# Problem 2440: Create Components With Same Value

## Problem Information

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

## Problem Description

There is an undirected tree with

n

nodes labeled from

0

to

n - 1

.

You are given a

0-indexed

integer array

nums

of length

n

where

nums[i]

represents the value of the

i

th

node. You are also given a 2D integer array

edges

of length

n - 1

where

edges[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

in the tree.

You are allowed to

delete

some edges, splitting the tree into multiple connected components. Let the

value

of a component be the sum of

all

nums[i]

for which node

i

is in the component.

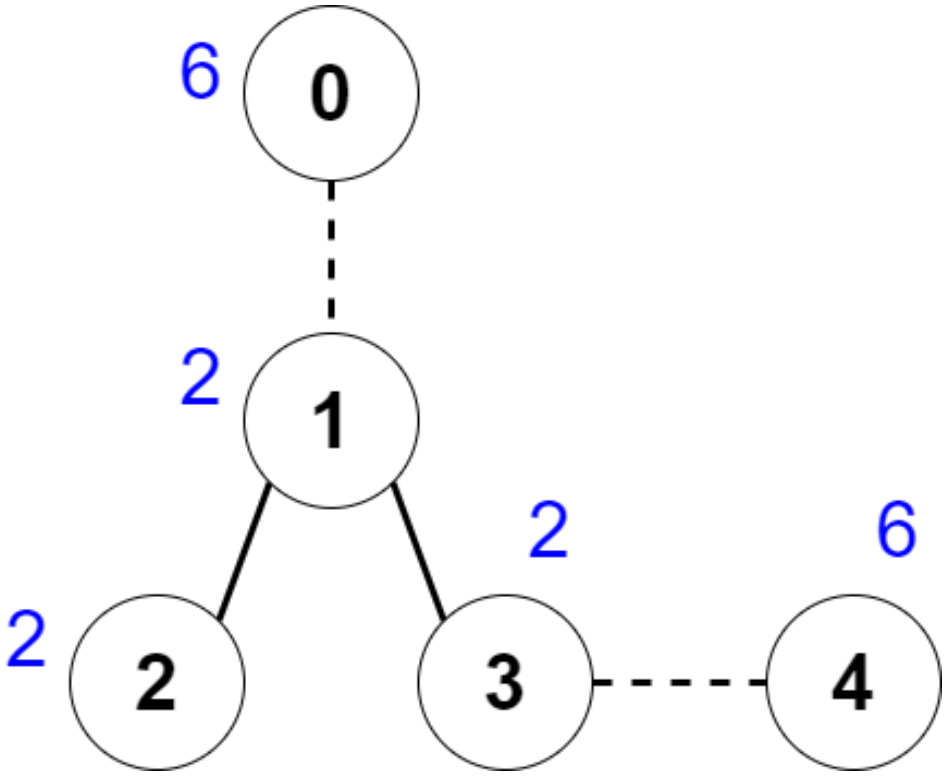
Return

the

maximum

number of edges you can delete, such that every connected component in the tree has the same value.

Example 1:




Input:

nums = [6,2,2,2,6], edges = [[0,1],[1,2],[1,3],[3,4]]

Output:

2

Explanation:

The above figure shows how we can delete the edges [0,1] and [3,4]. The created components are nodes [0], [1,2,3] and [4]. The sum of the values in each component equals 6. It can be proven that no better deletion exists, so the answer is 2.

Example 2:

Input:

nums = [2], edges = []

Output:

0

Explanation:

There are no edges to be deleted.

Constraints:

1 <= n <= 2 * 10

4

nums.length == n

1 <= nums[i] <= 50

edges.length == n - 1

edges[i].length == 2

0 <= edges[i][0], edges[i][1] <= n - 1

edges

represents a valid tree.

