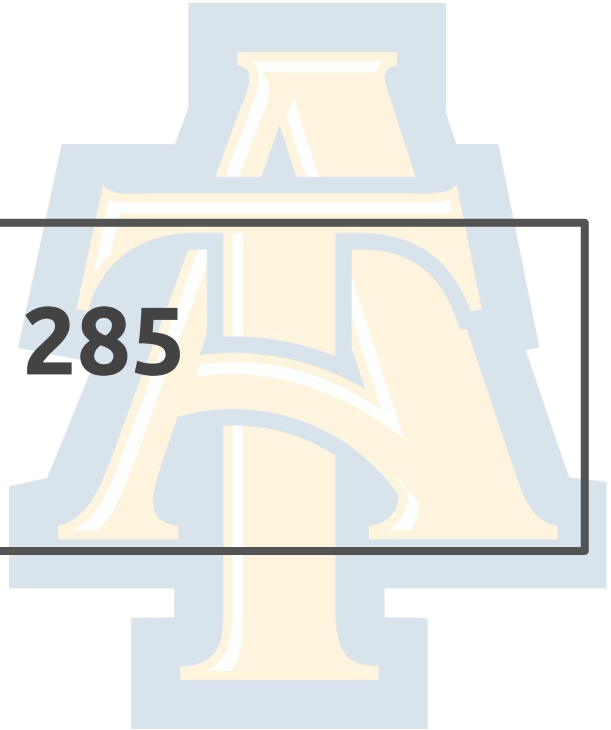
COMP - 285

Advanced Analysis of Algorithms

Welcome to COMP 285

Lecture 10: Binary Trees



Chris Lucas (cflucas@ncat.edu)

HW3 was released!

Due Tuesday @ 11:59pm ET

HW2 Grades!

By end of week!

(with solutions)

Extra Credit Opportunities!

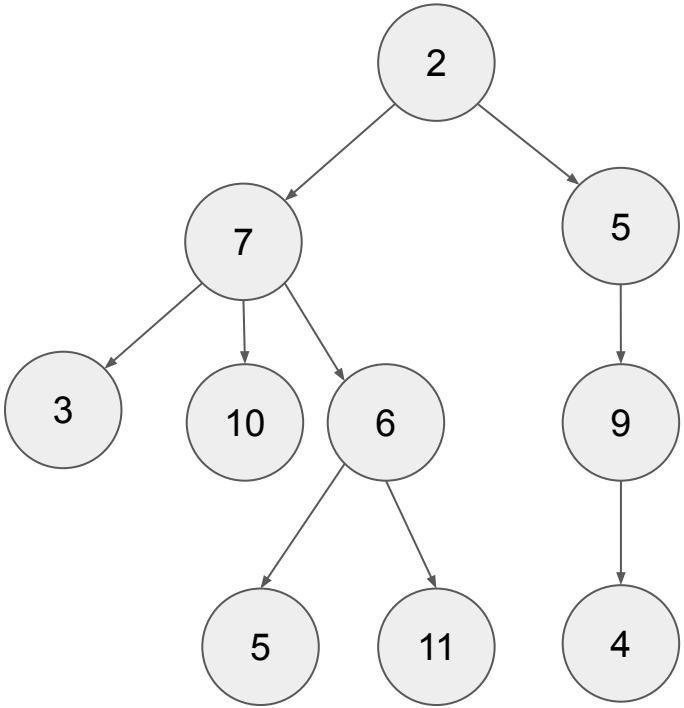
- Technical Interview Prep with Meta (Oct.) +0.5%
- Technical Interview Prep with Chris (Wed) +0.5%, up to 1%

Midpoint survey (link) +1%

Recall where we ended last lecture...

Tree

- A Tree is a **hierarchical** data structure that has a value and children. Each child is also a Tree, making this data structure **recursive** in nature.
- Don't confuse general N-ary Trees with Binary Trees (a special kind of tree where each node has at most two children) or Binary Search Trees (a special kind of binary tree where left subtree is less and right subtree is greater).



Representing Trees in C++

N-ary Tree (TreeNode.h)

- **`getValue()`:** returns value of node (generic type T)
- **`getChildren()`:** returns children pointers vector<TreeNode<T>*> (as we can have any # of children)

Constructor:

```
TreeNode(
  T value,
  std::vector<TreeNode<T>*> children = {})
```

Binary Tree (BinaryTree.h)

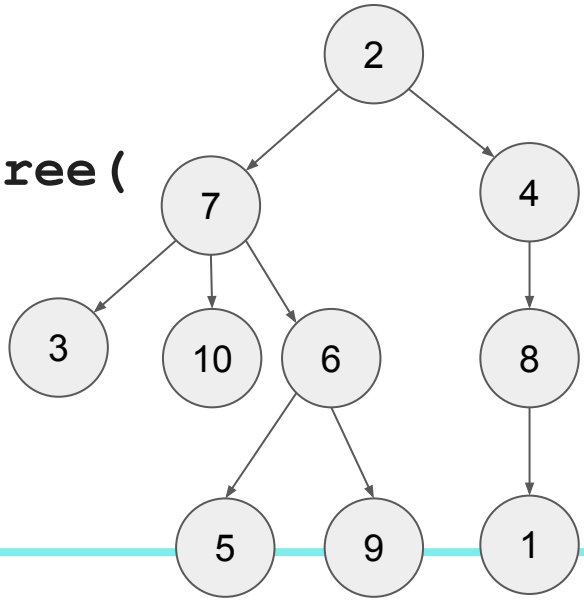
- **`getValue()`:** returns value of node (generic type T)
- **`getLeft()`:** returns left child (TreeNode<T>*)
- **`getRight()`:** returns right child (TreeNode<T>*)

Constructor:

```
BinaryTree(
  T value,
  BinaryTree<T>* left = nullptr,
  BinaryTree<T>* right = nullptr)
```

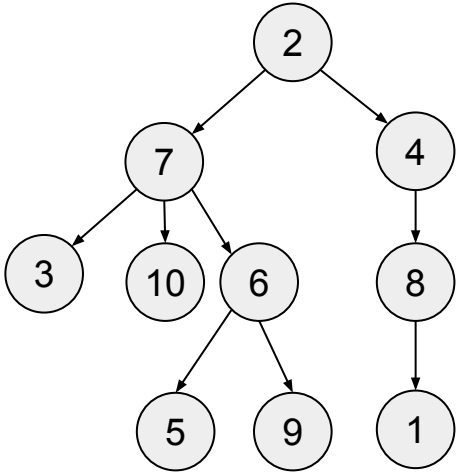
`findSumOfTree`

Write an algorithm that takes in a tree of ints, and returns the sum of all the values within the tree.

`findSumOfTree(`  `) outputs 55`

findSumOfTree Time/Space Complexity

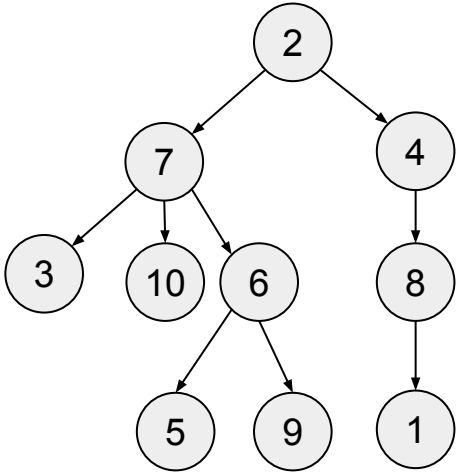
```
int findSumOfTree(TreeNode<int>* root) {
 if (root->isLeaf()) {
   return root->getValue();
 }
 int sumSoFar = 0;
 for (TreeNode<int>* n: root->getChildren()) {
   sumSoFar += findSumOfTree(n);
 }
 return sumSoFar + root->getValue();
}
```



- Base case
- Recursive call
- Building on recursive call

findSumOfTree Time/Space Complexity

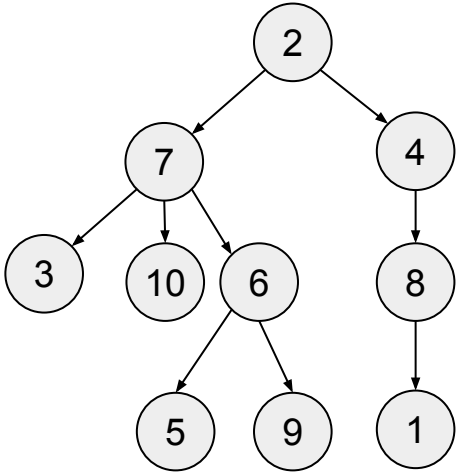
```
int findSumOfTree(TreeNode<int>* root) {
 if (root->isLeaf()) {
   return root->getValue();
 }
 int sumSoFar = 0;
 for (TreeNode<int>* n: root->getChildren()) {
   sumSoFar += findSumOfTree(n);
 }
 return sumSoFar + root->getValue();
}
```



- Best-case time complexity?
- Worst-case time complexity?
- Average-case time complexity?
- Space complexity?

findSumOfTree Time/Space Complexity

```
int findSumOfTree(TreeNode<int>* root) {
 if (root->isLeaf()) {
   return root->getValue();
 }
 int sumSoFar = 0;
 for (TreeNode<int>* n: root->getChildren()) {
   sumSoFar += findSumOfTree(n);
 }
 return sumSoFar + root->getValue();
}
```

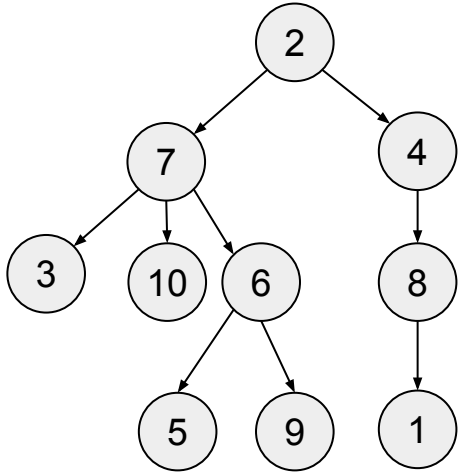


- Best-case time complexity?
- Worst-case time complexity?
- Average-case time complexity?
- Space complexity?

Hint: how many calls to findSumOfTree do we make, and how much work is being done in each call outside of the recursion?

findSumOfTree Time/Space Complexity

```
int findSumOfTree(TreeNode<int>* root) {
 if (root->isLeaf()) {
   return root->getValue();
 }
 int sumSoFar = 0;
 for (TreeNode<int>* n: root->getChildren()) {
   sumSoFar += findSumOfTree(n);
 }
 return sumSoFar + root->getValue();
}
```

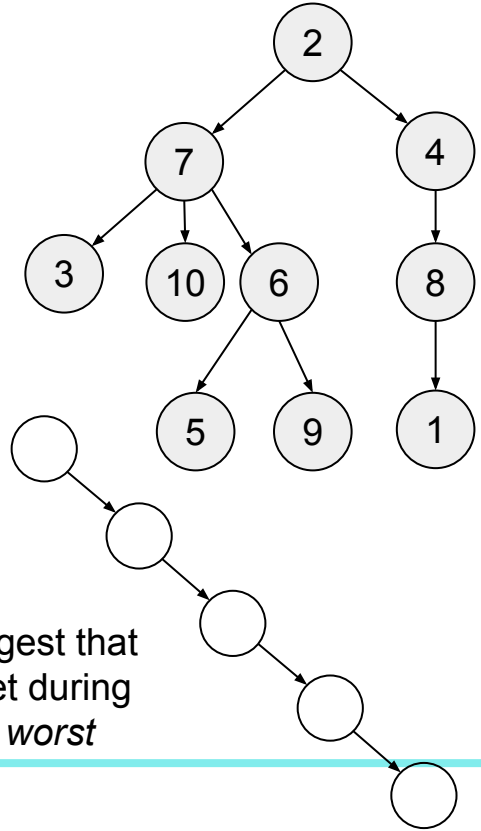


- Best-case time complexity? O(n)
- Worst-case time complexity? O(n)
- Average-case time complexity? O(n)
- Space complexity?

Hint: what is the largest that the call stack will get during the recursion *in the worst case?*

findSumOfTree Time/Space Complexity

```
int findSumOfTree(TreeNode<int>* root) {
 if (root->isLeaf()) {
   return root->getValue();
 }
 int sumSoFar = 0;
 for (TreeNode<int>* n: root->getChildren()) {
   sumSoFar += findSumOfTree(n);
 }
 return sumSoFar + root->getValue();
}
```

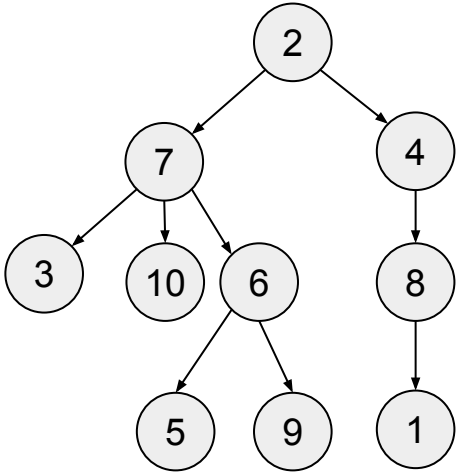


- Best-case time complexity? O(n)
- Worst-case time complexity? O(n)
- Average-case time complexity? O(n)
- Space complexity?

Hint: what is the largest that the call stack will get during the recursion *in the worst case?*

findSumOfTree Time/Space Complexity

```
int findSumOfTree(TreeNode<int>* root) {
 if (root->isLeaf()) {
   return root->getValue();
 }
 int sumSoFar = 0;
 for (TreeNode<int>* n: root->getChildren()) {
   sumSoFar += findSumOfTree(n);
 }
 return sumSoFar + root->getValue();
}
```



- Best-case time complexity? O(n)
- Worst-case time complexity? O(n)
- Average-case time complexity? O(n)
- Space complexity? O(n)

Big Questions!

- What are binary trees again?
- How do we traverse binary trees? (recursion!)
- How can I practice?

Big Questions!

- What are binary trees again?
- How do we traverse binary trees? (recursion!)
- How can I practice?

Binary Tree

- A Binary Tree is a specific kind of Tree **with at most two children.**
- Each Binary Tree node has a **`getLeft()`** (which could be nullptr), a **`getRight()`** (which could be nullptr) and a **`getValue()`** (int, string, bool, etc)
- Binary Trees are typically used to build data structures that allow for fast searches and updates (addition / removal) of elements.
- In C++, we'll use the **`BinaryTree<T>`** class.

