# Problem 2322: Minimum Score After Removals on a Tree

## Problem Information

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

## Problem Description

There is an undirected connected tree with

n

nodes labeled from

0

to

n - 1

and

n - 1

edges.

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.

Remove two

distinct

edges of the tree to form three connected components. For a pair of removed edges, the following steps are defined:

Get the XOR of all the values of the nodes for

each

of the three components respectively.

The

difference

between the

largest

XOR value and the

smallest

XOR value is the

score

of the pair.

For example, say the three components have the node values:

[4,5,7]

,

[1,9]

, and

[3,3,3]

. The three XOR values are

4 ^ 5 ^ 7 =

6

,

1 ^ 9 =

8

, and

3 ^ 3 ^ 3 =

3

. The largest XOR value is

8

and the smallest XOR value is

3

. The score is then

8 - 3 = 5

.

Return

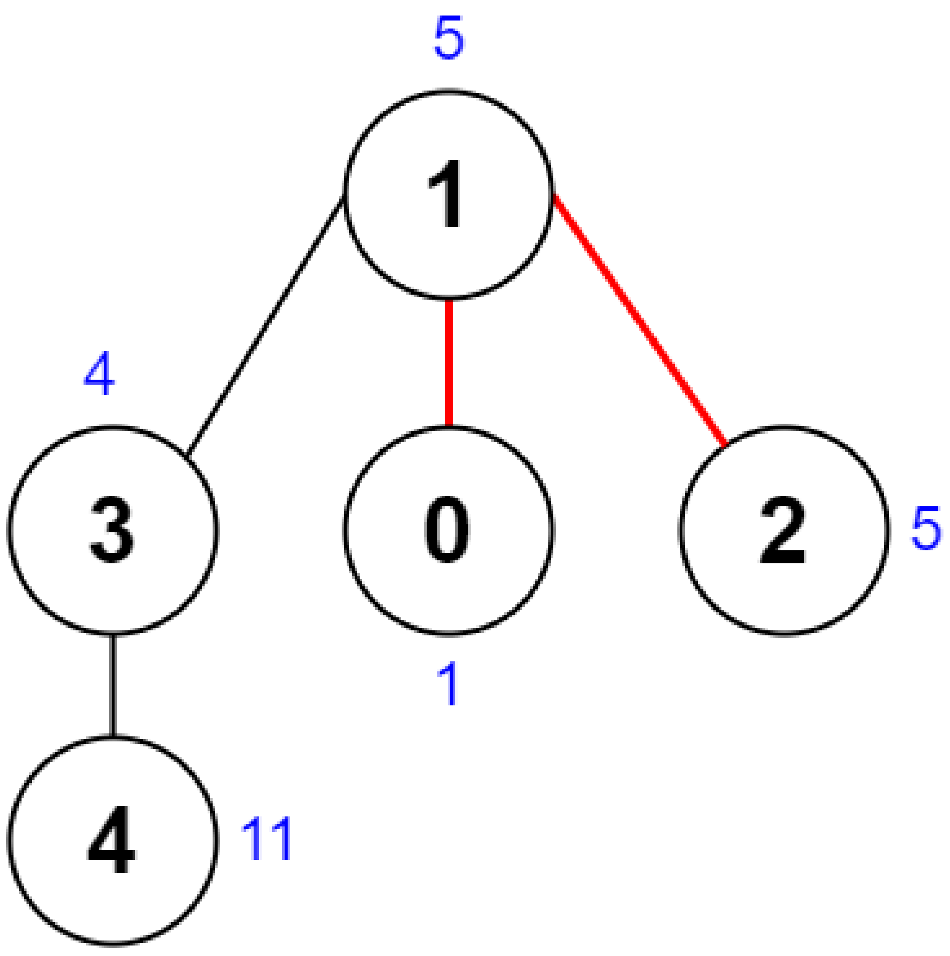
the

minimum

score of any possible pair of edge removals on the given tree

.

