Problem 1311: Get Watched Videos by Your Friends

Problem Information

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

Problem Description

There are

n

people, each person has a unique

id

between

0

and

n-1

. Given the arrays

watchedVideos

and

friends

, where

watchedVideos[i]

and

friends[i]

contain the list of watched videos and the list of friends respectively for the person with

id = i

.

Level

1

of videos are all watched videos by your friends, level

2

of videos are all watched videos by the friends of your friends and so on. In general, the level

k

of videos are all watched videos by people with the shortest path

exactly

equal to

k

with you. Given your

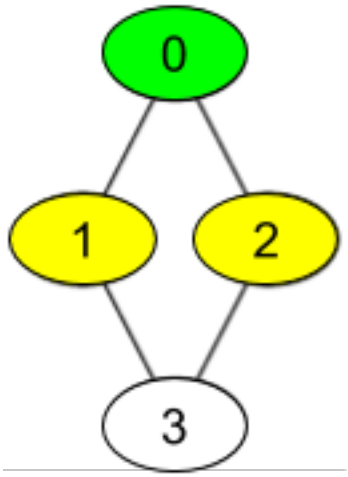
id

and the

level

of videos, return the list of videos ordered by their frequencies (increasing). For videos with the same frequency order them alphabetically from least to greatest.

Example 1:



Input:

watchedVideos = [["A","B"],["C"],["B","C"],["D"]], friends = [[1,2],[0,3],[0,3],[1,2]], id = 0, level = 1

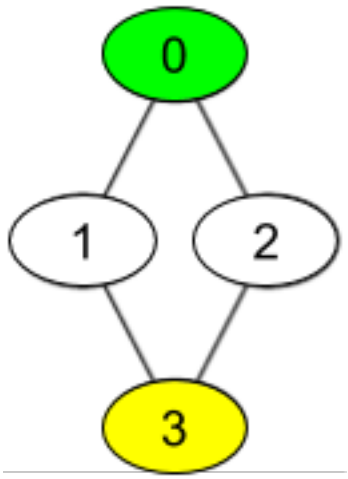
Output:

["B","C"]

Explanation:

You have id = 0 (green color in the figure) and your friends are (yellow color in the figure): Person with id = 1 -> watchedVideos = ["C"] Person with id = 2 -> watchedVideos = ["B","C"] The frequencies of watchedVideos by your friends are: B -> 1 C -> 2

Example 2:



Input:

watchedVideos = [["A","B"],["C"],["B","C"],["D"]], friends = [[1,2],[0,3],[0,3],[1,2]], id = 0, level = 2

Output:

["D"]

Explanation:

You have id = 0 (green color in the figure) and the only friend of your friends is the person with id = 3 (yellow color in the figure).

Constraints:

n == watchedVideos.length == friends.length

2 <= n <= 100

1 <= watchedVideos[i].length <= 100

1 <= watchedVideos[i][j].length <= 8

0 <= friends[i].length < n

0 <= friends[i][j] < n

0 <= id < n

1 <= level < n

if

friends[i]

contains

j

, then

friends[j]

contains

i

Code Snippets

C++:

```
class Solution {
public:
vector<string> watchedVideosByFriends(vector<vector<string>>& watchedVideos,
vector<vector<int>>& friends, int id, int level) {

}
};
```

Java:

```
class Solution {
public List<String> watchedVideosByFriends(List<List<String>> watchedVideos,
int[][] friends, int id, int level) {

}
```

```
}
```

Python3:

```
class Solution:
def watchedVideosByFriends(self, watchedVideos: List[List[str]], friends:
List[List[int]], id: int, level: int) -> List[str]:
```

Python:

```
class Solution(object):
def watchedVideosByFriends(self, watchedVideos, friends, id, level):
"""
:type watchedVideos: List[List[str]]
:type friends: List[List[int]]
:type id: int
:type level: int
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[][]} watchedVideos
* @param {number[][]} friends
* @param {number} id
* @param {number} level
* @return {string[]}
*/
var watchedVideosByFriends = function(watchedVideos, friends, id, level) {

};
```

TypeScript:

```
function watchedVideosByFriends(watchedVideos: string[][], friends:
number[][], id: number, level: number): string[] {

};
```

C#:

```
public class Solution {
public IList<string> WatchedVideosByFriends(IList<IList<string>>
watchedVideos, int[][] friends, int id, int level) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** watchedVideosByFriends(char*** watchedVideos, int watchedVideosSize,
int* watchedVideosColSize, int** friends, int friendsSize, int*
friendsColSize, int id, int level, int* returnSize) {

}
```

Go:

```
func watchedVideosByFriends(watchedVideos [][]string, friends [][]int, id
int, level int) []string {

}
```

Kotlin:

```
class Solution {
fun watchedVideosByFriends(watchedVideos: List<List<String>>, friends:
Array<IntArray>, id: Int, level: Int): List<String> {

}
}
```

Swift:

```
class Solution {
func watchedVideosByFriends(_ watchedVideos: [[String]], _ friends: [[Int]],
_ id: Int, _ level: Int) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn watched_videos_by_friends(watched_videos: Vec<Vec<String>>, friends:
Vec<Vec<i32>>, id: i32, level: i32) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[][]} watched_videos
# @param {Integer[][]} friends
# @param {Integer} id
# @param {Integer} level
# @return {String[]}
def watched_videos_by_friends(watched_videos, friends, id, level)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $watchedVideos
* @param Integer[][] $friends
* @param Integer $id
* @param Integer $level
* @return String[]
*/
function watchedVideosByFriends($watchedVideos, $friends, $id, $level) {

}
}
```

