Problem 2049: Count Nodes With the Highest Score

Problem Information

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

Problem Description

There is a

binary

tree rooted at

0

consisting of

n

nodes. The nodes are labeled from

0

to

n - 1

. You are given a

0-indexed

integer array

parents

representing the tree, where

parents[i]

is the parent of node

i

. Since node

0

is the root,

parents[0] == -1

.

Each node has a

score

. To find the score of a node, consider if the node and the edges connected to it were

removed

. The tree would become one or more

non-empty

subtrees. The

size

of a subtree is the number of the nodes in it. The score

of the node is the

product of the sizes

of all those subtrees.

Return

the

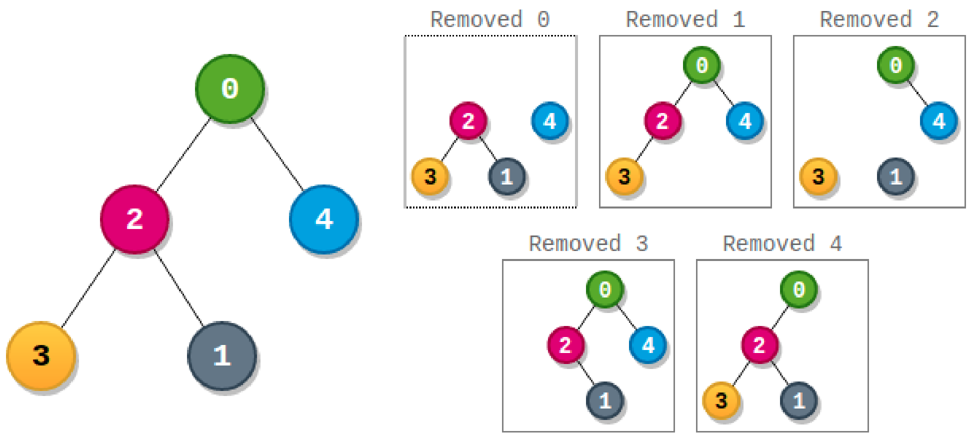
number

of nodes that have the

highest score

.

Example 1:



Input:

parents = [-1,2,0,2,0]

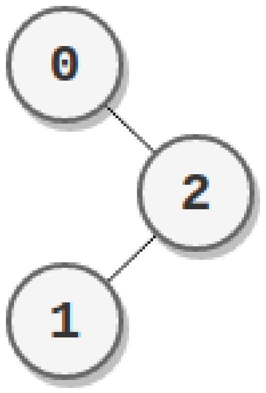
Output:

3

Explanation:

- The score of node 0 is: 3 * 1 = 3 - The score of node 1 is: 4 = 4 - The score of node 2 is: 1 * 1 * 2 = 2 - The score of node 3 is: 4 = 4 - The score of node 4 is: 4 = 4 The highest score is 4, and three nodes (node 1, node 3, and node 4) have the highest score.

Example 2:



Input:

parents = [-1,2,0]

Output:

2

Explanation:

- The score of node 0 is: 2 = 2 - The score of node 1 is: 2 = 2 - The score of node 2 is: 1 * 1 = 1 The highest score is 2, and two nodes (node 0 and node 1) have the highest score.

Constraints:

n == parents.length

2 <= n <= 10

5

parents[0] == -1

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

for

i != 0

parents

represents a valid binary tree.

Code Snippets

C++:

```
class Solution {
public:
int countHighestScoreNodes(vector<int>& parents) {

}
};
```

Java:

```
class Solution {
public int countHighestScoreNodes(int[] parents) {

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function countHighestScoreNodes(parents: number[]): number {

};
```

C#:

```
public class Solution {
public int CountHighestScoreNodes(int[] parents) {

}
}
```

C:

```
int countHighestScoreNodes(int* parents, int parentsSize) {

}
```

Go:

```
func countHighestScoreNodes(parents []int) int {

}
```

Kotlin:

```
class Solution {
fun countHighestScoreNodes(parents: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func countHighestScoreNodes(_ parents: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_highest_score_nodes(parents: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} parents
# @return {Integer}
def count_highest_score_nodes(parents)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $parents
* @return Integer
*/
function countHighestScoreNodes($parents) {

}
}
```

Dart:

```
class Solution {
int countHighestScoreNodes(List<int> parents) {

}
}
```

Scala:

```
object Solution {
def countHighestScoreNodes(parents: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_highest_score_nodes(parents :: [integer]) :: integer
def count_highest_score_nodes(parents) do

end
end
```

Erlang:

```
-spec count_highest_score_nodes(Parents :: [integer()]) -> integer().
count_highest_score_nodes(Parents) ->
.
```

Racket:

```
(define/contract (count-highest-score-nodes parents)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Nodes With the Highest Score
* 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 countHighestScoreNodes(vector<int>& parents) {

}
};
```

Java Solution:

```
/**
* Problem: Count Nodes With the Highest Score
* 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 countHighestScoreNodes(int[] parents) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Nodes With the Highest Score
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 countHighestScoreNodes(self, parents: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Nodes With the Highest Score
* 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[]} parents
* @return {number}
*/
var countHighestScoreNodes = function(parents) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Nodes With the Highest Score
* 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 countHighestScoreNodes(parents: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Nodes With the Highest Score
* 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 CountHighestScoreNodes(int[] parents) {

}
}
```

C Solution:

```
/*
* Problem: Count Nodes With the Highest Score
* 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 countHighestScoreNodes(int* parents, int parentsSize) {

}
```

Go Solution:

```
// Problem: Count Nodes With the Highest Score
// 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 countHighestScoreNodes(parents []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countHighestScoreNodes(parents: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func countHighestScoreNodes(_ parents: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Nodes With the Highest Score
// 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 count_highest_score_nodes(parents: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} parents
# @return {Integer}
def count_highest_score_nodes(parents)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $parents
* @return Integer
*/
function countHighestScoreNodes($parents) {

}
}
```

Dart Solution:

```
class Solution {
int countHighestScoreNodes(List<int> parents) {

}
}
```

Scala Solution:

```
object Solution {
def countHighestScoreNodes(parents: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_highest_score_nodes(parents :: [integer]) :: integer
def count_highest_score_nodes(parents) do

end
end
```

Erlang Solution:

```
-spec count_highest_score_nodes(Parents :: [integer()]) -> integer().
count_highest_score_nodes(Parents) ->
.
```

Racket Solution:

```
(define/contract (count-highest-score-nodes parents)
(-> (listof exact-integer?) exact-integer?)
)
```