Problem 3679: Minimum Discards to Balance Inventory

Problem Information

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

Problem Description

You are given two integers

w

and

m

, and an integer array

arrivals

, where

arrivals[i]

is the type of item arriving on day

i

(days are

1-indexed

).

Items are managed according to the following rules:

Each arrival may be

kept

or

discarded

; an item may only be discarded on its arrival day.

For each day

i

, consider the window of days

$[\max(1, i - w + 1), i]$

(the

w

most recent days up to day

i

):

For

any

such window, each item type may appear

at most

m

times among kept arrivals whose arrival day lies in that window.

If keeping the arrival on day

i

would cause its type to appear

more than

m

times in the window, that arrival

must

be discarded.

Return the

minimum

number of arrivals to be discarded so that every

w

-day window contains at most

m

occurrences of each type.

Example 1:

Input:

arrivals = [1,2,1,3,1], w = 4, m = 2

Output:

0

Explanation:

On day 1, Item 1 arrives; the window contains no more than

m

occurrences of this type, so we keep it.

On day 2, Item 2 arrives; the window of days 1 - 2 is fine.

On day 3, Item 1 arrives, window

[1, 2, 1]

has item 1 twice, within limit.

On day 4, Item 3 arrives, window

[1, 2, 1, 3]

has item 1 twice, allowed.

On day 5, Item 1 arrives, window

[2, 1, 3, 1]

has item 1 twice, still valid.

There are no discarded items, so return 0.

Example 2:

Input:

arrivals = [1,2,3,3,3,4], w = 3, m = 2

Output:

1

Explanation:

On day 1, Item 1 arrives. We keep it.

On day 2, Item 2 arrives, window

[1, 2]

is fine.

On day 3, Item 3 arrives, window

[1, 2, 3]

has item 3 once.

On day 4, Item 3 arrives, window

[2, 3, 3]

has item 3 twice, allowed.

On day 5, Item 3 arrives, window

[3, 3, 3]

has item 3 three times, exceeds limit, so the arrival must be discarded.

On day 6, Item 4 arrives, window

[3, 4]

is fine.

Item 3 on day 5 is discarded, and this is the minimum number of arrivals to discard, so return 1.

Constraints:

1 <= arrivals.length <= 10

5

1 <= arrivals[i] <= 10

5

1 <= w <= arrivals.length

1 <= m <= w

Code Snippets

C++:

```
class Solution {
public:
int minArrivalsToDiscard(vector<int>& arrivals, int w, int m) {

}
};
```

Java:

```
class Solution {
public int minArrivalsToDiscard(int[] arrivals, int w, int m) {

}
}
```

Python3:

```
class Solution:
def minArrivalsToDiscard(self, arrivals: List[int], w: int, m: int) -> int:
```

Python:

```
class Solution(object):
def minArrivalsToDiscard(self, arrivals, w, m):
"""
:type arrivals: List[int]
:type w: int
:type m: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} arrivals
* @param {number} w
* @param {number} m
* @return {number}
*/
var minArrivalsToDiscard = function(arrivals, w, m) {

};
```

TypeScript:

```
function minArrivalsToDiscard(arrivals: number[], w: number, m: number):
number {

};
```

C#:

```
public class Solution {
public int MinArrivalsToDiscard(int[] arrivals, int w, int m) {

}
}
```

C:

```
int minArrivalsToDiscard(int* arrivals, int arrivalsSize, int w, int m) {

}
```

Go:

```
func minArrivalsToDiscard(arrivals []int, w int, m int) int {

}
```

Kotlin:

```
class Solution {
fun minArrivalsToDiscard(arrivals: IntArray, w: Int, m: Int): Int {

}
}
```

Swift:

```
class Solution {
func minArrivalsToDiscard(_ arrivals: [Int], _ w: Int, _ m: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_arrivals_to_discard(arrivals: Vec<i32>, w: i32, m: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arrivals
# @param {Integer} w
# @param {Integer} m
# @return {Integer}
def min_arrivals_to_discard(arrivals, w, m)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arrivals
* @param Integer $w
* @param Integer $m
* @return Integer
*/
function minArrivalsToDiscard($arrivals, $w, $m) {

}
}
```

