Problem 3092: Most Frequent IDs

Problem Information

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

Problem Description

The problem involves tracking the frequency of IDs in a collection that changes over time. You have two integer arrays,

nums

and

freq

, of equal length

n

. Each element in

nums

represents an ID, and the corresponding element in

freq

indicates how many times that ID should be added to or removed from the collection at each step.

Addition of IDs:

If

freq[i]

is positive, it means

freq[i]

IDs with the value

nums[i]

are added to the collection at step

i

.

Removal of IDs:

If

freq[i]

is negative, it means

-freq[i]

IDs with the value

nums[i]

are removed from the collection at step

i

.

Return an array

ans

of length

n

, where

ans[i]

represents the

count

of the

most frequent ID

in the collection after the

i

th

step. If the collection is empty at any step,

ans[i]

should be 0 for that step.

Example 1:

Input:

nums = [2,3,2,1], freq = [3,2,-3,1]

Output:

[3,3,2,2]

Explanation:

After step 0, we have 3 IDs with the value of 2. So

ans[0] = 3

.

After step 1, we have 3 IDs with the value of 2 and 2 IDs with the value of 3. So

ans[1] = 3

.

After step 2, we have 2 IDs with the value of 3. So

ans[2] = 2

.

After step 3, we have 2 IDs with the value of 3 and 1 ID with the value of 1. So

ans[3] = 2

.

Example 2:

Input:

nums = [5,5,3], freq = [2,-2,1]

Output:

[2,0,1]

Explanation:

After step 0, we have 2 IDs with the value of 5. So

ans[0] = 2

.

After step 1, there are no IDs. So

ans[1] = 0

.

After step 2, we have 1 ID with the value of 3. So

ans[2] = 1

.

Constraints:

1 <= nums.length == freq.length <= 10

5

1 <= nums[i] <= 10

5

-10

5

<= freq[i] <= 10

5

freq[i] != 0

The input is generated

such that the occurrences of an ID will not be negative in any step.

Code Snippets

C++:

```
class Solution {
public:
vector<long long> mostFrequentIDs(vector<int>& nums, vector<int>& freq) {

}
};
```

Java:

```
class Solution {
public long[] mostFrequentIDs(int[] nums, int[] freq) {

}
}
```

Python3:

```
class Solution:
def mostFrequentIDs(self, nums: List[int], freq: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def mostFrequentIDs(self, nums, freq):
"""
:type nums: List[int]
:type freq: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number[]} freq
* @return {number[]}
*/
var mostFrequentIDs = function(nums, freq) {

};
```

TypeScript:

```
function mostFrequentIDs(nums: number[], freq: number[]): number[] {

};
```

C#:

```
public class Solution {
public long[] MostFrequentIDs(int[] nums, int[] freq) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* mostFrequentIDs(int* nums, int numsSize, int* freq, int freqSize,
int* returnSize) {

}
```

Go:

```
func mostFrequentIDs(nums []int, freq []int) []int64 {

}
```

Kotlin:

```
class Solution {
fun mostFrequentIDs(nums: IntArray, freq: IntArray): LongArray {
```

```
}
}
```

Swift:

```
class Solution {
func mostFrequentIDs(_ nums: [Int], _ freq: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn most_frequent_i_ds(nums: Vec<i32>, freq: Vec<i32>) -> Vec<i64> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} freq
# @return {Integer[]}
def most_frequent_i_ds(nums, freq)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $freq
* @return Integer[]
*/
function mostFrequentIDs($nums, $freq) {

}
}
```

Dart:

```
class Solution {
List<int> mostFrequentIDs(List<int> nums, List<int> freq) {

}
}
```

Scala:

```
object Solution {
def mostFrequentIDs(nums: Array[Int], freq: Array[Int]): Array[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec most_frequent_i_ds(nums :: [integer], freq :: [integer]) :: [integer]
def most_frequent_i_ds(nums, freq) do

end
end
```

Erlang:

```
-spec most_frequent_i_ds(Nums :: [integer()], Freq :: [integer()]) ->
[integer()].
most_frequent_i_ds(Nums, Freq) ->
.
```

Racket:

```
(define/contract (most-frequent-i-ds nums freq)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Most Frequent IDs
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<long long> mostFrequentIDs(vector<int>& nums, vector<int>& freq) {

}
};
```

Java Solution:

```
/**
* Problem: Most Frequent IDs
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public long[] mostFrequentIDs(int[] nums, int[] freq) {

}
}
```

Python3 Solution:

```
"""
Problem: Most Frequent IDs
Difficulty: Medium
Tags: array, hash, queue, heap
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def mostFrequentIDs(self, nums: List[int], freq: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mostFrequentIDs(self, nums, freq):
"""
:type nums: List[int]
:type freq: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Most Frequent IDs
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number[]} freq
* @return {number[]}
*/
var mostFrequentIDs = function(nums, freq) {

};
```

TypeScript Solution:

```
/**
* Problem: Most Frequent IDs
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function mostFrequentIDs(nums: number[], freq: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Most Frequent IDs
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public long[] MostFrequentIDs(int[] nums, int[] freq) {

}
}
```

C Solution:

```
/*
* Problem: Most Frequent IDs
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* mostFrequentIDs(int* nums, int numsSize, int* freq, int freqSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Most Frequent IDs
// Difficulty: Medium
// Tags: array, hash, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func mostFrequentIDs(nums []int, freq []int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun mostFrequentIDs(nums: IntArray, freq: IntArray): LongArray {

}
}
```

Swift Solution:

```
class Solution {
func mostFrequentIDs(_ nums: [Int], _ freq: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Most Frequent IDs
// Difficulty: Medium
// Tags: array, hash, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn most_frequent_i_ds(nums: Vec<i32>, freq: Vec<i32>) -> Vec<i64> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} freq
# @return {Integer[]}
def most_frequent_i_ds(nums, freq)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $freq
* @return Integer[]
*/
function mostFrequentIDs($nums, $freq) {

}
}
```

Dart Solution:

```
class Solution {
List<int> mostFrequentIDs(List<int> nums, List<int> freq) {

}
}
```

Scala Solution:

```
object Solution {
def mostFrequentIDs(nums: Array[Int], freq: Array[Int]): Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec most_frequent_i_ds(nums :: [integer], freq :: [integer]) :: [integer]
def most_frequent_i_ds(nums, freq) do

end
end
```

Erlang Solution:

```
-spec most_frequent_i_ds(Nums :: [integer()], Freq :: [integer()]) ->
[integer()].
most_frequent_i_ds(Nums, Freq) ->
.
```

Racket Solution:

```
(define/contract (most-frequent-i-ds nums freq)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```