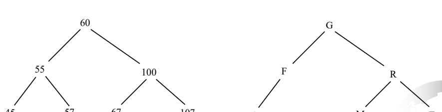
Chapter 25 Binary Search Trees

Objectives

- To design and implement a binary search tree (§25.2).
- To represent binary trees using linked data structures (§25.2.1).
- To search an element in binary search tree (§25.2.2).
- To insert an element into a binary search tree (§25.2.3).
- To traverse elements in a binary tree (§25.2.4).
- To delete elements from a binary search tree (§25.3).
- To display binary tree graphically (§25.4).
- To create iterators for traversing a binary tree (§25.5).

Binary Trees

A list, stack, or queue is a linear structure that consists of a sequence of elements. A binary tree is a hierarchical structure. It is either empty or consists of an element, called the *root*, and two distinct binary trees, called the *left subtree* and *right subtree*.



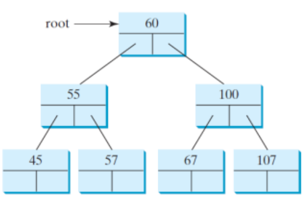
(A) (B)

Binary Tree Terms

- A Binary consists of
  - A root
  - A left binary tree (left child)
  - A right binary tree (right child)
- A node without children is a *leaf*. A node has one patent, except for the root.

Representing Binary Trees

A binary tree can be represented using a set of linked nodes. Each node contains a value and two links named *left* and *right* that reference the left child and right child, respectively.




```
class TreeNode<E> {
  E element;
  TreeNode<E> left;
  TreeNode<E> right;

  public TreeNode(E o) {
    element = o;
  }
}
```


Binary Search Tree

- A binary search tree of (key, value) pairs, with no duplicate keys, has the following properties
- Every node a left subtree has keys less than the key of the root
- Every node in its right subtree has keys greater than the key of the node.
- (often we only show the keys)
- What is the difference w.r.t heaps?

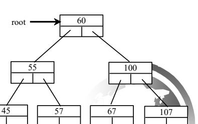
Searching an Element in a Binary Search Tree

```
public search(E element) {

  TreeNode<E> current = root; // Start from the root

  while (current != null)
    if (element key less than the key in current.element) {
      current = current.left; // Go left
    }

    else if (element value greater than the value in
current.element) {
      current = current.right; // Go right
    }

    else // Element matches current.element
      return found ; // Element is found

  return not found; // Element is not in the tree
}
```


Inserting an Element to a Binary Tree

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

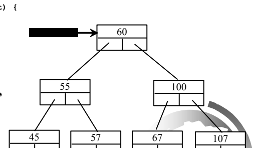
Insert 101 into the following tree.



