Problem 621: Task Scheduler

Problem Information

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

Problem Description

You are given an array of CPU

tasks

, each labeled with a letter from A to Z, and a number

n

. Each CPU interval can be idle or allow the completion of one task. Tasks can be completed in any order, but there's a constraint: there has to be a gap of

at least

n

intervals between two tasks with the same label.

Return the

minimum

number of CPU intervals required to complete all tasks.

Example 1:

Input:

tasks = ["A","A","A","B","B","B"], n = 2

Output:

8

Explanation:

A possible sequence is: A -> B -> idle -> A -> B -> idle -> A -> B.

After completing task A, you must wait two intervals before doing A again. The same applies to task B. In the 3

rd

interval, neither A nor B can be done, so you idle. By the 4

th

interval, you can do A again as 2 intervals have passed.

Example 2:

Input:

tasks = ["A","C","A","B","D","B"], n = 1

Output:

6

Explanation:

A possible sequence is: A -> B -> C -> D -> A -> B.

With a cooling interval of 1, you can repeat a task after just one other task.

Example 3:

Input:

tasks = ["A","A","A", "B","B","B"], n = 3

Output:

10

Explanation:

A possible sequence is: A -> B -> idle -> idle -> A -> B -> idle -> idle -> A -> B.

There are only two types of tasks, A and B, which need to be separated by 3 intervals. This leads to idling twice between repetitions of these tasks.

Constraints:

1 <= tasks.length <= 10

4

tasks[i]

is an uppercase English letter.

0 <= n <= 100

Code Snippets

C++:

```
class Solution {
public:
int leastInterval(vector<char>& tasks, int n) {

}
};
```

Java:

```
class Solution {
public int leastInterval(char[] tasks, int n) {

}
}
```

Python3:

```
class Solution:
def leastInterval(self, tasks: List[str], n: int) -> int:
```

Python:

```
class Solution(object):
def leastInterval(self, tasks, n):
"""
:type tasks: List[str]
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {character[]} tasks
* @param {number} n
* @return {number}
*/
var leastInterval = function(tasks, n) {

};
```

TypeScript:

```
function leastInterval(tasks: string[], n: number): number {

};
```

C#:

```
public class Solution {
public int LeastInterval(char[] tasks, int n) {

}
}
```

C:

```
int leastInterval(char* tasks, int tasksSize, int n) {

}
```

Go:

```
func leastInterval(tasks []byte, n int) int {

}
```

Kotlin:

```
class Solution {
fun leastInterval(tasks: CharArray, n: Int): Int {

}
}
```

Swift:

```
class Solution {
func leastInterval(_ tasks: [Character], _ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn least_interval(tasks: Vec<char>, n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Character[]} tasks
# @param {Integer} n
# @return {Integer}
def least_interval(tasks, n)

end
```

PHP:

```
class Solution {

/**
* @param String[] $tasks
* @param Integer $n
* @return Integer
*/
function leastInterval($tasks, $n) {

}
}
```

Dart:

```
class Solution {
int leastInterval(List<String> tasks, int n) {

}
}
```

Scala:

```
object Solution {
def leastInterval(tasks: Array[Char], n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec least_interval(tasks :: [char], n :: integer) :: integer
def least_interval(tasks, n) do
```

```
end
end
```

Erlang:

```
-spec least_interval(Tasks :: [char()], N :: integer()) -> integer().
least_interval(Tasks, N) ->
.
```

Racket:

```
(define/contract (least-interval tasks n)
(-> (listof char?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Task Scheduler
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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:
int leastInterval(vector<char>& tasks, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Task Scheduler
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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 int leastInterval(char[] tasks, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Task Scheduler
Difficulty: Medium
Tags: array, greedy, hash, sort, 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 leastInterval(self, tasks: List[str], n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def leastInterval(self, tasks, n):
"""
:type tasks: List[str]
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Task Scheduler
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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 {character[]} tasks
* @param {number} n
* @return {number}
*/
var leastInterval = function(tasks, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Task Scheduler
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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 leastInterval(tasks: string[], n: number): number {

};
```

C# Solution:

```
/*
* Problem: Task Scheduler
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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 int LeastInterval(char[] tasks, int n) {

}
}
```

C Solution:

```
/*
* Problem: Task Scheduler
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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
*/

int leastInterval(char* tasks, int tasksSize, int n) {

}
```

Go Solution:

```
// Problem: Task Scheduler
// Difficulty: Medium
// Tags: array, greedy, hash, sort, 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 leastInterval(tasks []byte, n int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun leastInterval(tasks: CharArray, n: Int): Int {

}
}
```

Swift Solution:

```swift
class Solution {
func leastInterval(_ tasks: [Character], _ n: Int) -> Int {

}
}
```

Rust Solution:

```rust
// Problem: Task Scheduler
// Difficulty: Medium
// Tags: array, greedy, hash, sort, 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 least_interval(tasks: Vec<char>, n: i32) -> i32 {

}
}
```

Ruby Solution:

```ruby
# @param {Character[]} tasks
# @param {Integer} n
# @return {Integer}
def least_interval(tasks, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $tasks
* @param Integer $n
* @return Integer
*/
function leastInterval($tasks, $n) {

}
}
```

Dart Solution:

```
class Solution {
int leastInterval(List<String> tasks, int n) {

}
}
```

Scala Solution:

```
object Solution {
def leastInterval(tasks: Array[Char], n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec least_interval(tasks :: [char], n :: integer) :: integer
def least_interval(tasks, n) do

end
end
```

Erlang Solution:

```
-spec least_interval(Tasks :: [char()], N :: integer()) -> integer().
least_interval(Tasks, N) ->
.
```

Racket Solution:

```
(define/contract (least-interval tasks n)
(-> (listof char?) exact-integer? exact-integer?)
)
```