Problem 2003: Smallest Missing Genetic Value in Each Subtree

Problem Information

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

Problem Description

There is a

family tree

rooted at

0

consisting of

n

nodes numbered

0

to

n - 1

. You are given a

0-indexed

integer array

parents

, where

parents[i]

is the parent for node

i

. Since node

0

is the

root

,

parents[0] == -1

.

There are

10

5

genetic values, each represented by an integer in the

inclusive

range

[1, 10

5

]

. You are given a

0-indexed

integer array

nums

, where

nums[i]

is a

distinct

genetic value for node

i

.

Return

an array

ans

of length

n

where

ans[i]

is

the

smallest

genetic value that is

missing

from the subtree rooted at node

i

.

The

subtree

rooted at a node

x

contains node

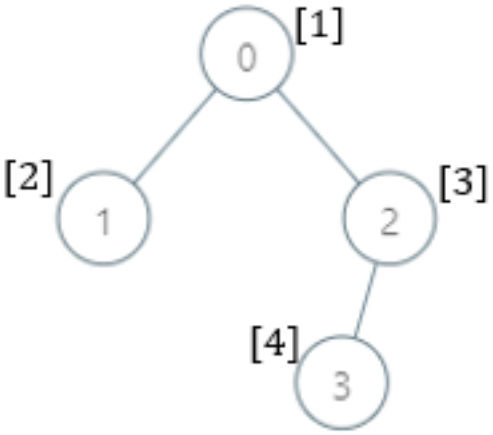
x

and all of its

descendant

nodes.

Example 1:



Input:

parents = [-1,0,0,2], nums = [1,2,3,4]

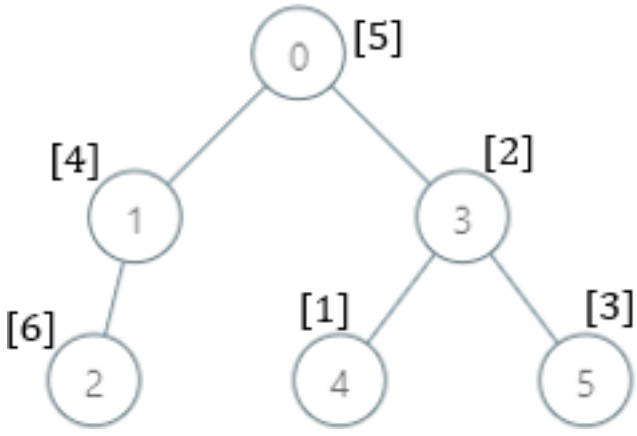
Output:

[5,1,1,1]

Explanation:

The answer for each subtree is calculated as follows: - 0: The subtree contains nodes [0,1,2,3] with values [1,2,3,4]. 5 is the smallest missing value. - 1: The subtree contains only node 1 with value 2. 1 is the smallest missing value. - 2: The subtree contains nodes [2,3] with values [3,4]. 1 is the smallest missing value. - 3: The subtree contains only node 3 with value 4. 1 is the smallest missing value.

Example 2:



Input:

parents = [-1,0,1,0,3,3], nums = [5,4,6,2,1,3]

Output:

[7,1,1,4,2,1]

Explanation:

The answer for each subtree is calculated as follows: - 0: The subtree contains nodes [0,1,2,3,4,5] with values [5,4,6,2,1,3]. 7 is the smallest missing value. - 1: The subtree contains nodes [1,2] with values [4,6]. 1 is the smallest missing value. - 2: The subtree contains only node 2 with value 6. 1 is the smallest missing value. - 3: The subtree contains nodes [3,4,5] with values [2,1,3]. 4 is the smallest missing value. - 4: The subtree contains only node 4 with value 1. 2 is the smallest missing value. - 5: The subtree contains only node 5 with value 3. 1 is the smallest missing value.

Example 3:

Input:

parents = [-1,2,3,0,2,4,1], nums = [2,3,4,5,6,7,8]

Output:

[1,1,1,1,1,1,1]

Explanation:

The value 1 is missing from all the subtrees.

Constraints:

n == parents.length == nums.length

2 <= n <= 10

5

0 <= parents[i] <= n - 1

for

i != 0

parents[0] == -1

parents

represents a valid tree.

1 <= nums[i] <= 10

5

Each

nums[i]

is distinct.

