Problem 3279: Maximum Total Area Occupied by Pistons

Problem Information

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

Problem Description

There are several pistons in an old car engine, and we want to calculate the

maximum

possible area

under

the pistons.

You are given:

An integer

height

, representing the

maximum

height a piston can reach.

An integer array

positions

, where

positions[i]

is the current position of piston

i

, which is equal to the current area

under

it.

A string

directions

, where

directions[i]

is the current moving direction of piston

i

,

'U'

for up, and

'D'

for down.

Each second:

Every piston moves in its current direction 1 unit. e.g., if the direction is up,

positions[i]

is incremented by 1.

If a piston has reached one of the ends, i.e.,

positions[i] == 0

or

positions[i] == height

, its direction will change.

Return the

maximum possible area

under all the pistons.

Example 1:

Input:

height = 5, positions = [2,5], directions = "UD"

Output:

7

Explanation:

The current position of the pistons has the maximum possible area under it.

Example 2:

Input:

height = 6, positions = [0,0,6,3], directions = "UUDU"

Output:

15

Explanation:

After 3 seconds, the pistons will be in positions

[3, 3, 3, 6]

, which has the maximum possible area under it.

Constraints:

1 <= height <= 10

6

1 <= positions.length == directions.length <= 10

5

0 <= positions[i] <= height

directions[i]

is either

'U'

or

'D'

.

Code Snippets

C++:

```
class Solution {
public:
long long maxArea(int height, vector<int>& positions, string directions) {

}
};
```

Java:

```
class Solution {
public long maxArea(int height, int[] positions, String directions) {

}
}
```

Python3:

```
class Solution:
def maxArea(self, height: int, positions: List[int], directions: str) -> int:
```

Python:

```
class Solution(object):
def maxArea(self, height, positions, directions):
"""
:type height: int
:type positions: List[int]
:type directions: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} height
* @param {number[]} positions
* @param {string} directions
* @return {number}
```

```
*/
var maxArea = function(height, positions, directions) {

};
```

TypeScript:

```
function maxArea(height: number, positions: number[], directions: string):
number {

};
```

C#:

```
public class Solution {
public long MaxArea(int height, int[] positions, string directions) {

}
}
```

C:

```
long long maxArea(int height, int* positions, int positionsSize, char*
directions) {

}
```

Go:

```
func maxArea(height int, positions []int, directions string) int64 {

}
```

Kotlin:

```
class Solution {
fun maxArea(height: Int, positions: IntArray, directions: String): Long {

}
}
```

Swift:

```
class Solution {
func maxArea(_ height: Int, _ positions: [Int], _ directions: String) -> Int
{

}
}
```

Rust:

```
impl Solution {
pub fn max_area(height: i32, positions: Vec<i32>, directions: String) -> i64
{

}
}
```

Ruby:

```
# @param {Integer} height
# @param {Integer[]} positions
# @param {String} directions
# @return {Integer}
def max_area(height, positions, directions)

end
```

PHP:

```
class Solution {

/**
* @param Integer $height
* @param Integer[] $positions
* @param String $directions
* @return Integer
*/
function maxArea($height, $positions, $directions) {

}
}
```

Dart:

```
class Solution {
int maxArea(int height, List<int> positions, String directions) {

}
}
```

Scala:

```
object Solution {
def maxArea(height: Int, positions: Array[Int], directions: String): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_area(height :: integer, positions :: [integer], directions ::
String.t) :: integer
def max_area(height, positions, directions) do

end
end
```

Erlang:

```
-spec max_area(Height :: integer(), Positions :: [integer()], Directions ::
unicode:unicode_binary()) -> integer().
max_area(Height, Positions, Directions) ->
.
```

Racket:

```
(define/contract (max-area height positions directions)
(-> exact-integer? (listof exact-integer?) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Total Area Occupied by Pistons
* Difficulty: Hard
* Tags: array, string, hash
*
* 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:
long long maxArea(int height, vector<int>& positions, string directions) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Total Area Occupied by Pistons
* Difficulty: Hard
* Tags: array, string, hash
*
* 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 long maxArea(int height, int[] positions, String directions) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Total Area Occupied by Pistons
Difficulty: Hard
Tags: array, string, hash
```

```
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 maxArea(self, height: int, positions: List[int], directions: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxArea(self, height, positions, directions):
"""
:type height: int
:type positions: List[int]
:type directions: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Total Area Occupied by Pistons
* Difficulty: Hard
* Tags: array, string, hash
*
* 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 {number} height
* @param {number[]} positions
* @param {string} directions
* @return {number}
*/
var maxArea = function(height, positions, directions) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Total Area Occupied by Pistons
* Difficulty: Hard
* Tags: array, string, hash
*
* 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 maxArea(height: number, positions: number[], directions: string):
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Total Area Occupied by Pistons
* Difficulty: Hard
* Tags: array, string, hash
*
* 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 long MaxArea(int height, int[] positions, string directions) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Total Area Occupied by Pistons
```

```
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

long long maxArea(int height, int* positions, int positionsSize, char*
directions) {

}
```

Go Solution:

```
// Problem: Maximum Total Area Occupied by Pistons
// Difficulty: Hard
// Tags: array, string, hash
//
// 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 maxArea(height int, positions []int, directions string) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxArea(height: Int, positions: IntArray, directions: String): Long {

}
}
```

Swift Solution:

```
class Solution {
func maxArea(_ height: Int, _ positions: [Int], _ directions: String) -> Int
{
```

```
}
}
```

Rust Solution:

```
// Problem: Maximum Total Area Occupied by Pistons
// Difficulty: Hard
// Tags: array, string, hash
//
// 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 max_area(height: i32, positions: Vec<i32>, directions: String) -> i64
{

}
}
```

Ruby Solution:

```
# @param {Integer} height
# @param {Integer[]} positions
# @param {String} directions
# @return {Integer}
def max_area(height, positions, directions)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $height
* @param Integer[] $positions
* @param String $directions
* @return Integer
*/
function maxArea($height, $positions, $directions) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int maxArea(int height, List<int> positions, String directions) {

}
}
```

Scala Solution:

```
object Solution {
def maxArea(height: Int, positions: Array[Int], directions: String): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_area(height :: integer, positions :: [integer], directions ::
String.t) :: integer
def max_area(height, positions, directions) do

end
end
```

Erlang Solution:

```
-spec max_area(Height :: integer(), Positions :: [integer()], Directions ::
unicode:unicode_binary()) -> integer().
max_area(Height, Positions, Directions) ->
.
```

Racket Solution:

```
(define/contract (max-area height positions directions)
(-> exact-integer? (listof exact-integer?) string? exact-integer?)
```

```
)
```