```
function MyArray() {
  this.array = [];
}

MyArray.prototype.add = function(data) {
  this.array.push(data);
};
MyArray.prototype.remove = function(data) {
  this.array = this.array.filter(function(current) {
    return current !== data;
  });
};
MyArray.prototype.search = function(data) {
  var foundIndex = this.array.indexOf(data);
  if(~foundIndex) {
    return foundIndex;
  }

  return null;
};
MyArray.prototype.getAtIndex = function(index) {
  return this.array[index];
};
MyArray.prototype.length = function() {
  return this.array.length;
};
MyArray.prototype.print = function() {
  console.log(this.array.join(' '));
};

var array = new MyArray();
array.add(1);
array.add(2);
array.add(3);
array.add(4);
array.print(); // => 1 2 3 4
console.log('search 3 gives index 2:', array.search(3)); // => 2
console.log('getAtIndex 2 gives 3:', array.getAtIndex(2)); // => 3
console.log('length is 4:', array.length()); // => 4
array.remove(3);
array.print(); // => 1 2 4
array.add(5);
array.add(5);
array.print(); // => 1 2 4 5 5
array.remove(5);
array.print(); // => 1 2 4

```

```
function Node(data) {
  this.data = data;
  this.left = null;
  this.right = null;
}

function BinarySearchTree() {
  this.root = null;
}

BinarySearchTree.prototype.add = function(data) {
  var node = new Node(data);
  if(!this.root) {
    this.root = node;
  } else {
    var current = this.root;
    while(current) {
      if(node.data < current.data) {
        if(!current.left) {
          current.left = node;
          break;
        }
        current = current.left;
      } else if (node.data > current.data) {
        if(!current.right) {
          current.right = node;
          break;
        }
        current = current.right;
      } else {
        break;
      }
    }
  }
};
BinarySearchTree.prototype.remove = function(data) {
  var that = this;
  var removeNode = function(node, data) {
    if(!node) {
      return null;
    }
    if(data === node.data) {
      if(!node.left && !node.right) {
        return null;
      }
      if(!node.left) {
        return node.right;
      }
      if(!node.right) {
        return node.left;
      }
      // 2 children
      var temp = that.getMin(node.right);
      node.data = temp;
      node.right = removeNode(node.right, temp);
      return node;
    } else if(data < node.data) {
      node.left = removeNode(node.left, data);
      return node;
    } else {
```

```
      node.right = removeNode(node.right, data);
      return node;
    }
  };
  this.root = removeNode(this.root, data);
};
BinarySearchTree.prototype.contains = function(data) {
  var current = this.root;
  while(current) {
    if(data === current.data) {
      return true;
    }
    if(data < current.data) {
      current = current.left;
    } else {
      current = current.right;
    }
  }
  return false;
};
BinarySearchTree.prototype._preOrder = function(node, fn) {
  if(node) {
    if(fn) {
      fn(node);
    }
    this._preOrder(node.left, fn);
    this._preOrder(node.right, fn);
  }
};
BinarySearchTree.prototype._inOrder = function(node, fn) {
  if(node) {
    this._inOrder(node.left, fn);
    if(fn) {
      fn(node);
    }
    this._inOrder(node.right, fn);
  }
};
BinarySearchTree.prototype._postOrder = function(node, fn) {
  if(node) {
    this._postOrder(node.left, fn);
    this._postOrder(node.right, fn);
    if(fn) {
      fn(node);
    }
  }
};
BinarySearchTree.prototype.traverseDFS = function(fn, method) {
  var current = this.root;
  if(method) {
    this['_' + method](current, fn);
  } else {
    this._preOrder(current, fn);
  }
};
BinarySearchTree.prototype.traverseBFS = function(fn) {
  this.queue = [];
  this.queue.push(this.root);
  while(this.queue.length) {
    var node = this.queue.shift();
```

```
    if(fn) {
      fn(node);
    }
    if(node.left) {
      this.queue.push(node.left);
    }
    if(node.right) {
      this.queue.push(node.right);
    }
  }
};
BinarySearchTree.prototype.print = function() {
  if(!this.root) {
    return console.log('No root node found');
  }
  var newline = new Node('|');
  var queue = [this.root, newline];
  var string = '';
  while(queue.length) {
    var node = queue.shift();
    string += node.data.toString() + ' ';
    if(node === newline && queue.length) {
      queue.push(newline);
    }
    if(node.left) {
      queue.push(node.left);
    }
    if(node.right) {
      queue.push(node.right);
    }
  }
  console.log(string.slice(0, -2).trim());
};
BinarySearchTree.prototype.printByLevel = function() {
  if(!this.root) {
    return console.log('No root node found');
  }
  var newline = new Node('\n');
  var queue = [this.root, newline];
  var string = '';
  while(queue.length) {
