Problem 2872: Maximum Number of K-Divisible Components

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 the integer

n

and 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 also given a

0-indexed

integer array

values

of length

n

, where

values[i]

is the

value

associated with the

i

th

node, and an integer

k

.

A

valid split

of the tree is obtained by removing any set of edges, possibly empty, from the tree such that the resulting components all have values that are divisible by

k

, where the

value of a connected component

is the sum of the values of its nodes.

Return

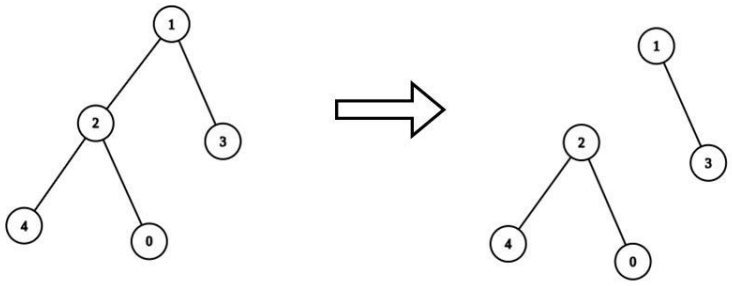
the

maximum number of components

in any valid split

.

Example 1:



Input:

n = 5, edges = [[0,2],[1,2],[1,3],[2,4]], values = [1,8,1,4,4], k = 6

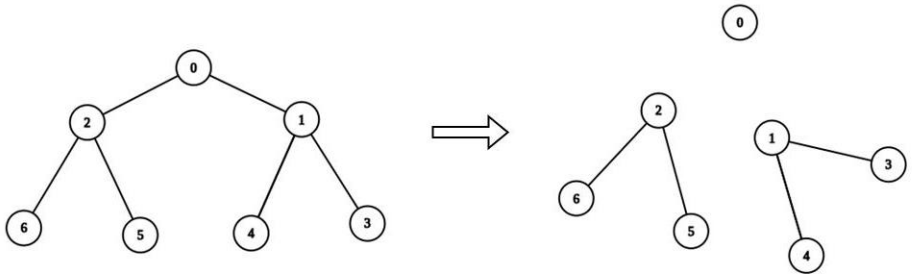
Output:

2

Explanation:

We remove the edge connecting node 1 with 2. The resulting split is valid because: - The value of the component containing nodes 1 and 3 is values[1] + values[3] = 12. - The value of the component containing nodes 0, 2, and 4 is values[0] + values[2] + values[4] = 6. It can be shown that no other valid split has more than 2 connected components.

Example 2:



Input:

n = 7, edges = [[0,1],[0,2],[1,3],[1,4],[2,5],[2,6]], values = [3,0,6,1,5,2,1], k = 3

Output:

3

Explanation:

We remove the edge connecting node 0 with 2, and the edge connecting node 0 with 1. The resulting split is valid because: - The value of the component containing node 0 is values[0] = 3. - The value of the component containing nodes 2, 5, and 6 is values[2] + values[5] + values[6] = 9. - The value of the component containing nodes 1, 3, and 4 is values[1] + values[3] + values[4] = 6. It can be shown that no other valid split has more than 3 connected components.

Constraints:

1 <= n <= 3 * 10

4

edges.length == n - 1

edges[i].length == 2

0 <= a

i

, b

i

< n

values.length == n

0 <= values[i] <= 10

9

1 <= k <= 10

9

Sum of

values

is divisible by

k

.

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
int maxKDivisibleComponents(int n, vector<vector<int>>& edges, vector<int>&
values, int k) {
```

```
}
};
```

Java:

```
class Solution {
public int maxKDivisibleComponents(int n, int[][] edges, int[] values, int k)
{

}
}
```

Python3:

```
class Solution:
def maxKDivisibleComponents(self, n: int, edges: List[List[int]], values:
List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def maxKDivisibleComponents(self, n, edges, values, k):
"""
:type n: int
:type edges: List[List[int]]
:type values: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number[]} values
* @param {number} k
* @return {number}
*/
var maxKDivisibleComponents = function(n, edges, values, k) {
```

```
};
```

TypeScript:

```
function maxKDivisibleComponents(n: number, edges: number[][], values:
number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxKDivisibleComponents(int n, int[][] edges, int[] values, int k)
{

