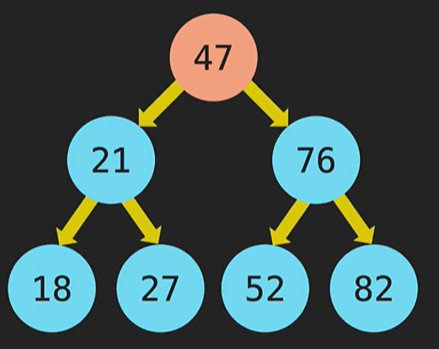
**Breadth First Search (BFS)**

We start at the root node then go down to the next layer and start from left to right and so on.

In this diagram, we would get: 47 🡪 21 🡪 76 🡪 18 🡪 27 🡪 52 🡪 82



We are going to use 2 arrays: queue[] and results[]

We put the root node into the queue: queue[47] results[]

Since 47 is the only node on that level, we will shift that value into the results[]: queue[] results[47]

If there is a left and right node of the 47 node, we push them into the queue[]: queue[21, 76] results[47]

Now we shift the 21 into results: queue[76] results[47, 21]

If there is a left and right node of 21, put them into the queue: queue[76, 18, 27] results[47, 21]

Once we added the left and right node to the queue, we shift the next node on that level into the results[]: queue[18, 27] results[47, 21, 76]

If there is a left and right node of 76, put them into the queue: queue[18, 27, 52, 82] results[47, 21, 76]

Now that layer is done.

Now we shift 18 into the results[]: queue[27, 52, 82] results[47, 21, 76, 18]

There are no left and right node so we add the 27 into the results[]: queue[52, 82] results[47, 21, 76, 18, 27]

There are no left and right node so we add the 52 into the results[]: queue[82] results[47, 21, 76, 18, 27, 52]

There are no left and right node so we add the 82 into the results[]: queue[] results[47, 21, 76, 18, 27, 82]

-------------------------------------------------

Step by step recap:

queue[] results[]

queue[47] results[]

queue[] results[47]

queue[21, 76] results[47]

queue[76] results[47, 21]

queue[76, 18, 27] results[47, 21]

queue[18, 27] results[47, 21, 76]

queue[18, 27, 52, 82] results[47, 21, 76]

queue[27, 52, 82] results[47, 21, 76, 18]

queue[52, 82] results[47, 21, 76, 18, 27]

queue[82] results[47, 21, 76, 18, 27, 52]

queue[] results[47, 21, 76, 18, 27, 82]

--------------------------------------------------

Summary:

1. We add a node into the queue[].
2. Then we shift that node from queue[] and add it into the results[].
3. Then we look to the left and right of the node we just added into the results[].
4. If there is a left or right node, then we would add them into the queue[].
5. Then we shift the next node from the queue[] into the results[].
6. Repeat steps 3 to 5 until the queue[] is empty.

class Node {

    constructor(value) {

        this.value = value;

        this.left = null;

        this.right = null;

    }

}

class BST {

    constructor() {

        this.root = null;

    }

    insert(value) {

        const newNode = new Node(value);

        if (this.root === null) {

            this.root = newNode;

            return this;

        }

        let temp = this.root;

        while (true) {

            if (newNode.value === temp.value) return undefined;

            if (newNode.value < temp.value) {

                if (temp.left === null) {

                    temp.left = newNode;

                    return this;

                }

                temp = temp.left;

            } else {

                if (temp.right === null) {

                    temp.right = newNode;

                    return this;

                }

                temp = temp.right;

            }

        }

    }

    contains(value) {

        if (this.root === null) return false;

        let temp = this.root;

        while (temp) {

            if (value < temp.value) {

                temp = temp.left;

            } else if (value > temp.value) {

                temp = temp.right;

            } else {

                return true;

            }

        }

        return false;

    }

    minValueNode(currentNode) {

        while (currentNode.left !== null) {

            currentNode = currentNode.left;

        }

        return currentNode;

    }

    BFS() {

        let currentNode = this.root;

        let results = [];

        let queue = [];

        queue.push(currentNode);

        while (queue.length) {

            currentNode = queue.shift();

            results.push(currentNode.value);

            if (currentNode.left) queue.push(currentNode.left);

            if (currentNode.right) queue.push(currentNode.right);

        }

        return results;

    }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Depth First Search (DFS)**

There are 3 types of DFS:

1. PreOrder
2. PostOrder
3. InOrder

**PreOrder**

Left first

We start with the root node then go down to the left until there are no nodes left, then look at the right.

47 🡪 21 🡪 18 🡪 27 🡪 76 🡪 52 🡪 82

A diagram of numbers and circles

Description automatically generated

DFSPreOrder() {

    let results = [];

    function traverse(currentNode) {

        results.push(currentNode.value);

        if (currentNode.left) traverse(currentNode.left);

        if (currentNode.right) traverse(currentNode.right);

    }

    traverse(this.root);

    return results;

}

**PostOrder**

Left first

We start with the root node then go down to the left until there are no nodes left, then look at the right.

18 🡪 27 🡪 21 🡪52 🡪82 🡪76 🡪47

A diagram of numbers and circles

Description automatically generated

DFSPostOrder() {

    let results = [];

    function traverse(currentNode) {

        if (currentNode.left) traverse(currentNode.left);

        if (currentNode.right) traverse(currentNode.right);

        results.push(currentNode.value);

    }

    traverse(this.root);

    return results;

}

**InOrder**

Left first

We start with the root node then go down to the left until there are no nodes left, then look at the right.

18 🡪 21 🡪 27 🡪 47 🡪 52 🡪 76 🡪 82

A diagram of numbers and circles

Description automatically generated

DFSInOrder() {

    let results = [];

    function traverse(currentNode) {

        if (currentNode.left) traverse(currentNode.left);

        results.push(currentNode.value);

        if (currentNode.right) traverse(currentNode.right);

    }

    traverse(this.root);

    return results;

}