Problem 3327: Check if DFS Strings Are Palindromes

Problem Information

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

Problem Description

You are given a tree rooted at node 0, consisting of

n

nodes numbered from

0

to

n - 1

. The tree is represented by an 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

.

Consider an empty string

dfsStr

, and define a recursive function

dfs(int x)

that takes a node

x

as a parameter and performs the following steps in order:

Iterate over each child

y

of

x

in increasing order of their numbers

, and call

dfs(y)

.

Add the character

s[x]

to the end of the string

dfsStr

.

Note

that

dfsStr

is shared across all recursive calls of

dfs

.

You need to find a boolean array

answer

of size

n

, where for each index

i

from

0

to

n - 1

, you do the following:

Empty the string

dfsStr

and call

dfs(i)

.

If the resulting string

dfsStr

is a

palindrome

, then set

answer[i]

to

true

. Otherwise, set

answer[i]

to

false

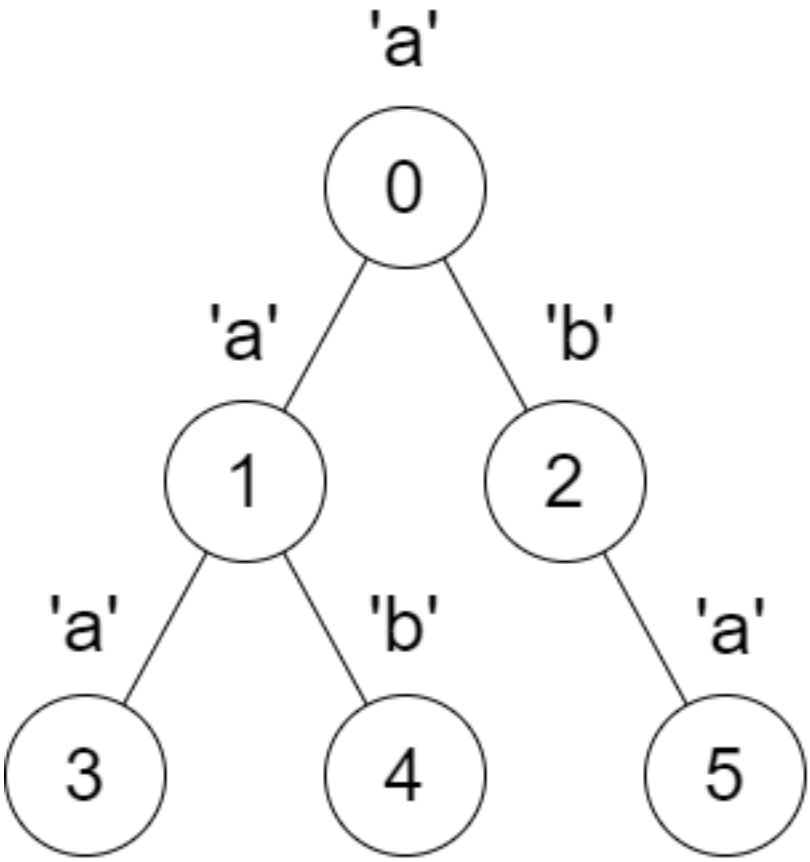
.

Return the array

answer

.

Example 1:



Input:

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

Output:

[true,true,false,true,true,true]

Explanation:

Calling

dfs(0)

results in the string

dfsStr = "abaaba"

, which is a palindrome.

Calling

dfs(1)

results in the string

dfsStr = "aba"

, which is a palindrome.

Calling

dfs(2)

results in the string

dfsStr = "ab"

, which is

not

a palindrome.

Calling

dfs(3)

results in the string

dfsStr = "a"

, which is a palindrome.

Calling

dfs(4)

results in the string

dfsStr = "b"

, which is a palindrome.

