Problem 1744: Can You Eat Your Favorite Candy on Your Favorite Day?

Problem Information

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

Problem Description

You are given a

(0-indexed)

array of positive integers

candiesCount

where

candiesCount[i]

represents the number of candies of the

i

th

type you have. You are also given a 2D array

queries

where

queries[i] = [favoriteType

i

, favoriteDay

i

, dailyCap

i

]

.

You play a game with the following rules:

You start eating candies on day

0

.

You

cannot

eat

any

candy of type

i

unless you have eaten

all

candies of type

i - 1

.

You must eat

at least

one

candy per day until you have eaten all the candies.

Construct a boolean array

answer

such that

answer.length == queries.length

and

answer[i]

is

true

if you can eat a candy of type

favoriteType

i

on day

favoriteDay

i

without eating

more than

dailyCap

i

candies on

any

day, and

false

otherwise. Note that you can eat different types of candy on the same day, provided that you follow rule 2.

Return

the constructed array

answer

.

Example 1:

Input:

candiesCount = [7,4,5,3,8], queries = [[0,2,2],[4,2,4],[2,13,1000000000]]

Output:

[true,false,true]

Explanation:

1- If you eat 2 candies (type 0) on day 0 and 2 candies (type 0) on day 1, you will eat a candy of type 0 on day 2. 2- You can eat at most 4 candies each day. If you eat 4 candies every day, you will eat 4 candies (type 0) on day 0 and 4 candies (type 0 and type 1) on day 1. On day 2, you can only eat 4 candies (type 1 and type 2), so you cannot eat a candy of type 4 on day 2. 3- If you eat 1 candy each day, you will eat a candy of type 2 on day 13.

Example 2:

Input:

candiesCount = [5,2,6,4,1], queries = [[3,1,2],[4,10,3],[3,10,100],[4,100,30],[1,3,1]]

Output:

[false,true,true,false,false]

Constraints:

1 <= candiesCount.length <= 10

5

1 <= candiesCount[i] <= 10

5

1 <= queries.length <= 10

5

queries[i].length == 3

0 <= favoriteType

i

< candiesCount.length

0 <= favoriteDay

i

<= 10

9

1 <= dailyCap

i

<= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<bool> canEat(vector<int>& candiesCount, vector<vector<int>>& queries)
{

}
};
```

Java:

```
class Solution {
public boolean[] canEat(int[] candiesCount, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} candiesCount
* @param {number[][]} queries
* @return {boolean[]}
*/
var canEat = function(candiesCount, queries) {

};
```

TypeScript:

```
function canEat(candiesCount: number[], queries: number[][]): boolean[] {

};
```

C#:

```
public class Solution {
public bool[] CanEat(int[] candiesCount, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
```

```
*/
bool* canEat(int* candiesCount, int candiesCountSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func canEat(candiesCount []int, queries [][]int) []bool {

}
```

Kotlin:

```
class Solution {
fun canEat(candiesCount: IntArray, queries: Array<IntArray>): BooleanArray {

}
}
```

Swift:

```
class Solution {
func canEat(_ candiesCount: [Int], _ queries: [[Int]]) -> [Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn can_eat(candies_count: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<bool>
{

}
}
```

Ruby:

```
# @param {Integer[]} candies_count
# @param {Integer[][]} queries
# @return {Boolean[]}
```

```
def can_eat(candies_count, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $candiesCount
* @param Integer[][] $queries
* @return Boolean[]
*/
function canEat($candiesCount, $queries) {

}
}
```

Dart:

```
class Solution {
List<bool> canEat(List<int> candiesCount, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def canEat(candiesCount: Array[Int], queries: Array[Array[Int]]):
Array[Boolean] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_eat(candies_count :: [integer], queries :: [[integer]]) ::
[boolean]
def can_eat(candies_count, queries) do
```

```
end
end
```

Erlang:

```
-spec can_eat(CandiesCount :: [integer()], Queries :: [[integer()]]) ->
[boolean()].
can_eat(CandiesCount, Queries) ->
.
```

Racket:

```
(define/contract (can-eat candiesCount queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Can You Eat Your Favorite Candy on Your Favorite Day?
* Difficulty: Medium
* Tags: array
*
* 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<bool> canEat(vector<int>& candiesCount, vector<vector<int>>& queries)
{

}
};
```

Java Solution:

```
/**
* Problem: Can You Eat Your Favorite Candy on Your Favorite Day?
* Difficulty: Medium
* Tags: array
*
* 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 boolean[] canEat(int[] candiesCount, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Can You Eat Your Favorite Candy on Your Favorite Day?
Difficulty: Medium
Tags: array

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 canEat(self, candiesCount: List[int], queries: List[List[int]]) ->
List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Can You Eat Your Favorite Candy on Your Favorite Day?
* Difficulty: Medium
* Tags: array
*
* 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[]} candiesCount
* @param {number[][]} queries
* @return {boolean[]}
*/
var canEat = function(candiesCount, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Can You Eat Your Favorite Candy on Your Favorite Day?
* Difficulty: Medium
* Tags: array
*
* 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 canEat(candiesCount: number[], queries: number[][]): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Can You Eat Your Favorite Candy on Your Favorite Day?
* Difficulty: Medium
* Tags: array
*
* 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 bool[] CanEat(int[] candiesCount, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Can You Eat Your Favorite Candy on Your Favorite Day?
* Difficulty: Medium
* Tags: array
*
* 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().
*/
bool* canEat(int* candiesCount, int candiesCountSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Can You Eat Your Favorite Candy on Your Favorite Day?
// Difficulty: Medium
// Tags: array
//
```

```
// 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 canEat(candiesCount []int, queries [][]int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun canEat(candiesCount: IntArray, queries: Array<IntArray>): BooleanArray {

}
}
```

Swift Solution:

```
class Solution {
func canEat(_ candiesCount: [Int], _ queries: [[Int]]) -> [Bool] {

}
}
```

Rust Solution:

```
// Problem: Can You Eat Your Favorite Candy on Your Favorite Day?
// Difficulty: Medium
// Tags: array
//
// 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 can_eat(candies_count: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<bool>
{

}
}
```

Ruby Solution:

```
# @param {Integer[]} candies_count
# @param {Integer[][]} queries
# @return {Boolean[]}
def can_eat(candies_count, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $candiesCount
* @param Integer[][] $queries
* @return Boolean[]
*/
function canEat($candiesCount, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> canEat(List<int> candiesCount, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def canEat(candiesCount: Array[Int], queries: Array[Array[Int]]):
Array[Boolean] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_eat(candies_count :: [integer], queries :: [[integer]]) ::
[boolean]
def can_eat(candies_count, queries) do

end
end
```

Erlang Solution:

```
-spec can_eat(CandiesCount :: [integer()], Queries :: [[integer()]]) ->
[boolean()].
can_eat(CandiesCount, Queries) ->
.
```

Racket Solution:

```
(define/contract (can-eat candiesCount queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
boolean?))
)
```