Breadth-first Tree Traversal

10 min

Breadth-first traversal visits each child in the children

Preview: Docs Loading link description

array

starting from the first child before visiting their children and further layers until the bottom level is visited. The

Preview: Docs Loading link description

algorithm

is as follows:

Assign an array to contain the current root node

While the array is not empty

Extract the first tree node from the array

Display tree node's data

Append tree node's children to the array

Based on this tree displayed using .print():

15

-- 3

-- -- 6

-- -- 9

-- 12

-- -- 19

-- -- 8

-- 0

-- -- 10

-- -- 19

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

15

3

12

0

6

9

19

8

10

19

Let’s implement our breadth-first traversal for TreeNode.

Instructions

1. Checkpoint 1 Passed

1.

Create a new method, .breadthFirstTraversal(), below .depthFirstTraversal() which takes no parameters.

2. Checkpoint 2 Passed

2.

Inside .breadthFirstTraversal(), declare a let variable, queue and assign it to an array that contains the current node as its only element.

Hint

Use an array literal to assign an object or a variable as its element. For example:

```
const number = 3;
let numberlist = [ number ];

const book = { name: 'Harry Potter', author: 'J. K. Rowling' };
let booklist = [ book ];
```

The current TreeNode is referenced using the this keyword.

3. Checkpoint 3 Passed

3.

Create a while loop evaluating if queue is not empty.

Hint

Use the .length property of an array to determine if the array is not empty.

4. Checkpoint 4 Passed

4.

Inside the while loop, extract the first element inside queue and assign it to a const variable, current. We do this so that we can display its data afterwards.

Log the data that belongs to current.

Hint

Use the .shift() array method to extract the first element of an array.

```
const grades = [ 75, 100, 84, 67, 95 ];
const firstNumber = numbers.shift(); // returns 75
```

5. Checkpoint 5 Passed

5.

While still inside the while loop, merge the current tree node's children to the queue and reassign the merger to queue. We do this so that we can traverse the current node's children after we finish traversing its siblings.

Hint

We can use the .concat() array method to merge two arrays. For example:

```
const firstLetters = ['a', 'b', 'c'];
const lastLetters = [ 'x', 'y', 'z' ];
const mergedLetters = firstLetters.concat(lastLetters);
// returns ['a', 'b', 'c', 'x', 'y', 'z'];
```

6. Checkpoint 6 Passed

6.

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

- Display the sample tree provided using the pretty 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 .breadthFirstTraversal().

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());
  }

  breadthFirstTraversal() {
    let queue = [ this ]
    while (queue.length > 0) {
      const current = queue.shift();
      console.log(current.data);
      queue = queue.concat(current.children);
    }
  }
};

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.breadthFirstTraversal();
```

>> Output

Output-only Terminal

Output:

```
15
-- 14
-- -- 11
-- -- 6
-- 17
-- -- 4
-- -- 16
-- 15
-- -- 7
```

-- -- 6

15

14

17

15

11

6

4

16

7

6