Comparison of Different Data Structures

	Time required in Linked list implementation	Time required in BST implementation	Time required in Array implementation
Insert	$\Theta(n)$	$\Theta(h)$	
Search	$\Theta(n)$	$\Theta(h)$	
Delete	$\Theta(1)$	$\Theta(h)$	
Min/Max	$\Theta(1), \Theta(n)$	$\Theta(h)$	
Succ/Pred	$\Theta(1)$	$\Theta(h)$	

Comparison of Different Data Structures

	Time required in Linked list implementation	Time required in BST implementation	Time required in Array implementation
Insert	$\Theta(n)$	$\Theta(h)$	
Search	$\Theta(n)$	$\Theta(h)$	
Delete	$\Theta(1)$	$\Theta(h)$	
Min/Max	$\Theta(1), \Theta(n)$	$\Theta(h)$	
Succ/Pred	$\Theta(1)$	$\Theta(h)$	

h is the height of the tree

Comparison of Different Data Structures

	Time required in Linked list implementation	Time required in BST implementation	Time required in Array implementation
Insert	$\Theta(n)$	$\Theta(h)$	??
Search	$\Theta(n)$	$\Theta(h)$	??
Delete	$\Theta(1)$	$\Theta(h)$	??
Min/Max	$\Theta(1), \Theta(n)$	$\Theta(h)$	??
Succ/Pred	$\Theta(1)$	$\Theta(h)$	??

h is the height of the tree

Comparison of Different Data Structures

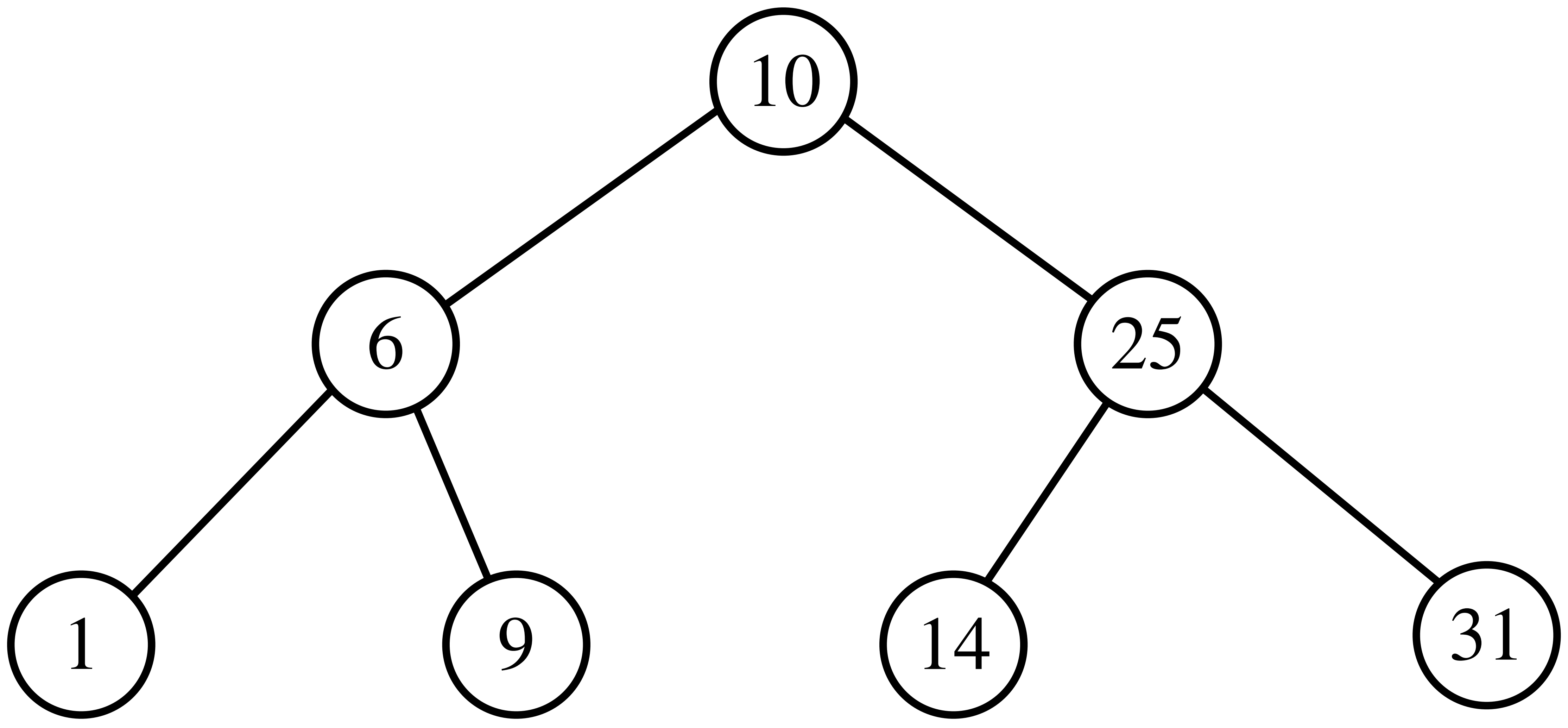
	Time required in Linked list implementation	Time required in BST implementation	Time required in Array implementation
Insert	$\Theta(n)$	$\Theta(h)$	??
Search	$\Theta(n)$	$\Theta(h)$	??
Delete	$\Theta(1)$	$\Theta(h)$	??
Min/Max	$\Theta(1), \Theta(n)$	$\Theta(h)$	??
Succ/Pred	$\Theta(1)$	$\Theta(h)$	??

DIY

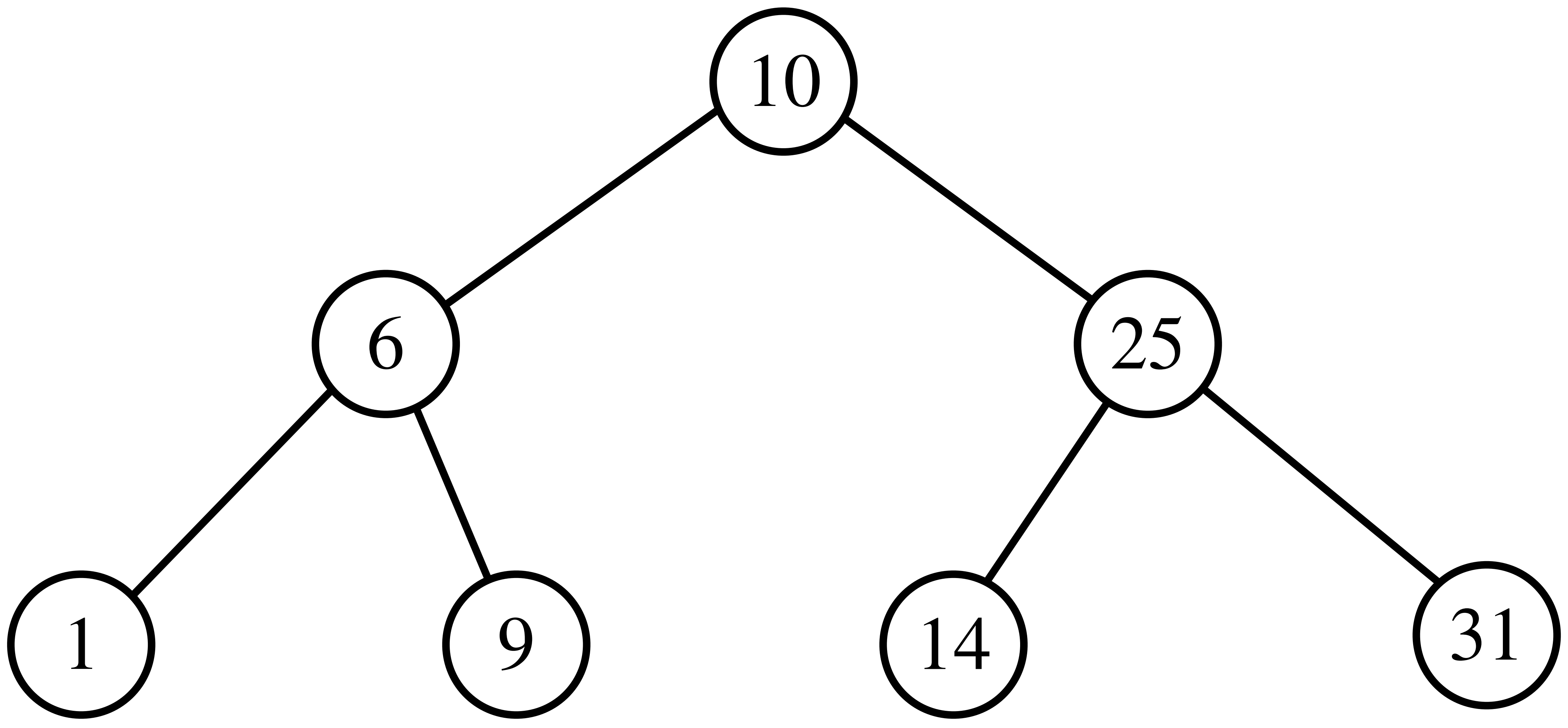
h is the height of the tree

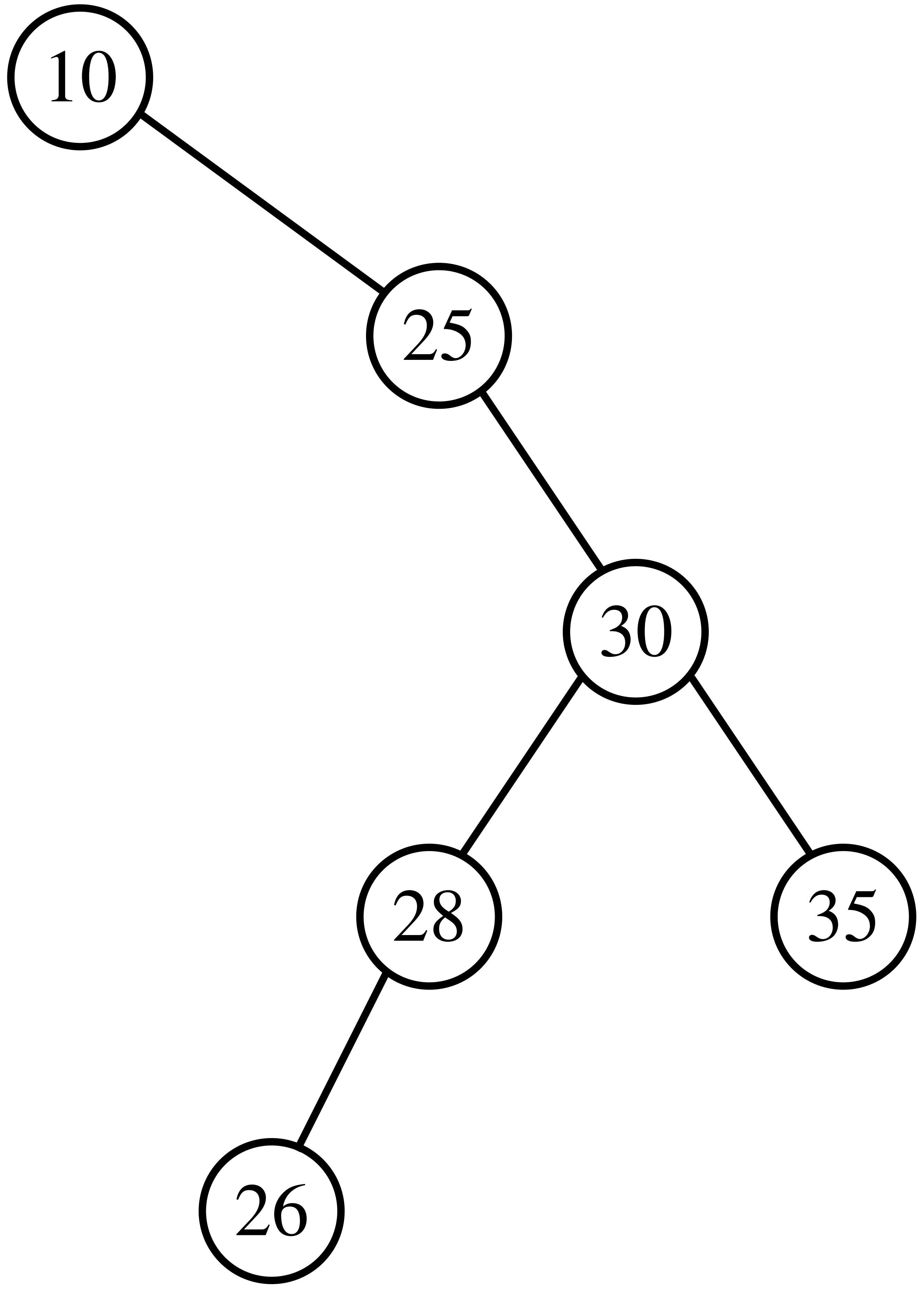
How does a BST look like?

How does a BST look like?



How does a BST look like?





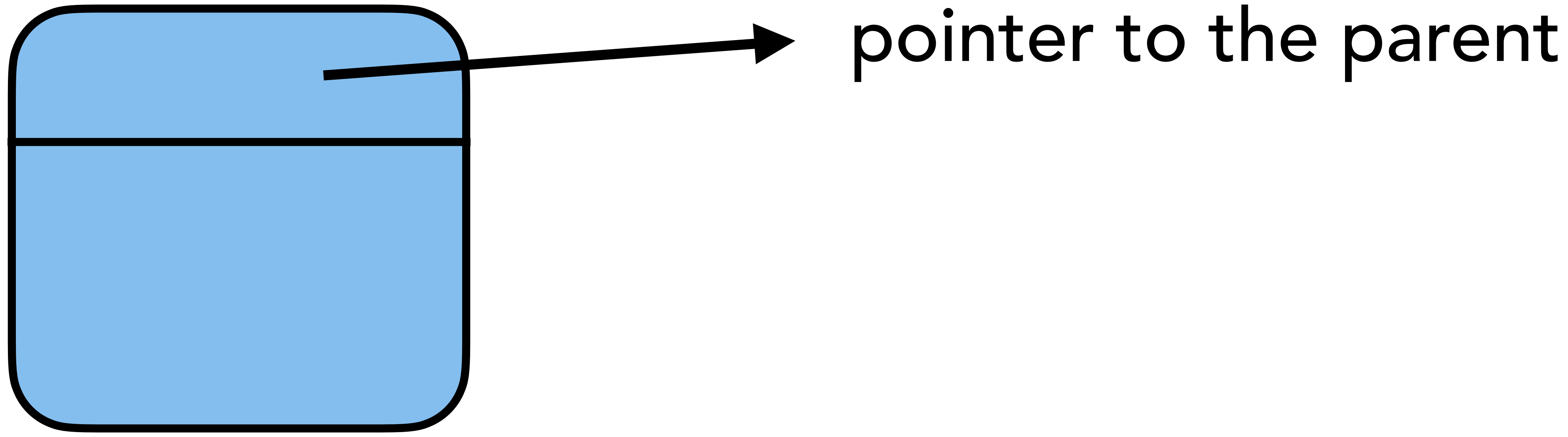
What is a BST?

What is a BST?

- A binary search tree is a collection of nodes of the following type:

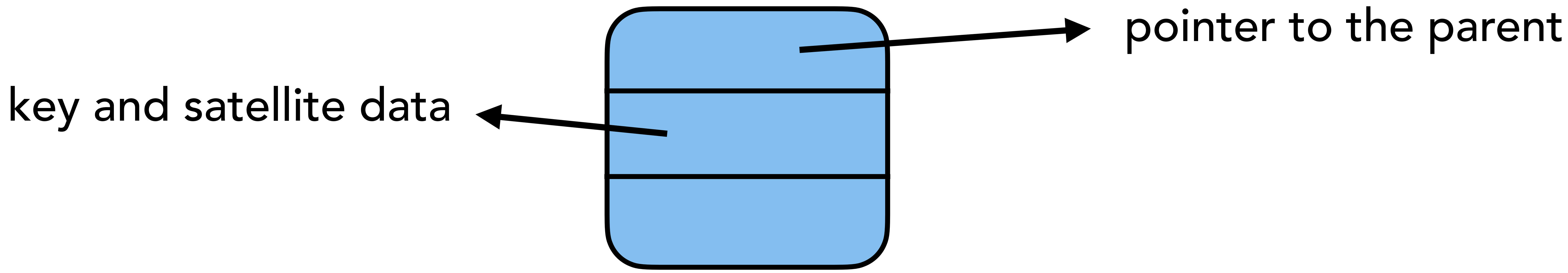
What is a BST?

- A binary search tree is a collection of nodes of the following type:



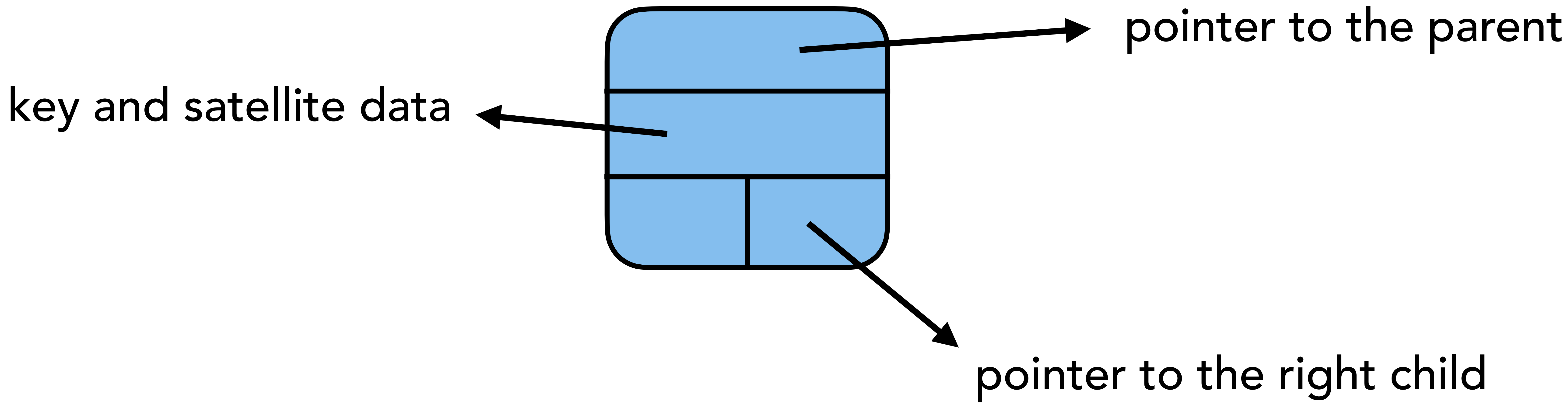
What is a BST?

- A binary search tree is a collection of nodes of the following type:



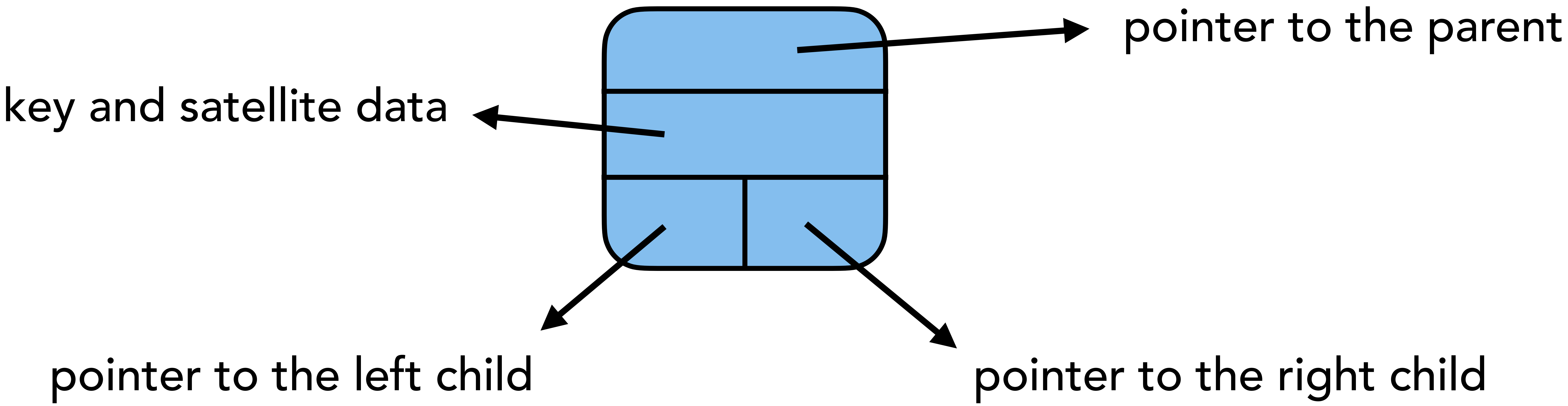
What is a BST?

