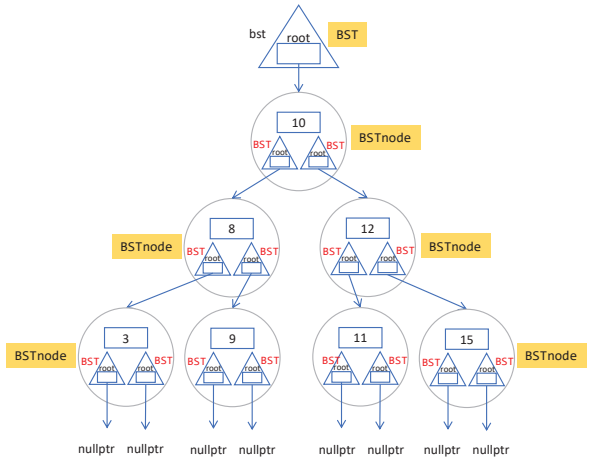
Binary Search Tree

Details of “remove” Member Function

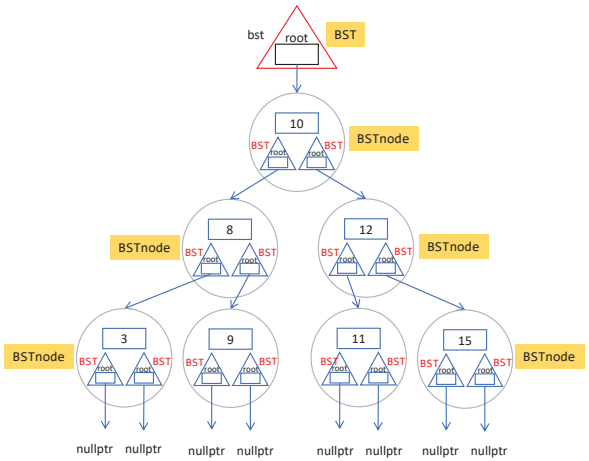
A BST which consists of 7 data




```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```


Suppose we want to remove 10, i.e. passing 10 to remove member function and accepted using x

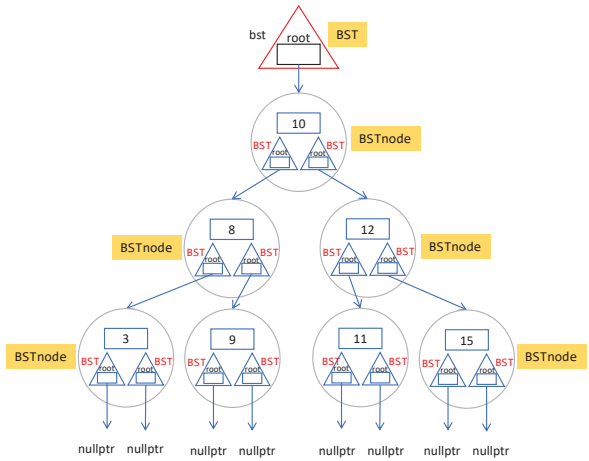
x 10




```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

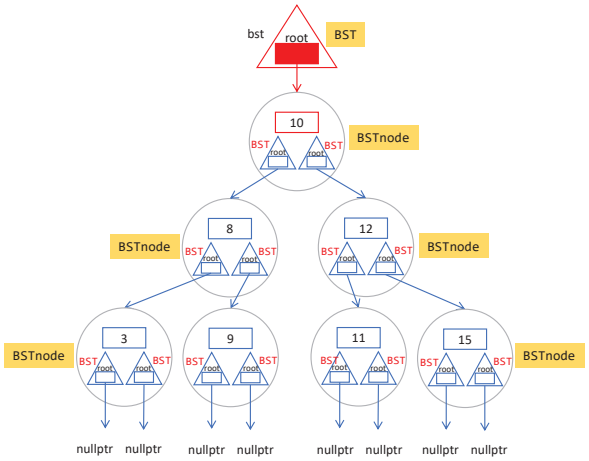

Check whether the current tree is empty. As the current tree is non-empty, is_empty() returns false