    var node = queue.shift();
    string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
    if(node === newline && queue.length) {
      queue.push(newline);
    }
    if(node.left) {
      queue.push(node.left);
    }
    if(node.right) {
      queue.push(node.right);
    }
  }
  console.log(string.trim());
};
BinarySearchTree.prototype.getMin = function(node) {
  if(!node) {
    node = this.root;
  }
  while(node.left) {
```

```
    node = node.left;
  }
  return node.data;
};
BinarySearchTree.prototype.getMax = function(node) {
  if(!node) {
    node = this.root;
  }
  while(node.right) {
    node = node.right;
  }
  return node.data;
};
BinarySearchTree.prototype._getHeight = function(node) {
  if(!node) {
    return -1;
  }
  var left = this._getHeight(node.left);
  var right = this._getHeight(node.right);
  return Math.max(left, right) + 1;
};
BinarySearchTree.prototype.getHeight = function(node) {
  if(!node) {
    node = this.root;
  }
  return this._getHeight(node);
};
BinarySearchTree.prototype._isBalanced = function(node) {
  if(!node) {
    return true;
  }
  var heigthLeft = this._getHeight(node.left);
  var heigthRight = this._getHeight(node.right);
  var diff = Math.abs(heigthLeft - heigthRight);
  if(diff > 1) {
    return false;
  } else {
    return this._isBalanced(node.left) && this._isBalanced(node.right);
  }
};
BinarySearchTree.prototype.isBalanced = function(node) {
  if(!node) {
    node = this.root;
  }
  return this._isBalanced(node);
};
BinarySearchTree.prototype._checkHeight = function(node) {
  if(!node) {
    return 0;
  }
  var left = this._checkHeight(node.left);
  if(left === -1) {
    return -1;
  }
  var right = this._checkHeight(node.right);
  if(right === -1) {
    return -1;
  }
  var diff = Math.abs(left - right);
  if(diff > 1) {
```

```
    return -1;
  } else {
    return Math.max(left, right) + 1;
  }
};
BinarySearchTree.prototype.isBalancedOptimized = function(node) {
  if(!node) {
    node = this.root;
  }
  if(!node) {
    return true;
  }
  if(this._checkHeight(node) === -1) {
    return false;
  } else {
    return true;
  }
};

var binarySearchTree = new BinarySearchTree();
binarySearchTree.add(5);
binarySearchTree.add(3);
binarySearchTree.add(7);
binarySearchTree.add(2);
binarySearchTree.add(4);
binarySearchTree.add(4);
binarySearchTree.add(6);
binarySearchTree.add(8);
binarySearchTree.print(); // => 5 | 3 7 | 2 4 6 8
binarySearchTree.printByLevel(); // => 5 \n 3 7 \n 2 4 6 8
console.log('--- DFS inOrder');
binarySearchTree.traverseDFS(function(node) { console.log(node.data); }, 'inOrder');
// => 2 3 4 5 6 7 8
console.log('--- DFS preOrder');
binarySearchTree.traverseDFS(function(node) { console.log(node.data); }, 'preOrder');
// => 5 3 2 4 7 6 8
console.log('--- DFS postOrder');
binarySearchTree.traverseDFS(function(node) { console.log(node.data); },
'postOrder'); // => 2 4 3 6 8 7 5
console.log('--- BFS');
binarySearchTree.traverseBFS(function(node) { console.log(node.data); }); // => 5 3 7
2 4 6 8
console.log('min is 2:', binarySearchTree.getMin()); // => 2
console.log('max is 8:', binarySearchTree.getMax()); // => 8
console.log('tree contains 3 is true:', binarySearchTree.contains(3)); // => true
console.log('tree contains 9 is false:', binarySearchTree.contains(9)); // => false
console.log('tree height is 2:', binarySearchTree.getHeight()); // => 2
console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
binarySearchTree.remove(11); // remove non existing node
binarySearchTree.print(); // => 5 | 3 7 | 2 4 6 8
binarySearchTree.remove(5); // remove 5, 6 goes up
binarySearchTree.print(); // => 6 | 3 7 | 2 4 8
binarySearchTree.remove(7); // remove 7, 8 goes up
binarySearchTree.print(); // => 6 | 3 8 | 2 4
binarySearchTree.remove(8); // remove 8, the tree becomes unbalanced
binarySearchTree.print(); // => 6 | 3 | 2 4
console.log('tree is balanced is false:', binarySearchTree.isBalanced()); // => true
binarySearchTree.remove(4);
binarySearchTree.remove(2);
binarySearchTree.remove(3);
```

```
297 binarySearchTree.remove(6);
298 binarySearchTree.print(); // => 'No root node found'
299 binarySearchTree.printByLevel(); // => 'No root node found'
300 console.log('tree height is -1:', binarySearchTree.getHeight()); // => -1
301 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
302 console.log('---');
303 binarySearchTree.add(10);
304 console.log('tree height is 0:', binarySearchTree.getHeight()); // => 0
305 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
306 binarySearchTree.add(6);
307 binarySearchTree.add(14);
308 binarySearchTree.add(4);
309 binarySearchTree.add(8);
310 binarySearchTree.add(12);
311 binarySearchTree.add(16);
