# Problem 3331: Find Subtree Sizes After Changes

## Problem Information

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

## Problem Description

You are given a tree rooted at node 0 that consists 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

.

We make the following changes on the tree

one

time

simultaneously

for all nodes

x

from

1

to

n - 1

:

Find the

closest

node

y

to node

x

such that

y

is an ancestor of

x

, and

s[x] == s[y]

.

If node

y

does not exist, do nothing.

Otherwise,

remove

the edge between

x

and its current parent and make node

y

the new parent of

x

by adding an edge between them.

Return an array

answer

of size

n

where

answer[i]

is the

size

of the

subtree

rooted at node

i

in the

final

tree.

Example 1:

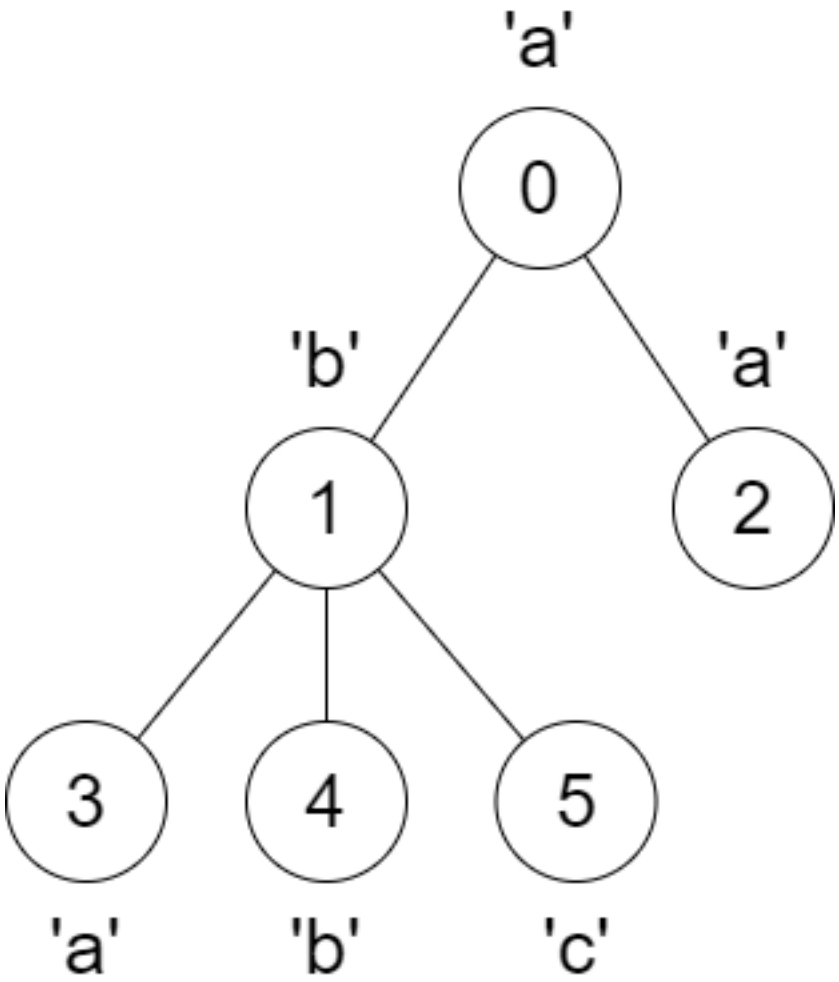
Input:

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

Output:

[6,3,1,1,1,1]

Explanation:




The parent of node 3 will change from node 1 to node 0.

Example 2:

