Problem 2076: Process Restricted Friend Requests

Problem Information

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

Problem Description

You are given an integer

n

indicating the number of people in a network. Each person is labeled from

0

to

n - 1

.

You are also given a

0-indexed

2D integer array

restrictions

, where

restrictions[i] = [x

i

, y

i

]

means that person

x

i

and person

y

i

cannot

become

friends

,

either

directly

or

indirectly

through other people.

Initially, no one is friends with each other. You are given a list of friend requests as a

0-indexed

2D integer array

requests

, where

requests[j] = [u

j

, v

j

]

is a friend request between person

u

j

and person

v

j

.

A friend request is

successful

if

u

j

and

v

j

can be

friends

. Each friend request is processed in the given order (i.e.,

requests[j]

occurs before

requests[j + 1]

), and upon a successful request,

u

j

and

v

j

become direct friends

for all future friend requests.

Return a boolean array result, where each result[j] is true if the jth friend request is successful or false if it is not.

Note:

If

u

j

and

v

j

are already direct friends, the request is still

successful

.

Example 1:

Input:

n = 3, restrictions = [[0,1]], requests = [[0,2],[2,1]]

Output:

[true,false]

Explanation:

Request 0: Person 0 and person 2 can be friends, so they become direct friends. Request 1: Person 2 and person 1 cannot be friends since person 0 and person 1 would be indirect friends (1--2--0).

Example 2:

Input:

n = 3, restrictions = [[0,1]], requests = [[1,2],[0,2]]

Output:

[true,false]

Explanation:

Request 0: Person 1 and person 2 can be friends, so they become direct friends. Request 1: Person 0 and person 2 cannot be friends since person 0 and person 1 would be indirect friends (0--2--1).

Example 3:

Input:

n = 5, restrictions = [[0,1],[1,2],[2,3]], requests = [[0,4],[1,2],[3,1],[3,4]]

Output:

[true,false,true,false]

Explanation:

Request 0: Person 0 and person 4 can be friends, so they become direct friends. Request 1: Person 1 and person 2 cannot be friends since they are directly restricted. Request 2: Person 3 and person 1 can be friends, so they become direct friends. Request 3: Person 3 and person 4 cannot be friends since person 0 and person 1 would be indirect friends (0--4--3--1).

Constraints:

2 <= n <= 1000

0 <= restrictions.length <= 1000

restrictions[i].length == 2

0 <= x

i

, y

i

<= n - 1

x

i

!= y

i

1 <= requests.length <= 1000

requests[j].length == 2

0 <= u

j

, v

j

<= n - 1

u

j

!= v

j

Code Snippets

C++:

```
class Solution {
public:
vector<bool> friendRequests(int n, vector<vector<int>>& restrictions,
vector<vector<int>>& requests) {

}
};
```

Java:

```
class Solution {
public boolean[] friendRequests(int n, int[][] restrictions, int[][]
requests) {

}
}
```

Python3:

```
class Solution:
def friendRequests(self, n: int, restrictions: List[List[int]], requests:
List[List[int]]) -> List[bool]:
```

Python:

```
class Solution(object):
def friendRequests(self, n, restrictions, requests):
"""
:type n: int
:type restrictions: List[List[int]]
:type requests: List[List[int]]
:rtype: List[bool]
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} restrictions
* @param {number[][]} requests
* @return {boolean[]}
*/
var friendRequests = function(n, restrictions, requests) {


};
```

TypeScript:

```
function friendRequests(n: number, restrictions: number[][], requests:
number[][]): boolean[] {

};
```

C#:

```
public class Solution {
public bool[] FriendRequests(int n, int[][] restrictions, int[][] requests) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* friendRequests(int n, int** restrictions, int restrictionsSize, int*
restrictionsColSize, int** requests, int requestsSize, int* requestsColSize,
int* returnSize) {

}
```

Go:

```
func friendRequests(n int, restrictions [][]int, requests [][]int) []bool {

}
```

Kotlin:

```
class Solution {
fun friendRequests(n: Int, restrictions: Array<IntArray>, requests:
Array<IntArray>): BooleanArray {

}
}
```

Swift:

```
class Solution {
func friendRequests(_ n: Int, _ restrictions: [[Int]], _ requests: [[Int]])
-> [Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn friend_requests(n: i32, restrictions: Vec<Vec<i32>>, requests:
Vec<Vec<i32>>) -> Vec<bool> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} restrictions
# @param {Integer[][]} requests
# @return {Boolean[]}
def friend_requests(n, restrictions, requests)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $restrictions
* @param Integer[][] $requests
* @return Boolean[]
*/
function friendRequests($n, $restrictions, $requests) {
}
}
```

