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

class BinarySearchTree {
  constructor() {
    this.root = null;
  }

  add(data) {
    const node = new Node(data);
    if(!this.root) {
      this.root = node;
    } else {
      let 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;
        }
      }
    }
  }

  remove(data) {
    const that = this;
    const removeNode = (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
        const 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);
```

```
61          return node;
62        } else {
63          node.right = removeNode(node.right, data);
64          return node;
65        }
66      };
67      this.root = removeNode(this.root, data);
68    }
69
70    contains(data) {
71      let current = this.root;
72      while(current) {
73        if(data === current.data) {
74          return true;
75        }
76        if(data < current.data) {
77          current = current.left;
78        } else {
79          current = current.right;
80        }
81      }
82      return false;
83    }
84
85    _preOrder(node, fn) {
86      if(node) {
87        if(fn) {
88          fn(node);
89        }
90        this._preOrder(node.left, fn);
91        this._preOrder(node.right, fn);
92      }
93    }
94
95    _inOrder(node, fn) {
96      if(node) {
97        this._inOrder(node.left, fn);
98        if(fn) {
99          fn(node);
100       }
101       this._inOrder(node.right, fn);
102     }
103   }
104
105   _postOrder(node, fn) {
106     if(node) {
107       this._postOrder(node.left, fn);
108       this._postOrder(node.right, fn);
109       if(fn) {
110         fn(node);
111       }
112     }
113   }
114
115   traverseDFS(fn, method) {
116     const current = this.root;
117     if(method) {
118       this[`_${method}`](current, fn);
119     } else {
120       this._preOrder(current, fn);
```

```
    }
  }

  traverseBFS(fn) {
    this.queue = [];
    this.queue.push(this.root);
    while(this.queue.length) {
      const node = this.queue.shift();
      if(fn) {
        fn(node);
      }
      if(node.left) {
        this.queue.push(node.left);
      }
      if(node.right) {
        this.queue.push(node.right);
      }
    }
  }

  print() {
    if(!this.root) {
      return console.log('No root node found');
    }
    const newline = new Node('|');
    const queue = [this.root, newline];
    let string = '';
    while(queue.length) {
      const 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());
  }

  printByLevel() {
    if(!this.root) {
      return console.log('No root node found');
    }
    const newline = new Node('\n');
    const queue = [this.root, newline];
    let string = '';
    while(queue.length) {
      const 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());
  }

  getMin(node) {
    if(!node) {
      node = this.root;
    }
    while(node.left) {
      node = node.left;
    }
    return node.data;
  }

  getMax(node) {
    if(!node) {
      node = this.root;
    }
    while(node.right) {
      node = node.right;
    }
    return node.data;
  }

  _getHeight(node) {
    if(!node) {
      return -1;
    }
    const left = this._getHeight(node.left);
    const right = this._getHeight(node.right);
    return Math.max(left, right) + 1;
  }

  getHeight(node) {
    if(!node) {
      node = this.root;
    }
    return this._getHeight(node);
  }

  _isBalanced(node) {
    if(!node) {
      return true;
    }
    const heigthLeft = this._getHeight(node.left);
    const heigthRight = this._getHeight(node.right);
    const diff = Math.abs(heigthLeft - heigthRight);
    if(diff > 1) {
      return false;
    } else {
      return this._isBalanced(node.left) && this._isBalanced(node.right);
    }
  }

  isBalanced(node) {
    if(!node) {
      node = this.root;
    }
```

```
    return this._isBalanced(node);
  }

  _checkHeight(node) {
    if(!node) {
      return 0;
    }
    const left = this._checkHeight(node.left);
    if(left === -1) {
      return -1;
    }
    const right = this._checkHeight(node.right);
    if(right === -1) {
      return -1;
    }
    const diff = Math.abs(left - right);
    if(diff > 1) {
      return -1;
    } else {
      return Math.max(left, right) + 1;
    }
  }

