Red-Black Tree Set in JavaScript
v1.0
Cpt S 489

(read all instructions carefully before writing any code; do all work individually!)

The zip that contains these instructions also includes an HTML and JavaScript code file. The HTML page is set up to reference the JavaScript code files and use them to run tests. Do not change anything in this file. The JavaScript file is where you must write the code.

Implement a class for a red-black tree set such that you can create the tree with a line of code like:

```
var setObj = new Set489(compareFunction);
```

or

```
var setObj = new Set489();
```

Have this class inherit from the BST class developed for the previous assignment. All requirements for that assignment apply to this one as well.

While it may have been declared as optional in the previous assignment, nodes must have a **parent** member in this assignment. You'll find it very difficult to implement the balancing rules for the tree without a reference to parent nodes.

Add the following function to the BST base class. It is used by the tests:

```
// Adds multiple values from an array to the BST. Returns the number of values
// that were added successfully.
BST.prototype.addMultiple = function(arrOfValues)
{
    var count = 0;
    for (var i = 0; i < arrOfValues.length; i++)
    {
        if (this.add(arrOfValues[i]) === true) { count++; }
    }
    return count;
}
```

Include two additional member functions in Set489 (by adding them to the BST base class):
1. A **clear()** member function that takes no parameters and clears out all items in the set. This should be easy to implement and is vital for making sure the tests work.
2. A **countLevels()** member function that can be called with no parameters and returns the number of levels in the tree.

The included .html page in conjunction with the scoring rules listed below will help outline which cases must pass for 2 of 3 points. The challenge point is the same as the previous assignment, where you support tracking the insertion order with a linked-list and iteration in insertion order with the **forEach** function. To serve as a reminder, an excerpt from the function declaration line follows.

```
forEach = function(callback, useInsertionOrder)
```

When you invoke the callback function in your forEach function implementation, give it the value of the node as the first parameter and the reference to the tree as the second parameter.

Scoring:
All tests pass: 3/3
One or more tests pertaining to insertion order and forEach fail, but ALL other tests pass: 2/3
One or more tests pertaining to insertion order and forEach fail, but at least 50% of all other tests pass: 1/3
Anything else: 0/3

Helpful links

Red-black tree on Wikipedia (use the add/insert and remove/delete rules here!):
https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

Red-black tree visualization:
https://www.cs.usfca.edu/~galles/visualization/RedBlack.html