9/22/2020

Assignment for W2D4 – Recursion practice

Use the class code given below to generate a little tree data structure. Write recursive functions for the following:

1. Log to the console the names of everyone in the tree.

2. Given a target value, return true or false if there is a node in the tree with the target value. E.g.,
   contains(tree, “Lisa”) → true
   contains(tree, “Crusty”) → false

3. Given a target value, return the subtree with the found node or null if no match. Extend the tree to have a more interesting test. Create a mocha test to run at least 2 or 3 tests on your tree.
   findSubtree(tree, “Homer”) → subtree with Homer as the root

4. Create a new class ListNode (based on TreeNode below) and use it to generate a linked list of Abe, Homer, Bart, Lisa, Maggie instead of a tree.

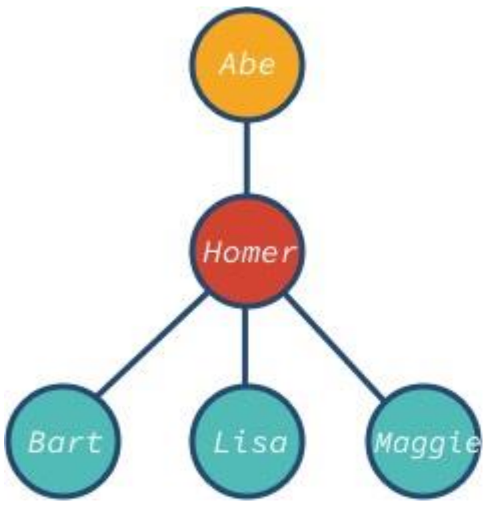
5. Given a target value in the list, return the node that contains the target value or null if no match.
   findListNode(list, “Bart”)

6. Write a recursive function, treeModifier, that will take a tree and a modifier function as parameters. Walk through the tree and apply the function to each node. The function should apply some operation to a node. Write a function that will change the value of a node to be all caps. Write another that will change the value to have *** in front and behind the node value. Write another that will reverse the string of the node value. Call your recursive function with each of these modifier functions.
   treeModifier(tree, modiferFunc)
   allCaps(node)
   addStars(node)
   reverseNode(node)

7. Write code to illustrate the use of the spread operator for the following use cases

   a) Copy an array
   b) Concatenate arrays into a new array
   c) Concatenate an array and a new array element
   d) Use an array as arguments
   e) Use Math.min and Math.max
   f) Combine several objects into a single object

8. Write code to illustrate the use of the rest operator
    a. In a destructurng assignment
    b. In a function call



https://adrianmejia.com/data-structures-for-beginners-trees-binary-search-tree-tutorial/

```
class TreeNode {
  constructor(value) {
    this.value = value;
    this.descendents = [];
  }
}

// create nodes with values
const abe = new TreeNode('Abe');
const homer = new TreeNode('Homer');
const bart = new TreeNode('Bart');
const lisa = new TreeNode('Lisa');
const maggie = new TreeNode('Maggie');

// associate root with is descendents
abe.descendents.push(homer);
homer.descendents.push(bart, lisa, maggie);
```

EXTRA CREDIT

Use the memorization technique from the decorator pattern example to write an efficient recursive algorithm to compute Fibonacci sequences.

- Look at the recursive solution for the Fibonacci exercise in the Recursion section
- Modify the recursive Fibonacci solution to use a decorator as described in the Decorators and forwarding, call/apply > Transparent caching section.
- Try running with n = 40 and n = 45 (careful about any numbers larger than 40, depending on your machine it might take a long time to compute without the decorator)
- Use the timestamp technique from the Date section to measure the performance with and without the decorator.

GRADING RUBRIC

- 8 points. Good status report including:
  - description of which exercises are complete or not complete,
  - Link to Github website page,
  - time spent,
  - problems encountered,
  - outstanding questions,
  - any noteworthy insights
- 9 points
  - Status report as above and most of assignments completed
- 10 points
  - Above and almost everything complete with neat code and output