Problem 1882: Process Tasks Using Servers

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays

servers

and

tasks

of lengths

n

and

m

respectively.

servers[i]

is the

weight

of the

i

th

server, and

tasks[j]

is the

time needed

to process the

j

th

task

in seconds

.

Tasks are assigned to the servers using a

task queue

. Initially, all servers are free, and the queue is

empty

.

At second

j

, the

j

th

task is

inserted

into the queue (starting with the

0

th

task being inserted at second

0

). As long as there are free servers and the queue is not empty, the task in the front of the queue will be assigned to a free server with the

smallest weight

, and in case of a tie, it is assigned to a free server with the

smallest index

.

If there are no free servers and the queue is not empty, we wait until a server becomes free and immediately assign the next task. If multiple servers become free at the same time, then multiple tasks from the queue will be assigned

in order of insertion

following the weight and index priorities above.

A server that is assigned task

j

at second

t

will be free again at second

t + tasks[j]

.

Build an array

ans

of length

m

, where

ans[j]

is the

index

of the server the

j

th

task will be assigned to.

Return

the array

ans

.

Example 1:

Input:

servers = [3,3,2], tasks = [1,2,3,2,1,2]

Output:

[2,2,0,2,1,2]

Explanation:

Events in chronological order go as follows: - At second 0, task 0 is added and processed using server 2 until second 1. - At second 1, server 2 becomes free. Task 1 is added and processed using server 2 until second 3. - At second 2, task 2 is added and processed using server 0 until second 5. - At second 3, server 2 becomes free. Task 3 is added and processed using server 2 until second 5. - At second 4, task 4 is added and processed using server 1 until second 5. - At second 5, all servers become free. Task 5 is added and processed using server 2 until second 7.

Example 2:

Input:

servers = [5,1,4,3,2], tasks = [2,1,2,4,5,2,1]

Output:

[1,4,1,4,1,3,2]

Explanation:

Events in chronological order go as follows: - At second 0, task 0 is added and processed using server 1 until second 2. - At second 1, task 1 is added and processed using server 4 until second 2. - At second 2, servers 1 and 4 become free. Task 2 is added and processed using server 1 until second 4. - At second 3, task 3 is added and processed using server 4 until second 7. - At second 4, server 1 becomes free. Task 4 is added and processed using server 1 until second 9. - At second 5, task 5 is added and processed using server 3 until second 7. - At second 6, task 6 is added and processed using server 2 until second 7.

Constraints:

servers.length == n

tasks.length == m

1 <= n, m <= 2 * 10

5

1 <= servers[i], tasks[j] <= 2 * 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> assignTasks(vector<int>& servers, vector<int>& tasks) {

}
};
```

Java:

```
class Solution {
public int[] assignTasks(int[] servers, int[] tasks) {

}
```

```
}
```

Python3:

```
class Solution:
def assignTasks(self, servers: List[int], tasks: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def assignTasks(self, servers, tasks):
"""
:type servers: List[int]
:type tasks: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} servers
* @param {number[]} tasks
* @return {number[]}
*/
var assignTasks = function(servers, tasks) {

};
```

TypeScript:

```
function assignTasks(servers: number[], tasks: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] AssignTasks(int[] servers, int[] tasks) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* assignTasks(int* servers, int serversSize, int* tasks, int tasksSize,
int* returnSize) {

}
```

Go:

```
func assignTasks(servers []int, tasks []int) []int {

}
```

Kotlin:

```
class Solution {
fun assignTasks(servers: IntArray, tasks: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func assignTasks(_ servers: [Int], _ tasks: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn assign_tasks(servers: Vec<i32>, tasks: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} servers
# @param {Integer[]} tasks
# @return {Integer[]}
def assign_tasks(servers, tasks)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $servers
* @param Integer[] $tasks
* @return Integer[]
*/
function assignTasks($servers, $tasks) {

}
}
```

Dart:

```
class Solution {
List<int> assignTasks(List<int> servers, List<int> tasks) {

}
}
```

Scala:

```
object Solution {
def assignTasks(servers: Array[Int], tasks: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec assign_tasks(servers :: [integer], tasks :: [integer]) :: [integer]
def assign_tasks(servers, tasks) do
```

```
end
end
```

Erlang:

```
-spec assign_tasks(Servers :: [integer()], Tasks :: [integer()]) ->
[integer()].
assign_tasks(Servers, Tasks) ->
.
```

Racket:

```
(define/contract (assign-tasks servers tasks)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Process Tasks Using Servers
* Difficulty: Medium
* Tags: array, queue, heap
*
* 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
*/

class Solution {
public:
vector<int> assignTasks(vector<int>& servers, vector<int>& tasks) {

}
};
```

Java Solution:

```
/**
* Problem: Process Tasks Using Servers
* Difficulty: Medium
* Tags: array, queue, heap
*
* 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
*/

class Solution {
public int[] assignTasks(int[] servers, int[] tasks) {

}
}
```

Python3 Solution:

```
"""
Problem: Process Tasks Using Servers
Difficulty: Medium
Tags: array, queue, heap

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
"""

class Solution:
def assignTasks(self, servers: List[int], tasks: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def assignTasks(self, servers, tasks):
"""
:type servers: List[int]
:type tasks: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Process Tasks Using Servers
* Difficulty: Medium
* Tags: array, queue, heap
*
* 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 {number[]} servers
* @param {number[]} tasks
* @return {number[]}
*/
var assignTasks = function(servers, tasks) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Process Tasks Using Servers
* Difficulty: Medium
* Tags: array, queue, heap
*
* 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
*/

function assignTasks(servers: number[], tasks: number[]): number[] {

};
```

C# Solution:

```csharp
/*
* Problem: Process Tasks Using Servers
* Difficulty: Medium
```

```
* Tags: array, queue, heap
*
* 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
*/

public class Solution {
public int[] AssignTasks(int[] servers, int[] tasks) {

}
}
```

C Solution:

```
/*
* Problem: Process Tasks Using Servers
* Difficulty: Medium
* Tags: array, queue, heap
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* assignTasks(int* servers, int serversSize, int* tasks, int tasksSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Process Tasks Using Servers
// Difficulty: Medium
// Tags: array, queue, heap
//
// 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

func assignTasks(servers []int, tasks []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun assignTasks(servers: IntArray, tasks: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func assignTasks(_ servers: [Int], _ tasks: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Process Tasks Using Servers
// Difficulty: Medium
// Tags: array, queue, heap
//
// 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

impl Solution {
pub fn assign_tasks(servers: Vec<i32>, tasks: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} servers
# @param {Integer[]} tasks
# @return {Integer[]}
def assign_tasks(servers, tasks)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $servers
* @param Integer[] $tasks
* @return Integer[]
*/
function assignTasks($servers, $tasks) {

}
}
```

Dart Solution:

```
class Solution {
List<int> assignTasks(List<int> servers, List<int> tasks) {

}
}
```

Scala Solution:

```
object Solution {
def assignTasks(servers: Array[Int], tasks: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec assign_tasks(servers :: [integer], tasks :: [integer]) :: [integer]
def assign_tasks(servers, tasks) do
```

```
end
end
```

Erlang Solution:

```
-spec assign_tasks(Servers :: [integer()], Tasks :: [integer()]) ->
[integer()].
assign_tasks(Servers, Tasks) ->
.
```

Racket Solution:

```
(define/contract (assign-tasks servers tasks)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```