Removing a Child

18 min

Like with .addChild(), we want to provide a flexible interface for removing a child from a tree based on either its data or a TreeNode match. For example, if our

Preview: Docs Loading link description

method

to remove a child is .removeChild(), we want to be able to execute the following:

```
const blue = 'blue';
const green = new TreeNode('green');
tree.addChild(blue);       // add data
tree.addChild(green);      // add TreeNode
tree.removeChild('blue');  // remove by data
tree.removeChild(green);   // remove by TreeNode
```

The generic steps to execute in removing a child from a tree are as follows:

If target child is an instance of TreeNode,

Compare target child with each child in the children array

Update the children array if target child is found

Else

Compare target child with each child's data in the children array

Update the children array if target child is found

If target child is not found in the children array

Recursively call .removeChild() for each grandchild.

Because we implemented the children as an

Preview: Docs Stores elements of various data types in an ordered collection.

array

, we can use the array .filter() method to update children. Like with .addChild(), we can also use instanceof to check if our target child is an instance of a TreeNode.

Instructions

1. Checkpoint 1 Passed

1.

Define a new method, .removeChild(), that takes one parameter, childToRemove.

2. Checkpoint 2 Passed

2.

Inside .removeChild(), we want to remove the target child from the children array. Use the JavaScript .filter() method to filter out the elements that do not match the target child and reassign the array returned by .filter() back to the children array.

Do the following:

- Call the .filter() method on the children array and supply a callback function with a single argument, child, that does the following:
- If childToRemove is a TreeNode, return true if childToRemove is not equal to child, else return false.
- If childToRemove is not a TreeNode, return true if childToRemove is not equal to child's data, else return false.
- Reassign the return value of .filter() back to children.

Hint

To implement .filter() and its callback function for an array, try something like this:

```
const numbers = [ 1, 2, 3, 4, 5, 6 ];
const evenNumbers = numbers.filter(number => {
  if (number % 2) {
    return false; // odd number
  } else {
    return true;  // even number
  }
})
console.log(evenNumbers); // returns [ 2, 4, 6 ]
```

When we remove a child based on data, we must check that the target child is equivalent to the TreeNode's data. However, when we remove a child based on TreeNode, we are only checking that their TreeNode instances are equivalent.

If target child is an instance of TreeNode,

  Compare target child with child in array

Else

  Compare target child data value with child's data in array

3. Checkpoint 3 Passed

3.

If the target child is not found in the children array, then we would have to descend another level by traversing each child in the array and repeat the process. How do we know that the target child has been removed from the children array? One way is to compare the length of the original children array with the updated children array that has been filtered.

Define a const variable called length and assign it to the length of the children array at the beginning of .removeChild() before the filtering.

Hint

To compute the length of an array, do this:

```
const myarray = [1, 2, 3];
const length = myarray.length;
```

4. Checkpoint 4 Passed

4.

Compare length with the updated children‘s length after filtering. If they are the same, recursively call .removeChild() for each child in the children array.

Hint

We can use the standard .forEach() array method to act upon each child in the children array. For example:

```
// print each element in the fruits array
const fruits = [ 'bananas', 'apples', 'oranges' ]
fruits.forEach(fruit => console.log(fruit); );
```

5. Checkpoint 5 Passed

5.

Now that we have completed implementing .removeChild(), let’s test it. Open **script.js**. A sample tree has been created for you and two children added, one by data and the other by TreeNode.

Do the following:

- Display the output of the tree.
- Remove the element in the tree by data and display the tree.
- Remove the element in the tree by TreeNode and display the tree.

TreeNode.js

```
class TreeNode {
 constructor(data) {
  this.data = data;
  this.children = [];
 }

 addChild(child) {
  if (child instanceof TreeNode) {
   this.children.push(child);
  } else {
```

```
      this.children.push(new TreeNode(child));
    }
  }
  removeChild(childToRemove) {
    const length = this.children.length;
    this.children = this.children.filter(child => {
      if (childToRemove instanceof TreeNode) {
        return childToRemove !== child;
      } else {
        return child.data !== childToRemove;
      }
    });

    if (length === this.children.length) {
      this.children.forEach(child => child.removeChild(childToRemove));
    }
  }
};

module.exports = TreeNode;
```

script.js

```
const TreeNode = require('./TreeNode');
const tree = new TreeNode(1);

tree.addChild(15);
const node = new TreeNode(30);
tree.addChild(node);
console.log(tree);

tree.removeChild(15);
console.log(tree);
```

```
tree.removeChild(node);
console.log(tree);
```

>>Output

```
TreeNode {
  data: 1,
  children:
   [ TreeNode { data: 15, children: [] },
     TreeNode { data: 30, children: [] } ] }
TreeNode { data: 1, children: [ TreeNode { data: 30, children: [] } ] }
TreeNode { data: 1, children: [] }
```