Problem 2795: Parallel Execution of Promises for Individual Results Retrieval

Problem Information

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

Problem Description

Given an array

functions

, return a promise

promise

.

functions

is an array of functions that return promises

fnPromise.

Each

fnPromise

can be resolved or rejected.

If

fnPromise

is resolved:

obj = { status: "fulfilled", value:

resolved value

}

If

fnPromise

is rejected:

obj = { status: "rejected", reason:

reason of rejection (catched error message)

}

The

promise

should resolve with an array of these objects

obj

. Each

obj

in the array should correspond to the promises in the original array function,

maintaining the same order

.

Try to implement it without using the built-in method

Promise.allSettled()

.

Example 1:

Input:

functions = [() => new Promise(resolve => setTimeout(() => resolve(15), 100))]

Output:

{"t":100,"values":[{"status":"fulfilled","value":15}]}

Explanation:

const time = performance.now() const promise = promiseAllSettled(functions); promise.then(res => { const out = {t: Math.floor(performance.now() - time), values: res} console.log(out) // {"t":100,"values":[{"status":"fulfilled","value":15}]} })

The returned promise resolves within 100 milliseconds. Since promise from the array functions is fulfilled, the resolved value of the returned promise is set to [{"status":"fulfilled","value":15}].

Example 2:

Input:

functions = [() => new Promise(resolve => setTimeout(() => resolve(20), 100)), () => new Promise(resolve => setTimeout(() => resolve(15), 100))]

Output:

{ "t":100, "values": [{"status":"fulfilled","value":20},   {"status":"fulfilled","value":15}] }

Explanation:

The returned promise resolves within 100 milliseconds, because the resolution time is determined by the promise that takes the longest time to fulfill. Since promises from the array functions are fulfilled, the resolved value of the returned promise is set to [{"status":"fulfilled","value":20},{"status":"fulfilled","value":15}].

Example 3:

Input:

functions = [() => new Promise(resolve => setTimeout(() => resolve(30), 200)),   () => new Promise((resolve, reject) => setTimeout(() => reject("Error"), 100))]

Output:

{ "t":200, "values": [{"status":"fulfilled","value":30}, {"status":"rejected","reason":"Error"}] }

Explanation:

The returned promise resolves within 200 milliseconds, as its resolution time is determined by the promise that takes the longest time to fulfill. Since one promise from the array function is fulfilled and another is rejected, the resolved value of the returned promise is set to an array containing objects in the following order: [{"status":"fulfilled","value":30}, {"status":"rejected","reason":"Error"}]. Each object in the array corresponds to the promises in the original array function, maintaining the same order.

Constraints:

1 <= functions.length <= 10

Code Snippets

JavaScript:

```
/**
* @param {Array<Function>} functions
* @return {Promise<Array>}
*/
var promiseAllSettled = function(functions) {
};



/**
* const functions = [
* () => new Promise(resolve => setTimeout(() => resolve(15), 100))
* ]
* const time = performance.now()
*
* const promise = promiseAllSettled(functions);
*
* promise.then(res => {
* const out = {t: Math.floor(performance.now() - time), values: res}
* console.log(out) // {"t":100,"values":[{"status":"fulfilled","value":15}]}
* })
*/
```

TypeScript:

```
type FulfilledObj = {
status: 'fulfilled';
value: string;
}
type RejectedObj = {
status: 'rejected';
reason: string;
}
type Obj = FulfilledObj | RejectedObj;

function promiseAllSettled(functions: Function[]): Promise<Obj[]> {

};



/**
* const functions = [
* () => new Promise(resolve => setTimeout(() => resolve(15), 100))
* ]
* const time = performance.now()
*
* const promise = promiseAllSettled(functions);
*
```

```
* promise.then(res => {
* const out = {t: Math.floor(performance.now() - time), values: res}
* console.log(out) // {"t":100,"values":[{"status":"fulfilled","value":15}]}
* })
*/
```

Solutions

JavaScript Solution:

```
/**
* Problem: Parallel Execution of Promises for Individual Results Retrieval
* Difficulty: Medium
* Tags: array, math
*
* 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
*/

/**
* @param {Array<Function>} functions
* @return {Promise<Array>}
*/
var promiseAllSettled = function(functions) {

};



/**
* const functions = [
* () => new Promise(resolve => setTimeout(() => resolve(15), 100))
* ]
* const time = performance.now()
*
* const promise = promiseAllSettled(functions);
*
* promise.then(res => {
* const out = {t: Math.floor(performance.now() - time), values: res}
* console.log(out) // {"t":100,"values":[{"status":"fulfilled","value":15}]}
* })
*/
```

TypeScript Solution:

```
/**
* Problem: Parallel Execution of Promises for Individual Results Retrieval
* Difficulty: Medium
* Tags: array, math
*
* 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 FulfilledObj = {
status: 'fulfilled';
value: string;
}
type RejectedObj = {
status: 'rejected';
reason: string;
}
type Obj = FulfilledObj | RejectedObj;

function promiseAllSettled(functions: Function[]): Promise<Obj[]> {

};



/**
* const functions = [
* () => new Promise(resolve => setTimeout(() => resolve(15), 100))
* ]
* const time = performance.now()
*
* const promise = promiseAllSettled(functions);
*
* promise.then(res => {
* const out = {t: Math.floor(performance.now() - time), values: res}
* console.log(out) // {"t":100,"values":[{"status":"fulfilled","value":15}]}
* })
*/
```