Problem 1129: Shortest Path with Alternating Colors

Problem Information

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

Problem Description

You are given an integer

n

, the number of nodes in a directed graph where the nodes are labeled from

0

to

n - 1

. Each edge is red or blue in this graph, and there could be self-edges and parallel edges.

You are given two arrays

redEdges

and

blueEdges

where:

redEdges[i] = [a

i

, b

i

]

indicates that there is a directed red edge from node

a

i

to node

b

i

in the graph, and

blueEdges[j] = [u

j

, v

j

]

indicates that there is a directed blue edge from node

u

j

to node

v

j

in the graph.

Return an array

answer

of length

n

, where each

answer[x]

is the length of the shortest path from node

0

to node

x

such that the edge colors alternate along the path, or

-1

if such a path does not exist.

Example 1:

Input:

n = 3, redEdges = [[0,1],[1,2]], blueEdges = []

Output:

[0,1,-1]

Example 2:

Input:

n = 3, redEdges = [[0,1]], blueEdges = [[2,1]]

Output:

[0,1,-1]

Constraints:

1 <= n <= 100

0 <= redEdges.length, blueEdges.length <= 400

redEdges[i].length == blueEdges[j].length == 2

0 <= a

i

, b

i

, u

j

, v

j

< n

Code Snippets

C++:

```
class Solution {
public:
vector<int> shortestAlternatingPaths(int n, vector<vector<int>>& redEdges,
vector<vector<int>>& blueEdges) {

}
};
```

Java:

```
class Solution {
public int[] shortestAlternatingPaths(int n, int[][] redEdges, int[][]
blueEdges) {

}
}
```

Python3:

```
class Solution:
def shortestAlternatingPaths(self, n: int, redEdges: List[List[int]],
blueEdges: List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def shortestAlternatingPaths(self, n, redEdges, blueEdges):
"""
:type n: int
:type redEdges: List[List[int]]
:type blueEdges: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} redEdges
* @param {number[][]} blueEdges
* @return {number[]}
*/
var shortestAlternatingPaths = function(n, redEdges, blueEdges) {

};
```

TypeScript:

```
function shortestAlternatingPaths(n: number, redEdges: number[][], blueEdges:
number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] ShortestAlternatingPaths(int n, int[][] redEdges, int[][]
blueEdges) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* shortestAlternatingPaths(int n, int** redEdges, int redEdgesSize, int*
redEdgesColSize, int** blueEdges, int blueEdgesSize, int* blueEdgesColSize,
int* returnSize) {

}
```

Go:

```
func shortestAlternatingPaths(n int, redEdges [][]int, blueEdges [][]int)
[]int {

}
```

Kotlin:

```
class Solution {
fun shortestAlternatingPaths(n: Int, redEdges: Array<IntArray>, blueEdges:
Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func shortestAlternatingPaths(_ n: Int, _ redEdges: [[Int]], _ blueEdges:
[[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn shortest_alternating_paths(n: i32, red_edges: Vec<Vec<i32>>,
blue_edges: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} red_edges
# @param {Integer[][]} blue_edges
# @return {Integer[]}
def shortest_alternating_paths(n, red_edges, blue_edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
```

```
* @param Integer[][] $redEdges
* @param Integer[][] $blueEdges
* @return Integer[]
*/
function shortestAlternatingPaths($n, $redEdges, $blueEdges) {

}
}
```

Dart:

```
class Solution {
List<int> shortestAlternatingPaths(int n, List<List<int>> redEdges,
List<List<int>> blueEdges) {

}
}
```

Scala:

```
object Solution {
def shortestAlternatingPaths(n: Int, redEdges: Array[Array[Int]], blueEdges:
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec shortest_alternating_paths(n :: integer, red_edges :: [[integer]],
blue_edges :: [[integer]]) :: [integer]
def shortest_alternating_paths(n, red_edges, blue_edges) do

end
end
```