Dart:

```
class Solution {
List<String> watchedVideosByFriends(List<List<String>> watchedVideos,
List<List<int>> friends, int id, int level) {

}
```

```
}
```

Scala:

```
object Solution {
def watchedVideosByFriends(watchedVideos: List[List[String]], friends:
Array[Array[Int]], id: Int, level: Int): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec watched_videos_by_friends(watched_videos :: [[String.t]], friends ::
[[integer]], id :: integer, level :: integer) :: [String.t]
def watched_videos_by_friends(watched_videos, friends, id, level) do

end
end
```

Erlang:

```
-spec watched_videos_by_friends(WatchedVideos ::
[[unicode:unicode_binary()]], Friends :: [[integer()]], Id :: integer(),
Level :: integer()) -> [unicode:unicode_binary()].
watched_videos_by_friends(WatchedVideos, Friends, Id, Level) ->
.
```

Racket:

```
(define/contract (watched-videos-by-friends watchedVideos friends id level)
(-> (listof (listof string?)) (listof (listof exact-integer?)) exact-integer?
exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Get Watched Videos by Your Friends
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* 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:
vector<string> watchedVideosByFriends(vector<vector<string>>& watchedVideos,
vector<vector<int>>& friends, int id, int level) {

}
};
```

Java Solution:

```
/**
* Problem: Get Watched Videos by Your Friends
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* 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 List<String> watchedVideosByFriends(List<List<String>> watchedVideos,
int[][] friends, int id, int level) {

}
}
```

Python3 Solution:

```
"""
Problem: Get Watched Videos by Your Friends
Difficulty: Medium
```

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

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 watchedVideosByFriends(self, watchedVideos: List[List[str]], friends:
List[List[int]], id: int, level: int) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def watchedVideosByFriends(self, watchedVideos, friends, id, level):
"""
:type watchedVideos: List[List[str]]
:type friends: List[List[int]]
:type id: int
:type level: int
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Get Watched Videos by Your Friends
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* 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 {string[][]} watchedVideos
* @param {number[][]} friends
* @param {number} id
```

```
* @param {number} level
* @return {string[]}
*/
var watchedVideosByFriends = function(watchedVideos, friends, id, level) {

};
```

TypeScript Solution:

```
/**
* Problem: Get Watched Videos by Your Friends
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* 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 watchedVideosByFriends(watchedVideos: string[][], friends:
number[][], id: number, level: number): string[] {

};
```

C# Solution:

```
/*
* Problem: Get Watched Videos by Your Friends
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* 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 IList<string> WatchedVideosByFriends(IList<IList<string>>
watchedVideos, int[][] friends, int id, int level) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Get Watched Videos by Your Friends
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** watchedVideosByFriends(char*** watchedVideos, int watchedVideosSize,
int* watchedVideosColSize, int** friends, int friendsSize, int*
friendsColSize, int id, int level, int* returnSize) {

}
```

Go Solution:

```
// Problem: Get Watched Videos by Your Friends
// Difficulty: Medium
// Tags: array, graph, hash, sort, search
//
// 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 watchedVideosByFriends(watchedVideos [][]string, friends [][]int, id
int, level int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun watchedVideosByFriends(watchedVideos: List<List<String>>, friends:
Array<IntArray>, id: Int, level: Int): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func watchedVideosByFriends(_ watchedVideos: [[String]], _ friends: [[Int]],
_ id: Int, _ level: Int) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Get Watched Videos by Your Friends
// Difficulty: Medium
// Tags: array, graph, hash, sort, search
//
// 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 watched_videos_by_friends(watched_videos: Vec<Vec<String>>, friends:
Vec<Vec<i32>>, id: i32, level: i32) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[][]} watched_videos
# @param {Integer[][]} friends
# @param {Integer} id
# @param {Integer} level
# @return {String[]}
def watched_videos_by_friends(watched_videos, friends, id, level)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $watchedVideos
* @param Integer[][] $friends
* @param Integer $id
* @param Integer $level
* @return String[]
*/
function watchedVideosByFriends($watchedVideos, $friends, $id, $level) {

}
}
```

Dart Solution:

```
class Solution {
List<String> watchedVideosByFriends(List<List<String>> watchedVideos,
List<List<int>> friends, int id, int level) {

}
}
```

Scala Solution:

```
object Solution {
def watchedVideosByFriends(watchedVideos: List[List[String]], friends:
Array[Array[Int]], id: Int, level: Int): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec watched_videos_by_friends(watched_videos :: [[String.t]], friends ::
[[integer]], id :: integer, level :: integer) :: [String.t]
```

```
def watched_videos_by_friends(watched_videos, friends, id, level) do

end
end
```

Erlang Solution:

```
-spec watched_videos_by_friends(WatchedVideos ::
[[unicode:unicode_binary()]], Friends :: [[integer()]], Id :: integer(),
Level :: integer()) -> [unicode:unicode_binary()].
watched_videos_by_friends(WatchedVideos, Friends, Id, Level) ->
.
```

Racket Solution:

```
(define/contract (watched-videos-by-friends watchedVideos friends id level)
(-> (listof (listof string?)) (listof (listof exact-integer?)) exact-integer?
exact-integer? (listof string?))
)
```