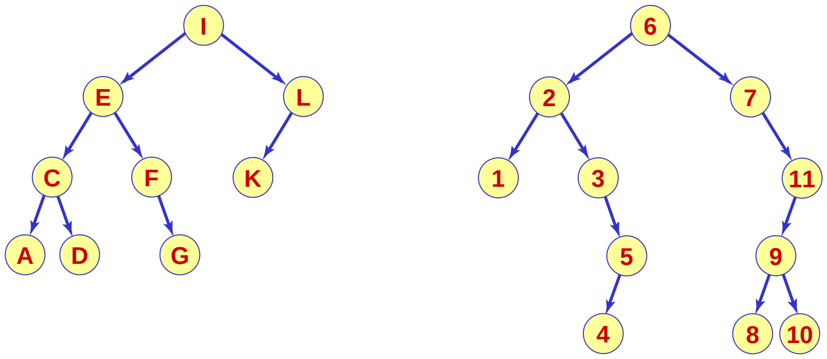
Non-Linear Data Structures

Binary Search Trees

Binary Search Tree Definition

- A **Binary Search Tree (BST)** is a binary tree such that for each node, N, the following conditions are true:
    1. If L is any node in the left subtree of N, then the key of L is less than the key of N.
    2. If R is any node in the right subtree of N, then the key of R is greater than the key of N.
- The key values of the elements must be from an ordered set

Binary Search Tree Examples



Binary Search Tree Operations

Some of the operations assume there is one element in the tree designated as current.

goRoot()
goLeft()
goRight()
search(target)

Change the location of Current

inset(entry)
remove(entry)
replace(entry)
retrieve()

Manipulate nodes and their data

preorder, inorder, postorder
treeClear(R), treeCopy(S)

Traverse the tree

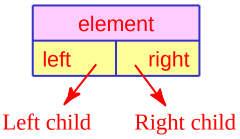
getRoot()
getCurrent()
size(), isEmpty()

Get tree information

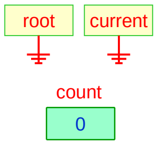
hasParent()
hasLeftChild()
hasRightChild()

Get node relation information

BST Implementation Using a Linked Structure



- Use the binary tree node generic class: BTNode<E>, as defined before.
- Use two BTNode<E> pointer variables:
  - root -- points to the root node of the tree.
  - current -- points to the current element node in the tree.
- Use one integer variable:
  - count -- stores the number of nodes in the tree.
- An empty BST is initialized by setting: root = current = null, and count = 0.



- The location of the current node is controlled by insert, remove, search, and the three go operations.

BST Implementation as a Java Generic class Using a Linked Structure

```
import BTNode;                        // Provides BTNode<E> generic class

public class BSTree<E>  {

   // DATA FEILDS:
   protected BTNode<E> root;
   protected BTNode<E> current;
   protected int count;

   // CONSTRUCTORS:
   public BSTree()  { root = null;  current = null;  count = 0; }
   public BSTree(BSTree source)  {         // This is a copy constructor
      this();                              // Calls the default constructor
      root = treeCopy(source.root);
      current = source.current;  count = source.count;
   }
```

BST Implementation as a Java Generic class Using a Linked Structure

```
// MUTATOR METHODS:
public void goRoot()  { if (count > 0) current = root; }
public void goLeft()  { if (hasLeftChild()) current = current.getLeft(); }
public void goRight() { if (hasRightChild()) current = current.getRight();}
public void search(E target)  {   …  …  …   }
public void insert(E entry)  {   …  …  …   }
public boolean remove(E entry)  {   …  …  …   }
public void replace(E entry)  {   …  …  …   }
public void treeClear(BTNode<E> p)  {   …  …  …   }

// PRIVATE HELPER FUNCTIONS:
private void createFirstNode(E entry)  {   …  …  …   }
private void addLeft(E entry)  {   …  …  …   }
private void addRight(E entry)  {   …  …  …   }
```

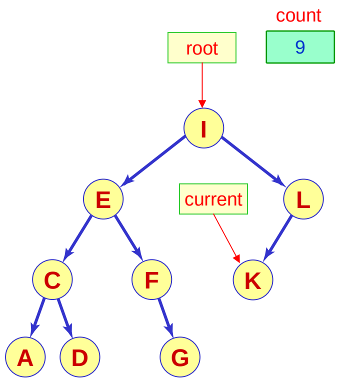
BST Implementation as a Java Generic class Using a Linked Structure

```
  // OBSERVER METHODS:
  public E retrieve() {
     if (current != null)
        return current.getElement()
     else
        return null;
  }
  public boolean hasParent()  { return current != root; }
  public boolean hasLeftChild()  { return current.getLeft() != null; }
  public boolean hasRightChild()  { return current.getRight() != null; }
  public boolean isEmpty()  { return (count == 0); }
  public BTNode<E> getRoot()  { return root; }
  public BTNode<E> getCurrent()  { return current; }
  public int size()  { return count; }
}
```

Mapping Operation: goRoot() … *O*(1)

- **Precondition:**
  - size( ) > 0 .
- **Postcondition**:
  - The current node is now the root of the tree .
- **Code**:

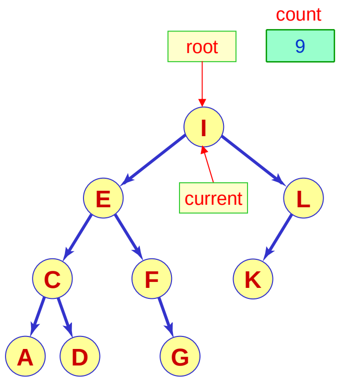
```
if (count > 0)
    current = root;
```



Mapping Operation: goRoot() … *O*(1)

- **Precondition:**
  - size( ) > 0 .
- **Postcondition**:
  - The current node is now the root of the tree .
- **Code**:

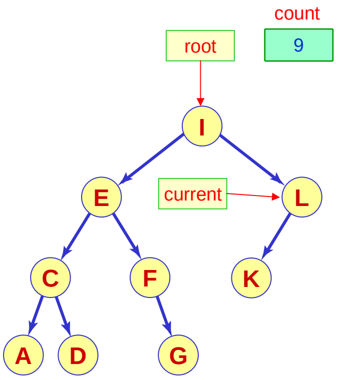
```
if (count > 0)
    current = root;
```



Mapping Operation: goLeft() … *O*(1)

- **Precondition:**
  - hasLeftChild( ) returns true.
- **Postcondition**:
  - The current pointer has been shifted down to point to the left child of the original current node.
- **Code**:

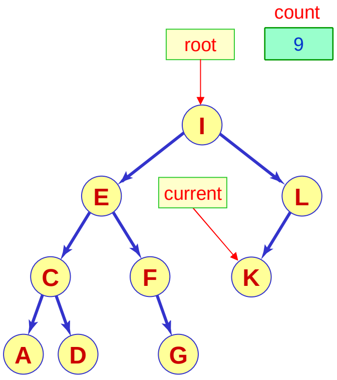
```
if (hasLeftChild())
    current = current.getLeft();
```



Mapping Operation: goLeft() … *O*(1)

- **Precondition:**
  - hasLeftChild( ) returns true.
- **Postcondition**:
  - The current pointer has been shifted down to point to the left child of the original current node.
- **Code**:

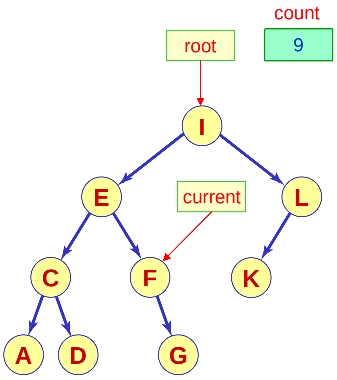
```
if (hasLeftChild())
    current = current.getLeft();
```



Mapping Operation: goRight() … *O*(1)

- **Precondition:**
  - hasRightChild( ) returns true.
- **Postcondition**:
  - The current pointer has been shifted down to point to the right child of the original current node.
- **Code**:

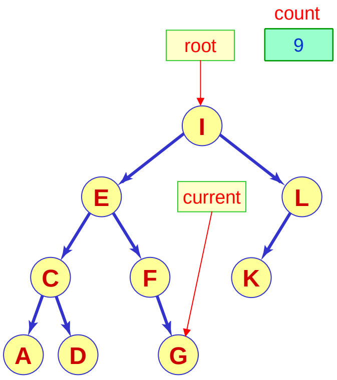
```
if (hasRightChild())
    current = current.getRight();
```