312 binarySearchTree.add(3);
313 binarySearchTree.add(5);
314 binarySearchTree.add(7);
315 binarySearchTree.add(9);
316 binarySearchTree.add(11);
317 binarySearchTree.add(13);
318 binarySearchTree.add(15);
319 binarySearchTree.add(17);
320 binarySearchTree.print(); // => 10 | 6 14 | 4 8 12 16 | 3 5 7 9 11 13 15 17
321 binarySearchTree.remove(10); // remove 10, 11 goes up
322 binarySearchTree.print(); // => 11 | 6 14 | 4 8 12 16 | 3 5 7 9 x 13 15 17
323 binarySearchTree.remove(12); // remove 12; 13 goes up
324 binarySearchTree.print(); // => 11 | 6 14 | 4 8 13 16 | 3 5 7 9 x x 15 17
325 console.log('tree is balanced is true:', binarySearchTree.isBalanced()); // => true
326 console.log('tree is balanced optimized is true:',
    binarySearchTree.isBalancedOptimized()); // => true
327 binarySearchTree.remove(13); // remove 13, 13 has no children so nothing changes
328 binarySearchTree.print(); // => 11 | 6 14 | 4 8 x 16 | 3 5 7 9 x x 15 17
329 console.log('tree is balanced is false:', binarySearchTree.isBalanced()); // => false
330 console.log('tree is balanced optimized is false:',
    binarySearchTree.isBalancedOptimized()); // => false
331
```

```
function Node(data) {
  this.data = data;
  this.previous = null;
  this.next = null;
}

function DoublyLinkedList() {
  this.head = null;
  this.tail = null;
  this.numberOfValues = 0;
}

DoublyLinkedList.prototype.add = function (data) {
  var node = new Node(data);
  if(!this.head) {
    this.head = node;
    this.tail = node;
  } else {
    node.previous = this.tail;
    this.tail.next = node;
    this.tail = node;
  }
  this.numberOfValues++;
};
DoublyLinkedList.prototype.remove = function(data) {
  var current = this.head;
  while(current) {
    if(current.data === data) {
      if(current === this.head && current === this.tail) {
        this.head = null;
        this.tail = null;
      } else if(current === this.head) {
        this.head = this.head.next;
        this.head.previous = null;
      } else if(current === this.tail) {
        this.tail = this.tail.previous;
        this.tail.next = null;
      } else {
        current.previous.next = current.next;
        current.next.previous = current.previous;
      }
      this.numberOfValues--;
    }
    current = current.next;
  }
};
DoublyLinkedList.prototype.insertAfter = function(data, toNodeData) {
  var current = this.head;
  while(current) {
    if(current.data === toNodeData) {
      var node = new Node(data);
      if(current === this.tail) {
        this.add(data);
      } else {
        current.next.previous = node;
        node.previous = current;
        node.next = current.next;
        current.next = node;
        this.numberOfValues++;
      }
```

```
    }
    current = current.next;
  }
};
DoublyLinkedList.prototype.traverse = function(fn) {
  var current = this.head;
  while(current) {
    if(fn) {
      fn(current);
    }
    current = current.next;
  }
};
DoublyLinkedList.prototype.traverseReverse = function(fn) {
  var current = this.tail;
  while(current) {
    if(fn) {
      fn(current);
    }
    current = current.previous;
  }
};
DoublyLinkedList.prototype.length = function() {
  return this.numberOfValues;
};
DoublyLinkedList.prototype.print = function() {
  var string = '';
  var current = this.head;
  while(current) {
    string += current.data + ' ';
    current = current.next;
  }
  console.log(string.trim());
};

var doublyLinkedList = new DoublyLinkedList();
doublyLinkedList.print(); // => ''
doublyLinkedList.add(1);
doublyLinkedList.add(2);
doublyLinkedList.add(3);
doublyLinkedList.add(4);
doublyLinkedList.print(); // => 1 2 3 4
console.log('length is 4:', doublyLinkedList.length()); // => 4
doublyLinkedList.remove(3); // remove value
doublyLinkedList.print(); // => 1 2 4
doublyLinkedList.remove(9); // remove non existing value
doublyLinkedList.print(); // => 1 2 4
doublyLinkedList.remove(1); // remove head
doublyLinkedList.print(); // => 2 4
doublyLinkedList.remove(4); // remove tail
doublyLinkedList.print(); // => 2
console.log('length is 1:', doublyLinkedList.length()); // => 1
doublyLinkedList.remove(2); // remove tail, the list should be empty
doublyLinkedList.print(); // => ''
console.log('length is 0:', doublyLinkedList.length()); // => 0
doublyLinkedList.add(2);
doublyLinkedList.add(6);
doublyLinkedList.print(); // => 2 6
doublyLinkedList.insertAfter(3, 2);
doublyLinkedList.print(); // => 2 3 6
```

```
doublyLinkedList.traverseReverse(function(node) { console.log(node.data); });
doublyLinkedList.insertAfter(4, 3);
doublyLinkedList.print(); // => 2 3 4 6
doublyLinkedList.insertAfter(5, 9); // insertAfter a non existing node
doublyLinkedList.print(); // => 2 3 4 6
