Problem 3476: Maximize Profit from Task Assignment

Problem Information

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

Problem Description

You are given an integer array

workers

, where

workers[i]

represents the skill level of the

i

th

worker. You are also given a 2D integer array

tasks

, where:

tasks[i][0]

represents the skill requirement needed to complete the task.

tasks[i][1]

represents the profit earned from completing the task.

Each worker can complete

at most

one task, and they can only take a task if their skill level is

equal

to the task's skill requirement. An

additional

worker joins today who can take up

any

task,

regardless

of the skill requirement.

Return the

maximum

total profit that can be earned by optimally assigning the tasks to the workers.

Example 1:

Input:

workers = [1,2,3,4,5], tasks = [[1,100],[2,400],[3,100],[3,400]]

Output:

1000

Explanation:

Worker 0 completes task 0.

Worker 1 completes task 1.

Worker 2 completes task 3.

The additional worker completes task 2.

Example 2:

Input:

workers = [10,10000,100000000], tasks = [[1,100]]

Output:

100

Explanation:

Since no worker matches the skill requirement, only the additional worker can complete task 0.

Example 3:

Input:

workers = [7], tasks = [[3,3],[3,3]]

Output:

3

Explanation:

The additional worker completes task 1. Worker 0 cannot work since no task has a skill requirement of 7.

Constraints:

1 <= workers.length <= 10

5

1 <= workers[i] <= 10

9

1 <= tasks.length <= 10

5

tasks[i].length == 2

1 <= tasks[i][0], tasks[i][1] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long maxProfit(vector<int>& workers, vector<vector<int>>& tasks) {

}
};
```

Java:

```
class Solution {
public long maxProfit(int[] workers, int[][] tasks) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxProfit(workers: number[], tasks: number[][]): number {

};
```

C#:

```
public class Solution {
public long MaxProfit(int[] workers, int[][] tasks) {

}
}
```

C:

```
long long maxProfit(int* workers, int workersSize, int** tasks, int
tasksSize, int* tasksColSize) {

}
```

Go:

```
func maxProfit(workers []int, tasks [][]int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxProfit(workers: IntArray, tasks: Array<IntArray>): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_profit(workers: Vec<i32>, tasks: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} workers
# @param {Integer[][]} tasks
# @return {Integer}
def max_profit(workers, tasks)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $workers
* @param Integer[][] $tasks
* @return Integer
*/
function maxProfit($workers, $tasks) {

}
}
```

Dart:

```
class Solution {
int maxProfit(List<int> workers, List<List<int>> tasks) {

}
}
```

Scala:

```
object Solution {
def maxProfit(workers: Array[Int], tasks: Array[Array[Int]]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_profit(workers :: [integer], tasks :: [[integer]]) :: integer
def max_profit(workers, tasks) do

end
end
```

Erlang:

```
-spec max_profit(Workers :: [integer()], Tasks :: [[integer()]]) ->
integer().
max_profit(Workers, Tasks) ->
.
```

Racket:

```
(define/contract (max-profit workers tasks)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Profit from Task Assignment
* Difficulty: Medium
* Tags: array, greedy, sort, 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:
long long maxProfit(vector<int>& workers, vector<vector<int>>& tasks) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Profit from Task Assignment
* Difficulty: Medium
* Tags: array, greedy, sort, 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 long maxProfit(int[] workers, int[][] tasks) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Profit from Task Assignment
Difficulty: Medium
Tags: array, greedy, sort, 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 maxProfit(self, workers: List[int], tasks: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```javascript
/**
* Problem: Maximize Profit from Task Assignment
* Difficulty: Medium
* Tags: array, greedy, sort, 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[]} workers
* @param {number[][]} tasks
* @return {number}
*/
var maxProfit = function(workers, tasks) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Maximize Profit from Task Assignment
* Difficulty: Medium
* Tags: array, greedy, sort, 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 maxProfit(workers: number[], tasks: number[][]): number {

};
```

C# Solution:

```csharp
/*
* Problem: Maximize Profit from Task Assignment
* Difficulty: Medium
* Tags: array, greedy, sort, 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 long MaxProfit(int[] workers, int[][] tasks) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Profit from Task Assignment
* Difficulty: Medium
* Tags: array, greedy, sort, 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
*/

long long maxProfit(int* workers, int workersSize, int** tasks, int
tasksSize, int* tasksColSize) {

}
```

Go Solution:

```
// Problem: Maximize Profit from Task Assignment
// Difficulty: Medium
// Tags: array, greedy, sort, 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 maxProfit(workers []int, tasks [][]int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxProfit(workers: IntArray, tasks: Array<IntArray>): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Profit from Task Assignment
// Difficulty: Medium
// Tags: array, greedy, sort, 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 max_profit(workers: Vec<i32>, tasks: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} workers
# @param {Integer[][]} tasks
# @return {Integer}
def max_profit(workers, tasks)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $workers
* @param Integer[][] $tasks
* @return Integer
*/
function maxProfit($workers, $tasks) {

}
}
```

Dart Solution:

```
class Solution {
int maxProfit(List<int> workers, List<List<int>> tasks) {

}
}
```

Scala Solution:

```
object Solution {
def maxProfit(workers: Array[Int], tasks: Array[Array[Int]]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_profit(workers :: [integer], tasks :: [[integer]]) :: integer
def max_profit(workers, tasks) do

end
end
```

Erlang Solution:

```
-spec max_profit(Workers :: [integer()], Tasks :: [[integer()]]) ->
integer().
max_profit(Workers, Tasks) ->
.
```

Racket Solution:

```
(define/contract (max-profit workers tasks)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```