Problem 1273: Delete Tree Nodes

Problem Information

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

Problem Description

A tree rooted at node 0 is given as follows:

The number of nodes is

nodes

;

The value of the

i

th

node is

value[i]

;

The parent of the

i

th

node is

parent[i]

.

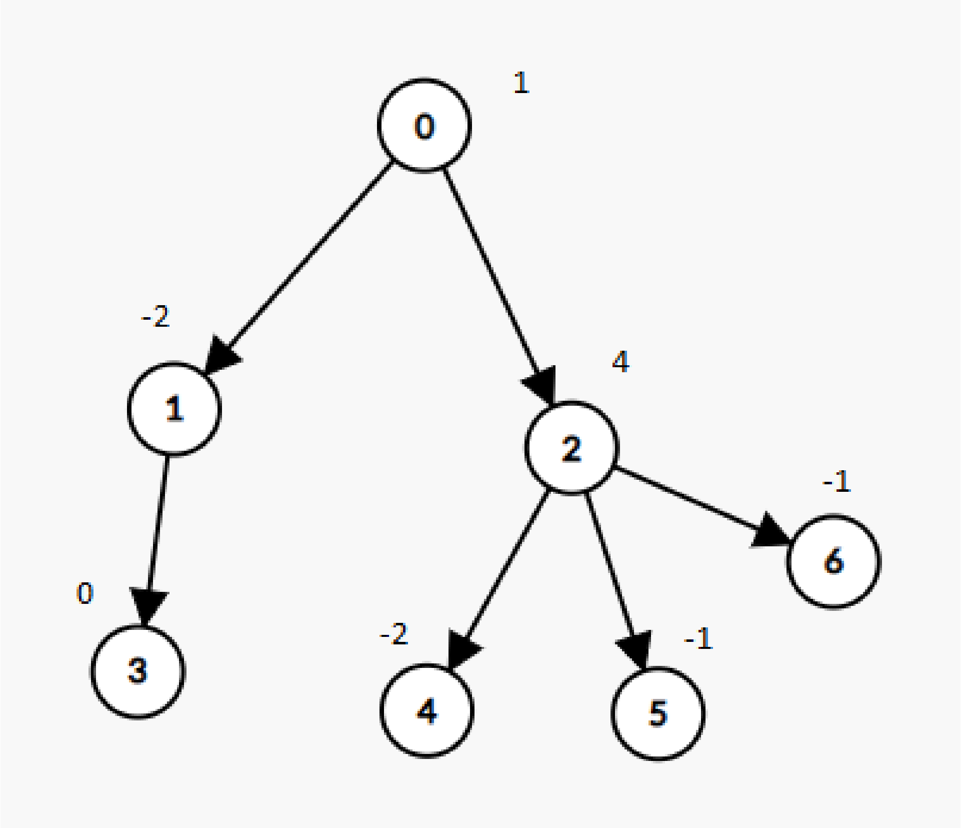
Remove every subtree whose sum of values of nodes is zero.

Return

the number of the remaining nodes in the tree

.

Example 1:



Input:

nodes = 7, parent = [-1,0,0,1,2,2,2], value = [1,-2,4,0,-2,-1,-1]

Output:

2

Example 2:

Input:

nodes = 7, parent = [-1,0,0,1,2,2,2], value = [1,-2,4,0,-2,-1,-2]

Output:

6

Constraints:

1 <= nodes <= 10

4

parent.length == nodes

0 <= parent[i] <= nodes - 1

parent[0] == -1

which indicates that

0

is the root.

value.length == nodes

-10

5

<= value[i] <= 10

5

The given input is

guaranteed

to represent a

valid tree

.

Code Snippets

C++:

```
class Solution {
public:
int deleteTreeNodes(int nodes, vector<int>& parent, vector<int>& value) {

}
};
```

Java:

```
class Solution {
public int deleteTreeNodes(int nodes, int[] parent, int[] value) {

}
}
```

Python3:

```
class Solution:
def deleteTreeNodes(self, nodes: int, parent: List[int], value: List[int]) ->
int:
```

Python:

```
class Solution(object):
def deleteTreeNodes(self, nodes, parent, value):
"""
:type nodes: int
:type parent: List[int]
:type value: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} nodes
* @param {number[]} parent
* @param {number[]} value
* @return {number}
*/
var deleteTreeNodes = function(nodes, parent, value) {

};
```

TypeScript:

```
function deleteTreeNodes(nodes: number, parent: number[], value: number[]):
number {

};
```

C#:

```
public class Solution {
public int DeleteTreeNodes(int nodes, int[] parent, int[] value) {

}
}
```

C:

```
int deleteTreeNodes(int nodes, int* parent, int parentSize, int* value, int
valueSize) {

}
```

