

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

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

```
121     this._getWords(this.root, words, word);
122     return words;
123 }
124
125 _getWords(node, words, word) {
126     for(const child in node.children) {
127         if(node.children.hasOwnProperty(child)) {
128             word += child;
129             if (node.children[child].isWord) {
130                 words.push(word);
131             }
132             this._getWords(node.children[child], words, word);
133             word = word.substring(0, word.length - 1);
134         }
135     }
136 }
137
138 print() {
139     if(!this.root) {
140         return console.log('No root node found');
141     }
142     const newline = new Node('|');
143     const queue = [this.root, newline];
144     let string = '';
145     while(queue.length) {
146         const node = queue.shift();
147         string += `${node.data.toString()} `;
148         if(node === newline && queue.length) {
149             queue.push(newline);
150         }
151         for(const child in node.children) {
152             if(node.children.hasOwnProperty(child)) {
153                 queue.push(node.children[child]);
154             }
155         }
156     }
157     console.log(string.slice(0, -2).trim());
158 }
159
160 printByLevel() {
161     if(!this.root) {
162         return console.log('No root node found');
163     }
164     const newline = new Node('\n');
165     const queue = [this.root, newline];
166     let string = '';
167     while(queue.length) {
168         const node = queue.shift();
169         string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
170         if(node === newline && queue.length) {
171             queue.push(newline);
172         }
173         for(const child in node.children) {
174             if(node.children.hasOwnProperty(child)) {
175                 queue.push(node.children[child]);
176             }
177         }
178     }
179     console.log(string.trim());
180 }
```

```
181 }
182
183 const trie = new Trie();
184 trie.add('one');
185 trie.add('two');
186 trie.add('fifth');
187 trie.add('fifty');
188 trie.print(); // => | o t f | n w i | e o f | t | h y
189 trie.printByLevel(); // => o t f \n n w i \n e o f \n t \n h y
190 console.log('words are: one, two, fifth, fifty:', trie.getWords()); // => [ 'one',
    'two', 'fifth', 'fifty' ]
191 console.log('trie count words is 4:', trie.countWords()); // => 4
192 console.log('trie contains one is true:', trie.contains('one')); // => true
193 console.log('trie contains on is false:', trie.contains('on')); // => false
194 trie.remove('one');
195 console.log('trie contains one is false:', trie.contains('one')); // => false
196 console.log('trie count words is 3:', trie.countWords()); // => 3
197 console.log('words are two, fifth, fifty:', trie.getWords()); // => [ 'two', 'fifth',
    'fifty' ]
198
```