Problem 1899: Merge Triplets to Form Target Triplet

Problem Information

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

Problem Description

A

triplet

is an array of three integers. You are given a 2D integer array

triplets

, where

triplets[i] = [a

i

, b

i

, c

i

]

describes the

i

th

triplet

. You are also given an integer array

target = [x, y, z]

that describes the

triplet

you want to obtain.

To obtain

target

, you may apply the following operation on

triplets

any number

of times (possibly

zero

):

Choose two indices (

0-indexed

)

i

and

j

(

i != j

) and

update

triplets[j]

to become

[max(a

i

, a

j

), max(b

i

, b

j

), max(c

i

, c

j

)]

.

For example, if

triplets[i] = [2, 5, 3]

and

triplets[j] = [1, 7, 5]

,

triplets[j]

will be updated to

[max(2, 1), max(5, 7), max(3, 5)] = [2, 7, 5]

.

Return

true

if it is possible to obtain the

target

triplet

[x, y, z]

as an

element

of

triplets

, or

false

otherwise

.

Example 1:

Input:

triplets = [[2,5,3],[1,8,4],[1,7,5]], target = [2,7,5]

Output:

true

Explanation:

Perform the following operations: - Choose the first and last triplets [

[2,5,3]

,[1,8,4],

[1,7,5]

]. Update the last triplet to be [max(2,1), max(5,7), max(3,5)] = [2,7,5]. triplets = [[2,5,3],[1,8,4],

[2,7,5]

] The target triplet [2,7,5] is now an element of triplets.

Example 2:

Input:

triplets = [[3,4,5],[4,5,6]], target = [3,2,5]

Output:

false

Explanation:

It is impossible to have [3,2,5] as an element because there is no 2 in any of the triplets.

Example 3:

Input:

triplets = [[2,5,3],[2,3,4],[1,2,5],[5,2,3]], target = [5,5,5]

Output:

true

Explanation:

Perform the following operations: - Choose the first and third triplets [

[2,5,3]

,[2,3,4],

[1,2,5]

,[5,2,3]]. Update the third triplet to be [max(2,1), max(5,2), max(3,5)] = [2,5,5]. triplets = [[2,5,3],[2,3,4],

[2,5,5]

,[5,2,3]]. - Choose the third and fourth triplets [[2,5,3],[2,3,4],

[2,5,5]

,

[5,2,3]

]. Update the fourth triplet to be [max(2,5), max(5,2), max(5,3)] = [5,5,5]. triplets = [[2,5,3],[2,3,4],[2,5,5],

[5,5,5]

]. The target triplet [5,5,5] is now an element of triplets.

Constraints:

1 <= triplets.length <= 10

5

triplets[i].length == target.length == 3

1 <= a

i

, b

i

, c

i

, x, y, z <= 1000

Code Snippets

C++:

```
class Solution {
public:
bool mergeTriplets(vector<vector<int>>& triplets, vector<int>& target) {

}
};
```

Java:

```
class Solution {
public boolean mergeTriplets(int[][] triplets, int[] target) {

}
}
```

Python3:

```
class Solution:
def mergeTriplets(self, triplets: List[List[int]], target: List[int]) ->
bool:
```

Python:

```
class Solution(object):
def mergeTriplets(self, triplets, target):
"""
:type triplets: List[List[int]]
:type target: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[][]} triplets
* @param {number[]} target
* @return {boolean}
*/
var mergeTriplets = function(triplets, target) {
};
```

TypeScript:

```
function mergeTriplets(triplets: number[][], target: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool MergeTriplets(int[][] triplets, int[] target) {

}
}
```

C:

```
bool mergeTriplets(int** triplets, int tripletsSize, int* tripletsColSize,
int* target, int targetSize) {

}
```

Go:

```
func mergeTriplets(triplets [][]int, target []int) bool {

}
```

Kotlin:

```
class Solution {
fun mergeTriplets(triplets: Array<IntArray>, target: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func mergeTriplets(_ triplets: [[Int]], _ target: [Int]) -> Bool {
}
}
```

