Depth-first Tree Traversal

6 min

Now that we can add nodes to our tree, the next step is to be able to traverse the tree and display its content. We can do this in one of two ways: depth-first or breadth-first.

Depth-first traversal visits the first child in the children

Preview: Docs Loading link description

array

and that node's children recursively before visiting its siblings and their children recursively. The

Preview: Docs An algorithm is a formal process used to solve a problem. They can be represented in several formats but are usually represented in pseudocode in order to communicate the process by which the algorithms solve the problems they were created to tackle.

algorithm

is as follows:

For each node

Display its data

For each child in children, call itself recursively

Based on this tree displayed using .print():

15

-- 3

-- -- 6

-- -- 9

-- 12

-- -- 19

-- -- 8

-- 0

-- -- 10

-- -- 19

we can traverse it depth-wise to produce this result:

15

3

6

9

12

19

8

0

10

19

Instructions

1. Checkpoint 1 Passed

1.

In **TreeNode.js**, define a method, .depthFirstTraversal() below .print() that takes no parameters.

2. Checkpoint 2 Passed

2.

Inside .depthFirstTraversal(), display the data of the current node with console.log.

3. Checkpoint 3 Passed

3.

For each child in the children array, call .depthFirstTraversal() recursively.

Hint

Use the .forEach() method to traverse the children array.

4. Checkpoint 4 Passed

4.

Open **script.js**. Do the following:

- Display the sample tree provided using the .print() method.
- Then, traverse the sample tree using the traversal method you have just created.
- Run the script.
- Study the results by comparing the output
  from .print() and .depthFirstTraversal(). Did you notice anything particular about the ordering of data from both methods?

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 => {
      return childToRemove instanceof TreeNode
      ? child !== childToRemove
      : child.data !== childToRemove;
    });

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

  print(level = 0) {
```

```
    let result = '';
    for (let i = 0; i < level; i++) {
      result += '-- ';
    }
    console.log(`${result}${this.data}`);
    this.children.forEach(child => child.print(level + 1));
  }

  depthFirstTraversal() {
    console.log(this.data);
    this.children.forEach(child => {
      child.depthFirstTraversal();
    });
  }
};

module.exports = TreeNode;
```

script.js

```
const TreeNode = require('./TreeNode');
const tree = new TreeNode(15);
const randomize = () => Math.floor(Math.random() * 20);

// add first-level children
for (let i = 0; i < 3; i++) {
  tree.addChild(randomize());
}

// add second-level children
```

```
for (let i = 0; i < 3; i++) {
  for (let j = 0; j < 2; j++) {
    tree.children[i].addChild(randomize());
  }
}

tree.print()
tree.depthFirstTraversal();
```

>> Output

Output-only Terminal

Output:

```
15
-- 3
-- -- 12
-- -- 14
-- 10
-- -- 6
-- -- 5
-- 10
-- -- 17
-- -- 11
15
3
12
14
10
6
```

5

10

17

11