Problem 1409: Queries on a Permutation With Key

Problem Information

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

Problem Description

Given the array

queries

of positive integers between

1

and

m

, you have to process all

queries[i]

(from

i=0

to

i=queries.length-1

) according to the following rules:

In the beginning, you have the permutation

P=[1,2,3,...,m]

.

For the current

i

, find the position of

queries[i]

in the permutation

P

(

indexing from 0

) and then move this at the beginning of the permutation

P

. Notice that the position of

queries[i]

in

P

is the result for

queries[i]

.

Return an array containing the result for the given

queries

.

Example 1:

Input:

queries = [3,1,2,1], m = 5

Output:

[2,1,2,1]

Explanation:

The queries are processed as follow: For i=0: queries[i]=3, P=[1,2,3,4,5], position of 3 in P is

2

, then we move 3 to the beginning of P resulting in P=[3,1,2,4,5]. For i=1: queries[i]=1, P=[3,1,2,4,5], position of 1 in P is

1

, then we move 1 to the beginning of P resulting in P=[1,3,2,4,5]. For i=2: queries[i]=2, P=[1,3,2,4,5], position of 2 in P is

2

, then we move 2 to the beginning of P resulting in P=[2,1,3,4,5]. For i=3: queries[i]=1, P=[2,1,3,4,5], position of 1 in P is

1

, then we move 1 to the beginning of P resulting in P=[1,2,3,4,5]. Therefore, the array containing the result is [2,1,2,1].

Example 2:

Input:

queries = [4,1,2,2], m = 4

Output:

[3,1,2,0]

Example 3:

Input:

queries = [7,5,5,8,3], m = 8

Output:

[6,5,0,7,5]

Constraints:

1 <= m <= 10^3

1 <= queries.length <= m

1 <= queries[i] <= m

Code Snippets

C++:

```
class Solution {
public:
vector<int> processQueries(vector<int>& queries, int m) {
}
};
```

Java:

```
class Solution {
public int[] processQueries(int[] queries, int m) {

}
}
```

Python3:

```
class Solution:
def processQueries(self, queries: List[int], m: int) -> List[int]:
```

Python:

```
class Solution(object):
def processQueries(self, queries, m):
"""
:type queries: List[int]
:type m: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} queries
* @param {number} m
* @return {number[]}
*/
var processQueries = function(queries, m) {

};
```

TypeScript:

```
function processQueries(queries: number[], m: number): number[] {

};
```

C#:

```
public class Solution {
public int[] ProcessQueries(int[] queries, int m) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* processQueries(int* queries, int queriesSize, int m, int* returnSize) {

}
```

Go:

```
func processQueries(queries []int, m int) []int {

}
```

Kotlin:

```
class Solution {
fun processQueries(queries: IntArray, m: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func processQueries(_ queries: [Int], _ m: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn process_queries(queries: Vec<i32>, m: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} queries
# @param {Integer} m
# @return {Integer[]}
def process_queries(queries, m)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $queries
* @param Integer $m
* @return Integer[]
*/
function processQueries($queries, $m) {

}
}
```

Dart:

```
class Solution {
List<int> processQueries(List<int> queries, int m) {

}
}
```

Scala:

```
object Solution {
def processQueries(queries: Array[Int], m: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec process_queries(queries :: [integer], m :: integer) :: [integer]
def process_queries(queries, m) do

end
end
```

Erlang:

```
-spec process_queries(Queries :: [integer()], M :: integer()) -> [integer()].
process_queries(Queries, M) ->
.
```

Racket:

```
(define/contract (process-queries queries m)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Queries on a Permutation With Key
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> processQueries(vector<int>& queries, int m) {
}
};
```

Java Solution:

```
/**
* Problem: Queries on a Permutation With Key
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int[] processQueries(int[] queries, int m) {

}
}
```

Python3 Solution:

```
"""
Problem: Queries on a Permutation With Key
Difficulty: Medium
Tags: array, tree

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def processQueries(self, queries: List[int], m: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def processQueries(self, queries, m):
"""
:type queries: List[int]
:type m: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Queries on a Permutation With Key
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} queries
* @param {number} m
* @return {number[]}
*/
var processQueries = function(queries, m) {

};
```

TypeScript Solution:

```
/**
* Problem: Queries on a Permutation With Key
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function processQueries(queries: number[], m: number): number[] {
};
```

C# Solution:

```
/*
* Problem: Queries on a Permutation With Key
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int[] ProcessQueries(int[] queries, int m) {

}
}
```

C Solution:

```
/*
* Problem: Queries on a Permutation With Key
* Difficulty: Medium
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* processQueries(int* queries, int queriesSize, int m, int* returnSize) {

}
```

Go Solution:

```
// Problem: Queries on a Permutation With Key
// Difficulty: Medium
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func processQueries(queries []int, m int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun processQueries(queries: IntArray, m: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func processQueries(_ queries: [Int], _ m: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Queries on a Permutation With Key
// Difficulty: Medium
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn process_queries(queries: Vec<i32>, m: i32) -> Vec<i32> {

}

}
```

Ruby Solution:

```
# @param {Integer[]} queries
# @param {Integer} m
# @return {Integer[]}
def process_queries(queries, m)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $queries
* @param Integer $m
* @return Integer[]
*/
function processQueries($queries, $m) {

}
}
```

Dart Solution:

```
class Solution {
List<int> processQueries(List<int> queries, int m) {

}
}
```

Scala Solution:

```
object Solution {
def processQueries(queries: Array[Int], m: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec process_queries(queries :: [integer], m :: integer) :: [integer]
def process_queries(queries, m) do

end
end
```

Erlang Solution:

```
-spec process_queries(Queries :: [integer()], M :: integer()) -> [integer()].
process_queries(Queries, M) ->
.
```

Racket Solution:

```
(define/contract (process-queries queries m)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```