  isBalancedOptimized(node) {
    if(!node) {
      node = this.root;
    }
    if(!node) {
      return true;
    }
    if(this._checkHeight(node) === -1) {
      return false;
    } else {
      return true;
    }
  }
}

const 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(node => { console.log(node.data); }, 'inOrder'); // => 2 3 4 5 6 7 8
console.log('--- DFS preOrder');
binarySearchTree.traverseDFS(node => { console.log(node.data); }, 'preOrder'); // => 5 3 2 4 7 6 8
console.log('--- DFS postOrder');
binarySearchTree.traverseDFS(node => { console.log(node.data); }, 'postOrder'); // => 2 4 3 6 8 7 5
console.log('--- BFS');
```

```
297 binarySearchTree.traverseBFS(node => { console.log(node.data); }); // => 5 3 7 2 4 6
    8
298 console.log('min is 2:', binarySearchTree.getMin()); // => 2
299 console.log('max is 8:', binarySearchTree.getMax()); // => 8
300 console.log('tree contains 3 is true:', binarySearchTree.contains(3)); // => true
301 console.log('tree contains 9 is false:', binarySearchTree.contains(9)); // => false
302 console.log('tree height is 2:', binarySearchTree.getHeight()); // => 2
303 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
304 binarySearchTree.remove(11); // remove non existing node
305 binarySearchTree.print(); // => 5 | 3 7 | 2 4 6 8
306 binarySearchTree.remove(5); // remove 5, 6 goes up
307 binarySearchTree.print(); // => 6 | 3 7 | 2 4 8
308 binarySearchTree.remove(7); // remove 7, 8 goes up
309 binarySearchTree.print(); // => 6 | 3 8 | 2 4
310 binarySearchTree.remove(8); // remove 8, the tree becomes unbalanced
311 binarySearchTree.print(); // => 6 | 3 | 2 4
312 console.log('tree is balanced is false:', binarySearchTree.isBalanced()); // => true
313 binarySearchTree.remove(4);
314 binarySearchTree.remove(2);
315 binarySearchTree.remove(3);
316 binarySearchTree.remove(6);
317 binarySearchTree.print(); // => 'No root node found'
318 binarySearchTree.printByLevel(); // => 'No root node found'
319 console.log('tree height is -1:', binarySearchTree.getHeight()); // => -1
320 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
321 console.log('---');
322 binarySearchTree.add(10);
323 console.log('tree height is 0:', binarySearchTree.getHeight()); // => 0
324 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
325 binarySearchTree.add(6);
326 binarySearchTree.add(14);
327 binarySearchTree.add(4);
328 binarySearchTree.add(8);
329 binarySearchTree.add(12);
330 binarySearchTree.add(16);
331 binarySearchTree.add(3);
332 binarySearchTree.add(5);
333 binarySearchTree.add(7);
334 binarySearchTree.add(9);
335 binarySearchTree.add(11);
336 binarySearchTree.add(13);
337 binarySearchTree.add(15);
338 binarySearchTree.add(17);
339 binarySearchTree.print(); // => 10 | 6 14 | 4 8 12 16 | 3 5 7 9 11 13 15 17
340 binarySearchTree.remove(10); // remove 10, 11 goes up
341 binarySearchTree.print(); // => 11 | 6 14 | 4 8 12 16 | 3 5 7 9 x 13 15 17
342 binarySearchTree.remove(12); // remove 12; 13 goes up
343 binarySearchTree.print(); // => 11 | 6 14 | 4 8 13 16 | 3 5 7 9 x x 15 17
344 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
345 console.log('tree is balanced optimized is true:',
    binarySearchTree.isBalancedOptimized()); // => true
346 binarySearchTree.remove(13); // remove 13, 13 has no children so nothing changes
347 binarySearchTree.print(); // => 11 | 6 14 | 4 8 x 16 | 3 5 7 9 x x 15 17
348 console.log('tree is balanced is false:', binarySearchTree.isBalanced()); // => false
349 console.log('tree is balanced optimized is false:',
    binarySearchTree.isBalancedOptimized()); // => false
350
```