x 10




```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty()) false
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

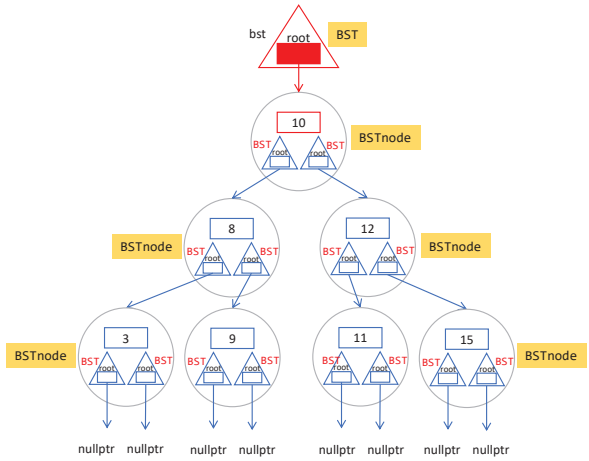
x is 10, root->value is 10, so x < root->value returns false



x 10

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value) false
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

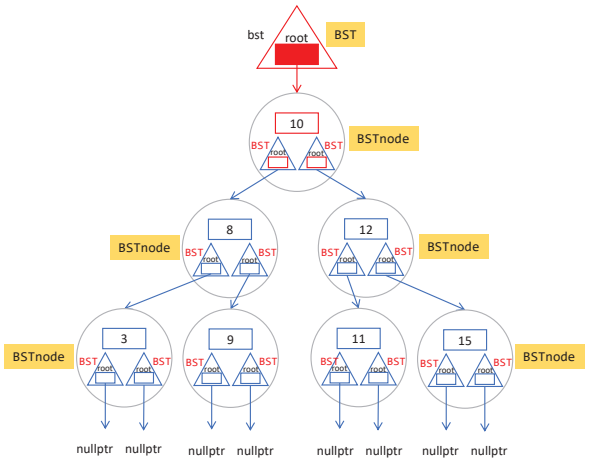

x is 10, root->value is 10, so x > root->value returns false



x 10

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value) false
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

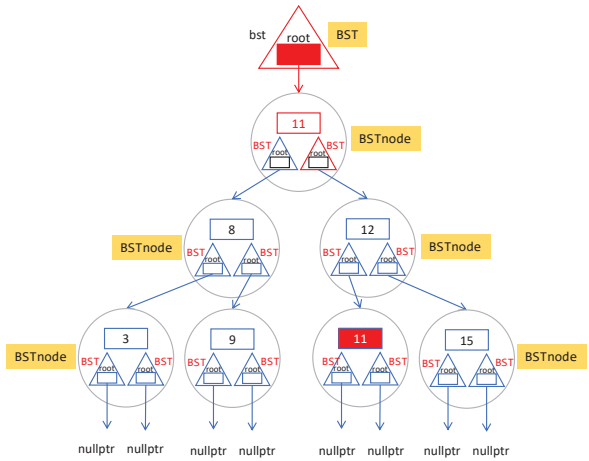

As root->left.root is not nullptr and root->right.root is also not nulltpr, root->left.root && root->right.root is true



x 10

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) { true