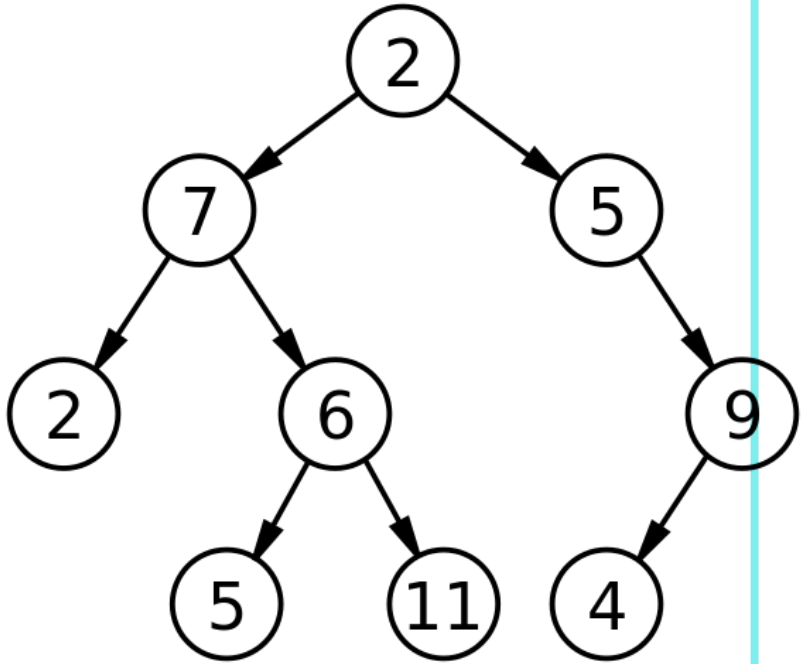
Examples

Let's say the root of this tree is called **`ex`** and our member functions are getLeft(), getRight(), getValue()

How would we get 4 from **`ex`**?

```
ex->getRight()->getRight()->getLeft()->getValue()
```

Poll: How many **`nullptr`**'s are there?



Examples

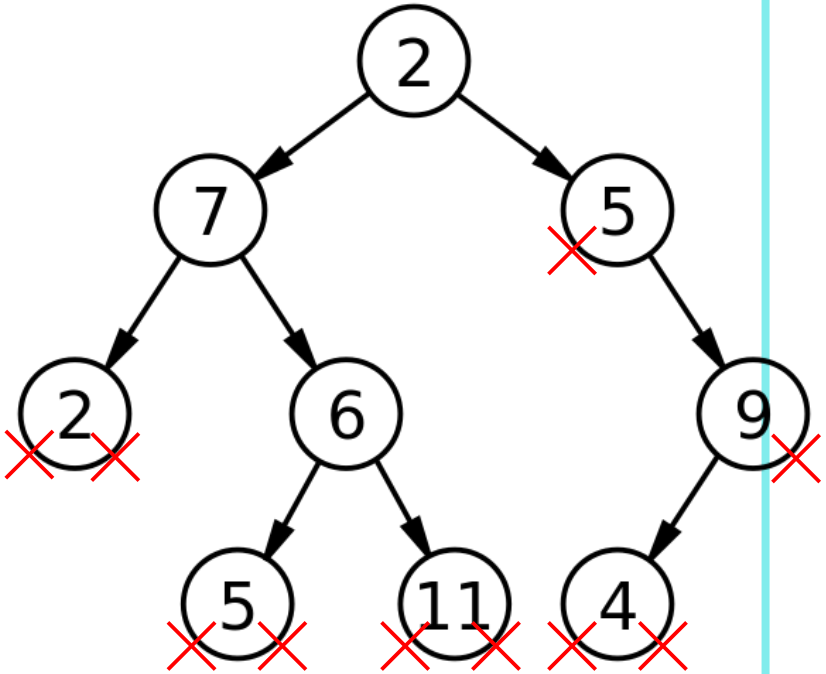
Let's say the root of this tree is called **`ex`** and our member functions are getLeft(), getRight(), getValue()

How would we get 4 from **`ex`**?

```
ex->getRight()->getRight()->getLeft()->getValue()
```

Poll: How many **`nullptr`**'s are there?

10



Big Questions!

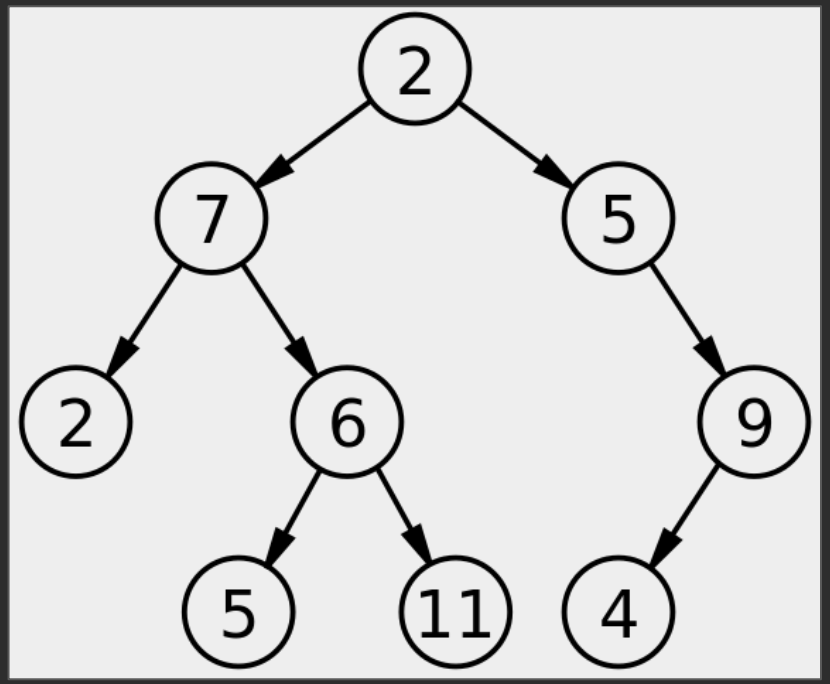
- What are binary trees again?
- How do we traverse binary trees? (recursion!)
- How can I practice?

Binary Tree Traversals

- As with generic Trees, we have algorithms for traversing every node.
- Specific to Binary Trees, we have three **subtypes of depth-first traversals**:
  - Pre-order: visit current node **before** recursing on my children (current, left, right)
  - In-order: recurse on left child, visit myself, recurse on right child (left, current, right)
  - Post-order: visit current node **after** recursing on my children (left, right, current)

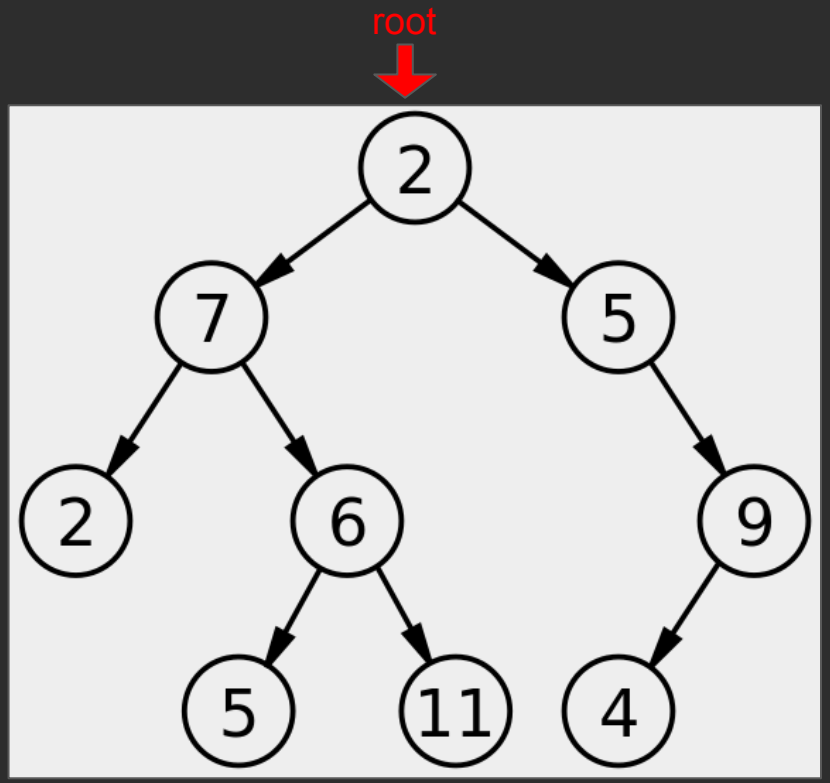
preOrder

```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



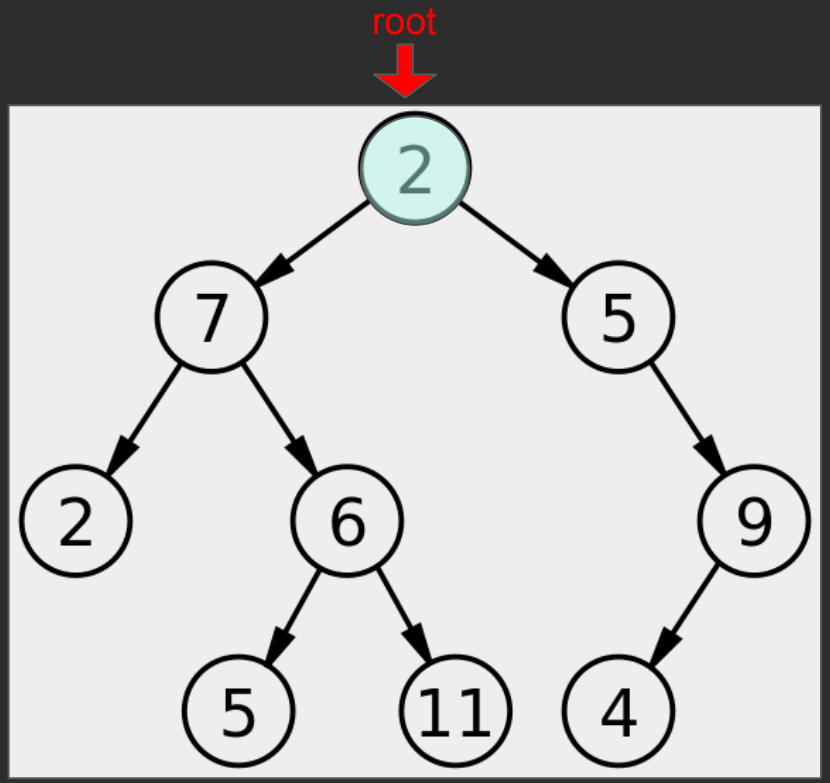
preOrder

```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



preOrder

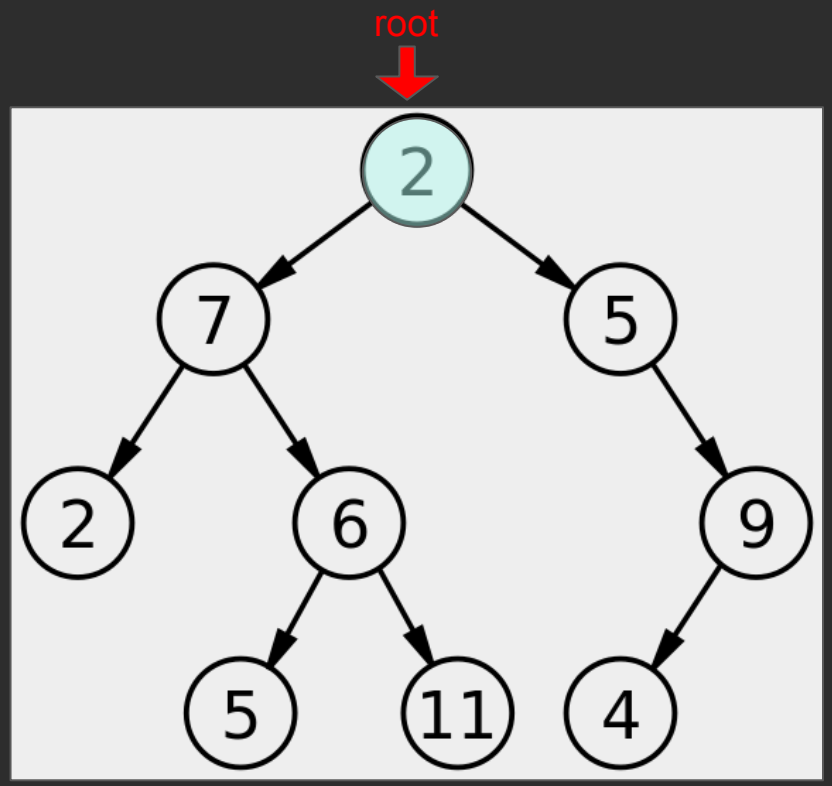
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2

preOrder

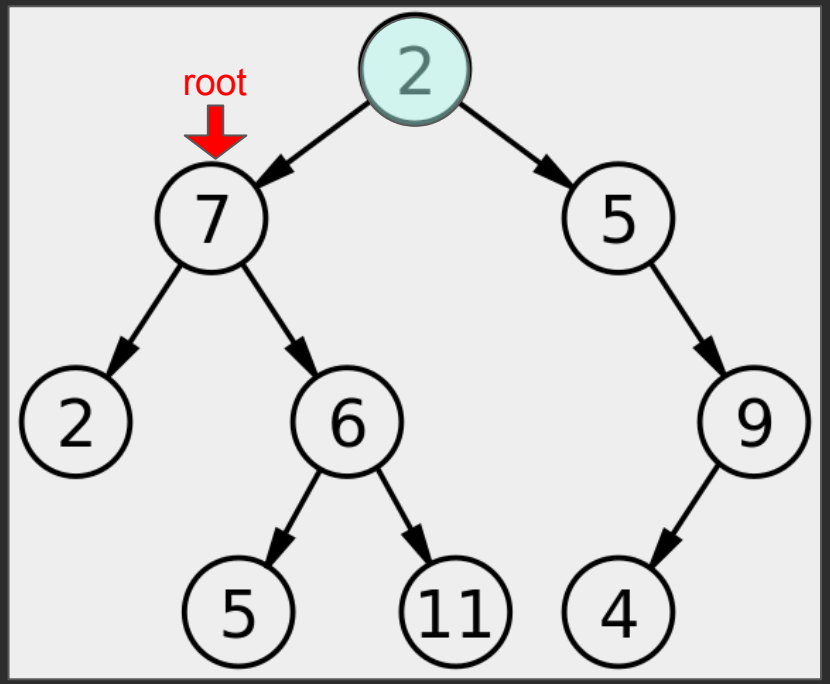
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2

preOrder

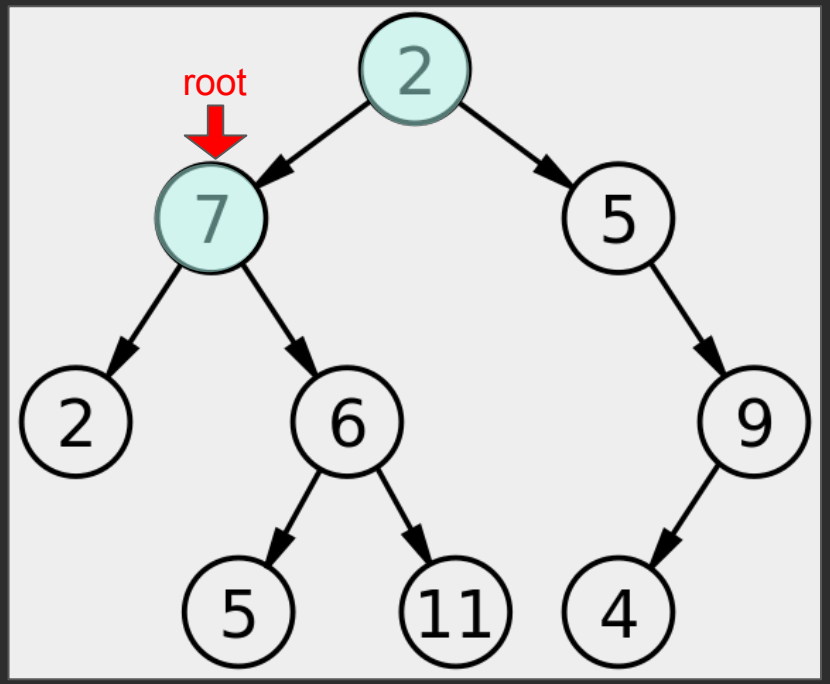
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2