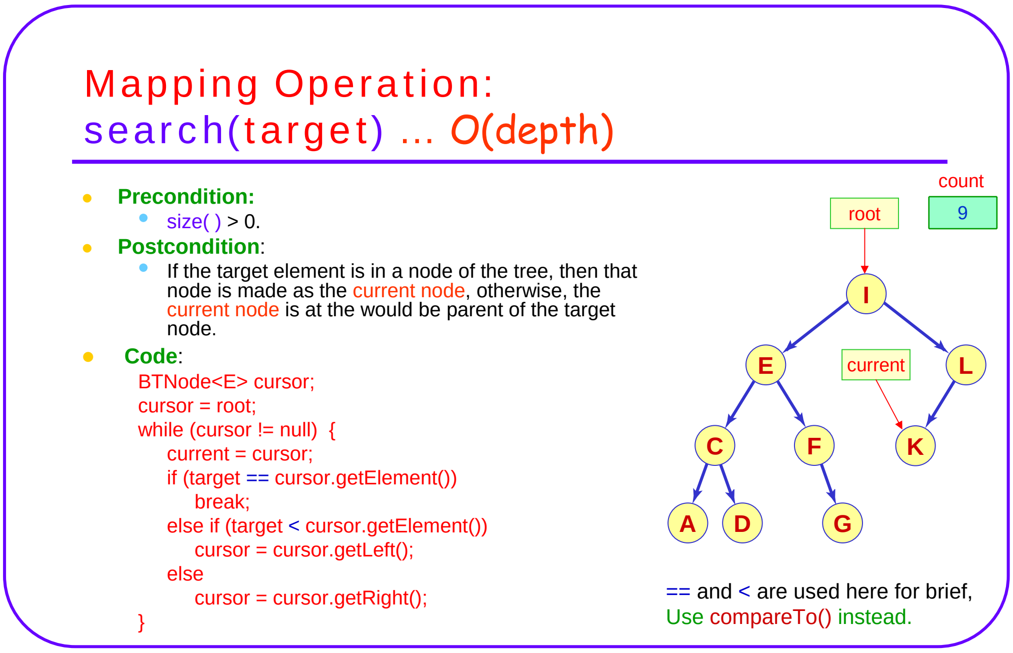


Mapping Operation: goRight() … *O*(1)

- **Precondition:**
  - hasRightChild( ) returns true.
- **Postcondition**:
  - The current pointer has been shifted down to point to the right child of the original current node.
- **Code**:

```
if (hasRightChild())
    current = current.getRight();
```



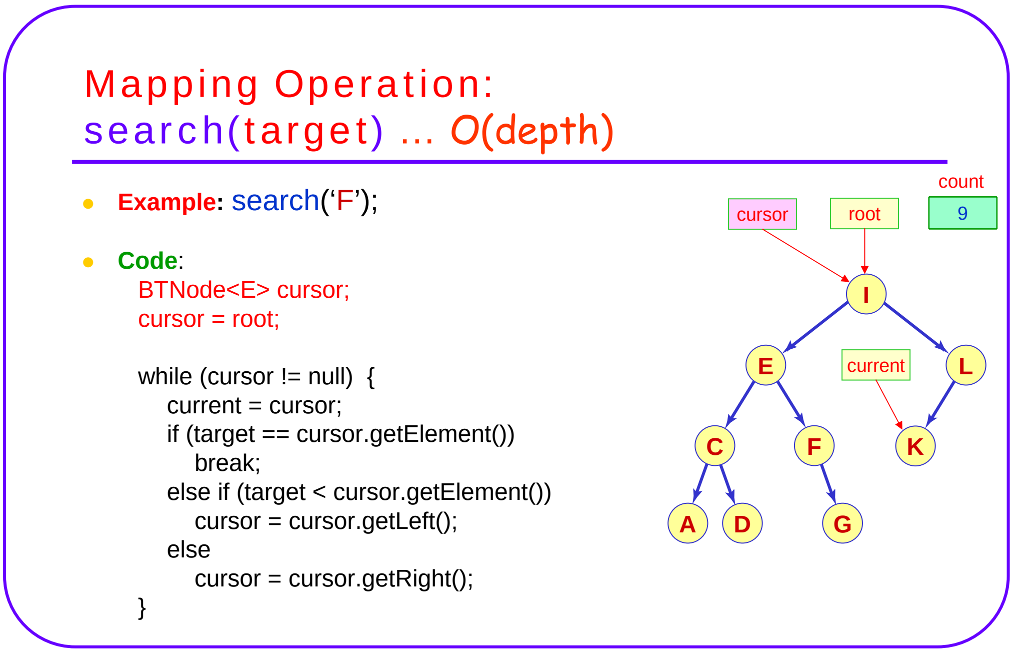


Mapping Operation: search(target) … O(depth)

- **Precondition:**
  - size( ) > 0.
- **Postcondition**:
  - If the target element is in a node of the tree, then that node is made as the current node, otherwise, the current node is at the would be parent of the target node.
- **Code**:

```
BTNode<E> cursor;
cursor = root;
while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```

== and < are used here for brief,
Use compareTo() instead.



Mapping Operation: search(target) … O(depth)

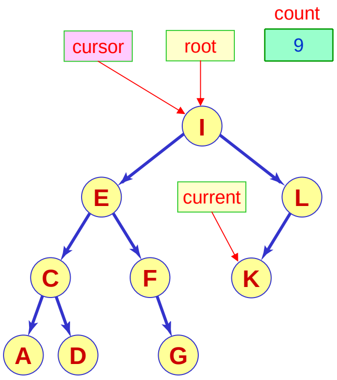
- **Example:** search(‘F’);
- **Code**:

```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```

Mapping Operation: search(target) … O(depth)

- **Example:** search('F');
- **Code**:

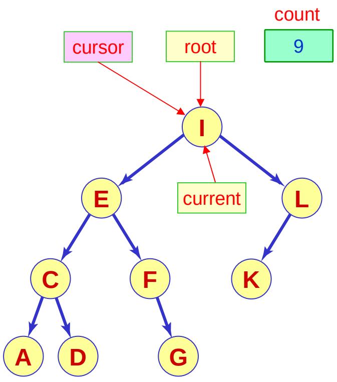
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation: search(target) … O(depth)

- **Example:** search('F');
- **Code**:

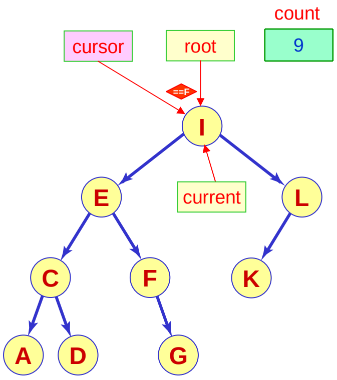
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation: search(target) … O(depth)

- **Example:** search(‘F’);
- **Code**:

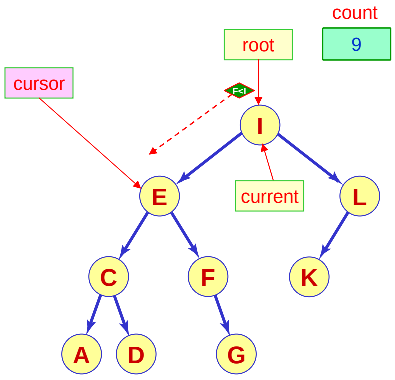
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation: search(target) … O(depth)

- **Example:** search(‘F’);
- **Code**:

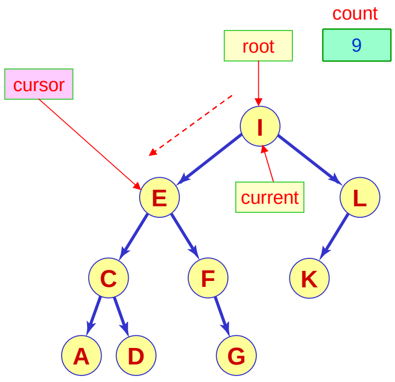
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation: search(target) … O(depth)

