Review

2 min

Nice work! You've implemented a min-heap in JavaScript, and that's no small feat (although it could efficiently track the smallest feat).

To recap: MinHeap tracks the minimum element as the element at

Preview: Docs Loading link description

index

1 within an internal Javascript

Preview: Docs Loading link description

array

.

When adding elements, we use .bubbleUp() to compare the new element with its parent, making swaps if it violates the heap condition: children must be greater than their parents.

When removing the minimum element, we swap it with the last element in the heap. Then we use .heapify() to compare the new root with its children, swapping with the smaller child if necessary.

Heaps are useful because they're efficient in maintaining their heap condition. Building a heap using elements that decrease in value would ensure that we continually violate the heap condition. How many swaps would that cause?

Instructions

1. Checkpoint 1 Passed

1.

Run the code in **script.js** to see how many swaps are made in a dataset of 10000 elements! We added extra lines of code to keep a count on the number of swaps in .bubbleUp() and .heapify() and log a message when the heap size has reached the 10000th element in .bubbleUp() and 9999 elements in .heapify().

The number of swaps can be at most the height of the binary tree. The relationship between the maximum number of nodes, N, of a binary tree and the height, h, is:

N=2h+1−1*N*=2*h*+1−1

For a height of 13, the maximum number of nodes is

214−1=16383.214−1=16383.

For a height of 12, the maximum number of nodes is

213−1=8191.213−1=8191.

Since 10000 falls between 8191 and 16383, the number of swaps can be at most 13.

MinHeap.js

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

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

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

  bubbleUp() {
```

```
  let current = this.size;
  let swapCount = 0;
  while (current > 1 && this.heap[getParent(current)] > this.heap[current]) {
    this.swap(current, getParent(current));
    current = getParent(current);
    swapCount++;
  }
  if (this.size == 10000) {
    console.log(`Heap of ${this.size} elements restored with ${swapCount} swaps`);
  }
}

heapify() {
  let current = 1;
  let leftChild = getLeft(current);
  let rightChild = getRight(current);
  let swapCount = 0;

  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;
  swapCount++;
      } else {
        this.swap(current, rightChild);
    current = rightChild;
 swapCount++;
   }
  } else {
   // If only one child exist, always swap with the left
   this.swap(current, leftChild);
   current = leftChild;
swapCount++;
  }
  leftChild = getLeft(current);
  rightChild = getRight(current);

 }
 if (this.size == 9999) {
  console.log(`Heap of ${this.size} elements restored with ${swapCount} swaps`);
 }
}

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]
 );
}
```

```
  }

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

}

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

module.exports = MinHeap;
```

script.js

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

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

// populate minHeap with descending numbers from 10001 to 1
console.log('Adding');
for (let i=10000; i >=1; i--) {
  minHeap.add(i);
}

// remove the minimum value from heap
```

```
console.log('Removing');

console.log('Minimum value = ' + minHeap.popMin());
```

>> Output

```
Adding

Heap of 10000 elements restored with 13 swaps

Removing

Heap of 9999 elements restored with 12 swaps

Minimum value = 1
```