Problem 370: Range Addition

Problem Information

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

Problem Description

You are given an integer

length

and an array

updates

where

updates[i] = [startIdx

i

, endIdx

i

, inc

i

]

.

You have an array

arr

of length

length

with all zeros, and you have some operation to apply on

arr

. In the

i

th

operation, you should increment all the elements

arr[startIdx

i

], arr[startIdx

i

+ 1], ..., arr[endIdx

i

]

by

inc

i

.

Return

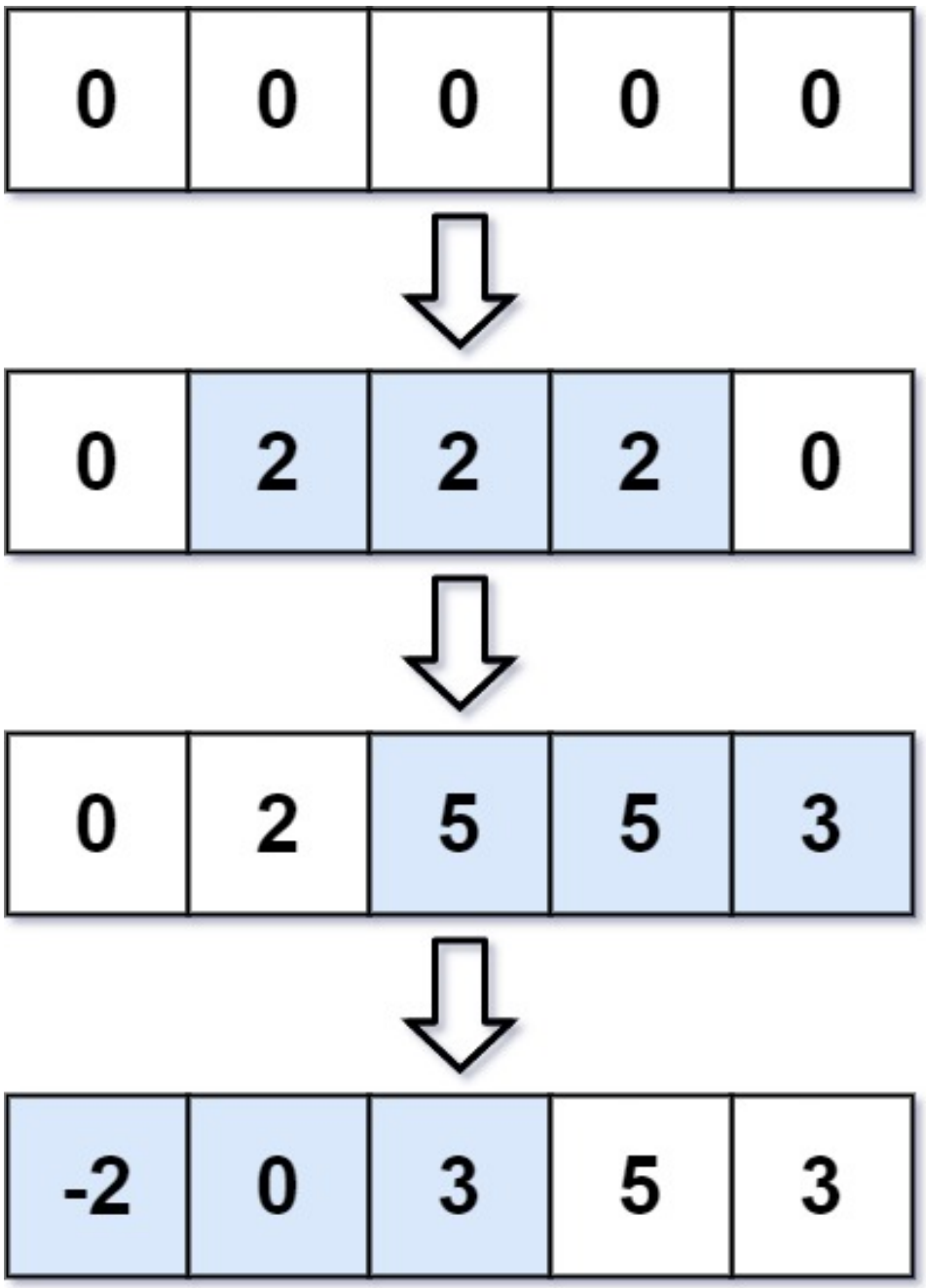
arr

after applying all the

updates

.