- **Example:** search('F');
- **Code:**

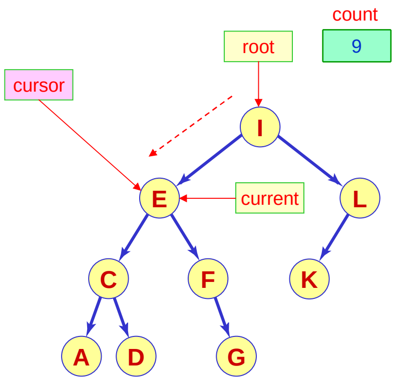
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation: search(target) … O(depth)

- **Example:** search('F');
- **Code:**

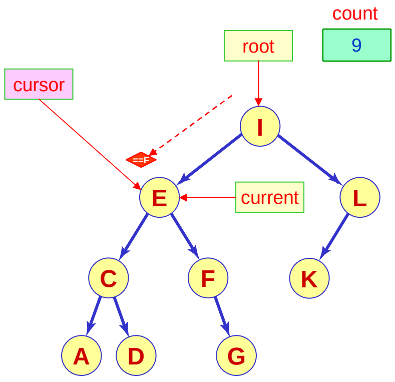
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation: search(target) … O(depth)

- **Example:** search(‘F’);
- **Code**:

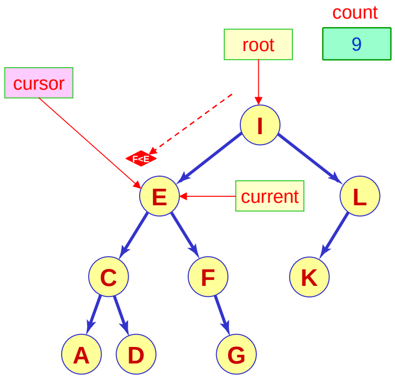
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation: search(target) … O(depth)

- **Example:** search(‘F’);
- **Code**:

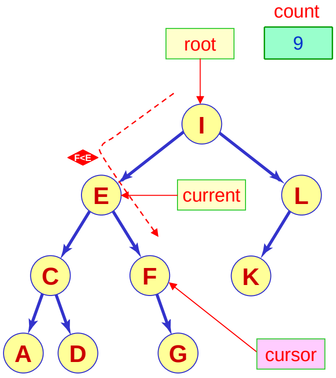
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation: search(target) … O(depth)

- **Example:** search(‘F’);
- **Code**:

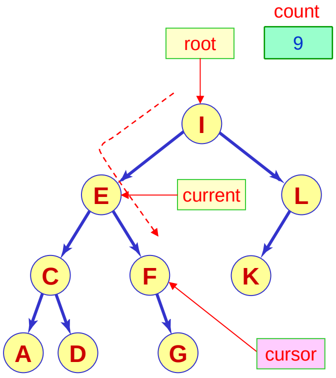
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation: search(target) … O(depth)

- **Example:** search(‘F’);
- **Code**:

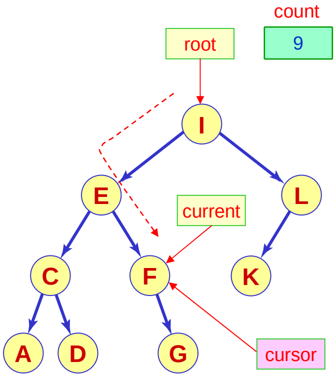
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation:
search(target) … O(depth)

- **Example:** search(‘F’);
- **Code**:

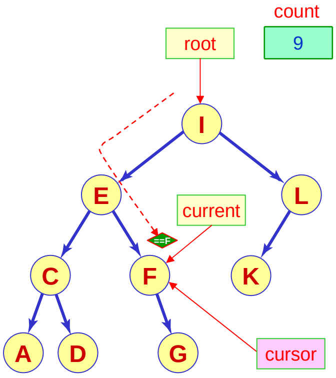
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation:
search(target) … O(depth)

- **Example:** search(‘F’);
- **Code**:

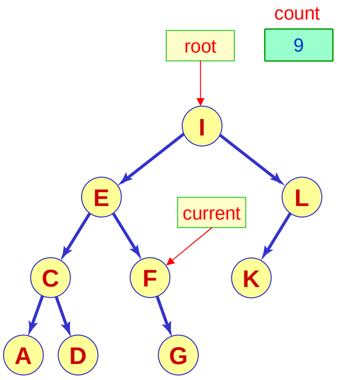
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation:
search(target) … O(depth)

- **Example:** search(‘F’);
- **Code**:

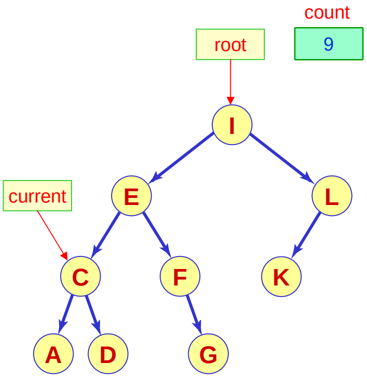
```
BTNode<E> cursor;
cursor = root;

while (cursor != null)  {
    current = cursor;
    if (target == cursor.getElement())
        break;
    else if (target < cursor.getElement())
        cursor = cursor.getLeft();
    else
        cursor = cursor.getRight();
}
```



Mapping Operation:
insert(entry) … O(depth)

- **Precondition:**
  - The entry's key is not in the tree.
- **Postcondition**:
  - The tree now has one more node containing the specified entry. The binary search tree's integrity is maintained and the new node is the current node.
- **Code**:

```
if (root == null) createFirstNode(entry);
else  {
   search(entry);
   if (entry < current.getElement()) addLeft(entry);
   if (entry > current.getElement()) addRight(entry);
}
```

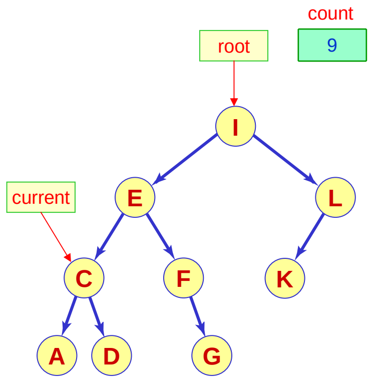


< and > are used here for brief,
Use compareTo() instead.

Mapping Operation: insert(entry) … O(depth)

- **Code Expanded:**

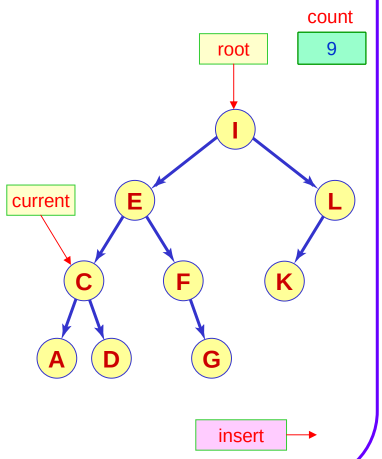
```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: insert(entry) … O(depth)

- **Example: insert(‘J’)**

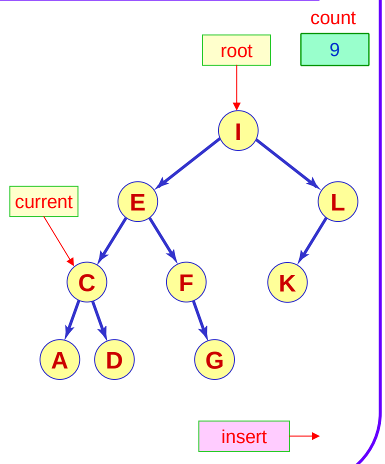
```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: insert(entry) … O(depth)

- **Example: insert('J')**

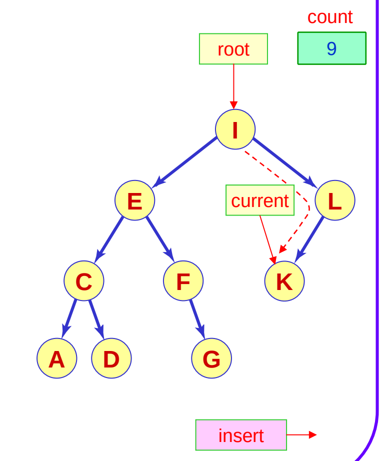
```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: insert(entry) … O(depth)

- **Example: insert('J')**

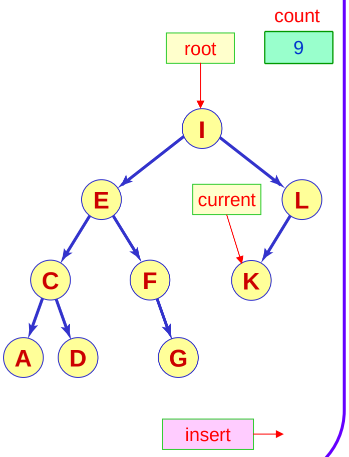
```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: insert(entry) … O(depth)

