Collisions: Looping

13 min

We've added code to .assign() that takes care of an empty list, but what happens when there is a collision and there are already values stored at a particular index?

If there are already values stored in nodes at an index, we need to loop over each node in the list in order to determine how to proceed.

The two possibilities we'll encounter while looping are:

- The key we are looking for and the key of the current node is the same, so we should overwrite the value
- No node in the linked list matches the key, so we should add the key-value pair to the list as the tail node

After both cases, if we haven't already exited the loop, we should reset the loop's condition.

With this in mind, the .assign() code for looping should look like this:

store the head node of the linked list in a variable current

while there is a current node

  if the current node's key is the same as the key

    store the key and value in current

  if the current node is the tail node

    store the key-value pair in the node after current

    exit the loop

  set current to the next node

Instructions

1. Checkpoint 1 Passed

1.

After the if statement in .assign() that checks for a head node, declare a variable that can be changed, current. Store the head node of the linked list in current.

We'll use current to begin iterating over the linked list until we find the tail node.

Hint

The head node can be accessed through the LinkedList property .head.

2. Checkpoint 2 Passed

2.

After declaring current in .assign(), iterate over the linked list to find the tail using a while loop.

While we haven't visited all the nodes in a linked list, we should keep looping. Set the while loop condition to continue while current isn't null.

Hint

We stored the current node in the current variable.

3. Checkpoint 3 Passed

3.

Key-value pairs are stored in the data property of nodes. (To review how nodes are implemented, see the code for the Node class in **Node.js**.)

There are two possibilities when iterating over nodes:

- The current key and the node's key are the same, and we should overwrite the node's value with the current value
- The current key and the node's key aren't the same and we should check if there are more nodes in the linked list

Add an if statement in the while loop that does a strict equality check of the current node's .key and key. If the two keys are the same, overwrite the current node's key and value properties with the key-value pair we want to store.

Hint

Nodes have a .data property that can be used to store a key-value pair in an object.

```
// You can reassign the data property with a literal
node.data = {key, value};

// Or dot notation
node.data.key = key;
node.data.value = value;
```

4. Checkpoint 4 Passed

4.

Each node knows the node after it. We can use this to determine the tail of the linked list.

Outside of the last if statement, write a condition that checks if the current node is the tail node. Check what Node methods are available to you in the **Node.js** file.

Hint

The .getNextNode() Node method returns the next node.

5. Checkpoint 5 Passed

5.

If the current node is the end of the linked list, there are no more nodes to loop over and check for a matching key.

Inside the if condition that checks for a tail node, set the next node after current to a new node with the key-value pair stored in it. (Check the **Node.js** file to see what methods are available to you to do this.) Then break out of the while loop.

Hint

Nodes have methods that allow us to interact with the next node in a linked list.

```
// The next node can be set by passing in a new node
const lastNode = new Node({ key, value });
currentNode.setNextNode(lastNode);
```

6. Checkpoint 6 Passed

6.

If we don't reach the end of the linked list on this iteration, we need to check the next node.

Outside of the last if statement in the while loop set current to the next node in the linked list to continue the loop. Use the Node method that gets the next node in a linked list.

HashMap.js

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

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

class HashMap {
 constructor(size = 0) {
  this.hashmap = new Array(size)
   .fill(null)
   .map(() => new LinkedList());
 }


 hash(key) {
  let hashCode = 0;
  for (let i = 0; i < key.length; i++) {
   hashCode += hashCode + key.charCodeAt(i);
  }
  return hashCode % this.hashmap.length;
 }


 assign(key, value) {
```

```
    const arrayIndex = this.hash(key);
    const linkedList = this.hashmap[arrayIndex];
    if (linkedList.head === null) {
      linkedList.addToHead({ key, value });
      return;
    }
    let current = linkedList.head;
    while (current !== null) {
      if (current.data.key === key) {
        current.data.key = key;
        current.data.value = value;
      }

      if (current.getNextNode() === null) {
        const lastNode = new Node({ key, value});
        current.setNextNode(lastNode);
        break;
      }

      current = current.getNextNode();
    }
  }
}

module.exports = HashMap;
```