- A binary search tree is a collection of nodes of the following type:



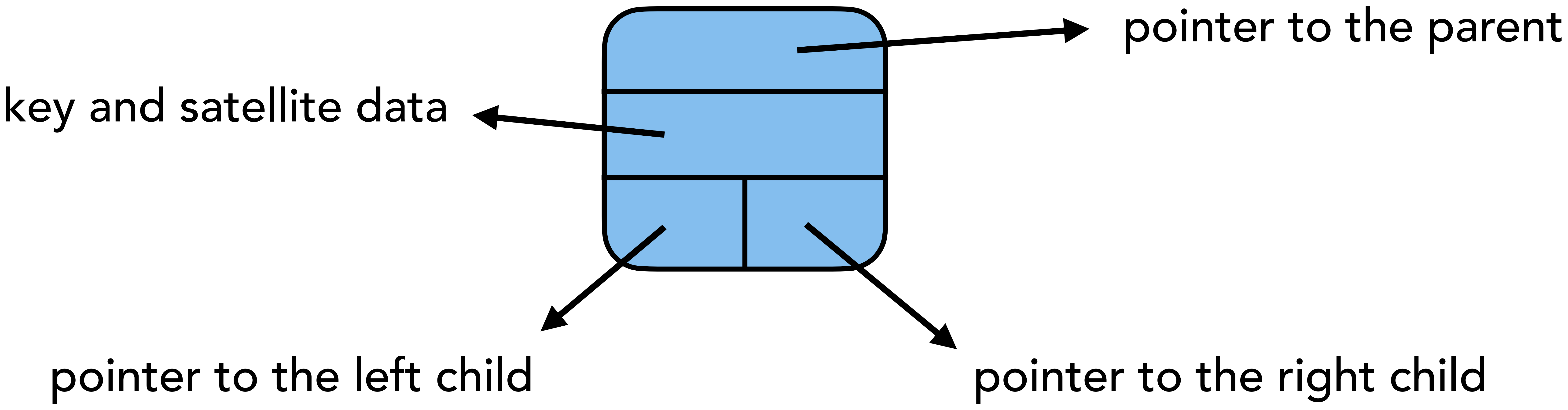
What is a BST?

- A binary search tree is a collection of nodes of the following type:



What is a BST?

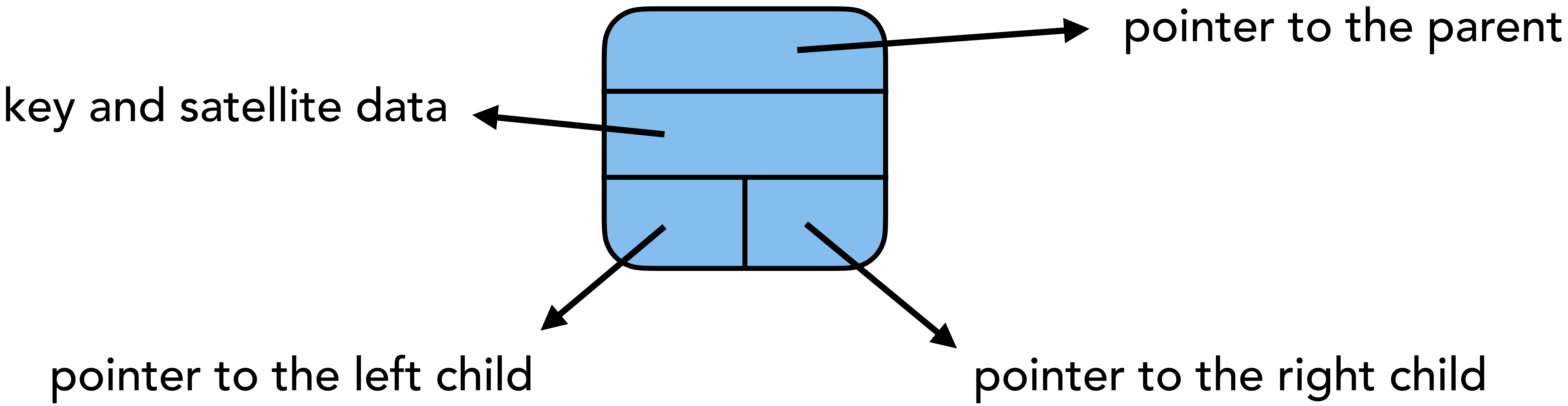
- A binary search tree is a collection of nodes of the following type:



- Every tree has a root. It's the only node without the parent.

What is a BST?

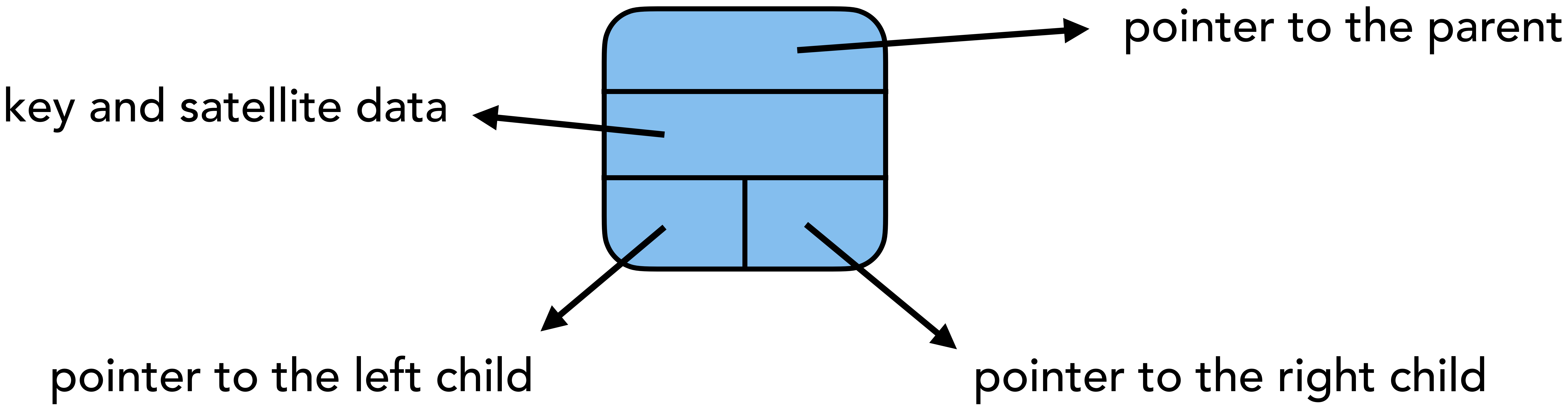
- A binary search tree is a collection of nodes of the following type:



- Every tree has a root. It's the only node without the parent.
- There is a path from every node to the root.

What is a BST?

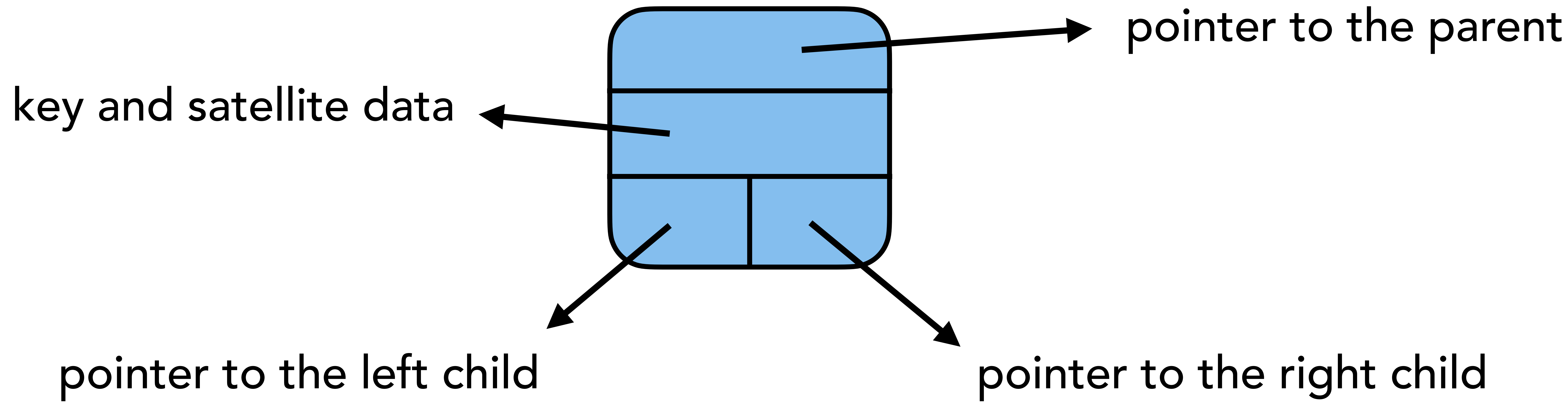
- A binary search tree is a collection of nodes of the following type:



- Every tree has a root. It's the only node without the parent.
- There is a path from every node to the root.
- BST property:

What is a BST?

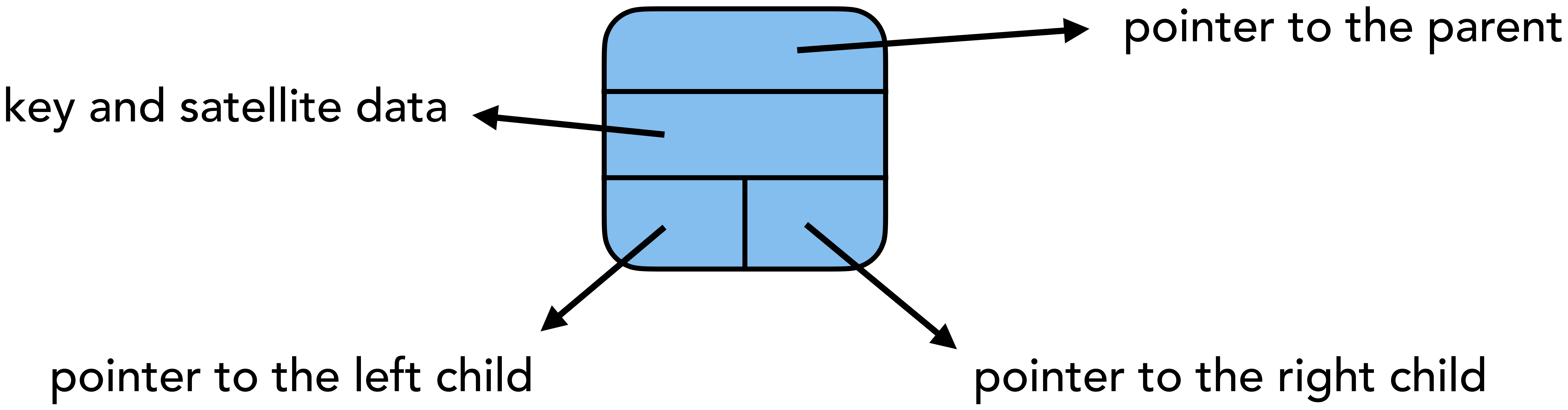
- A binary search tree is a collection of nodes of the following type:



- Every tree has a root. It's the only node without the parent.
- There is a path from every node to the root.
- BST property: Let x be a node in a BST

What is a BST?

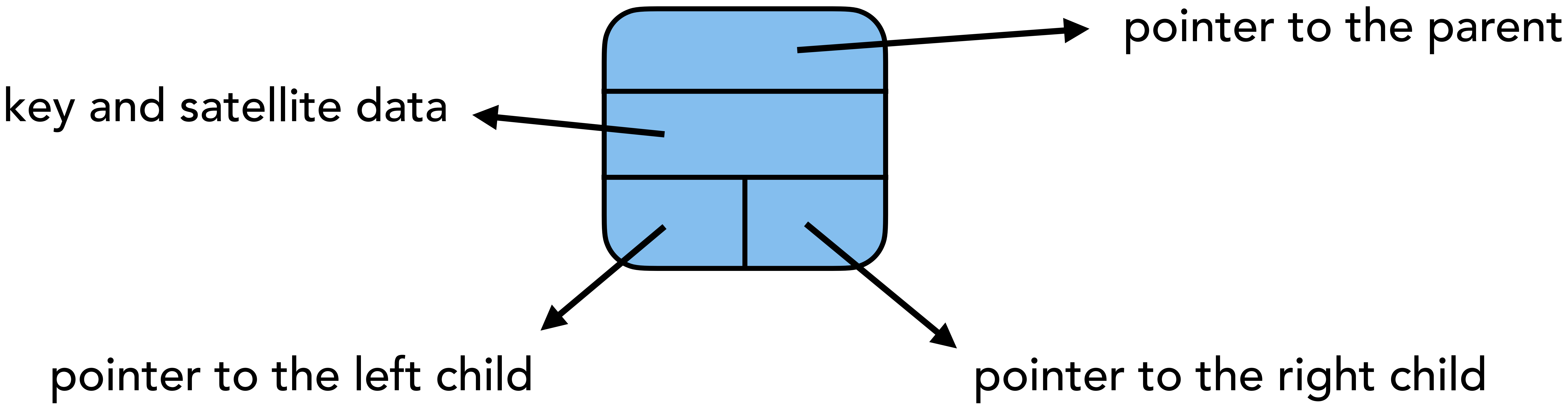
- A binary search tree is a collection of nodes of the following type:



- Every tree has a root. It's the only node without the parent.
- There is a path from every node to the root.
- BST property: Let x be a node in a BST and y, z be the nodes in its left, right subtree, resp.

What is a BST?

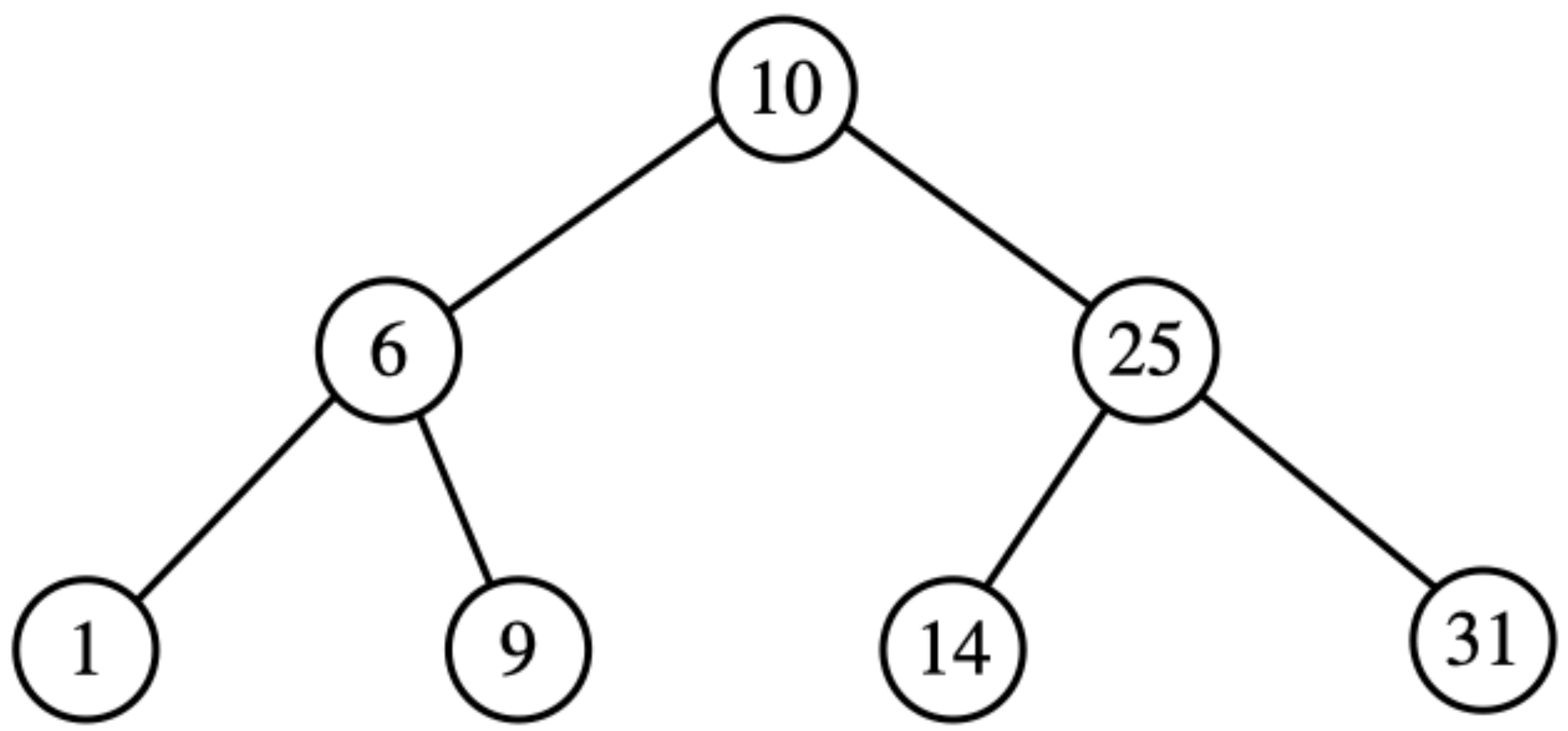
- A binary search tree is a collection of nodes of the following type:



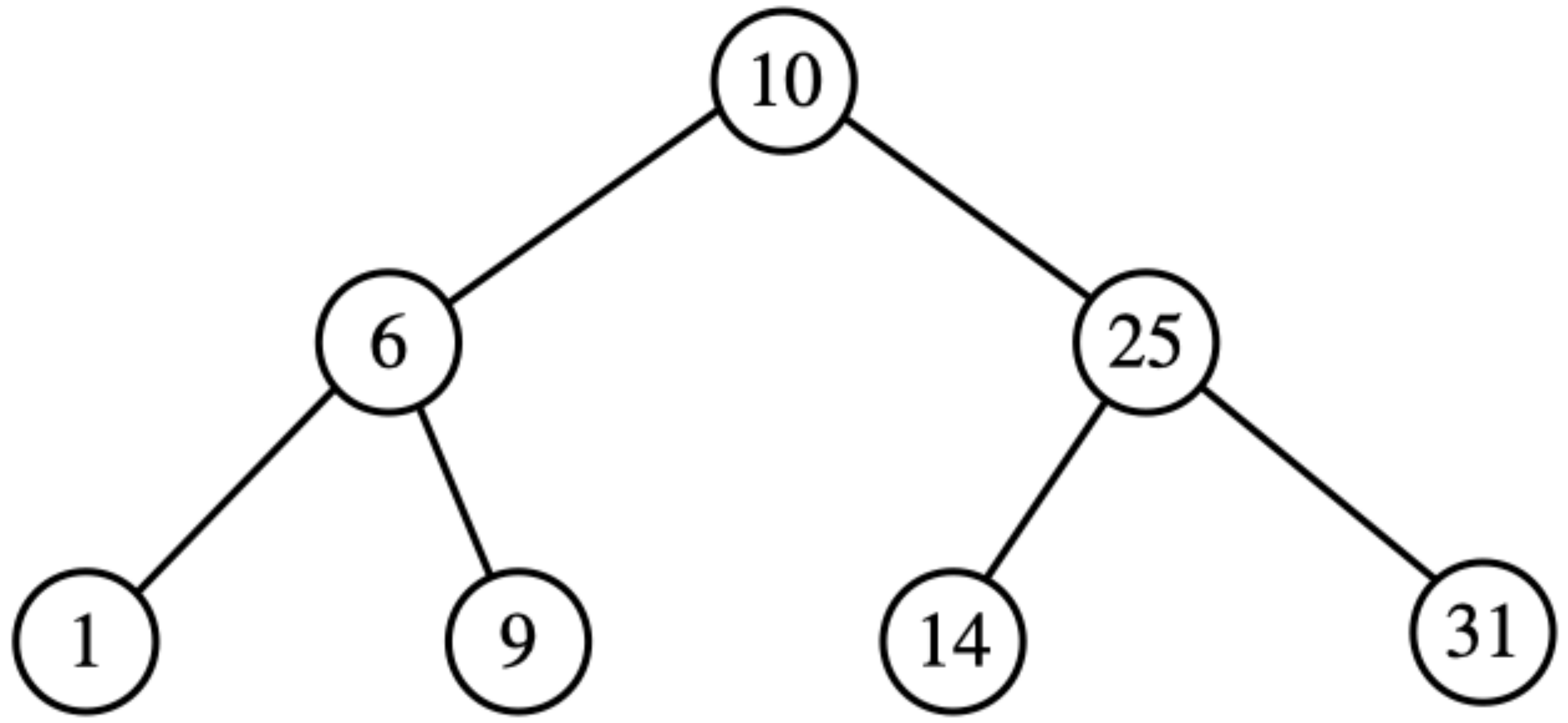
- Every tree has a root. It's the only node without the parent.
- There is a path from every node to the root.
- BST property: Let x be a node in a BST and y, z be the nodes in its left, right subtree, resp. Then, $y.key \leq x.key \leq z.key$.

