Problem 3522: Calculate Score After Performing Instructions

Problem Information

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

Problem Description

You are given two arrays,

instructions

and

values

, both of size

n

.

You need to simulate a process based on the following rules:

You start at the first instruction at index

i = 0

with an initial score of 0.

If

instructions[i]

is

"add"

:

Add

values[i]

to your score.

Move to the next instruction

(i + 1)

.

If

instructions[i]

is

"jump"

:

Move to the instruction at index

(i + values[i])

without modifying your score.

The process ends when you either:

Go out of bounds (i.e.,

i < 0 or i >= n

), or

Attempt to revisit an instruction that has been previously executed. The revisited instruction is not executed.

Return your score at the end of the process.

Example 1:

Input:

instructions = ["jump","add","add","jump","add","jump"], values = [2,1,3,1,-2,-3]

Output:

1

Explanation:

Simulate the process starting at instruction 0:

At index 0: Instruction is

"jump"

, move to index

0 + 2 = 2

.

At index 2: Instruction is

"add"

, add

values[2] = 3

to your score and move to index 3. Your score becomes 3.

At index 3: Instruction is

"jump"

, move to index

3 + 1 = 4

.

At index 4: Instruction is

"add"

, add

values[4] = -2

to your score and move to index 5. Your score becomes 1.

At index 5: Instruction is

"jump"

, move to index

5 + (-3) = 2

.

At index 2: Already visited. The process ends.

Example 2:

Input:

instructions = ["jump","add","add"], values = [3,1,1]

Output:

0

Explanation:

Simulate the process starting at instruction 0:

At index 0: Instruction is

"jump"

, move to index

0 + 3 = 3

.

At index 3: Out of bounds. The process ends.

Example 3:

Input:

instructions = ["jump"], values = [0]

Output:

0

Explanation:

Simulate the process starting at instruction 0:

At index 0: Instruction is

"jump"

, move to index

0 + 0 = 0

.

At index 0: Already visited. The process ends.

Constraints:

n == instructions.length == values.length

1 <= n <= 10

5

instructions[i]

is either

"add"

or

"jump"

.

-10

5

<= values[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long calculateScore(vector<string>& instructions, vector<int>& values) {

}
};
```

Java:

```
class Solution {
public long calculateScore(String[] instructions, int[] values) {

}
}
```

Python3:

```
class Solution:
def calculateScore(self, instructions: List[str], values: List[int]) -> int:
```

Python:

```
class Solution(object):
def calculateScore(self, instructions, values):
"""
:type instructions: List[str]
:type values: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} instructions
* @param {number[]} values
* @return {number}
*/
var calculateScore = function(instructions, values) {
```

```
};
```

TypeScript:

```
function calculateScore(instructions: string[], values: number[]): number {

};
```

C#:

```
public class Solution {
public long CalculateScore(string[] instructions, int[] values) {

}
}
```

C:

```
long long calculateScore(char** instructions, int instructionsSize, int*
values, int valuesSize) {

}
```

Go:

```
func calculateScore(instructions []string, values []int) int64 {

}
```

Kotlin:

```
class Solution {
fun calculateScore(instructions: Array<String>, values: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func calculateScore(_ instructions: [String], _ values: [Int]) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn calculate_score(instructions: Vec<String>, values: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {String[]} instructions
# @param {Integer[]} values
# @return {Integer}
def calculate_score(instructions, values)

end
```

PHP:

```
class Solution {

/**
* @param String[] $instructions
* @param Integer[] $values
* @return Integer
*/
function calculateScore($instructions, $values) {

}
}
```

Dart:

```
class Solution {
int calculateScore(List<String> instructions, List<int> values) {

}
}
```

Scala:

```
object Solution {
def calculateScore(instructions: Array[String], values: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec calculate_score(instructions :: [String.t], values :: [integer]) ::
integer
def calculate_score(instructions, values) do

end
end
```

Erlang:

```
-spec calculate_score(Instructions :: [unicode:unicode_binary()], Values ::
[integer()]) -> integer().
calculate_score(Instructions, Values) ->
.
```

Racket:

```
(define/contract (calculate-score instructions values)
(-> (listof string?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Calculate Score After Performing Instructions
* Difficulty: Medium
* 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 calculateScore(vector<string>& instructions, vector<int>& values) {

}
};
```

Java Solution:

```
/**
* Problem: Calculate Score After Performing Instructions
* Difficulty: Medium
* 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 calculateScore(String[] instructions, int[] values) {

}
}
```

Python3 Solution:

```
"""
Problem: Calculate Score After Performing Instructions
Difficulty: Medium
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 calculateScore(self, instructions: List[str], values: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def calculateScore(self, instructions, values):
"""
:type instructions: List[str]
:type values: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Calculate Score After Performing Instructions
* Difficulty: Medium
* 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 {string[]} instructions
* @param {number[]} values
* @return {number}
*/
var calculateScore = function(instructions, values) {

};
```

TypeScript Solution:

```
/**
* Problem: Calculate Score After Performing Instructions
* Difficulty: Medium
* 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 calculateScore(instructions: string[], values: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Calculate Score After Performing Instructions
* Difficulty: Medium
* 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 CalculateScore(string[] instructions, int[] values) {

}
}
```

C Solution:

```
/*
* Problem: Calculate Score After Performing Instructions
* Difficulty: Medium
* 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 calculateScore(char** instructions, int instructionsSize, int*
```

```
values, int valuesSize) {

}
```

Go Solution:

```
// Problem: Calculate Score After Performing Instructions
// Difficulty: Medium
// 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 calculateScore(instructions []string, values []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun calculateScore(instructions: Array<String>, values: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func calculateScore(_ instructions: [String], _ values: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Calculate Score After Performing Instructions
// Difficulty: Medium
// 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 calculate_score(instructions: Vec<String>, values: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {String[]} instructions
# @param {Integer[]} values
# @return {Integer}
def calculate_score(instructions, values)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $instructions
* @param Integer[] $values
* @return Integer
*/
function calculateScore($instructions, $values) {

}
}
```

Dart Solution:

```
class Solution {
int calculateScore(List<String> instructions, List<int> values) {

}
}
```

Scala Solution:

```
object Solution {
def calculateScore(instructions: Array[String], values: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec calculate_score(instructions :: [String.t], values :: [integer]) ::
integer
def calculate_score(instructions, values) do

end
end
```

Erlang Solution:

```
-spec calculate_score(Instructions :: [unicode:unicode_binary()], Values ::
[integer()]) -> integer().
calculate_score(Instructions, Values) ->
.
```

Racket Solution:

```
(define/contract (calculate-score instructions values)
(-> (listof string?) (listof exact-integer?) exact-integer?)
)
```