Trace Inserting 101 into the following tree

```
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

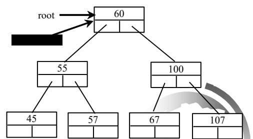
Insert 101 into the following tree.



Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node

  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

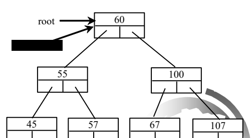
Insert 101 into the following tree.



Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;

    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

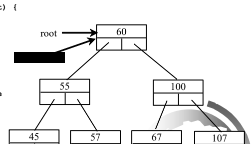


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
                                                       {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 < 60?

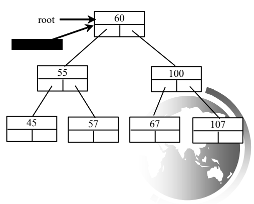


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else                                       {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 > 60?

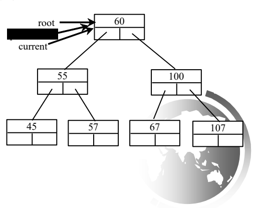


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {

      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 > 60 true

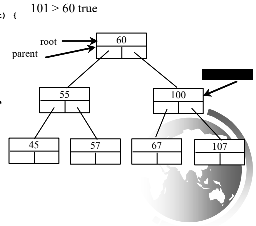


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;

    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 > 60 true

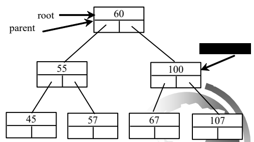


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;

    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 > 60 true

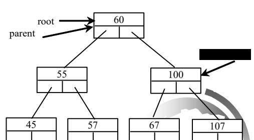


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
                                                   {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 < 100 false

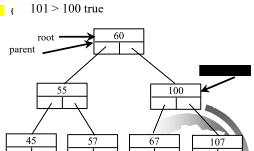


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else                                                 {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 > 100 true

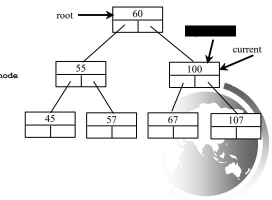


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {

      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 > 100 true

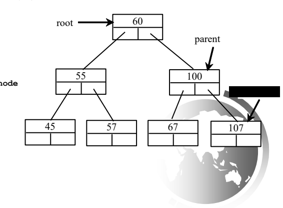


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;

    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 > 100 true

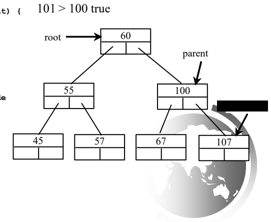


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;

    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 > 100 true

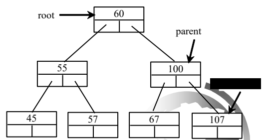


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if                                                  {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 < 107 true

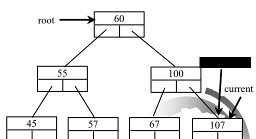


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {

      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

101 < 107 true

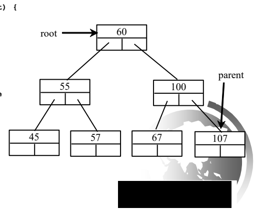


Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;

    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

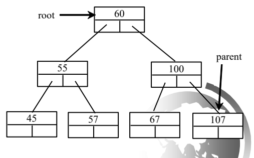
101 < 107 true



Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
                                    current is null now
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

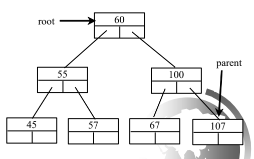


[redacted]

Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
                                 101 < 107 true
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.

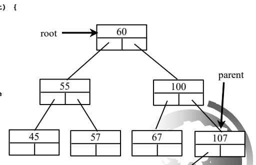


[redacted]

Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
                                 101 < 107 true
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```

Insert 101 into the following tree.



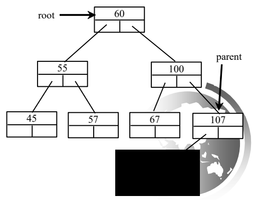
[redacted]

Trace Inserting 101 into the following tree, cont.

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);


}
```

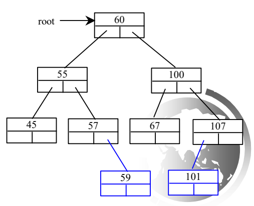
101 < 107 true

Insert 101 into the following tree.



Inserting 59 into the Tree

```
if (root == null)
  root = new TreeNode(element);
else {
  // Locate the parent node
  current = root;
  while (current != null)
    if (element value < the value in current.element) {
      parent = current;
      current = current.left;
    }
    else if (element value > the value in current.element) {
      parent = current;
      current = current.right;
    }
    else
      return false; // Duplicate node not inserted

  // Create the new node and attach it to the parent node
  if (element < parent.element)
    parent.left = new TreeNode(elemenet);
  else
    parent.right = new TreeNode(elemenet);

  return true; // Element inserted
}
```



Tree Traversal

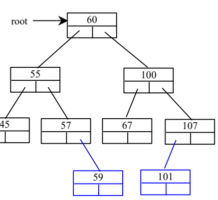
Tree traversal is the process of visiting each node in the tree exactly once. There are several ways to traverse a tree. This section presents *depth-first: in-*, *pre-*, *post order and breadth-first: level order* traversals.

- InOrder
  - The inorder traversal is to visit the left subtree of the current node first recursively, then the current node itself, and finally the right subtree of the current node recursively.
- Postorder
  - The postorder traversal is to visit the left subtree of the current node first, then the right subtree of the current node, and finally the current node itself.

Tree Traversal, cont.

- Preorder
  - The preorder traversal is to visit the current node first, then the left subtree of the current node recursively, and finally the right subtree of the current node recursively.
- Level order
  - The level order (breadth-first) traversal is to visit the nodes level by level. First visit the root, then all children of the root from left to right, then grandchildren of the root from left to right, and so on.

Tree Traversal, cont.