preOrder

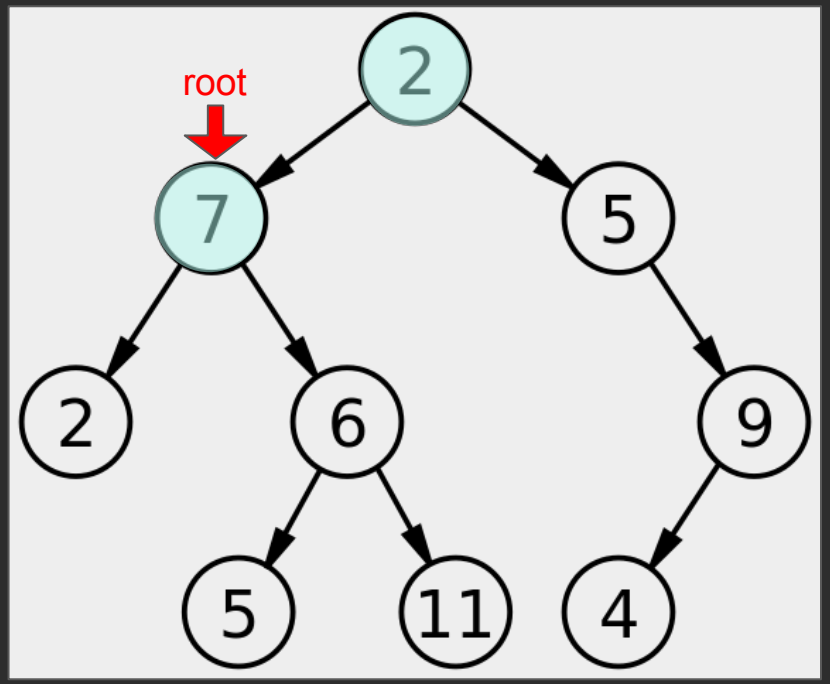
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7