doublyLinkedList.insertAfter(5, 4);
doublyLinkedList.insertAfter(7, 6); // insertAfter the tail
doublyLinkedList.print(); // => 2 3 4 5 6 7
doublyLinkedList.add(8); // add node with normal method
doublyLinkedList.print(); // => 2 3 4 5 6 7 8
console.log('length is 7:', doublyLinkedList.length()); // => 7
doublyLinkedList.traverse(function(node) { node.data = node.data + 10; });
doublyLinkedList.print(); // => 12 13 14 15 16 17 18
doublyLinkedList.traverse(function(node) { console.log(node.data); }); // => 12 13 14
15 16 17 18
console.log('length is 7:', doublyLinkedList.length()); // => 7
doublyLinkedList.traverseReverse(function(node) { console.log(node.data); }); // =>
18 17 16 15 14 13 12
doublyLinkedList.print(); // => 12 13 14 15 16 17 18
console.log('length is 7:', doublyLinkedList.length()); // => 7

```

```
function Graph() {
  this.vertices = [];
  this.edges = [];
  this.numberOfEdges = 0;
}

Graph.prototype.addVertex = function(vertex) {
  this.vertices.push(vertex);
  this.edges[vertex] = [];
};
Graph.prototype.removeVertex = function(vertex) {
  var index = this.vertices.indexOf(vertex);
  if(~index) {
    this.vertices.splice(index, 1);
  }
  while(this.edges[vertex].length) {
    var adjacentVertex = this.edges[vertex].pop();
    this.removeEdge(adjacentVertex, vertex);
  }
};
Graph.prototype.addEdge = function(vertex1, vertex2) {
  this.edges[vertex1].push(vertex2);
  this.edges[vertex2].push(vertex1);
  this.numberOfEdges++;
};
Graph.prototype.removeEdge = function(vertex1, vertex2) {
  var index1 = this.edges[vertex1] ? this.edges[vertex1].indexOf(vertex2) : -1;
  var index2 = this.edges[vertex2] ? this.edges[vertex2].indexOf(vertex1) : -1;
  if(~index1) {
    this.edges[vertex1].splice(index1, 1);
    this.numberOfEdges--;
  }
  if(~index2) {
    this.edges[vertex2].splice(index2, 1);
  }
};
Graph.prototype.size = function() {
  return this.vertices.length;
};
Graph.prototype.relations = function() {
  return this.numberOfEdges;
};
Graph.prototype.traverseDFS = function(vertex, fn) {
  if(!~this.vertices.indexOf(vertex)) {
    return console.log('Vertex not found');
  }
  var visited = [];
  this._traverseDFS(vertex, visited, fn);
};
Graph.prototype._traverseDFS = function(vertex, visited, fn) {
  visited[vertex] = true;
  if(this.edges[vertex] !== undefined) {
    fn(vertex);
  }
  for(var i = 0; i < this.edges[vertex].length; i++) {
    if(!visited[this.edges[vertex][i]]) {
      this._traverseDFS(this.edges[vertex][i], visited, fn);
    }
  }
};
```

```
61 Graph.prototype.traverseBFS = function(vertex, fn) {
62   if(!~this.vertices.indexOf(vertex)) {
63     return console.log('Vertex not found');
64   }
65   var queue = [];
66   queue.push(vertex);
67   var visited = [];
68   visited[vertex] = true;
69
70   while(queue.length) {
71     vertex = queue.shift();
72     fn(vertex);
73     for(var i = 0; i < this.edges[vertex].length; i++) {
74       if(!visited[this.edges[vertex][i]]) {
75         visited[this.edges[vertex][i]] = true;
76         queue.push(this.edges[vertex][i]);
77       }
78     }
79   }
80 };
81 Graph.prototype.pathFromTo = function(vertexSource, vertexDestination) {
82   if(!~this.vertices.indexOf(vertexSource)) {
83     return console.log('Vertex not found');
84   }
85   var queue = [];
86   queue.push(vertexSource);
87   var visited = [];
88   visited[vertexSource] = true;
89   var paths = [];
90
91   while(queue.length) {
92     var vertex = queue.shift();
93     for(var i = 0; i < this.edges[vertex].length; i++) {
94       if(!visited[this.edges[vertex][i]]) {
95         visited[this.edges[vertex][i]] = true;
96         queue.push(this.edges[vertex][i]);
97         // save paths between vertices
98         paths[this.edges[vertex][i]] = vertex;
99       }
100    }
101  }
102  if(!visited[vertexDestination]) {
103    return undefined;
104  }
105
106  var path = [];
107  for(var j = vertexDestination; j != vertexSource; j = paths[j]) {
108    path.push(j);
109  }
110  path.push(j);
111  return path.reverse().join('-');
112 };
113 Graph.prototype.print = function() {
114   console.log(this.vertices.map(function(vertex) {
115     return (vertex + ' -> ' + this.edges[vertex].join(', ')).trim();
116   }, this).join(' | '));
117 };
118
119 var graph = new Graph();
120 graph.addVertex(1);
```

```
graph.addVertex(2);
graph.addVertex(3);
graph.addVertex(4);
graph.addVertex(5);
graph.addVertex(6);
graph.print(); // 1 -> | 2 -> | 3 -> | 4 -> | 5 -> | 6 ->
graph.addEdge(1, 2);
graph.addEdge(1, 5);
graph.addEdge(2, 3);
graph.addEdge(2, 5);
graph.addEdge(3, 4);
graph.addEdge(4, 5);
graph.addEdge(4, 6);
graph.print(); // 1 -> 2, 5 | 2 -> 1, 3, 5 | 3 -> 2, 4 | 4 -> 3, 5, 6 | 5 -> 1, 2, 4 | 6 -> 4