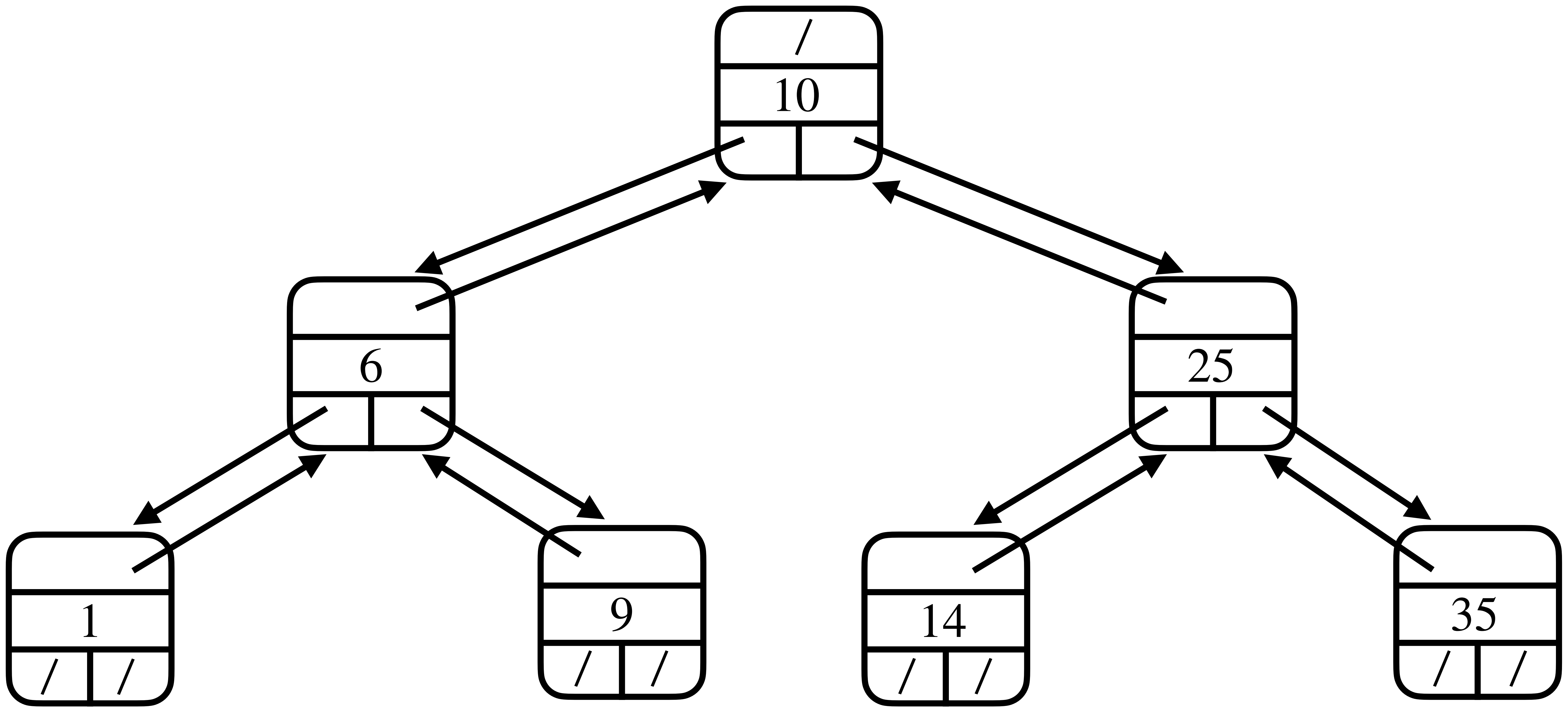
What is a BST?

What is a BST?

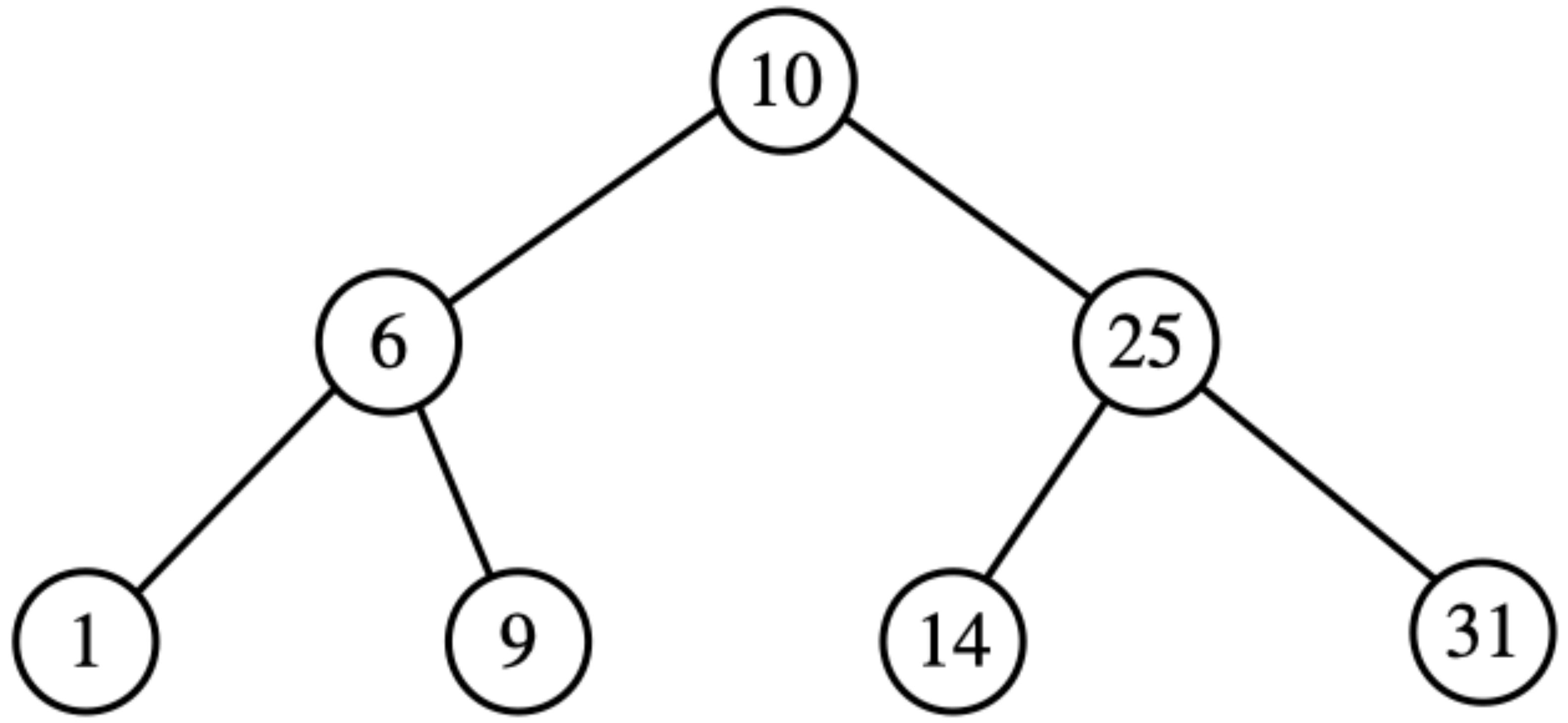


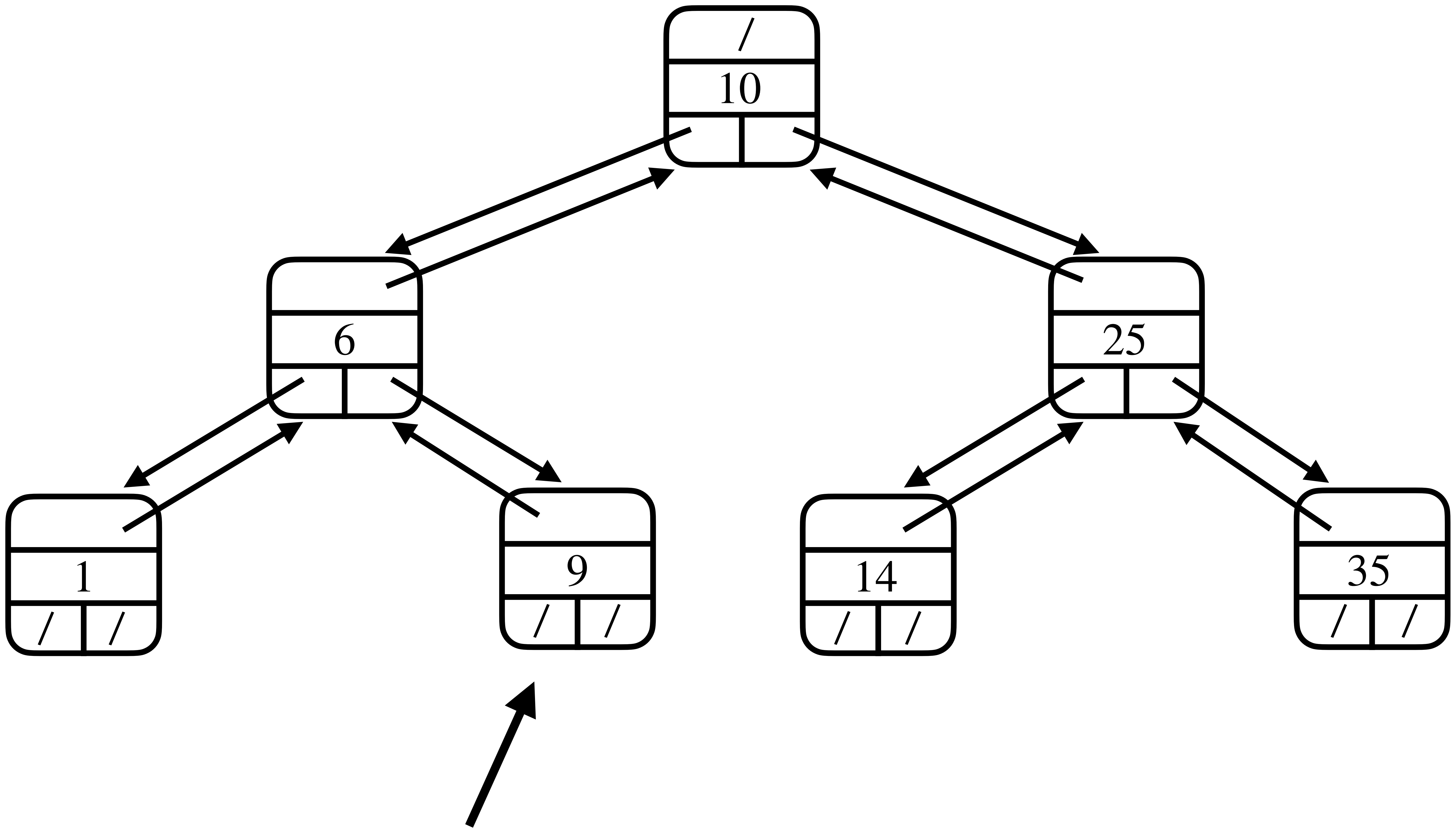
What is a BST?





What is a BST?

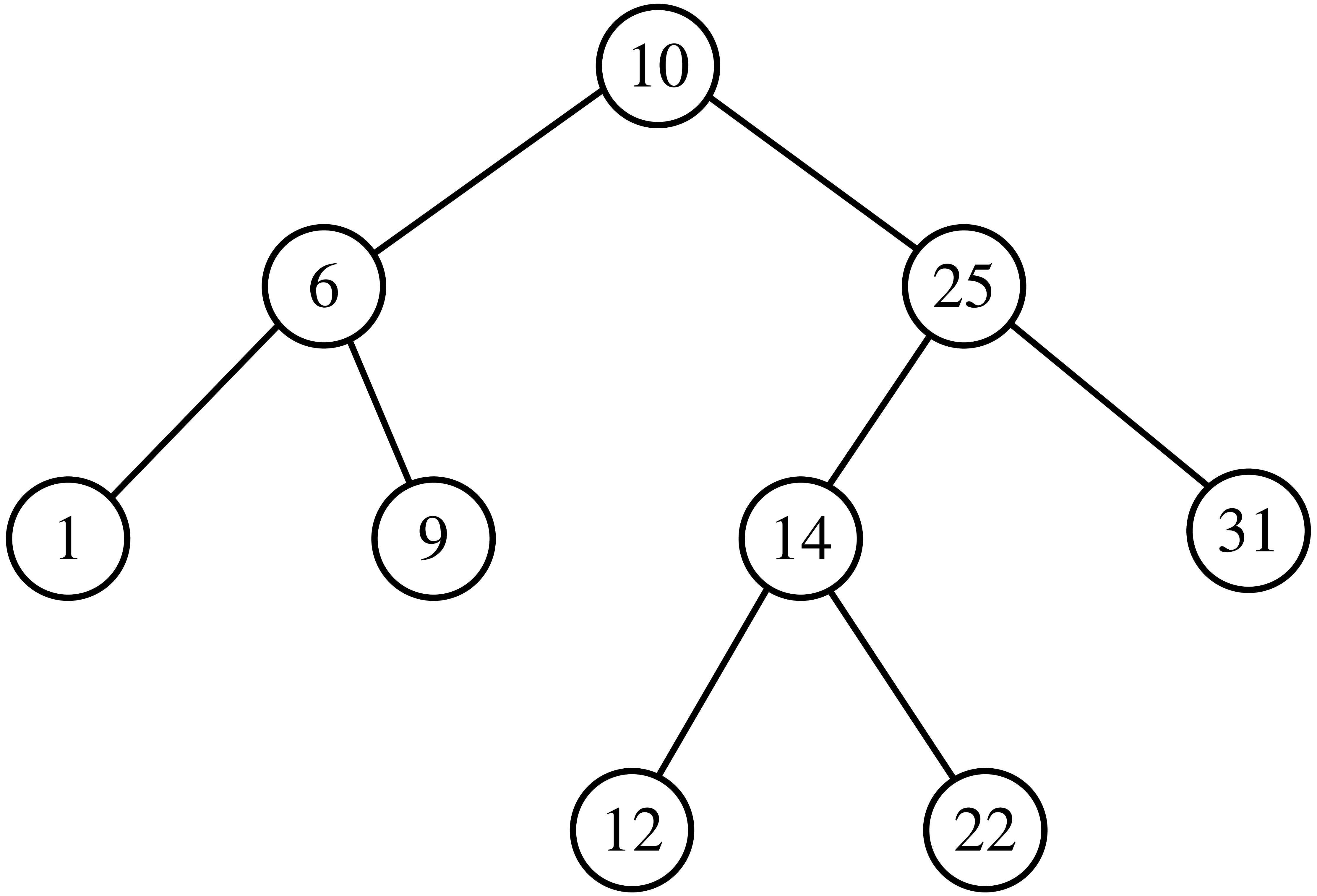




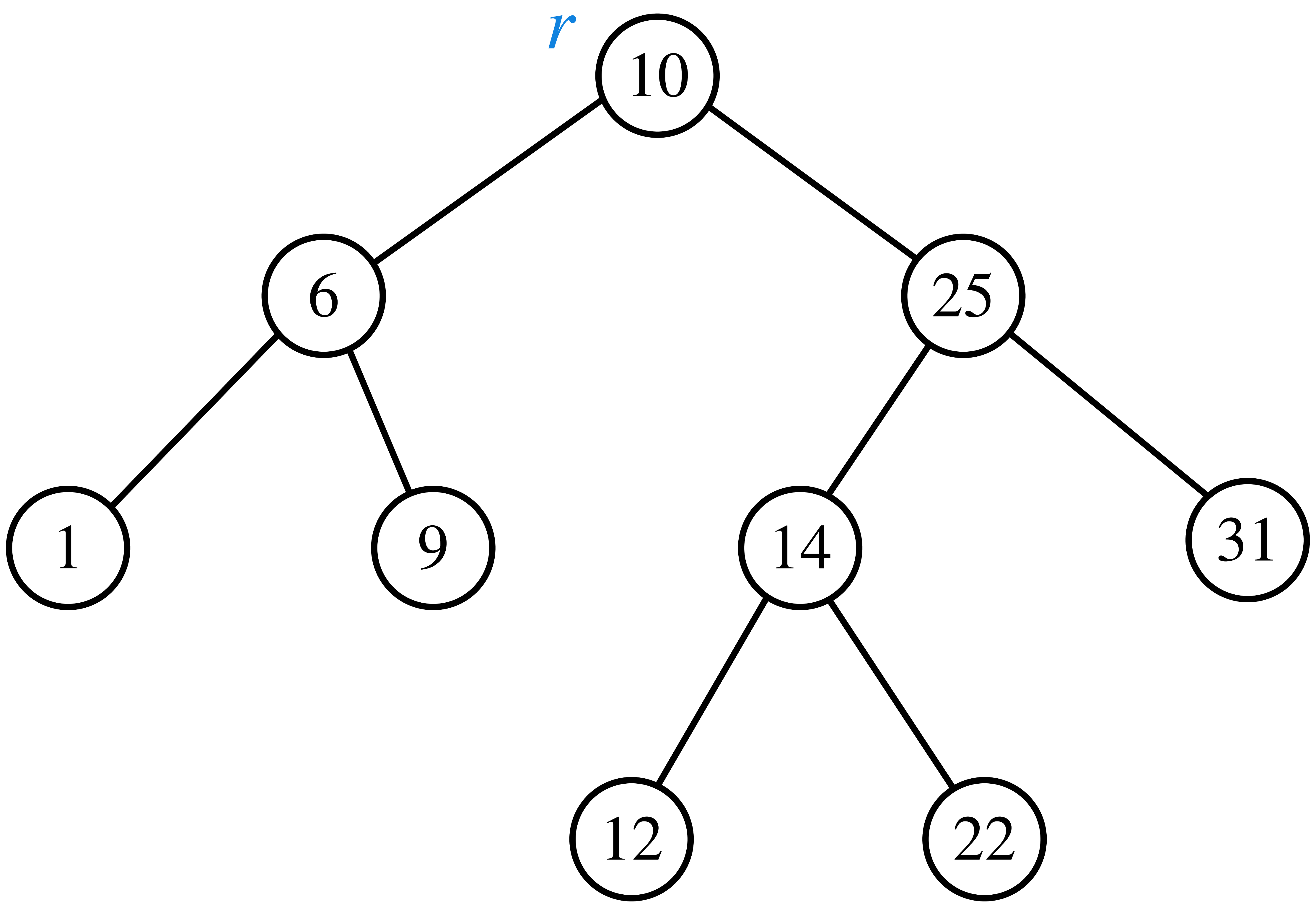
NIL values in the absence of parent, left or right child.