Example 1:



Input:

nums = [1,5,5,4,11], edges = [[0,1],[1,2],[1,3],[3,4]]

Output:

9

Explanation:

The diagram above shows a way to make a pair of removals. - The 1

st

component has nodes [1,3,4] with values [5,4,11]. Its XOR value is 5 ^ 4 ^ 11 = 10. - The 2

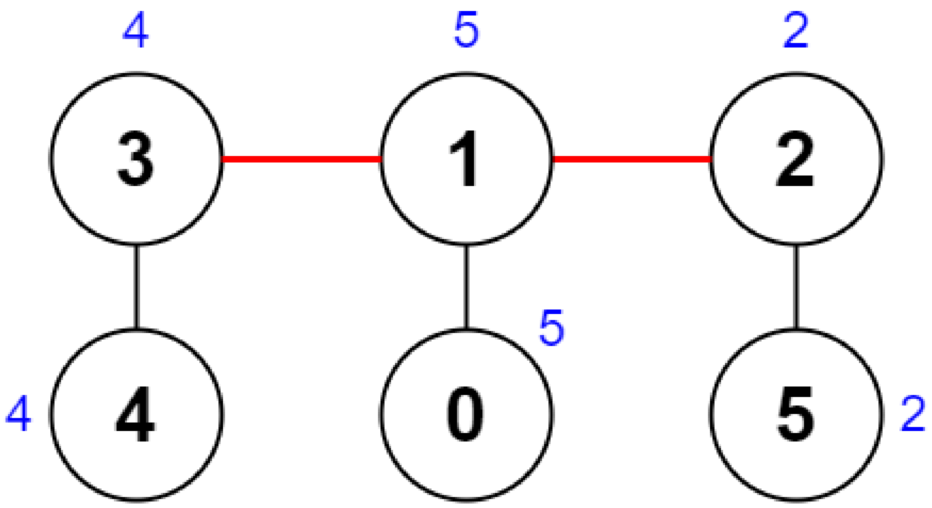
nd

component has node [0] with value [1]. Its XOR value is 1 = 1. - The 3

rd

component has node [2] with value [5]. Its XOR value is 5 = 5. The score is the difference between the largest and smallest XOR value which is 10 - 1 = 9. It can be shown that no other pair of removals will obtain a smaller score than 9.

Example 2:



Input:

nums = [5,5,2,4,4,2], edges = [[0,1],[1,2],[5,2],[4,3],[1,3]]

Output:

0

Explanation:

The diagram above shows a way to make a pair of removals. - The 1

st

component has nodes [3,4] with values [4,4]. Its XOR value is 4 ^ 4 = 0. - The 2

nd

component has nodes [1,0] with values [5,5]. Its XOR value is 5 ^ 5 = 0. - The 3

rd

component has nodes [2,5] with values [2,2]. Its XOR value is 2 ^ 2 = 0. The score is the difference between the largest and smallest XOR value which is 0 - 0 = 0. We cannot obtain a smaller score than 0.

Constraints:

n == nums.length

3 <= n <= 1000

1 <= nums[i] <= 10

8

edges.length == n - 1

edges[i].length == 2

0 <= a

i

, b

i

< n

a

i

!= b

i

edges

represents a valid tree.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (minimum-score nums edges)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Minimum Score After Removals on a Tree
* Difficulty: Hard
* Tags: array, tree, 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 minimumScore(vector<int>& nums, vector<vector<int>>& edges) {

}
};
```

### Java Solution:

```
/**
* Problem: Minimum Score After Removals on a Tree
* Difficulty: Hard
* Tags: array, tree, 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 minimumScore(int[] nums, int[][] edges) {

}
}
```

### Python3 Solution:

```
"""
Problem: Minimum Score After Removals on a Tree
Difficulty: Hard
Tags: array, tree, 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 minimumScore(self, nums: List[int], edges: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

**JavaScript Solution:**

```
/**
* Problem: Minimum Score After Removals on a Tree
* Difficulty: Hard
* Tags: array, tree, 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 minimumScore = function(nums, edges) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Minimum Score After Removals on a Tree
* Difficulty: Hard
* Tags: array, tree, 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 minimumScore(nums: number[], edges: number[][]): number {
};
```

**C# Solution:**

```
/*
* Problem: Minimum Score After Removals on a Tree
* Difficulty: Hard
* Tags: array, tree, 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 MinimumScore(int[] nums, int[][] edges) {

}
}
```

**C Solution:**

```
/*
* Problem: Minimum Score After Removals on a Tree
* Difficulty: Hard
* Tags: array, tree, 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 minimumScore(int* nums, int numsSize, int** edges, int edgesSize, int*
edgesColSize) {

}
```

**Go Solution:**

```
// Problem: Minimum Score After Removals on a Tree
// Difficulty: Hard
// Tags: array, tree, 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 minimumScore(nums []int, edges [][]int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Minimum Score After Removals on a Tree
// Difficulty: Hard
// Tags: array, tree, 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 minimum_score(nums: Vec<i32>, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

**Ruby Solution:**

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

end
```

**PHP Solution:**

```
class Solution {

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

}
}
```

**Dart Solution:**

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

}
}
```

**Scala Solution:**

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (minimum-score nums edges)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```