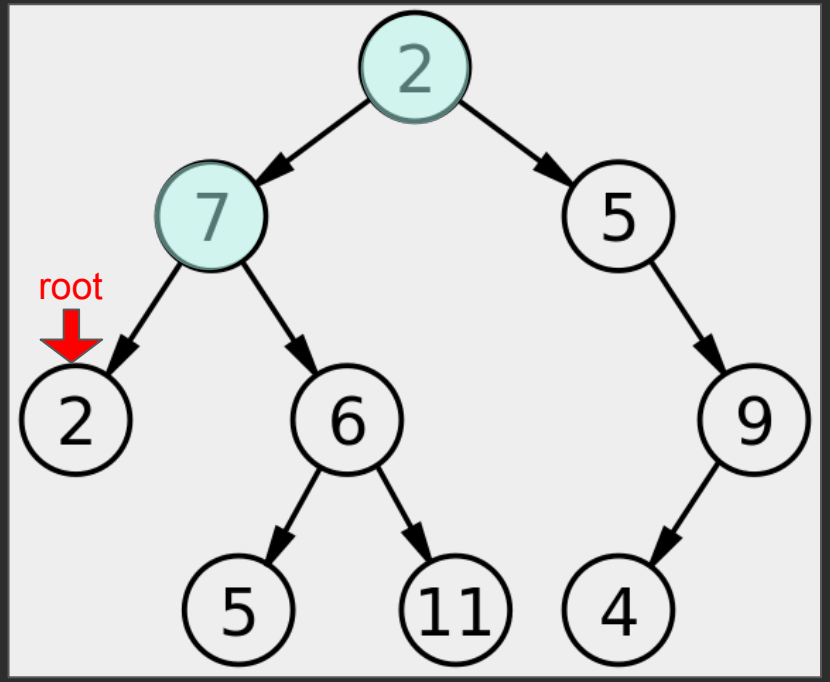
preOrder

```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



preOrder

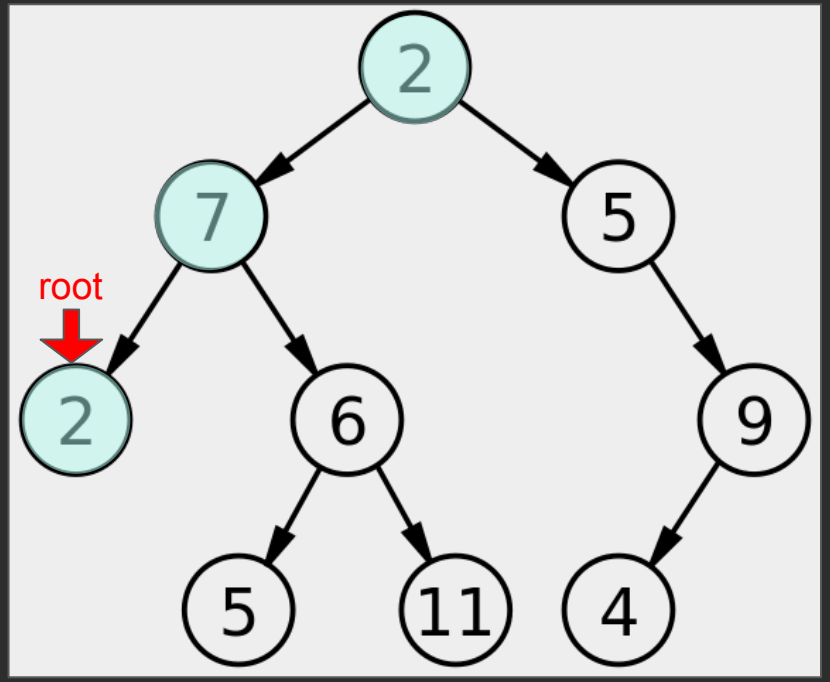
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7

preOrder

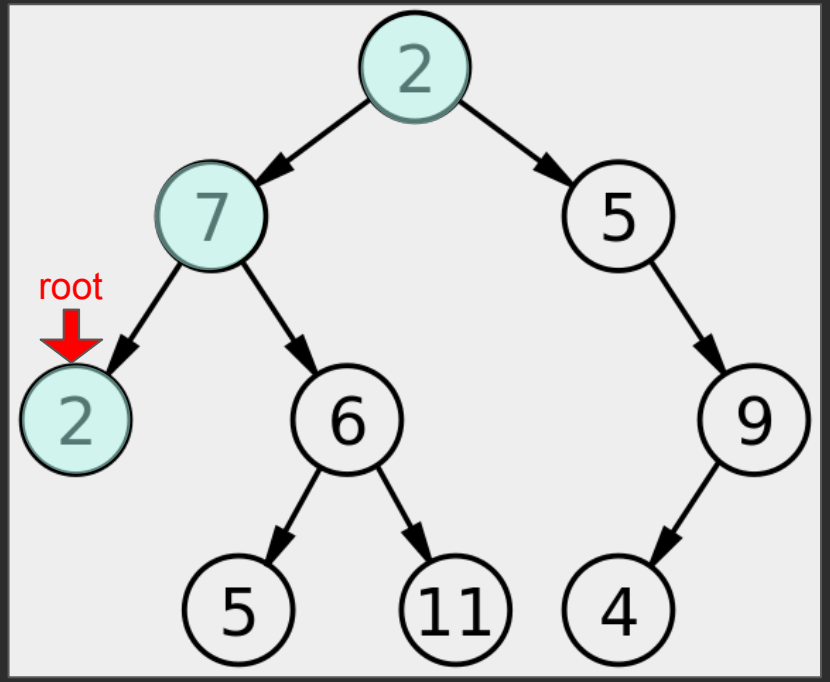
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

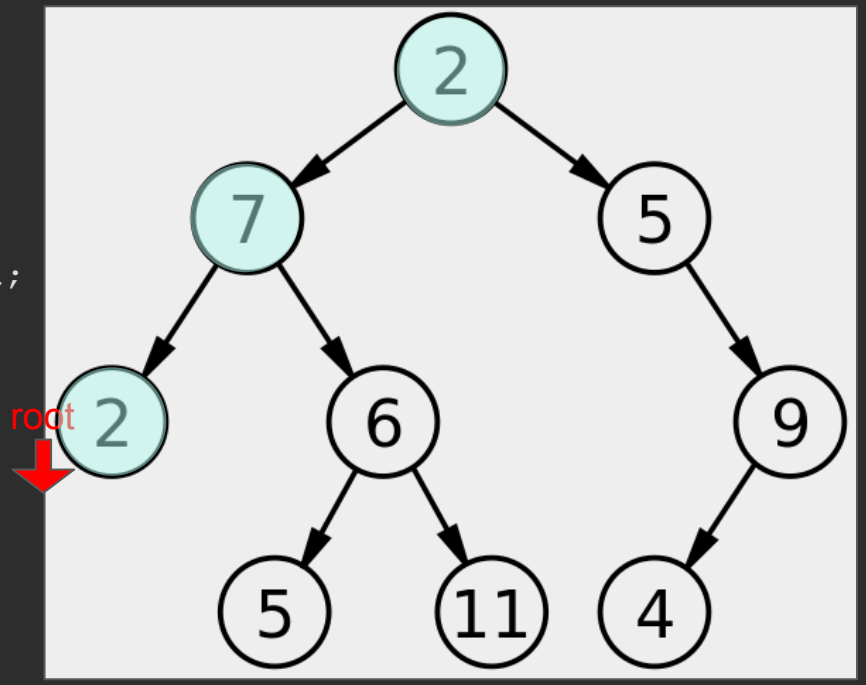
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

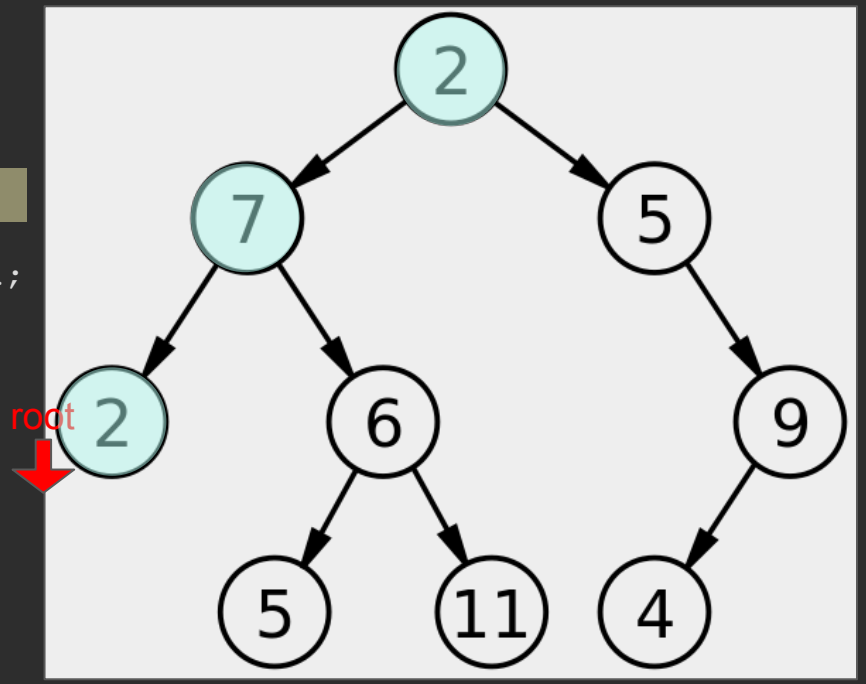
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

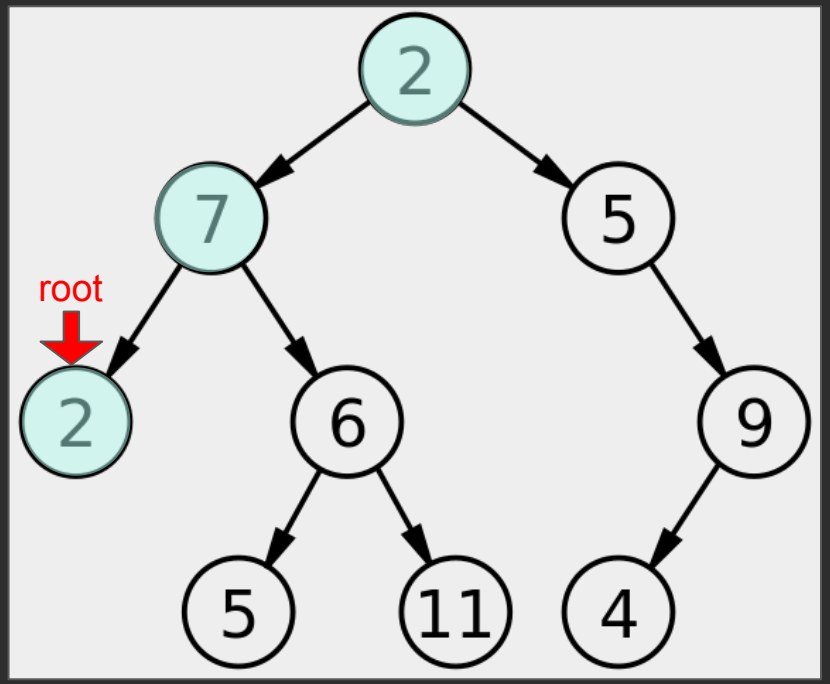
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

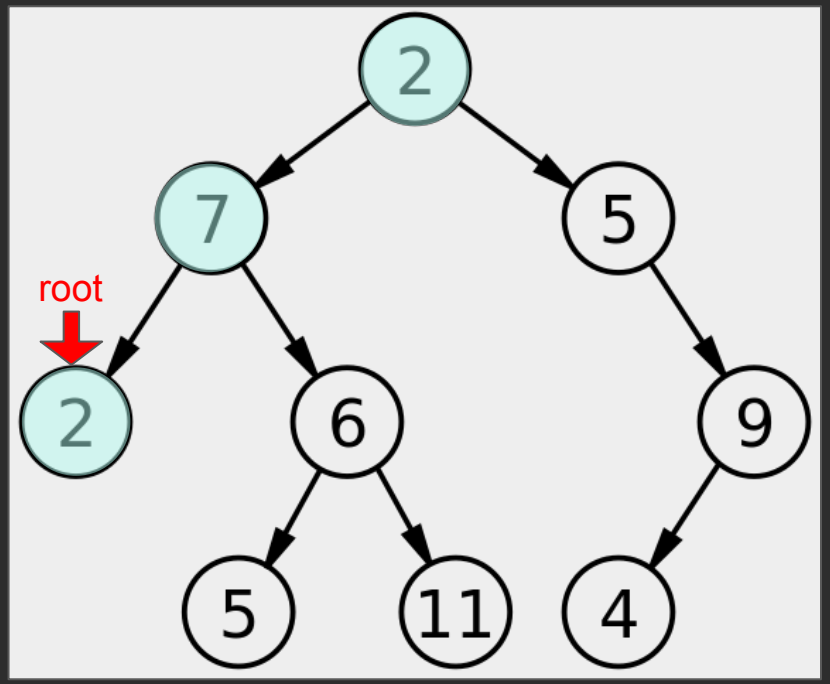
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

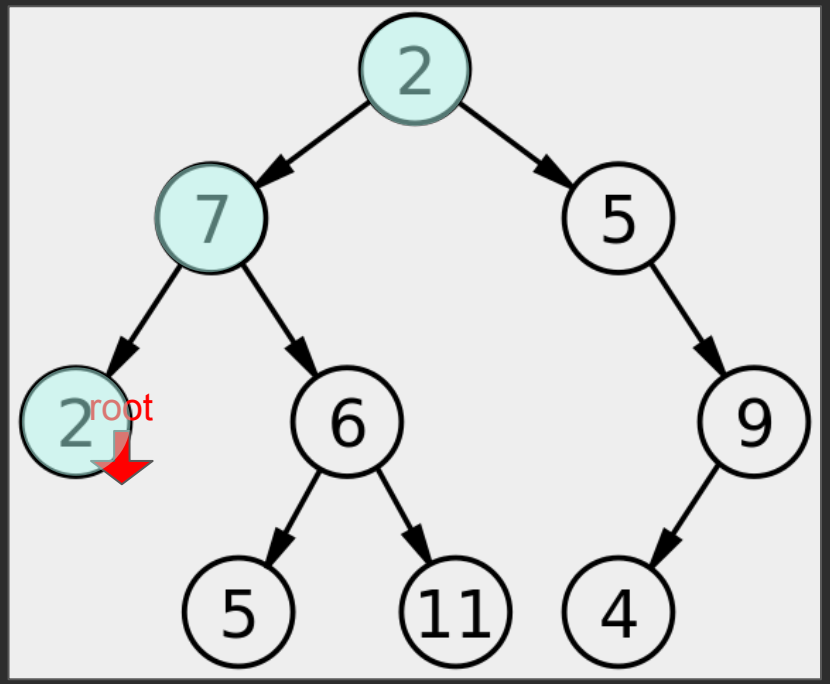
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

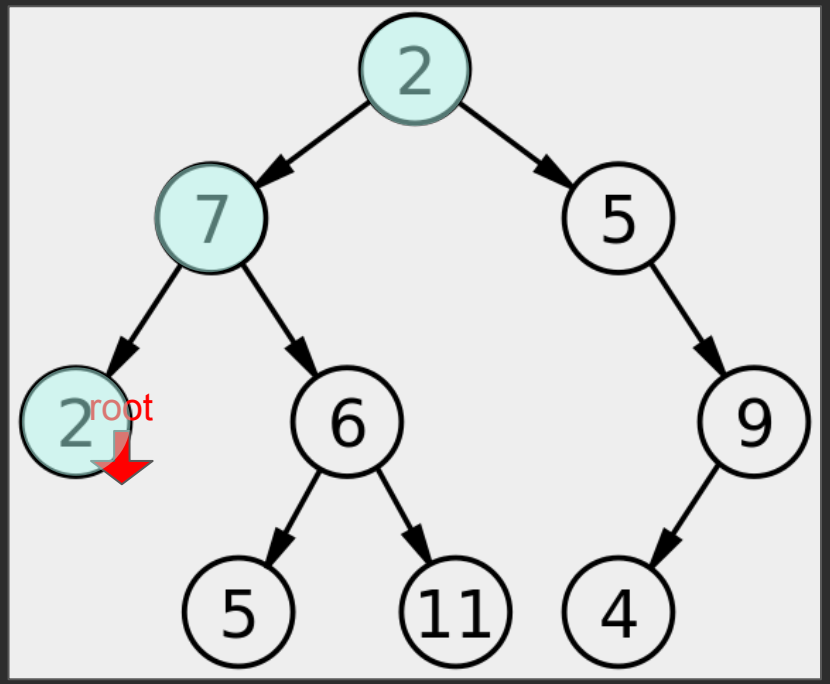
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

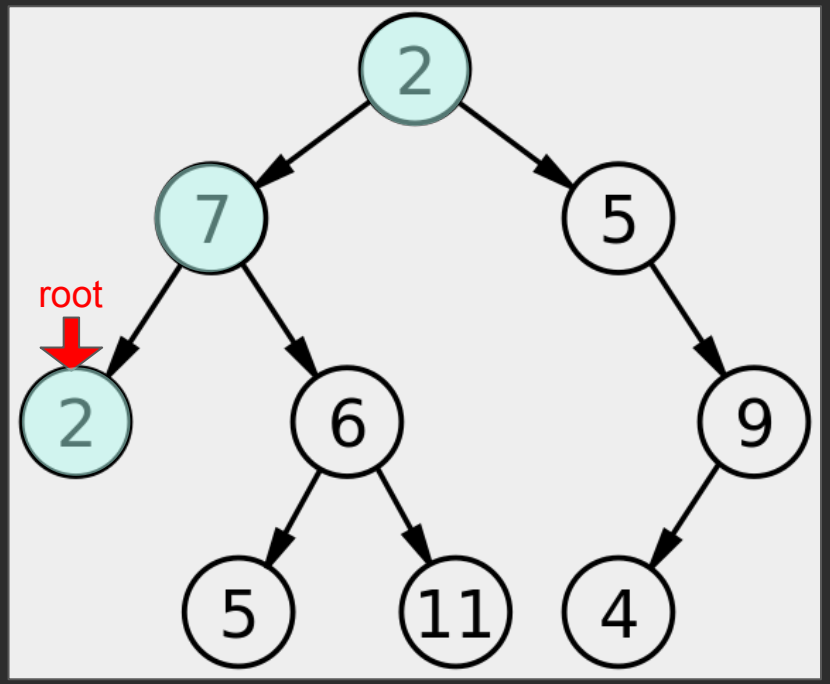
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

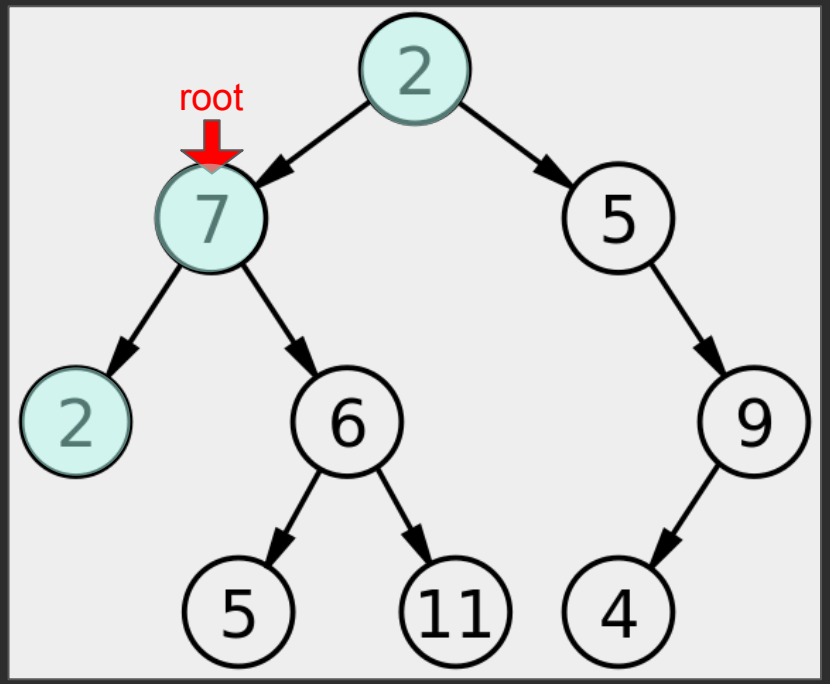
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

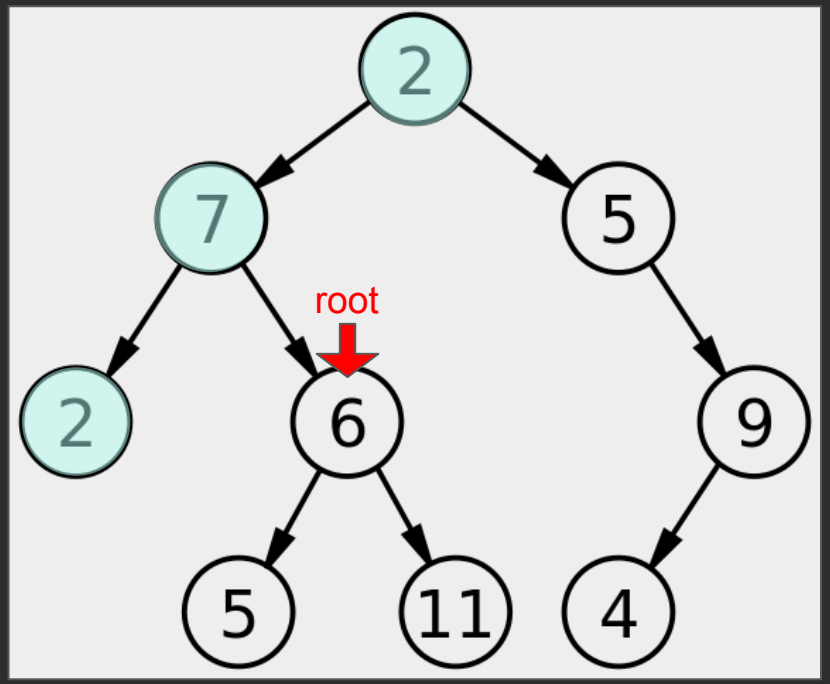
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

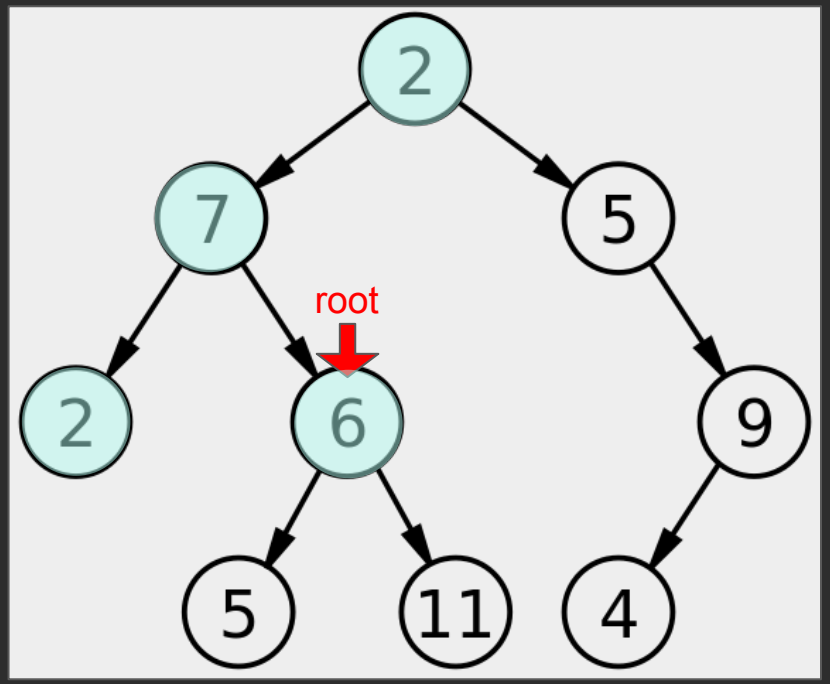
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2

preOrder

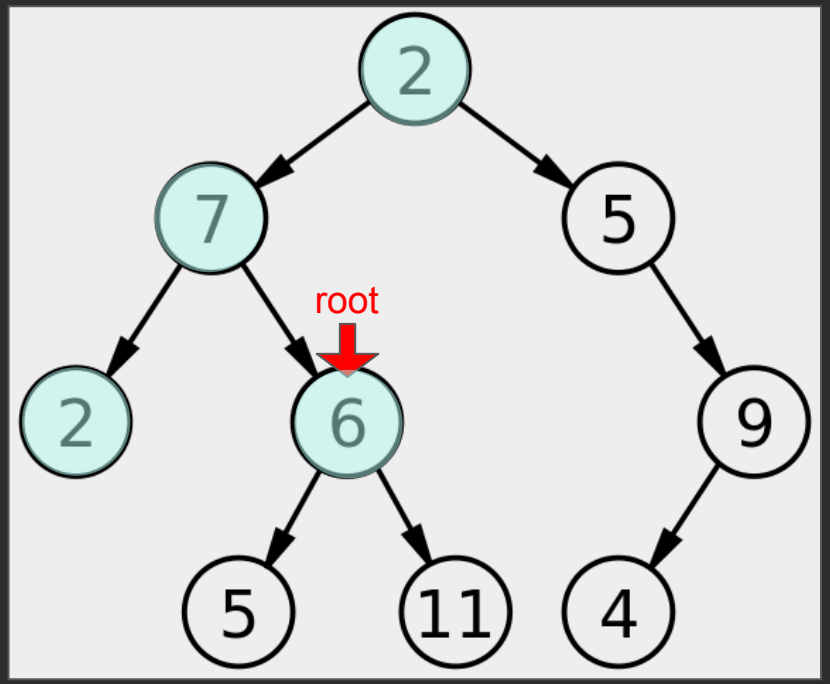
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6

preOrder

```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```

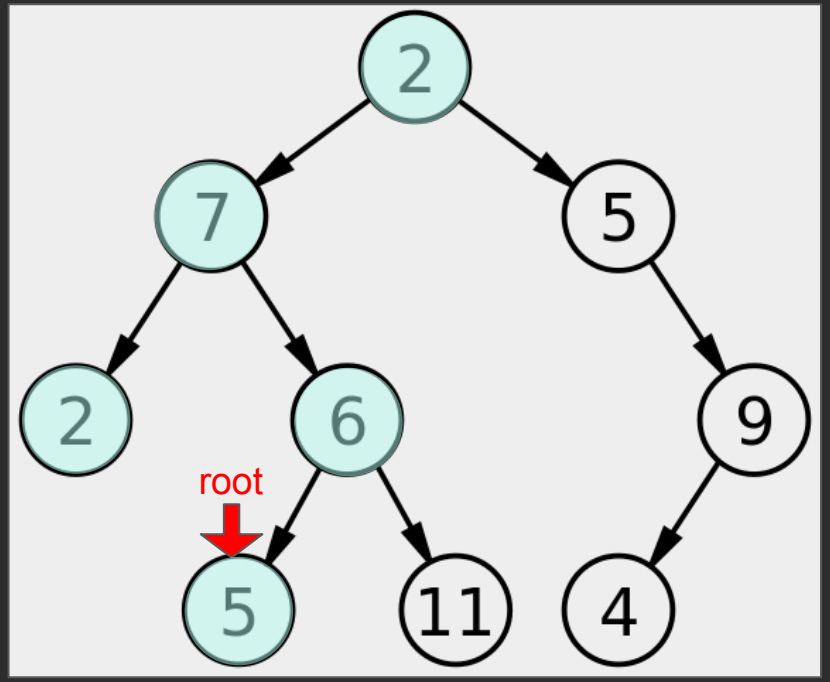


2
7
2
6

And it continues...

