Problem 3004: Maximum Subtree of the Same Color

Problem Information

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

Problem Description

You are given a 2D integer array

edges

representing a tree with

n

nodes, numbered from

0

to

n - 1

, rooted at node

0

, where

edges[i] = [u

i

, v

i

]

means there is an edge between the nodes

v

i

and

u

i

.

You are also given a

0-indexed

integer array

colors

of size

n

, where

colors[i]

is the color assigned to node

i

.

We want to find a node

v

such that every node in the

subtree

of

v

has the

same

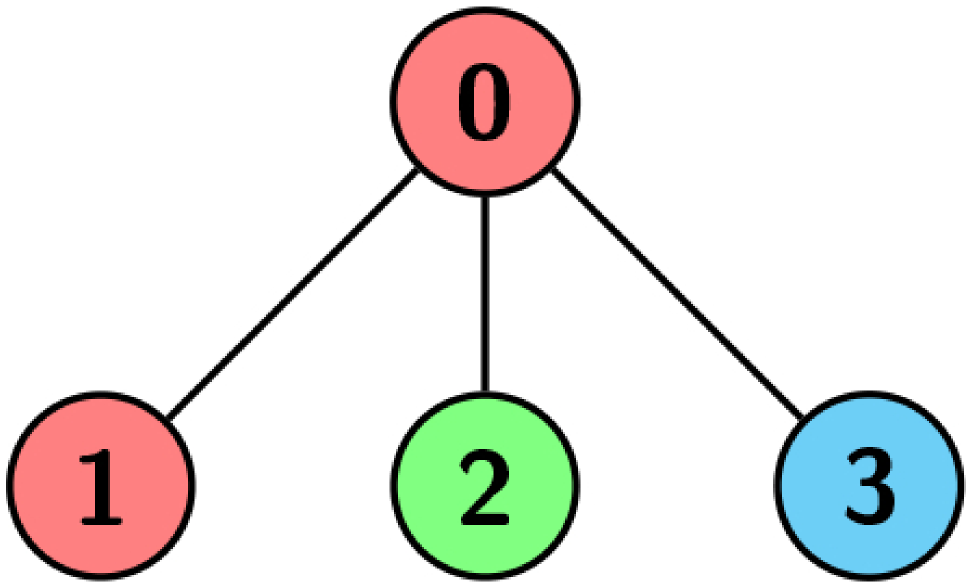
color.

Return

the size of such subtree with the

maximum

number of nodes possible.



Example 1:

Input:

edges = [[0,1],[0,2],[0,3]], colors = [1,1,2,3]

Output:

1

Explanation:

Each color is represented as: 1 -> Red, 2 -> Green, 3 -> Blue. We can see that the subtree rooted at node 0 has children with different colors. Any other subtree is of the same color and has a size of 1. Hence, we return 1.

Example 2:

Input:

