Problem 2961: Double Modular Exponentiation

Problem Information

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

Problem Description

You are given a

0-indexed

2D array

variables

where

variables[i] = [a

i

, b

i

, c

i,

m

i

]

, and an integer

target

.

An index

i

is

good

if the following formula holds:

0 <= i < variables.length

((a

i

b

i

% 10)

c

i

) % m

i

== target

Return

an array consisting of

good

indices in

any order

.

Example 1:

Input:

variables = [[2,3,3,10],[3,3,3,1],[6,1,1,4]], target = 2

Output:

[0,2]

Explanation:

For each index i in the variables array: 1) For the index 0, variables[0] = [2,3,3,10], (2

3

% 10)

3

% 10 = 2. 2) For the index 1, variables[1] = [3,3,3,1], (3

3

% 10)

3

% 1 = 0. 3) For the index 2, variables[2] = [6,1,1,4], (6

1

% 10)

1

% 4 = 2. Therefore we return [0,2] as the answer.

Example 2:

Input:

variables = [[39,3,1000,1000]], target = 17

Output:

[]

Explanation:

For each index i in the variables array: 1) For the index 0, variables[0] = [39,3,1000,1000], (39

3

% 10)

1000

% 1000 = 1. Therefore we return [] as the answer.

Constraints:

1 <= variables.length <= 100

variables[i] == [a

i

, b

i

, c

i

, m

i

]

1 <= a

i

, b

i

, c

i

, m

i

<= 10

3

0 <= target <= 10

Code Snippets

C++:

```
class Solution {
public:
vector<int> getGoodIndices(vector<vector<int>>& variables, int target) {

}
};
```

Java:

```
class Solution {
public List<Integer> getGoodIndices(int[][] variables, int target) {

}
}
```

Python3:

```
class Solution:
def getGoodIndices(self, variables: List[List[int]], target: int) ->
List[int]:
```

Python:

```
class Solution(object):
def getGoodIndices(self, variables, target):
"""
:type variables: List[List[int]]
:type target: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} variables
```

```
 * @param {number} target
 * @return {number[]}
 */
var getGoodIndices = function(variables, target) {

};
```

TypeScript:

```
function getGoodIndices(variables: number[][], target: number): number[] {

};
```

C#:

```
public class Solution {
public IList<int> GetGoodIndices(int[][] variables, int target) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getGoodIndices(int** variables, int variablesSize, int*
variablesColSize, int target, int* returnSize) {

}
```

Go:

```
func getGoodIndices(variables [][]int, target int) []int {

}
```

Kotlin:

```
class Solution {
fun getGoodIndices(variables: Array<IntArray>, target: Int): List<Int> {
```

```
}
}
```

Swift:

```
class Solution {
func getGoodIndices(_ variables: [[Int]], _ target: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn get_good_indices(variables: Vec<Vec<i32>>, target: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} variables
# @param {Integer} target
# @return {Integer[]}
def get_good_indices(variables, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $variables
* @param Integer $target
* @return Integer[]
*/
function getGoodIndices($variables, $target) {

}
}
```

Dart:

```
class Solution {
List<int> getGoodIndices(List<List<int>> variables, int target) {

}
}
```

Scala:

```
object Solution {
def getGoodIndices(variables: Array[Array[Int]], target: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_good_indices(variables :: [[integer]], target :: integer) ::
[integer]
def get_good_indices(variables, target) do

end
end
```

Erlang:

```
-spec get_good_indices(Variables :: [[integer()]], Target :: integer()) ->
[integer()].
get_good_indices(Variables, Target) ->
.
```

Racket:

```
(define/contract (get-good-indices variables target)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Double Modular Exponentiation
* Difficulty: Medium
* Tags: array, math
*
* 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:
vector<int> getGoodIndices(vector<vector<int>>& variables, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Double Modular Exponentiation
* Difficulty: Medium
* Tags: array, math
*
* 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 List<Integer> getGoodIndices(int[][] variables, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Double Modular Exponentiation
Difficulty: Medium
Tags: array, math
```

```
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 getGoodIndices(self, variables: List[List[int]], target: int) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getGoodIndices(self, variables, target):
"""
:type variables: List[List[int]]
:type target: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Double Modular Exponentiation
* Difficulty: Medium
* Tags: array, math
*
* 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[][]} variables
* @param {number} target
* @return {number[]}
*/
var getGoodIndices = function(variables, target) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Double Modular Exponentiation
* Difficulty: Medium
* Tags: array, math
*
* 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 getGoodIndices(variables: number[][], target: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Double Modular Exponentiation
* Difficulty: Medium
* Tags: array, math
*
* 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 IList<int> GetGoodIndices(int[][] variables, int target) {

}
}
```

C Solution:

```
/*
* Problem: Double Modular Exponentiation
* Difficulty: Medium
```

```
* Tags: array, math
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getGoodIndices(int** variables, int variablesSize, int*
variablesColSize, int target, int* returnSize) {

}
```

Go Solution:

```
// Problem: Double Modular Exponentiation
// Difficulty: Medium
// Tags: array, math
//
// 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 getGoodIndices(variables [][]int, target int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun getGoodIndices(variables: Array<IntArray>, target: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func getGoodIndices(_ variables: [[Int]], _ target: Int) -> [Int] {
```

```
}
}
```

Rust Solution:

```
// Problem: Double Modular Exponentiation
// Difficulty: Medium
// Tags: array, math
//
// 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 get_good_indices(variables: Vec<Vec<i32>>, target: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} variables
# @param {Integer} target
# @return {Integer[]}
def get_good_indices(variables, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $variables
* @param Integer $target
* @return Integer[]
*/
function getGoodIndices($variables, $target) {

}
```

```
}
```

Dart Solution:

```
class Solution {
List<int> getGoodIndices(List<List<int>> variables, int target) {

}
}
```

Scala Solution:

```
object Solution {
def getGoodIndices(variables: Array[Array[Int]], target: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_good_indices(variables :: [[integer]], target :: integer) ::
[integer]
def get_good_indices(variables, target) do

end
end
```

Erlang Solution:

```
-spec get_good_indices(Variables :: [[integer()]], Target :: integer()) ->
[integer()].
get_good_indices(Variables, Target) ->
.
```

Racket Solution:

```
(define/contract (get-good-indices variables target)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```