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

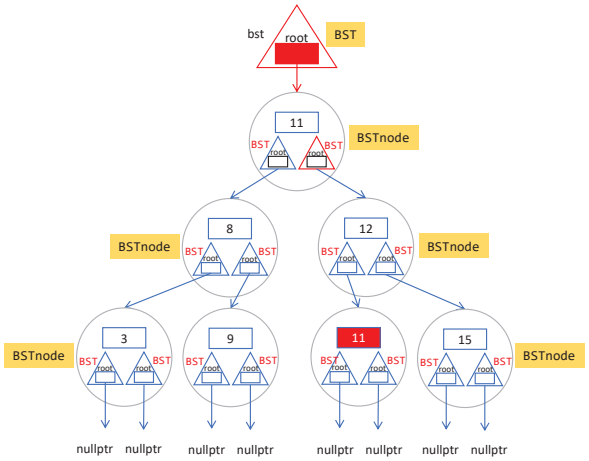

root->right.find_min() is 11. 11 is assigned to root->value



x 10

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min(); 11
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

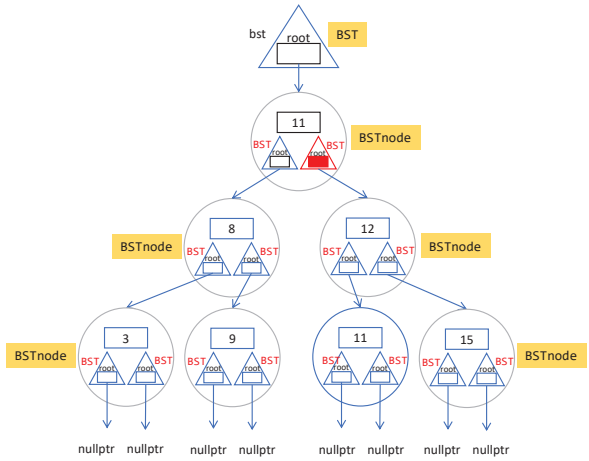
Remove the node with 11 using the right BST in the node pointed by the root pointer



x 10

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {                                  11
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

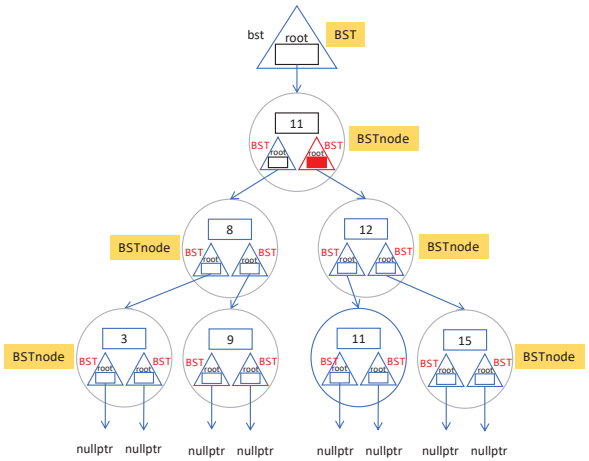

Remove 11, i.e. passing 11 to remove member function and accepted using x



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

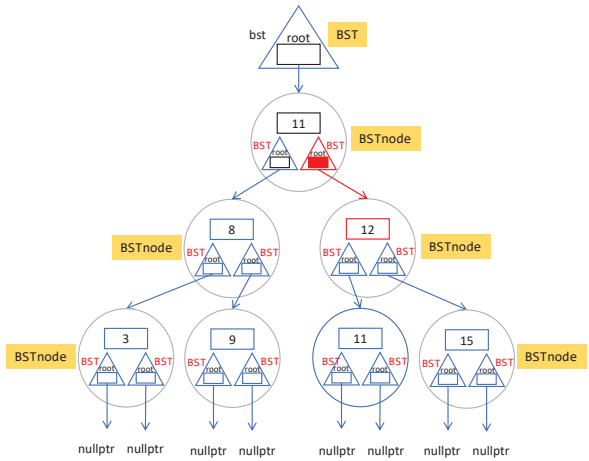