Dart:

```
class Solution {
int minArrivalsToDiscard(List<int> arrivals, int w, int m) {

}
}
```

Scala:

```
object Solution {
def minArrivalsToDiscard(arrivals: Array[Int], w: Int, m: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_arrivals_to_discard(arrivals :: [integer], w :: integer, m ::
integer) :: integer
def min_arrivals_to_discard(arrivals, w, m) do

end
end
```

Erlang:

```
-spec min_arrivals_to_discard(Arrivals :: [integer()], W :: integer(), M ::
integer()) -> integer().
min_arrivals_to_discard(Arrivals, W, M) ->
.
```

Racket:

```
(define/contract (min-arrivals-to-discard arrivals w m)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Discards to Balance Inventory
* Difficulty: Medium
* Tags: array, hash
*
* 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:
int minArrivalsToDiscard(vector<int>& arrivals, int w, int m) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Discards to Balance Inventory
* Difficulty: Medium
* Tags: array, hash
*
* 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 int minArrivalsToDiscard(int[] arrivals, int w, int m) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Discards to Balance Inventory
Difficulty: Medium
Tags: array, hash

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 minArrivalsToDiscard(self, arrivals: List[int], w: int, m: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minArrivalsToDiscard(self, arrivals, w, m):
"""
:type arrivals: List[int]
:type w: int
:type m: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Discards to Balance Inventory
* Difficulty: Medium
* Tags: array, hash
*
* 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[]} arrivals
* @param {number} w
* @param {number} m
* @return {number}
*/
var minArrivalsToDiscard = function(arrivals, w, m) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Discards to Balance Inventory
* Difficulty: Medium
* Tags: array, hash
*
* 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 minArrivalsToDiscard(arrivals: number[], w: number, m: number):
number {

};
```

C# Solution:

```
/*
* Problem: Minimum Discards to Balance Inventory
* Difficulty: Medium
* Tags: array, hash
```

```
*
* 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 int MinArrivalsToDiscard(int[] arrivals, int w, int m) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Discards to Balance Inventory
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minArrivalsToDiscard(int* arrivals, int arrivalsSize, int w, int m) {

}
```

Go Solution:

```
// Problem: Minimum Discards to Balance Inventory
// Difficulty: Medium
// Tags: array, hash
//
// 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 minArrivalsToDiscard(arrivals []int, w int, m int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minArrivalsToDiscard(arrivals: IntArray, w: Int, m: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minArrivalsToDiscard(_ arrivals: [Int], _ w: Int, _ m: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Discards to Balance Inventory
// Difficulty: Medium
// Tags: array, hash
//
// 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 min_arrivals_to_discard(arrivals: Vec<i32>, w: i32, m: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arrivals
# @param {Integer} w
# @param {Integer} m
# @return {Integer}
def min_arrivals_to_discard(arrivals, w, m)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arrivals
* @param Integer $w
* @param Integer $m
* @return Integer
*/
function minArrivalsToDiscard($arrivals, $w, $m) {

}
}
```

Dart Solution:

```
class Solution {
int minArrivalsToDiscard(List<int> arrivals, int w, int m) {

}
}
```

Scala Solution:

```
object Solution {
def minArrivalsToDiscard(arrivals: Array[Int], w: Int, m: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_arrivals_to_discard(arrivals :: [integer], w :: integer, m ::
integer) :: integer
def min_arrivals_to_discard(arrivals, w, m) do

end
end
```

Erlang Solution:

```
-spec min_arrivals_to_discard(Arrivals :: [integer()], W :: integer(), M ::
integer()) -> integer().
min_arrivals_to_discard(Arrivals, W, M) ->
  .
```

Racket Solution:

```
(define/contract (min-arrivals-to-discard arrivals w m)
  (-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
  )
```