

```
1 function Node(data) {
2   this.data = data;
3   this.children = [];
4 }
5
6 function Tree() {
7   this.root = null;
8 }
9
10 Tree.prototype.add = function(data, toNodeData) {
11   var node = new Node(data);
12   var parent = toNodeData ? this.findBFS(toNodeData) : null;
13   if(parent) {
14     parent.children.push(node);
15   } else {
16     if(!this.root) {
17       this.root = node;
18     } else {
19       return 'Root node is already assigned';
20     }
21   }
22 };
23 Tree.prototype.remove = function(data) {
24   if(this.root.data === data) {
25     this.root = null;
26   }
27
28   var queue = [this.root];
29   while(queue.length) {
30     var node = queue.shift();
31     for(var i = 0; i < node.children.length; i++) {
32       if(node.children[i].data === data) {
33         node.children.splice(i, 1);
34       } else {
35         queue.push(node.children[i]);
36       }
37     }
38   }
39 };
40 Tree.prototype.contains = function(data) {
41   return this.findBFS(data) ? true : false;
42 };
43 Tree.prototype.findBFS = function(data) {
44   var queue = [this.root];
45   while(queue.length) {
46     var node = queue.shift();
47     if(node.data === data) {
48       return node;
49     }
50     for(var i = 0; i < node.children.length; i++) {
51       queue.push(node.children[i]);
52     }
53   }
54   return null;
55 };
56 Tree.prototype._preOrder = function(node, fn) {
57   if(node) {
58     if(fn) {
59       fn(node);
60     }

```

```
61     for(var i = 0; i < node.children.length; i++) {
62         this._preOrder(node.children[i], fn);
63     }
64 }
65 };
66 Tree.prototype._postOrder = function(node, fn) {
67     if(node) {
68         for(var i = 0; i < node.children.length; i++) {
69             this._postOrder(node.children[i], fn);
70         }
71         if(fn) {
72             fn(node);
73         }
74     }
75 };
76 Tree.prototype.traverseDFS = function(fn, method) {
77     var current = this.root;
78     if(method) {
79         this['_' + method](current, fn);
80     } else {
81         this._preOrder(current, fn);
82     }
83 };
84 Tree.prototype.traverseBFS = function(fn) {
85     var queue = [this.root];
86     while(queue.length) {
87         var node = queue.shift();
88         if(fn) {
89             fn(node);
90         }
91         for(var i = 0; i < node.children.length; i++) {
92             queue.push(node.children[i]);
93         }
94     }
95 };
96 Tree.prototype.print = function() {
97     if(!this.root) {
98         return console.log('No root node found');
99     }
100     var newline = new Node('|');
101     var queue = [this.root, newline];
102     var string = '';
103     while(queue.length) {
104         var node = queue.shift();
105         string += node.data.toString() + ' ';
106         if(node === newline && queue.length) {
107             queue.push(newline);
108         }
109         for(var i = 0; i < node.children.length; i++) {
110             queue.push(node.children[i]);
111         }
112     }
113     console.log(string.slice(0, -2).trim());
114 };
115 Tree.prototype.printByLevel = function() {
116     if(!this.root) {
117         return console.log('No root node found');
118     }
119     var newline = new Node('\n');
120     var queue = [this.root, newline];
```

```
121 var string = '';
122 while(queue.length) {
123     var node = queue.shift();
124     string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
125     if(node === newline && queue.length) {
126         queue.push(newline);
127     }
128     for(var i = 0; i < node.children.length; i++) {
129         queue.push(node.children[i]);
130     }
131 }
132 console.log(string.trim());
133 };
134
135 var tree = new Tree();
136 tree.add('ceo');
137 tree.add('cto', 'ceo');
138 tree.add('dev1', 'cto');
139 tree.add('dev2', 'cto');
140 tree.add('dev3', 'cto');
141 tree.add('cfo', 'ceo');
142 tree.add('accountant', 'cfo');
143 tree.add('cmo', 'ceo');
144 tree.print(); // => ceo | cto cfo cmo | dev1 dev2 dev3 accountant
145 tree.printByLevel(); // => ceo \n cto cfo cmo \n dev1 dev2 dev3 accountant
146 console.log('tree contains dev1 is true:', tree.contains('dev1')); // => true
147 console.log('tree contains dev4 is false:', tree.contains('dev4')); // => false
148 console.log('--- BFS');
149 tree.traverseBFS(function(node) { console.log(node.data); }); // => ceo cto cfo cmo
    dev1 dev2 dev3 accountant
150 console.log('--- DFS preOrder');
151 tree.traverseDFS(function(node) { console.log(node.data); }, 'preOrder'); // => ceo
    cto dev1 dev2 dev3 cfo accountant cmo
152 console.log('--- DFS postOrder');
153 tree.traverseDFS(function(node) { console.log(node.data); }, 'postOrder'); // => dev1
    dev2 dev3 cto accountant cfo cmo ceo
154 tree.remove('cmo');
155 tree.print(); // => ceo | cto cfo | dev1 dev2 dev3 accountant
156 tree.remove('cfo');
157 tree.print(); // => ceo | cto | dev1 dev2 dev3
158
```