Check whether the current tree is empty. As the current tree is non-empty, is_empty() returns false



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty()) false
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

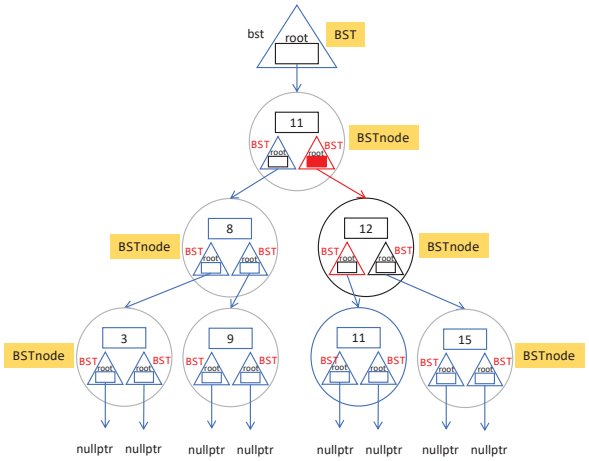

x is 11, root->value is 12, so x < root->value returns true



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value) true
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

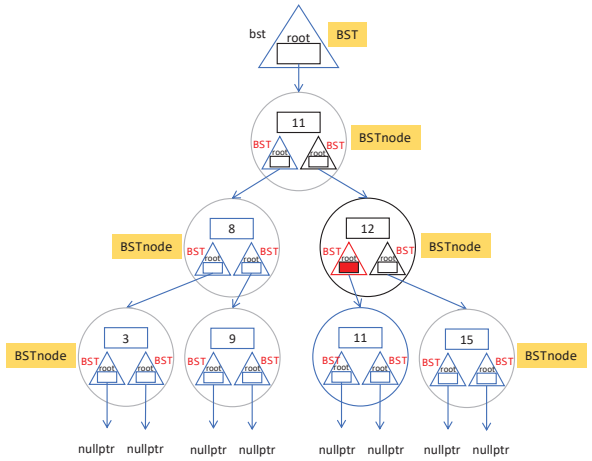
Remove the node with 11 using the right BST in the node pointed by the root pointer



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

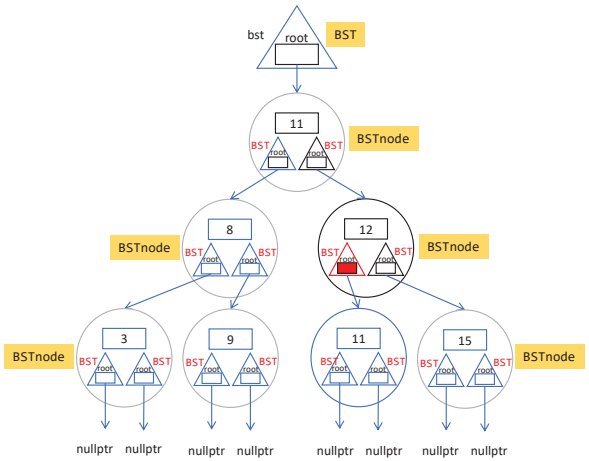

Remove 11, i.e. passing 11 to remove member function and accepted using x



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

