2623. Memoize

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.

You can assume there are **3** possible input functions: `sum`, `fib`, and `factorial`.

- `sum` accepts two integers `a` and `b` and returns `a + b`.
- `fib` accepts a single integer `n` and returns `1` if `n <= 1` or `fib(n - 1) + fib(n - 2)` otherwise.
- `factorial` accepts a single integer `n` and returns `1` if `n <= 1` or `factorial(n - 1) * n` otherwise.

Example 1:

```
Input:
fnName = "sum"
actions = ["call","call","getCallCount","call","getCallCount"]
values = [[2,2],[2,2],[],[1,2],[]]
Output: [4,4,1,3,2]
Explanation:
const sum = (a, b) => a + b;
const memoizedSum = memoize(sum);
memoizedSum(2, 2); // "call" - returns 4. sum() was called as (2, 2) was not seen before.
memoizedSum(2, 2); // "call" - returns 4. However sum() was not called because the same inputs were seen before.
// "getCallCount" - total call count: 1
memoizedSum(1, 2); // "call" - returns 3. sum() was called as (1, 2) was not seen before.
// "getCallCount" - total call count: 2
```

Example 2:

```
Input:
fnName = "factorial"
actions = ["call","call","call","getCallCount","call","getCallCount"]
values = [[2],[3],[2],[],[3],[]]
Output: [2,6,2,2,6,2]
Explanation:
const factorial = (n) => (n <= 1) ? 1 : (n * factorial(n - 1));
const memoFactorial = memoize(factorial);
memoFactorial(2); // "call" - returns 2.
memoFactorial(3); // "call" - returns 6.
memoFactorial(2); // "call" - returns 2. However factorial was not called because 2 was seen before.
// "getCallCount" - total call count: 2
memoFactorial(3); // "call" - returns 6. However factorial was not called because 3 was seen before.
// "getCallCount" - total call count: 2
```

Example 3:

```
Input:
fnName = "fib"
actions = ["call","getCallCount"]
values = [[5],[]]
Output: [8,1]
Explanation:
fib(5) = 8 // "call"
// "getCallCount" - total call count: 1
```

Constraints:

- $0 <= a, b <= 10^5$
- `1 <= n <= 10`
- $0 <= actions.length <= 10^5$
- `actions.length === values.length`
- `actions[i]` is one of "call" and "getCallCount"
- `fnName` is one of "sum", "factorial" and "fib"