- **Example: insert('J')**

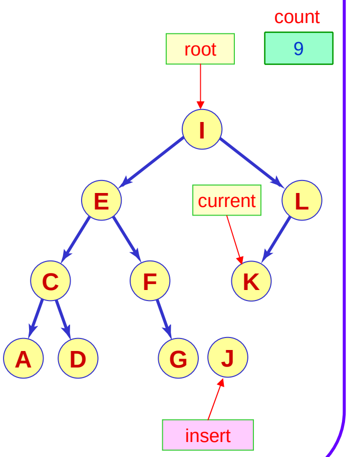
```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: insert(entry) … O(depth)

- **Example: insert('J')**

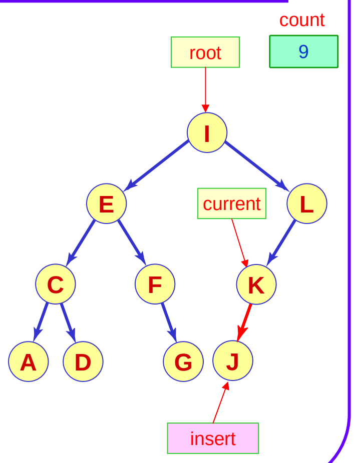
```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: insert(entry) … O(depth)

- **Example: insert(‘J’)**

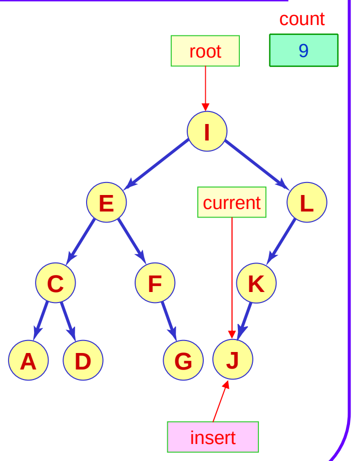
```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: insert(entry) … O(depth)

- **Example: insert(‘J’)**

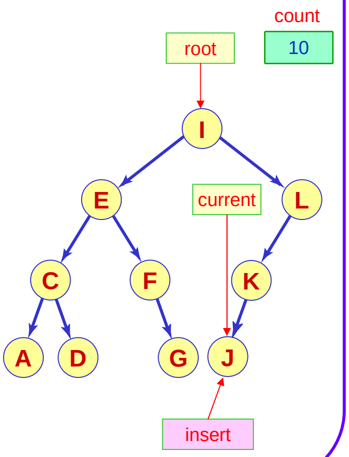
```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: insert(entry) ... O(depth)

- **Example: insert('J')**

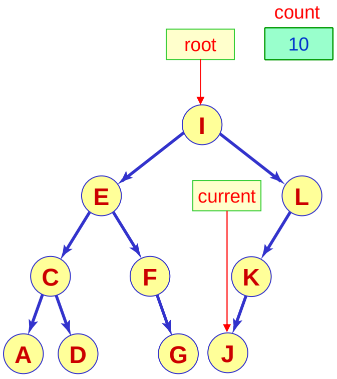
```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: insert(entry) ... O(depth)

- **Example: insert('J')**

```
BTNode<E> insert;
if (root == null)  {
   root = new BTNode<E>(entry);
   current = root;
   count++;
}
else  {
   search(entry);
   if (entry < current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setLeft(insert);
        current = insert;
        count++;
   }
   if (entry > current.getElement())  {
        insert = new BTNode<E>(entry);
        current.setRight(insert);
        current = insert;
        count++;
   }
}
```



Mapping Operation: remove(entry) … O(depth)

- **Precondition:**
  - size( ) > 0.
- **Postcondition**:
  - If entry is in the tree then it has been deleted from it. The tree now has one node less than before. The root of the tree is now the current node, the return value is true, and the binary search tree's integrity is maintained. If entry is not in the tree then nothing happens and the return value is false.

Mapping Operation: remove(entry) … O(depth)

- **Algorithm:**

  This is the most complex operation on a BST. Two cases exist:

Case1, If either subtree of the node to be deleted is empty:

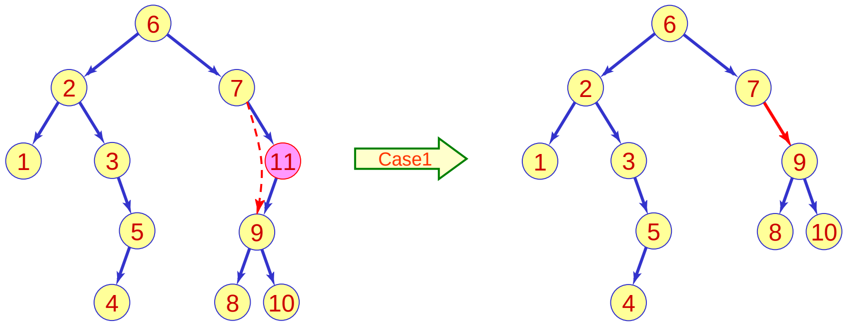
- Replace the pointer from its parent with the pointer to its non-empty subtree.

Case2, If both subtrees of the node, N, to be deleted are non-empty:

1. Find the rightmost node, R, in the left subtree of N.
   This is the node with the maximum key value on the left subtree of N, and it is guaranteed to have at least one non-empty subtree.
2. Move the contents of R to N.
3. Replace the pointer from R's parent with the pointer to R's left subtree.

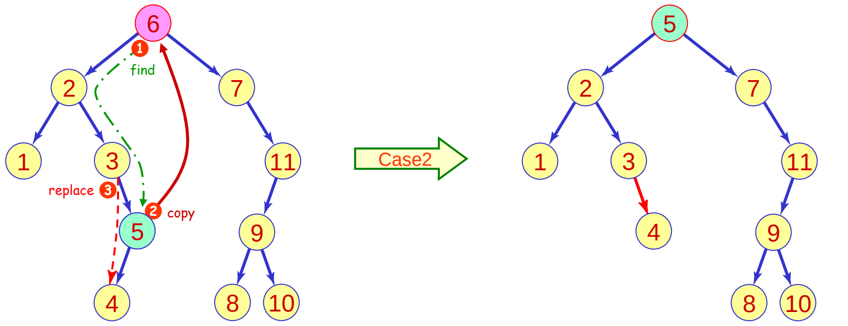
Mapping Operation: remove(entry) … *O*(depth)

- **Example 1: remove node with key value = 11.**



Mapping Operation: remove(entry) … *O*(depth)

- **Example 2: remove node with key value = 6.**



Mapping Operation: remove(entry) … O(depth)

- **Code**:

```
// This code depends on the findParent(entry) helper function, which is used to give
// access to the parent of the node containing the given entry. It has the following
// postcondition: If the entry is found in a node of the tree, then current points at that
// node and its parent link is returned. If the found node is the root, then both current
// and the returned link point at the root node. If the entry is not found in the tree, then
// current is set at the would be parent of entry and the value null is returned.

