Problem 2691: Immutability Helper

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Creating clones of immutable objects with minor alterations can be a tedious process. Write a class

ImmutableHelper

that serves as a tool to help with this requirement. The constructor accepts an immutable object

obj

which will be a JSON object or array.

The class has a single method

produce

which accepts a function

mutator

. The function returns a new object which is similar to the original except it has those mutations applied.

mutator

accepts a

proxied

version of

obj

. A user of this function can (appear to) mutate this object, but the original object

obj

should not actually be effected.

For example, a user could write code like this:

const originalObj = {"x": 5}; const helper = new ImmutableHelper(originalObj); const newObj = helper.produce((proxy) => { proxy.x = proxy.x + 1; }); console.log(originalObj); // {"x": 5} console.log(newObj); // {"x": 6}

Properties of the

mutator

function:

It will always return

undefined

.

It will never access keys that don't exist.

It will never delete keys (

delete obj.key

)

It will never call methods on a proxied object (

push

,

shift

, etc).

It will never set keys to objects (

proxy.x = {}

)

Note on how the solution will be tested:

the solution validator will only analyze differences between what was returned and the original

obj

. Doing a full comparison would be too computationally expensive. Also, any mutations to the original object will result in a wrong answer.

Example 1:

Input:

obj = {"val": 10}, mutators = [proxy => { proxy.val += 1; }, proxy => { proxy.val -= 1; }]

Output:

[{"val": 11}, {"val": 9}]

Explanation:

const helper = new ImmutableHelper({val: 10}); helper.produce(proxy => { proxy.val += 1; }); // { "val": 11 } helper.produce(proxy => { proxy.val -= 1; }); // { "val": 9 }

Example 2:

Input:

obj = {"arr": [1, 2, 3]} mutators = [proxy => {   proxy.arr[0] = 5;   proxy.newVal = proxy.arr[0] + proxy.arr[1]; }]

Output:

[{"arr": [5, 2, 3], "newVal": 7 }]

Explanation:

Two edits were made to the original array. The first element in the array was to set 5. Then a new key was added with a value of 7.

Example 3:

Input:

obj = {"obj": {"val": {"x": 10, "y": 20}}} mutators = [proxy => {   let data = proxy.obj.val;   let temp = data.x;   data.x = data.y;   data.y = temp;   }]

Output:

[{"obj": {"val": {"x": 20, "y": 10}}}]

Explanation:

The values of "x" and "y" were swapped.

Constraints:

2 <= JSON.stringify(obj).length <= 4 * 10

5

mutators

is an array of functions

total calls to produce() < 10

5

Code Snippets

JavaScript:

```
var ImmutableHelper = function(obj) {

};

/**
* @param {Function} mutator
* @return {JSON} clone of obj
*/
ImmutableHelper.prototype.produce = function(mutator) {

};

/**
* const originalObj = {"x": 5};
* const mutator = new ImmutableHelper(originalObj);
* const newObj = mutator.produce((proxy) => {
* proxy.x = proxy.x + 1;
* });
* console.log(originalObj); // {"x: 5"}
* console.log(newObj); // {"x": 6}
*/
```

TypeScript:

```
type JSONValue = null | boolean | number | string | JSONValue[] | { [key:
string]: JSONValue };
type InputObj = Record<string, JSONValue> | Array<JSONValue>;

class ImmutableHelper {

constructor(obj: InputObj) {

}
```

```
produce(mutator: (obj: InputObj) => void) {

}
}

/**
* const originalObj = {"x": 5};
* const mutator = new ImmutableHelper(originalObj);
* const newObj = mutator.produce((proxy) => {
* proxy.x = proxy.x + 1;
* });
* console.log(originalObj); // {"x: 5"}
* console.log(newObj); // {"x": 6}
*/
```

Solutions

JavaScript Solution:

```
/**
* Problem: Immutability Helper
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

var ImmutableHelper = function(obj) {

};

/**
* @param {Function} mutator
* @return {JSON} clone of obj
*/
ImmutableHelper.prototype.produce = function(mutator) {
```

```
};

/**
* const originalObj = {"x": 5};
* const mutator = new ImmutableHelper(originalObj);
* const newObj = mutator.produce((proxy) => {
* proxy.x = proxy.x + 1;
* });
* console.log(originalObj); // {"x: 5"}
* console.log(newObj); // {"x": 6}
*/
```

TypeScript Solution:

```
/**
* Problem: Immutability Helper
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

type JSONValue = null | boolean | number | string | JSONValue[] | { [key:
string]: JSONValue };
type InputObj = Record<string, JSONValue> | Array<JSONValue>;

class ImmutableHelper {

constructor(obj: InputObj) {

}

produce(mutator: (obj: InputObj) => void) {

}
}

/**
* const originalObj = {"x": 5};
```

```
* const mutator = new ImmutableHelper(originalObj);
* const newObj = mutator.produce((proxy) => {
* proxy.x = proxy.x + 1;
* });
* console.log(originalObj); // {"x: 5"}
* console.log(newObj); // {"x": 6}
*/
```