Problem 1042: Flower Planting With No Adjacent

Problem Information

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

Problem Description

You have

n

gardens, labeled from

1

to

n

, and an array

paths

where

paths[i] = [x

i

, y

i

]

describes a bidirectional path between garden

x

i

to garden

y

i

. In each garden, you want to plant one of 4 types of flowers.

All gardens have

at most 3

paths coming into or leaving it.

Your task is to choose a flower type for each garden such that, for any two gardens connected by a path, they have different types of flowers.

Return

any

such a choice as an array

answer

, where

answer[i]

is the type of flower planted in the

(i+1)

th

garden. The flower types are denoted

1

,

2

,

3

, or

4

. It is guaranteed an answer exists.

Example 1:

Input:

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

Output:

[1,2,3]

Explanation:

Gardens 1 and 2 have different types. Gardens 2 and 3 have different types. Gardens 3 and 1 have different types. Hence, [1,2,3] is a valid answer. Other valid answers include [1,2,4], [1,4,2], and [3,2,1].

Example 2:

Input:

n = 4, paths = [[1,2],[3,4]]

Output:

[1,2,1,2]

Example 3:

Input:

n = 4, paths = [[1,2],[2,3],[3,4],[4,1],[1,3],[2,4]]

Output:

[1,2,3,4]

Constraints:

1 <= n <= 10

4

0 <= paths.length <= 2 * 10

4

paths[i].length == 2

1 <= x

i

, y

i

<= n

x

i

!= y

i

Every garden has

at most 3

paths coming into or leaving it.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* gardenNoAdj(int n, int** paths, int pathsSize, int* pathsColSize, int*
returnSize) {
```

```
}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (garden-no-adj n paths)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Flower Planting With No Adjacent
* 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> gardenNoAdj(int n, vector<vector<int>>& paths) {

}
};
```

Java Solution:

```
/**
* Problem: Flower Planting With No Adjacent
* 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[] gardenNoAdj(int n, int[][] paths) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Flower Planting With No Adjacent
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 gardenNoAdj(self, n: int, paths: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Flower Planting With No Adjacent
* 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[][]} paths
* @return {number[]}
*/
var gardenNoAdj = function(n, paths) {

};
```

TypeScript Solution:

```
/**
* Problem: Flower Planting With No Adjacent
* 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 gardenNoAdj(n: number, paths: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Flower Planting With No Adjacent
* 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[] GardenNoAdj(int n, int[][] paths) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Flower Planting With No Adjacent
* 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* gardenNoAdj(int n, int** paths, int pathsSize, int* pathsColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Flower Planting With No Adjacent
// 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 gardenNoAdj(n int, paths [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Flower Planting With No Adjacent
// 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 garden_no_adj(n: i32, paths: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (garden-no-adj n paths)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```