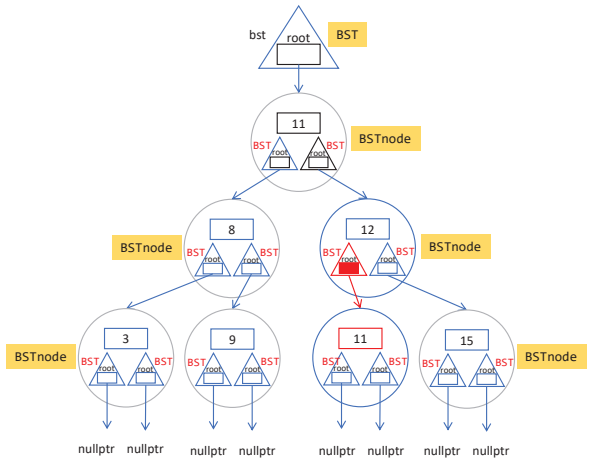

Check whether the current tree is empty. As the current tree is non-empty, is_empty() returns false



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty()) false
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

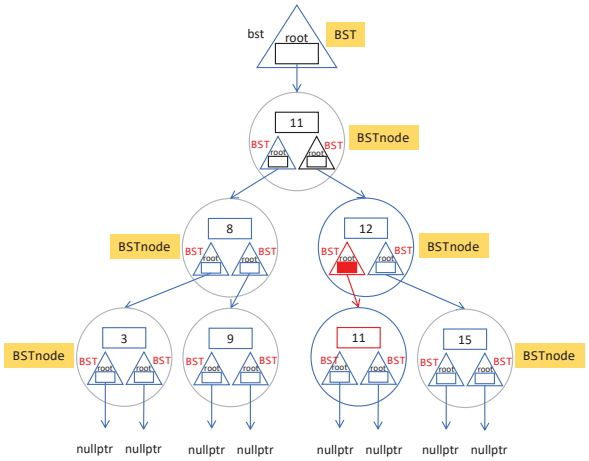

x is 11, root->value is 11, so x < root->value returns false



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value) false
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

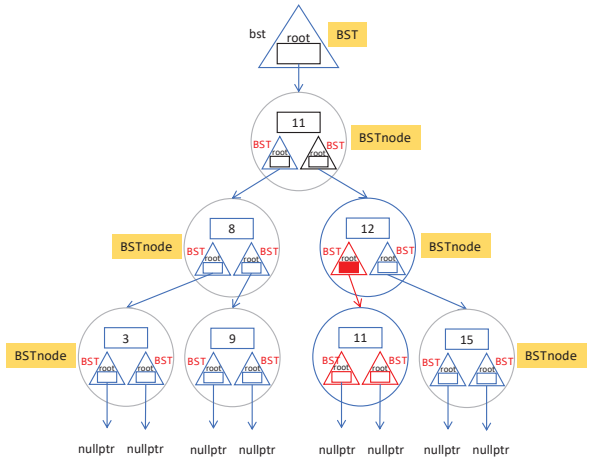
x is 11, root->value is 11, so x > root->value returns false



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value) false
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

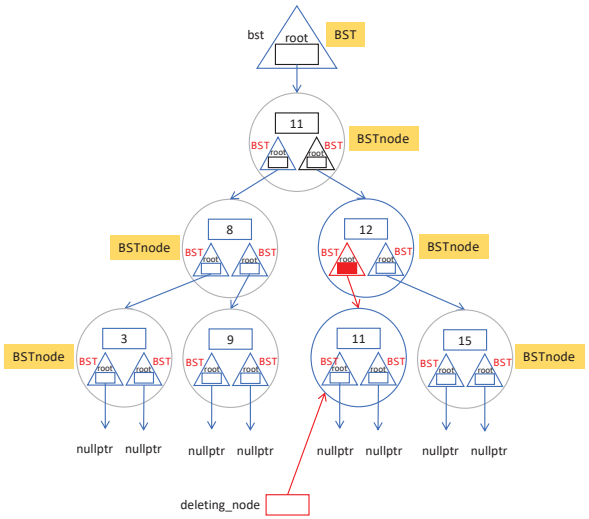

As root->left.root is nullptr and root->right.root is also nulltpr, root->left.root && root->right.root is false