console.log('graph size (number of vertices):', graph.size()); // => 6
console.log('graph relations (number of edges):', graph.relations()); // => 7
graph.traverseDFS(1, function(vertex) { console.log(vertex); }); // => 1 2 3 4 5 6
console.log('---');
graph.traverseBFS(1, function(vertex) { console.log(vertex); }); // => 1 2 5 3 4 6
graph.traverseDFS(0, function(vertex) { console.log(vertex); }); // => 'Vertex not found'
graph.traverseBFS(0, function(vertex) { console.log(vertex); }); // => 'Vertex not found'
console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-5-1
console.log('path from 3 to 5:', graph.pathFromTo(3, 5)); // => 3-2-5
graph.removeEdge(1, 2);
graph.removeEdge(4, 5);
graph.removeEdge(10, 11);
console.log('graph relations (number of edges):', graph.relations()); // => 5
console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-3-2-5-1
graph.addEdge(1, 2);
graph.addEdge(4, 5);
console.log('graph relations (number of edges):', graph.relations()); // => 7
console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-5-1
graph.removeVertex(5);
console.log('graph size (number of vertices):', graph.size()); // => 5
console.log('graph relations (number of edges):', graph.relations()); // => 4
console.log('path from 6 to 1:', graph.pathFromTo(6, 1)); // => 6-4-3-2-1

```

```
function HashTable(size) {
  this.values = {};
  this.numberOfValues = 0;
  this.size = size;
}

HashTable.prototype.add = function(key, value) {
  var hash = this.calculateHash(key);
  if(!this.values.hasOwnProperty(hash)) {
    this.values[hash] = {};
  }
  if(!this.values[hash].hasOwnProperty(key)) {
    this.numberOfValues++;
  }
  this.values[hash][key] = value;
};
HashTable.prototype.remove = function(key) {
  var hash = this.calculateHash(key);
  if(this.values.hasOwnProperty(hash) && this.values[hash].hasOwnProperty(key)) {
    delete this.values[hash][key];
    this.numberOfValues--;
  }
};
HashTable.prototype.calculateHash = function(key) {
  return key.toString().length % this.size;
};
HashTable.prototype.search = function(key) {
  var hash = this.calculateHash(key);
  if(this.values.hasOwnProperty(hash) && this.values[hash].hasOwnProperty(key)) {
    return this.values[hash][key];
  } else {
    return null;
  }
};
HashTable.prototype.length = function() {
  return this.numberOfValues;
};
HashTable.prototype.print = function() {
  var string = '';
  for(var value in this.values) {
    for(var key in this.values[value]) {
      string += this.values[value][key] + ' ';
    }
  }
  console.log(string.trim());
};

var hashTable = new HashTable(3);
hashTable.add('first', 1);
hashTable.add('second', 2);
hashTable.add('third', 3);
hashTable.add('fourth', 4);
hashTable.add('fifth', 5);
hashTable.print(); // => 2 4 1 3 5
console.log('length gives 5:', hashTable.length()); // => 5
console.log('search second gives 2:', hashTable.search('second')); // => 2
hashTable.remove('fourth');
hashTable.remove('first');
hashTable.print(); // => 2 3 5
console.log('length gives 3:', hashTable.length()); // => 3
```

```
61
```

```
function Queue() {
  this.queue = [];
}

Queue.prototype.enqueue = function(value) {
  this.queue.push(value);
};
Queue.prototype.dequeue = function() {
  return this.queue.shift();
};
Queue.prototype.peek = function() {
  return this.queue[0];
};
Queue.prototype.length = function() {
  return this.queue.length;
};
Queue.prototype.print = function() {
  console.log(this.queue.join(' '));
};

var queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(3);
queue.print(); // => 1 2 3
console.log('length is 3:', queue.length()); // => 3
console.log('peek is 1:', queue.peek()); // => 3
console.log('dequeue is 1:', queue.dequeue()); // => 1
queue.print(); // => 2 3
console.log('dequeue is 2:', queue.dequeue());  // => 2
console.log('length is 1:', queue.length()); // => 1
console.log('dequeue is 3:', queue.dequeue()); // => 3
queue.print(); // => ''
console.log('peek is undefined:', queue.peek()); // => undefined
console.log('dequeue is undefined:', queue.dequeue()); // => undefined

```

```
function Set() {
  this.values = [];
  this.numberOfValues = 0;
}

Set.prototype.add = function(value) {
  if(!~this.values.indexOf(value)) {
    this.values.push(value);
    this.numberOfValues++;
  }
};
Set.prototype.remove = function(value) {
  var index = this.values.indexOf(value);
  if(~index) {
    this.values.splice(index, 1);
    this.numberOfValues--;
  }
};
Set.prototype.contains = function(value) {
  return this.values.indexOf(value) !== -1;
};
Set.prototype.union = function(set) {
  var newSet = new Set();
  set.values.forEach(function(value) {
    newSet.add(value);
  });
  this.values.forEach(function(value) {