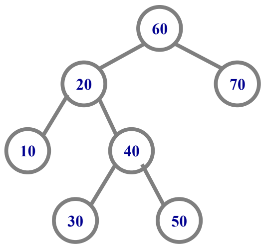


Inorder: 45 55 57 59 60 67 100 101 107
Postorder: 45 59 57 55 67 101 107 100 60
Preorder: 60 55 45 57 59 100 67 107 101
Level order: 60 55 100 45 57 67 107 59 101

Breadth-first traversal (BFS)

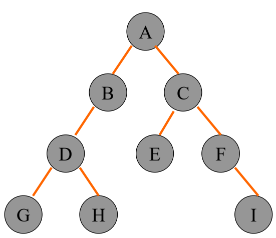
- Breadth-first processes the tree **level by level** starting at the root and handling all the nodes at a particular level from **left to right**.
- To achieve we use a Queue, because the parent child references are not sufficient

Breadth-first traversal



60 – 20 – 70 – 10 – 40 – 30 – 50

LevelOrder



	Queue	Output
Init	[A]	-
Step 1	[B,C]	A
Step 2	[C,D]	A B
Step 3	[D,E,F]	A B C
Step 4	[E,F,G,H]	A B C D
Step 5	[F,G,H]	A B C D E
Step 6	[G,H,I]	A B C D E F
Step 7	[H,I]	A B C D E F G
Step 8	[I]	A B C D E F G H
Step 9	[ ]	A B C D E F G H I

The Tree Interface

«interface»
java.lang.Collection<E>

«interface»
Tree<E>

The Tree interface defines common operations for trees.

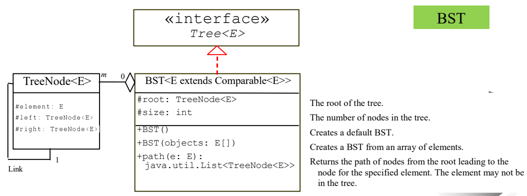
+search(e: E): boolean	Returns true if the specified element is in the tree.
+insert(e: E): boolean	Returns true if the element is added successfully.
+delete(e: E): boolean	Returns true if the element is removed from the tree successfully.
+inorder(): void	Prints the nodes in inorder traversal.
+preorder(): void	Prints the nodes in preorder traversal.
+postorder(): void	Prints the nodes in postorder traversal.
+getSize(): int	Returns the number of elements in the tree.
+isEmpty(): boolean	Returns true if the tree is empty.
+clear(): void	Removes all elements from the tree.

Override the add, isEmpty, remove, containsAll, addAll, removeAll, retainAll, toArray(), and toArray(T[]) methods defined in Collection using default methods.

Tree

The BST Class

Let's define the binary tree class, named BST with A concrete BST class can be defined to extend AbstractTree.

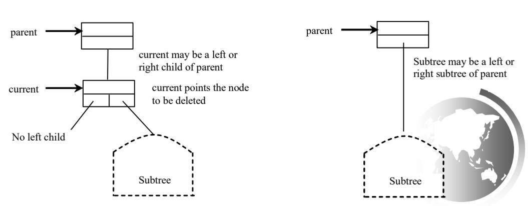


Deleting Elements in a Binary Search Tree

- Locate the node that contains the element and its parent node.
- Let current point to the node that contains the element in the binary tree and parent point to the parent of the current node. (notice: parent can be the root reference)
- The current node may be a left child or a right child of the parent node.
- There are two cases.

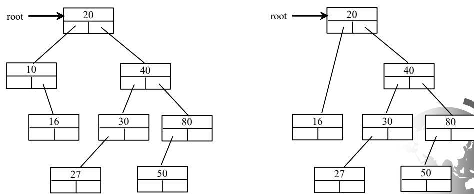
Deleting Elements in a Binary Search Tree

Case 1: The current node does not have a left child, as shown in this figure (a). Simply connect the parent with the right child of the current node, as shown in this figure (b).



Deleting Elements in a Binary Search Tree

For example, to delete node 10 in Figure 25.9a. Connect the parent of node 10 with the right child of node 10, as shown in Figure 25.9b.



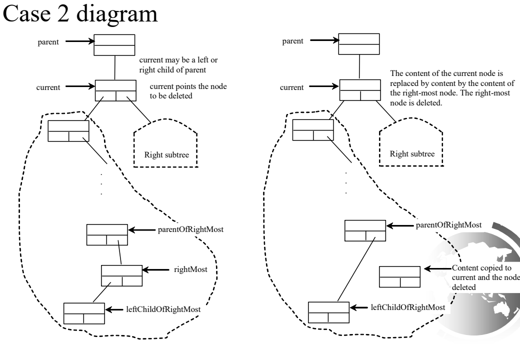
Deleting Elements in a Binary Search Tree

Case 2: The current node has a left child.

- Let rightMost point to the node that contains the largest element in the left subtree and parentOfRightMost point to its parent node.
- Note that the rightMost node cannot have a right child, but may have a left child.
- Replace the element value in the current node with the one in the rightMost node
- Connect the parentOfRightMost node with the left child of the rightMost node, and delete the rightmost node.

