Problem 2446: Determine if Two Events Have Conflict

Problem Information

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

Problem Description

You are given two arrays of strings that represent two inclusive events that happened

on the same day

,

event1

and

event2

, where:

event1 = [startTime

1

, endTime

1

]

and

event2 = [startTime

2

, endTime

2

]

.

Event times are valid 24 hours format in the form of

HH:MM

.

A

conflict

happens when two events have some non-empty intersection (i.e., some moment is common to both events).

Return

true

if there is a conflict between two events. Otherwise, return

false

.

Example 1:

Input:

event1 = ["01:15","02:00"], event2 = ["02:00","03:00"]

Output:

true

Explanation:

The two events intersect at time 2:00.

Example 2:

Input:

event1 = ["01:00","02:00"], event2 = ["01:20","03:00"]

Output:

true

Explanation:

The two events intersect starting from 01:20 to 02:00.

Example 3:

Input:

event1 = ["10:00","11:00"], event2 = ["14:00","15:00"]

Output:

false

Explanation:

The two events do not intersect.

Constraints:

event1.length == event2.length == 2

event1[i].length == event2[i].length == 5

startTime

1

<= endTime

1

startTime

2

<= endTime

2

All the event times follow the

HH:MM

format.

Code Snippets

C++:

```
class Solution {
public:
bool haveConflict(vector<string>& event1, vector<string>& event2) {

}
};
```

Java:

```
class Solution {
public boolean haveConflict(String[] event1, String[] event2) {

}
}
```

Python3:

```
class Solution:
def haveConflict(self, event1: List[str], event2: List[str]) -> bool:
```

Python:

```
class Solution(object):
def haveConflict(self, event1, event2):
"""
:type event1: List[str]
:type event2: List[str]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string[]} event1
* @param {string[]} event2
* @return {boolean}
*/
var haveConflict = function(event1, event2) {

};
```

TypeScript:

```
function haveConflict(event1: string[], event2: string[]): boolean {

};
```

C#:

```
public class Solution {
public bool HaveConflict(string[] event1, string[] event2) {

}
}
```

C:

```
bool haveConflict(char** event1, int event1Size, char** event2, int
event2Size) {

}
```

Go:

```
func haveConflict(event1 []string, event2 []string) bool {

}
```

Kotlin:

```
class Solution {
fun haveConflict(event1: Array<String>, event2: Array<String>): Boolean {

}
}
```

Swift:

```
class Solution {
func haveConflict(_ event1: [String], _ event2: [String]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn have_conflict(event1: Vec<String>, event2: Vec<String>) -> bool {

}
}
```

Ruby:

```
# @param {String[]} event1
# @param {String[]} event2
# @return {Boolean}
def have_conflict(event1, event2)

end
```

PHP:

```
class Solution {

/**
* @param String[] $event1
* @param String[] $event2
* @return Boolean
*/
function haveConflict($event1, $event2) {

}
}
```

Dart:

```
class Solution {
bool haveConflict(List<String> event1, List<String> event2) {

}
}
```

Scala:

```
object Solution {
def haveConflict(event1: Array[String], event2: Array[String]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec have_conflict(event1 :: [String.t], event2 :: [String.t]) :: boolean
```

```
def have_conflict(event1, event2) do

end
end
```

Erlang:

```
-spec have_conflict(Event1 :: [unicode:unicode_binary()], Event2 ::
[unicode:unicode_binary()]) -> boolean().
have_conflict(Event1, Event2) ->
.
```

Racket:

```
(define/contract (have-conflict event1 event2)
(-> (listof string?) (listof string?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Determine if Two Events Have Conflict
* Difficulty: Easy
* Tags: array, string
*
* 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 haveConflict(vector<string>& event1, vector<string>& event2) {

}
};
```

Java Solution:

```
/**
* Problem: Determine if Two Events Have Conflict
* Difficulty: Easy
* Tags: array, string
*
* 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 haveConflict(String[] event1, String[] event2) {

}
}
```

Python3 Solution:

```
"""
Problem: Determine if Two Events Have Conflict
Difficulty: Easy
Tags: array, string

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 haveConflict(self, event1: List[str], event2: List[str]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def haveConflict(self, event1, event2):
"""
:type event1: List[str]
:type event2: List[str]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Determine if Two Events Have Conflict
* Difficulty: Easy
* Tags: array, string
*
* 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 {string[]} event1
* @param {string[]} event2
* @return {boolean}
*/
var haveConflict = function(event1, event2) {

};
```

TypeScript Solution:

```
/**
* Problem: Determine if Two Events Have Conflict
* Difficulty: Easy
* Tags: array, string
*
* 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 haveConflict(event1: string[], event2: string[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Determine if Two Events Have Conflict
* Difficulty: Easy
```

```
* Tags: array, string
*
* 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 HaveConflict(string[] event1, string[] event2) {

}
}
```

C Solution:

```
/*
* Problem: Determine if Two Events Have Conflict
* Difficulty: Easy
* Tags: array, string
*
* 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 haveConflict(char** event1, int event1Size, char** event2, int
event2Size) {

}
```

Go Solution:

```
// Problem: Determine if Two Events Have Conflict
// Difficulty: Easy
// Tags: array, string
//
// 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 haveConflict(event1 []string, event2 []string) bool {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun haveConflict(event1: Array<String>, event2: Array<String>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func haveConflict(_ event1: [String], _ event2: [String]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Determine if Two Events Have Conflict
// Difficulty: Easy
// Tags: array, string
//
// 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 have_conflict(event1: Vec<String>, event2: Vec<String>) -> bool {

}
}
```

Ruby Solution:

```
# @param {String[]} event1
# @param {String[]} event2
# @return {Boolean}
def have_conflict(event1, event2)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $event1
* @param String[] $event2
* @return Boolean
*/
function haveConflict($event1, $event2) {

}
}
```

Dart Solution:

```
class Solution {
bool haveConflict(List<String> event1, List<String> event2) {

}
}
```

Scala Solution:

```
object Solution {
def haveConflict(event1: Array[String], event2: Array[String]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec have_conflict(event1 :: [String.t], event2 :: [String.t]) :: boolean
def have_conflict(event1, event2) do

end
end
```

Erlang Solution:

```
-spec have_conflict(Event1 :: [unicode:unicode_binary()], Event2 ::
[unicode:unicode_binary()]) -> boolean().
have_conflict(Event1, Event2) ->
.
```

Racket Solution:

```
(define/contract (have-conflict event1 event2)
(-> (listof string?) (listof string?) boolean?)
)
```