Problem 2188: Minimum Time to Finish the Race

Problem Information

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

Problem Description

You are given a

0-indexed

2D integer array

tires

where

tires[i] = [f

i

, r

i

]

indicates that the

i

th

tire can finish its

x

th

successive lap in

f

i

* r

i

(x-1)

seconds.

For example, if

f

i

= 3

and

r

i

= 2

, then the tire would finish its

1

st

lap in

3

seconds, its

2

nd

lap in

3 * 2 = 6

seconds, its

3

rd

lap in

3 * 2

2

= 12

seconds, etc.

You are also given an integer

changeTime

and an integer

numLaps

.

The race consists of

numLaps

laps and you may start the race with

any

tire. You have an

unlimited

supply of each tire and after every lap, you may

change

to any given tire (including the current tire type) if you wait

changeTime

seconds.

Return

the

minimum

time to finish the race.

Example 1:

Input:

tires = [[2,3],[3,4]], changeTime = 5, numLaps = 4

Output:

21

Explanation:

Lap 1: Start with tire 0 and finish the lap in 2 seconds. Lap 2: Continue with tire 0 and finish the lap in 2 * 3 = 6 seconds. Lap 3: Change tires to a new tire 0 for 5 seconds and then finish the lap in another 2 seconds. Lap 4: Continue with tire 0 and finish the lap in 2 * 3 = 6 seconds. Total time = 2 + 6 + 5 + 2 + 6 = 21 seconds. The minimum time to complete the race is 21 seconds.

Example 2:

Input:

tires = [[1,10],[2,2],[3,4]], changeTime = 6, numLaps = 5

Output:

25

Explanation:

Lap 1: Start with tire 1 and finish the lap in 2 seconds. Lap 2: Continue with tire 1 and finish the lap in 2 * 2 = 4 seconds. Lap 3: Change tires to a new tire 1 for 6 seconds and then finish the lap in another 2 seconds. Lap 4: Continue with tire 1 and finish the lap in 2 * 2 = 4 seconds. Lap 5: Change tires to tire 0 for 6 seconds then finish the lap in another 1 second. Total time = 2 + 4 + 6 + 2 + 4 + 6 + 1 = 25 seconds. The minimum time to complete the race is 25 seconds.

Constraints:

1 <= tires.length <= 10

5

tires[i].length == 2

1 <= f

i

, changeTime <= 10

5

2 <= r

i

<= 10

5

1 <= numLaps <= 1000

Code Snippets

C++:

```
class Solution {
public:
int minimumFinishTime(vector<vector<int>>& tires, int changeTime, int
numLaps) {

}
};
```

Java:

```
class Solution {
public int minimumFinishTime(int[][] tires, int changeTime, int numLaps) {

}
}
```

Python3:

```
class Solution:
def minimumFinishTime(self, tires: List[List[int]], changeTime: int, numLaps:
int) -> int:
```

Python:

```
class Solution(object):
def minimumFinishTime(self, tires, changeTime, numLaps):
"""
:type tires: List[List[int]]
:type changeTime: int
:type numLaps: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} tires
* @param {number} changeTime
* @param {number} numLaps
* @return {number}
*/
var minimumFinishTime = function(tires, changeTime, numLaps) {

};
```

TypeScript:

```
function minimumFinishTime(tires: number[][], changeTime: number, numLaps:
number): number {

};
```

C#:

```
public class Solution {
public int MinimumFinishTime(int[][] tires, int changeTime, int numLaps) {

}
}
```

C:

```
int minimumFinishTime(int** tires, int tiresSize, int* tiresColSize, int
changeTime, int numLaps) {

}
```

Go:

```
func minimumFinishTime(tires [][]int, changeTime int, numLaps int) int {

}
```

Kotlin:

```
class Solution {
fun minimumFinishTime(tires: Array<IntArray>, changeTime: Int, numLaps: Int):
Int {

}
}
```