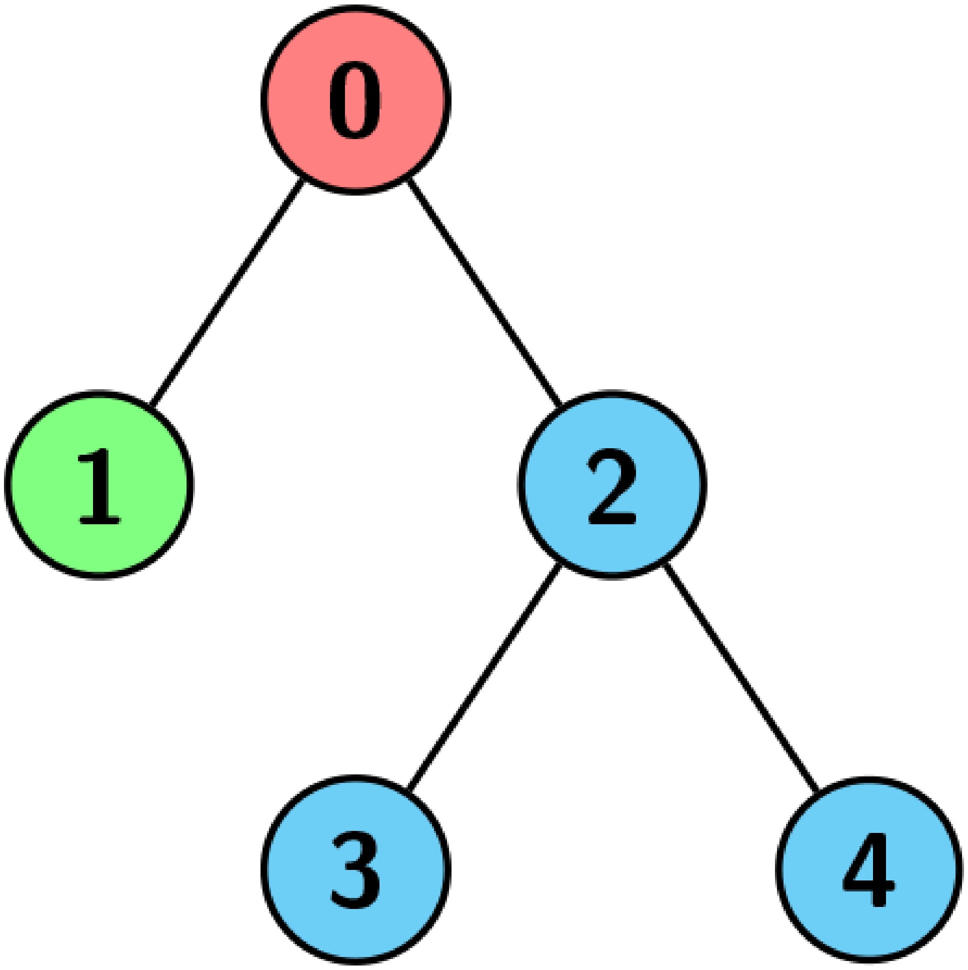
edges = [[0,1],[0,2],[0,3]], colors = [1,1,1,1]

Output:

4

Explanation:

The whole tree has the same color, and the subtree rooted at node 0 has the most number of nodes which is 4. Hence, we return 4.



Example 3:

Input:

edges = [[0,1],[0,2],[2,3],[2,4]], colors = [1,2,3,3,3]

Output:

3

Explanation:

Each color is represented as: 1 -> Red, 2 -> Green, 3 -> Blue. We can see that the subtree rooted at node 0 has children with different colors. Any other subtree is of the same color, but the subtree rooted at node 2 has a size of 3 which is the maximum. Hence, we return 3.

Constraints:

n == edges.length + 1

1 <= n <= 5 * 10

4

edges[i] == [u

i

, v

i

]

0 <= u

i

, v

i

< n

colors.length == n

1 <= colors[i] <= 10

5

The input is generated such that the graph represented by edges

is a tree.

Code Snippets

C++:

```
class Solution {
public:
int maximumSubtreeSize(vector<vector<int>>& edges, vector<int>& colors) {

}
};
```

Java:

```
class Solution {
public int maximumSubtreeSize(int[][] edges, int[] colors) {

}
}
```

Python3:

```
class Solution:
def maximumSubtreeSize(self, edges: List[List[int]], colors: List[int]) ->
int:
```

Python:

```
class Solution(object):
def maximumSubtreeSize(self, edges, colors):
"""
:type edges: List[List[int]]
:type colors: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} edges
* @param {number[]} colors
* @return {number}
*/
var maximumSubtreeSize = function(edges, colors) {

};
```

TypeScript:

```
function maximumSubtreeSize(edges: number[][], colors: number[]): number {

};
```

C#:

```
public class Solution {
public int MaximumSubtreeSize(int[][] edges, int[] colors) {

}
}
```