Go:

```
func deleteTreeNodes(nodes int, parent []int, value []int) int {

}
```

Kotlin:

```
class Solution {
fun deleteTreeNodes(nodes: Int, parent: IntArray, value: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func deleteTreeNodes(_ nodes: Int, _ parent: [Int], _ value: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn delete_tree_nodes(nodes: i32, parent: Vec<i32>, value: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} nodes
# @param {Integer[]} parent
# @param {Integer[]} value
# @return {Integer}
def delete_tree_nodes(nodes, parent, value)

end
```

PHP:

```
class Solution {

/**
* @param Integer $nodes
* @param Integer[] $parent
* @param Integer[] $value
* @return Integer
*/
function deleteTreeNodes($nodes, $parent, $value) {

}
}
```

Dart:

```
class Solution {
int deleteTreeNodes(int nodes, List<int> parent, List<int> value) {

}
}
```

Scala:

```
object Solution {
def deleteTreeNodes(nodes: Int, parent: Array[Int], value: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec delete_tree_nodes(nodes :: integer, parent :: [integer], value :: [integer]) :: integer
def delete_tree_nodes(nodes, parent, value) do

end
end
```

Erlang:

```
-spec delete_tree_nodes(Nodes :: integer(), Parent :: [integer()], Value ::
[integer()]) -> integer().
delete_tree_nodes(Nodes, Parent, Value) ->
.
```

Racket:

```
(define/contract (delete-tree-nodes nodes parent value)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Delete Tree Nodes
* Difficulty: Medium
* 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 deleteTreeNodes(int nodes, vector<int>& parent, vector<int>& value) {

}
};
```

Java Solution:

```
/**
* Problem: Delete Tree Nodes
* Difficulty: Medium
* 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 deleteTreeNodes(int nodes, int[] parent, int[] value) {

}
}
```

Python3 Solution:

```
"""
Problem: Delete Tree Nodes
Difficulty: Medium
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 deleteTreeNodes(self, nodes: int, parent: List[int], value: List[int]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def deleteTreeNodes(self, nodes, parent, value):
"""
:type nodes: int
:type parent: List[int]
:type value: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Delete Tree Nodes
* Difficulty: Medium
* 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} nodes
* @param {number[]} parent
* @param {number[]} value
* @return {number}
*/
var deleteTreeNodes = function(nodes, parent, value) {

};
```

TypeScript Solution:

```
/**
* Problem: Delete Tree Nodes
* Difficulty: Medium
* 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 deleteTreeNodes(nodes: number, parent: number[], value: number[]):
number {

};
```

C# Solution:

```
/*
* Problem: Delete Tree Nodes
* Difficulty: Medium
* 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 DeleteTreeNodes(int nodes, int[] parent, int[] value) {

}
}
```

C Solution:

```
/*
* Problem: Delete Tree Nodes
* Difficulty: Medium
* 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 deleteTreeNodes(int nodes, int* parent, int parentSize, int* value, int
valueSize) {

}
```

Go Solution:

```
// Problem: Delete Tree Nodes
// Difficulty: Medium
// 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 deleteTreeNodes(nodes int, parent []int, value []int) int {
}
```

Kotlin Solution:

```
class Solution {
fun deleteTreeNodes(nodes: Int, parent: IntArray, value: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func deleteTreeNodes(_ nodes: Int, _ parent: [Int], _ value: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Delete Tree Nodes
// Difficulty: Medium
// 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 delete_tree_nodes(nodes: i32, parent: Vec<i32>, value: Vec<i32>) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} nodes
# @param {Integer[]} parent
# @param {Integer[]} value
# @return {Integer}
def delete_tree_nodes(nodes, parent, value)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $nodes
* @param Integer[] $parent
* @param Integer[] $value
* @return Integer
*/
function deleteTreeNodes($nodes, $parent, $value) {

}
}
```

Dart Solution:

```
class Solution {
int deleteTreeNodes(int nodes, List<int> parent, List<int> value) {

}
}
```

Scala Solution:

```
object Solution {
def deleteTreeNodes(nodes: Int, parent: Array[Int], value: Array[Int]): Int =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec delete_tree_nodes(nodes :: integer, parent :: [integer], value ::
```

```
[integer]) :: integer
def delete_tree_nodes(nodes, parent, value) do

end
end
```

Erlang Solution:

```
-spec delete_tree_nodes(Nodes :: integer(), Parent :: [integer()], Value ::
[integer()]) -> integer().
delete_tree_nodes(Nodes, Parent, Value) ->
.
```

Racket Solution:

```
(define/contract (delete-tree-nodes nodes parent value)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```