BST: Basic Terminology

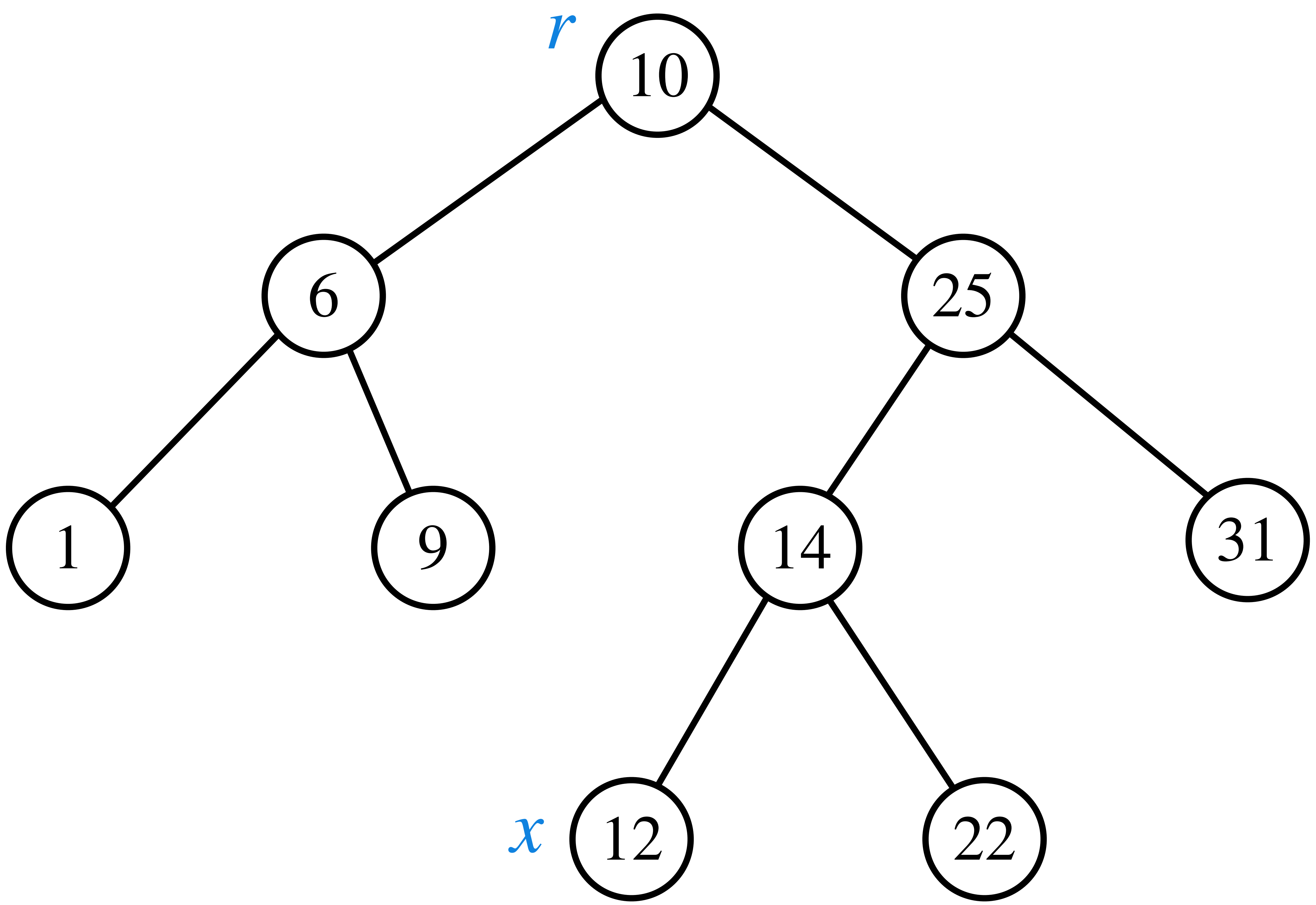
BST: Basic Terminology



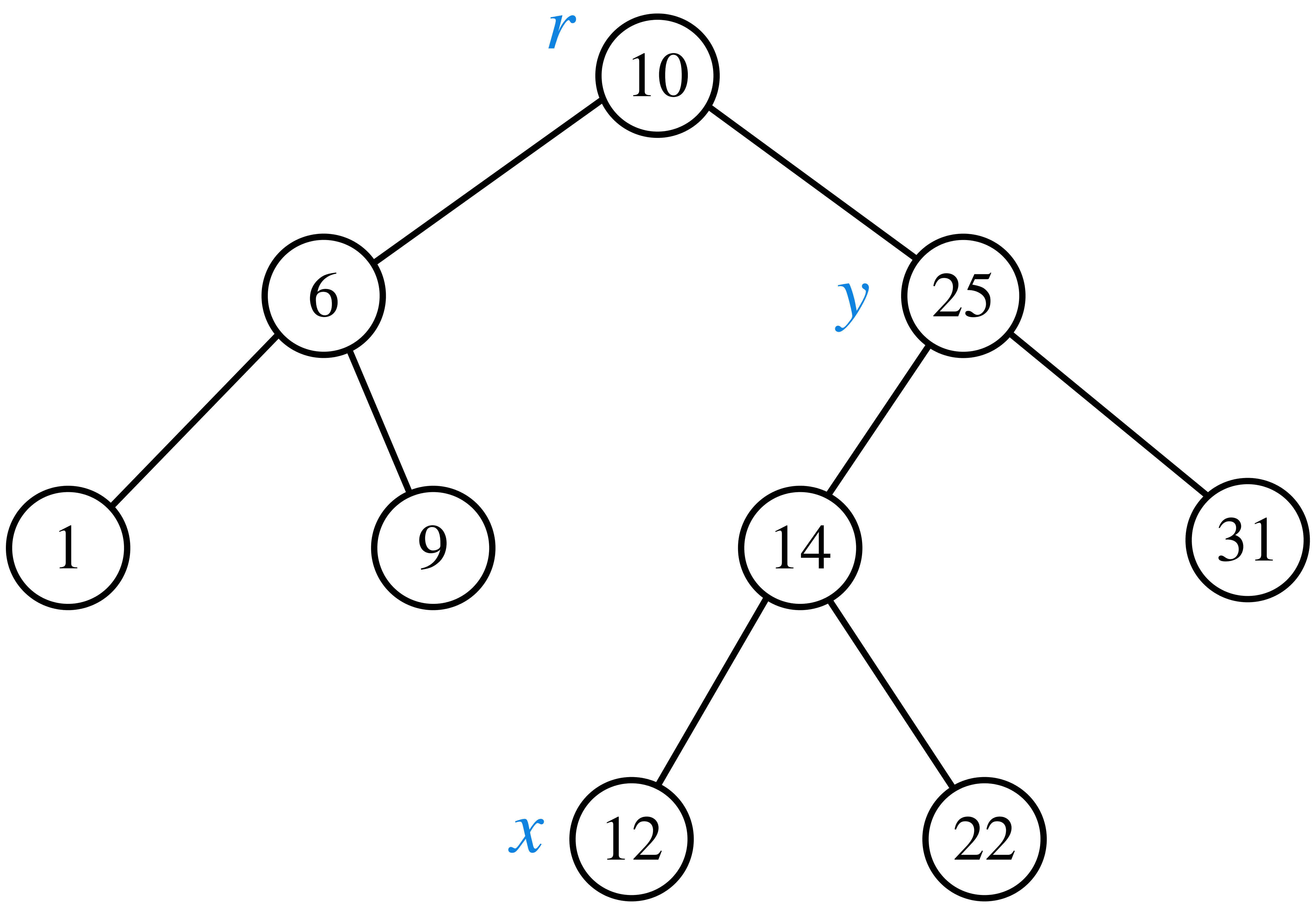
BST: Basic Terminology



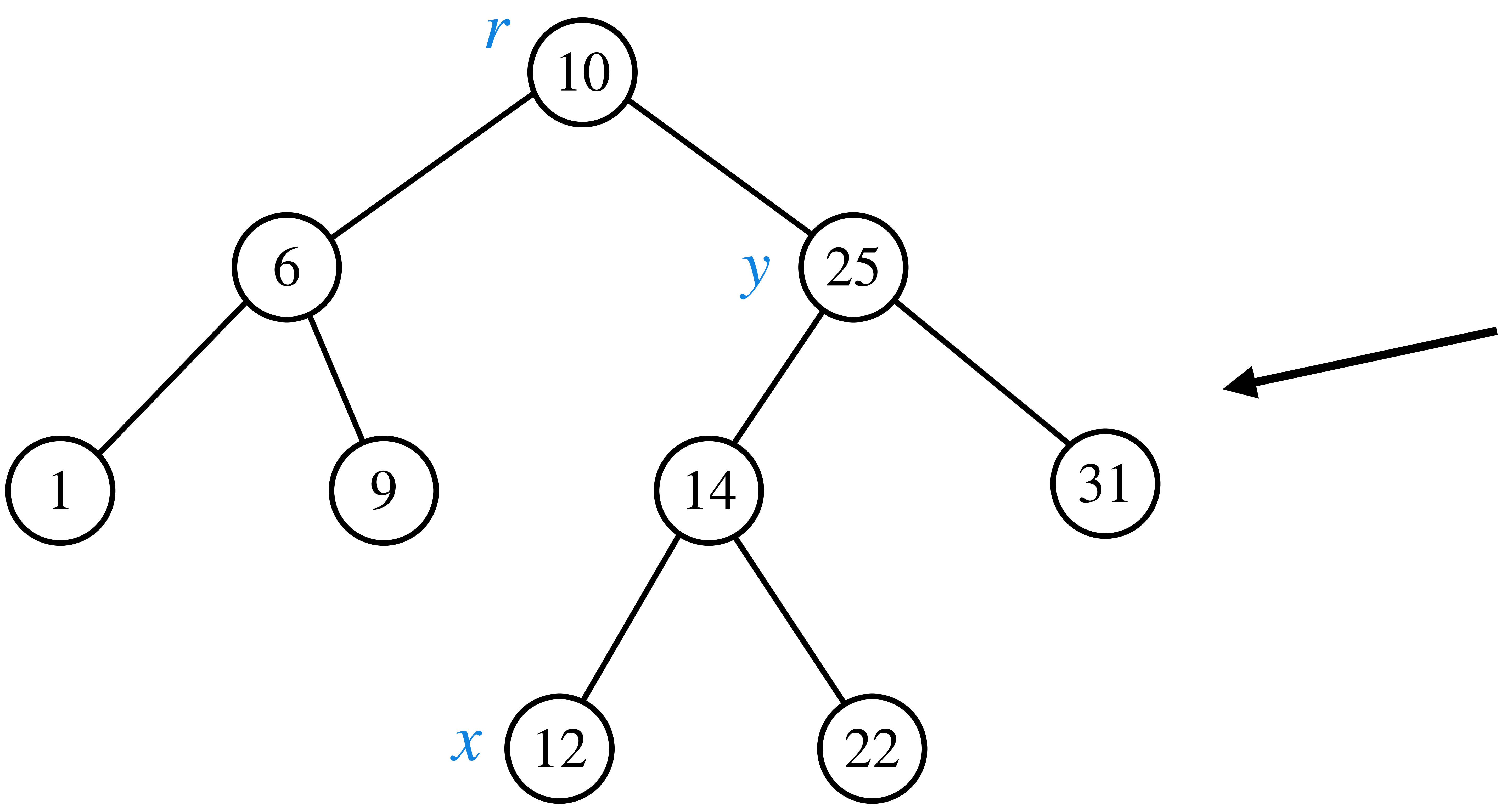
BST: Basic Terminology



BST: Basic Terminology

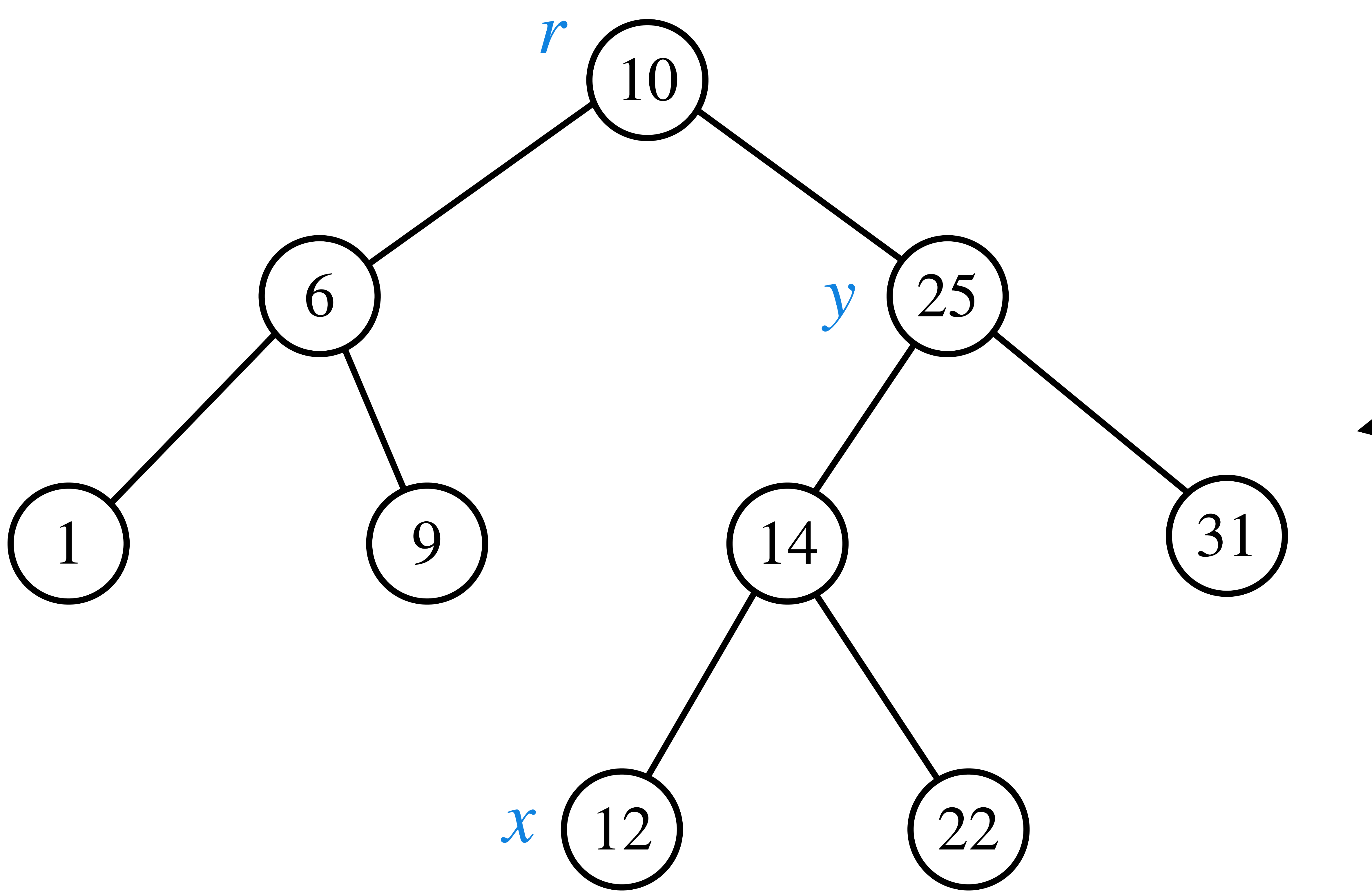


BST: Basic Terminology



y is an ancestor of x,

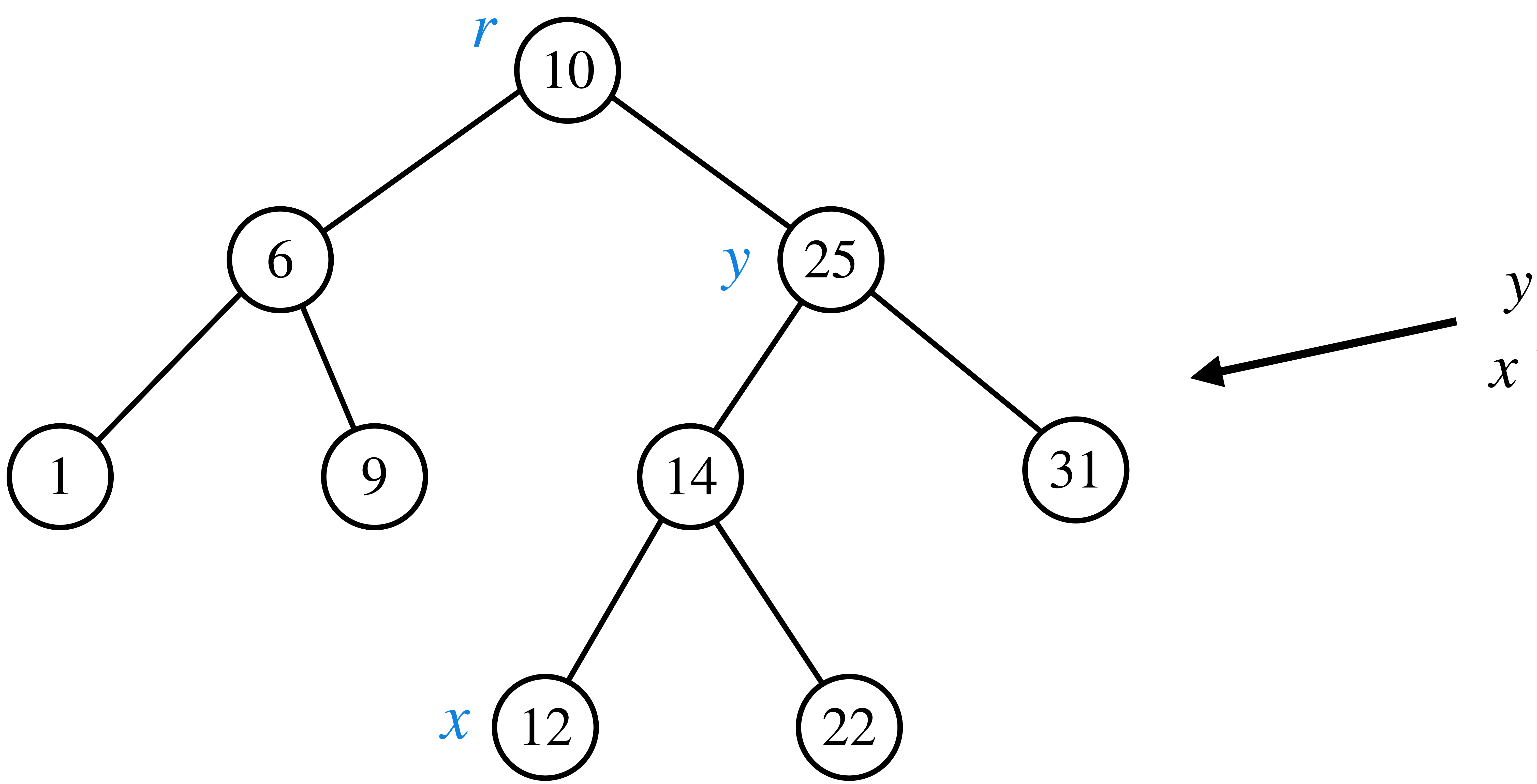
BST: Basic Terminology



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

BST: Basic Terminology

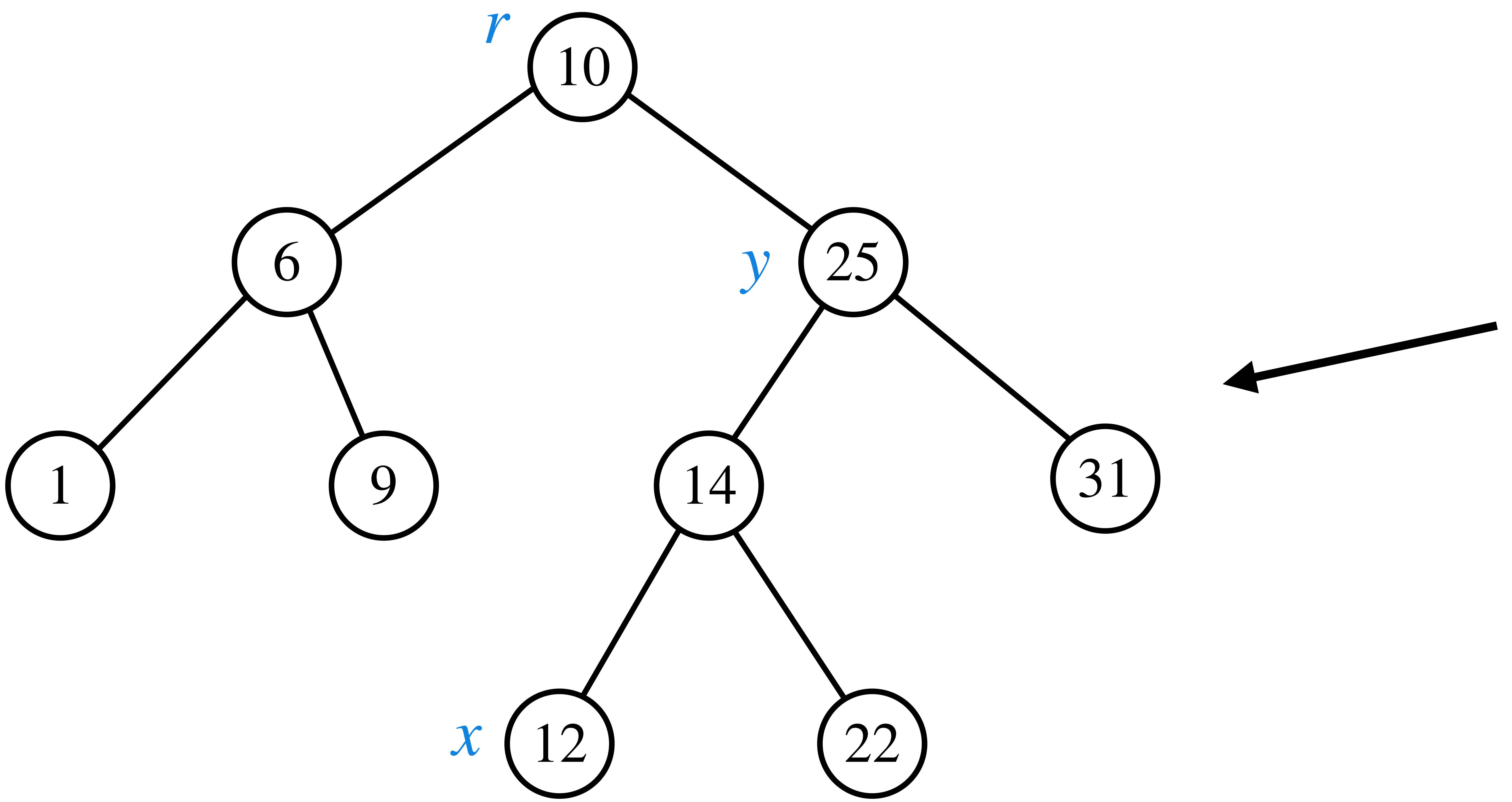
Defn: Let x be a node in a tree T with root r.



