Heapify II

9 min

In .bubbleUp(), we were always comparing our element with its parent. In .heapify(), we have potentially two options: the left child and the right child.

Which should we choose? We'll use an example to think it through. Imagine we have a heap with four elements:

```
console.log(minHeap.heap)
// [null, 21, 36, 58, 42]
minHeap.popMin()
// 21
// [null, 42, 36, 58]
// Should we swap 42 with 36 or 58?
```

We want to swap with the smaller of the two children, otherwise, we wouldn't maintain our heap condition!

Instructions

1. Checkpoint 1 Passed

1.

In .heapify() at the beginning of the while loop, check to see if leftChild and rightChild both exist. Use the helper method .exists() to check for the existence of an element at a particular index.

Hint

To check if an element exists using .exists() try the following:

```
if (this.exists(element)) {
}
```

2. Checkpoint 2 Passed

2.

If both children exist, we need to only swap with the smaller of the two. Inside the if statement that checks for the existence of both children, compare the left child's value with the right child's value.

If the left child is smaller than the right child:

- swap the current element with the left child
- update the current element index to be the left child

Otherwise, if the right child is smaller than the left child:

- swap the current element with the right child
- update the current element index to be the right child

Caveat: Executing “Run” may cause an infinite while loop if the if and else statements are left blank or have incorrect content. You can refresh the page to stop it.

Hint

To compare elements in a heap, do something like this:

```
if (this.heap[index1] < this.heap[index2]) {
  // statements to follow
}
```

To swap two elements with .swap() do the following:

```
const a = 5, b = 7;
this.swap(a, b);
console.log(a, b); // should return 7 5
```

3. Checkpoint 3 Passed

3.

If only one child exists, it has to be the left child. Write an else block to the outer if statement that:

- swaps the current element with the left child, and
- updates the current element index to be the left child

Caveat: Executing “Run” may cause an infinite while loop if the else statement is left blank or has incorrect content.

Hint

Write your else statement block to correspond to the outer if statement block.

```
if (this.exists(leftChild) && this.exists(rightChild)) {
  ...
} else {
  // swap the current element and the left child
  // update the current element index to the left child
}
```

4. Checkpoint 4 Passed

4.

Go back into .popMin() and make a call to .heapify() before we return min.

5. Checkpoint 5 Passed

5.

Open **script.js** and run the code inside. Study the output to strengthen your understanding of the .heapify() method.

MinHeap.js

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

 popMin() {
  if (this.size === 0) {
   return null
  }
  console.log(`\n.. Swap ${this.heap[1]} with last element ${this.heap[this.size]}`);
  this.swap(1, this.size);
  const min = this.heap.pop();
  this.size--;
  console.log(`.. Removed ${min} from heap`);
  console.log('..',this.heap);
  this.heapify();
  return min;
 }

 add(value) {
```

```
  console.log(`.. adding ${value}`);
  this.heap.push(value);
  this.size++;
  this.bubbleUp();
  console.log(`added ${value} to heap`, this.heap);
 }

 bubbleUp() {
  let current = this.size;
  while (current > 1 && this.heap[getParent(current)] > this.heap[current]) {
   console.log(`.. swap ${this.heap[current]} with parent ${this.heap[getParent(current)]}`);
   this.swap(current, getParent(current));
   console.log('..', this.heap);
   current = getParent(current);
  }
 }

 heapify() {
  console.log('Heapify');
  let current = 1;
  let leftChild = getLeft(current);
  let rightChild = getRight(current);

  while (this.canSwap(current, leftChild, rightChild)) {
   if (this.exists(leftChild) && this.exists(rightChild)) {
    if (this.heap[leftChild] < this.heap[rightChild]) {
     this.swap(current, leftChild);
     current = leftChild;
    } else {
```

```
        this.swap(current, rightChild);
        current = rightChild;
      }
    } else {
      this.swap(current, leftChild);
      current = leftChild;
    }
    leftChild = getLeft(current);
    rightChild = getRight(current);
  }
}

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();

// helper function to return a random integer
function randomize() { return Math.floor(Math.random() * 40); }

// populate minHeap with random numbers
for (let i=0; i < 6; i++) {
  minHeap.add(randomize());
}

// display the bubbled up numbers in the heap
console.log('Bubbled Up', minHeap.heap);

// remove the minimum value from heap
for (let i=0; i < 6; i++) {
  minHeap.popMin();
  console.log('Heapified', minHeap.heap);
```

}

>> Output

.. adding 8

added 8 to heap [null, 8]

.. adding 39

added 39 to heap [null, 8, 39]

.. adding 0

.. swap 0 with parent 8

.. [null, 0, 39, 8]

added 0 to heap [null, 0, 39, 8]

.. adding 5

.. swap 5 with parent 39

.. [null, 0, 5, 8, 39]

added 5 to heap [null, 0, 5, 8, 39]

.. adding 38

added 38 to heap [null, 0, 5, 8, 39, 38]

.. adding 30

added 30 to heap [null, 0, 5, 8, 39, 38, 30]

Bubbled Up [null, 0, 5, 8, 39, 38, 30]

.. Swap 0 with last element 30

.. Removed 0 from heap

.. [null, 30, 5, 8, 39, 38]

Heapify

Heapified [null, 5, 30, 8, 39, 38]

.. Swap 5 with last element 38

.. Removed 5 from heap

.. [null, 38, 30, 8, 39]

Heapify

Heapified [null, 8, 30, 38, 39]

.. Swap 8 with last element 39

.. Removed 8 from heap

.. [null, 39, 30, 38]

Heapify

Heapified [null, 30, 39, 38]

.. Swap 30 with last element 38

.. Removed 30 from heap

.. [null, 38, 39]

Heapify

Heapified [null, 38, 39]

.. Swap 38 with last element 39

.. Removed 38 from heap

.. [null, 39]

Heapify

Heapified [null, 39]

.. Swap 39 with last element 39

.. Removed 39 from heap

.. [null]

Heapify

Heapified [null]