```javascript
function Node(data) {
  this.data = data;
  this.left = null;
  this.right = null;
}

function BinarySearchTree() {
  this.root = null;
}

BinarySearchTree.prototype.add = function(data) {
  var node = new Node(data);
  if(!this.root) {
    this.root = node;
  } else {
    var current = this.root;
    while(current) {
      if(node.data < current.data) {
        if(!current.left) {
          current.left = node;
          break;
        }
        current = current.left;
      } else if (node.data > current.data) {
        if(!current.right) {
          current.right = node;
          break;
        }
        current = current.right;
      } else {
        break;
      }
    }
  }
};
BinarySearchTree.prototype.remove = function(data) {
  var that = this;
  var removeNode = function(node, data) {
    if(!node) {
      return null;
    }
    if(data === node.data) {
      if(!node.left && !node.right) {
        return null;
      }
      if(!node.left) {
        return node.right;
      }
      if(!node.right) {
        return node.left;
      }
      // 2 children
      var temp = that.getMin(node.right);
      node.data = temp;
      node.right = removeNode(node.right, temp);
      return node;
    } else if(data < node.data) {
      node.left = removeNode(node.left, data);
      return node;
    } else {
      node.right = removeNode(node.right, data);
      return node;
    }
  };
  this.root = removeNode(this.root, data);
};
BinarySearchTree.prototype.contains = function(data) {
  var current = this.root;
  while(current) {
    if(data === current.data) {
      return true;
    }
    if(data < current.data) {
      current = current.left;
    } else {
      current = current.right;
    }
  }
  return false;
};
BinarySearchTree.prototype._preOrder = function(node, fn) {
  if(node) {
    if(fn) {
      fn(node);
    }
    this._preOrder(node.left, fn);
    this._preOrder(node.right, fn);
  }
};
BinarySearchTree.prototype._inOrder = function(node, fn) {
  if(node) {
    this._inOrder(node.left, fn);
    if(fn) {
      fn(node);
    }
    this._inOrder(node.right, fn);
  }
};
BinarySearchTree.prototype._postOrder = function(node, fn) {
  if(node) {
    this._postOrder(node.left, fn);
    this._postOrder(node.right, fn);
    if(fn) {
      fn(node);
    }
  }
};
BinarySearchTree.prototype.traverseDFS = function(fn, method) {
  var current = this.root;
  if(method) {
    this['_' + method](current, fn);
  } else {
    this._preOrder(current, fn);
  }
};
BinarySearchTree.prototype.traverseBFS = function(fn) {
  this.queue = [];
  this.queue.push(this.root);
  while(this.queue.length) {
    var node = this.queue.shift();
    if(fn) {
      fn(node);
    }
    if(node.left) {
      this.queue.push(node.left);
    }
    if(node.right) {
      this.queue.push(node.right);
    }
  }
};
BinarySearchTree.prototype.print = function() {
  if(!this.root) {
    return console.log('No root node found');
  }
  var newline = new Node('|');
  var queue = [this.root, newline];
  var string = '';
  while(queue.length) {
    var node = queue.shift();
    string += node.data.toString() + ' ';
    if(node === newline && queue.length) {
      queue.push(newline);
    }
    if(node.left) {
      queue.push(node.left);
    }
    if(node.right) {
      queue.push(node.right);
    }
  }
  console.log(string.slice(0, -2).trim());
};
BinarySearchTree.prototype.printByLevel = function() {
  if(!this.root) {
    return console.log('No root node found');
  }
  var newline = new Node('\n');
  var queue = [this.root, newline];
  var string = '';
  while(queue.length) {
    var node = queue.shift();
    string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
    if(node === newline && queue.length) {
      queue.push(newline);
    }
    if(node.left) {
      queue.push(node.left);
    }
    if(node.right) {
      queue.push(node.right);
    }
  }
  console.log(string.trim());
};
BinarySearchTree.prototype.getMin = function(node) {
  if(!node) {
    node = this.root;
  }
  while(node.left) {
```

```javascript
    node = node.left;
  }
  return node.data;
};
BinarySearchTree.prototype.getMax = function(node) {
  if(!node) {
    node = this.root;
  }
  while(node.right) {
    node = node.right;
  }
  return node.data;
};
BinarySearchTree.prototype._getHeight = function(node) {
  if(!node) {
    return -1;
  }
  var left = this._getHeight(node.left);
  var right = this._getHeight(node.right);
  return Math.max(left, right) + 1;
};
BinarySearchTree.prototype.getHeight = function(node) {
  if(!node) {
    node = this.root;
  }
  return this._getHeight(node);
};