Code Snippets

C++:

```
class Solution {
public:
vector<int> smallestMissingValueSubtree(vector<int>& parents, vector<int>&
nums) {

}
};
```

Java:

```
class Solution {
public int[] smallestMissingValueSubtree(int[] parents, int[] nums) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function smallestMissingValueSubtree(parents: number[], nums: number[]):
number[] {

};
```

C#:

```
public class Solution {
public int[] SmallestMissingValueSubtree(int[] parents, int[] nums) {
```

```
}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* smallestMissingValueSubtree(int* parents, int parentsSize, int* nums,
int numsSize, int* returnSize) {

}
```

Go:

```
func smallestMissingValueSubtree(parents []int, nums []int) []int {

}
```

Kotlin:

```
class Solution {
fun smallestMissingValueSubtree(parents: IntArray, nums: IntArray): IntArray
{

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

```
# @param {Integer[]} parents
# @param {Integer[]} nums
# @return {Integer[]}
def smallest_missing_value_subtree(parents, nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $parents
* @param Integer[] $nums
* @return Integer[]
*/
function smallestMissingValueSubtree($parents, $nums) {

}
}
```

Dart:

```
class Solution {
List<int> smallestMissingValueSubtree(List<int> parents, List<int> nums) {

}
}
```

Scala:

```
object Solution {
def smallestMissingValueSubtree(parents: Array[Int], nums: Array[Int]):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec smallest_missing_value_subtree(parents :: [integer], nums :: [integer])
:: [integer]
def smallest_missing_value_subtree(parents, nums) do

end
end
```

Erlang:

```
-spec smallest_missing_value_subtree(Parents :: [integer()], Nums ::
[integer()]) -> [integer()].
smallest_missing_value_subtree(Parents, Nums) ->
.
```

Racket:

```
(define/contract (smallest-missing-value-subtree parents nums)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Missing Genetic Value in Each Subtree
* Difficulty: Hard
* 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:
vector<int> smallestMissingValueSubtree(vector<int>& parents, vector<int>&
nums) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Smallest Missing Genetic Value in Each Subtree
* Difficulty: Hard
* 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 int[] smallestMissingValueSubtree(int[] parents, int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Missing Genetic Value in Each Subtree
Difficulty: Hard
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 smallestMissingValueSubtree(self, parents: List[int], nums: List[int]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Smallest Missing Genetic Value in Each Subtree
* Difficulty: Hard
* 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[]} parents
* @param {number[]} nums
* @return {number[]}
*/
var smallestMissingValueSubtree = function(parents, nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Missing Genetic Value in Each Subtree
* Difficulty: Hard
* 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 smallestMissingValueSubtree(parents: number[], nums: number[]):
```

```
number[] {

};
```

C# Solution:

```
/*
* Problem: Smallest Missing Genetic Value in Each Subtree
* Difficulty: Hard
* 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 int[] SmallestMissingValueSubtree(int[] parents, int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Missing Genetic Value in Each Subtree
* Difficulty: Hard
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* smallestMissingValueSubtree(int* parents, int parentsSize, int* nums,
int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Smallest Missing Genetic Value in Each Subtree
// Difficulty: Hard
// 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 smallestMissingValueSubtree(parents []int, nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun smallestMissingValueSubtree(parents: IntArray, nums: IntArray): IntArray
{

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Smallest Missing Genetic Value in Each Subtree
// Difficulty: Hard
// 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 smallest_missing_value_subtree(parents: Vec<i32>, nums: Vec<i32>) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} parents
# @param {Integer[]} nums
# @return {Integer[]}
def smallest_missing_value_subtree(parents, nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $parents
* @param Integer[] $nums
* @return Integer[]
*/
function smallestMissingValueSubtree($parents, $nums) {

}
}
```

Dart Solution:

```
class Solution {
List<int> smallestMissingValueSubtree(List<int> parents, List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def smallestMissingValueSubtree(parents: Array[Int], nums: Array[Int]):
```

```
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec smallest_missing_value_subtree(parents :: [integer], nums :: [integer])
:: [integer]
def smallest_missing_value_subtree(parents, nums) do

end
end
```

Erlang Solution:

```
-spec smallest_missing_value_subtree(Parents :: [integer()], Nums ::
[integer()]) -> [integer()].
smallest_missing_value_subtree(Parents, Nums) ->
.
```

Racket Solution:

```
(define/contract (smallest-missing-value-subtree parents nums)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```