preOrder

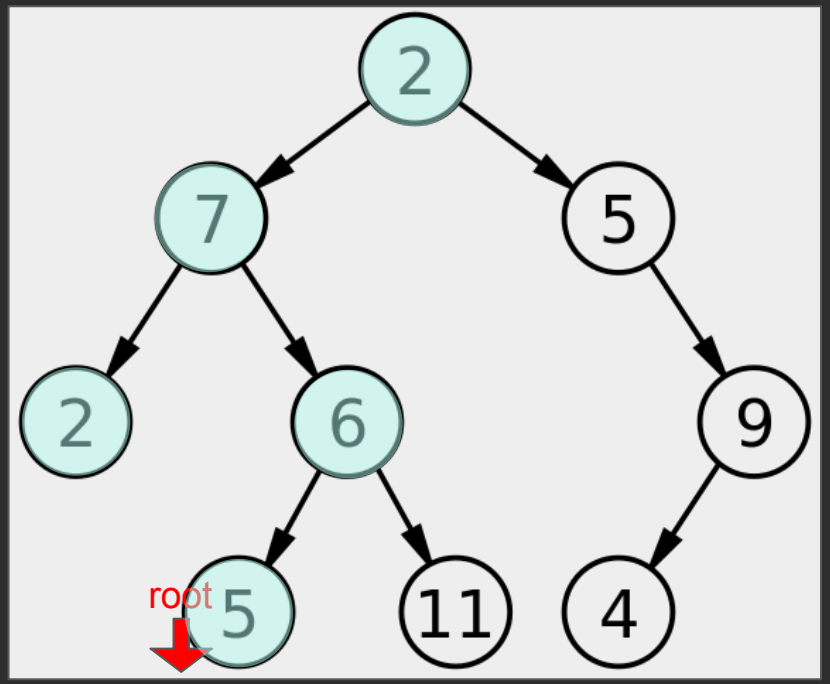
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5

preOrder

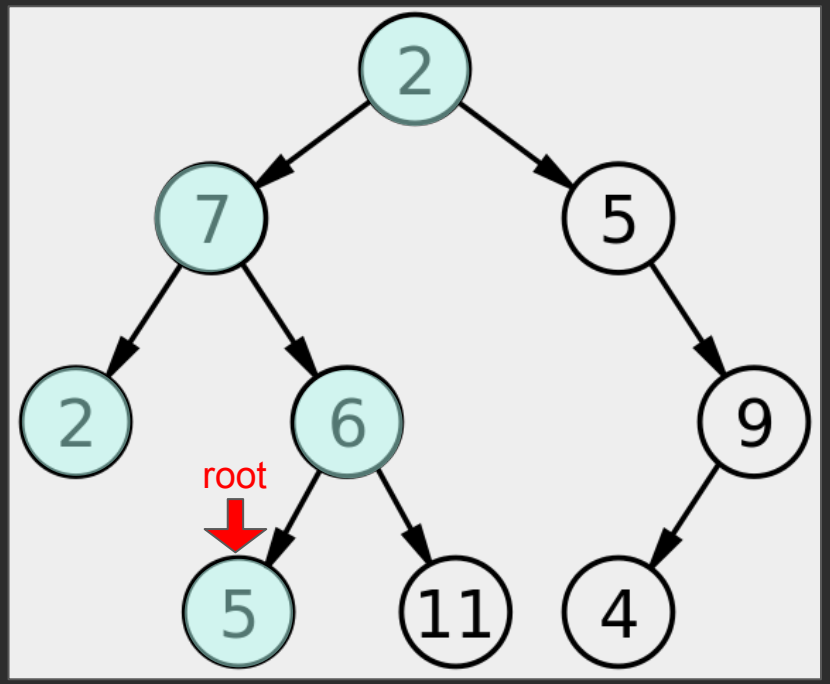
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5

preOrder

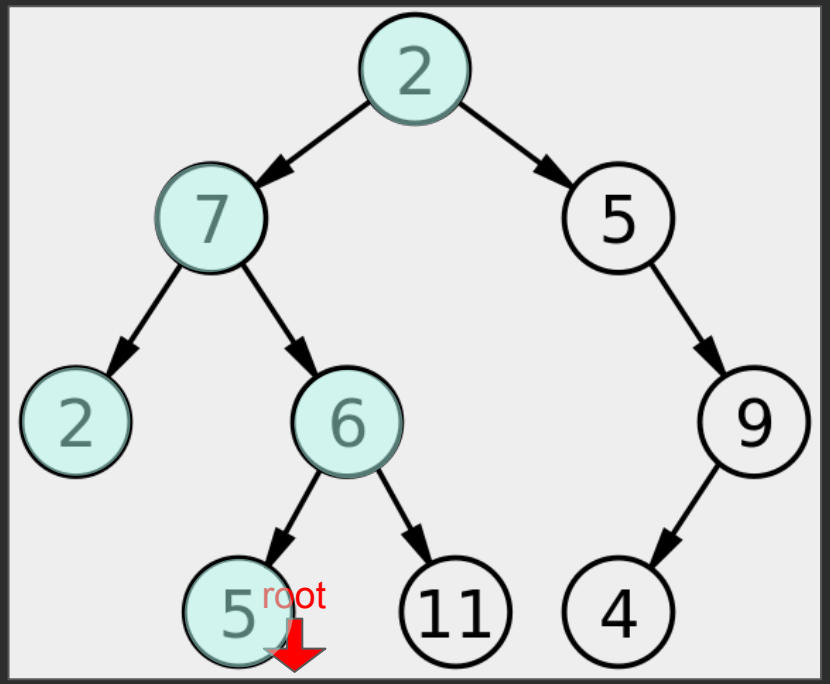
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5

preOrder

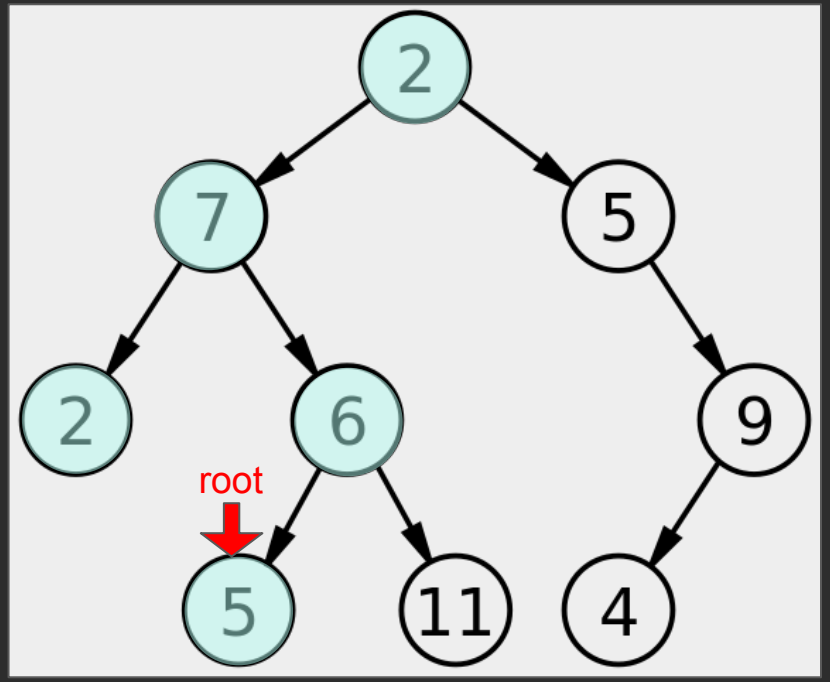
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5

preOrder

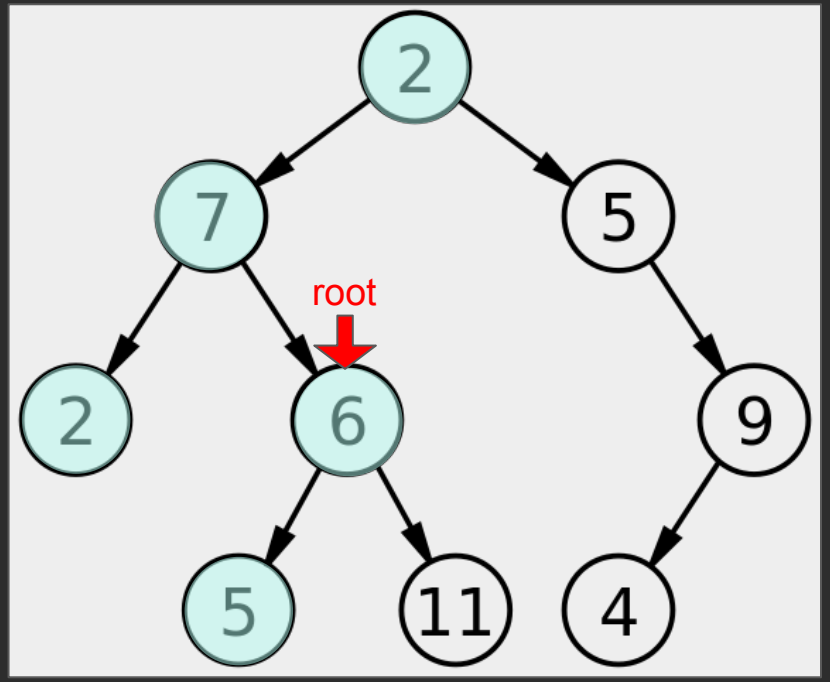
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5

preOrder

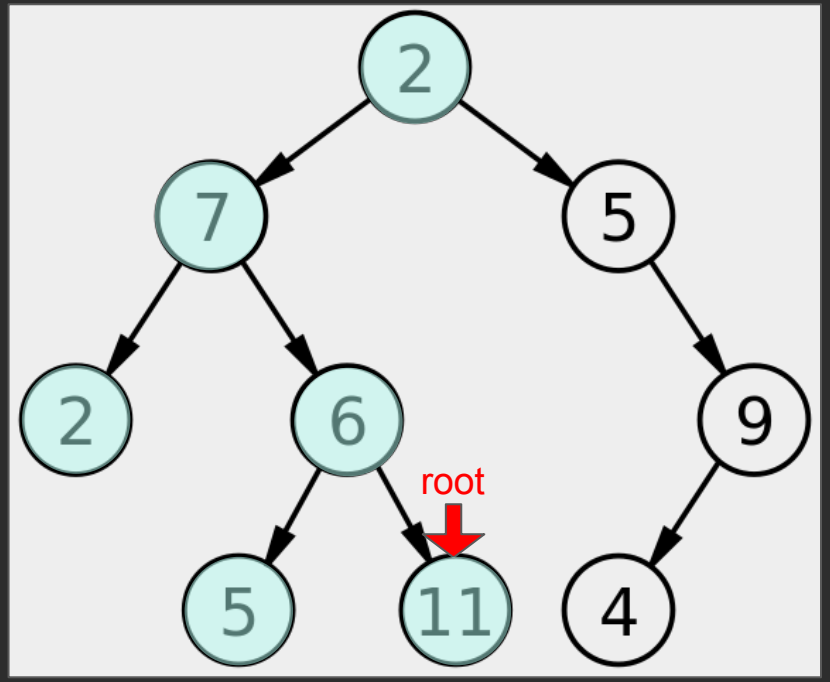
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5

preOrder

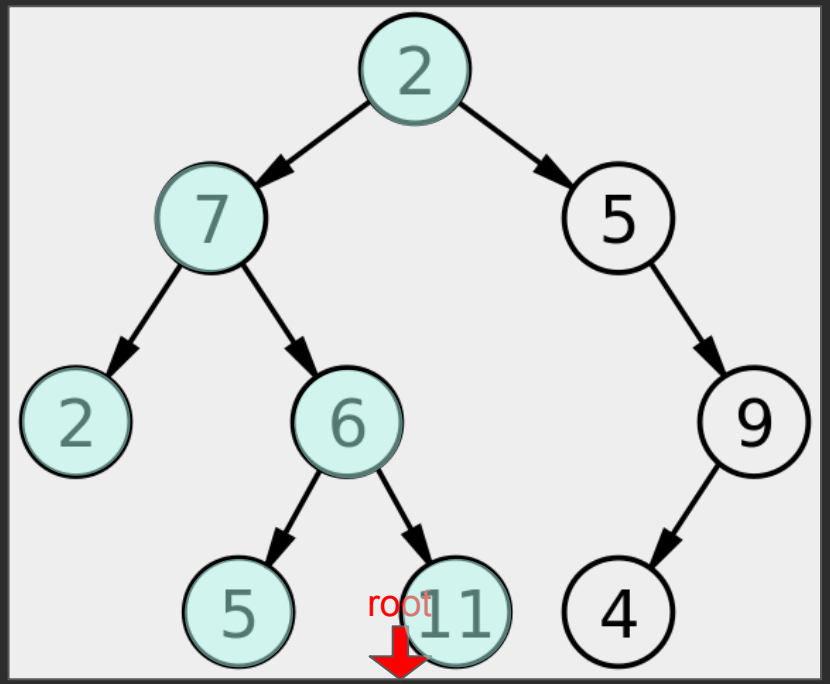
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11

preOrder

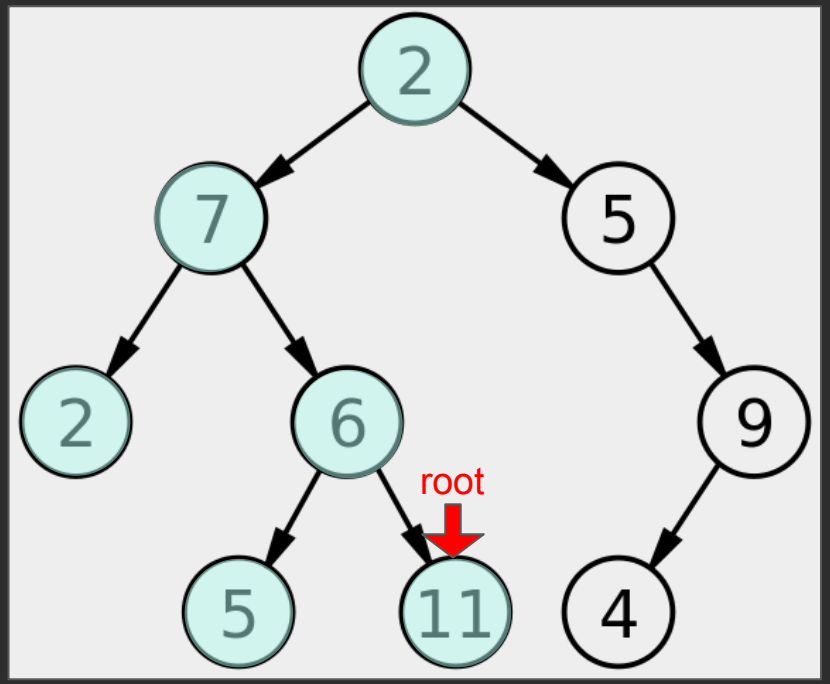
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11

