Recursive Case

2 min

Now it's time to add a recursive case. The recursive case should execute when the node has not been found and the end of the list has not been reached.

Because you've covered both of the base cases, you can use an else statement to call your recursive case.

Instructions

1. Checkpoint 1 Passed

1.

Add an else block to .findNodeRecursively() that returns a call to .findNodeRecursively().

Pass data and the next node as arguments.

Hint

Under the base cases, add an else block.

Inside the else block, return a call to this.findNodeRecursively(). Pass data and currentNode.next as arguments.

2. Checkpoint 2 Passed

2.

In **index.js**, change the argument passed to .findNodeRecursively() to 'Node 2'.

Node.js

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

  setNextNode(node) {
    if (!(node instanceof Node)) {
      throw new Error('Next node must be a member of the Node class');
    }
    this.next = node;
  }

  setNext(data) {
    this.next = data;
  }

  getNextNode() {
    return this.next;
  }
}

module.exports = Node;
```

index.js

```
const Node = require('./Node');
const LinkedList = require('./LinkedList');

const myList = new LinkedList();

myList.addToHead('Node 1');
myList.addToHead('Node 2');
myList.addToHead('Node 3');
myList.addToHead('Node 4');

// Add checkpoint 2 code below:
const myNodeRecursive = myList.findNodeRecursively('Node 2');
console.log(myNodeRecursive);
```

LinkedList.js

```
const Node = require('./Node');

class LinkedList {
  constructor() {
```

```
  this.head = null;
}

addToHead(data) {
  const nextNode = new Node(data);
  const currentHead = this.head;
  this.head = nextNode;
  if (currentHead) {
    this.head.setNextNode(currentHead);
  }
}

addToTail(data) {
  let lastNode = this.head;
  if (!lastNode) {
    this.head = new Node(data);
  } else {
    let temp = this.head;
    while (temp.getNextNode() !== null) {
      temp = temp.getNextNode();
    }
    temp.setNextNode(new Node(data));
  }
}

removeHead() {
  const removedHead = this.head;
  if (!removedHead) {
    return;
  }
  if (removedHead.next) {
    this.head = removedHead.next;
  }
  return removedHead.data;
}

printList() {
  let currentNode = this.head;
  let output = '<head> ';
  while (currentNode !== null) {
    output += currentNode.data + ' ';
    currentNode = currentNode.next;
  }
  output = output.concat("<tail>");
  console.log(output);
}

findNodeIteratively(data) {
  let currentNode = this.head;
  while (currentNode !== null) {
    if (currentNode.data === data) {
      return currentNode;
    }
    currentNode = currentNode.next;
  }
  return null;
}

findNodeRecursively(data, currentNode = this.head) {
  if (currentNode === null) {
    return null;
  } else if (currentNode.data === data) {
    return currentNode;
  }
```

```
    else {
      return this.findNodeRecursively(data, currentNode.next)
    }

  }

}

module.exports = LinkedList;
```