Problem 2791: Count Paths That Can Form a Palindrome in a Tree

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 the edge between

i

and

parent[i]

.

s[0]

can be ignored.

Return

the number of pairs of nodes

(u, v)

such that

u < v

and the characters assigned to edges on the path from

u

to

v

can be

rearranged

to form a

palindrome

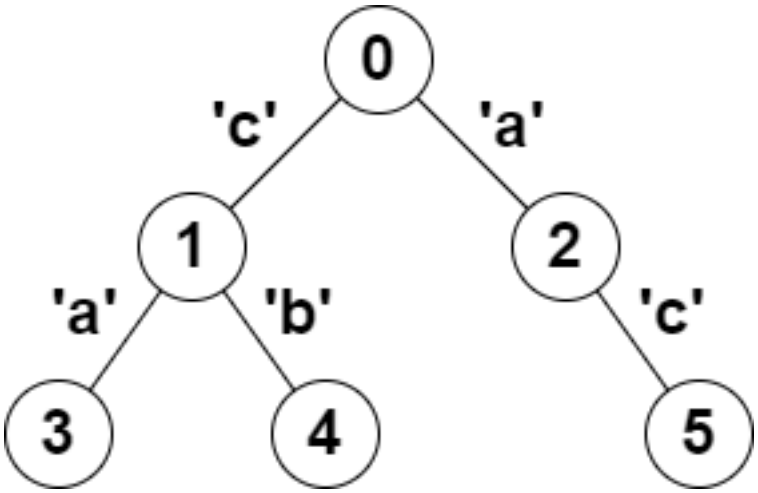
.

A string is a

palindrome

when it reads the same backwards as forwards.

Example 1:



Input:

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

Output:

8

Explanation:

The valid pairs are: - All the pairs (0,1), (0,2), (1,3), (1,4) and (2,5) result in one character which is always a palindrome. - The pair (2,3) result in the string "aca" which is a palindrome. - The pair (1,5) result in the string "cac" which is a palindrome. - The pair (3,5) result in the string "acac" which can be rearranged into the palindrome "acca".

Example 2:

Input:

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

Output:

10

Explanation:

Any pair of nodes (u,v) where u < v is valid.

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:
long long countPalindromePaths(vector<int>& parent, string s) {

}
};
```

Java:

```
class Solution {
public long countPalindromePaths(List<Integer> parent, String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long CountPalindromePaths(IList<int> parent, string s) {

}
}
```

C:

```
long long countPalindromePaths(int* parent, int parentSize, char* s) {

}
```

Go:

```
func countPalindromePaths(parent []int, s string) int64 {

}
```

Kotlin:

```
class Solution {
fun countPalindromePaths(parent: List<Int>, s: String): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn count_palindrome_paths(parent: Vec<i32>, s: String) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def countPalindromePaths(parent: List[Int], s: String): Long = {

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-palindrome-paths parent s)
(-> (listof exact-integer?) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Paths That Can Form a Palindrome in a Tree
* Difficulty: Hard
* Tags: array, string, 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:
```

```
long long countPalindromePaths(vector<int>& parent, string s) {

}
};
```

Java Solution:

```
/**
* Problem: Count Paths That Can Form a Palindrome in a Tree
* Difficulty: Hard
* Tags: array, string, 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 long countPalindromePaths(List<Integer> parent, String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Paths That Can Form a Palindrome in a Tree
Difficulty: Hard
Tags: array, string, 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 countPalindromePaths(self, parent: List[int], s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Paths That Can Form a Palindrome in a Tree
* Difficulty: Hard
* Tags: array, string, 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[]} parent
* @param {string} s
* @return {number}
*/
var countPalindromePaths = function(parent, s) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Paths That Can Form a Palindrome in a Tree
* Difficulty: Hard
* Tags: array, string, 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 countPalindromePaths(parent: number[], s: string): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Count Paths That Can Form a Palindrome in a Tree
* Difficulty: Hard
* Tags: array, string, 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 long CountPalindromePaths(IList<int> parent, string s) {

}
}
```

C Solution:

```
/*
* Problem: Count Paths That Can Form a Palindrome in a Tree
* Difficulty: Hard
* Tags: array, string, 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
*/

long long countPalindromePaths(int* parent, int parentSize, char* s) {

}
```

Go Solution:

```
// Problem: Count Paths That Can Form a Palindrome in a Tree
// Difficulty: Hard
```

```
// Tags: array, string, 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 countPalindromePaths(parent []int, s string) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countPalindromePaths(parent: List<Int>, s: String): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Paths That Can Form a Palindrome in a Tree
// Difficulty: Hard
// Tags: array, string, 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 count_palindrome_paths(parent: Vec<i32>, s: String) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def countPalindromePaths(parent: List[Int], s: String): Long = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-palindrome-paths parent s)
(-> (listof exact-integer?) string? exact-integer?)
)
```