Erlang:

```
-spec shortest_alternating_paths(N :: integer(), RedEdges :: [[integer()]],
BlueEdges :: [[integer()]]) -> [integer()].
shortest_alternating_paths(N, RedEdges, BlueEdges) ->
```

```
.
```

Racket:

```
(define/contract (shortest-alternating-paths n redEdges blueEdges)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Path with Alternating Colors
* Difficulty: Medium
* Tags: array, graph, 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> shortestAlternatingPaths(int n, vector<vector<int>>& redEdges,
vector<vector<int>>& blueEdges) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Path with Alternating Colors
* Difficulty: Medium
* Tags: array, graph, 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 int[] shortestAlternatingPaths(int n, int[][] redEdges, int[][]
blueEdges) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Path with Alternating Colors
Difficulty: Medium
Tags: array, graph, 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 shortestAlternatingPaths(self, n: int, redEdges: List[List[int]],
blueEdges: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def shortestAlternatingPaths(self, n, redEdges, blueEdges):
"""
:type n: int
:type redEdges: List[List[int]]
:type blueEdges: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Shortest Path with Alternating Colors
* Difficulty: Medium
* Tags: array, graph, 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[][]} redEdges
* @param {number[][]} blueEdges
* @return {number[]}
*/
var shortestAlternatingPaths = function(n, redEdges, blueEdges) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Path with Alternating Colors
* Difficulty: Medium
* Tags: array, graph, 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 shortestAlternatingPaths(n: number, redEdges: number[][], blueEdges:
number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Shortest Path with Alternating Colors
* Difficulty: Medium
```

```
* Tags: array, graph, 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 int[] ShortestAlternatingPaths(int n, int[][] redEdges, int[][]
blueEdges) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Path with Alternating Colors
* Difficulty: Medium
* Tags: array, graph, 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* shortestAlternatingPaths(int n, int** redEdges, int redEdgesSize, int*
redEdgesColSize, int** blueEdges, int blueEdgesSize, int* blueEdgesColSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Shortest Path with Alternating Colors
// Difficulty: Medium
// Tags: array, graph, 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 shortestAlternatingPaths(n int, redEdges [][]int, blueEdges [][]int)
[]int {

}
```

Kotlin Solution:

```
class Solution {
fun shortestAlternatingPaths(n: Int, redEdges: Array<IntArray>, blueEdges:
Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func shortestAlternatingPaths(_ n: Int, _ redEdges: [[Int]], _ blueEdges:
[[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Shortest Path with Alternating Colors
// Difficulty: Medium
// Tags: array, graph, 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 shortest_alternating_paths(n: i32, red_edges: Vec<Vec<i32>>,
blue_edges: Vec<Vec<i32>>) -> Vec<i32> {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} red_edges
# @param {Integer[][]} blue_edges
# @return {Integer[]}
def shortest_alternating_paths(n, red_edges, blue_edges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $redEdges
* @param Integer[][] $blueEdges
* @return Integer[]
*/
function shortestAlternatingPaths($n, $redEdges, $blueEdges) {

}
}
```

Dart Solution:

```
class Solution {
List<int> shortestAlternatingPaths(int n, List<List<int>> redEdges,
List<List<int>> blueEdges) {

}
}
```

Scala Solution:

```
object Solution {
def shortestAlternatingPaths(n: Int, redEdges: Array[Array[Int]], blueEdges:
```

```
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shortest_alternating_paths(n :: integer, red_edges :: [[integer]],
blue_edges :: [[integer]]) :: [integer]
def shortest_alternating_paths(n, red_edges, blue_edges) do

end
end
```

Erlang Solution:

```
-spec shortest_alternating_paths(N :: integer(), RedEdges :: [[integer()]],
BlueEdges :: [[integer()]]) -> [integer()].
shortest_alternating_paths(N, RedEdges, BlueEdges) ->
.
```

Racket Solution:

```
(define/contract (shortest-alternating-paths n redEdges blueEdges)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```