function BST(value) {

this.value = value;

this.right = null;

this.left = null;

}

BST.prototype.insert = function(value) {

if (value <= this.value) {

if (!this.left) this.left = new BST(value);

else this.left.insert(value);

}

else if (value > this.value) {

if (!this.right) this.right = new BST(value);

else this.right.insert(value);

}

};

BST.prototype.contains = function(value) {

if (this.value === value) return true;

if (value < this.value) {

if (!this.left) return false;

else return this.left.contains(value);

}

else if (value > this.value) {

if (!this.right) return false;

else return this.right.contains(value);

}

};

BST.prototype.depthFirstTraversal = function(iteratorFunc, order) {

if (order === 'pre-order') iteratorFunc(this.value);

if (this.left) this.left.depthFirstTraversal(iteratorFunc, order);

if (order === 'in-order') iteratorFunc(this.value);

if (this.right) this.right.depthFirstTraversal(iteratorFunc, order);

if (order === 'post-order') iteratorFunc(this.value);

};

BST.prototype.breadthFirstTraversal = function(iteratorFunc) {

var queue = [this];

while (queue.length) {

var treeNode = queue.shift();

iteratorFunc(treeNode);

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

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

}

};

function log(value) {

console.log(value);

};

BST.prototype.getMinVal = function() {

if (this.left) return this.left.getMinVal();

else return this.value;

};

BST.prototype.getMaxVal = function() {

if (this.right) return this.right.getMaxVal();

else return this.value;

};

var bst = new BST(50);

bst.insert(30);

bst.insert(70);

bst.insert(100);

bst.insert(60);

bst.insert(59);

bst.insert(20);

bst.insert(45);

bst.insert(35);

bst.insert(85);

bst.insert(105);

bst.insert(10);

function log(node) {

console.log(node.value);

}

bst.breadthFirstTraversal(log);