Problem 2092: Find All People With Secret

Problem Information

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

Problem Description

You are given an integer

n

indicating there are

n

people numbered from

0

to

n - 1

. You are also given a

0-indexed

2D integer array

meetings

where

meetings[i] = [x

i

, y

i

, time

i

]

indicates that person

x

i

and person

y

i

have a meeting at

time

i

. A person may attend

multiple meetings

at the same time. Finally, you are given an integer

firstPerson

.

Person

0

has a

secret

and initially shares the secret with a person

firstPerson

at time

0

. This secret is then shared every time a meeting takes place with a person that has the secret. More formally, for every meeting, if a person

x

i

has the secret at

time

i

, then they will share the secret with person

y

i

, and vice versa.

The secrets are shared

instantaneously

. That is, a person may receive the secret and share it with people in other meetings within the same time frame.

Return

a list of all the people that have the secret after all the meetings have taken place.

You may return the answer in

any order

.

Example 1:

Input:

n = 6, meetings = [[1,2,5],[2,3,8],[1,5,10]], firstPerson = 1

Output:

[0,1,2,3,5]

Explanation:

At time 0, person 0 shares the secret with person 1. At time 5, person 1 shares the secret with person 2. At time 8, person 2 shares the secret with person 3. At time 10, person 1 shares the secret with person 5. Thus, people 0, 1, 2, 3, and 5 know the secret after all the meetings.

Example 2:

Input:

n = 4, meetings = [[3,1,3],[1,2,2],[0,3,3]], firstPerson = 3

Output:

[0,1,3]

Explanation:

At time 0, person 0 shares the secret with person 3. At time 2, neither person 1 nor person 2 know the secret. At time 3, person 3 shares the secret with person 0 and person 1. Thus, people 0, 1, and 3 know the secret after all the meetings.

Example 3:

Input:

n = 5, meetings = [[3,4,2],[1,2,1],[2,3,1]], firstPerson = 1

Output:

[0,1,2,3,4]

Explanation:

At time 0, person 0 shares the secret with person 1. At time 1, person 1 shares the secret with person 2, and person 2 shares the secret with person 3. Note that person 2 can share the secret at the same time as receiving it. At time 2, person 3 shares the secret with person 4. Thus, people 0, 1, 2, 3, and 4 know the secret after all the meetings.

Constraints:

2 <= n <= 10

5

1 <= meetings.length <= 10

5

meetings[i].length == 3

0 <= x

i

, y

i

<= n - 1

x

i

!= y

i

1 <= time

i

<= 10

5

1 <= firstPerson <= n - 1

Code Snippets

C++:

```
class Solution {
public:
vector<int> findAllPeople(int n, vector<vector<int>>& meetings, int
firstPerson) {

}
};
```

Java:

```
class Solution {
public List<Integer> findAllPeople(int n, int[][] meetings, int firstPerson)
{

}
}
```

Python3:

```
class Solution:
def findAllPeople(self, n: int, meetings: List[List[int]], firstPerson: int)
-> List[int]:
```

Python:

```
class Solution(object):
def findAllPeople(self, n, meetings, firstPerson):
"""
:type n: int
:type meetings: List[List[int]]
:type firstPerson: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} meetings
* @param {number} firstPerson
* @return {number[]}
*/
var findAllPeople = function(n, meetings, firstPerson) {

};
```

TypeScript:

```
function findAllPeople(n: number, meetings: number[][], firstPerson: number):
number[] {
```

```
};
```

C#:

```
public class Solution {
public IList<int> FindAllPeople(int n, int[][] meetings, int firstPerson) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findAllPeople(int n, int** meetings, int meetingsSize, int*
meetingsColSize, int firstPerson, int* returnSize) {

}
```

Go:

```
func findAllPeople(n int, meetings [][]int, firstPerson int) []int {

}
```

Kotlin:

```
class Solution {
fun findAllPeople(n: Int, meetings: Array<IntArray>, firstPerson: Int):
List<Int> {

}
}
```