BinarySearchTree.prototype._isBalanced = function(node) {
  if(!node) {
    return true;
  }
  var heigthLeft = this._getHeight(node.left);
  var heigthRight = this._getHeight(node.right);
  var diff = Math.abs(heigthLeft - heigthRight);
  if(diff > 1) {
    return false;
  } else {
    return this._isBalanced(node.left) && this._isBalanced(node.right);
  }
};
BinarySearchTree.prototype.isBalanced = function(node) {
  if(!node) {
    node = this.root;
  }
  return this._isBalanced(node);
};
BinarySearchTree.prototype._checkHeight = function(node) {
  if(!node) {
    return 0;
  }
  var left = this._checkHeight(node.left);
  if(left === -1) {
    return -1;
  }
  var right = this._checkHeight(node.right);
  if(right === -1) {
    return -1;
  }
  var diff = Math.abs(left - right);
  if(diff > 1) {
    return -1;
  } else {
    return Math.max(left, right) + 1;
  }
};
BinarySearchTree.prototype.isBalancedOptimized = function(node) {
  if(!node) {
    node = this.root;
  }
  if(!node) {
    return true;
  }
  if(this._checkHeight(node) === -1) {
    return false;
  } else {
    return true;
  }
};

var binarySearchTree = new BinarySearchTree();
binarySearchTree.add(5);
binarySearchTree.add(3);
binarySearchTree.add(7);
binarySearchTree.add(2);
binarySearchTree.add(4);
binarySearchTree.add(4);
binarySearchTree.add(6);
binarySearchTree.add(8);
binarySearchTree.print(); // => 5 | 3 7 | 2 4 6 8
binarySearchTree.printByLevel(); // => 5 \n 3 7 \n 2 4 6 8
console.log('--- DFS inOrder');
binarySearchTree.traverseDFS(function(node) { console.log(node.data); }, 'inOrder');
// => 2 3 4 5 6 7 8
console.log('--- DFS preOrder');
binarySearchTree.traverseDFS(function(node) { console.log(node.data); }, 'preOrder');
// => 5 3 2 4 7 6 8
console.log('--- DFS postOrder');
binarySearchTree.traverseDFS(function(node) { console.log(node.data); },
'postOrder'); // => 2 4 3 6 8 7 5
console.log('--- BFS');
binarySearchTree.traverseBFS(function(node) { console.log(node.data); }); // => 5 3 7
2 4 6 8
console.log('min is 2:', binarySearchTree.getMin()); // => 2
console.log('max is 8:', binarySearchTree.getMax()); // => 8
console.log('tree contains 3 is true:', binarySearchTree.contains(3)); // => true
console.log('tree contains 9 is false:', binarySearchTree.contains(9)); // => false
console.log('tree height is 2:', binarySearchTree.getHeight()); // => 2
console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
binarySearchTree.remove(11); // remove non existing node
binarySearchTree.print(); // => 5 | 3 7 | 2 4 6 8
binarySearchTree.remove(5); // remove 5, 6 goes up
binarySearchTree.print(); // => 6 | 3 7 | 2 4 8
binarySearchTree.remove(7); // remove 7, 8 goes up
binarySearchTree.print(); // => 6 | 3 8 | 2 4
binarySearchTree.remove(8); // remove 8, the tree becomes unbalanced
binarySearchTree.print(); // => 6 | 3 | 2 4
console.log('tree is balanced is false:', binarySearchTree.isBalanced()); // => true
binarySearchTree.remove(4);
binarySearchTree.remove(2);
binarySearchTree.remove(3);
binarySearchTree.remove(6);
binarySearchTree.print(); // => 'No root node found'
binarySearchTree.printByLevel(); // => 'No root node found'
console.log('tree height is -1:', binarySearchTree.getHeight()); // => -1
console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
console.log('---');
binarySearchTree.add(10);
console.log('tree height is 0:', binarySearchTree.getHeight()); // => 0
console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
binarySearchTree.add(6);
binarySearchTree.add(14);
binarySearchTree.add(4);
binarySearchTree.add(8);
binarySearchTree.add(12);
binarySearchTree.add(16);
binarySearchTree.add(3);
binarySearchTree.add(5);
binarySearchTree.add(7);
binarySearchTree.add(9);
binarySearchTree.add(11);
binarySearchTree.add(13);
binarySearchTree.add(15);
binarySearchTree.add(17);
binarySearchTree.print(); // => 10 | 6 14 | 4 8 12 16 | 3 5 7 9 11 13 15 17
binarySearchTree.remove(10); // remove 10, 11 goes up
binarySearchTree.print(); // => 11 | 6 14 | 4 8 12 16 | 3 5 7 9 x 13 15 17
binarySearchTree.remove(12); // remove 12; 13 goes up
binarySearchTree.print(); // => 11 | 6 14 | 4 8 13 16 | 3 5 7 9 x x 15 17
console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
console.log('tree is balanced optimized is true:',
binarySearchTree.isBalancedOptimized()); // => true
binarySearchTree.remove(13); // remove 13, 13 has no children so nothing changes
binarySearchTree.print(); // => 11 | 6 14 | 4 8 x 16 | 3 5 7 9 x x 15 17
console.log('tree is balanced is false:', binarySearchTree.isBalanced()); // => false
console.log('tree is balanced optimized is false:',
binarySearchTree.isBalancedOptimized()); // => false

```