Rust:

```
impl Solution {
pub fn merge_triplets(triplets: Vec<Vec<i32>>, target: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[][]} triplets
# @param {Integer[]} target
# @return {Boolean}
def merge_triplets(triplets, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $triplets
* @param Integer[] $target
* @return Boolean
*/
function mergeTriplets($triplets, $target) {

}
}
```

Dart:

```
class Solution {
bool mergeTriplets(List<List<int>> triplets, List<int> target) {

}
}
```

Scala:

```
object Solution {
def mergeTriplets(triplets: Array[Array[Int]], target: Array[Int]): Boolean =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec merge_triplets(triplets :: [[integer]], target :: [integer]) :: boolean
def merge_triplets(triplets, target) do

end
end
```

Erlang:

```
-spec merge_triplets(Triplets :: [[integer()]], Target :: [integer()]) ->
boolean().
merge_triplets(Triplets, Target) ->
.
```

Racket:

```
(define/contract (merge-triplets triplets target)
(-> (listof (listof exact-integer?)) (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Merge Triplets to Form Target Triplet
* Difficulty: Medium
* Tags: array, greedy
*
* 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:
bool mergeTriplets(vector<vector<int>>& triplets, vector<int>& target) {

}
};
```

Java Solution:

```
/**
* Problem: Merge Triplets to Form Target Triplet
* Difficulty: Medium
* Tags: array, greedy
*
* 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 mergeTriplets(int[][] triplets, int[] target) {

}
}
```

Python3 Solution:

```
"""
Problem: Merge Triplets to Form Target Triplet
Difficulty: Medium
Tags: array, greedy

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 mergeTriplets(self, triplets: List[List[int]], target: List[int]) ->
bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mergeTriplets(self, triplets, target):
"""
:type triplets: List[List[int]]
:type target: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Merge Triplets to Form Target Triplet
* Difficulty: Medium
* Tags: array, greedy
*
* 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[][]} triplets
* @param {number[]} target
* @return {boolean}
*/
var mergeTriplets = function(triplets, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Merge Triplets to Form Target Triplet
* Difficulty: Medium
* Tags: array, greedy
*
* 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 mergeTriplets(triplets: number[][], target: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Merge Triplets to Form Target Triplet
* Difficulty: Medium
* Tags: array, greedy
*
* 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 MergeTriplets(int[][] triplets, int[] target) {

}
}
```

C Solution:

```
/*
* Problem: Merge Triplets to Form Target Triplet
* Difficulty: Medium
* Tags: array, greedy
*
* 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
*/
```

```
bool mergeTriplets(int** triplets, int tripletsSize, int* tripletsColSize,
int* target, int targetSize) {

}
```

Go Solution:

```
// Problem: Merge Triplets to Form Target Triplet
// Difficulty: Medium
// Tags: array, greedy
//
// 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 mergeTriplets(triplets [][]int, target []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun mergeTriplets(triplets: Array<IntArray>, target: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func mergeTriplets(_ triplets: [[Int]], _ target: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Merge Triplets to Form Target Triplet
// Difficulty: Medium
// Tags: array, greedy
```

```
//
// 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 merge_triplets(triplets: Vec<Vec<i32>>, target: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} triplets
# @param {Integer[]} target
# @return {Boolean}
def merge_triplets(triplets, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $triplets
* @param Integer[] $target
* @return Boolean
*/
function mergeTriplets($triplets, $target) {

}
}
```

Dart Solution:

```
class Solution {
bool mergeTriplets(List<List<int>> triplets, List<int> target) {

}
}
```

Scala Solution:

```
object Solution {
def mergeTriplets(triplets: Array[Array[Int]], target: Array[Int]): Boolean =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec merge_triplets(triplets :: [[integer]], target :: [integer]) :: boolean
def merge_triplets(triplets, target) do

end
end
```

Erlang Solution:

```
-spec merge_triplets(Triplets :: [[integer()]], Target :: [integer()]) ->
boolean().
merge_triplets(Triplets, Target) ->
.
```

Racket Solution:

```
(define/contract (merge-triplets triplets target)
(-> (listof (listof exact-integer?)) (listof exact-integer?) boolean?)
)
```