Problem 3486: Longest Special Path II

Problem Information

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

Problem Description

You are given an undirected tree rooted at node

0

, with

n

nodes numbered from

0

to

n - 1

. This is represented by a 2D array

edges

of length

n - 1

, where

edges[i] = [u

i

, v

i

, length

i

]

indicates an edge between nodes

u

i

and

v

i

with length

length

i

. You are also given an integer array

nums

, where

nums[i]

represents the value at node

i

.

A

special path

is defined as a

downward

path from an ancestor node to a descendant node in which all node values are

distinct

, except for

at most

one value that may appear twice.

Return an array

result

of size 2, where

result[0]

is the

length

of the

longest

special path, and

result[1]

is the

minimum

number of nodes in all

possible

longest

special paths.

Example 1:

Input:

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

Output:

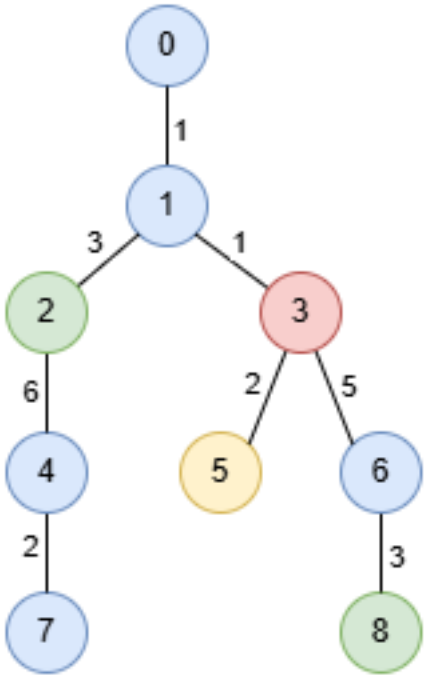
[9,3]

Explanation:

In the image below, nodes are colored by their corresponding values in

nums

.



The longest special paths are

1 -> 2 -> 4

and

1 -> 3 -> 6 -> 8

, both having a length of 9. The minimum number of nodes across all longest special paths is 3.

Example 2:

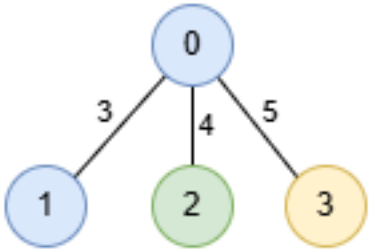
Input:

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

Output:

[5,2]

Explanation:



The longest path is

0 -> 3

consisting of 2 nodes with a length of 5.

Constraints:

2 <= n <= 5 * 10

4

edges.length == n - 1

edges[i].length == 3

0 <= u

i

, v

i

< n

1 <= length

i

<= 10

3

nums.length == n

0 <= nums[i] <= 5 * 10

4

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
vector<int> longestSpecialPath(vector<vector<int>>& edges, vector<int>& nums)
{

}
};
```

Java:

```
class Solution {
public int[] longestSpecialPath(int[][] edges, int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function longestSpecialPath(edges: number[][], nums: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] LongestSpecialPath(int[][] edges, int[] nums) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* longestSpecialPath(int** edges, int edgesSize, int* edgesColSize, int*
nums, int numsSize, int* returnSize) {
```

```
}
```

Go:

```
func longestSpecialPath(edges [][]int, nums []int) []int {

}
```

Kotlin:

```
class Solution {
fun longestSpecialPath(edges: Array<IntArray>, nums: IntArray): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer[]} nums
# @return {Integer[]}
def longest_special_path(edges, nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $nums
* @return Integer[]
*/
function longestSpecialPath($edges, $nums) {

}
}
```

Dart:

```
class Solution {
List<int> longestSpecialPath(List<List<int>> edges, List<int> nums) {

}
}
```

Scala:

```
object Solution {
def longestSpecialPath(edges: Array[Array[Int]], nums: Array[Int]):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_special_path(edges :: [[integer]], nums :: [integer]) ::
[integer]
def longest_special_path(edges, nums) do

end
end
```

Erlang:

```
-spec longest_special_path(Edges :: [[integer()]], Nums :: [integer()]) ->
[integer()].
longest_special_path(Edges, Nums) ->
.
```

Racket:

```
(define/contract (longest-special-path edges nums)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Special Path II
* Difficulty: Hard
* Tags: array, tree, hash, 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:
vector<int> longestSpecialPath(vector<vector<int>>& edges, vector<int>& nums)
{

}
};
```

Java Solution:

```
/**
* Problem: Longest Special Path II
* Difficulty: Hard
* Tags: array, tree, hash, 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[] longestSpecialPath(int[][] edges, int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Special Path II
Difficulty: Hard
Tags: array, tree, hash, 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 longestSpecialPath(self, edges: List[List[int]], nums: List[int]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Special Path II
* Difficulty: Hard
* Tags: array, tree, hash, 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[][]} edges
* @param {number[]} nums
* @return {number[]}
*/
var longestSpecialPath = function(edges, nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Special Path II
* Difficulty: Hard
* Tags: array, tree, hash, 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 longestSpecialPath(edges: number[][], nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Longest Special Path II
* Difficulty: Hard
* Tags: array, tree, hash, 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[] LongestSpecialPath(int[][] edges, int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Longest Special Path II
* Difficulty: Hard
* Tags: array, tree, hash, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* longestSpecialPath(int** edges, int edgesSize, int* edgesColSize, int*
nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Longest Special Path II
// Difficulty: Hard
// Tags: array, tree, hash, 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 longestSpecialPath(edges [][]int, nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun longestSpecialPath(edges: Array<IntArray>, nums: IntArray): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Special Path II
// Difficulty: Hard
// Tags: array, tree, hash, 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 longest_special_path(edges: Vec<Vec<i32>>, nums: Vec<i32>) -> Vec<i32>
{

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer[]} nums
```

```
# @return {Integer[]}
def longest_special_path(edges, nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $nums
* @return Integer[]
*/
function longestSpecialPath($edges, $nums) {

}
}
```

Dart Solution:

```
class Solution {
List<int> longestSpecialPath(List<List<int>> edges, List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def longestSpecialPath(edges: Array[Array[Int]], nums: Array[Int]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_special_path(edges :: [[integer]], nums :: [integer]) ::
[integer]
```

```
def longest_special_path(edges, nums) do

end
end
```

Erlang Solution:

```
-spec longest_special_path(Edges :: [[integer()]], Nums :: [integer()]) ->
[integer()].
longest_special_path(Edges, Nums) ->
.
```

Racket Solution:

```
(define/contract (longest-special-path edges nums)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```