Problem 3068: Find the Maximum Sum of Node Values

Problem Information

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

Problem Description

There exists an

undirected

tree with

n

nodes numbered

0

to

n - 1

. You are given a

0-indexed

2D integer array

edges

of length

n - 1

, where

edges[i] = [u

i

, v

i

]

indicates that there is an edge between nodes

u

i

and

v

i

in the tree. You are also given a

positive

integer

k

, and a

0-indexed

array of

non-negative

integers

nums

of length

n

, where

nums[i]

represents the

value

of the node numbered

i

.

Alice wants the sum of values of tree nodes to be

maximum

, for which Alice can perform the following operation

any

number of times (

including zero

) on the tree:

Choose any edge

[u, v]

connecting the nodes

u

and

v

, and update their values as follows:

nums[u] = nums[u] XOR k

nums[v] = nums[v] XOR k

Return

the

maximum

possible

sum

of the

values

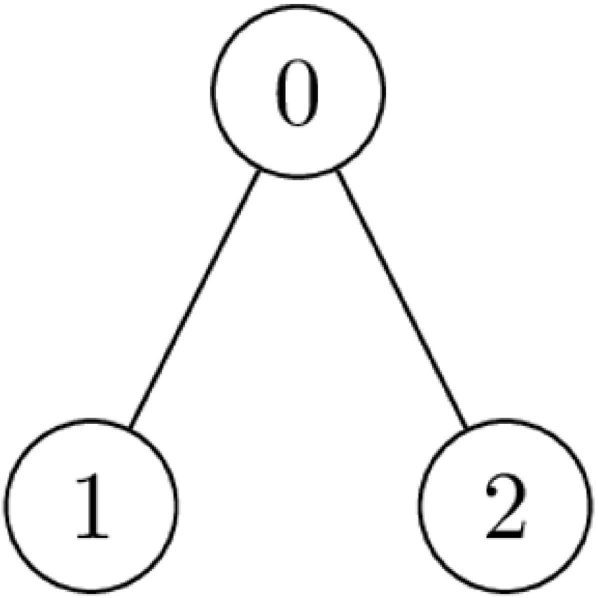
Alice can achieve by performing the operation

any

number of times

.

Example 1:



Input:

nums = [1,2,1], k = 3, edges = [[0,1],[0,2]]

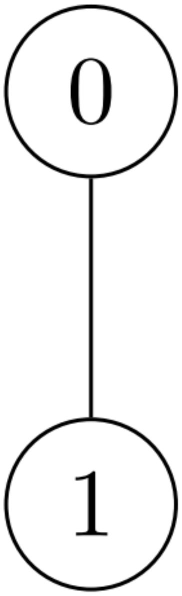
Output:

6

Explanation:

Alice can achieve the maximum sum of 6 using a single operation: - Choose the edge [0,2]. nums[0] and nums[2] become: 1 XOR 3 = 2, and the array nums becomes: [1,2,1] -> [2,2,2]. The total sum of values is 2 + 2 + 2 = 6. It can be shown that 6 is the maximum achievable sum of values.

Example 2:



Input:

nums = [2,3], k = 7, edges = [[0,1]]

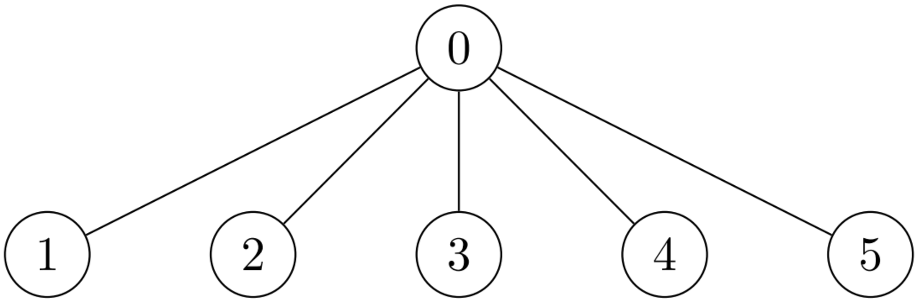
Output:

9

Explanation:

Alice can achieve the maximum sum of 9 using a single operation: - Choose the edge [0,1]. nums[0] becomes: 2 XOR 7 = 5 and nums[1] become: 3 XOR 7 = 4, and the array nums becomes: [2,3] -> [5,4]. The total sum of values is 5 + 4 = 9. It can be shown that 9 is the maximum achievable sum of values.

