Problem 2925: Maximum Score After Applying Operations on a Tree

Problem Information

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

Problem Description

There is an undirected tree with

n

nodes labeled from

0

to

n - 1

, and rooted at node

0

. You are 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 also given a

0-indexed

integer array

values

of length

n

, where

values[i]

is the

value

associated with the

i

th

node.

You start with a score of

0

. In one operation, you can:

Pick any node

i

.

Add

values[i]

to your score.

Set

values[i]

to

0

.

A tree is

healthy

if the sum of values on the path from the root to any leaf node is different than zero.

Return

the

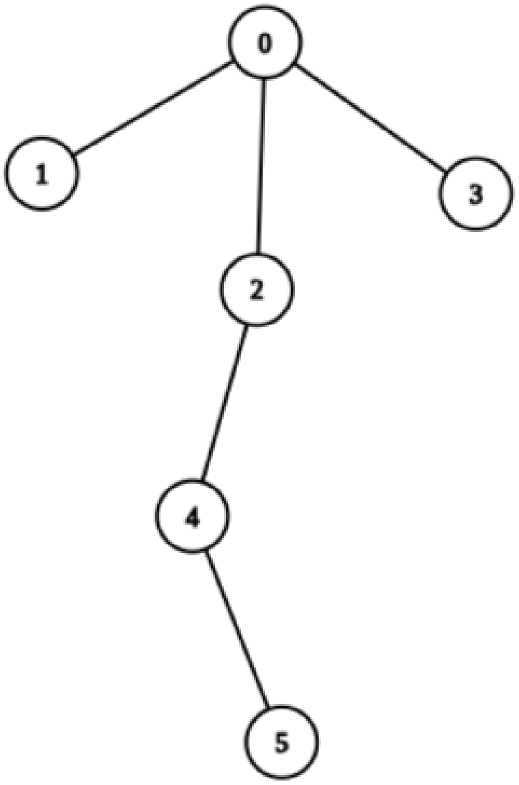
maximum score

you can obtain after performing these operations on the tree any number of times so that it remains

healthy

.

Example 1:



Input:

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

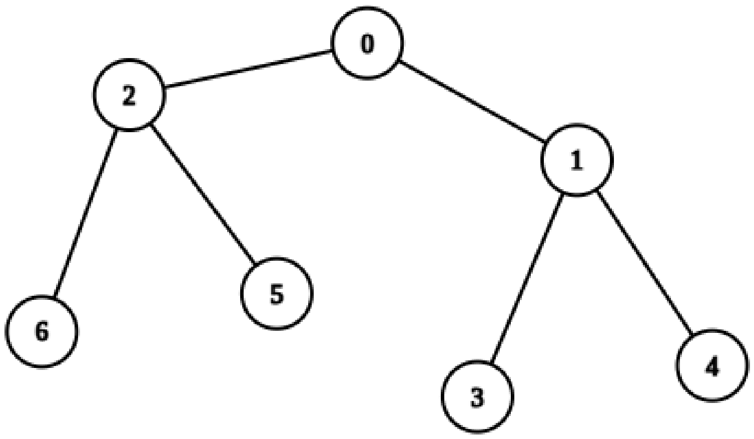
Output:

11

Explanation:

We can choose nodes 1, 2, 3, 4, and 5. The value of the root is non-zero. Hence, the sum of values on the path from the root to any leaf is different than zero. Therefore, the tree is healthy and the score is values[1] + values[2] + values[3] + values[4] + values[5] = 11. It can be shown that 11 is the maximum score obtainable after any number of operations on the tree.

Example 2:



Input:

edges = [[0,1],[0,2],[1,3],[1,4],[2,5],[2,6]], values = [20,10,9,7,4,3,5]

Output:

40

Explanation:

We can choose nodes 0, 2, 3, and 4. - The sum of values on the path from 0 to 4 is equal to 10. - The sum of values on the path from 0 to 3 is equal to 10. - The sum of values on the path from 0 to 5 is equal to 3. - The sum of values on the path from 0 to 6 is equal to 5. Therefore, the tree is healthy and the score is values[0] + values[2] + values[3] + values[4] = 40. It can be shown that 40 is the maximum score obtainable after any number of operations on the tree.