C:

```
int maximumSubtreeSize(int** edges, int edgesSize, int* edgesColSize, int*
colors, int colorsSize) {

}
```

Go:

```
func maximumSubtreeSize(edges [][]int, colors []int) int {

}
```

Kotlin:

```
class Solution {
fun maximumSubtreeSize(edges: Array<IntArray>, colors: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maximumSubtreeSize(_ edges: [[Int]], _ colors: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_subtree_size(edges: Vec<Vec<i32>>, colors: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer[]} colors
# @return {Integer}
def maximum_subtree_size(edges, colors)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $colors
* @return Integer
*/
function maximumSubtreeSize($edges, $colors) {

}
}
```

Dart:

```
class Solution {
int maximumSubtreeSize(List<List<int>> edges, List<int> colors) {



}
}
```

Scala:

```
object Solution {
def maximumSubtreeSize(edges: Array[Array[Int]], colors: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_subtree_size(edges :: [[integer]], colors :: [integer]) ::
integer
def maximum_subtree_size(edges, colors) do

end
end
```

Erlang:

```
-spec maximum_subtree_size(Edges :: [[integer()]], Colors :: [integer()]) ->
integer().
maximum_subtree_size(Edges, Colors) ->
.
```

Racket:

```
(define/contract (maximum-subtree-size edges colors)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Subtree of the Same Color
* Difficulty: Medium
* Tags: array, tree, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maximumSubtreeSize(vector<vector<int>>& edges, vector<int>& colors) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Subtree of the Same Color
* Difficulty: Medium
* Tags: array, tree, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maximumSubtreeSize(int[][] edges, int[] colors) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Subtree of the Same Color
Difficulty: Medium
Tags: array, tree, graph, dp, search
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maximumSubtreeSize(self, edges: List[List[int]], colors: List[int]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumSubtreeSize(self, edges, colors):
"""
:type edges: List[List[int]]
:type colors: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Subtree of the Same Color
* Difficulty: Medium
* Tags: array, tree, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} edges
* @param {number[]} colors
* @return {number}
*/
var maximumSubtreeSize = function(edges, colors) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Subtree of the Same Color
* Difficulty: Medium
* Tags: array, tree, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maximumSubtreeSize(edges: number[][], colors: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Subtree of the Same Color
* Difficulty: Medium
* Tags: array, tree, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaximumSubtreeSize(int[][] edges, int[] colors) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Subtree of the Same Color
* Difficulty: Medium
* Tags: array, tree, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maximumSubtreeSize(int** edges, int edgesSize, int* edgesColSize, int*
colors, int colorsSize) {

}
```

Go Solution:

```
// Problem: Maximum Subtree of the Same Color
// Difficulty: Medium
// Tags: array, tree, graph, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maximumSubtreeSize(edges [][]int, colors []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumSubtreeSize(edges: Array<IntArray>, colors: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumSubtreeSize(_ edges: [[Int]], _ colors: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Subtree of the Same Color
// Difficulty: Medium
// Tags: array, tree, graph, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn maximum_subtree_size(edges: Vec<Vec<i32>>, colors: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer[]} colors
# @return {Integer}
def maximum_subtree_size(edges, colors)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $colors
* @return Integer
*/
function maximumSubtreeSize($edges, $colors) {

}
}
```

Dart Solution:

```
class Solution {
int maximumSubtreeSize(List<List<int>> edges, List<int> colors) {
}
}
```

Scala Solution:

```
object Solution {
def maximumSubtreeSize(edges: Array[Array[Int]], colors: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_subtree_size(edges :: [[integer]], colors :: [integer]) ::
integer
def maximum_subtree_size(edges, colors) do

end
end
```

Erlang Solution:

```
-spec maximum_subtree_size(Edges :: [[integer()]], Colors :: [integer()]) ->
integer().
maximum_subtree_size(Edges, Colors) ->
.
```

Racket Solution:

```
(define/contract (maximum-subtree-size edges colors)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?)
)
```