Problem 1733: Minimum Number of People to Teach

Problem Information

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

Problem Description

On a social network consisting of

m

users and some friendships between users, two users can communicate with each other if they know a common language.

You are given an integer

n

, an array

languages

, and an array

friendships

where:

There are

n

languages numbered

1

through

n

,

languages[i]

is the set of languages the

i

th

user knows, and

friendships[i] = [u

i

, v

i

]

denotes a friendship between the users

u

i

and

v

i

.

You can choose

one

language and teach it to some users so that all friends can communicate with each other. Return

the

minimum

number of users you need to teach.

Note that friendships are not transitive, meaning if

x

is a friend of

y

and

y

is a friend of

z

, this doesn't guarantee that

x

is a friend of

z

.

Example 1:

Input:

n = 2, languages = [[1],[2],[1,2]], friendships = [[1,2],[1,3],[2,3]]

Output:

1

Explanation:

You can either teach user 1 the second language or user 2 the first language.

Example 2:

Input:

n = 3, languages = [[2],[1,3],[1,2],[3]], friendships = [[1,4],[1,2],[3,4],[2,3]]

Output:

2

Explanation:

Teach the third language to users 1 and 3, yielding two users to teach.

Constraints:

2 <= n <= 500

languages.length == m

1 <= m <= 500

1 <= languages[i].length <= n

1 <= languages[i][j] <= n

1 <= u

i

< v

i

<= languages.length

1 <= friendships.length <= 500

All tuples

(u

i,

v

i

)

are unique

languages[i]

contains only unique values

Code Snippets

C++:

```
class Solution {
public:
int minimumTeachings(int n, vector<vector<int>>& languages,
vector<vector<int>>& friendships) {

}
};
```

Java:

```
class Solution {
public int minimumTeachings(int n, int[][] languages, int[][] friendships) {

}
}
```

Python3:

```
class Solution:
def minimumTeachings(self, n: int, languages: List[List[int]], friendships:
List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minimumTeachings(self, n, languages, friendships):
"""
:type n: int
:type languages: List[List[int]]
:type friendships: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} languages
* @param {number[][]} friendships
* @return {number}
*/
```

```
var minimumTeachings = function(n, languages, friendships) {

};
```

TypeScript:

```
function minimumTeachings(n: number, languages: number[][], friendships:
number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumTeachings(int n, int[][] languages, int[][] friendships) {

}
}
```

C:

```
int minimumTeachings(int n, int** languages, int languagesSize, int*
languagesColSize, int** friendships, int friendshipsSize, int*
friendshipsColSize) {

}
```

Go:

```
func minimumTeachings(n int, languages [][]int, friendships [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumTeachings(n: Int, languages: Array<IntArray>, friendships:
Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minimumTeachings(_ n: Int, _ languages: [[Int]], _ friendships: [[Int]])
-> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_teachings(n: i32, languages: Vec<Vec<i32>>, friendships:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} languages
# @param {Integer[][]} friendships
# @return {Integer}
def minimum_teachings(n, languages, friendships)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $languages
* @param Integer[][] $friendships
* @return Integer
*/
function minimumTeachings($n, $languages, $friendships) {

}
}
```

Dart:

```
class Solution {
int minimumTeachings(int n, List<List<int>> languages, List<List<int>>
friendships) {

}
}
```

Scala:

```
object Solution {
def minimumTeachings(n: Int, languages: Array[Array[Int]], friendships:
Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_teachings(n :: integer, languages :: [[integer]], friendships
:: [[integer]]) :: integer
def minimum_teachings(n, languages, friendships) do

end
end
```

Erlang:

```
-spec minimum_teachings(N :: integer(), Languages :: [[integer()]],
Friendships :: [[integer()]]) -> integer().
minimum_teachings(N, Languages, Friendships) ->
.
```

Racket:

```
(define/contract (minimum-teachings n languages friendships)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of People to Teach
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int minimumTeachings(int n, vector<vector<int>>& languages,
vector<vector<int>>& friendships) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of People to Teach
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int minimumTeachings(int n, int[][] languages, int[][] friendships) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of People to Teach
Difficulty: Medium
Tags: array, greedy, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def minimumTeachings(self, n: int, languages: List[List[int]], friendships:
List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumTeachings(self, n, languages, friendships):
"""
:type n: int
:type languages: List[List[int]]
:type friendships: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of People to Teach
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} n
* @param {number[][]} languages
* @param {number[][]} friendships
* @return {number}
*/
var minimumTeachings = function(n, languages, friendships) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of People to Teach
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function minimumTeachings(n: number, languages: number[][], friendships:
number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of People to Teach
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MinimumTeachings(int n, int[][] languages, int[][] friendships) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of People to Teach
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minimumTeachings(int n, int** languages, int languagesSize, int*
languagesColSize, int** friendships, int friendshipsSize, int*
friendshipsColSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of People to Teach
// Difficulty: Medium
// Tags: array, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func minimumTeachings(n int, languages [][]int, friendships [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumTeachings(n: Int, languages: Array<IntArray>, friendships:
Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumTeachings(_ n: Int, _ languages: [[Int]], _ friendships: [[Int]])
-> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of People to Teach
// Difficulty: Medium
// Tags: array, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn minimum_teachings(n: i32, languages: Vec<Vec<i32>>, friendships:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} languages
# @param {Integer[][]} friendships
# @return {Integer}
def minimum_teachings(n, languages, friendships)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $languages
* @param Integer[][] $friendships
* @return Integer
*/
function minimumTeachings($n, $languages, $friendships) {

}
}
```

Dart Solution:

```
class Solution {
int minimumTeachings(int n, List<List<int>> languages, List<List<int>>
friendships) {

}
}
```

Scala Solution:

```
object Solution {
def minimumTeachings(n: Int, languages: Array[Array[Int]], friendships:
Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_teachings(n :: integer, languages :: [[integer]], friendships
:: [[integer]]) :: integer
def minimum_teachings(n, languages, friendships) do

end
end
```

Erlang Solution:

```
-spec minimum_teachings(N :: integer(), Languages :: [[integer()]],
Friendships :: [[integer()]]) -> integer().
```

```
minimum_teachings(N, Languages, Friendships) ->
.
```

Racket Solution:

```
(define/contract (minimum-teachings n languages friendships)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) exact-integer?)
)
```