Dart:

```
class Solution {
List<bool> friendRequests(int n, List<List<int>> restrictions,
List<List<int>> requests) {

}
}
```

Scala:

```
object Solution {
def friendRequests(n: Int, restrictions: Array[Array[Int]], requests:
Array[Array[Int]]): Array[Boolean] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec friend_requests(n :: integer, restrictions :: [[integer]], requests ::
[[integer]]) :: [boolean]
def friend_requests(n, restrictions, requests) do

end
end
```

Erlang:

```
-spec friend_requests(N :: integer(), Restrictions :: [[integer()]], Requests
:: [[integer()]]) -> [boolean()].
friend_requests(N, Restrictions, Requests) ->
.
```

Racket:

```
(define/contract (friend-requests n restrictions requests)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Process Restricted Friend Requests
* Difficulty: Hard
* Tags: array, graph
*
* 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> friendRequests(int n, vector<vector<int>>& restrictions,
vector<vector<int>>& requests) {

}
};
```

Java Solution:

```
/**
* Problem: Process Restricted Friend Requests
* Difficulty: Hard
* Tags: array, graph
*
* 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[] friendRequests(int n, int[][] restrictions, int[][]
requests) {
}
}
```

Python3 Solution:

```
"""
Problem: Process Restricted Friend Requests
Difficulty: Hard
Tags: array, graph

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 friendRequests(self, n: int, restrictions: List[List[int]], requests:
List[List[int]]) -> List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def friendRequests(self, n, restrictions, requests):
"""
:type n: int
:type restrictions: List[List[int]]
:type requests: List[List[int]]
:rtype: List[bool]
"""
```

JavaScript Solution:

```
/**
* Problem: Process Restricted Friend Requests
* Difficulty: Hard
* Tags: array, graph
*
* 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} n
* @param {number[][]} restrictions
* @param {number[][]} requests
* @return {boolean[]}
*/
var friendRequests = function(n, restrictions, requests) {

};
```

TypeScript Solution:

```
/**
* Problem: Process Restricted Friend Requests
* Difficulty: Hard
* Tags: array, graph
*
* 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 friendRequests(n: number, restrictions: number[][], requests:
number[][]): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Process Restricted Friend Requests
* Difficulty: Hard
* Tags: array, graph
*
* 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[] FriendRequests(int n, int[][] restrictions, int[][] requests) {

}
}
```

C Solution:

```
/*
* Problem: Process Restricted Friend Requests
* Difficulty: Hard
* Tags: array, graph
*
* 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* friendRequests(int n, int** restrictions, int restrictionsSize, int*
restrictionsColSize, int** requests, int requestsSize, int* requestsColSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Process Restricted Friend Requests
// Difficulty: Hard
// Tags: array, graph
//
// 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 friendRequests(n int, restrictions [][]int, requests [][]int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun friendRequests(n: Int, restrictions: Array<IntArray>, requests:
Array<IntArray>): BooleanArray {

}
}
```

Swift Solution:

```
class Solution {
func friendRequests(_ n: Int, _ restrictions: [[Int]], _ requests: [[Int]])
-> [Bool] {

}
}
```

Rust Solution:

```
// Problem: Process Restricted Friend Requests
// Difficulty: Hard
// Tags: array, graph
//
// 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 friend_requests(n: i32, restrictions: Vec<Vec<i32>>, requests:
Vec<Vec<i32>>) -> Vec<bool> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} restrictions
# @param {Integer[][]} requests
# @return {Boolean[]}
def friend_requests(n, restrictions, requests)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $restrictions
* @param Integer[][] $requests
* @return Boolean[]
*/
function friendRequests($n, $restrictions, $requests) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> friendRequests(int n, List<List<int>> restrictions,
List<List<int>> requests) {

}
}
```

Scala Solution:

```
object Solution {
def friendRequests(n: Int, restrictions: Array[Array[Int]], requests:
Array[Array[Int]]): Array[Boolean] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec friend_requests(n :: integer, restrictions :: [[integer]], requests ::
[[integer]]) :: [boolean]
```

```
def friend_requests(n, restrictions, requests) do

end
end
```

Erlang Solution:

```
-spec friend_requests(N :: integer(), Restrictions :: [[integer()]], Requests
:: [[integer()]]) -> [boolean()].
friend_requests(N, Restrictions, Requests) ->
.
```

Racket Solution:

```
(define/contract (friend-requests n restrictions requests)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof boolean?))
)
```