Problem 2622: Cache With Time Limit

Problem Information

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

Problem Description

Write a class that allows getting and setting key-value pairs, however a

time until expiration

is associated with each key.

The class has three public methods:

set(key, value, duration)

: accepts an integer

key

, an integer

value

, and a

duration

in milliseconds. Once the

duration

has elapsed, the key should be inaccessible. The method should return

true

if the same un-expired key already exists and

false

otherwise. Both the value and duration should be overwritten if the key already exists.

get(key)

: if an un-expired key exists, it should return the associated value. Otherwise it should return

-1

.

count()

: returns the count of un-expired keys.

Example 1:

Input:

actions = ["TimeLimitedCache", "set", "get", "count", "get"] values = [[], [1, 42, 100], [1], [], [1]] timeDelays = [0, 0, 50, 50, 150]

Output:

[null, false, 42, 1, -1]

Explanation:

At t=0, the cache is constructed. At t=0, a key-value pair (1: 42) is added with a time limit of 100ms. The value doesn't exist so false is returned. At t=50, key=1 is requested and the value of 42 is returned. At t=50, count() is called and there is one active key in the cache. At t=100, key=1 expires. At t=150, get(1) is called but -1 is returned because the cache is empty.

Example 2:

Input:

actions = ["TimeLimitedCache", "set", "set", "get", "get", "get", "count"] values = [[], [1, 42, 50], [1, 50, 100], [1], [1], [1], []] timeDelays = [0, 0, 40, 50, 120, 200, 250]

Output:

[null, false, true, 50, 50, -1, 0]

Explanation:

At t=0, the cache is constructed. At t=0, a key-value pair (1: 42) is added with a time limit of 50ms. The value doesn't exist so false is returned. At t=40, a key-value pair (1: 50) is added with a time limit of 100ms. A non-expired value already existed so true is returned and the old value was overwritten. At t=50, get(1) is called which returned 50. At t=120, get(1) is called which returned 50. At t=140, key=1 expires. At t=200, get(1) is called but the cache is empty so -1 is returned. At t=250, count() returns 0 because the cache is empty.

Constraints:

0 <= key, value <= 10

9

0 <= duration <= 1000

1 <= actions.length <= 100

actions.length === values.length

actions.length === timeDelays.length

0 <= timeDelays[i] <= 1450

actions[i]

is one of "TimeLimitedCache", "set", "get" and "count"

First action is always "TimeLimitedCache" and must be executed immediately, with a 0-millisecond delay

Code Snippets

JavaScript:

```
var TimeLimitedCache = function() {

};

/**
* @param {number} key
* @param {number} value
* @param {number} duration time until expiration in ms
* @return {boolean} if un-expired key already existed
*/
TimeLimitedCache.prototype.set = function(key, value, duration) {

};

/**
* @param {number} key
* @return {number} value associated with key
*/
TimeLimitedCache.prototype.get = function(key) {

};

/**
* @return {number} count of non-expired keys
*/
TimeLimitedCache.prototype.count = function() {

};

/**
* const timeLimitedCache = new TimeLimitedCache()
* timeLimitedCache.set(1, 42, 1000); // false
* timeLimitedCache.get(1) // 42
* timeLimitedCache.count() // 1
```

```
*/
```

TypeScript:

```
class TimeLimitedCache {

constructor() {

}

set(key: number, value: number, duration: number): boolean {

}

get(key: number): number {

}

count(): number {

}
}

/**
* const timeLimitedCache = new TimeLimitedCache()
* timeLimitedCache.set(1, 42, 1000); // false
* timeLimitedCache.get(1) // 42
* timeLimitedCache.count() // 1
*/
```

Solutions

JavaScript Solution:

```
/**
* Problem: Cache With Time Limit
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
```

```
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

var TimeLimitedCache = function() {

};

/**
* @param {number} key
* @param {number} value
* @param {number} duration time until expiration in ms
* @return {boolean} if un-expired key already existed
*/
TimeLimitedCache.prototype.set = function(key, value, duration) {

};

/**
* @param {number} key
* @return {number} value associated with key
*/
TimeLimitedCache.prototype.get = function(key) {

};

/**
* @return {number} count of non-expired keys
*/
TimeLimitedCache.prototype.count = function() {

};

/**
* const timeLimitedCache = new TimeLimitedCache()
* timeLimitedCache.set(1, 42, 1000); // false
* timeLimitedCache.get(1) // 42
* timeLimitedCache.count() // 1
*/
```

TypeScript Solution:

```
/**
* Problem: Cache With Time Limit
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class TimeLimitedCache {

constructor() {

}

set(key: number, value: number, duration: number): boolean {

}

get(key: number): number {

}

count(): number {

}
}

/**
* const timeLimitedCache = new TimeLimitedCache()
* timeLimitedCache.set(1, 42, 1000); // false
* timeLimitedCache.get(1) // 42
* timeLimitedCache.count() // 1
*/
```