

```
1 function Node(data) {
2   this.data = data;
3   this.children = [];
4 }
5
6 class Tree {
7   constructor() {
8     this.root = null;
9   }
10
11   add(data, toNodeData) {
12     const node = new Node(data);
13     const parent = toNodeData ? this.findBFS(toNodeData) : null;
14     if(parent) {
15       parent.children.push(node);
16     } else {
17       if(!this.root) {
18         this.root = node;
19       } else {
20         return 'Root node is already assigned';
21       }
22     }
23   }
24
25   remove(data) {
26     if(this.root.data === data) {
27       this.root = null;
28     }
29
30     const queue = [this.root];
31     while(queue.length) {
32       const node = queue.shift();
33       for (let [index, child] of node.children.entries()) {
34         if(child.data === data) {
35           node.children.splice(index, 1);
36         } else {
37           queue.push(child);
38         }
39       }
40     }
41   }
42
43   contains(data) {
44     return !!this.findBFS(data);
45   }
46
47   findBFS(data) {
48     const queue = [this.root];
49     while(queue.length) {
50       const node = queue.shift();
51       if(node.data === data) {
52         return node;
53       }
54       for(const child of node.children) {
55         queue.push(child);
56       }
57     }
58     return null;
59   }
60 }
```

```
61 _preOrder(node, fn) {
62   if(node) {
63     if(fn) {
64       fn(node);
65     }
66     for(const child of node.children) {
67       this._preOrder(child, fn);
68     }
69   }
70 }
71
72 _postOrder(node, fn) {
73   if(node) {
74     for(const child of node.children) {
75       this._postOrder(child, fn);
76     }
77     if(fn) {
78       fn(node);
79     }
80   }
81 }
82
83 traverseDFS(fn, method) {
84   const current = this.root;
85   if(method) {
86     this[`_${method}`](current, fn);
87   } else {
88     this._preOrder(current, fn);
89   }
90 }
91
92 traverseBFS(fn) {
93   const queue = [this.root];
94   while(queue.length) {
95     const node = queue.shift();
96     if(fn) {
97       fn(node);
98     }
99     for(const child of node.children) {
100       queue.push(child);
101     }
102   }
103 }
104
105 print() {
106   if(!this.root) {
107     return console.log('No root node found');
108   }
109   const newline = new Node('|');
110   const queue = [this.root, newline];
111   let string = '';
112   while(queue.length) {
113     const node = queue.shift();
114     string += `${node.data.toString()} `;
115     if(node === newline && queue.length) {
116       queue.push(newline);
117     }
118     for(const child of node.children) {
119       queue.push(child);
120     }
121   }
122 }
```

```
121     }
122     console.log(string.slice(0, -2).trim());
123 }
124
125 printByLevel() {
126     if(!this.root) {
127         return console.log('No root node found');
128     }
129     const newline = new Node('\n');
130     const queue = [this.root, newline];
131     let string = '';
132     while(queue.length) {
133         const node = queue.shift();
134         string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
135         if(node === newline && queue.length) {
136             queue.push(newline);
137         }
138         for(const child of node.children) {
139             queue.push(child);
140         }
141     }
142     console.log(string.trim());
143 }
144 }
145
146 const tree = new Tree();
147 tree.add('ceo');
148 tree.add('cto', 'ceo');
149 tree.add('dev1', 'cto');
150 tree.add('dev2', 'cto');
151 tree.add('dev3', 'cto');
152 tree.add('cfo', 'ceo');
153 tree.add('accountant', 'cfo');
154 tree.add('cmo', 'ceo');
155 tree.print(); // => ceo | cto cfo cmo | dev1 dev2 dev3 accountant
156 tree.printByLevel(); // => ceo \n cto cfo cmo \n dev1 dev2 dev3 accountant
157 console.log('tree contains dev1 is true:', tree.contains('dev1')); // => true
158 console.log('tree contains dev4 is false:', tree.contains('dev4')); // => false
159 console.log('--- BFS');
160 tree.traverseBFS(node => { console.log(node.data); }); // => ceo cto cfo cmo dev1
    dev2 dev3 accountant
161 console.log('--- DFS preOrder');
162 tree.traverseDFS(node => { console.log(node.data); }, 'preOrder'); // => ceo cto dev1
    dev2 dev3 cfo accountant cmo
163 console.log('--- DFS postOrder');
164 tree.traverseDFS(node => { console.log(node.data); }, 'postOrder'); // => dev1 dev2
    dev3 cto accountant cfo cmo ceo
165 tree.remove('cmo');
166 tree.print(); // => ceo | cto cfo | dev1 dev2 dev3 accountant
167 tree.remove('cfo');
168 tree.print(); // => ceo | cto | dev1 dev2 dev3
169
```