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

BST: Basic Terminology

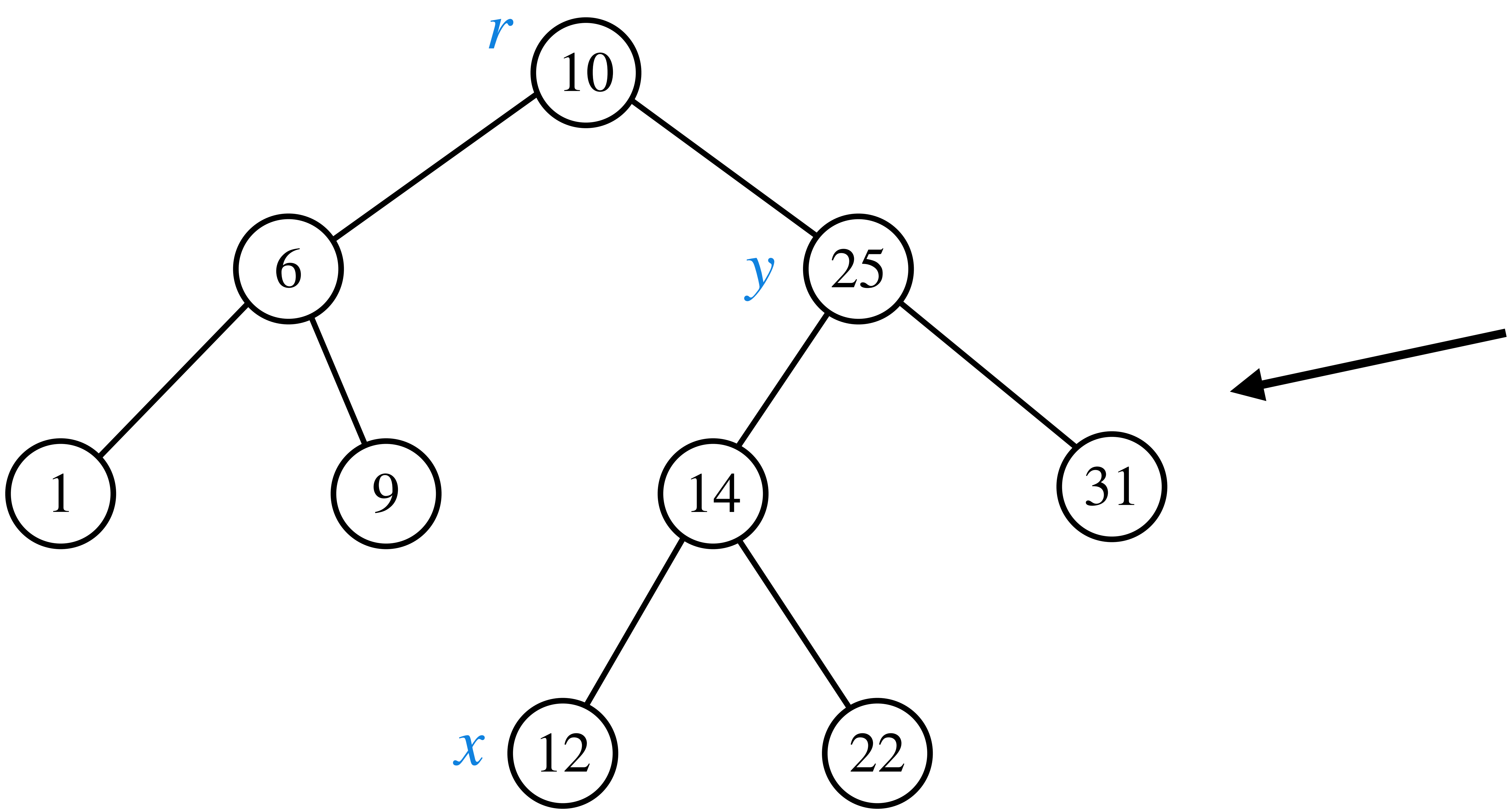
Defn: Let x be a node in a tree T with root r. Then any node y on the unique path from r to x is



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

BST: Basic Terminology

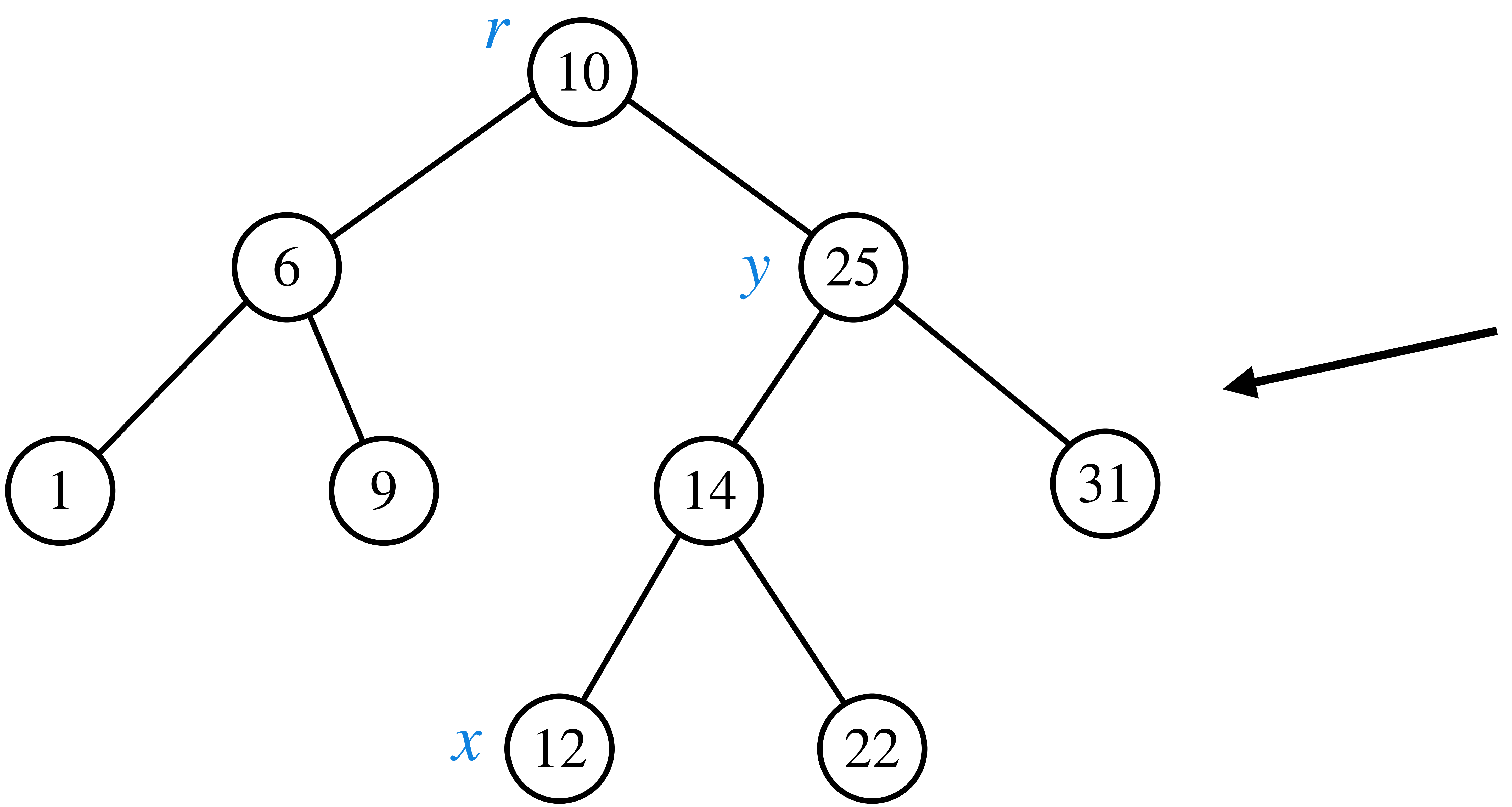
Defn: Let x be a node in a tree T with root r. Then any node y on the unique path from r to x is called an ancestor of x



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

BST: Basic Terminology

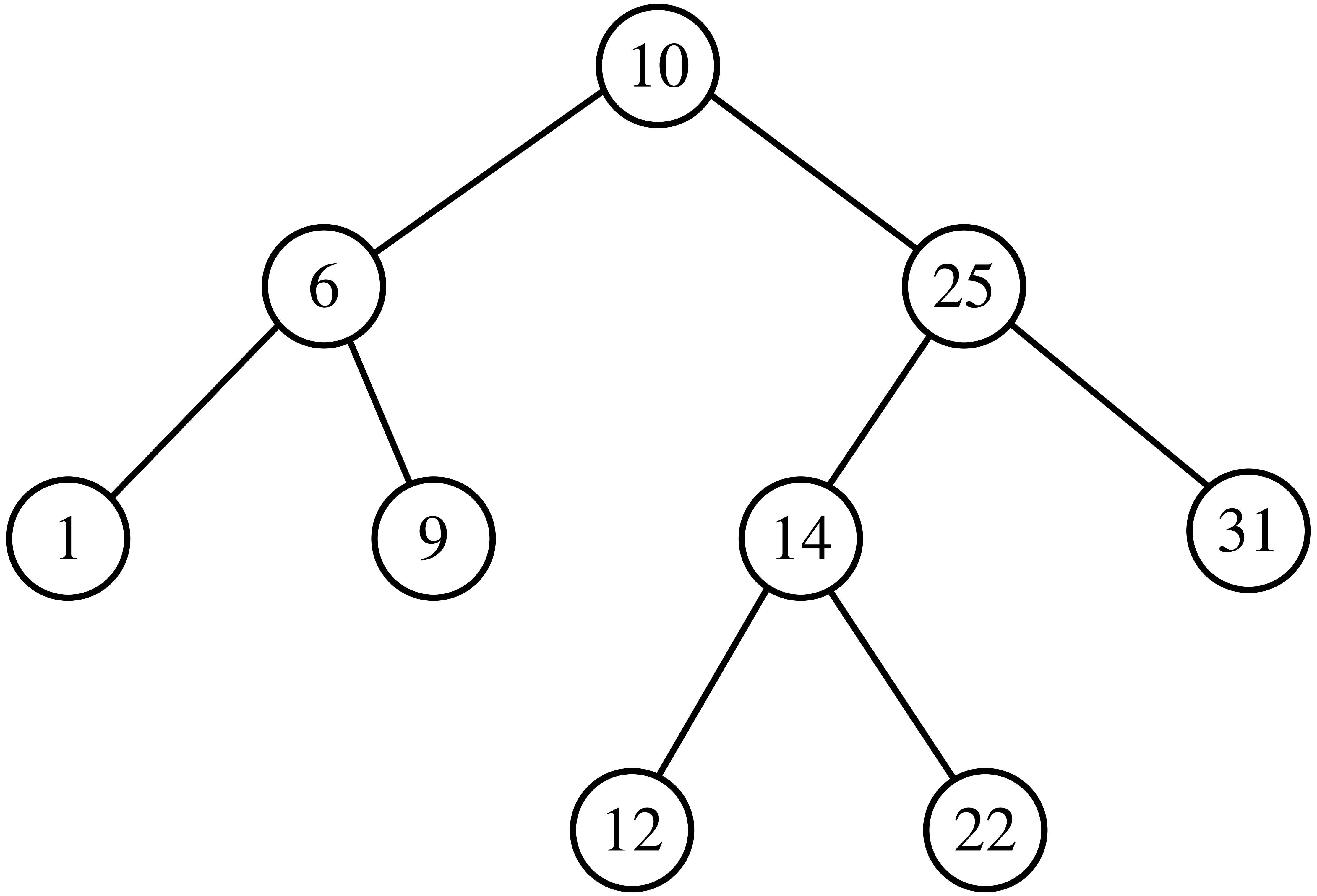
Defn: Let x be a node in a tree T with root r. Then any node y on the unique path from r to x is called an ancestor of x and x is called a descendant of y.



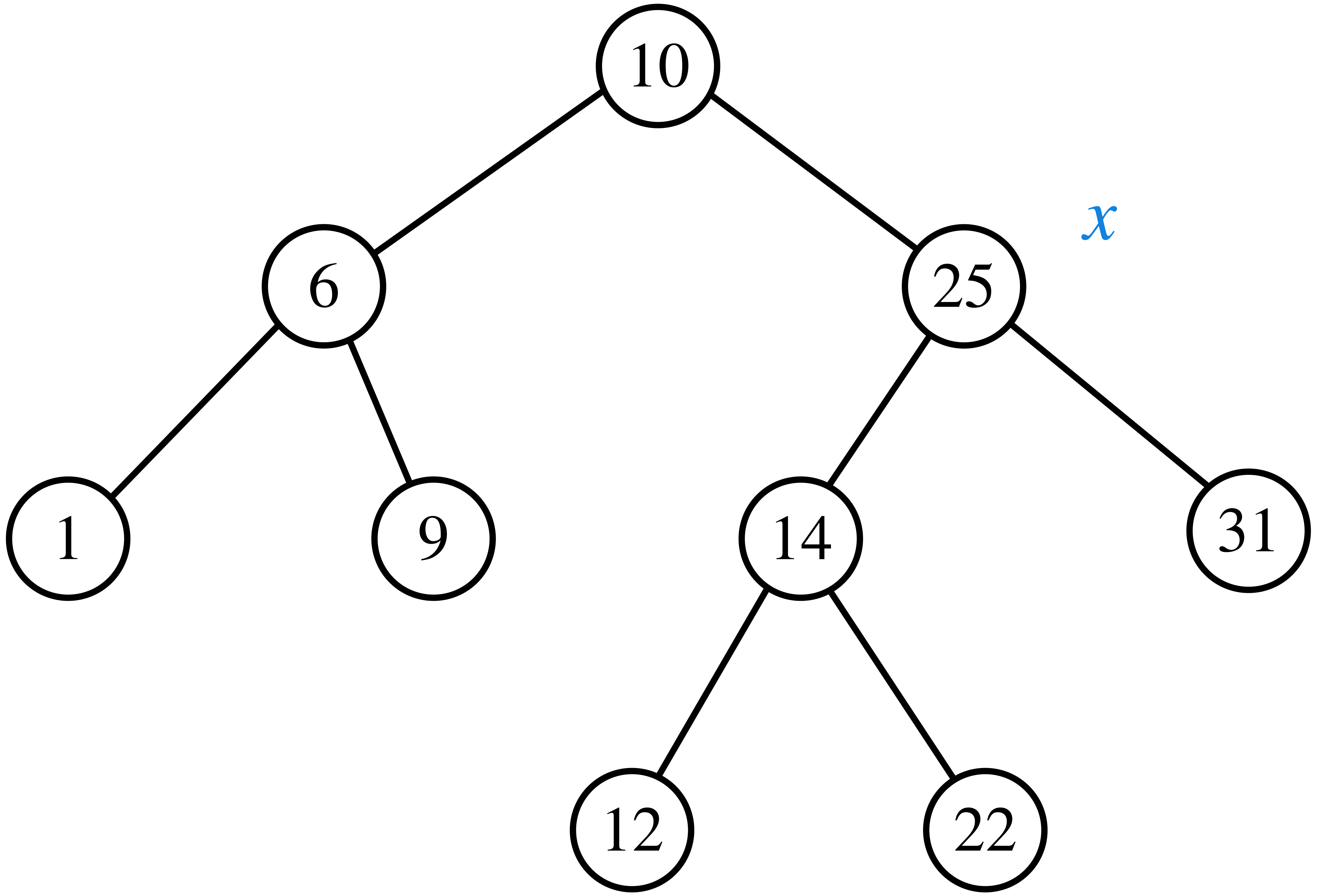
y is an ancestor of x,
x is a descendant of y