    newSet.add(value);
  });
  return newSet;
};
Set.prototype.intersect = function(set) {
  var newSet = new Set();
  this.values.forEach(function(value) {
    if(set.contains(value)) {
      newSet.add(value);
    }
  });
  return newSet;
};
Set.prototype.difference = function(set) {
  var newSet = new Set();
  this.values.forEach(function(value) {
    if(!set.contains(value)) {
      newSet.add(value);
    }
  });
  return newSet;
};
Set.prototype.isSubset = function(set) {
  return set.values.every(function(value) {
    return this.contains(value);
  }, this);
};
Set.prototype.length = function() {
  return this.numberOfValues;
};
Set.prototype.print = function() {
  console.log(this.values.join(' '));
};
```

```
61
62 var set = new Set();
63 set.add(1);
64 set.add(2);
65 set.add(3);
66 set.add(4);
67 set.print(); // => 1 2 3 4
68 set.remove(3);
69 set.print(); // => 1 2 4
70 console.log('contains 4 is true:', set.contains(4)); // => true
71 console.log('contains 3 is false:', set.contains(3)); // => false
72 console.log('---');
73 var set1 = new Set();
74 set1.add(1);
75 set1.add(2);
76 var set2 = new Set();
77 set2.add(2);
78 set2.add(3);
79 var set3 = set2.union(set1);
80 set3.print(); // => 1 2 3
81 var set4 = set2.intersect(set1);
82 set4.print(); // => 2
83 var set5 = set.difference(set3); // 1 2 4 diff 1 2 3
84 set5.print(); // => 4
85 var set6 = set3.difference(set); // 1 2 3 diff 1 2 4
86 set6.print(); // => 3
87 console.log('set1 subset of set is true:', set.isSubset(set1)); // => true
88 console.log('set2 subset of set is false:', set.isSubset(set2)); // => false
89 console.log('set1 length gives 2:', set1.length()); // => 2
90 console.log('set3 length gives 3:', set3.length()); // => 3
91
```

```
function Node(data) {
  this.data = data;
  this.next = null;
}

function SinglyLinkedList() {
  this.head = null;
  this.tail = null;
  this.numberOfValues = 0;
}

SinglyLinkedList.prototype.add = function(data) {
  var node = new Node(data);
  if(!this.head) {
    this.head = node;
    this.tail = node;
  } else {
    this.tail.next = node;
    this.tail = node;
  }
  this.numberOfValues++;
};
SinglyLinkedList.prototype.remove = function(data) {
  var previous = this.head;
  var current = this.head;
  while(current) {
    if(current.data === data) {
      if(current === this.head) {
        this.head = this.head.next;
      }
      if(current === this.tail) {
        this.tail = previous;
      }
      previous.next = current.next;
      this.numberOfValues--;
    } else {
      previous = current;
    }
    current = current.next;
  }
};
SinglyLinkedList.prototype.insertAfter = function(data, toNodeData) {
  var current = this.head;
  while(current) {
    if(current.data === toNodeData) {
      var node = new Node(data);
      if(current === this.tail) {
        this.tail.next = node;
        this.tail = node;
      } else {
        node.next = current.next;
        current.next = node;
      }
      this.numberOfValues++;
    }
    current = current.next;
  }
};
SinglyLinkedList.prototype.traverse = function(fn) {
  var current = this.head;
```

```
  while(current) {
    if(fn) {
      fn(current);
    }
    current = current.next;
  }
};
SinglyLinkedList.prototype.length = function() {
  return this.numberOfValues;
};
SinglyLinkedList.prototype.print = function() {
  var string = '';
  var current = this.head;
  while(current) {
    string += current.data + ' ';
    current = current.next;
  }
  console.log(string.trim());
};

var singlyLinkedList = new SinglyLinkedList();
singlyLinkedList.print(); // => ''
singlyLinkedList.add(1);
singlyLinkedList.add(2);
singlyLinkedList.add(3);
singlyLinkedList.add(4);
singlyLinkedList.print(); // => 1 2 3 4
console.log('length is 4:', singlyLinkedList.length()); // => 4
singlyLinkedList.remove(3); // remove value
singlyLinkedList.print(); // => 1 2 4
singlyLinkedList.remove(9); // remove non existing value
singlyLinkedList.print(); // => 1 2 4
singlyLinkedList.remove(1); // remove head
singlyLinkedList.print(); // => 2 4
singlyLinkedList.remove(4); // remove tail
singlyLinkedList.print(); // => 2
console.log('length is 1:', singlyLinkedList.length()); // => 1
singlyLinkedList.add(6);
singlyLinkedList.print(); // => 2 6
singlyLinkedList.insertAfter(3, 2);
singlyLinkedList.print(); // => 2 3 6
singlyLinkedList.insertAfter(4, 3);
singlyLinkedList.print(); // => 2 3 4 6
singlyLinkedList.insertAfter(5, 9); // insertAfter a non existing node
singlyLinkedList.print(); // => 2 3 4 6
singlyLinkedList.insertAfter(5, 4);
singlyLinkedList.insertAfter(7, 6); // insertAfter the tail
singlyLinkedList.print(); // => 2 3 4 5 6 7
singlyLinkedList.add(8); // add node with normal method
