Problem 1101: The Earliest Moment When Everyone Become Friends

Problem Information

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

Problem Description

There are n people in a social group labeled from

0

to

n - 1

. You are given an array

logs

where

logs[i] = [timestamp

i

, x

i

, y

i

]

indicates that

x

i

and

y

i

will be friends at the time

timestamp

i

.

Friendship is

symmetric

. That means if

a

is friends with

b

, then

b

is friends with

a

. Also, person

a

is acquainted with a person

b

if

a

is friends with

b

, or

a

is a friend of someone acquainted with

b

.

Return

the earliest time for which every person became acquainted with every other person

. If there is no such earliest time, return

-1

.

Example 1:

Input:

logs = [[20190101,0,1],[20190104,3,4],[20190107,2,3],[20190211,1,5],[20190224,2,4],[20190301,0,3],[20190312,1,2],[20190322,4,5]], n = 6

Output:

20190301

Explanation:

The first event occurs at timestamp = 20190101, and after 0 and 1 become friends, we have the following friendship groups [0,1], [2], [3], [4], [5]. The second event occurs at timestamp = 20190104, and after 3 and 4 become friends, we have the following friendship groups [0,1], [2], [3,4], [5]. The third event occurs at timestamp = 20190107, and after 2 and 3 become friends, we have the following friendship groups [0,1], [2,3,4], [5]. The fourth event occurs at timestamp = 20190211, and after 1 and 5 become friends, we have the following friendship groups [0,1,5], [2,3,4]. The fifth event occurs at timestamp = 20190224, and as 2 and 4 are already friends, nothing happens. The sixth event occurs at timestamp = 20190301, and after 0 and 3 become friends, we all become friends.

Example 2:

Input:

logs = [[0,2,0],[1,0,1],[3,0,3],[4,1,2],[7,3,1]], n = 4

Output:

3

Explanation:

At timestamp = 3, all the persons (i.e., 0, 1, 2, and 3) become friends.

Constraints:

2 <= n <= 100

1 <= logs.length <= 10

4

logs[i].length == 3

0 <= timestamp

i

<= 10

9

0 <= x

i

, y

i

<= n - 1

x

i

!= y

i

All the values

timestamp

i

are

unique

.

All the pairs

(x

i

, y

i

)

occur at most one time in the input.

Code Snippets

C++:

```
class Solution {
public:
int earliestAcq(vector<vector<int>>& logs, int n) {

}
};
```

Java:

```
class Solution {
public int earliestAcq(int[][] logs, int n) {

}
}
```

Python3:

```
class Solution:
def earliestAcq(self, logs: List[List[int]], n: int) -> int:
```

Python:

```
class Solution(object):
def earliestAcq(self, logs, n):
"""
:type logs: List[List[int]]
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} logs
* @param {number} n
* @return {number}
*/
var earliestAcq = function(logs, n) {

};
```

TypeScript:

```
function earliestAcq(logs: number[][], n: number): number {

};
```

C#:

```
public class Solution {
public int EarliestAcq(int[][] logs, int n) {

}
}
```

C:

```
int earliestAcq(int** logs, int logsSize, int* logsColSize, int n) {

}
```

Go:

```
func earliestAcq(logs [][]int, n int) int {

}
```

Kotlin:

```
class Solution {
fun earliestAcq(logs: Array<IntArray>, n: Int): Int {

}
}
```

Swift:

```
class Solution {
func earliestAcq(_ logs: [[Int]], _ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn earliest_acq(logs: Vec<Vec<i32>>, n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} logs
# @param {Integer} n
# @return {Integer}
def earliest_acq(logs, n)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $logs
* @param Integer $n
* @return Integer
*/
function earliestAcq($logs, $n) {

}
}
```

Dart:

```
class Solution {
int earliestAcq(List<List<int>> logs, int n) {

}
}
```

Scala:

```
object Solution {
def earliestAcq(logs: Array[Array[Int]], n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec earliest_acq(logs :: [[integer]], n :: integer) :: integer
def earliest_acq(logs, n) do

end
end
```

Erlang:

```
-spec earliest_acq(Logs :: [[integer()]], N :: integer()) -> integer().
earliest_acq(Logs, N) ->
```

```
.
```

Racket:

```
(define/contract (earliest-acq logs n)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Earliest Moment When Everyone Become Friends
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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 earliestAcq(vector<vector<int>>& logs, int n) {

}
};
```

Java Solution:

```
/**
* Problem: The Earliest Moment When Everyone Become Friends
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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 earliestAcq(int[][] logs, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: The Earliest Moment When Everyone Become Friends
Difficulty: Medium
Tags: array, graph, sort

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 earliestAcq(self, logs: List[List[int]], n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def earliestAcq(self, logs, n):
"""
:type logs: List[List[int]]
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: The Earliest Moment When Everyone Become Friends
* Difficulty: Medium
* Tags: array, graph, sort
*
```

```
* 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[][]} logs
* @param {number} n
* @return {number}
*/
var earliestAcq = function(logs, n) {

};
```

TypeScript Solution:

```
/**
* Problem: The Earliest Moment When Everyone Become Friends
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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 earliestAcq(logs: number[][], n: number): number {

};
```

C# Solution:

```
/*
* Problem: The Earliest Moment When Everyone Become Friends
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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 EarliestAcq(int[][] logs, int n) {

}
}
```

C Solution:

```
/*
* Problem: The Earliest Moment When Everyone Become Friends
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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
*/

int earliestAcq(int** logs, int logsSize, int* logsColSize, int n) {

}
```

Go Solution:

```
// Problem: The Earliest Moment When Everyone Become Friends
// Difficulty: Medium
// Tags: array, graph, sort
//
// 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 earliestAcq(logs [][]int, n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun earliestAcq(logs: Array<IntArray>, n: Int): Int {
```

```
}
}
```

Swift Solution:

```
class Solution {
func earliestAcq(_ logs: [[Int]], _ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: The Earliest Moment When Everyone Become Friends
// Difficulty: Medium
// Tags: array, graph, sort
//
// 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 earliest_acq(logs: Vec<Vec<i32>>, n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} logs
# @param {Integer} n
# @return {Integer}
def earliest_acq(logs, n)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[][] $logs
* @param Integer $n
* @return Integer
*/
function earliestAcq($logs, $n) {

}
}
```

Dart Solution:

```
class Solution {
int earliestAcq(List<List<int>> logs, int n) {

}
}
```

Scala Solution:

```
object Solution {
def earliestAcq(logs: Array[Array[Int]], n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec earliest_acq(logs :: [[integer]], n :: integer) :: integer
def earliest_acq(logs, n) do

end
end
```

Erlang Solution:

```
-spec earliest_acq(Logs :: [[integer()]], N :: integer()) -> integer().
earliest_acq(Logs, N) ->
.
```

Racket Solution:

```
(define/contract (earliest-acq logs n)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```