

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

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

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

```
241     return this._isBalanced(node);
242 }
243
244 _checkHeight(node) {
245     if(!node) {
246         return 0;
247     }
248     const left = this._checkHeight(node.left);
249     if(left === -1) {
250         return -1;
251     }
252     const right = this._checkHeight(node.right);
253     if(right === -1) {
254         return -1;
255     }
256     const diff = Math.abs(left - right);
257     if(diff > 1) {
258         return -1;
259     } else {
260         return Math.max(left, right) + 1;
261     }
262 }
263
264 isBalancedOptimized(node) {
265     if(!node) {
266         node = this.root;
267     }
268     if(!node) {
269         return true;
270     }
271     if(this._checkHeight(node) === -1) {
272         return false;
273     } else {
274         return true;
275     }
276 }
277 }
278
279 const binarySearchTree = new BinarySearchTree();
280 binarySearchTree.add(5);
281 binarySearchTree.add(3);
282 binarySearchTree.add(7);
283 binarySearchTree.add(2);
284 binarySearchTree.add(4);
285 binarySearchTree.add(4);
286 binarySearchTree.add(6);
287 binarySearchTree.add(8);
288 binarySearchTree.print(); // => 5 | 3 7 | 2 4 6 8
289 binarySearchTree.printByLevel(); // => 5 \n 3 7 \n 2 4 6 8
290 console.log('--- DFS inOrder');
291 binarySearchTree.traverseDFS(node => { console.log(node.data); }, 'inOrder'); // => 2
3 4 5 6 7 8
292 console.log('--- DFS preOrder');
293 binarySearchTree.traverseDFS(node => { console.log(node.data); }, 'preOrder'); // =>
5 3 2 4 7 6 8
294 console.log('--- DFS postOrder');
295 binarySearchTree.traverseDFS(node => { console.log(node.data); }, 'postOrder'); // =>
2 4 3 6 8 7 5
296 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
```