Problem 3145: Find Products of Elements of Big Array

Problem Information

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

Problem Description

The

powerful array

of a non-negative integer

x

is defined as the shortest sorted array of powers of two that sum up to

x

. The table below illustrates examples of how the

powerful array

is determined. It can be proven that the powerful array of

x

is unique.

num

Binary Representation

powerful array

1

0000

1

[1]

8

0

1

000

[8]

10

0

1

0

1

0

[2, 8]

13

0

11

0

1

[1, 4, 8]

23

1

0

111

[1, 2, 4, 16]

The array

big_nums

is created by concatenating the

powerful arrays

for every positive integer

i

in ascending order: 1, 2, 3, and so on. Thus,

big_nums

begins as

[

1

,

2

,

1, 2

,

4

,

1, 4

,

2, 4

,

1, 2, 4

,

8

, ...]

.

You are given a 2D integer matrix

queries

, where for

queries[i] = [from

i

, to

i

, mod

i

]

you should calculate

(big_nums[from

i

] * big_nums[from

i

+ 1] * ... * big_nums[to

i

]) % mod

i

.

Return an integer array

answer

such that

answer[i]

is the answer to the

i

th

query.

Example 1:

Input:

queries = [[1,3,7]]

Output:

[4]

Explanation:

There is one query.

big_nums[1..3] = [2,1,2]

. The product of them is 4. The result is

4 % 7 = 4.

Example 2:

Input:

queries = [[2,5,3],[7,7,4]]

Output:

[2,2]

Explanation:

There are two queries.

First query:

big_nums[2..5] = [1,2,4,1]

. The product of them is 8. The result is

8 % 3 = 2

.

Second query:

big_nums[7] = 2

. The result is

2 % 4 = 2

.

Constraints:

1 <= queries.length <= 500

queries[i].length == 3

0 <= queries[i][0] <= queries[i][1] <= 10

15

1 <= queries[i][2] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> findProductsOfElements(vector<vector<long long>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] findProductsOfElements(long[][] queries) {

}
}
```

Python3:

```
class Solution:
def findProductsOfElements(self, queries: List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def findProductsOfElements(self, queries):
"""
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} queries
* @return {number[]}
*/
var findProductsOfElements = function(queries) {
};
```

TypeScript:

```
function findProductsOfElements(queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] FindProductsOfElements(long[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findProductsOfElements(long long** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go:

```
func findProductsOfElements(queries [][]int64) []int {

}
```

Kotlin:

```
class Solution {
fun findProductsOfElements(queries: Array<LongArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func findProductsOfElements(_ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_products_of_elements(queries: Vec<Vec<i64>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} queries
# @return {Integer[]}
def find_products_of_elements(queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $queries
* @return Integer[]
*/
function findProductsOfElements($queries) {

}
}
```

Dart:

```
class Solution {
List<int> findProductsOfElements(List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def findProductsOfElements(queries: Array[Array[Long]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_products_of_elements(queries :: [[integer]]) :: [integer]
def find_products_of_elements(queries) do

end
end
```

Erlang:

```
-spec find_products_of_elements(Queries :: [[integer()]]) -> [integer()].
find_products_of_elements(Queries) ->
.
```

Racket:

```
(define/contract (find-products-of-elements queries)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Products of Elements of Big Array
* Difficulty: Hard
* Tags: array, sort, search
*
* 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> findProductsOfElements(vector<vector<long long>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Find Products of Elements of Big Array
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 int[] findProductsOfElements(long[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Products of Elements of Big Array
Difficulty: Hard
Tags: array, sort, search

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 findProductsOfElements(self, queries: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def findProductsOfElements(self, queries):
"""
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find Products of Elements of Big Array
* Difficulty: Hard
* Tags: array, sort, search
*
* 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[][]} queries
* @return {number[]}
*/
var findProductsOfElements = function(queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Products of Elements of Big Array
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 findProductsOfElements(queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Products of Elements of Big Array
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 int[] FindProductsOfElements(long[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Find Products of Elements of Big Array
* Difficulty: Hard
* Tags: array, sort, search
*
* 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* findProductsOfElements(long long** queries, int queriesSize, int*
queriesColSize, int* returnSize) {
}
```

Go Solution:

```
// Problem: Find Products of Elements of Big Array
// Difficulty: Hard
// Tags: array, sort, search
//
// 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 findProductsOfElements(queries [][]int64) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findProductsOfElements(queries: Array<LongArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func findProductsOfElements(_ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Products of Elements of Big Array
// Difficulty: Hard
// Tags: array, sort, search
//
// 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 find_products_of_elements(queries: Vec<Vec<i64>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} queries
# @return {Integer[]}
def find_products_of_elements(queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $queries
* @return Integer[]
*/
function findProductsOfElements($queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findProductsOfElements(List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def findProductsOfElements(queries: Array[Array[Long]]): Array[Int] = {


}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_products_of_elements(queries :: [[integer]]) :: [integer]
def find_products_of_elements(queries) do

end
end
```

Erlang Solution:

```
-spec find_products_of_elements(Queries :: [[integer()]]) -> [integer()].
find_products_of_elements(Queries) ->
.
```

Racket Solution:

```
(define/contract (find-products-of-elements queries)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```