Problem 2630: Memoize II

Problem Information

Difficulty: Hard
Acceptance Rate: 37.88%
Paid Only: No

Problem Description

Given a function `fn`, return a **memoized** version of that function.

A **memoized **function is a function that will never be called twice with the same inputs. Instead it will return a cached value.

`fn` can be any function and there are no constraints on what type of values it accepts. Inputs are considered identical if they are `===` to each other.

Example 1:

Input: getInputs = () => [[2,2],[2,2],[1,2]] fn = function (a, b) { return a + b; } **Output:** [{"val":4,"calls":1},{"val":4,"calls":1},{"val":3,"calls":2}] **Explanation:** const inputs = getInputs(); const memoized = memoize(fn); for (const arr of inputs) { memoized(...arr); } For the inputs of (2, 2): 2 + 2 = 4, and it required a call to fn(). For the inputs of (2, 2): 2 + 2 = 4, but those inputs were seen before so no call to fn() was required. For the inputs of (1, 2): 1 + 2 = 3, and it required another call to fn() for a total of 2.

Example 2:

Input: getInputs = () => [[{},{}],[{},{}],[{},{}]] fn = function (a, b) { return ({...a, ...b}); } **Output:** [{"val":{},"calls":1},{"val":{},"calls":2},{"val":{},"calls":3}] **Explanation:** Merging two empty objects will always result in an empty object. It may seem like there should only be 1 call to fn() because of cache-hits, however none of those objects are === to each other.

Example 3:

Input: getInputs = () => { const o = {}; return [[o,o],[o,o],[o,o]]; } fn = function (a, b) { return ({...a, ...b}); } **Output:** [{"val":{},"calls":1},{"val":{},"calls":1},{"val":{},"calls":1}] **Explanation:** Merging two empty objects will always result in an empty object. The 2nd and 3rd third function calls result in a cache-hit. This is because every object passed in is identical.

Constraints:

* `1 <= inputs.length <= 105` * `0 <= inputs.flat().length <= 105` * `inputs[i][j] != NaN`

Code Snippets

JavaScript:

```
/**
* @param {Function} fn
* @return {Function}
*/
function memoize(fn) {

return function() {

}
}



/**
* let callCount = 0;
* const memoizedFn = memoize(function (a, b) {
* callCount += 1;
* return a + b;
* })
* memoizedFn(2, 3) // 5
* memoizedFn(2, 3) // 5
* console.log(callCount) // 1
*/
```

TypeScript:

```
type Fn = (...params: any) => any

function memoize(fn: Fn): Fn {
```

```
return function() {

}
}


/**
* let callCount = 0;
* const memoizedFn = memoize(function (a, b) {
* callCount += 1;
* return a + b;
* })
* memoizedFn(2, 3) // 5
* memoizedFn(2, 3) // 5
* console.log(callCount) // 1
*/
```