BTNode<E> parent, temp;

if (root == null)  return false;              // no nodes to delete

temp = current;
parent = findParent(entry);                   // find the entry and its parent node

if (parent == null)  {                        // the entry was not found in the tree
   current = temp;                            // restore current
   return false;
}
```

Mapping Operation: remove(entry) … O(depth)

```
// At this point, the node to be deleted is found in the tree. current is at the found node
// and parent is at its parent node. If entry is at the root then both pointers are at the root.
// Two cases exist:     1: The node has at least one null child.
//                      2. The node has two non-null children.

if (current.getRight() == null)  {                      // Case1, with right null child.
   // Attach its left child to its parent.
   if (current == parent) root = current.getLeft();
   if (current == parent.getLeft()) parent.setLeft(current.getLeft());
   if (current == parent.getRight()) parent.setRight(current.getLeft());

   // Delete the node.
   if (current != null)
       { current.setLeft(null);  current.setRight(null);  current.setElement(null); }
   current = root;
   count--;
   return true;
}
```

Mapping Operation: remove(entry) ... O(depth)

```
if (current.getLeft() == null)  {                          // Case1, with left null child.
   // Attach its right child to its parent.
   if (current == parent) root = current.getRight();
   if (current == parent.getLeft()) parent.setLeft(current.getRight());
   if (current == parent.getRight()) parent.setRight(current.getRight());

   // Delete the node.
   if (current != null)  {
        current.setLeft(null);
        current.setRight(null);
        current.setElement(null);
   }

   // Adjust current and count, then return true.
   current = root;
   count--;
   return true;
}
```

Mapping Operation: remove(entry) ... O(depth)

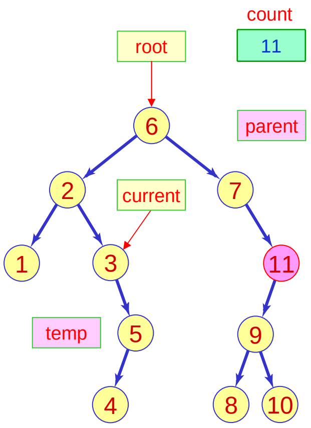
```
// Case2: The node has two non-null children.
// Find the node with the maximum element value in the left subtree,
// This maximum node is guaranteed to have at least one null child.
parent = current;  temp = current.getLeft();
while (temp.getRight() != null) { parent = temp; temp = temp.getRight(); }

// The maximum node is found. Copy its contents to subtree's root node.
current.setElement(temp.getElement());

// Attach its left child to its parent node.
if (temp == parent.getLeft()) parent.setLeft(temp.getLeft());
if (temp == parent.getRight()) parent.setRight(temp.getLeft());

// Delete the maximum node.
if (temp != null)
   { temp.setLeft(null);  temp.setRight(null);  temp.setElement(null);  temp = null; }
current = root;
count--;
return true;
```

The helper function: findParent(target)

```
BTNode<E> parent = root;
int where = 0;                                                    // used to trace current direction
current = root;
while (current != null)  {
   if (c.compare(target, current.getElement()) == 0)  return parent;       // target is found.
   else if (c.compare(target, current.getElement()) < 0)  {                // check left subtree.
        current = current.getLeft();                                       // move current left
        if (where == -1) parent = parent.getLeft();               // move parent following current
        if (where == +1) parent = parent.getRight();
        where = -1;                                               // record that current moved left
   }
   else  {                                                                 // check right subtree.
        current = current.getRight();                                      // move current right
        if (where == -1) parent = parent.getLeft();               // move parent following current
        if (where == +1) parent = parent.getRight();
        where = +1;                                               // record that current moved right
   }
}
return null;
```

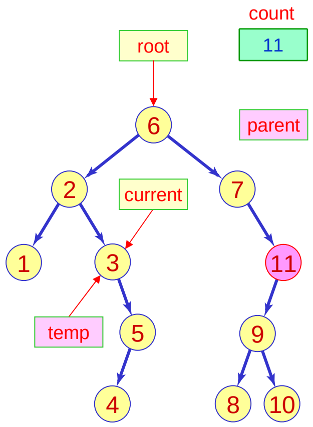
Example #1, for Operation remove(entry): —— remove(11)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



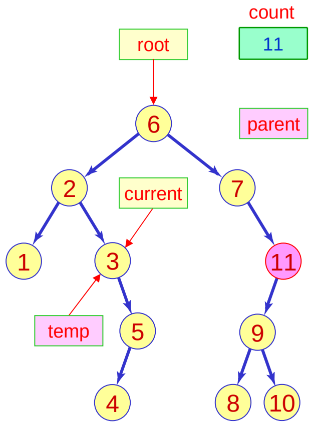
Example #1, for Operation remove(entry): —— remove(11)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



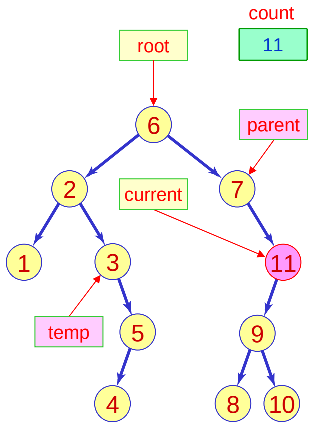
Example #1, for Operation remove(entry): —— remove(11)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



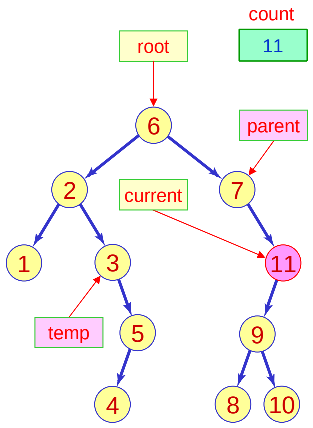
Example #1, for Operation remove(entry): — remove(11)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



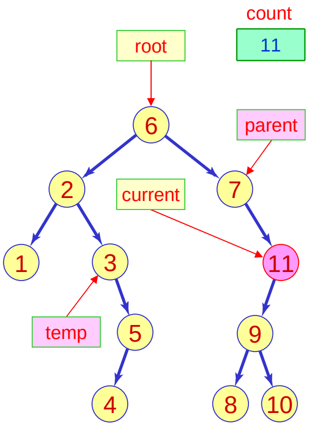
Example #1, for Operation remove(entry): — remove(11)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



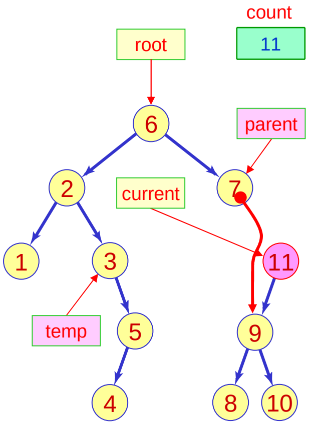
Example #1, for Operation remove(entry): —— remove(11)

```
if (current.getRight() == null)  {
   if (current == parent) root = current.getLeft();
   if (current == parent.getLeft())
       parent.setLeft(current.getLeft());
   if (current == parent.getRight())
       parent.setRight(current.getLeft());

   if (current != null)  {
       current.setLeft(null);
       current.setRight(null);
       current.setElement(null);
   }
   current = root;
   count--;
   return true;
}
```



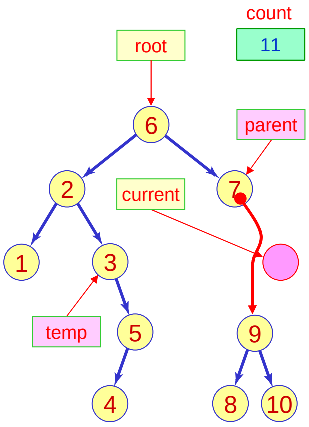
Example #1, for Operation remove(entry): —— remove(11)

```
if (current.getRight() == null)  {
   if (current == parent) root = current.getLeft();
   if (current == parent.getLeft())
       parent.setLeft(current.getLeft());
   if (current == parent.getRight())
       parent.setRight(current.getLeft());

   if (current != null)  {
       current.setLeft(null);
       current.setRight(null);
       current.setElement(null);
   }
   current = root;
   count--;
   return true;
}
```



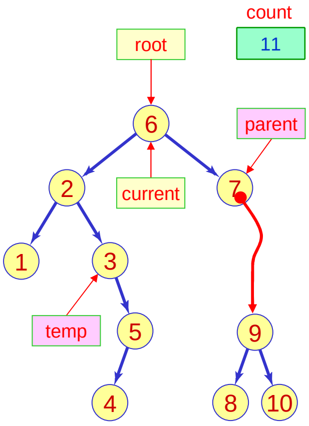
Example #1, for Operation remove(entry): —— remove(11)

```
if (current.getRight() == null)  {
  if (current == parent) root = current.getLeft();
  if (current == parent.getLeft())
     parent.setLeft(current.getLeft());
  if (current == parent.getRight())
     parent.setRight(current.getLeft());

  if (current != null)  {
     current.setLeft(null);
     current.setRight(null);
     current.setElement(null);
  }
  current = root;
  count--;
  return true;
}
```



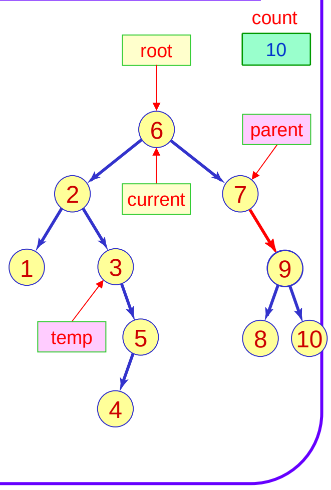
Example #1, for Operation remove(entry): —— remove(11)