Example 1:



Input:

length = 5, updates = [[1,3,2],[2,4,3],[0,2,-2]]

Output:

[-2,0,3,5,3]

Example 2:

Input:

length = 10, updates = [[2,4,6],[5,6,8],[1,9,-4]]

Output:

[0,-4,2,2,2,4,4,-4,-4,-4]

Constraints:

1 <= length <= 10

5

0 <= updates.length <= 10

4

0 <= startIdx

i

<= endIdx

i

< length

-1000 <= inc

i

<= 1000

Code Snippets

C++:

```
class Solution {
public:
vector<int> getModifiedArray(int length, vector<vector<int>>& updates) {

}
};
```

Java:

```
class Solution {
public int[] getModifiedArray(int length, int[][] updates) {

}
}
```

Python3:

```
class Solution:
def getModifiedArray(self, length: int, updates: List[List[int]]) ->
List[int]:
```

Python:

```
class Solution(object):
def getModifiedArray(self, length, updates):
"""
:type length: int
:type updates: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} length
* @param {number[][]} updates
* @return {number[]}
*/
var getModifiedArray = function(length, updates) {

};
```

TypeScript:

```
function getModifiedArray(length: number, updates: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] GetModifiedArray(int length, int[][] updates) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getModifiedArray(int length, int** updates, int updatesSize, int*
updatesColSize, int* returnSize) {

}
```

Go:

```
func getModifiedArray(length int, updates [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun getModifiedArray(length: Int, updates: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func getModifiedArray(_ length: Int, _ updates: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn get_modified_array(length: i32, updates: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} length
# @param {Integer[][]} updates
# @return {Integer[]}
def get_modified_array(length, updates)

end
```

PHP:

```
class Solution {

/**
* @param Integer $length
* @param Integer[][] $updates
* @return Integer[]
*/
function getModifiedArray($length, $updates) {

}
}
```

Dart:

```
class Solution {
List<int> getModifiedArray(int length, List<List<int>> updates) {

}
}
```

Scala:

```
object Solution {
def getModifiedArray(length: Int, updates: Array[Array[Int]]): Array[Int] = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec get_modified_array(length :: integer, updates :: [[integer]]) ::
[integer]
def get_modified_array(length, updates) do

end
end
```

Erlang:

```
-spec get_modified_array(Length :: integer(), Updates :: [[integer()]]) ->
[integer()].
get_modified_array(Length, Updates) ->
.
```

Racket:

```
(define/contract (get-modified-array length updates)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Range Addition
* 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<int> getModifiedArray(int length, vector<vector<int>>& updates) {

}
};
```

Java Solution:

```
/**
* Problem: Range Addition
* 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 int[] getModifiedArray(int length, int[][] updates) {

}
}
```

Python3 Solution:

```
"""
Problem: Range Addition
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 getModifiedArray(self, length: int, updates: List[List[int]]) ->
List[int]:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getModifiedArray(self, length, updates):
"""
:type length: int
:type updates: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Range Addition
* 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} length
* @param {number[][]} updates
* @return {number[]}
*/
var getModifiedArray = function(length, updates) {

};
```

TypeScript Solution:

```
/**
* Problem: Range Addition
* 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 getModifiedArray(length: number, updates: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Range Addition
* 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 int[] GetModifiedArray(int length, int[][] updates) {

}
}
```

C Solution:

```
/*
* Problem: Range Addition
* 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().
```

```
*/
int* getModifiedArray(int length, int** updates, int updatesSize, int*
updatesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Range Addition
// 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 getModifiedArray(length int, updates [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun getModifiedArray(length: Int, updates: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func getModifiedArray(_ length: Int, _ updates: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Range Addition
// 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 get_modified_array(length: i32, updates: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} length
# @param {Integer[][]} updates
# @return {Integer[]}
def get_modified_array(length, updates)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $length
* @param Integer[][] $updates
* @return Integer[]
*/
function getModifiedArray($length, $updates) {

}
}
```

Dart Solution:

```
class Solution {
List<int> getModifiedArray(int length, List<List<int>> updates) {

}
}
```

Scala Solution:

```
object Solution {
def getModifiedArray(length: Int, updates: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_modified_array(length :: integer, updates :: [[integer]]) ::
[integer]
def get_modified_array(length, updates) do

end
end
```

Erlang Solution:

```
-spec get_modified_array(Length :: integer(), Updates :: [[integer()]]) ->
[integer()].
get_modified_array(Length, Updates) ->
.
```

Racket Solution:

```
(define/contract (get-modified-array length updates)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```