singlyLinkedList.print(); // => 2 3 4 5 6 7 8
console.log('length is 7:', singlyLinkedList.length()); // => 7
singlyLinkedList.traverse(function(node) { node.data = node.data + 10; });
singlyLinkedList.print(); // => 12 13 14 15 16 17 18
singlyLinkedList.traverse(function(node) { console.log(node.data); }); // => 12 13 14
15 16 17 18
console.log('length is 7:', singlyLinkedList.length()); // => 7

```

```
function Stack() {
  this.stack = [];
}

Stack.prototype.push = function(value) {
  this.stack.push(value);
};
Stack.prototype.pop = function() {
  return this.stack.pop();
};
Stack.prototype.peek = function() {
  return this.stack[this.stack.length - 1];
};
Stack.prototype.length = function() {
  return this.stack.length;
};
Stack.prototype.print = function() {
  console.log(this.stack.join(' '));
};

var stack = new Stack();
stack.push(1);
stack.push(2);
stack.push(3);
stack.print(); // => 1 2 3
console.log('length is 3:', stack.length()); // => 3
console.log('peek is 3:', stack.peek()); // => 3
console.log('pop is 3:', stack.pop()); // => 3
stack.print(); // => 1 2
console.log('pop is 2:', stack.pop());  // => 2
console.log('length is 1:', stack.length()); // => 1
console.log('pop is 1:', stack.pop()); // => 1
stack.print(); // => ''
console.log('peek is undefined:', stack.peek()); // => undefined
console.log('pop is undefined:', stack.pop()); // => undefined

```

```
function Node(data) {
  this.data = data;
  this.children = [];
}

function Tree() {
  this.root = null;
}

Tree.prototype.add = function(data, toNodeData) {
  var node = new Node(data);
  var parent = toNodeData ? this.findBFS(toNodeData) : null;
  if(parent) {
    parent.children.push(node);
  } else {
    if(!this.root) {
      this.root = node;
    } else {
      return 'Root node is already assigned';
    }
  }
};
Tree.prototype.remove = function(data) {
  if(this.root.data === data) {
    this.root = null;
  }

  var queue = [this.root];
  while(queue.length) {
    var node = queue.shift();
    for(var i = 0; i < node.children.length; i++) {
      if(node.children[i].data === data) {
        node.children.splice(i, 1);
      } else {
        queue.push(node.children[i]);
      }
    }
  }
};
Tree.prototype.contains = function(data) {
  return this.findBFS(data) ? true : false;
};
Tree.prototype.findBFS = function(data) {
  var queue = [this.root];
  while(queue.length) {
    var node = queue.shift();
    if(node.data === data) {
      return node;
    }
    for(var i = 0; i < node.children.length; i++) {
      queue.push(node.children[i]);
    }
  }
  return null;
};
Tree.prototype._preOrder = function(node, fn) {
  if(node) {
    if(fn) {
      fn(node);
    }
```

```
    for(var i = 0; i < node.children.length; i++) {
      this._preOrder(node.children[i], fn);
    }
  }
};
Tree.prototype._postOrder = function(node, fn) {
  if(node) {
    for(var i = 0; i < node.children.length; i++) {
      this._postOrder(node.children[i], fn);
    }
    if(fn) {
      fn(node);
    }
  }
};
Tree.prototype.traverseDFS = function(fn, method) {
  var current = this.root;
  if(method) {
    this['_' + method](current, fn);
  } else {
    this._preOrder(current, fn);
  }
};
Tree.prototype.traverseBFS = function(fn) {
  var queue = [this.root];
  while(queue.length) {
    var node = queue.shift();
    if(fn) {
      fn(node);
    }
    for(var i = 0; i < node.children.length; i++) {
      queue.push(node.children[i]);
    }
  }
};
Tree.prototype.print = function() {
  if(!this.root) {
    return console.log('No root node found');
  }
  var newline = new Node('|');
  var queue = [this.root, newline];
  var string = '';
  while(queue.length) {
    var node = queue.shift();
    string += node.data.toString() + ' ';
    if(node === newline && queue.length) {
      queue.push(newline);
    }
    for(var i = 0; i < node.children.length; i++) {
      queue.push(node.children[i]);
    }
  }
  console.log(string.slice(0, -2).trim());
};
Tree.prototype.printByLevel = function() {
  if(!this.root) {
    return console.log('No root node found');
  }
  var newline = new Node('\n');
  var queue = [this.root, newline];
```

```
  var string = '';
  while(queue.length) {
    var node = queue.shift();
    string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
    if(node === newline && queue.length) {
      queue.push(newline);
    }
    for(var i = 0; i < node.children.length; i++) {
      queue.push(node.children[i]);
    }
  }
  console.log(string.trim());
};

var tree = new Tree();
tree.add('ceo');
tree.add('cto', 'ceo');
tree.add('dev1', 'cto');
tree.add('dev2', 'cto');
tree.add('dev3', 'cto');
tree.add('cfo', 'ceo');
tree.add('accountant', 'cfo');