```
if (current.getRight() == null)  {
  if (current == parent) root = current.getLeft();
  if (current == parent.getLeft())
     parent.setLeft(current.getLeft());
  if (current == parent.getRight())
     parent.setRight(current.getLeft());

  if (current != null)  {
     current.setLeft(null);
     current.setRight(null);
     current.setElement(null);
  }
  current = root;
  count--;
  return true;
}
```



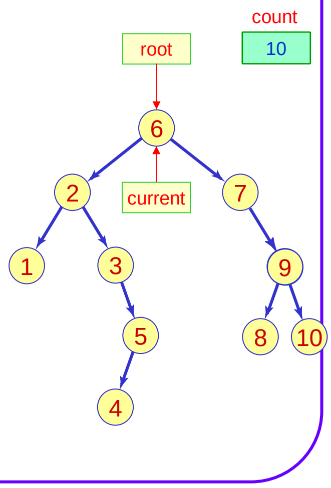
Example #1, for Operation remove(entry): —— remove(11)

```
if (current.getRight() == null)  {
   if (current == parent) root = current.getLeft();
   if (current == parent.getLeft())
       parent.setLeft(current.getLeft());
   if (current == parent.getRight())
       parent.setRight(current.getLeft());

   if (current != null)  {
       current.setLeft(null);
       current.setRight(null);
       current.setElement(null);
   }
   current = root;
   count--;
   return true;
}
```



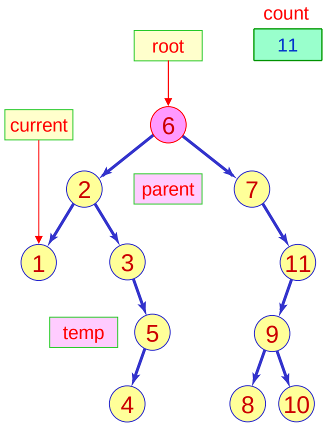
Example #1, for Operation remove(entry): —— remove(11)

```
if (current.getRight() == null)  {
   if (current == parent) root = current.getLeft();
   if (current == parent.getLeft())
       parent.setLeft(current.getLeft());
   if (current == parent.getRight())
       parent.setRight(current.getLeft());

   if (current != null)  {
       current.setLeft(null);
       current.setRight(null);
       current.setElement(null);
   }
   current = root;
   count--;
   return true;
}
```



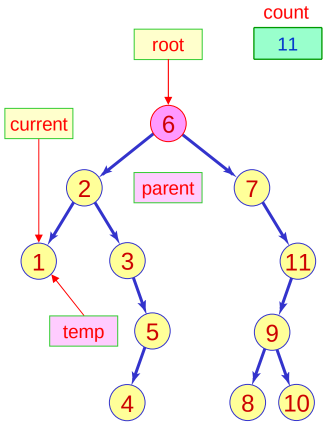
Example #2, for Operation remove(entry): —— remove(6)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



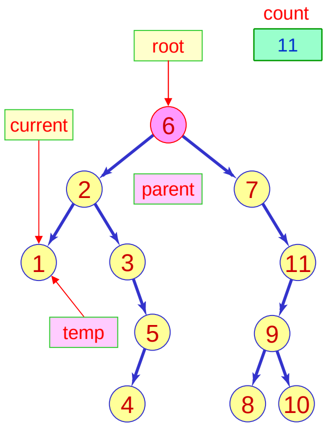
Example #2, for Operation remove(entry): —— remove(6)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



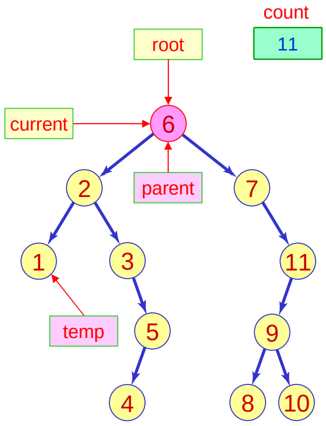
Example #2, for Operation remove(entry): —— remove(6)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



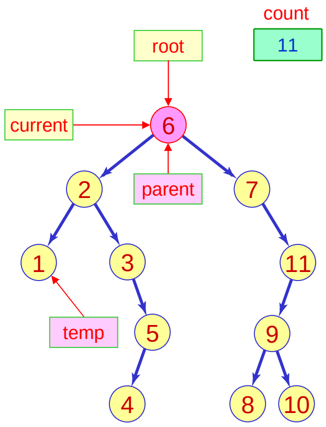
Example #2, for Operation remove(entry): —— remove(6)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



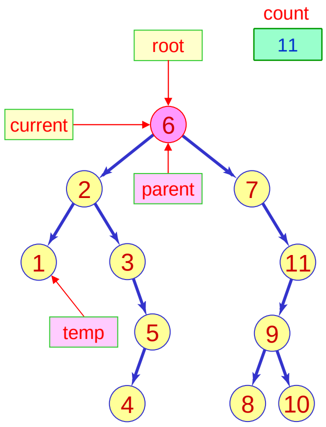
Example #2, for Operation remove(entry): —— remove(6)

```
BTNode<E> parent, temp;
if (root == null)  return false;

temp = current;
parent = findParent(entry);

if (parent == null)  {
   current = temp;
   return false;
}
```



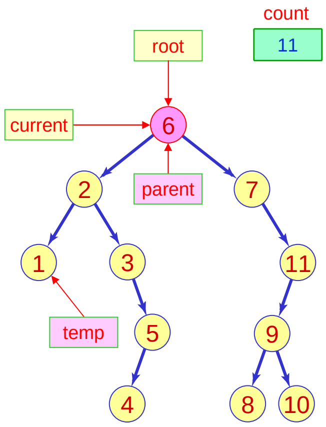
Example #2, for Operation remove(entry): —— remove(6)

```
if (current.getRight() == null)  {
   if (current == parent) root = current.getLeft();
   if (current == parent.getLeft())
       parent.setLeft(current.getLeft());
   if (current == parent.getRight())
       parent.setRight(current.getLeft());

   if (current != null)  {
       current.setLeft(null);
       current.setRight(null);
       current.setElement(null);
   }
   current = root;
   count--;
   return true;
}
```



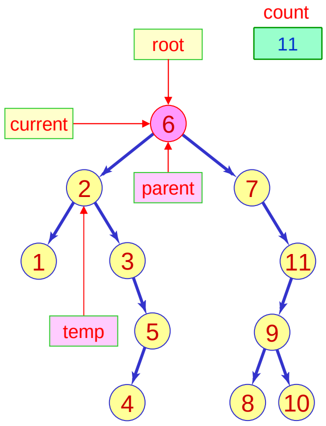
Example #2, for Operation remove(entry): —— remove(6)

```
if (current.getLeft() == null)  {
   if (current == parent) root = current.getRight();
   if (current == parent.getLeft())
       parent.setLeft(current.getRight());
   if (current == parent.getRight())
       parent.setRight(current.getRight());

   if (current != null)  {
       current.setLeft(null);
       current.setRight(null);
       current.setElement(null);
   }
   current = root;
   count--;
   return true;
}
```



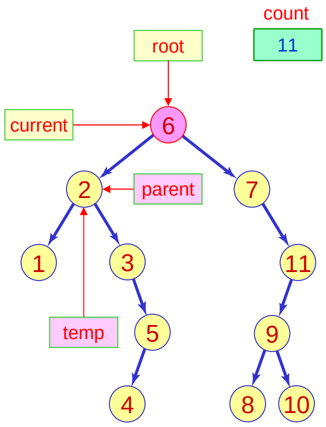
Example #2, for Operation remove(entry): —— remove(6)

```
parent = current;
temp = current.getLeft();
while (temp.getRight() != null)  {
   parent = temp;
   temp = temp.getRight();
}

current.setElement(temp.getElement());

if (temp == parent.getLeft())
   parent.setLeft(temp.getLeft());

if (temp == parent.getRight())
   parent.setRight(temp.getLeft());
```



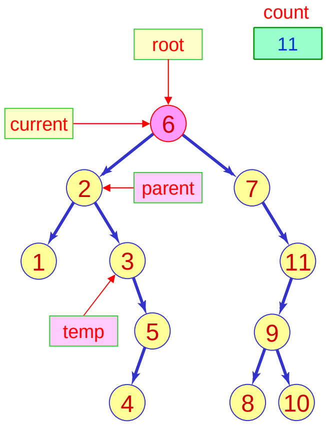
Example #2, for Operation remove(entry): —— remove(6)