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) { false
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

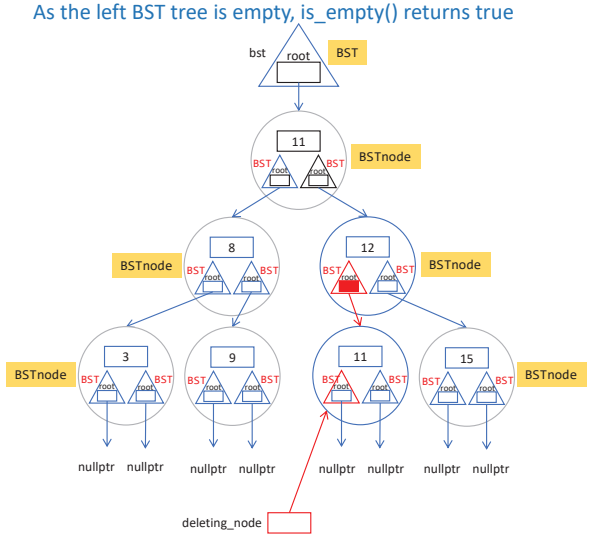

Make deleting_node pointer points at the node pointed by root



x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```


Check whether the left BST tree in the node pointed by root is empty.
As the left BST tree is empty, is_empty() returns true

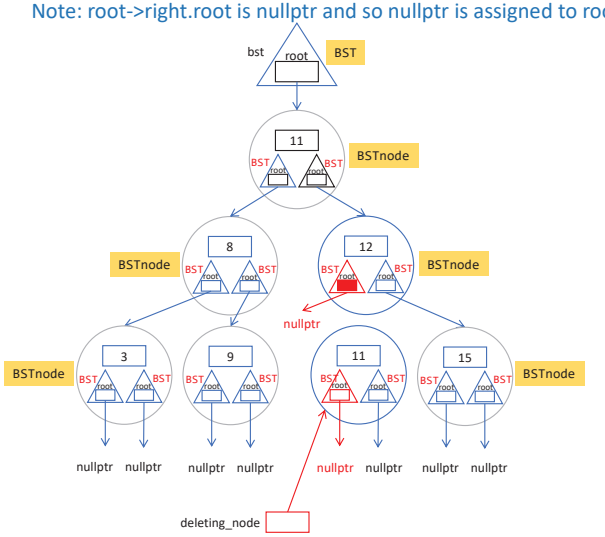


x 11

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root; true
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

As the condition is true, assign root->right.root to root
Note: root->right.root is nullptr and so nullptr is assigned to root

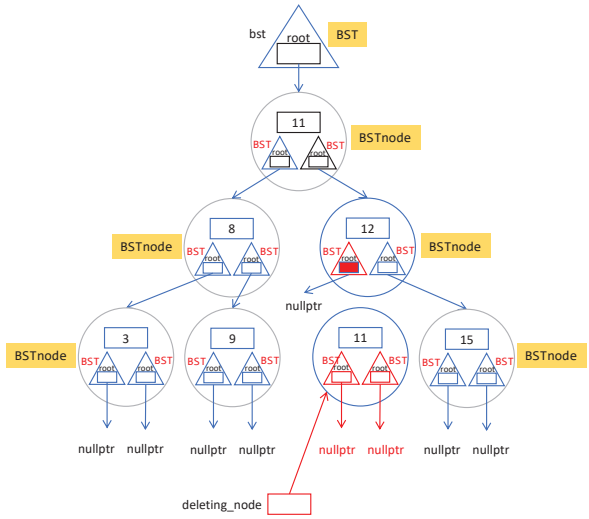
x 11