Swift:

```
class Solution {
func minimumFinishTime(_ tires: [[Int]], _ changeTime: Int, _ numLaps: Int)
-> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_finish_time(tires: Vec<Vec<i32>>, change_time: i32, num_laps:
i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} tires
# @param {Integer} change_time
# @param {Integer} num_laps
# @return {Integer}
def minimum_finish_time(tires, change_time, num_laps)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $tires
* @param Integer $changeTime
* @param Integer $numLaps
* @return Integer
*/
function minimumFinishTime($tires, $changeTime, $numLaps) {

}
}
```

Dart:

```
class Solution {
int minimumFinishTime(List<List<int>> tires, int changeTime, int numLaps) {

}
}
```

Scala:

```
object Solution {
def minimumFinishTime(tires: Array[Array[Int]], changeTime: Int, numLaps:
Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_finish_time(tires :: [[integer]], change_time :: integer,
num_laps :: integer) :: integer
```

```
def minimum_finish_time(tires, change_time, num_laps) do

end
end
```

Erlang:

```
-spec minimum_finish_time(Tires :: [[integer()]], ChangeTime :: integer(),
NumLaps :: integer()) -> integer().
minimum_finish_time(Tires, ChangeTime, NumLaps) ->
.
```

Racket:

```
(define/contract (minimum-finish-time tires changeTime numLaps)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Finish the Race
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minimumFinishTime(vector<vector<int>>& tires, int changeTime, int
numLaps) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Finish the Race
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimumFinishTime(int[][] tires, int changeTime, int numLaps) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Finish the Race
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimumFinishTime(self, tires: List[List[int]], changeTime: int, numLaps:
int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumFinishTime(self, tires, changeTime, numLaps):
"""
:type tires: List[List[int]]
```

```
:type changeTime: int
:type numLaps: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Finish the Race
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} tires
* @param {number} changeTime
* @param {number} numLaps
* @return {number}
*/
var minimumFinishTime = function(tires, changeTime, numLaps) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Finish the Race
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumFinishTime(tires: number[][], changeTime: number, numLaps:
number): number {
};
```

C# Solution:

```
/*
* Problem: Minimum Time to Finish the Race
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinimumFinishTime(int[][] tires, int changeTime, int numLaps) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Finish the Race
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumFinishTime(int** tires, int tiresSize, int* tiresColSize, int
changeTime, int numLaps) {

}
```

Go Solution:

```
// Problem: Minimum Time to Finish the Race
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumFinishTime(tires [][]int, changeTime int, numLaps int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumFinishTime(tires: Array<IntArray>, changeTime: Int, numLaps: Int):
Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumFinishTime(_ tires: [[Int]], _ changeTime: Int, _ numLaps: Int)
-> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Finish the Race
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_finish_time(tires: Vec<Vec<i32>>, change_time: i32, num_laps:
```

```
i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} tires
# @param {Integer} change_time
# @param {Integer} num_laps
# @return {Integer}
def minimum_finish_time(tires, change_time, num_laps)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $tires
* @param Integer $changeTime
* @param Integer $numLaps
* @return Integer
*/
function minimumFinishTime($tires, $changeTime, $numLaps) {

}
}
```

Dart Solution:

```
class Solution {
int minimumFinishTime(List<List<int>> tires, int changeTime, int numLaps) {

}
}
```

Scala Solution:

```
object Solution {
def minimumFinishTime(tires: Array[Array[Int]], changeTime: Int, numLaps:
```

```
Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_finish_time(tires :: [[integer]], change_time :: integer,
num_laps :: integer) :: integer
def minimum_finish_time(tires, change_time, num_laps) do

end
end
```

Erlang Solution:

```
-spec minimum_finish_time(Tires :: [[integer()]], ChangeTime :: integer(),
NumLaps :: integer()) -> integer().
minimum_finish_time(Tires, ChangeTime, NumLaps) ->
.
```

Racket Solution:

```
(define/contract (minimum-finish-time tires changeTime numLaps)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer?)
)
```