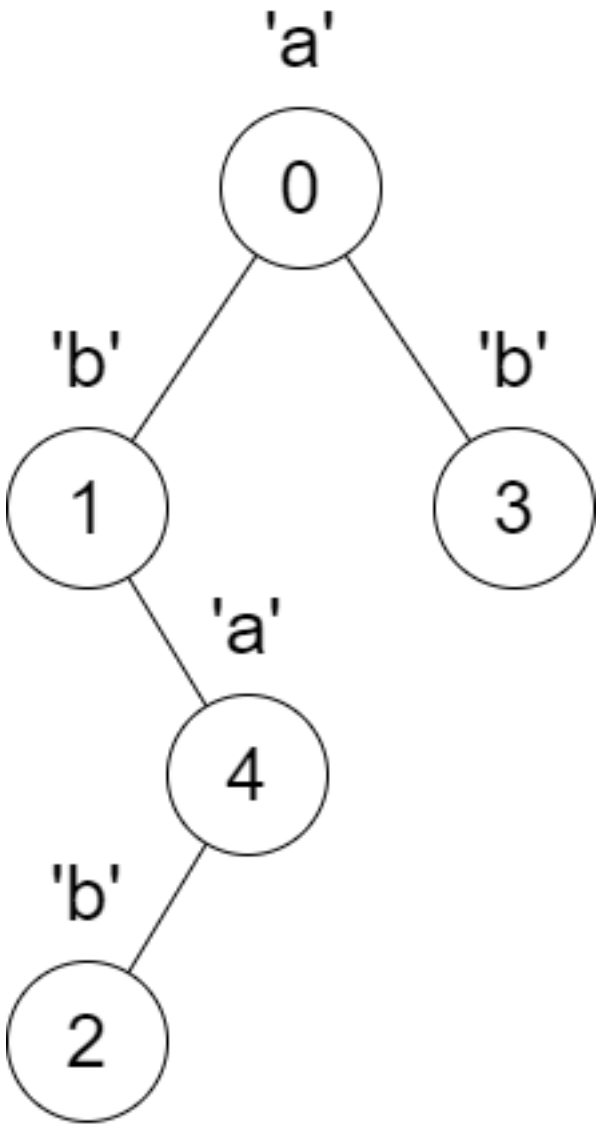
Input:

parent = [-1,0,4,0,1], s = "abbba"

Output:

[5,2,1,1,1]

Explanation:




The following changes will happen at the same time:

The parent of node 4 will change from node 1 to node 0.

The parent of node 2 will change from node 4 to node 1.

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<int> findSubtreeSizes(vector<int>& parent, string s) {

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* findSubtreeSizes(int* parent, int parentSize, char* s, int* returnSize)
{

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

```
end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

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

## Solutions

### C++ Solution:

```
/*
* Problem: Find Subtree Sizes After Changes
* Difficulty: Medium
* Tags: array, string, tree, graph, 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> findSubtreeSizes(vector<int>& parent, string s) {

}
};
```

### Java Solution:

```
/**
* Problem: Find Subtree Sizes After Changes
* Difficulty: Medium
* Tags: array, string, tree, graph, 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[] findSubtreeSizes(int[] parent, String s) {

}
}
```

### Python3 Solution:

```
"""
Problem: Find Subtree Sizes After Changes
Difficulty: Medium
Tags: array, string, tree, graph, 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 findSubtreeSizes(self, parent: List[int], s: str) -> List[int]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Find Subtree Sizes After Changes
* Difficulty: Medium
```

```
* Tags: array, string, tree, graph, 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 {number[]}
*/
var findSubtreeSizes = function(parent, s) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Find Subtree Sizes After Changes
* Difficulty: Medium
* Tags: array, string, tree, graph, 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 findSubtreeSizes(parent: number[], s: string): number[] {

};
```

**C# Solution:**

```
/*
* Problem: Find Subtree Sizes After Changes
* Difficulty: Medium
* Tags: array, string, tree, graph, 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[] FindSubtreeSizes(int[] parent, string s) {

}
}
```

### C Solution:

```
/*
* Problem: Find Subtree Sizes After Changes
* Difficulty: Medium
* Tags: array, string, tree, graph, 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* findSubtreeSizes(int* parent, int parentSize, char* s, int* returnSize)
{

}
```

### Go Solution:

```
// Problem: Find Subtree Sizes After Changes
// Difficulty: Medium
// Tags: array, string, tree, graph, 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 findSubtreeSizes(parent []int, s string) []int {
```

```
}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Find Subtree Sizes After Changes
// Difficulty: Medium
// Tags: array, string, tree, graph, 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_subtree_sizes(parent: Vec<i32>, s: String) -> Vec<i32> {

}
}
```

### Ruby Solution:

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

```
end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

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