

```
1 function Node(data) {
2   this.data = data;
3   this.left = null;
4   this.right = null;
5 }
6
7 function BinarySearchTree() {
8   this.root = null;
9 }
10
11 BinarySearchTree.prototype.add = function(data) {
12   var node = new Node(data);
13   if(!this.root) {
14     this.root = node;
15   } else {
16     var current = this.root;
17     while(current) {
18       if(node.data < current.data) {
19         if(!current.left) {
20           current.left = node;
21           break;
22         }
23         current = current.left;
24       } else if (node.data > current.data) {
25         if(!current.right) {
26           current.right = node;
27           break;
28         }
29         current = current.right;
30       } else {
31         break;
32       }
33     }
34   }
35 };
36 BinarySearchTree.prototype.remove = function(data) {
37   var that = this;
38   var removeNode = function(node, data) {
39     if(!node) {
40       return null;
41     }
42     if(data === node.data) {
43       if(!node.left && !node.right) {
44         return null;
45       }
46       if(!node.left) {
47         return node.right;
48       }
49       if(!node.right) {
50         return node.left;
51       }
52       // 2 children
53       var temp = that.getMin(node.right);
54       node.data = temp;
55       node.right = removeNode(node.right, temp);
56       return node;
57     } else if(data < node.data) {
58       node.left = removeNode(node.left, data);
59       return node;
60     } else {
```

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

```
121     if(fn) {
122         fn(node);
123     }
124     if(node.left) {
125         this.queue.push(node.left);
126     }
127     if(node.right) {
128         this.queue.push(node.right);
129     }
130 }
131 };
132 BinarySearchTree.prototype.print = function() {
133     if(!this.root) {
134         return console.log('No root node found');
135     }
136     var newline = new Node('|');
137     var queue = [this.root, newline];
138     var string = '';
139     while(queue.length) {
140         var node = queue.shift();
141         string += node.data.toString() + ' ';
142         if(node === newline && queue.length) {
143             queue.push(newline);
144         }
145         if(node.left) {
146             queue.push(node.left);
147         }
148         if(node.right) {
149             queue.push(node.right);
150         }
151     }
152     console.log(string.slice(0, -2).trim());
153 };
154 BinarySearchTree.prototype.printByLevel = function() {
155     if(!this.root) {
156         return console.log('No root node found');
157     }
158     var newline = new Node('\n');
159     var queue = [this.root, newline];
160     var string = '';
161     while(queue.length) {
162         var node = queue.shift();
163         string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
164         if(node === newline && queue.length) {
165             queue.push(newline);
166         }
167         if(node.left) {
168             queue.push(node.left);
169         }
170         if(node.right) {
171             queue.push(node.right);
172         }
173     }
174     console.log(string.trim());
175 };
176 BinarySearchTree.prototype.getMin = function(node) {
177     if(!node) {
178         node = this.root;
179     }
180     while(node.left) {
```

```
181     node = node.left;
182 }
183 return node.data;
184 };
185 BinarySearchTree.prototype.getMax = function(node) {
186     if(!node) {
187         node = this.root;
188     }
189     while(node.right) {
190         node = node.right;
191     }
192     return node.data;
193 };
194 BinarySearchTree.prototype._getHeight = function(node) {
195     if(!node) {
196         return -1;
197     }
198     var left = this._getHeight(node.left);
199     var right = this._getHeight(node.right);
200     return Math.max(left, right) + 1;
201 };
202 BinarySearchTree.prototype.getHeight = function(node) {
203     if(!node) {
204         node = this.root;
205     }
206     return this._getHeight(node);
207 };
208 BinarySearchTree.prototype._isBalanced = function(node) {
209     if(!node) {
210         return true;
211     }
212     var heightLeft = this._getHeight(node.left);
213     var heightRight = this._getHeight(node.right);
214     var diff = Math.abs(heightLeft - heightRight);
215     if(diff > 1) {
216         return false;
217     } else {
218         return this._isBalanced(node.left) && this._isBalanced(node.right);
219     }
220 };
221 BinarySearchTree.prototype.isBalanced = function(node) {
222     if(!node) {
223         node = this.root;
224     }
225     return this._isBalanced(node);
226 };