Example 3:



Input:

nums = [7,7,7,7,7,7], k = 3, edges = [[0,1],[0,2],[0,3],[0,4],[0,5]]

Output:

42

Explanation:

The maximum achievable sum is 42 which can be achieved by Alice performing no operations.

Constraints:

2 <= n == nums.length <= 2 * 10

4

1 <= k <= 10

9

0 <= nums[i] <= 10

9

edges.length == n - 1

edges[i].length == 2

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

The input is generated such that

edges

represent a valid tree.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maximumValueSum(int[] nums, int k, int[][] edges) {

}
}
```

Python3:

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

Python:

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

```
:type k: int
:type edges: List[List[int]]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MaximumValueSum(int[] nums, int k, int[][] edges) {

}
}
```

C:

```
long long maximumValueSum(int* nums, int numsSize, int k, int** edges, int
edgesSize, int* edgesColSize) {

}
```

Go:

```
func maximumValueSum(nums []int, k int, edges [][]int) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn maximum_value_sum(nums: Vec<i32>, k: i32, edges: Vec<Vec<i32>>) -> i64
{

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
maximum_value_sum(Nums, K, Edges) ->
.
```

Racket:

```
(define/contract (maximum-value-sum nums k edges)
(-> (listof exact-integer?) exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Maximum Sum of Node Values
* Difficulty: Hard
* Tags: array, tree, dp, greedy, sort
*
* 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:
long long maximumValueSum(vector<int>& nums, int k, vector<vector<int>>&
edges) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Maximum Sum of Node Values
* Difficulty: Hard
* Tags: array, tree, dp, greedy, sort
*
* 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 long maximumValueSum(int[] nums, int k, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Maximum Sum of Node Values
Difficulty: Hard
Tags: array, tree, dp, greedy, sort

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 maximumValueSum(self, nums: List[int], k: int, edges: List[List[int]]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Maximum Sum of Node Values
* Difficulty: Hard
* Tags: array, tree, dp, greedy, sort
*
* 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[]} nums
* @param {number} k
* @param {number[][]} edges
* @return {number}
*/
var maximumValueSum = function(nums, k, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Maximum Sum of Node Values
* Difficulty: Hard
* Tags: array, tree, dp, greedy, sort
*
* 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 maximumValueSum(nums: number[], k: number, edges: number[][]):
number {

};
```

C# Solution:

```
/*
* Problem: Find the Maximum Sum of Node Values
* Difficulty: Hard
* Tags: array, tree, dp, greedy, sort
*
* 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 long MaximumValueSum(int[] nums, int k, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Find the Maximum Sum of Node Values
* Difficulty: Hard
* Tags: array, tree, dp, greedy, sort
*
* 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
*/

long long maximumValueSum(int* nums, int numsSize, int k, int** edges, int
edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Find the Maximum Sum of Node Values
// Difficulty: Hard
// Tags: array, tree, dp, greedy, sort
//
// 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 maximumValueSum(nums []int, k int, edges [][]int) int64 {
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Maximum Sum of Node Values
// Difficulty: Hard
// Tags: array, tree, dp, greedy, sort
//
// 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_value_sum(nums: Vec<i32>, k: i32, edges: Vec<Vec<i32>>) -> i64
{

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_value_sum(nums :: [integer], k :: integer, edges ::
```

```
[[integer]]) :: integer
def maximum_value_sum(nums, k, edges) do

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-value-sum nums k edges)
(-> (listof exact-integer?) exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```