preOrder

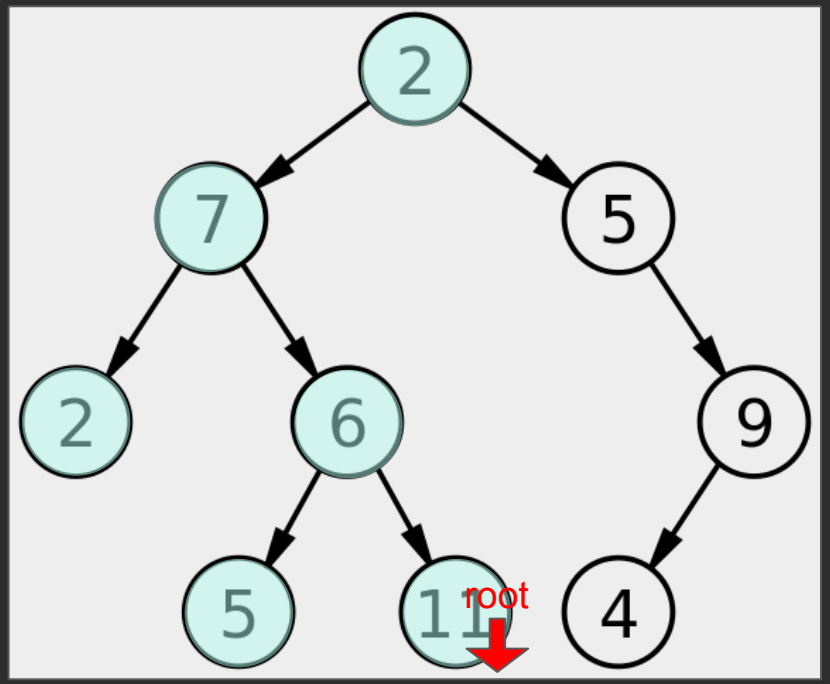
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11

preOrder

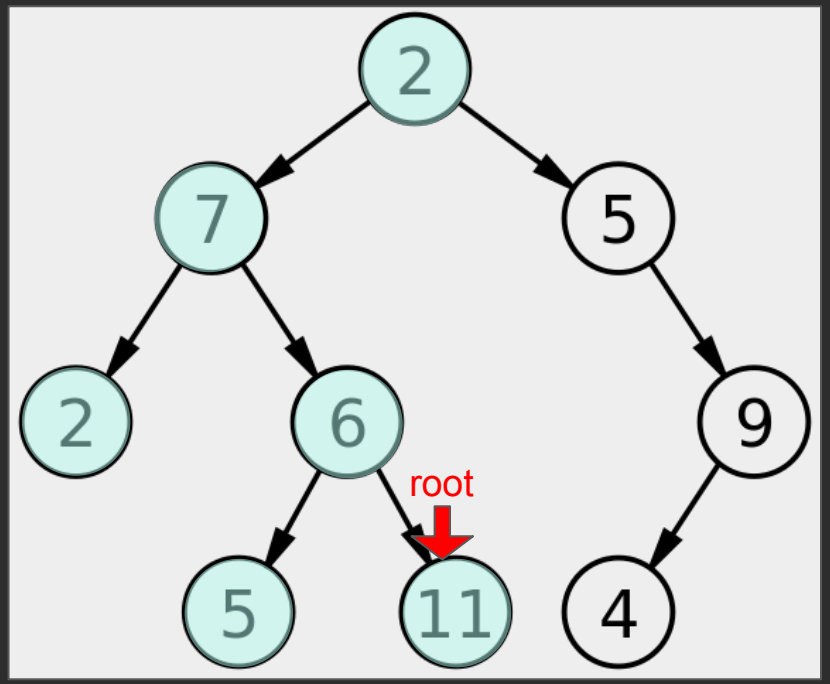
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11

preOrder

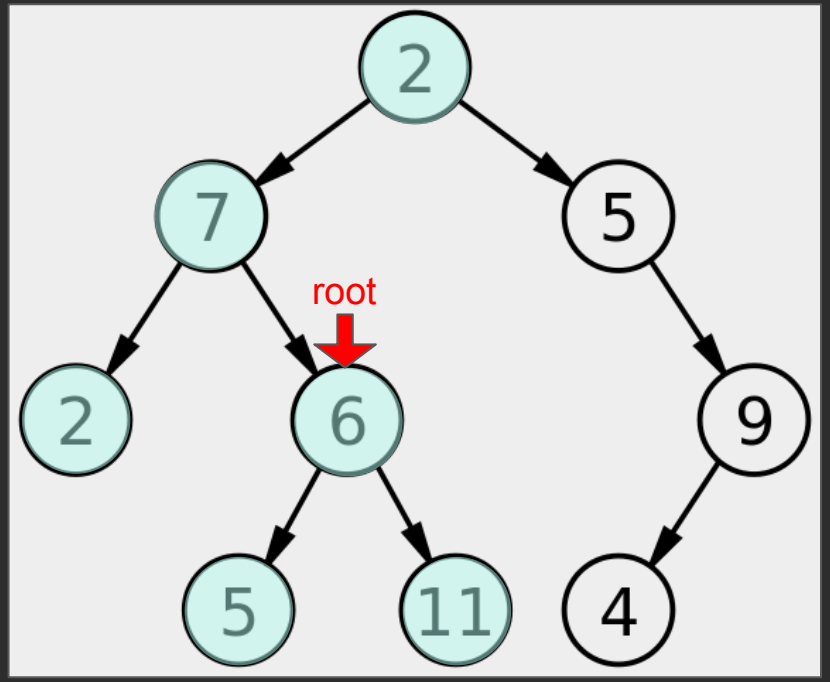
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11

preOrder

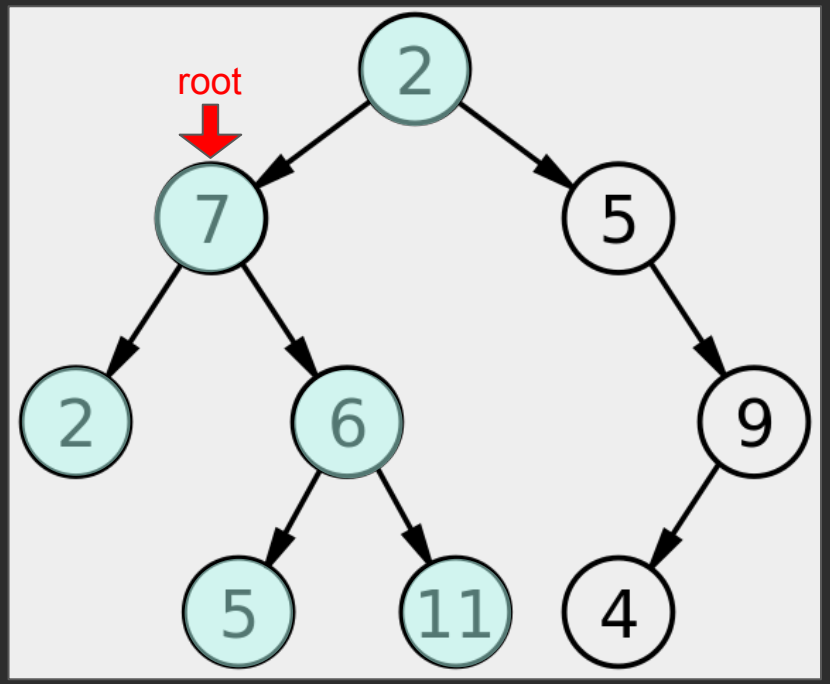
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11

preOrder

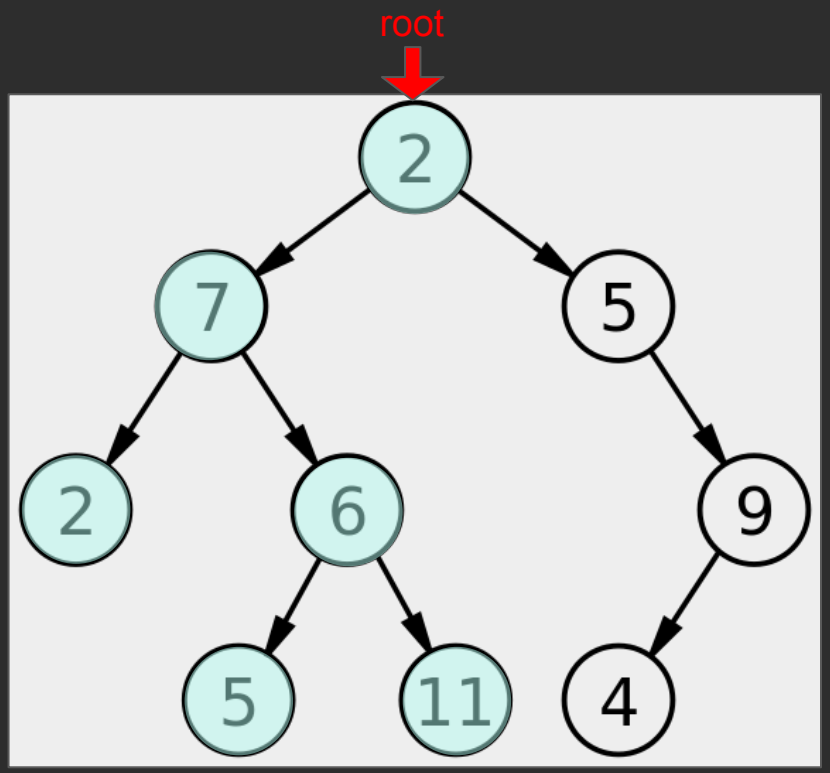
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11

preOrder

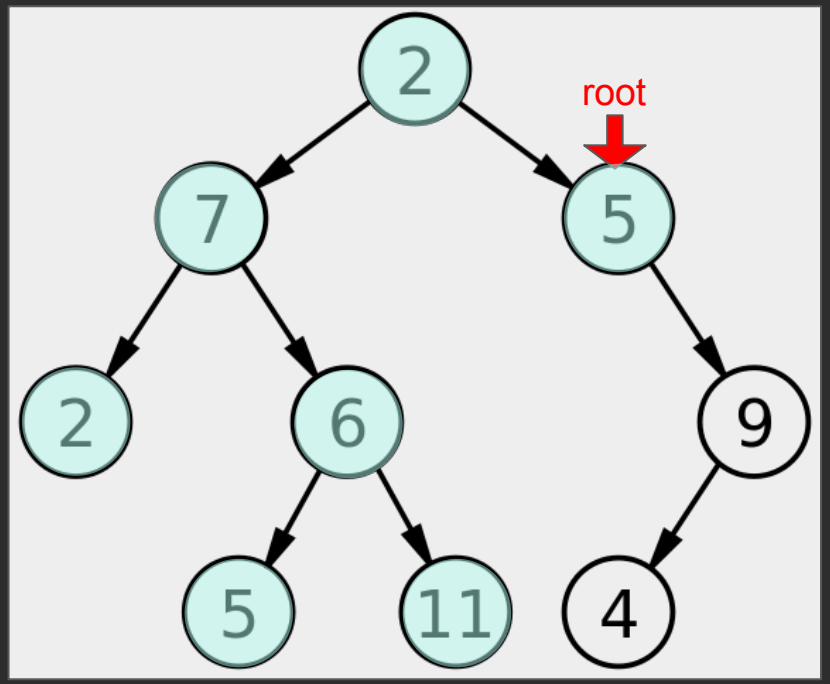
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11

preOrder

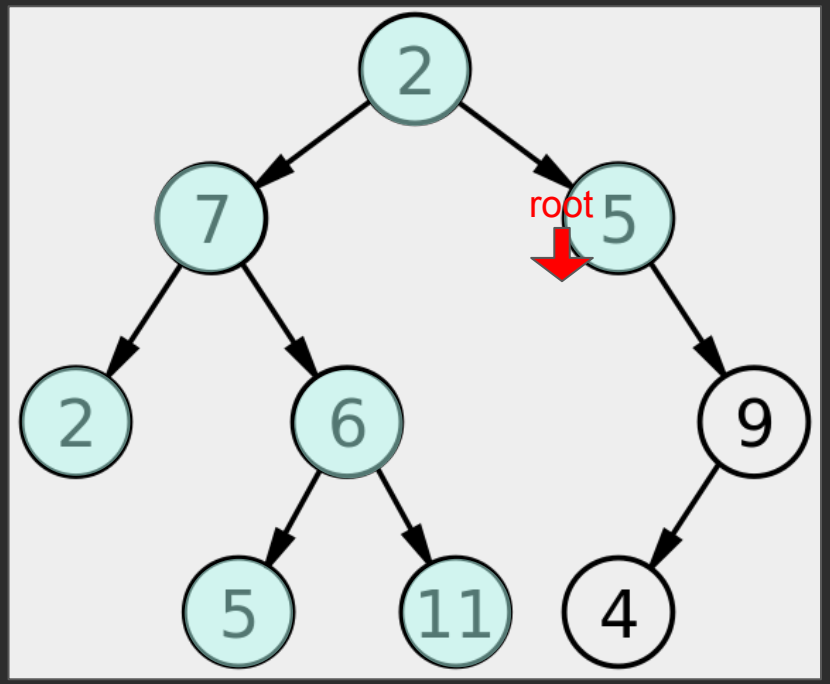
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5

preOrder

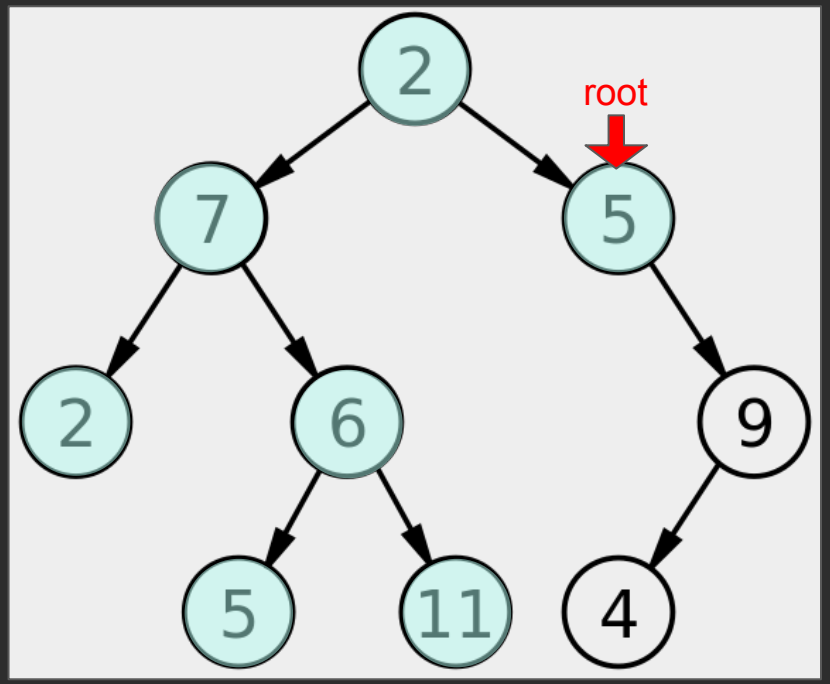
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5