}
}
```

C:

```
int maxKDivisibleComponents(int n, int** edges, int edgesSize, int*
edgesColSize, int* values, int valuesSize, int k) {

}
```

Go:

```
func maxKDivisibleComponents(n int, edges [][]int, values []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxKDivisibleComponents(n: Int, edges: Array<IntArray>, values: IntArray,
k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxKDivisibleComponents(_ n: Int, _ edges: [[Int]], _ values: [Int], _
k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_k_divisible_components(n: i32, edges: Vec<Vec<i32>>, values:
Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer[]} values
# @param {Integer} k
# @return {Integer}
def max_k_divisible_components(n, edges, values, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $values
* @param Integer $k
* @return Integer
*/
function maxKDivisibleComponents($n, $edges, $values, $k) {

}
}
```

Dart:

```
class Solution {
int maxKDivisibleComponents(int n, List<List<int>> edges, List<int> values,
int k) {

}
}
```

Scala:

```
object Solution {
def maxKDivisibleComponents(n: Int, edges: Array[Array[Int]], values:
Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_k_divisible_components(n :: integer, edges :: [[integer]], values
:: [integer], k :: integer) :: integer
def max_k_divisible_components(n, edges, values, k) do

end
end
```

Erlang:

```
-spec max_k_divisible_components(N :: integer(), Edges :: [[integer()]],
Values :: [integer()], K :: integer()) -> integer().
max_k_divisible_components(N, Edges, Values, K) ->
.
```

Racket:

```
(define/contract (max-k-divisible-components n edges values k)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of K-Divisible Components
* 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 maxKDivisibleComponents(int n, vector<vector<int>>& edges, vector<int>&
values, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of K-Divisible Components
* 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 maxKDivisibleComponents(int n, int[][] edges, int[] values, int k)
{

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of K-Divisible Components
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 maxKDivisibleComponents(self, n: int, edges: List[List[int]], values:
List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxKDivisibleComponents(self, n, edges, values, k):
"""
:type n: int
:type edges: List[List[int]]
:type values: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of K-Divisible Components
* 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} n
```

```
* @param {number[][]} edges
* @param {number[]} values
* @param {number} k
* @return {number}
*/
var maxKDivisibleComponents = function(n, edges, values, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of K-Divisible Components
* 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 maxKDivisibleComponents(n: number, edges: number[][], values:
number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of K-Divisible Components
* 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 MaxKDivisibleComponents(int n, int[][] edges, int[] values, int k)
{
```

```
}
}
```

C Solution:

```
/*
* Problem: Maximum Number of K-Divisible Components
* 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 maxKDivisibleComponents(int n, int** edges, int edgesSize, int*
edgesColSize, int* values, int valuesSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Number of K-Divisible Components
// 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 maxKDivisibleComponents(n int, edges [][]int, values []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxKDivisibleComponents(n: Int, edges: Array<IntArray>, values: IntArray,
k: Int): Int {
```

```
}
}
```

Swift Solution:

```
class Solution {
func maxKDivisibleComponents(_ n: Int, _ edges: [[Int]], _ values: [Int], _
k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of K-Divisible Components
// 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 max_k_divisible_components(n: i32, edges: Vec<Vec<i32>>, values:
Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer[]} values
# @param {Integer} k
# @return {Integer}
def max_k_divisible_components(n, edges, values, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $values
* @param Integer $k
* @return Integer
*/
function maxKDivisibleComponents($n, $edges, $values, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxKDivisibleComponents(int n, List<List<int>> edges, List<int> values,
int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxKDivisibleComponents(n: Int, edges: Array[Array[Int]], values:
Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_k_divisible_components(n :: integer, edges :: [[integer]], values
:: [integer], k :: integer) :: integer
def max_k_divisible_components(n, edges, values, k) do

end
end
```

Erlang Solution:

```
-spec max_k_divisible_components(N :: integer(), Edges :: [[integer()]],
Values :: [integer()], K :: integer()) -> integer().
max_k_divisible_components(N, Edges, Values, K) ->
.
```

Racket Solution:

```
(define/contract (max-k-divisible-components n edges values k)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer? exact-integer?)
)
```