BST: Basic Terminology

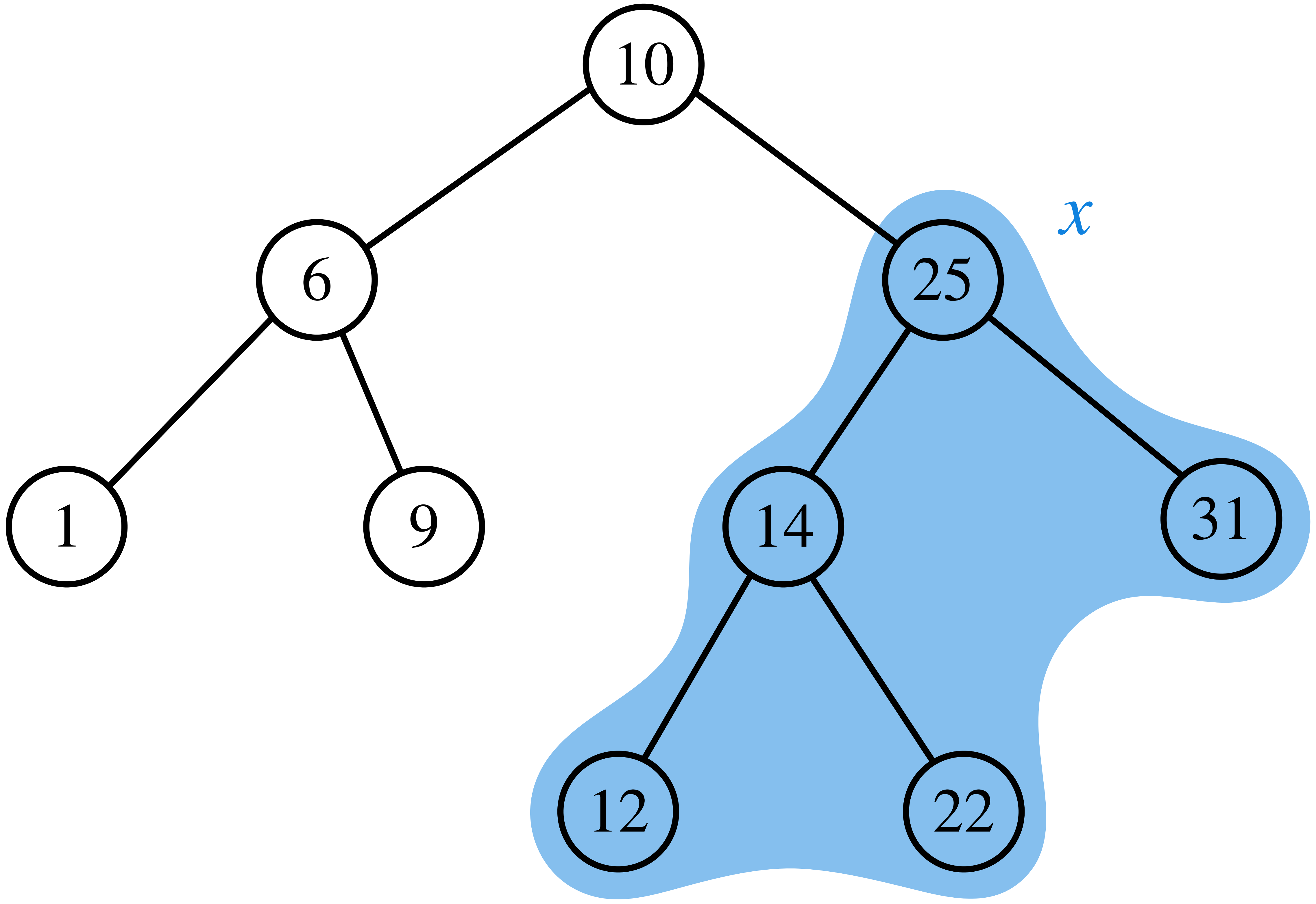
BST: Basic Terminology



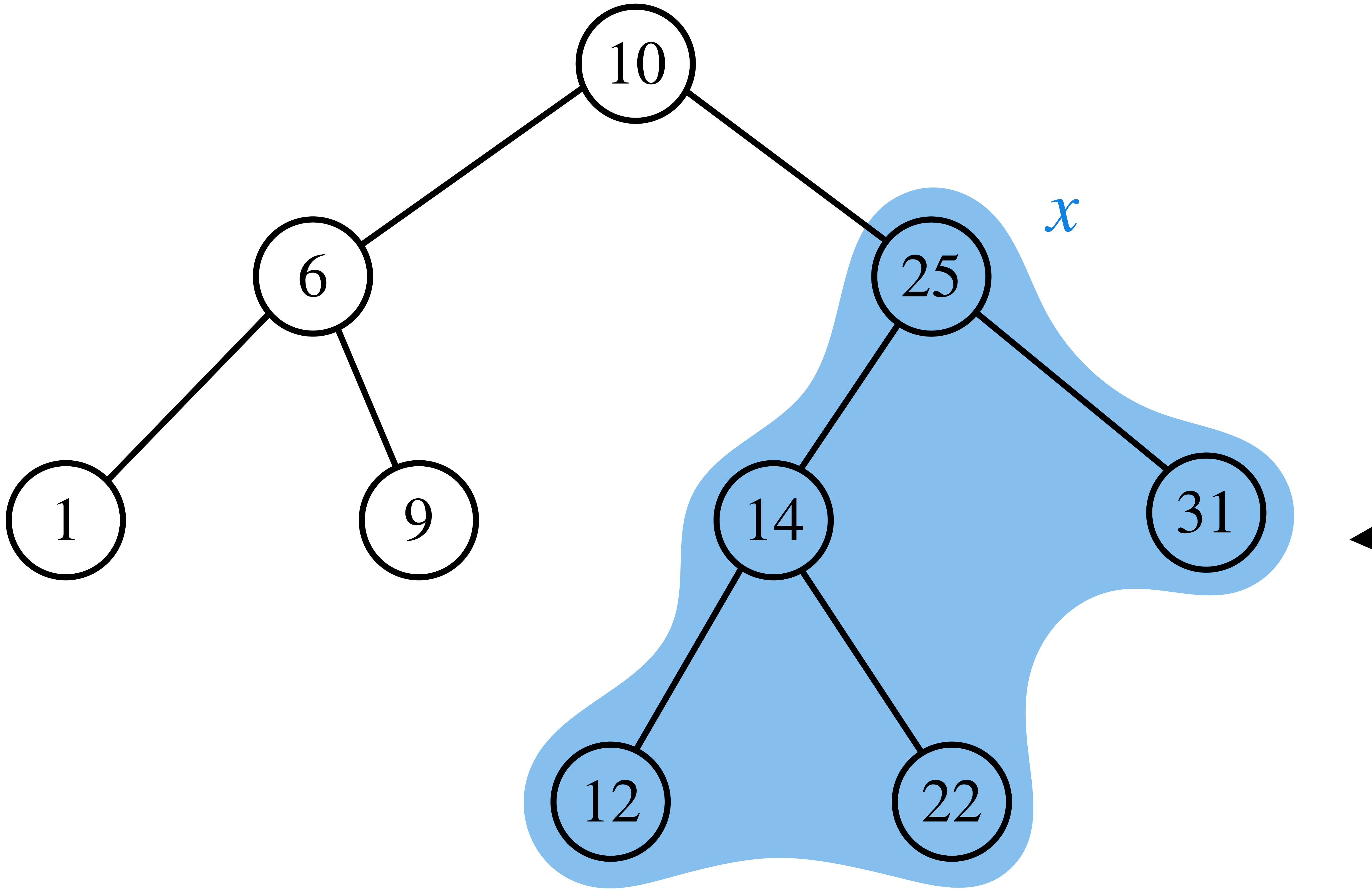
BST: Basic Terminology



BST: Basic Terminology



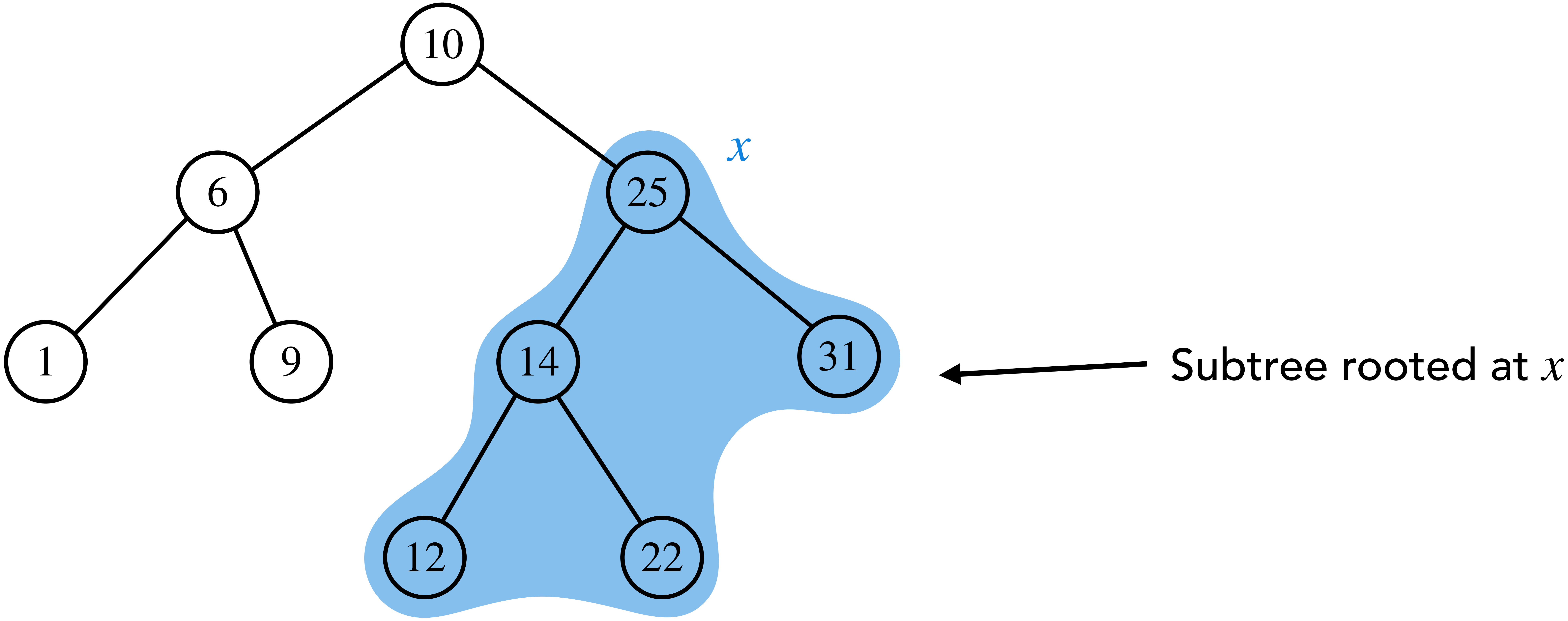
BST: Basic Terminology



Subtree rooted at x

BST: Basic Terminology

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



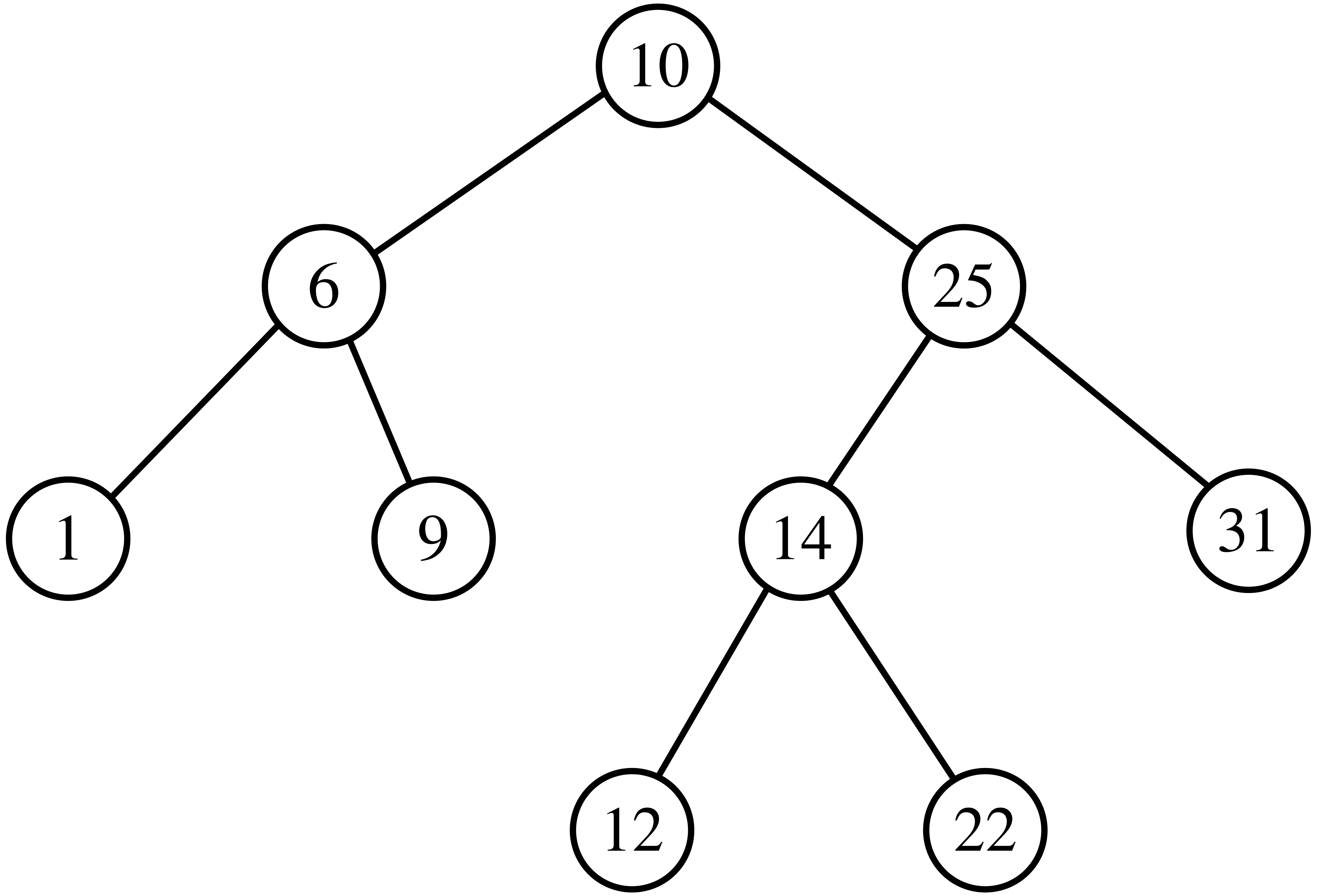
BST: Basic Terminology

BST: Basic Terminology

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

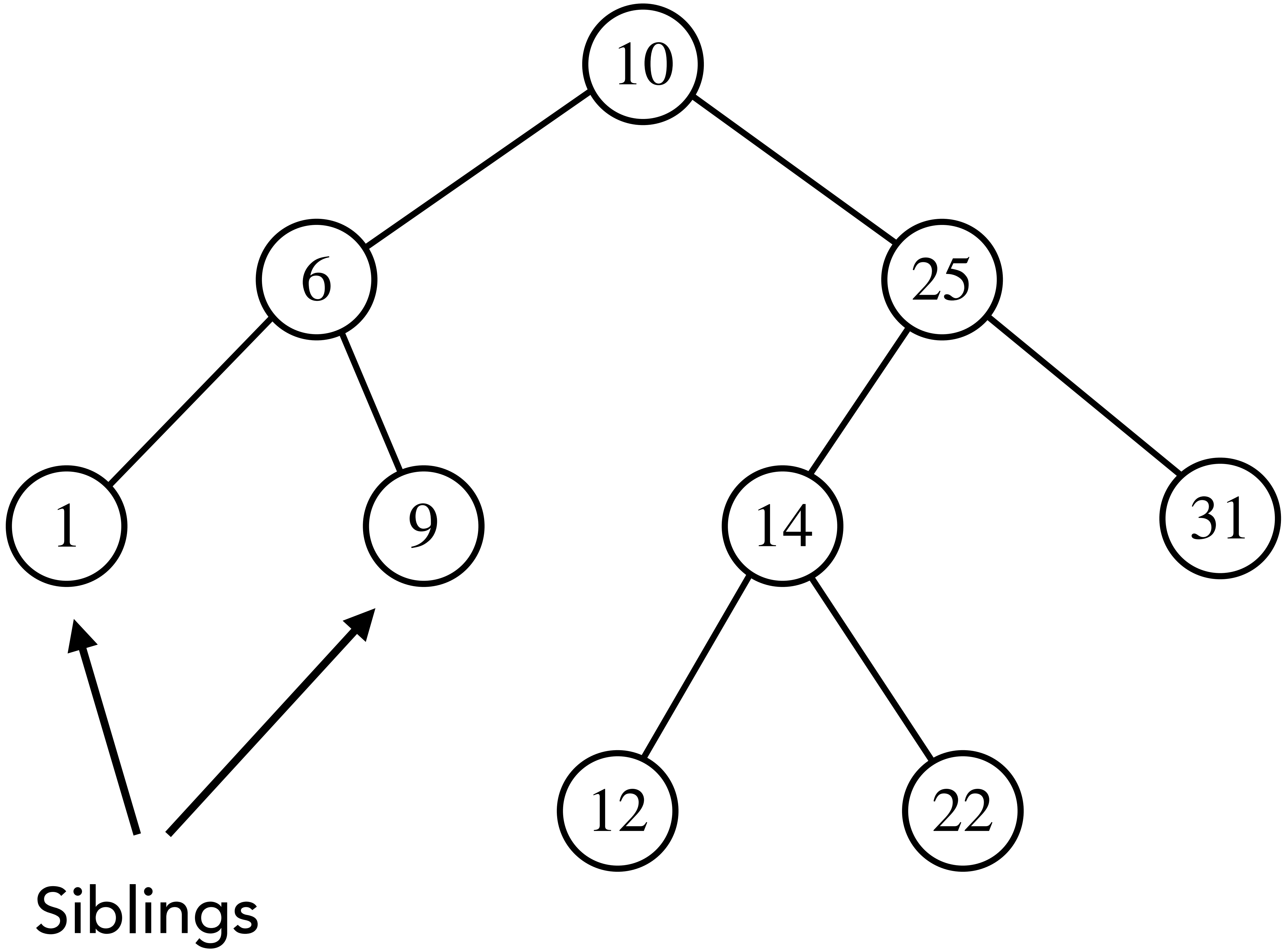
BST: Basic Terminology

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



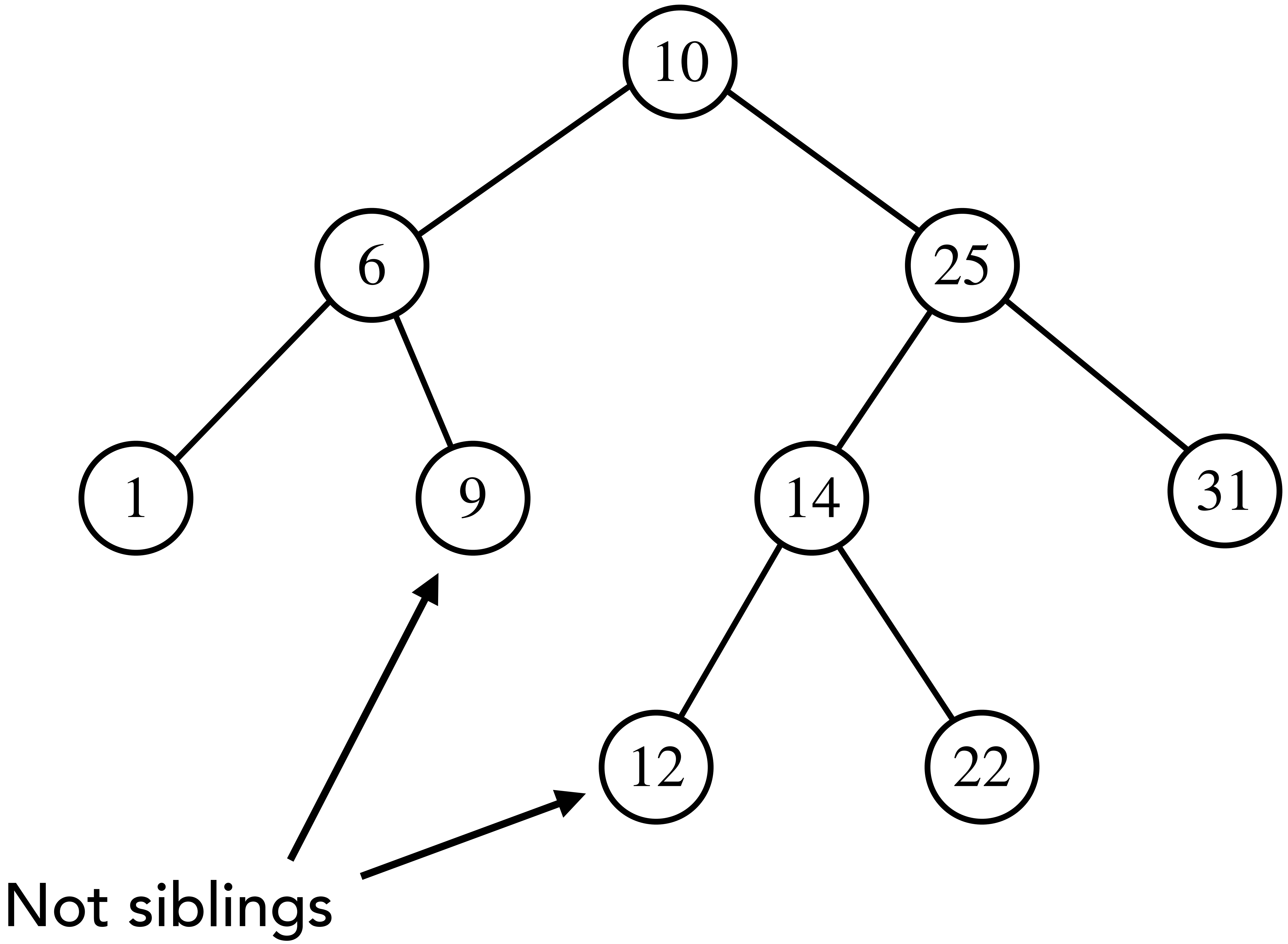
BST: Basic Terminology

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



BST: Basic Terminology

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



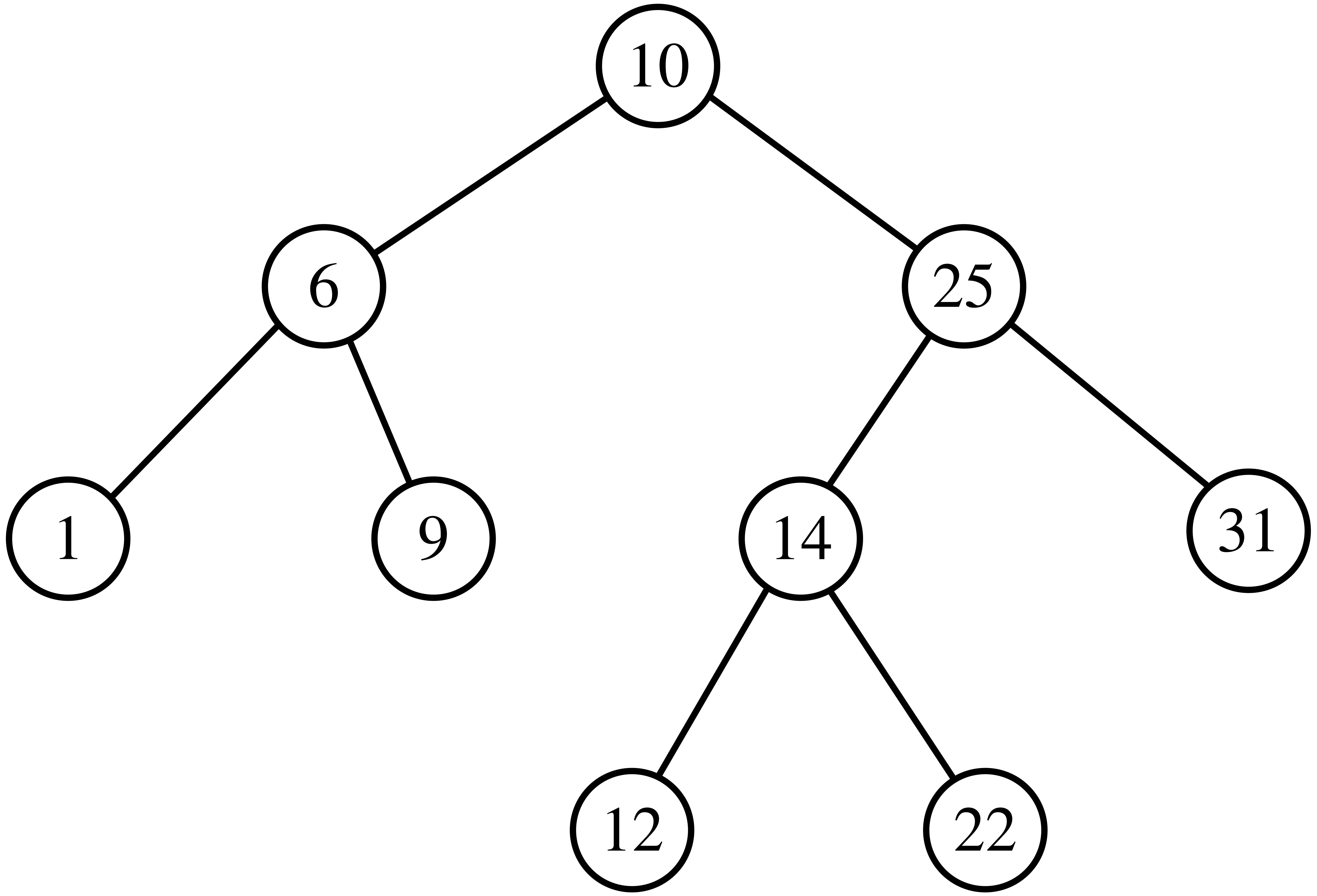
BST: Basic Terminology

BST: Basic Terminology

Defn: Nodes with no children are called leaves.

