Problem 2246: Longest Path With Different Adjacent Characters

Problem Information

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

Problem Description

You are given a

tree

(i.e. a connected, undirected graph that has no cycles)

rooted

at node

0

consisting of

n

nodes numbered from

0

to

n - 1

. The tree is represented by a

0-indexed

array

parent

of size

n

, where

parent[i]

is the parent of node

i

. Since node

0

is the root,

parent[0] == -1

.

You are also given a string

s

of length

n

, where

s[i]

is the character assigned to node

i

.

Return

the length of the

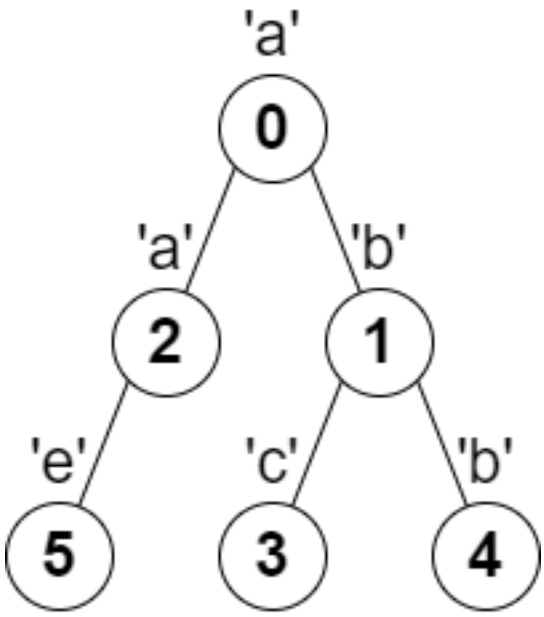
longest path

in the tree such that no pair of

adjacent

nodes on the path have the same character assigned to them.

Example 1:



Input:

parent = [-1,0,0,1,1,2], s = "abacbe"

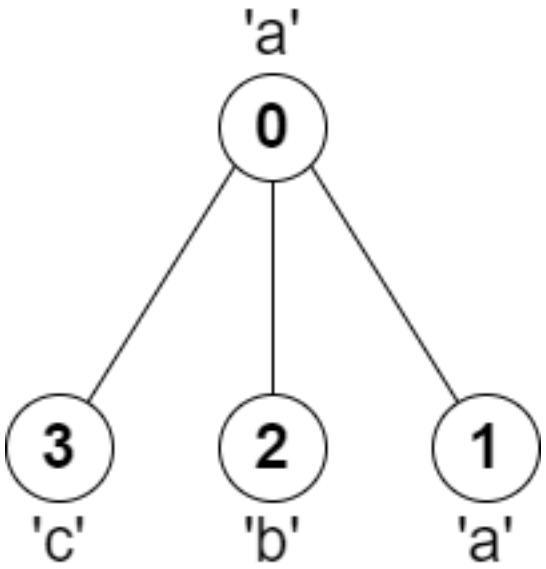
Output:

3

Explanation:

The longest path where each two adjacent nodes have different characters in the tree is the path: 0 -> 1 -> 3. The length of this path is 3, so 3 is returned. It can be proven that there is no longer path that satisfies the conditions.

Example 2:



Input:

parent = [-1,0,0,0], s = "aabc"

Output:

3

Explanation:

The longest path where each two adjacent nodes have different characters is the path: 2 -> 0 -> 3. The length of this path is 3, so 3 is returned.

