Heaps: JavaScript

Introduction

4 min

A heap data structure is a specialized tree data structure that satisfies the heap condition:

- In a max-heap, for any given element, its parent's value is greater than or equal to its value.
- In a min-heap, for any given element, its parent's value is less than or equal to its value.

A heap data structure is commonly implemented as a

Preview: Docs Loading link description

binary

tree. In this lesson, we're going to implement a min-heap in JavaScript. Min-heaps efficiently keep track of the minimum value in a dataset, even as we add and remove elements.

Heaps enable solutions for complex problems such as finding the shortest path (Dijkstra's Algorithm) or efficiently sorting a dataset (heapsort).

They're an essential tool for confidently navigating some of the difficult questions posed in a technical interview.

By understanding the operations of a heap, you will have made a valuable addition to your problem-solving toolkit.

Instructions

1. Checkpoint 1 Passed

1.

The code in **script.js** creates a min-heap one element at a time from a random collection of numbers. It then removes the minimum value from the min-heap one at a time as well.

Run the code a few times to see the effects of adding and removing items in the min-heap printed to the screen.

Move to the next exercise when you're ready to dig in further!

script.js

```
// import MinHeap class
const MinHeap = require('./MinHeap');

// instantiate a MinHeap class
const minHeap = new MinHeap();

// helper function to return a random integer
const randomize = () => Math.floor(Math.random() * 40);

// populate minHeap with random numbers
for (let i = 0; i < 6; i++) {
  const num = randomize();
  console.log(`.. Adding value ${num}`);
  minHeap.add(num);
  console.log('Content of min-heap', minHeap.heap);
}

// return the minimum value in the heap until heap is empty
console.log('\n');
for (let i = 0; i < 6; i++) {
  console.log(`.. Removing minimum value ${minHeap.popMin()}`);
  console.log('Content of min-heap', minHeap.heap);
}
```

MinHeap.js

```
class MinHeap {
  constructor() {
    this.heap = [null];
    this.size = 0;
  }

  add(value) {
    this.heap.push(value);
    this.size++;
    this.bubbleUp();
  }

  popMin() {
    if (this.size === 0) {
      return null
    }
    const min = this.heap[1];
    this.heap[1] = this.heap[this.size];
    this.size--;
    this.heap.pop();
    this.heapify();
    return min;
  }

  bubbleUp() {
    let current = this.size;
    while (current > 1 && this.heap[getParent(current)] > this.heap[current]) {
      this.swap(current, getParent(current));
      current = getParent(current);
    }
  }


  heapify() {
    let current = 1;
    let leftChild = getLeft(current);
    let rightChild = getRight(current);
    // Check that there is something to swap (only need to check the left if both exist)
    while (this.canSwap(current, leftChild, rightChild)){
      // Only compare left & right if they both exist
      if (this.exists(leftChild) && this.exists(rightChild)) {
        // Make sure to swap with the smaller of the two children
        if (this.heap[leftChild] < this.heap[rightChild]) {
          this.swap(current, leftChild);
          current = leftChild;
        } else {
          this.swap(current, rightChild);
          current = rightChild;
        }
      } else {
        // If only one child exist, always swap with the left
        this.swap(current, leftChild);
        current = leftChild;
      }
      leftChild = getLeft(current);
      rightChild = getRight(current);
    }
  }
```

```
  swap(a, b) {
    [this.heap[a], this.heap[b]] = [this.heap[b], this.heap[a]];
  }

  exists(index) {
    return index <= this.size;
  }

  canSwap(current, leftChild, rightChild) {
    // Check that one of the possible swap conditions exists
    return (
      this.exists(leftChild) && this.heap[current] > this.heap[leftChild]
      || this.exists(rightChild) && this.heap[current] > this.heap[rightChild]
    );
  }
}

const getParent = current => Math.floor((current / 2));
const getLeft = current => current * 2;
const getRight = current => current * 2 + 1;

module.exports = MinHeap;
```

>> Output

```
.. Adding value 17
Content of min-heap [ null, 17 ]
.. Adding value 9
Content of min-heap [ null, 9, 17 ]
.. Adding value 38
Content of min-heap [ null, 9, 17, 38 ]
.. Adding value 37
Content of min-heap [ null, 9, 17, 38, 37 ]
.. Adding value 27
Content of min-heap [ null, 9, 17, 38, 37, 27 ]
.. Adding value 28
Content of min-heap [ null, 9, 17, 28, 37, 27, 38 ]


.. Removing minimum value 9
Content of min-heap [ null, 17, 27, 28, 37, 38 ]
.. Removing minimum value 17
Content of min-heap [ null, 27, 37, 28, 38 ]
.. Removing minimum value 27
Content of min-heap [ null, 28, 37, 38 ]
.. Removing minimum value 28
Content of min-heap [ null, 37, 38 ]
.. Removing minimum value 37
Content of min-heap [ null, 38 ]
.. Removing minimum value 38
Content of min-heap [ null ]
```