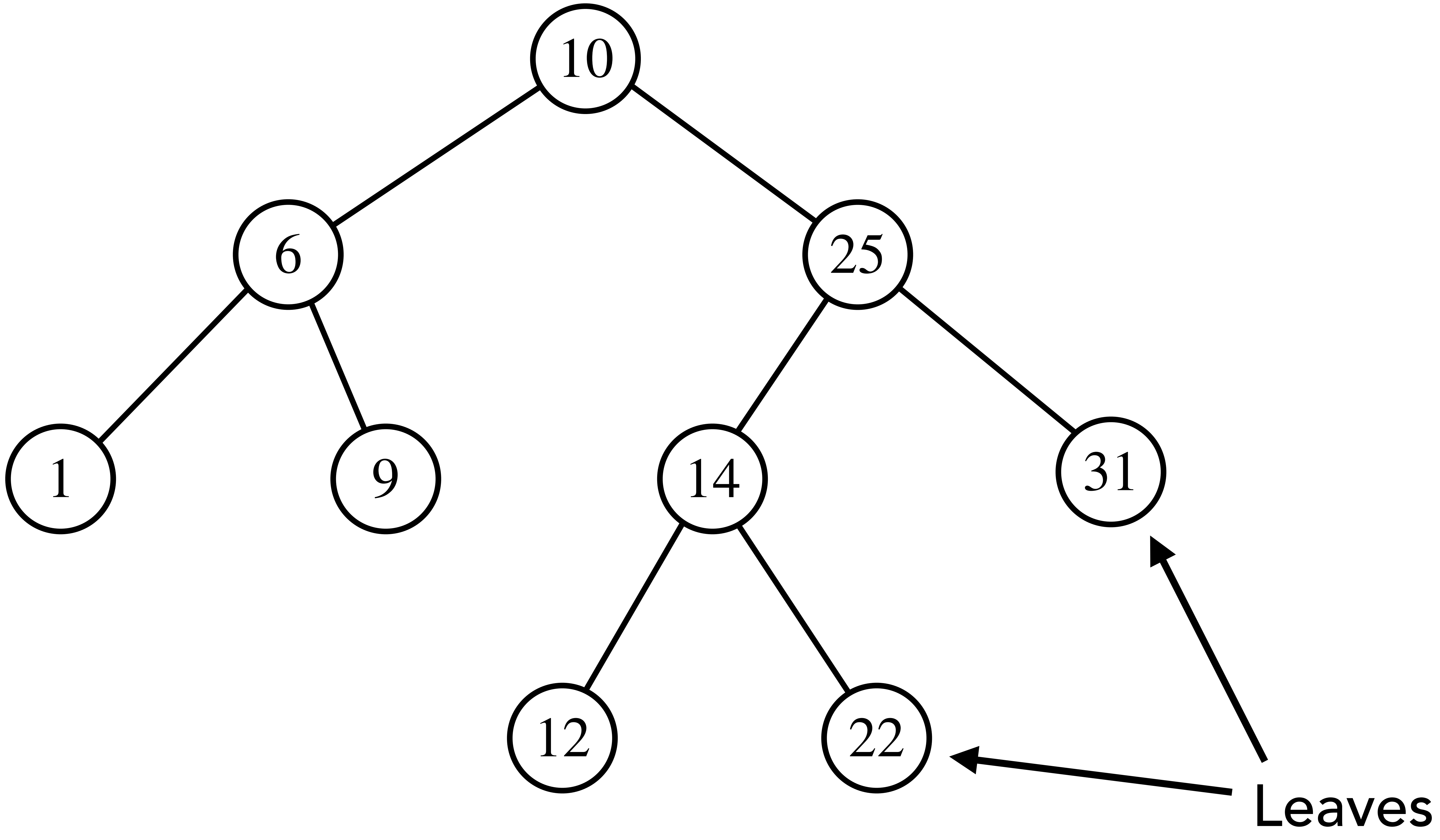
BST: Basic Terminology

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

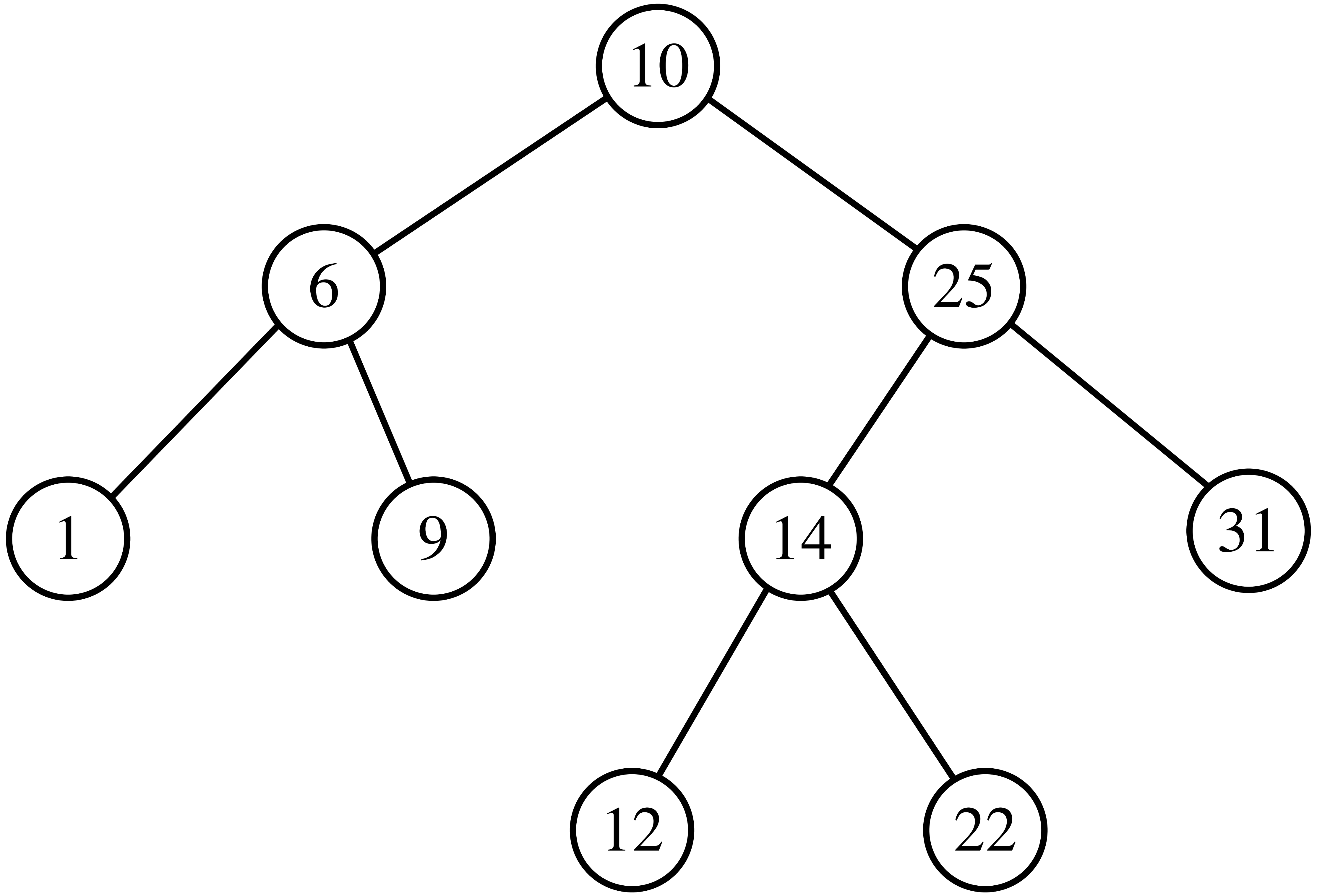
Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf.

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.

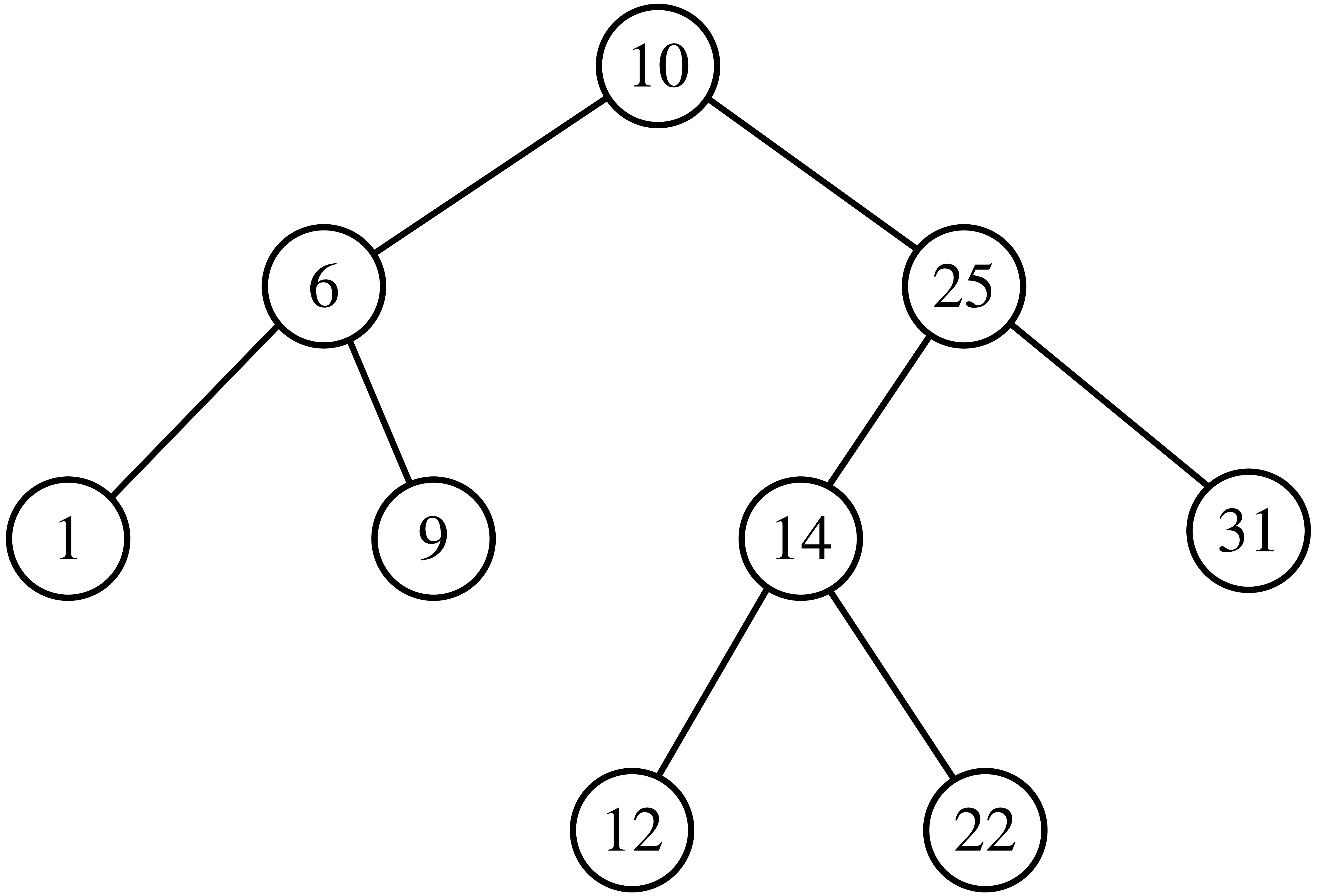
BST: Basic Terminology

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

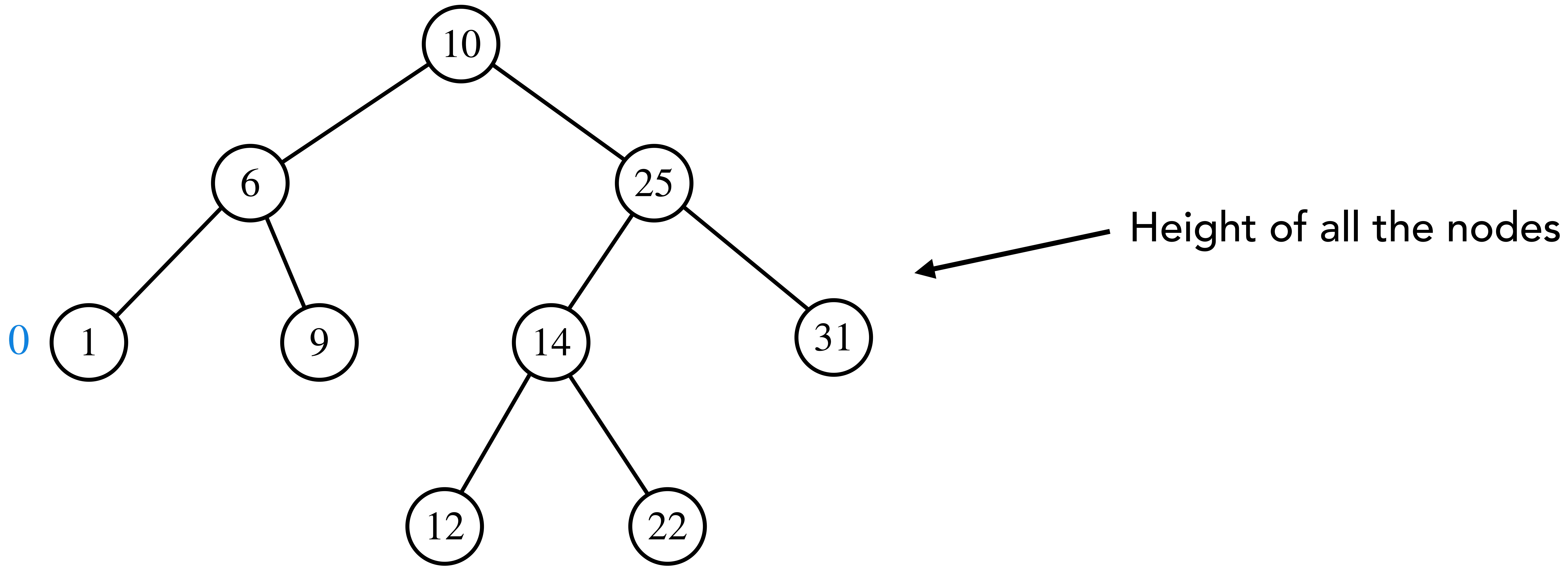
Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



Height of all the nodes

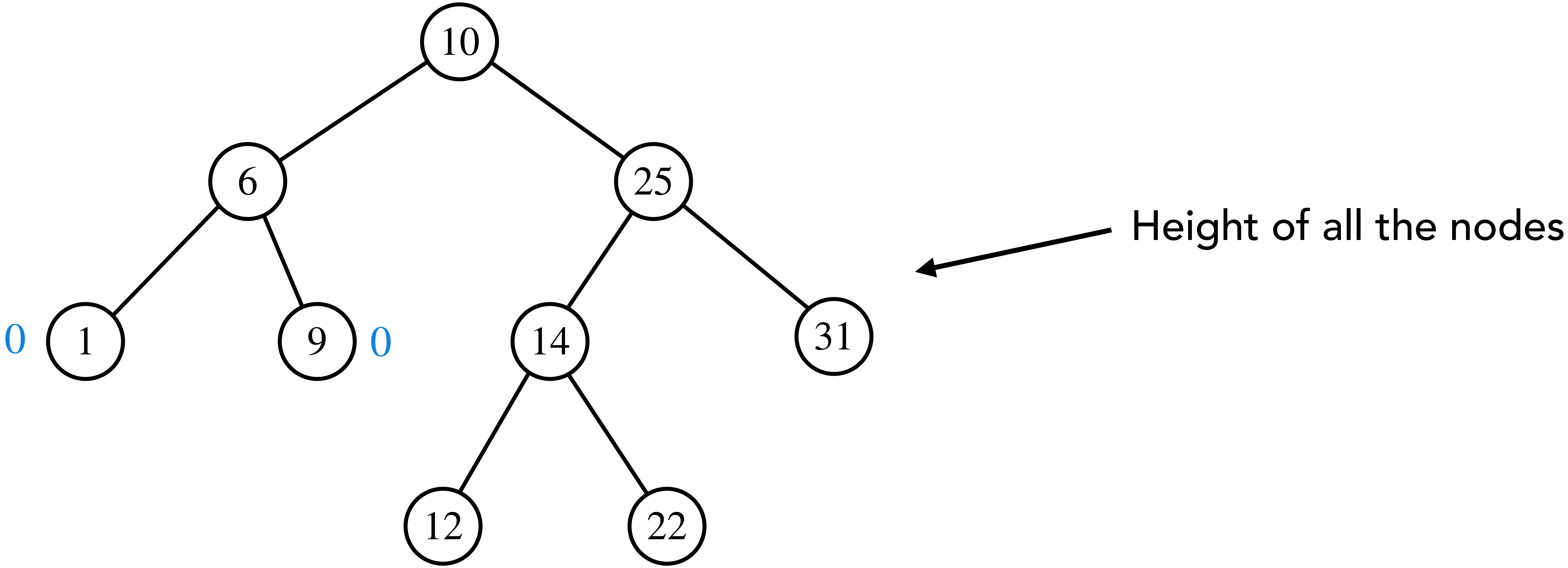
BST: Basic Terminology

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.