Constraints:

2 <= n <= 2 * 10

4

edges.length == n - 1

edges[i].length == 2

0 <= a

i

, b

i

< n

values.length == n

1 <= values[i] <= 10

9

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
long long maximumScoreAfterOperations(vector<vector<int>>& edges,
vector<int>& values) {

}
};
```

Java:

```
class Solution {
public long maximumScoreAfterOperations(int[][] edges, int[] values) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumScoreAfterOperations(edges: number[][], values: number[]):
number {

};
```

C#:

```
public class Solution {
public long MaximumScoreAfterOperations(int[][] edges, int[] values) {

}
}
```

C:

```
long long maximumScoreAfterOperations(int** edges, int edgesSize, int*
edgesColSize, int* values, int valuesSize) {

}
```

Go:

```
func maximumScoreAfterOperations(edges [][]int, values []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumScoreAfterOperations(edges: Array<IntArray>, values: IntArray):
Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn maximum_score_after_operations(edges: Vec<Vec<i32>>, values: Vec<i32>)
-> i64 {

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int maximumScoreAfterOperations(List<List<int>> edges, List<int> values) {

}
}
```

Scala:

```
object Solution {
def maximumScoreAfterOperations(edges: Array[Array[Int]], values:
Array[Int]): Long = {

}
}
```

Elixir:

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

end
```

```
end
```

Erlang:

```
-spec maximum_score_after_operations(Edges :: [[integer()]], Values ::
[integer()]) -> integer().
maximum_score_after_operations(Edges, Values) ->
.
```

Racket:

```
(define/contract (maximum-score-after-operations edges values)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Score After Applying Operations on a Tree
* 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:
long long maximumScoreAfterOperations(vector<vector<int>>& edges,
vector<int>& values) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Score After Applying Operations on a Tree
* 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 long maximumScoreAfterOperations(int[][] edges, int[] values) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Score After Applying Operations on a Tree
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 maximumScoreAfterOperations(self, edges: List[List[int]], values:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Score After Applying Operations on a Tree
* 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[]} values
* @return {number}
*/
var maximumScoreAfterOperations = function(edges, values) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Score After Applying Operations on a Tree
* 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 maximumScoreAfterOperations(edges: number[][], values: number[]):
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Score After Applying Operations on a Tree
* 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 long MaximumScoreAfterOperations(int[][] edges, int[] values) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Score After Applying Operations on a Tree
* 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
*/

long long maximumScoreAfterOperations(int** edges, int edgesSize, int*
edgesColSize, int* values, int valuesSize) {

}
```

Go Solution:

```
// Problem: Maximum Score After Applying Operations on a Tree
// 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 maximumScoreAfterOperations(edges [][]int, values []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumScoreAfterOperations(edges: Array<IntArray>, values: IntArray):
Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Score After Applying Operations on a Tree
// 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_score_after_operations(edges: Vec<Vec<i32>>, values: Vec<i32>)
-> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int maximumScoreAfterOperations(List<List<int>> edges, List<int> values) {

}
}
```

Scala Solution:

```
object Solution {
def maximumScoreAfterOperations(edges: Array[Array[Int]], values:
Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_score_after_operations(edges :: [[integer]], values ::
```

```
[integer]) :: integer
def maximum_score_after_operations(edges, values) do

end
end
```

Erlang Solution:

```
-spec maximum_score_after_operations(Edges :: [[integer()]], Values ::
[integer()]) -> integer().
maximum_score_after_operations(Edges, Values) ->
.
```

Racket Solution:

```
(define/contract (maximum-score-after-operations edges values)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?)
)
```