Swift:

```
class Solution {
func findAllPeople(_ n: Int, _ meetings: [[Int]], _ firstPerson: Int) ->
[Int] {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn find_all_people(n: i32, meetings: Vec<Vec<i32>>, first_person: i32) ->
Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} meetings
# @param {Integer} first_person
# @return {Integer[]}
def find_all_people(n, meetings, first_person)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $meetings
* @param Integer $firstPerson
* @return Integer[]
*/
function findAllPeople($n, $meetings, $firstPerson) {

}
}
```

Dart:

```
class Solution {
List<int> findAllPeople(int n, List<List<int>> meetings, int firstPerson) {

}
```

```
}
```

Scala:

```
object Solution {
def findAllPeople(n: Int, meetings: Array[Array[Int]], firstPerson: Int):
List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_all_people(n :: integer, meetings :: [[integer]], first_person ::
integer) :: [integer]
def find_all_people(n, meetings, first_person) do

end
end
```

Erlang:

```
-spec find_all_people(N :: integer(), Meetings :: [[integer()]], FirstPerson
:: integer()) -> [integer()].
find_all_people(N, Meetings, FirstPerson) ->
.
```

Racket:

```
(define/contract (find-all-people n meetings firstPerson)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer? (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find All People With Secret
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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> findAllPeople(int n, vector<vector<int>>& meetings, int
firstPerson) {

}
};
```

Java Solution:

```
/**
* Problem: Find All People With Secret
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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 List<Integer> findAllPeople(int n, int[][] meetings, int firstPerson)
{

}
}
```

Python3 Solution:

```
"""
Problem: Find All People With Secret
Difficulty: Hard
```

```
Tags: array, graph, sort, search

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 findAllPeople(self, n: int, meetings: List[List[int]], firstPerson: int)
-> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findAllPeople(self, n, meetings, firstPerson):
"""
:type n: int
:type meetings: List[List[int]]
:type firstPerson: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find All People With Secret
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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[][]} meetings
* @param {number} firstPerson
* @return {number[]}
```

```
*/
var findAllPeople = function(n, meetings, firstPerson) {

};
```

TypeScript Solution:

```
/**
* Problem: Find All People With Secret
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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 findAllPeople(n: number, meetings: number[][], firstPerson: number):
number[] {

};
```

C# Solution:

```
/*
* Problem: Find All People With Secret
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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 IList<int> FindAllPeople(int n, int[][] meetings, int firstPerson) {

}
}
```

C Solution:

```
/*
* Problem: Find All People With Secret
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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* findAllPeople(int n, int** meetings, int meetingsSize, int*
meetingsColSize, int firstPerson, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find All People With Secret
// Difficulty: Hard
// Tags: array, graph, sort, search
//
// 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 findAllPeople(n int, meetings [][]int, firstPerson int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findAllPeople(n: Int, meetings: Array<IntArray>, firstPerson: Int):
List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func findAllPeople(_ n: Int, _ meetings: [[Int]], _ firstPerson: Int) ->
[Int] {

}
}
```

Rust Solution:

```
// Problem: Find All People With Secret
// Difficulty: Hard
// Tags: array, graph, sort, search
//
// 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 find_all_people(n: i32, meetings: Vec<Vec<i32>>, first_person: i32) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} meetings
# @param {Integer} first_person
# @return {Integer[]}
def find_all_people(n, meetings, first_person)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $meetings
* @param Integer $firstPerson
```

```
* @return Integer[]
*/
function findAllPeople($n, $meetings, $firstPerson) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findAllPeople(int n, List<List<int>> meetings, int firstPerson) {

}
}
```

Scala Solution:

```
object Solution {
def findAllPeople(n: Int, meetings: Array[Array[Int]], firstPerson: Int):
List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_all_people(n :: integer, meetings :: [[integer]], first_person ::
integer) :: [integer]
def find_all_people(n, meetings, first_person) do

end
end
```

Erlang Solution:

```
-spec find_all_people(N :: integer(), Meetings :: [[integer()]], FirstPerson
:: integer()) -> [integer()].
find_all_people(N, Meetings, FirstPerson) ->
.
```

Racket Solution:

```
(define/contract (find-all-people n meetings firstPerson)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer? (listof
exact-integer?))
)
```