227 BinarySearchTree.prototype._checkHeight = function(node) {
228     if(!node) {
229         return 0;
230     }
231     var left = this._checkHeight(node.left);
232     if(left === -1) {
233         return -1;
234     }
235     var right = this._checkHeight(node.right);
236     if(right === -1) {
237         return -1;
238     }
239     var diff = Math.abs(left - right);
240     if(diff > 1) {
```

```

241     return -1;
242 } else {
243     return Math.max(left, right) + 1;
244 }
245 };
246 BinarySearchTree.prototype.isBalancedOptimized = function(node) {
247     if(!node) {
248         node = this.root;
249     }
250     if(!node) {
251         return true;
252     }
253     if(this._checkHeight(node) === -1) {
254         return false;
255     } else {
256         return true;
257     }
258 };
259
260 var binarySearchTree = new BinarySearchTree();
261 binarySearchTree.add(5);
262 binarySearchTree.add(3);
263 binarySearchTree.add(7);
264 binarySearchTree.add(2);
265 binarySearchTree.add(4);
266 binarySearchTree.add(4);
267 binarySearchTree.add(6);
268 binarySearchTree.add(8);
269 binarySearchTree.print(); // => 5 | 3 7 | 2 4 6 8
270 binarySearchTree.printByLevel(); // => 5 \n 3 7 \n 2 4 6 8
271 console.log('--- DFS inOrder');
272 binarySearchTree.traverseDFS(function(node) { console.log(node.data); }, 'inOrder');
    // => 2 3 4 5 6 7 8
273 console.log('--- DFS preOrder');
274 binarySearchTree.traverseDFS(function(node) { console.log(node.data); }, 'preOrder');
    // => 5 3 2 4 7 6 8
275 console.log('--- DFS postOrder');
276 binarySearchTree.traverseDFS(function(node) { console.log(node.data); },
    'postOrder'); // => 2 4 3 6 8 7 5
277 console.log('--- BFS');
278 binarySearchTree.traverseBFS(function(node) { console.log(node.data); }); // => 5 3 7
    2 4 6 8
279 console.log('min is 2:', binarySearchTree.getMin()); // => 2
280 console.log('max is 8:', binarySearchTree.getMax()); // => 8
281 console.log('tree contains 3 is true:', binarySearchTree.contains(3)); // => true
282 console.log('tree contains 9 is false:', binarySearchTree.contains(9)); // => false
283 console.log('tree height is 2:', binarySearchTree.getHeight()); // => 2
284 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
285 binarySearchTree.remove(11); // remove non existing node
286 binarySearchTree.print(); // => 5 | 3 7 | 2 4 6 8
287 binarySearchTree.remove(5); // remove 5, 6 goes up
288 binarySearchTree.print(); // => 6 | 3 7 | 2 4 8
289 binarySearchTree.remove(7); // remove 7, 8 goes up
290 binarySearchTree.print(); // => 6 | 3 8 | 2 4
291 binarySearchTree.remove(8); // remove 8, the tree becomes unbalanced
292 binarySearchTree.print(); // => 6 | 3 | 2 4
293 console.log('tree is balanced is false:', binarySearchTree.isBalanced()); // => true
294 binarySearchTree.remove(4);
295 binarySearchTree.remove(2);
296 binarySearchTree.remove(3);

```

```
297 binarySearchTree.remove(6);
298 binarySearchTree.print(); // => 'No root node found'
299 binarySearchTree.printByLevel(); // => 'No root node found'
300 console.log('tree height is -1:', binarySearchTree.getHeight()); // => -1
301 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
302 console.log('---');
303 binarySearchTree.add(10);
304 console.log('tree height is 0:', binarySearchTree.getHeight()); // => 0
305 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
306 binarySearchTree.add(6);
307 binarySearchTree.add(14);
308 binarySearchTree.add(4);
309 binarySearchTree.add(8);
310 binarySearchTree.add(12);
311 binarySearchTree.add(16);
312 binarySearchTree.add(3);
313 binarySearchTree.add(5);
314 binarySearchTree.add(7);
315 binarySearchTree.add(9);
316 binarySearchTree.add(11);
317 binarySearchTree.add(13);
318 binarySearchTree.add(15);
319 binarySearchTree.add(17);
320 binarySearchTree.print(); // => 10 | 6 14 | 4 8 12 16 | 3 5 7 9 11 13 15 17
321 binarySearchTree.remove(10); // remove 10, 11 goes up
322 binarySearchTree.print(); // => 11 | 6 14 | 4 8 12 16 | 3 5 7 9 x 13 15 17
323 binarySearchTree.remove(12); // remove 12; 13 goes up
324 binarySearchTree.print(); // => 11 | 6 14 | 4 8 13 16 | 3 5 7 9 x x 15 17
325 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
326 console.log('tree is balanced optimized is true:',
  binarySearchTree.isBalancedOptimized()); // => true
327 binarySearchTree.remove(13); // remove 13, 13 has no children so nothing changes
328 binarySearchTree.print(); // => 11 | 6 14 | 4 8 x 16 | 3 5 7 9 x x 15 17
329 console.log('tree is balanced is false:', binarySearchTree.isBalanced()); // => false
330 console.log('tree is balanced optimized is false:',
  binarySearchTree.isBalancedOptimized()); // => false
331
```