

```
1 function Node(data) {
2   this.data = data;
3   this.isWord = false;
4   this.prefixes = 0;
5   this.children = {};
6 }
7
8 function Trie() {
9   this.root = new Node('');
10 }
11
12 Trie.prototype.add = function(word) {
13   if(!this.root) {
14     return null;
15   }
16   this._addNode(this.root, word);
17 };
18 Trie.prototype._addNode = function(node, word) {
19   if(!node || !word) {
20     return null;
21   }
22   node.prefixes++;
23   var letter = word.charAt(0);
24   var child = node.children[letter];
25   if(!child) {
26     child = new Node(letter);
27     node.children[letter] = child;
28   }
29   var remainder = word.substring(1);
30   if(!remainder) {
31     child.isWord = true;
32   }
33   this._addNode(child, remainder);
34 };
35 Trie.prototype.remove = function(word) {
36   if(!this.root) {
37     return;
38   }
39   if(this.contains(word)) {
40     this._removeNode(this.root, word);
41   }
42 };
43 Trie.prototype._removeNode = function(node, word) {
44   if(!node || !word) {
45     return;
46   }
47   node.prefixes--;
48   var letter = word.charAt(0);
49
50   var child = node.children[letter];
51   if(child) {
52     var remainder = word.substring(1);
53     if(remainder) {
54       if(child.prefixes === 1) {
55         delete node.children[letter];
56       } else {
57         this._removeNode(child, remainder);
58       }
59     } else {
60       if(child.prefixes === 0) {
```

```
61         delete node.children[letter];
62     } else {
63         child.isWord = false;
64     }
65 }
66 }
67 };
68 Trie.prototype.contains = function(word) {
69     if(!this.root) {
70         return false;
71     }
72     return this._contains(this.root, word);
73 };
74 Trie.prototype._contains = function(node, word) {
75     if(!node || !word) {
76         return false;
77     }
78     var letter = word.charAt(0);
79     var child = node.children[letter];
80     if(child) {
81         var remainder = word.substring(1);
82         if(!remainder && child.isWord) {
83             return true;
84         } else {
85             return this._contains(child, remainder);
86         }
87     } else {
88         return false;
89     }
90 };
91 Trie.prototype.countWords = function() {
92     if(!this.root) {
93         return console.log('No root node found');
94     }
95     var queue = [this.root];
96     var counter = 0;
97     while(queue.length) {
98         var node = queue.shift();
99         if(node.isWord) {
100             counter++;
101         }
102         for(var child in node.children) {
103             if(node.children.hasOwnProperty(child)) {
104                 queue.push(node.children[child]);
105             }
106         }
107     }
108     return counter;
109 };
110 Trie.prototype.getWords = function() {
111     var words = [];
112     var word = '';
113     this._getWords(this.root, words, word);
114     return words;
115 };
116 Trie.prototype._getWords = function(node, words, word) {
117     for(var child in node.children) {
118         if(node.children.hasOwnProperty(child)) {
119             word += child;
120             if (node.children[child].isWord) {
```

```

121     words.push(word);
122 }
123 this._getWords(node.children[child], words, word);
124 word = word.substring(0, word.length - 1);
125 }
126 }
127 };
128 Trie.prototype.print = function() {
129     if(!this.root) {
130         return console.log('No root node found');
131     }
132     var newline = new Node('|');
133     var queue = [this.root, newline];
134     var string = '';
135     while(queue.length) {
136         var node = queue.shift();
137         string += node.data.toString() + ' ';
138         if(node === newline && queue.length) {
139             queue.push(newline);
140         }
141         for(var child in node.children) {
142             if(node.children.hasOwnProperty(child)) {
143                 queue.push(node.children[child]);
144             }
145         }
146     }
147     console.log(string.slice(0, -2).trim());
148 };
149 Trie.prototype.printByLevel = function() {
150     if(!this.root) {
151         return console.log('No root node found');
152     }
153     var newline = new Node('\n');
154     var queue = [this.root, newline];
155     var string = '';
156     while(queue.length) {
157         var node = queue.shift();
158         string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
159         if(node === newline && queue.length) {
160             queue.push(newline);
161         }
162         for(var child in node.children) {
163             if(node.children.hasOwnProperty(child)) {
164                 queue.push(node.children[child]);
165             }
166         }
167     }
168     console.log(string.trim());
169 };
170
171 var trie = new Trie();
172 trie.add('one');
173 trie.add('two');
174 trie.add('fifth');
175 trie.add('fifty');
176 trie.print(); // => | o t f | n w i | e o f | t | h y
177 trie.printByLevel(); // => o t f \n n w i \n e o f \n t \n h y
178 console.log('words are: one, two, fifth, fifty:', trie.getWords()); // => [ 'one',
    'two', 'fifth', 'fifty' ]
179 console.log('trie count words is 4:', trie.countWords()); // => 4

```

```
180 console.log('trie contains one is true:', trie.contains('one')); // => true
181 console.log('trie contains on is false:', trie.contains('on')); // => false
182 trie.remove('one');
183 console.log('trie contains one is false:', trie.contains('one')); // => false
184 console.log('trie count words is 3:', trie.countWords()); // => 3
185 console.log('words are two, fifth, fifty:', trie.getWords()); // => [ 'two', 'fifth',
    'fifty' ]
186
```