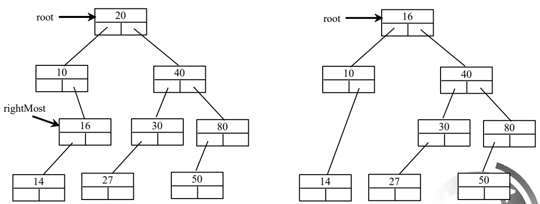
Deleting Elements in a Binary Search Tree

Case 2 diagram

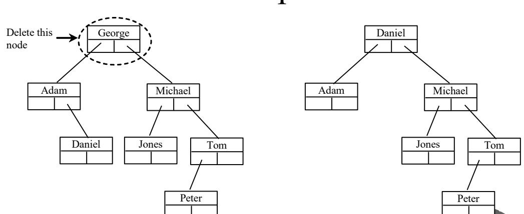


Deleting Elements in a Binary Search Tree

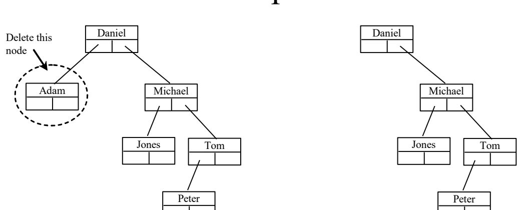
Case 2 example, delete 20



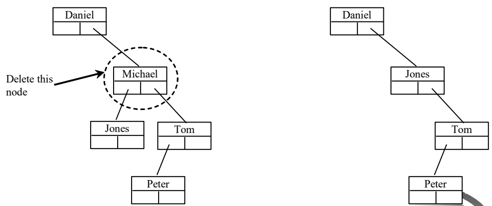
Examples



Examples



Examples



TestBSTDelete Run

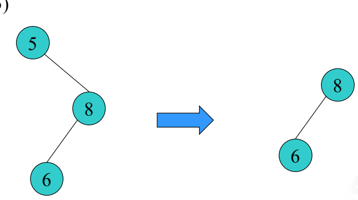
Alternative, more balanced approach

- Cases to Consider
  - Delete something that is not there
    - Throw exception
  - Delete a leaf
    - Easy, just set link from parent to null
  - Delete a node with one child
  - Delete a node with two children

Delete

Case 1: one child

delete(5)



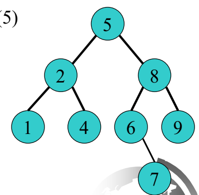
Child becomes root

Delete

Case 2: two children

Which are valid replacement nodes?

delete(5)



4 and 6, WHY?

max of left, min of right

what would be a good one here?

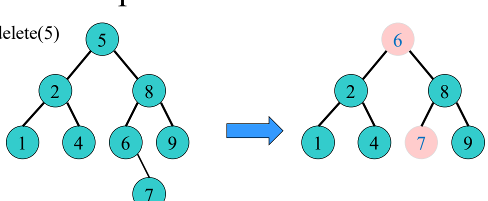
6, WHY?

Digression: inorder traversal of BST

- In order:
  - go left
  - visit the node
  - go right
- The keys of an inorder traversal of a BST are in sorted order!

Replace with successor

delete(5)

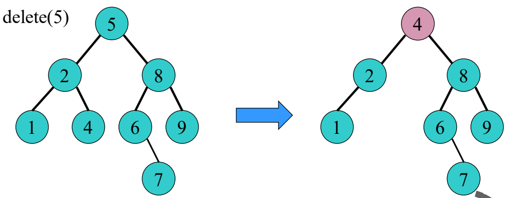


Replace root with its **leftmost right descendant** and replace that node **with its right child,** if necessary (an easy delete case).
That node is the inorder successor of the root.

Can that node have two children? A left child?

Replace with predecessor

delete(5)



Replace root with its **rightmost leftt descendant** and replace that node **with its lef]t child,** if necessary (an easy delete case).
That node is the inorder predecessor of the root.

Can that node have two children? A right child?

Delete
Case 2: two children

1. Find the ***inorder successor or predecessor M*** of N's search key.
   – The node whose search key comes immediately after or before N's search key
2. Copy the item of M, to the deleting node N.
3. Remove the node M from the tree.

Iterators

An *iterator* is an object that provides a uniform way for traversing the elements in a container such as a set, list, binary tree, etc.

«interface» *java.util.Iterator<E>*	
+hasNext(): boolean	Returns true if the iterator has more elements.
+next(): E	Returns the next element in the iterator.
+remove(): void	Removes from the underlying container the last element returned by the iterator (optional operation).

TestBSTWithIterator Run