```
parent = current;
temp = current.getLeft();
while (temp.getRight() != null)  {
   parent = temp;
   temp = temp.getRight();
}

current.setElement(temp.getElement());

if (temp == parent.getLeft())
   parent.setLeft(temp.getLeft());

if (temp == parent.getRight())
   parent.setRight(temp.getLeft());
```



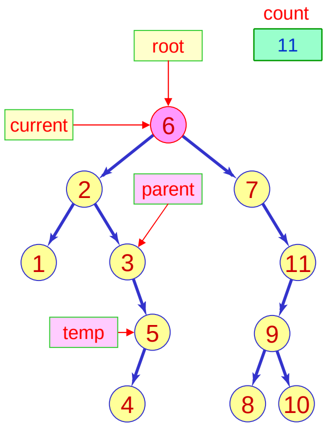
Example #2, for Operation remove(entry): —— remove(6)

```
parent = current;
temp = current.getLeft();
while (temp.getRight() != null)  {
   parent = temp;
   temp = temp.getRight();
}

current.setElement(temp.getElement());

if (temp == parent.getLeft())
   parent.setLeft(temp.getLeft());

if (temp == parent.getRight())
   parent.setRight(temp.getLeft());
```



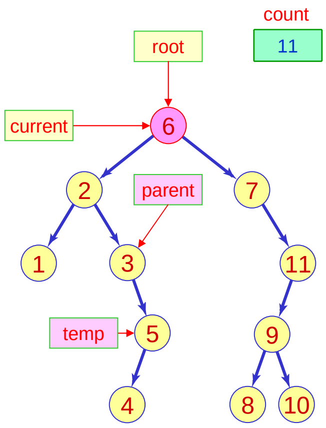
Example #2, for Operation remove(entry): —— remove(6)

```
parent = current;
temp = current.getLeft();
while (temp.getRight() != null)  {
  parent = temp;
  temp = temp.getRight();
}

current.setElement(temp.getElement());

if (temp == parent.getLeft())
  parent.setLeft(temp.getLeft());

if (temp == parent.getRight())
  parent.setRight(temp.getLeft());
```



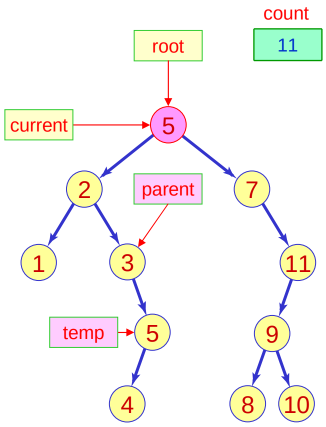
Example #2, for Operation remove(entry): —— remove(6)

```
parent = current;
temp = current.getLeft();
while (temp.getRight() != null)  {
  parent = temp;
  temp = temp.getRight();
}

current.setElement(temp.getElement());

if (temp == parent.getLeft())
  parent.setLeft(temp.getLeft());

if (temp == parent.getRight())
  parent.setRight(temp.getLeft());
```



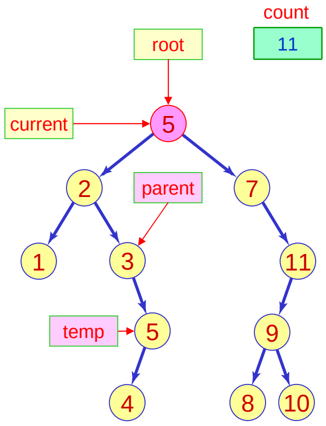
Example #2, for Operation remove(entry): —— remove(6)

```
parent = current;
temp = current.getLeft();
while (temp.getRight() != null)  {
  parent = temp;
  temp = temp.getRight();
}

current.setElement(temp.getElement());

if (temp == parent.getLeft())
  parent.setLeft(temp.getLeft());

if (temp == parent.getRight())
  parent.setRight(temp.getLeft());
```



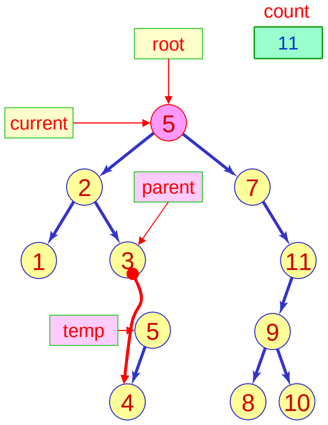
Example #2, for Operation remove(entry): —— remove(6)

```
parent = current;
temp = current.getLeft();
while (temp.getRight() != null)  {
  parent = temp;
  temp = temp.getRight();
}

current.setElement(temp.getElement());

if (temp == parent.getLeft())
  parent.setLeft(temp.getLeft());

if (temp == parent.getRight())
  parent.setRight(temp.getLeft());
```



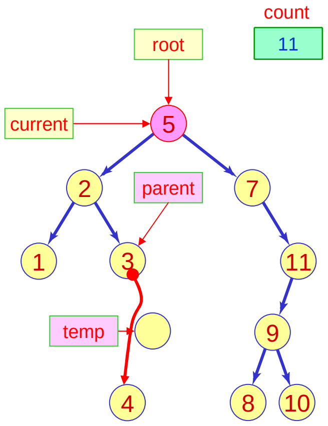
Example #2, for Operation remove(entry): —— remove(6)

```
parent = current;
temp = current.getLeft();
while (temp.getRight() != null)  {
   parent = temp;
   temp = temp.getRight();
}

current.setElement(temp.getElement());

if (temp == parent.getLeft())
   parent.setLeft(temp.getLeft());

if (temp == parent.getRight())
   parent.setRight(temp.getLeft());
```



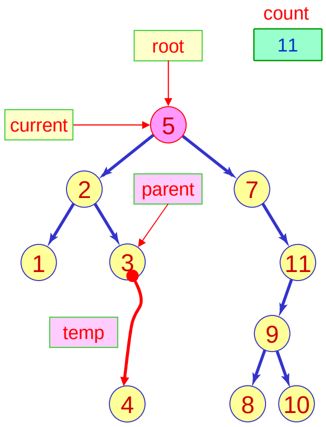
Example #2, for Operation remove(entry): —— remove(6)

```
if (temp != null)  {
   temp.setLeft(null);
   temp.setRight(null);
   temp.setElement(null);
   temp = null;
}

current = root;
count--;
return true;
```



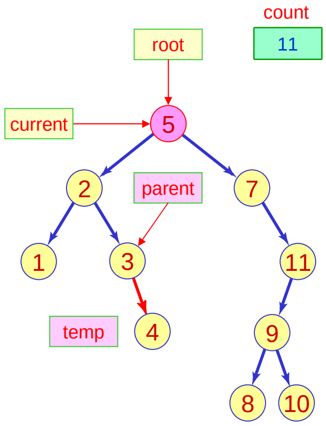
Example #2, for Operation remove(entry): —— remove(6)

```
if (temp != null)  {
   temp.setLeft(null);
   temp.setRight(null);
   temp.setElement(null);
   temp = null;
}

current = root;
count--;
return true;
```



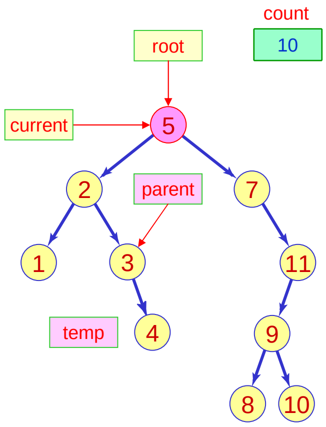
Example #2, for Operation remove(entry): —— remove(6)

```
if (temp != null)  {
   temp.setLeft(null);
   temp.setRight(null);
   temp.setElement(null);
   temp = null;
}

current = root;
count--;
return true;
```



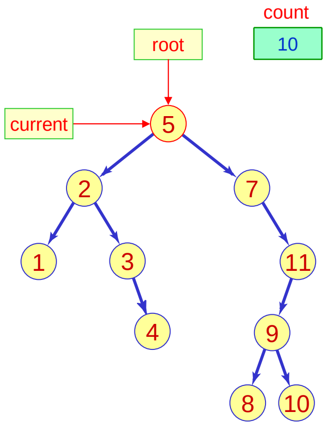
Example #2, for Operation remove(entry): —— remove(6)