Constraints:

n == parent.length == s.length

1 <= n <= 10

5

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

for all

i >= 1

parent[0] == -1

parent

represents a valid tree.

s

consists of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int longestPath(vector<int>& parent, string s) {

}
};
```

Java:

```
class Solution {
public int longestPath(int[] parent, String s) {
```

```
}
}
```

Python3:

```
class Solution:
def longestPath(self, parent: List[int], s: str) -> int:
```

Python:

```
class Solution(object):
def longestPath(self, parent, s):
"""
:type parent: List[int]
:type s: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} parent
* @param {string} s
* @return {number}
*/
var longestPath = function(parent, s) {

};
```

TypeScript:

```
function longestPath(parent: number[], s: string): number {

};
```

C#:

```
public class Solution {
public int LongestPath(int[] parent, string s) {

}
```

```
}
```

C:

```
int longestPath(int* parent, int parentSize, char* s) {

}
```

Go:

```
func longestPath(parent []int, s string) int {

}
```

Kotlin:

```
class Solution {
fun longestPath(parent: IntArray, s: String): Int {

}
}
```

Swift:

```
class Solution {
func longestPath(_ parent: [Int], _ s: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn longest_path(parent: Vec<i32>, s: String) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} parent
# @param {String} s
```

```
# @return {Integer}
def longest_path(parent, s)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $parent
* @param String $s
* @return Integer
*/
function longestPath($parent, $s) {

}
}
```

Dart:

```
class Solution {
int longestPath(List<int> parent, String s) {

}
}
```

Scala:

```
object Solution {
def longestPath(parent: Array[Int], s: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_path(parent :: [integer], s :: String.t) :: integer
def longest_path(parent, s) do

end
```

```
end
```

Erlang:

```
-spec longest_path(Parent :: [integer()], S :: unicode:unicode_binary()) ->
integer().
longest_path(Parent, S) ->
.
```

Racket:

```
(define/contract (longest-path parent s)
(-> (listof exact-integer?) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Path With Different Adjacent Characters
* Difficulty: Hard
* Tags: array, string, tree, graph, sort, 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 longestPath(vector<int>& parent, string s) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Path With Different Adjacent Characters
```

```
* Difficulty: Hard
* Tags: array, string, tree, graph, sort, 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 longestPath(int[] parent, String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Path With Different Adjacent Characters
Difficulty: Hard
Tags: array, string, tree, graph, sort, 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 longestPath(self, parent: List[int], s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def longestPath(self, parent, s):
"""
:type parent: List[int]
:type s: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Path With Different Adjacent Characters
* Difficulty: Hard
* Tags: array, string, tree, graph, sort, 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[]} parent
* @param {string} s
* @return {number}
*/
var longestPath = function(parent, s) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Path With Different Adjacent Characters
* Difficulty: Hard
* Tags: array, string, tree, graph, sort, 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 longestPath(parent: number[], s: string): number {

};
```

C# Solution:

```
/*
* Problem: Longest Path With Different Adjacent Characters
* Difficulty: Hard
* Tags: array, string, tree, graph, sort, 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 LongestPath(int[] parent, string s) {

}
}
```

C Solution:

```
/*
* Problem: Longest Path With Different Adjacent Characters
* Difficulty: Hard
* Tags: array, string, tree, graph, sort, 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 longestPath(int* parent, int parentSize, char* s) {

}
```

Go Solution:

```
// Problem: Longest Path With Different Adjacent Characters
// Difficulty: Hard
// Tags: array, string, tree, graph, sort, 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 longestPath(parent []int, s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestPath(parent: IntArray, s: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestPath(_ parent: [Int], _ s: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Path With Different Adjacent Characters
// Difficulty: Hard
// Tags: array, string, tree, graph, sort, 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_path(parent: Vec<i32>, s: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} parent
# @param {String} s
# @return {Integer}
def longest_path(parent, s)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $parent
* @param String $s
* @return Integer
*/
function longestPath($parent, $s) {

}
}
```

Dart Solution:

```
class Solution {
int longestPath(List<int> parent, String s) {

}
}
```

Scala Solution:

```
object Solution {
def longestPath(parent: Array[Int], s: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_path(parent :: [integer], s :: String.t) :: integer
def longest_path(parent, s) do

end
end
```

Erlang Solution:

```
-spec longest_path(Parent :: [integer()], S :: unicode:unicode_binary()) ->
integer().
longest_path(Parent, S) ->
.
```

Racket Solution:

```
(define/contract (longest-path parent s)
(-> (listof exact-integer?) string? exact-integer?)
)
```