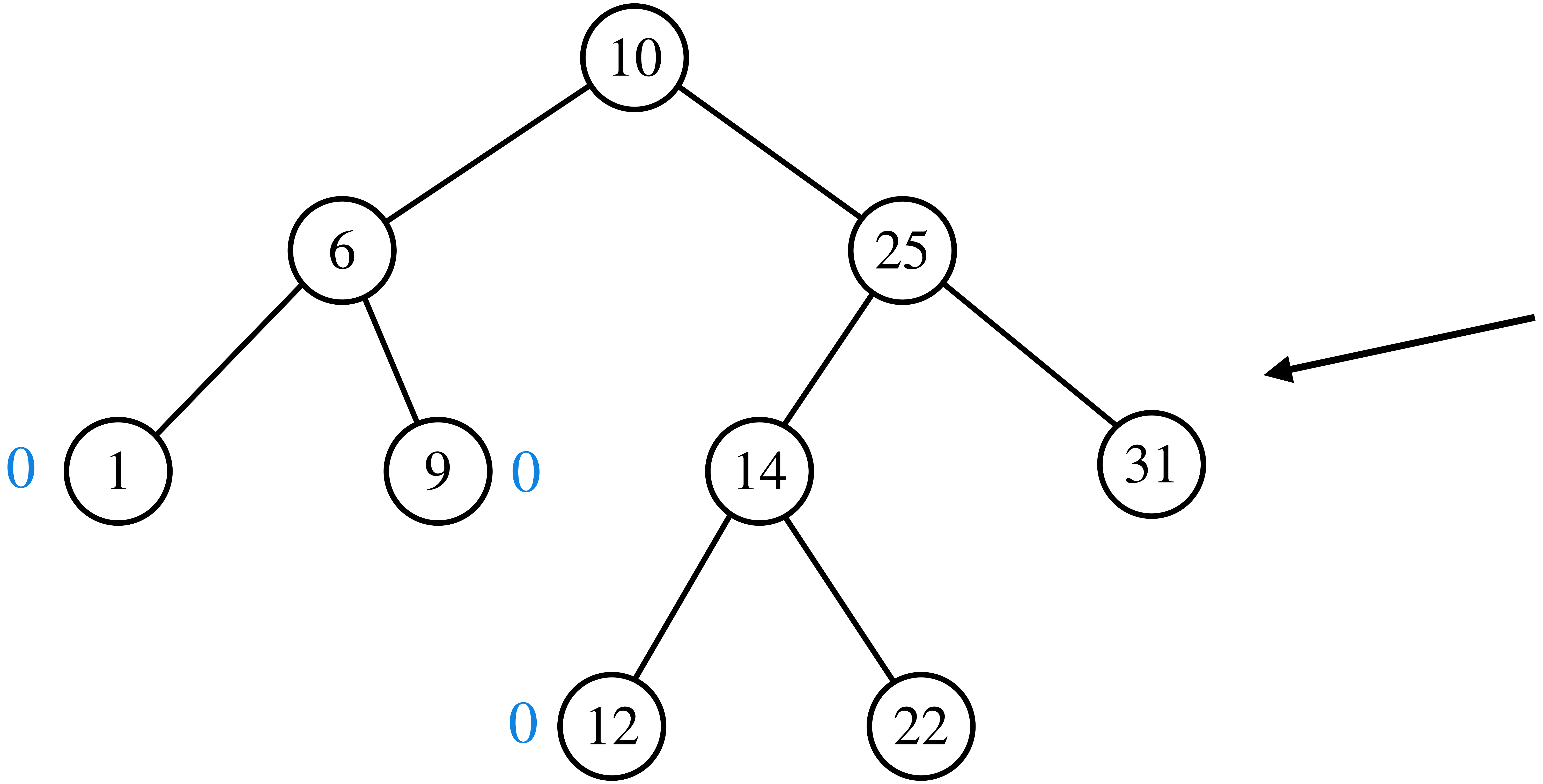
BST: Basic Terminology

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

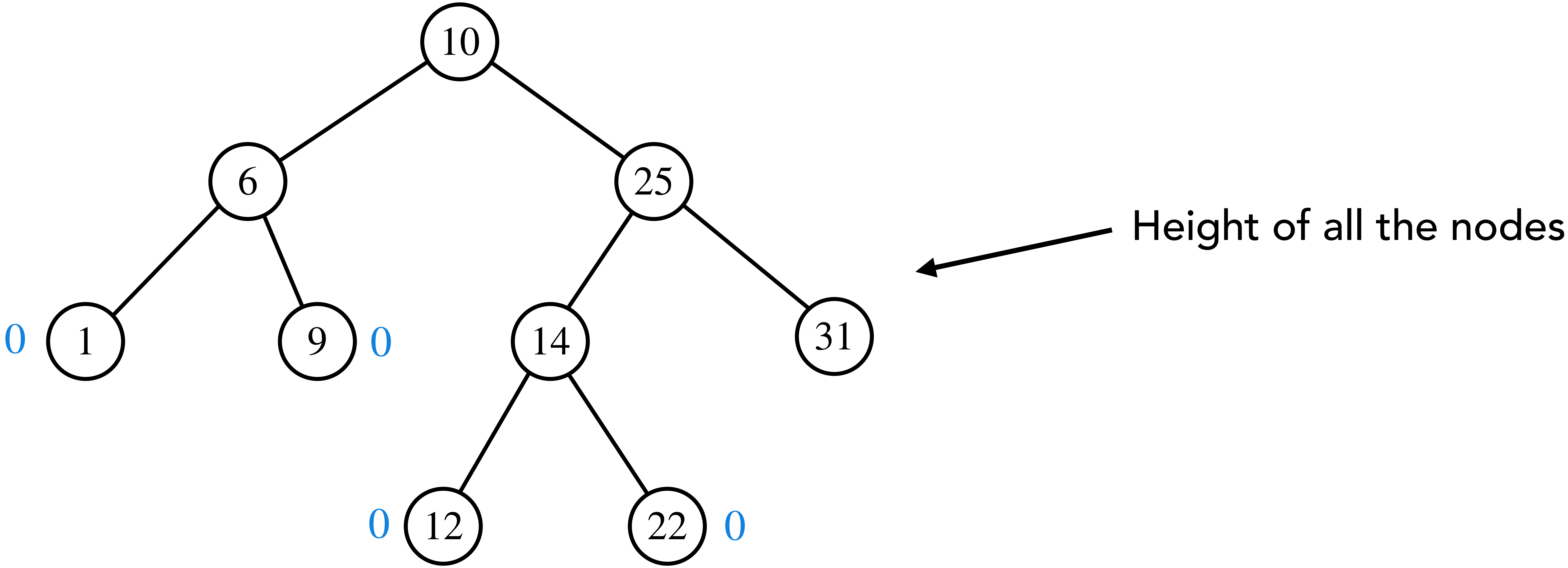
Defn: The height of a node in a tree is the number of edges on the longest downward path from the node to a leaf. Height of a tree is the height of its root.



Height of all the nodes

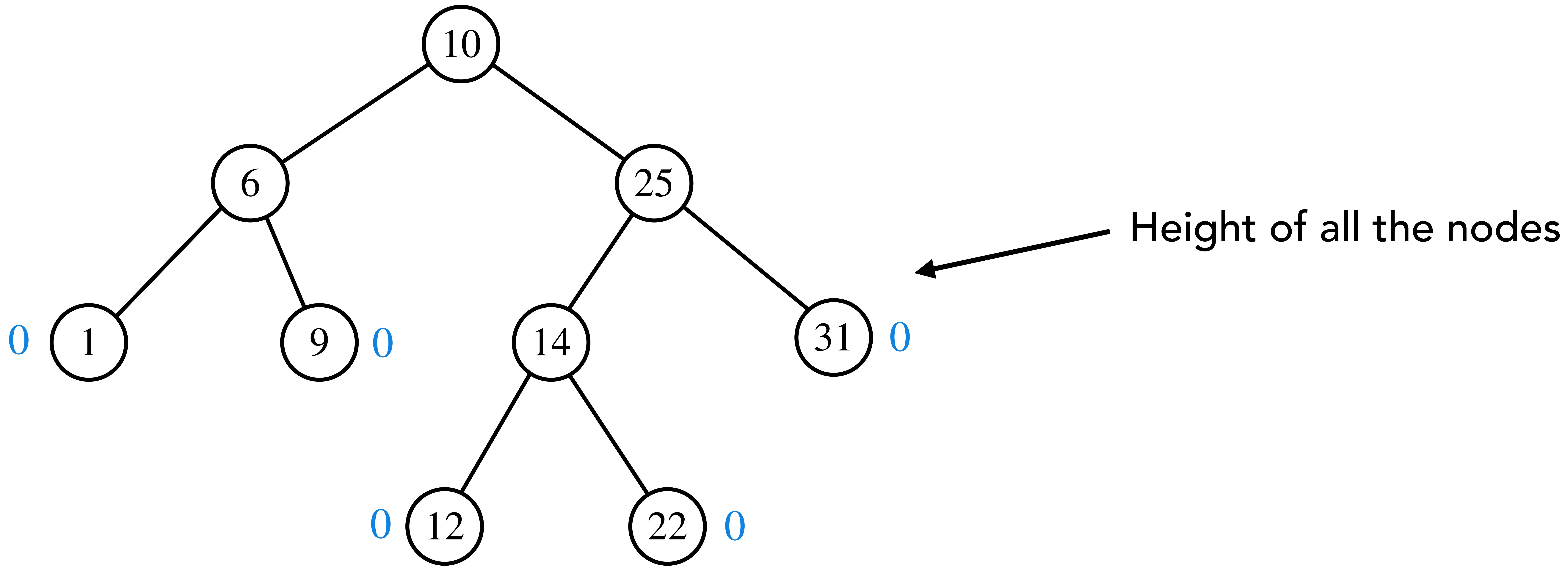
BST: Basic Terminology

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.



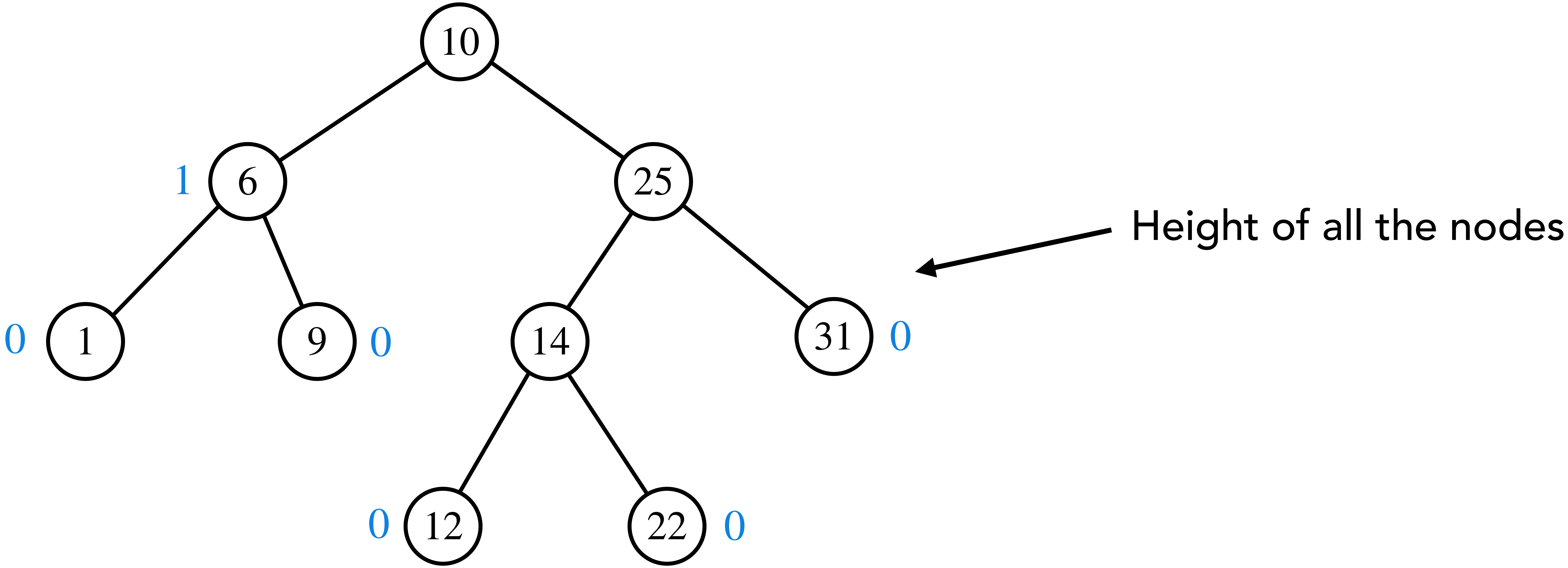
BST: Basic Terminology

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.



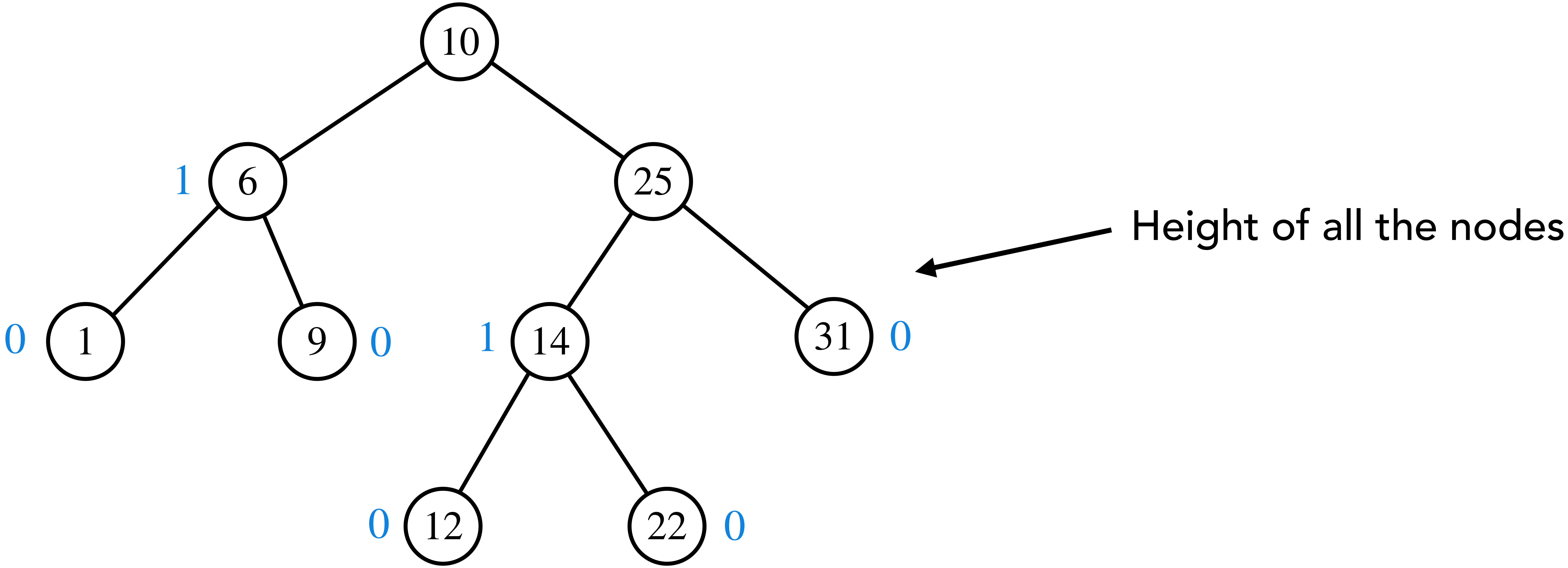
BST: Basic Terminology

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.



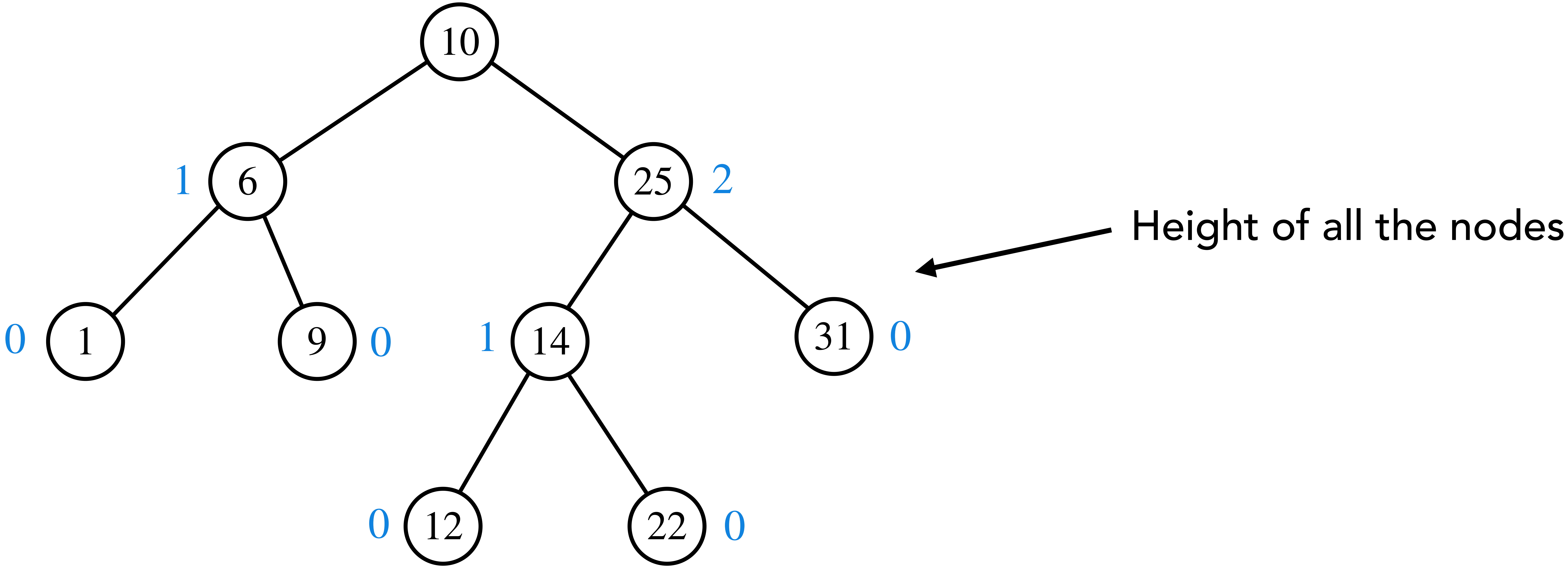
BST: Basic Terminology

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.



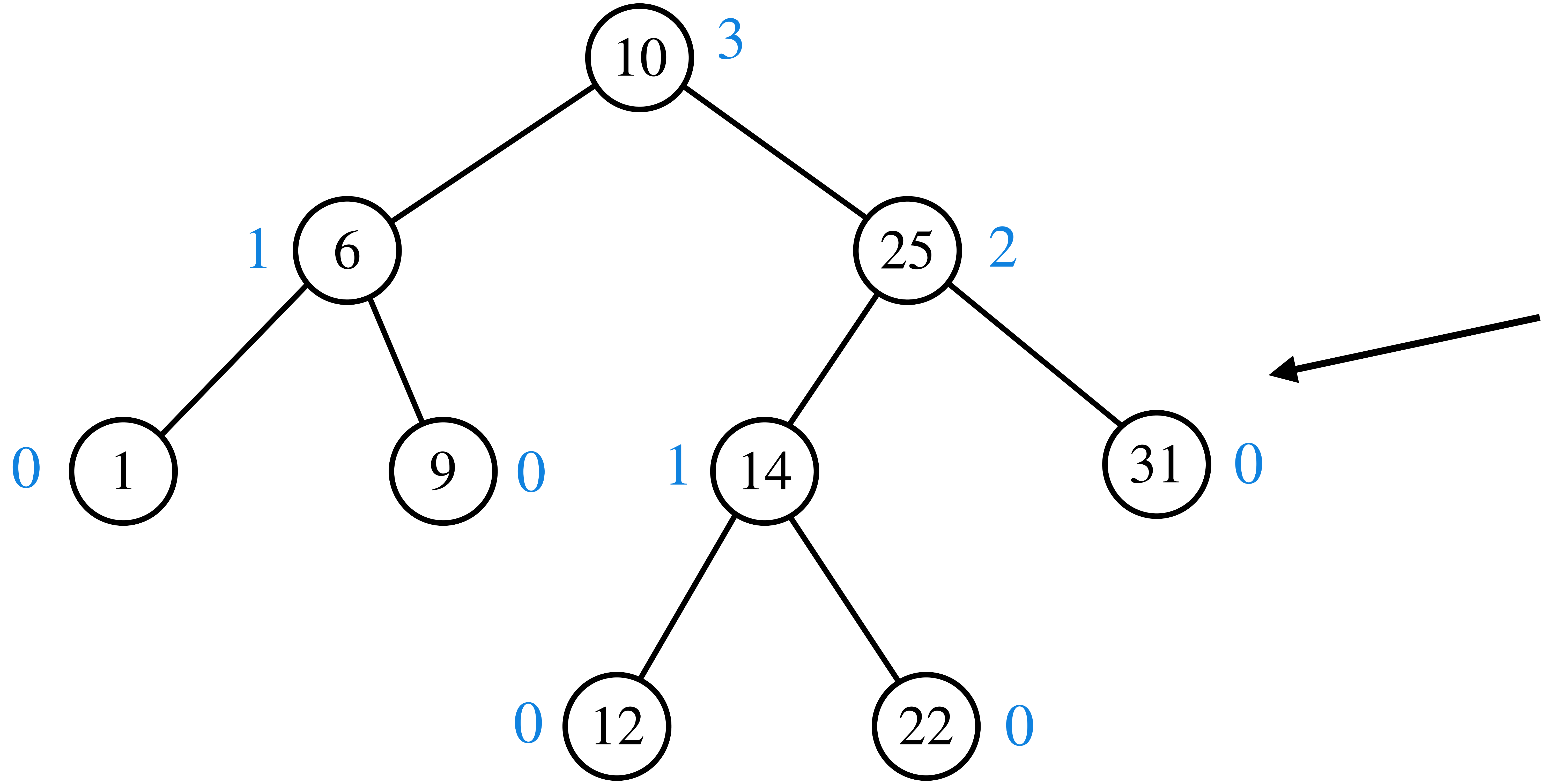
BST: Basic Terminology

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

Inorder-Tree-Walk(x):

Inorder Tree Walk

Inorder-Tree-Walk(x):

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

Inorder Tree Walk

Inorder-Tree-Walk(x)**:**

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

Inorder Tree Walk

```
Inorder-Tree-Walk(x):
  if x ≠ NIL
      Inorder-Tree-Walk(x . left)
      print x . key
```

Inorder Tree Walk

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 the BST T 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 the BST T in sorted order.

Inorder-Tree-Walk(x):

if $x \neq$ NIL
    Inorder-Tree-Walk($x\,.\,left$)
    print $x\,.\,key$
    Inorder-Tree-Walk($x\,.\,right$)

Proof of Correctness: We will prove it using induction on the number of nodes in the tree.