```
if (temp != null)  {
   temp.setLeft(null);
   temp.setRight(null);
   temp.setElement(null);
   temp = null;
}

current = root;
count--;
return true;
```



Example #2, for Operation remove(entry): —— remove(6)

```
if (temp != null)  {
   temp.setLeft(null);
   temp.setRight(null);
   temp.setElement(null);
   temp = null;
}

current = root;
count--;
return true;
```



Mapping Operation: replace(entry) … O(depth)

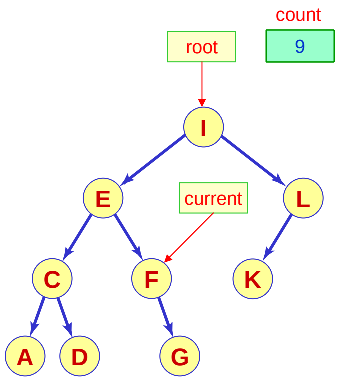
- **Precondition:**
  - size( ) > 0.
- **Postcondition**:
  - The data at the current node has been replaced with the new entry, and it has been moved to its proper location in the tree so that the binary search tree's integrity is maintained.
- **Code**:

```
if (current != null)  {
    remove(current.getElement());
    insert(entry);
}
```

Mapping Operation: replace(entry) … O(depth)

- **Example**: replace(‘B’);
- **Code**:

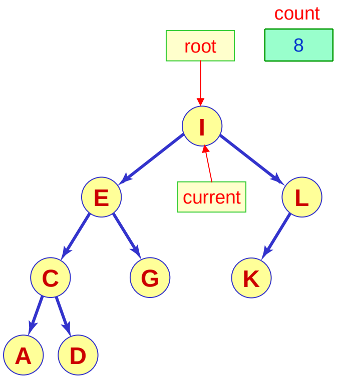
```
if (current != null)  {
    remove(current.getElement());
    insert(entry);
}
```



Mapping Operation: replace(entry) ... O(depth)

- **Example**: replace('B');
- **Code**:

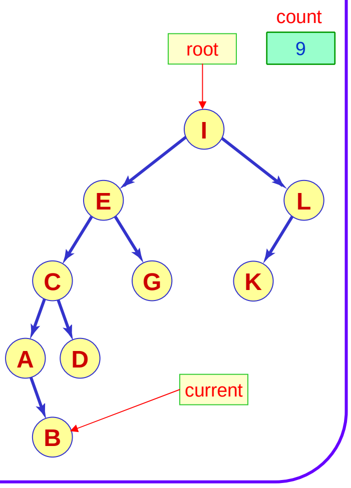
```
if (current != null)  {
    remove(current.getElement());
    insert(entry);
}
```



Mapping Operation: replace(entry) ... O(depth)

- **Example**: replace('B');
- **Code**:

```
if (current != null)  {
    remove(current.getElement());
    insert(entry);
}
```

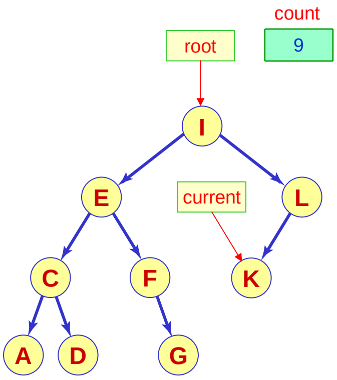


Mapping Operation: retrieve() ... *O*(1)

- **Precondition:**
  - size( ) > 0.
- **Postcondition**:
  - The return value is the element from the current node.
- **Code**:

```
if (current != null)
    return current.getElement();
```

- **Example**: retrieve() ➔ returns ‘K’.



Mapping Operation: Tree Traversal print(p, depth) ... *O*(*n*)

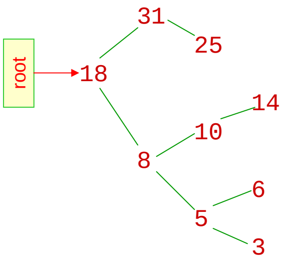
- **Precondition**:
  - p is a pointer to a node in a binary tree or null to indicate the empty tree.
  - If the pointer is not null, then depth is the level of the node at p.
- **Postcondition**:
  - If p is non-null, then the elements of node p and all its descendants have been written out, using a backward in-order traversal. Each node is indented four times its depth, so to resemble the tree shape.
- **Code**:

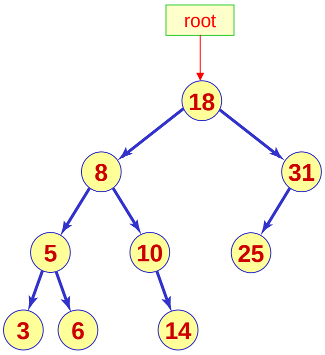
```
String s = “”;
if (p != null)  {
    print(p.getRight( ), depth+1);            // Do the right branch first
    for (int i=0; i<depth; i++)  s += “    “;  // Indent 4*depth spaces
    System.out.println(s+p.getElement());  // Print the node contents
    print(p.getLeft( ),  depth+1);             // Do the left branch last
}
```

Recursive

Mapping Operation: Tree Traversal print(p, depth) ... *O*(n)

- **Example**: print (root,0);
- **Result**:





Mapping Operation: treeClear(p) ... *O*(n)

- **Precondition**:
  - p is the root pointer of a binary tree which may be null for the empty tree.
- **Postcondition**:
  - All nodes at the root or below have been returned to the heap, and root have been set to null. The tree is now empty.
- **Code**:

```
if (p != null)  {
    treeClear( p.getLeft( ) );
    treeClear( p.getRight( ) );
    if (p == root)  {
        root = null;
        current == null;
        count = 0;
    }
    p.element = null;
    p := null;
}
```

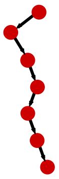
Recursive

BST Implementation Performance Issues

- Most of the operations of the BST are O(1).
- All traversal operations are O(n).
- The most important operation of the BST is the search operation. We define the following:
  - The search length of a node:
    - The number of nodes examined to find the node.
  - The search length of a tree:
    - The average of the search lengths for all of its nodes
- The time complexity of the search operation depends on the height of the tree rather than its size.
- There are three more operations that need to find a node before proceeding, these are: insert, remove, and replace. Their time complexity depends on the time complexity of the search operation.

BST Implementation Performance Issues (cont.)

- Binary trees have many shapes, but can be grouped into one of the following categories:
  1. The degenerate binary tree: Where each node has exactly one child, except the only leaf node.
  2. The random binary tree: Which is formed by adding elements in random order to an empty BST.
  3. The minimum height binary tree: as given before.
  4. The full binary tree: as given before.



BST Implementation Performance Issues (cont.)

Search lengths for binary trees of size n:

Tree Type	Average Search Length	Maximum Search Length of any node
Degenerate	½ (n+1)	n
Random	1.4 $\log_2$ (n+1)	n
Minimum Height	$\log_2$ (n+1)	$\log_2$ (n+1)
Full	$\log_2$ (n+1)	$\log_2$ (n+1)

The search operation for a random BST is O($\log_2$ n)

Binary Search Tree Application in Sorting a Linear List -- tree sort

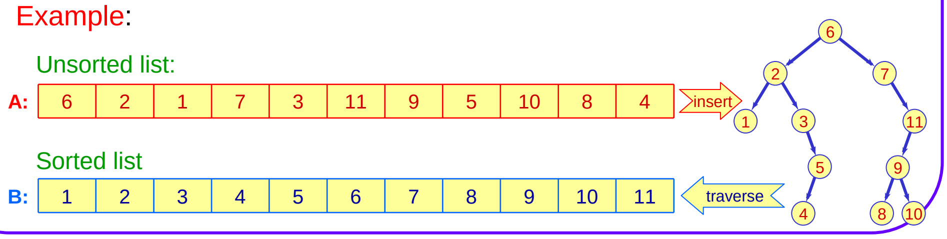
- Beside its use in searching, a BST can also be used for sorting:

Tree Sort Algorithm:

Given a linear list of items, A, produce a sorted list of them, B.

1. Form a binary search tree by inserting the items from A, one by one, into an empty BST. O(n log n)
2. Use the inorder traversal operation on the BST to get the sorted list, B. O(n)

Example:



Unsorted list:

A:	6	2	1	7	3	11	9	5	10	8	4

Sorted list

B:	1	2	3	4	5	6	7	8	9	10	11