Calling

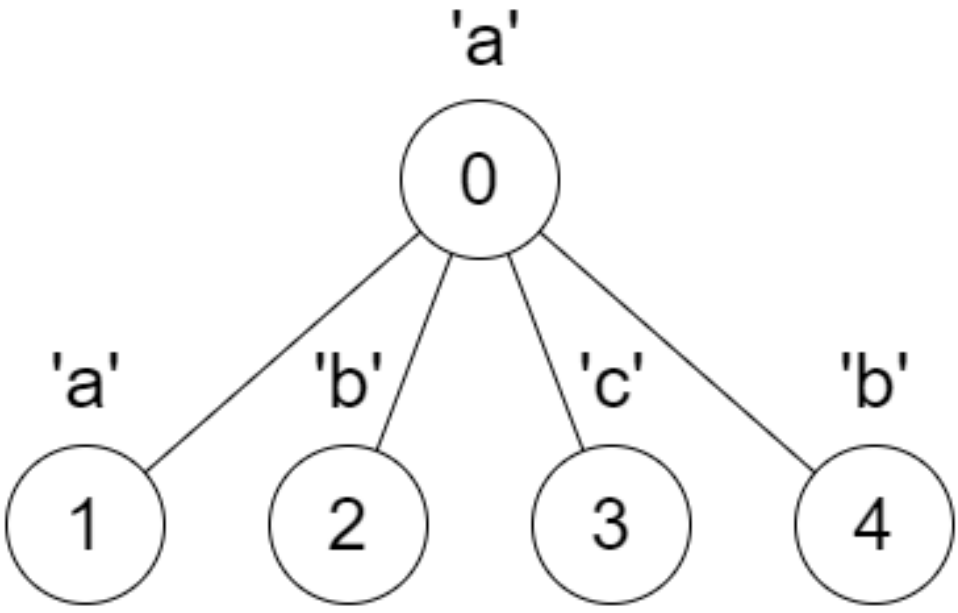
dfs(5)

results in the string

dfsStr = "a"

, which is a palindrome.

Example 2:



Input:

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

Output:

[true,true,true,true,true]

Explanation:

Every call on

dfs(x)

results in a palindrome string.

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 only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* findAnswer(int* parent, int parentSize, char* s, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun findAnswer(parent: IntArray, s: String): BooleanArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

```
}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-answer parent s)
(-> (listof exact-integer?) string? (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if DFS Strings Are Palindromes
* Difficulty: Hard
* Tags: array, string, 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<bool> findAnswer(vector<int>& parent, string s) {

}
};
```

Java Solution:

```
/**
* Problem: Check if DFS Strings Are Palindromes
* Difficulty: Hard
* Tags: array, string, 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 boolean[] findAnswer(int[] parent, String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if DFS Strings Are Palindromes
Difficulty: Hard
Tags: array, string, 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 findAnswer(self, parent: List[int], s: str) -> List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check if DFS Strings Are Palindromes
* Difficulty: Hard
* Tags: array, string, 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[]} parent
* @param {string} s
* @return {boolean[]}
*/
var findAnswer = function(parent, s) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if DFS Strings Are Palindromes
* Difficulty: Hard
* Tags: array, string, 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 findAnswer(parent: number[], s: string): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Check if DFS Strings Are Palindromes
* Difficulty: Hard
* Tags: array, string, 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 bool[] FindAnswer(int[] parent, string s) {

}
}
```

C Solution:

```
/*
* Problem: Check if DFS Strings Are Palindromes
* Difficulty: Hard
* Tags: array, string, 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().
*/
bool* findAnswer(int* parent, int parentSize, char* s, int* returnSize) {

}
```

Go Solution:

```
// Problem: Check if DFS Strings Are Palindromes
// Difficulty: Hard
// Tags: array, string, 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 findAnswer(parent []int, s string) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun findAnswer(parent: IntArray, s: String): BooleanArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check if DFS Strings Are Palindromes
// Difficulty: Hard
// Tags: array, string, 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 find_answer(parent: Vec<i32>, s: String) -> Vec<bool> {
```

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
```

```
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-answer parent s)
(-> (listof exact-integer?) string? (listof boolean?))
)
```