tree.add('cmo', 'ceo');
tree.print(); // => ceo | cto cfo cmo | dev1 dev2 dev3 accountant
tree.printByLevel();  // => ceo \n cto cfo cmo \n dev1 dev2 dev3 accountant
console.log('tree contains dev1 is true:', tree.contains('dev1')); // => true
console.log('tree contains dev4 is false:', tree.contains('dev4')); // => false
console.log('--- BFS');
tree.traverseBFS(function(node) { console.log(node.data); }); // => ceo cto cfo cmo
dev1 dev2 dev3 accountant
console.log('--- DFS preOrder');
tree.traverseDFS(function(node) { console.log(node.data); }, 'preOrder'); // => ceo
cto dev1 dev2 dev3 cfo accountant cmo
console.log('--- DFS postOrder');
tree.traverseDFS(function(node) { console.log(node.data); }, 'postOrder'); // => dev1
dev2 dev3 cto accountant cfo cmo ceo
tree.remove('cmo');
tree.print(); // => ceo | cto cfo | dev1 dev2 dev3 accountant
tree.remove('cfo');
tree.print(); // => ceo | cto | dev1 dev2 dev3

```

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

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

Trie.prototype.add = function(word) {
  if(!this.root) {
    return null;
  }
  this._addNode(this.root, word);
};
Trie.prototype._addNode = function(node, word) {
  if(!node || !word) {
    return null;
  }
  node.prefixes++;
  var letter = word.charAt(0);
  var child = node.children[letter];
  if(!child) {
    child = new Node(letter);
    node.children[letter] = child;
  }
  var remainder = word.substring(1);
  if(!remainder) {
    child.isWord = true;
  }
  this._addNode(child, remainder);
};
Trie.prototype.remove = function(word) {
  if(!this.root) {
    return;
  }
  if(this.contains(word)) {
    this._removeNode(this.root, word);
  }
};
Trie.prototype._removeNode = function(node, word) {
  if(!node || !word) {
    return;
  }
  node.prefixes--;
  var letter = word.charAt(0);

  var child = node.children[letter];
  if(child) {
    var remainder = word.substring(1);
    if(remainder) {
      if(child.prefixes === 1) {
        delete node.children[letter];
      } else {
        this._removeNode(child, remainder);
      }
    } else {
      if(child.prefixes === 0) {
```

```
        delete node.children[letter];
      } else {
        child.isWord = false;
      }
    }
  }
};
Trie.prototype.contains = function(word) {
  if(!this.root) {
    return false;
  }
  return this._contains(this.root, word);
};
Trie.prototype._contains = function(node, word) {
  if(!node || !word) {
    return false;
  }
  var letter = word.charAt(0);
  var child = node.children[letter];
  if(child) {
    var remainder = word.substring(1);
    if(!remainder && child.isWord) {
      return true;
    } else {
      return this._contains(child, remainder);
    }
  } else {
    return false;
  }
};
Trie.prototype.countWords = function() {
  if(!this.root) {
    return console.log('No root node found');
  }
  var queue = [this.root];
  var counter = 0;
  while(queue.length) {
    var node = queue.shift();
    if(node.isWord) {
      counter++;
    }
    for(var child in node.children) {
      if(node.children.hasOwnProperty(child)) {
        queue.push(node.children[child]);
      }
    }
  }
  return counter;
};
Trie.prototype.getWords = function() {
  var words = [];
  var word = '';
  this._getWords(this.root, words, word);
  return words;
};
Trie.prototype._getWords = function(node, words, word) {
  for(var 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);
    }
  }
};
Trie.prototype.print = function() {
  if(!this.root) {
    return console.log('No root node found');
  }
  var newline = new Node('|');
  var queue = [this.root, newline];
  var string = '';
  while(queue.length) {
    var node = queue.shift();
    string += node.data.toString() + ' ';
    if(node === newline && queue.length) {
      queue.push(newline);
    }
    for(var child in node.children) {
      if(node.children.hasOwnProperty(child)) {
        queue.push(node.children[child]);
      }
    }
  }
  console.log(string.slice(0, -2).trim());
};
Trie.prototype.printByLevel = function() {
  if(!this.root) {
    return console.log('No root node found');
  }
  var newline = new Node('\n');
  var queue = [this.root, newline];
  var string = '';
  while(queue.length) {
    var node = queue.shift();
    string += node.data.toString() + (node.data !== '\n' ? ' ' : '');
    if(node === newline && queue.length) {
      queue.push(newline);
    }
    for(var child in node.children) {
      if(node.children.hasOwnProperty(child)) {
        queue.push(node.children[child]);
      }
    }
  }
  console.log(string.trim());
};

var 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
```

```
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
```