Review

<1 min

In this lesson, you learned how to implement a recursive solution to a linked list search. The solution includes the following cases:

- *Base case 1* – return the current node if it matches the data

Preview: Docs Loading link description

argument

.

- *Base case 2* – return null if the end of the linked list is reached.
- *Recursive Case* – return a call to .findNodeRecursively() with the next node as an argument.

The recursive approach laid out in this lesson is similar to implementations for traversing other

Preview: Docs Loading link description

data structures

, like trees and directories. This is an important insight to keep in mind as you encounter more recursive implementations.

Instructions

Nice work completing this lesson. Recursion is tough! Don't worry if you struggle to implement recursion solutions going forward. The more exposure you have to the concept, the easier it will become.

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;
```

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.findNodeIteratively('Node 2');
console.log(myNodeRecursive);
```