```
function Node(data) {
  this.data = data;
  this.isWord = false;
  this.prefixes = 0;
  this.children = {};
}

class Trie {
  constructor() {
    this.root = new Node('');
  }

  add(word) {
    if(!this.root) {
      return null;
    }
    this._addNode(this.root, word);
  }

  _addNode(node, word) {
    if(!node || !word) {
      return null;
    }
    node.prefixes++;
    const letter = word.charAt(0);
    let child = node.children[letter];
    if(!child) {
      child = new Node(letter);
      node.children[letter] = child;
    }
    const remainder = word.substring(1);
    if(!remainder) {
      child.isWord = true;
    }
    this._addNode(child, remainder);
  }

  remove(word) {
    if(!this.root) {
      return;
    }
    if(this.contains(word)) {
      this._removeNode(this.root, word);
    }
  }

  _removeNode(node, word) {
    if(!node || !word) {
      return;
    }
    node.prefixes--;
    const letter = word.charAt(0);

    const child = node.children[letter];
    if(child) {
      const remainder = word.substring(1);
      if(remainder) {
        if(child.prefixes === 1) {
          delete node.children[letter];
        } 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 = '';
```

```
    this._getWords(this.root, words, word);
    return words;
  }

  _getWords(node, words, word) {
    for(const child in node.children) {
      if(node.children.hasOwnProperty(child)) {
        word += child;
        if (node.children[child].isWord) {
          words.push(word);
        }
        this._getWords(node.children[child], words, word);
        word = word.substring(0, word.length - 1);
      }
    }
  }

  print() {
    if(!this.root) {
      return console.log('No root node found');
    }
    const newline = new Node('|');
    const queue = [this.root, newline];
    let string = '';
    while(queue.length) {
      const node = queue.shift();
      string += `${node.data.toString()} `;
      if(node === newline && queue.length) {
        queue.push(newline);
      }
      for(const child in node.children) {
        if(node.children.hasOwnProperty(child)) {
          queue.push(node.children[child]);
        }
      }
    }
    console.log(string.slice(0, -2).trim());
  }

  printByLevel() {
    if(!this.root) {
      return console.log('No root node found');
    }
    const newline = new Node('\n');
    const queue = [this.root, newline];
    let string = '';
    while(queue.length) {
      const node = queue.shift();
      string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
      if(node === newline && queue.length) {
        queue.push(newline);
      }
      for(const child in node.children) {
        if(node.children.hasOwnProperty(child)) {
          queue.push(node.children[child]);
        }
      }
    }
    console.log(string.trim());
  }
```

```
}

const trie = new Trie();
trie.add('one');
trie.add('two');
trie.add('fifth');
trie.add('fifty');
trie.print(); // => | o t f | n w i | e o f | t | h y
trie.printByLevel(); // => o t f \n n w i \n e o f \n t \n h y
console.log('words are: one, two, fifth, fifty:', trie.getWords()); // => [ 'one', 'two', 'fifth', 'fifty' ]
console.log('trie count words is 4:', trie.countWords()); // => 4
console.log('trie contains one is true:', trie.contains('one')); // => true
console.log('trie contains on is false:', trie.contains('on')); // => false
trie.remove('one');
console.log('trie contains one is false:', trie.contains('one')); // => false
console.log('trie count words is 3:', trie.countWords()); // => 3
console.log('words are two, fifth, fifty:', trie.getWords()); // => [ 'two', 'fifth', 'fifty' ]

```