preOrder

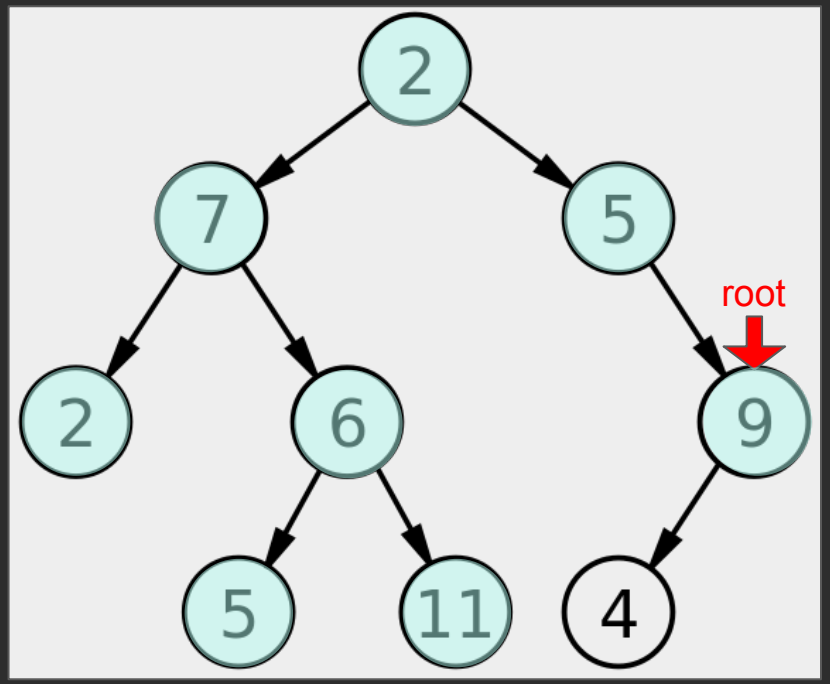
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5

preOrder

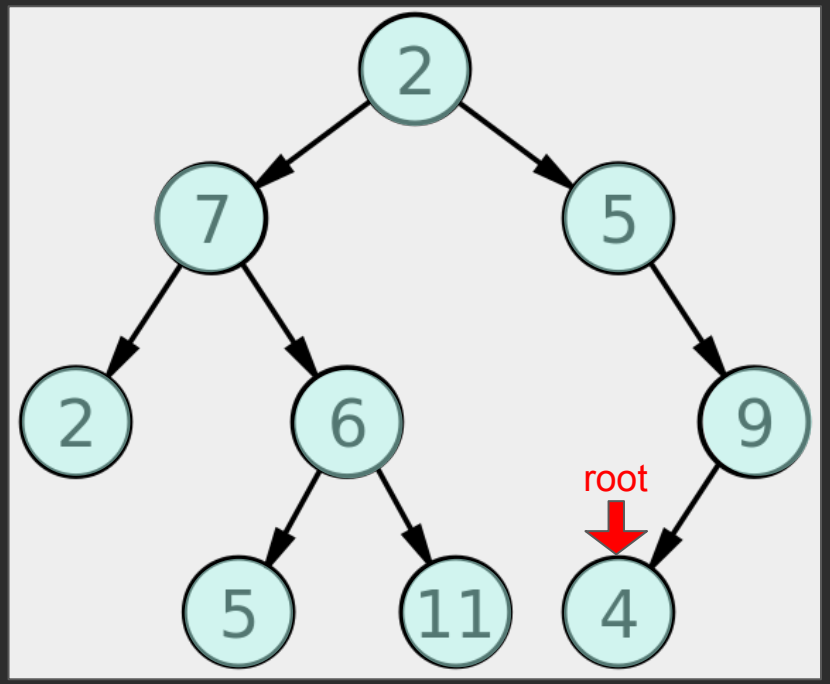
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5
9

preOrder

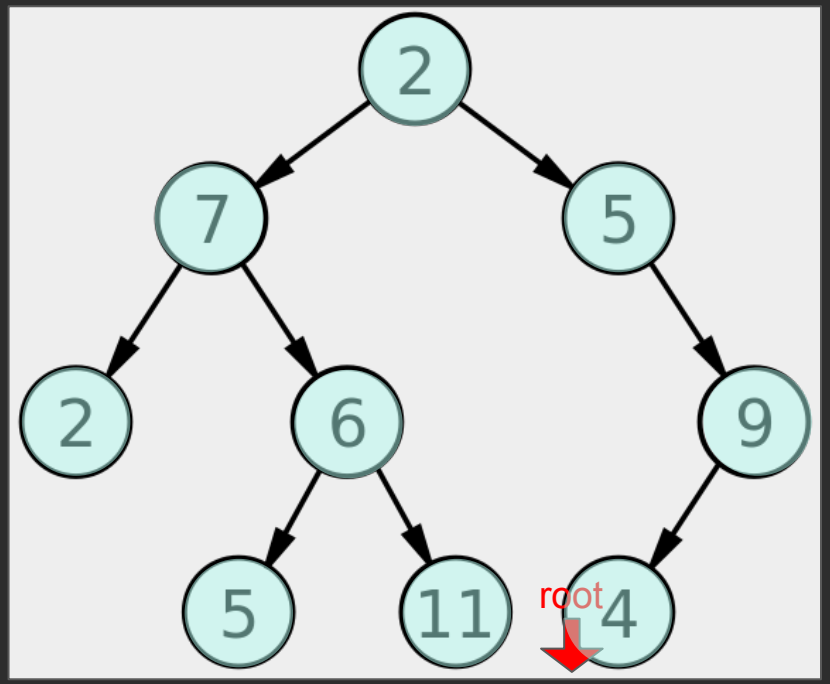
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5
9

preOrder

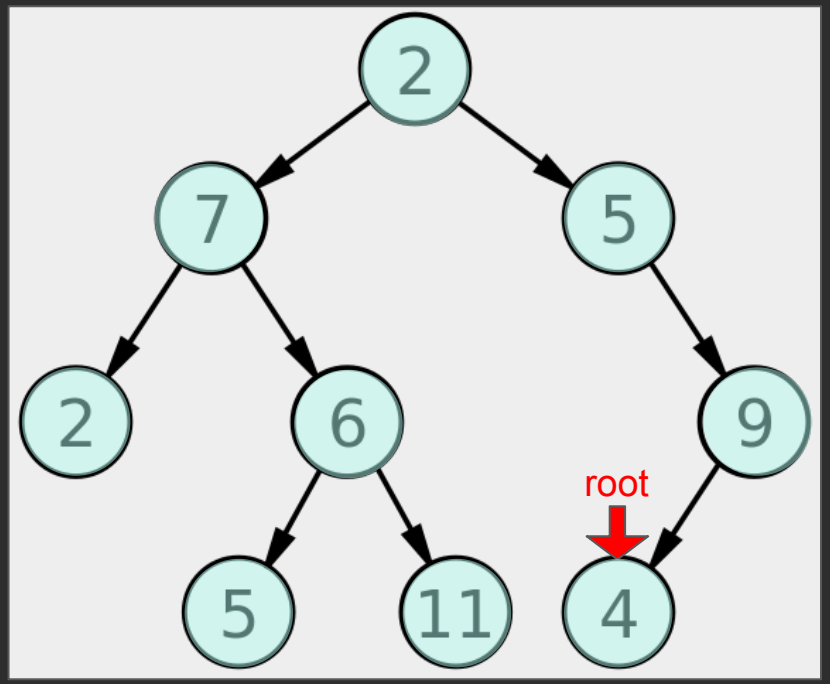
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5
9
4

preOrder

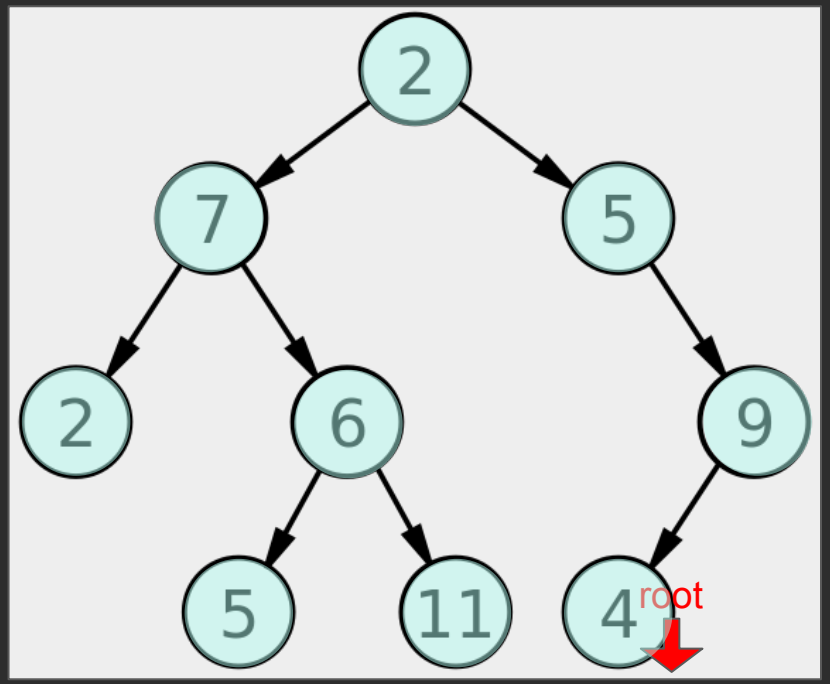
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5
9
4

preOrder

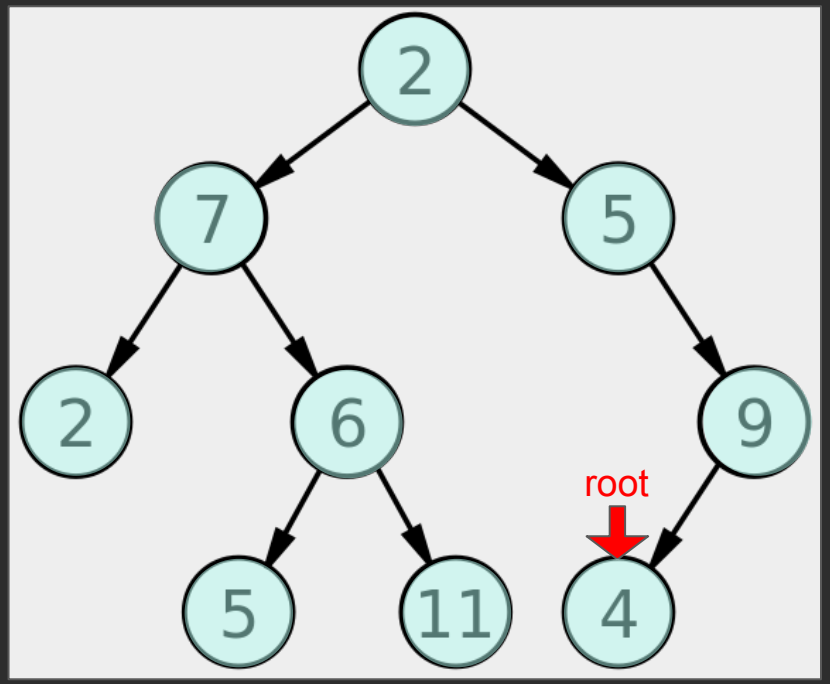
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5
9
4

preOrder

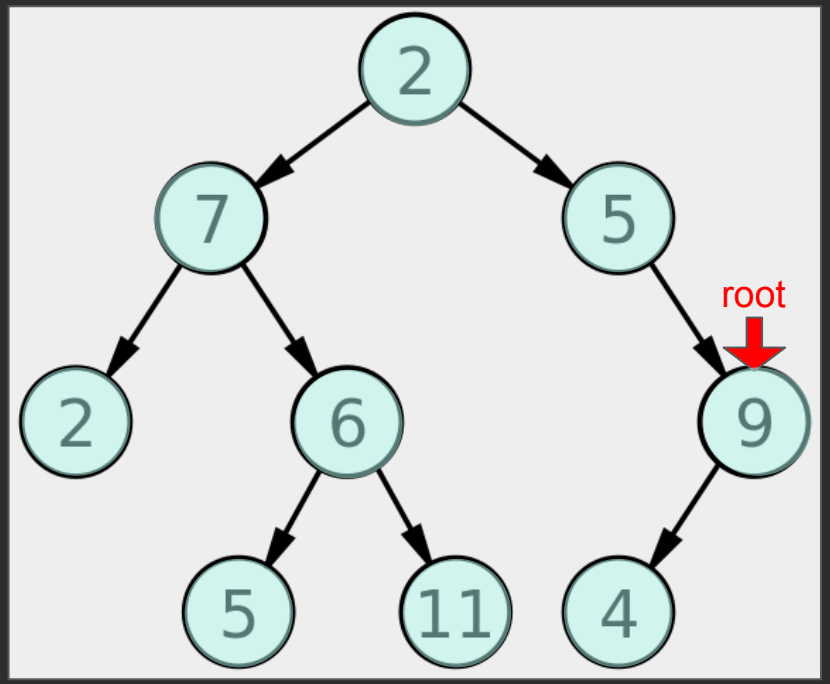
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5
9
4

preOrder

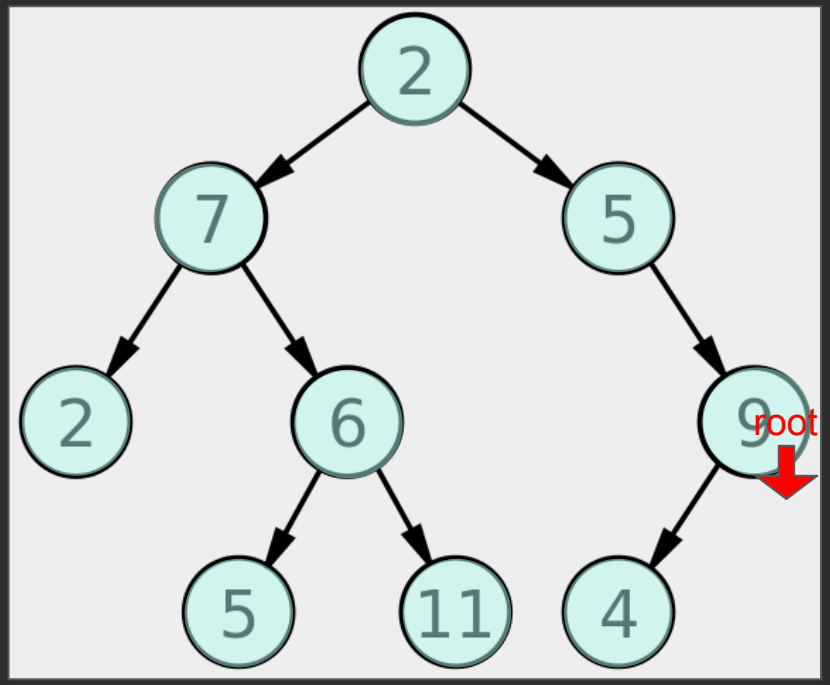
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5
9
4

