Review

4 min

Now that we have all of the functionality of a hash map, it's time to review what we've learned. Use what you know about hash maps to track birds for New York's annual bird census.

Instructions

1. Checkpoint 1 Passed

1.

Create a constant that stores a hash map, birdCensus. We'll use the hash map data structure to store all bird sightings. Give it an array size of 16.

2. Checkpoint 2 Passed

2.

It's essential for our census that we know the type of bird seen and the location where it was spotted.

Assign the following key-value pairs to birdCensus:

- Key: 'mandarin duck', Value: 'Central Park Pond'
- Key: 'monk parakeet', Value: 'Brooklyn College'
- Key: 'horned owl', Value: 'Pelham Bay Park'

3. Checkpoint 3 Passed

3.

Retrieve the location for each of the three birds counted in birdCensus. Log them to the terminal.

4. Checkpoint 4 Passed

4.

Congratulations, you implemented a fully-functional hash map! Some things to consider:

- How would you delete a key-value pair from this hash map?
- Are there any other ways of handling collisions besides separate chaining? What would be the advantages or disadvantages of a method of avoiding separate chaining?

census.js

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


const birdCensus = new HashMap(16);


birdCensus.assign('mandarin duck', 'Central Park Pond');

birdCensus.assign('monk parakeet', 'Brooklyn College');
```

```
birdCensus.assign('horned owl', 'Pelham Bay Park');

console.log(birdCensus.retrieve('mandarin duck'));
console.log(birdCensus.retrieve('monk parakeet'));
console.log(birdCensus.retrieve('horned owl'));
```

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];
    console.log(`Storing ${value} at index ${arrayIndex}`);
    if (linkedList.head === null) {
      linkedList.addToHead({ key, value });
      return;
```

```
    }
    let current = linkedList.head;
    while (current) {
      if (current.data.key === key) {
        current.data = { key, value };
      }
      if (!current.next) {
        current.next = new Node({ key, value });
        break;
      }
      current = current.next;
    }
  }

  retrieve(key) {
    const arrayIndex = this.hash(key);
    let current = this.hashmap[arrayIndex].head;
    while (current) {
      if (current.data.key === key) {
        console.log(`\nRetrieving ${current.data.value} from index ${arrayIndex}`);
        return current.data.value;
      }
      current = current.next;
    }
    return null;
  }
}

module.exports = HashMap;
```

LinkedList.js

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

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

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

  addToTail(data) {
    let tail = this.head;
    if (!tail) {
      this.head = new Node(data);
    } else {
      while (tail.getNextNode() !== null) {
        tail = tail.getNextNode();
      }
      tail.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 += `<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;
```

test.js

```
let userLog = '';
console.log = function(userPrint) {
  userLog += userPrint;
};
const { expect } = require('chai');
const sinon = require('sinon');

describe('', function () {
  it('', function() {
    const HashMap = require('../HashMap');
    const retrieveSpy = sinon.spy(HashMap.prototype, 'retrieve');

    const birdCensus = require('../census');
    const msgs = [
      'Retrieving Central Park Pond from index 5',
      'Central Park Pond',
      'Retrieving Brooklyn College from index 10',
      'Brooklyn College',
      'Retrieving Pelham Bay Park from index 6',
      'Pelham Bay Park'
    ];

    expect(retrieveSpy.called, 'Make sure to call the `.retrieve()` method on your `HashMap` instance.').to.equal(true);
```

```
    // check all values logged
    for (let i = 0; i < msgs.length; i++) {
      let pattern = new RegExp(msgs[i], 'gi');
      expect(pattern.test(userLog), `Did you log all the retrieved values? We did not see \`${msgs[i]}\` logged.`).to.equal(true);
    }
  });
});
```