```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```


Set the root of both left and right BST trees in the node pointed by deleting_node to nullptr

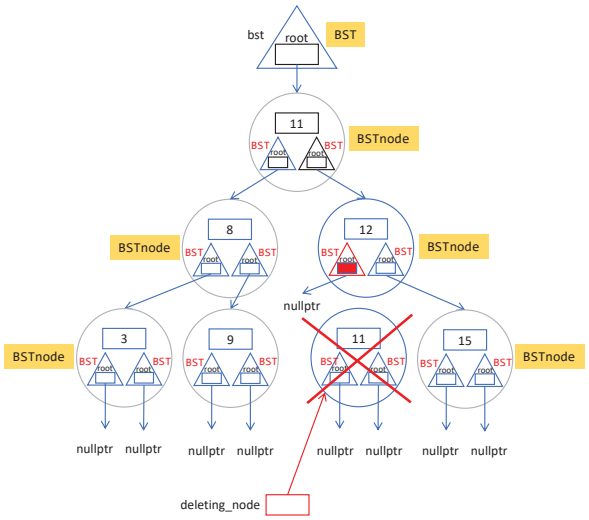
x 11




```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```


Deallocate the node pointed by deleting_node

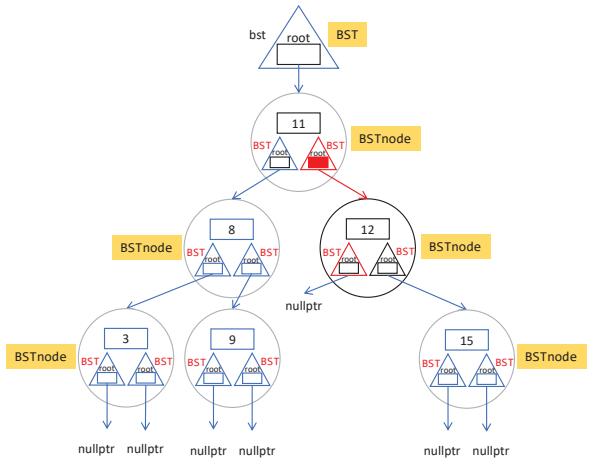
x 11




```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

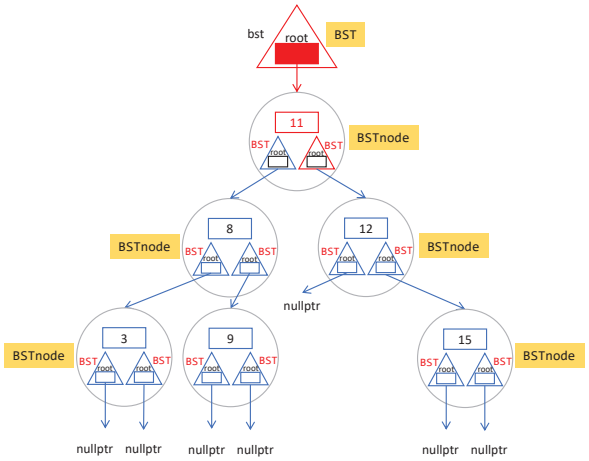

root->left.remove(x) is done

x 11




```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x); Done
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

root->right.remove(root->value) is done

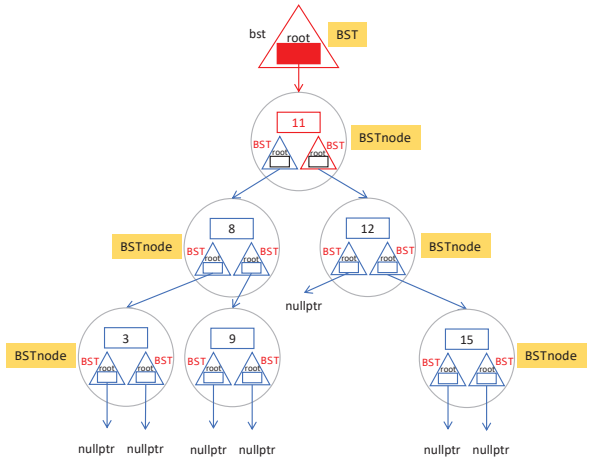


x 10

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

Done

All done!!!



x 10

```
template /* File: bst-remove.cpp */
void BST::remove(const T& x) {
  if (is_empty())
    return;
  if (x < root->value)
    root->left.remove(x);
  else if (x > root->value)
    root->right.remove(x);
  else if (root->left.root && root->right.root) {
    root->value = root->right.find_min();
    root->right.remove(root->value);
  } else {
    BSTnode* deleting_node = root;
    root = (root->left.is_empty()) ?
           root->right.root : root->left.root;
    deleting_node->left.root =
    deleting_node->right.root = nullptr;
    delete deleting_node;
  }
}
```

All Done