preOrder

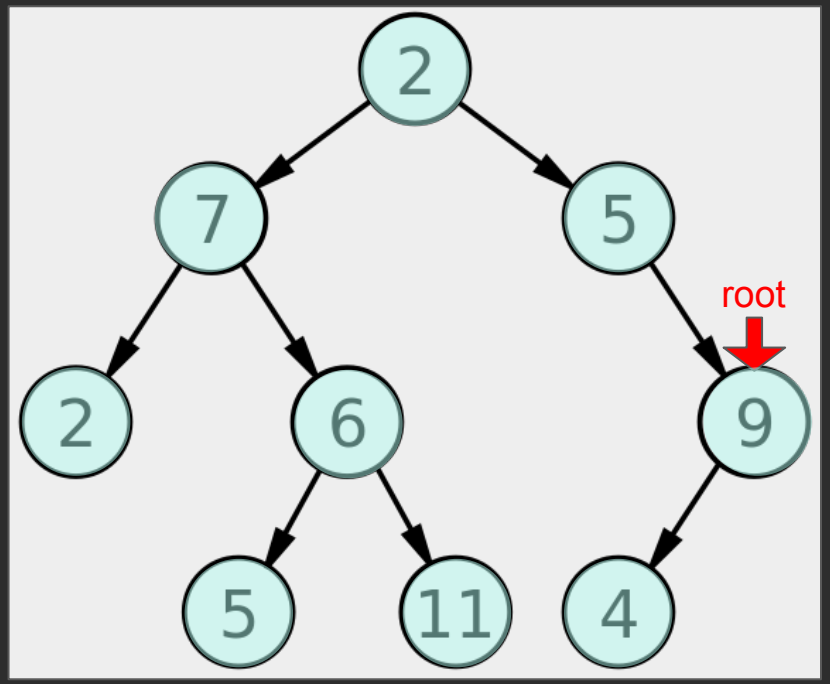
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5
9
4

preOrder

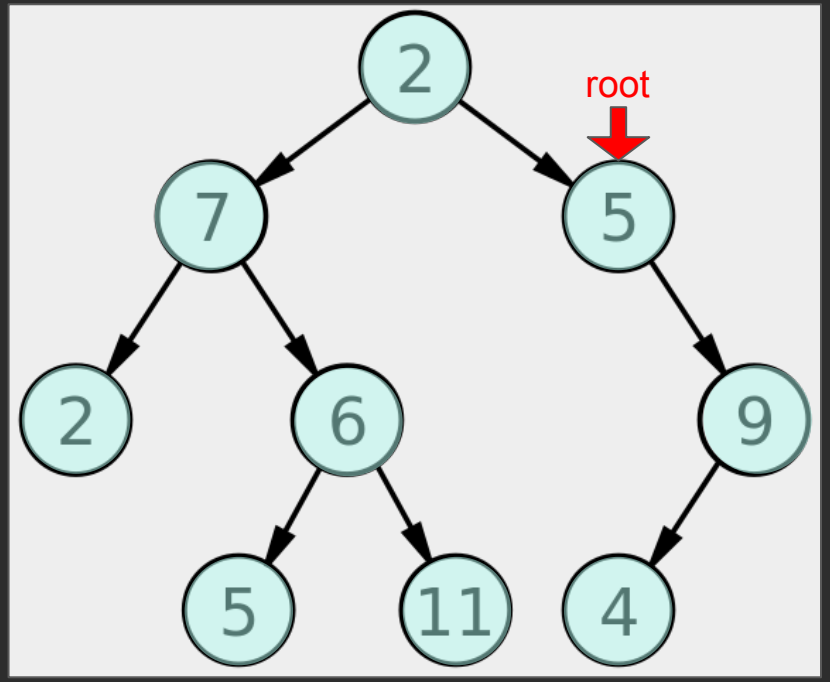
```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```



2
7
2
6
5
11
5
9
4

preOrder

```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```

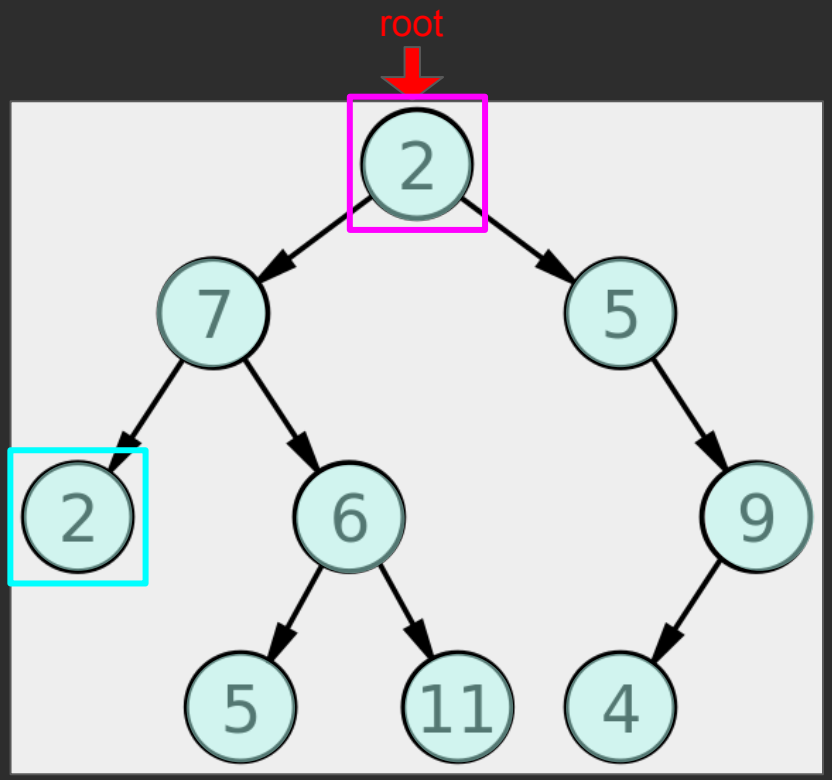


2
7
2
6
5
11
5
9
4

preOrder

```
void preOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 std::cout << root->getValue() << std::endl;
 preOrder(root->getLeft());
 preOrder(root->getRight());
}
```

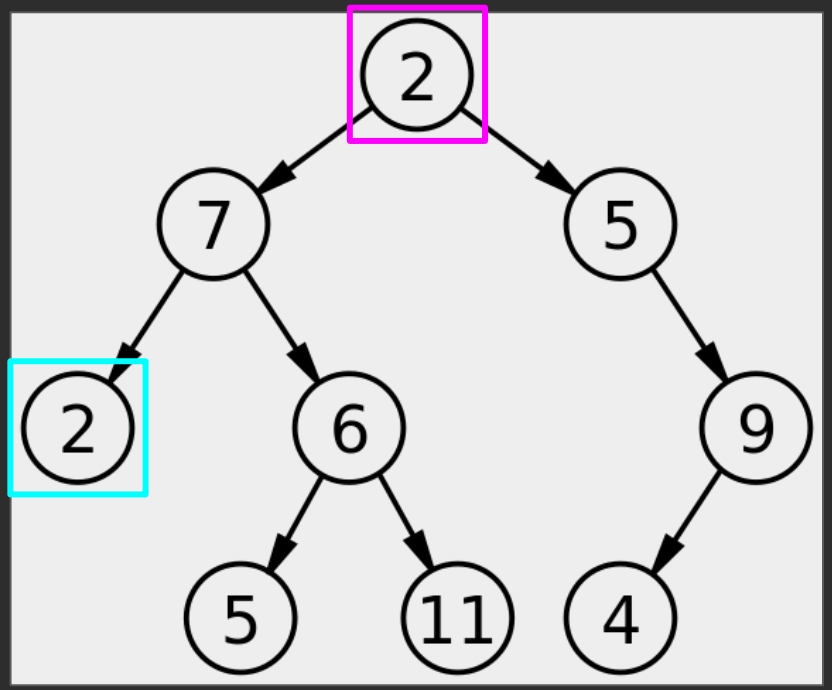
(Colored boxes added to disambiguate 2s)



inOrder

```
void inOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 inOrder(root->getLeft());
 std::cout << root->getValue() << std::endl;
 inOrder(root->getRight());
}
```

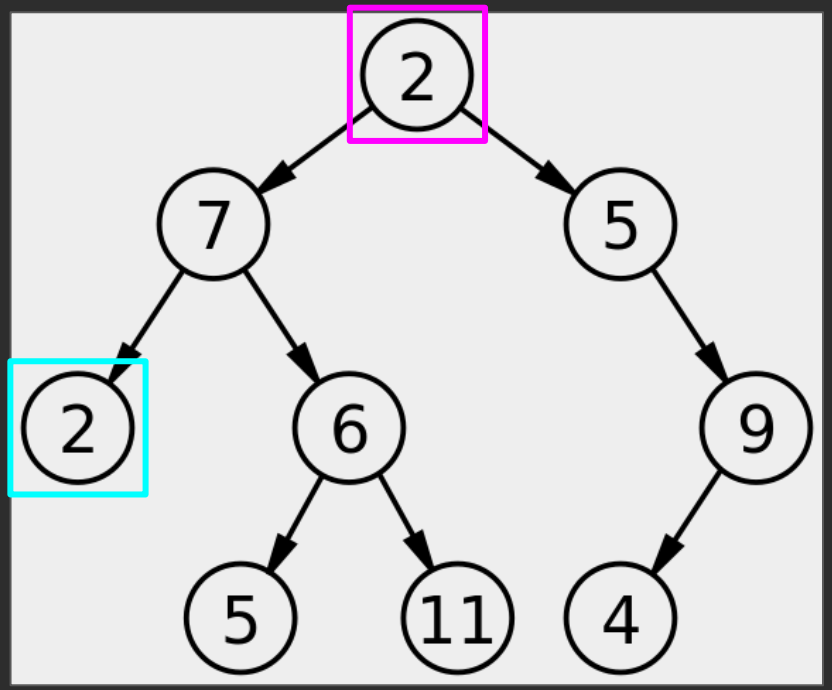
(Colored boxes added to disambiguate 2s)



postOrder

```
void postOrder(BinaryTree<int>* root) {
 if (root == nullptr) {
   return;
 }
 postOrder(root->getLeft());
 postOrder(root->getRight());
 std::cout << root->getValue() << std::endl;
}
```

(Colored boxes added to disambiguate 2s)



Big Questions!

- What are binary trees again?
- How do we traverse binary trees? (recursion!)
- How can I practice?

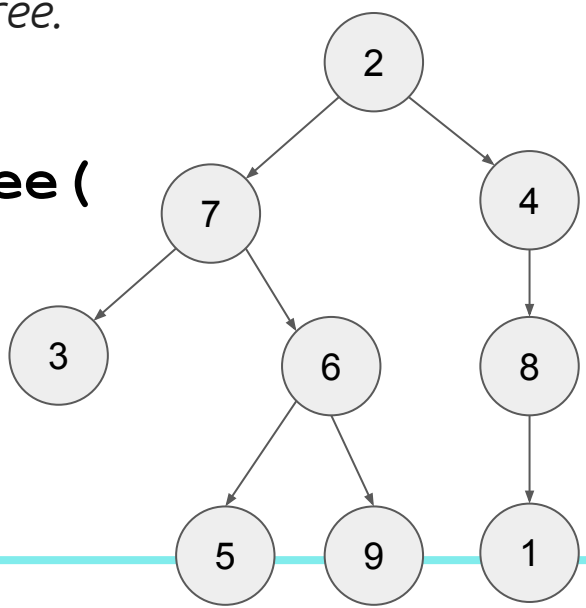
Great Question!

With an in-class activity!

`findMaxOfTree`

Instructions: Course Website or Blackboard -> Lectures -> Lecture 10

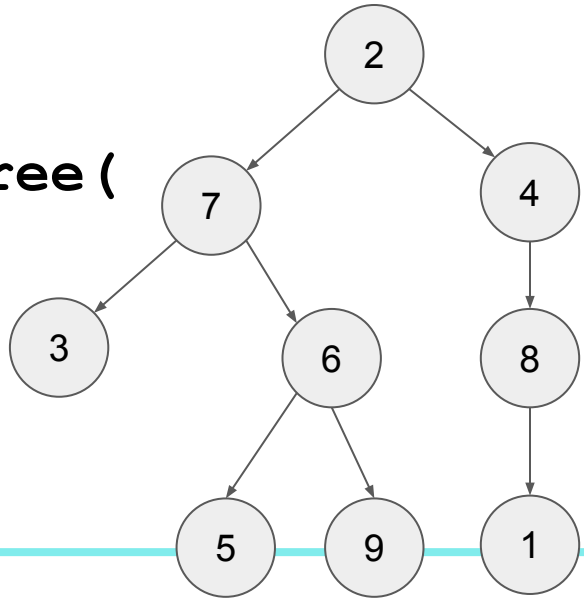
Write an algorithm that takes in a tree of ints, and returns the max of all the values within the tree.

`findMaxOfTree(`  `) outputs 10`

findHeightOfTree

Instructions: Course Website or Blackboard -> Lectures -> Lecture 10

Write an algorithm that takes in a tree, and returns the height of the tree.

findHeightOfTree(

) outputs 3

With an in-class activity!

Head over to course website or Blackboard

Let’s code it!!!

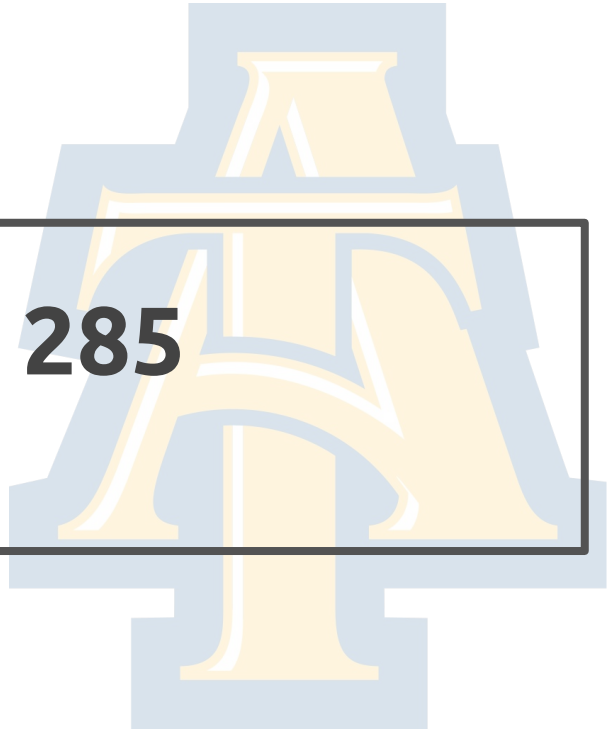
How was the pace today?

COMP - 285

Advanced Analysis of Algorithms

Welcome to COMP 285

Lecture 10: Binary Trees



Chris Lucas (cflucas@ncat.edu)