Traversals

In this lesson, we'll look at the different ways that a BST can be traversed.

We'll cover the following

- In-Order
- Pre-order
- Post-order

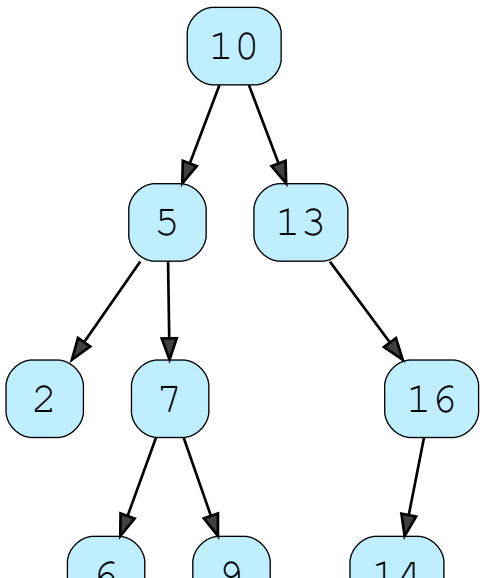
Knowing how to traverse a binary search will help in understanding graph traversals later on. Different types of traversals have different properties and use cases.

Even though you will not need to implement traversals over Binary Search Trees in competitions, this will be the basis for higher-level graph data structures.

We'll go over three types of traversals over binary trees (not only BSTs) in general:

- In-Order
- Pre-Order
- Post-Order

They differ in the order in which the left child, the right child, and the root are traversed recursively.



In-Order

Recursive Algorithm

1. Visit Left subtree
2. Visit Root
3. Visit Right Subtree

The order would be `2 5 6 7 9 10 13 14 16`

Note: In-order traversal of a BST gives all the nodes in *non-decreasing* order.

```
void in_order(struct Node* node)  {
    if (node == NULL)
        return;

    in_order(node -> left);
    cout << node -> data << " ";
    in_order(node -> right);
}
```

Pre-order

Recursive Algorithm

1. Visit Root
2. Visit Left subtree
3. Visit Right Subtree

The order would be `10 5 2 7 6 9 13 16 14`

```
void pre_order(struct Node* node) {
    if (node == NULL)
        return;

    cout << node -> data << " ";
    pre_order(node -> left);
    pre_order(node -> right);
}
```

Post-order

Recursive Algorithm

1. Visit Left subtree

2. Visit Right Subtree
3. Visit Root

The order would be `2 6 9 7 5 14 16 13 10`

```
void post_order(struct Node* node) {
    if (node == NULL)
        return;

    post_order(node -> left);
    post_order(node -> right);
    cout << node -> data << " ";
}
```

In the next lesson, we'll discuss a few properties of a BST.