## Code Snippets

**C++:**

```
class Solution {
public:
int componentValue(vector<int>& nums, vector<vector<int>>& edges) {

}
};
```

**Java:**

```
class Solution {
public int componentValue(int[] nums, int[][] edges) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function componentValue(nums: number[], edges: number[][]): number {

};
```

**C#:**

```
public class Solution {
public int ComponentValue(int[] nums, int[][] edges) {
```

```
}
}
```

**C:**

```
int componentValue(int* nums, int numsSize, int** edges, int edgesSize, int*
edgesColSize) {

}
```

**Go:**

```
func componentValue(nums []int, edges [][]int) int {

}
```

**Kotlin:**

```
class Solution {
fun componentValue(nums: IntArray, edges: Array<IntArray>): Int {

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

```
# @param {Integer[]} nums
# @param {Integer[][]} edges
# @return {Integer}
def component_value(nums, edges)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $edges
* @return Integer
*/
function componentValue($nums, $edges) {

}
}
```

**Dart:**

```
class Solution {
int componentValue(List<int> nums, List<List<int>> edges) {

}
}
```

**Scala:**

```
object Solution {
def componentValue(nums: Array[Int], edges: Array[Array[Int]]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec component_value(nums :: [integer], edges :: [[integer]]) :: integer
def component_value(nums, edges) do
```

```
end
end
```

**Erlang:**

```
-spec component_value(Nums :: [integer()], Edges :: [[integer()]]) ->
integer().
component_value(Nums, Edges) ->
.
```

**Racket:**

```
(define/contract (component-value nums edges)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Create Components With Same Value
* Difficulty: Hard
* Tags: array, tree, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int componentValue(vector<int>& nums, vector<vector<int>>& edges) {

}
};
```

### Java Solution:

```
/**
* Problem: Create Components With Same Value
* Difficulty: Hard
* Tags: array, tree, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int componentValue(int[] nums, int[][] edges) {

}
}
```

### Python3 Solution:

```
"""
Problem: Create Components With Same Value
Difficulty: Hard
Tags: array, tree, math, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Create Components With Same Value
* Difficulty: Hard
* Tags: array, tree, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

### TypeScript Solution:

```
/**
* Problem: Create Components With Same Value
* Difficulty: Hard
* Tags: array, tree, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function componentValue(nums: number[], edges: number[][]): number {

};
```

### C# Solution:

```
/*
* Problem: Create Components With Same Value
* Difficulty: Hard
```

```
* Tags: array, tree, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int ComponentValue(int[] nums, int[][] edges) {

}
}
```

### C Solution:

```
/*
* Problem: Create Components With Same Value
* Difficulty: Hard
* Tags: array, tree, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int componentValue(int* nums, int numsSize, int** edges, int edgesSize, int*
edgesColSize) {

}
```

### Go Solution:

```
// Problem: Create Components With Same Value
// Difficulty: Hard
// Tags: array, tree, math, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func componentValue(nums []int, edges [][]int) int {
```

```
}
```

**Kotlin Solution:**

```
class Solution {
fun componentValue(nums: IntArray, edges: Array<IntArray>): Int {

}
}
```

**Swift Solution:**

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

}
}
```

**Rust Solution:**

```
// Problem: Create Components With Same Value
// Difficulty: Hard
// Tags: array, tree, math, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn component_value(nums: Vec<i32>, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

**Ruby Solution:**

```
# @param {Integer[]} nums
# @param {Integer[][]} edges
# @return {Integer}
def component_value(nums, edges)
```

```
end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $edges
* @return Integer
*/
function componentValue($nums, $edges) {

}
}
```

### Dart Solution:

```
class Solution {
int componentValue(List<int> nums, List<List<int>> edges) {

}
}
```

### Scala Solution:

```
object Solution {
def componentValue(nums: Array[Int], edges: Array[Array[Int]]): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec component_value(nums :: [integer], edges :: [[integer]]) :: integer
def component_value(nums, edges) do

end
end
```

### Erlang Solution:

```
-spec component_value(Nums :: [integer()], Edges :: [[integer()]]) ->
integer().
component_value(Nums, Edges) ->
.
```

### Racket Solution:

```
(define/contract (component-value nums edges)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```