Problem 1923: Longest Common Subpath

Problem Information

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

Problem Description

There is a country of

n

cities numbered from

0

to

n - 1

. In this country, there is a road connecting

every pair

of cities.

There are

m

friends numbered from

0

to

m - 1

who are traveling through the country. Each one of them will take a path consisting of some cities. Each path is represented by an integer array that contains the visited cities in order. The path may contain a city

more than once

, but the same city will not be listed consecutively.

Given an integer

n

and a 2D integer array

paths

where

paths[i]

is an integer array representing the path of the

i

th

friend, return

the length of the

longest common subpath

that is shared by

every

friend's path, or

0

if there is no common subpath at all

.

A

subpath

of a path is a contiguous sequence of cities within that path.

Example 1:

Input:

n = 5, paths = [[0,1,

2,3

,4], [

2,3

,4], [4,0,1,

2,3

]]

Output:

2

Explanation:

The longest common subpath is [2,3].

Example 2:

Input:

n = 3, paths = [[0],[1],[2]]

Output:

0

Explanation:

There is no common subpath shared by the three paths.

Example 3:

Input:

n = 5, paths = [[

0

,1,2,3,4], [4,3,2,1,

0

]]

Output:

1

Explanation:

The possible longest common subpaths are [0], [1], [2], [3], and [4]. All have a length of 1.

Constraints:

1 <= n <= 10

5

m == paths.length

2 <= m <= 10

5

sum(paths[i].length) <= 10

5

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

The same city is not listed multiple times consecutively in

paths[i]

.

Code Snippets

C++:

```
class Solution {
public:
int longestCommonSubpath(int n, vector<vector<int>>& paths) {

}
};
```

Java:

```
class Solution {
public int longestCommonSubpath(int n, int[][] paths) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function longestCommonSubpath(n: number, paths: number[][]): number {

};
```

C#:

```
public class Solution {
public int LongestCommonSubpath(int n, int[][] paths) {

}
}
```

C:

```
int longestCommonSubpath(int n, int** paths, int pathsSize, int*
pathsColSize) {

}
```

Go:

```
func longestCommonSubpath(n int, paths [][]int) int {

}
```

Kotlin:

```
class Solution {
fun longestCommonSubpath(n: Int, paths: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} paths
# @return {Integer}
def longest_common_subpath(n, paths)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $paths
* @return Integer
*/
function longestCommonSubpath($n, $paths) {

}
}
```

Dart:

```
class Solution {
int longestCommonSubpath(int n, List<List<int>> paths) {

}
}
```

Scala:

```
object Solution {
def longestCommonSubpath(n: Int, paths: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_common_subpath(n :: integer, paths :: [[integer]]) :: integer
def longest_common_subpath(n, paths) do

end
end
```

Erlang:

```
-spec longest_common_subpath(N :: integer(), Paths :: [[integer()]]) ->
integer().
longest_common_subpath(N, Paths) ->
.
```

Racket:

```
(define/contract (longest-common-subpath n paths)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Common Subpath
* Difficulty: Hard
* Tags: array, hash, 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:
int longestCommonSubpath(int n, vector<vector<int>>& paths) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Common Subpath
* Difficulty: Hard
* Tags: array, hash, 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 int longestCommonSubpath(int n, int[][] paths) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Common Subpath
Difficulty: Hard
Tags: array, hash, 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 longestCommonSubpath(self, n: int, paths: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Common Subpath
* Difficulty: Hard
* Tags: array, hash, 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 {number} n
* @param {number[][]} paths
* @return {number}
*/
var longestCommonSubpath = function(n, paths) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Common Subpath
* Difficulty: Hard
* Tags: array, hash, 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 longestCommonSubpath(n: number, paths: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Common Subpath
* Difficulty: Hard
* Tags: array, hash, 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 int LongestCommonSubpath(int n, int[][] paths) {

}
}
```

C Solution:

```
/*
* Problem: Longest Common Subpath
* Difficulty: Hard
* Tags: array, hash, 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
*/

int longestCommonSubpath(int n, int** paths, int pathsSize, int*
pathsColSize) {

}
```

Go Solution:

```
// Problem: Longest Common Subpath
// Difficulty: Hard
// Tags: array, hash, 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 longestCommonSubpath(n int, paths [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestCommonSubpath(n: Int, paths: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Common Subpath
// Difficulty: Hard
// Tags: array, hash, 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 longest_common_subpath(n: i32, paths: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} paths
# @return {Integer}
def longest_common_subpath(n, paths)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $paths
* @return Integer
*/
function longestCommonSubpath($n, $paths) {

}
}
```

Dart Solution:

```
class Solution {
int longestCommonSubpath(int n, List<List<int>> paths) {

}
}
```

Scala Solution:

```
object Solution {
def longestCommonSubpath(n: Int, paths: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_common_subpath(n :: integer, paths :: [[integer]]) :: integer
def longest_common_subpath(n, paths) do

end
end
```

Erlang Solution:

```
-spec longest_common_subpath(N :: integer(), Paths :: [[integer()]]) ->
integer().
longest_common_subpath(N, Paths) ->
.
```

Racket Solution:

```
(define/contract (longest-common-subpath n paths)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```