Prompt | Scratchpad | Our Solution(s) | Video Explanation | Run Code

Solution 1

```
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

class Node {
  constructor(value) {
    this.value = value;
    this.prev = null;
    this.next = null;
  }
}

class DoublyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  // O(1) time | O(1) space
  setHead(node) {
    if (this.head === null) {
      this.head = node;
      this.tail = node;
      return;
    }
    this.insertBefore(this.head, node);
  }

  // O(1) time | O(1) space
  setTail(node) {
    if (this.tail === null) {
      this.setHead(node);
      return;
    }
    this.insertAfter(this.tail, node);
  }

  // O(1) time | O(1) space
  insertBefore(node, nodeToInsert) {
    if (nodeToInsert === this.head && nodeToInsert === this.tail) ret
    this.remove(nodeToInsert);
    nodeToInsert.prev = node.prev;
    nodeToInsert.next = node;
    if (node.prev === null) {
      this.head = nodeToInsert;
    } else {
      node.prev.next = nodeToInsert;
    }
    node.prev = nodeToInsert;
  }

  // O(1) time | O(1) space
  insertAfter(node, nodeToInsert) {
    if (nodeToInsert === this.head && nodeToInsert === this.tail) ret
    this.remove(nodeToInsert);
    nodeToInsert.prev = node;
    nodeToInsert.next = node.next;
    if (node.next === null) {
      this.tail = nodeToInsert;
    } else {
      node.next.prev = nodeToInsert;
    }
    node.next = nodeToInsert;
  }

  // O(p) time | O(1) space
  insertAtPosition(position, nodeToInsert) {
    if (position === 1) {
      this.setHead(nodeToInsert);
      return;
    }
    let node = this.head;
    let currentPosition = 1;
    while (node !== null && currentPosition++ !== position) node = nc
    if (node !== null) {
      this.insertBefore(node, nodeToInsert);
    } else {
      this.setTail(nodeToInsert);
    }
```

Your Solutions | Run Code

Solution 1 | Solution 2 | Solution 3

```
// This is an input class. Do not edit.
class Node {
  constructor(value) {
    this.value = value;
    this.prev = null;
    this.next = null;
  }
}

// Feel free to add new properties and methods to the class.
class DoublyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  setHead(node) {
    // Write your code here.
  }

  setTail(node) {
    // Write your code here.
  }

  insertBefore(node, nodeToInsert) {
    // Write your code here.
  }

  insertAfter(node, nodeToInsert) {
    // Write your code here.
  }

  insertAtPosition(position, nodeToInsert) {
    // Write your code here.
  }

  removeNodesWithValue(value) {
    // Write your code here.
  }

  remove(node) {
    // Write your code here.
  }

  containsNodeWithValue(value) {
    // Write your code here.
  }
}

// Do not edit the line below.
exports.Node = Node;
exports.DoublyLinkedList = DoublyLinkedList;
```

Custom Output | Raw Output | Submit Code

```
78  }
79
80  // O(n) time | O(1) space
81  removeNodesWithValue(value) {
82    let node = this.head;
83    while (node !== null) {
84      const nodeToRemove = node;
85      node = node.next;
86      if (nodeToRemove.value === value) this.remove(nodeToRemove);
87    }
88  }
89
90  // O(1) time | O(1) space
91  remove(node) {
92    if (node === this.head) this.head = this.head.next;
93    if (node === this.tail) this.tail = this.tail.prev;
94    this.removeNodeBindings(node);
95  }
96
97  // O(n) time | O(1) space
98  containsNodeWithValue(value) {
99    let node = this.head;
100   while (node !== null && node.value !== value) node = node.next;
101   return node !== null;
102 }
103
104 removeNodeBindings(node) {
```

Run or submit code when you're ready.