# Problem 834: Sum of Distances in Tree

## Problem Information

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

## Problem Description

There is an undirected connected tree with

n

nodes labeled from

0

to

n - 1

and

n - 1

edges.

You are given the integer

n

and the array

edges

where

edges[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

in the tree.

Return an array

answer

of length

n

where

answer[i]

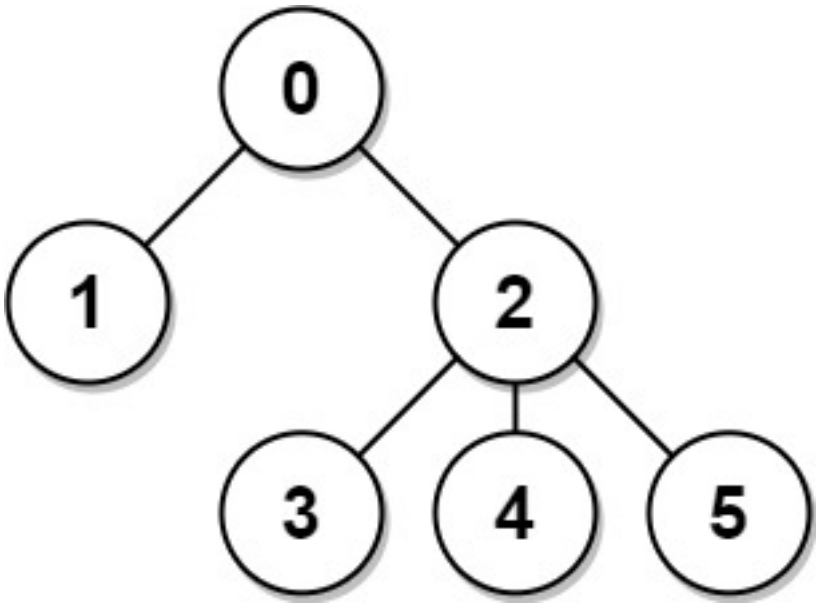
is the sum of the distances between the

i

th

node in the tree and all other nodes.

Example 1:




Input:

n = 6, edges = [[0,1],[0,2],[2,3],[2,4],[2,5]]

Output:

[8,12,6,10,10,10]

Explanation:

The tree is shown above. We can see that dist(0,1) + dist(0,2) + dist(0,3) + dist(0,4) + dist(0,5) equals 1 + 1 + 2 + 2 + 2 = 8. Hence, answer[0] = 8, and so on.

Example 2:

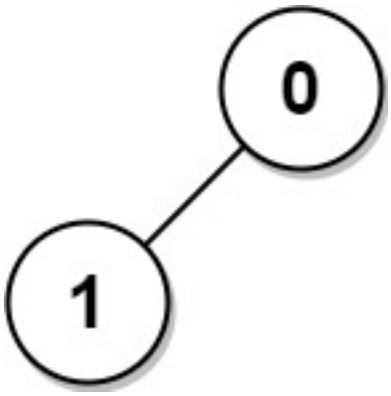

Input:

n = 1, edges = []

Output:

[0]

Example 3:




Input:

n = 2, edges = [[1,0]]

Output:

[1,1]

Constraints:

1 <= n <= 3 * 10

4

edges.length == n - 1

edges[i].length == 2

0 <= a

i

, b

i

< n

a

i

!= b

i

The given input represents a valid tree.

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> sumOfDistancesInTree(int n, vector<vector<int>>& edges) {

}
};
```

**Java:**

```
class Solution {
public int[] sumOfDistancesInTree(int n, int[][] edges) {

}
```

```
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function sumOfDistancesInTree(n: number, edges: number[][]): number[] {

};
```

**C#:**

```
public class Solution {
public int[] SumOfDistancesInTree(int n, int[][] edges) {

}
}
```

**C:**

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

}
```

**Go:**

```
func sumOfDistancesInTree(n int, edges [][]int) []int {

}
```

**Kotlin:**

```
class Solution {
fun sumOfDistancesInTree(n: Int, edges: Array<IntArray>): IntArray {

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer[]}
def sum_of_distances_in_tree(n, edges)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Integer[]
*/
function sumOfDistancesInTree($n, $edges) {

}
}
```

**Dart:**

```
class Solution {
List<int> sumOfDistancesInTree(int n, List<List<int>> edges) {

}
}
```

**Scala:**

```
object Solution {
def sumOfDistancesInTree(n: Int, edges: Array[Array[Int]]): Array[Int] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec sum_of_distances_in_tree(n :: integer, edges :: [[integer]]) ::
[integer]
def sum_of_distances_in_tree(n, edges) do
```

```
end
end
```

**Erlang:**

```
-spec sum_of_distances_in_tree(N :: integer(), Edges :: [[integer()]]) ->
[integer()].
sum_of_distances_in_tree(N, Edges) ->
.
```

**Racket:**

```
(define/contract (sum-of-distances-in-tree n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Sum of Distances in Tree
* Difficulty: Hard
* Tags: array, 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:
vector<int> sumOfDistancesInTree(int n, vector<vector<int>>& edges) {

}
};
```

**Java Solution:**

```
/**
* Problem: Sum of Distances in Tree
* Difficulty: Hard
* Tags: array, 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 int[] sumOfDistancesInTree(int n, int[][] edges) {

}
}
```

### Python3 Solution:

```
"""
Problem: Sum of Distances in Tree
Difficulty: Hard
Tags: array, 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 sumOfDistancesInTree(self, n: int, edges: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Sum of Distances in Tree
* Difficulty: Hard
* Tags: array, 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} n
* @param {number[][]} edges
* @return {number[]}
*/
var sumOfDistancesInTree = function(n, edges) {

};
```

### TypeScript Solution:

```
/**
* Problem: Sum of Distances in Tree
* Difficulty: Hard
* Tags: array, 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 sumOfDistancesInTree(n: number, edges: number[][]): number[] {

};
```

### C# Solution:

```
/*
* Problem: Sum of Distances in Tree
* Difficulty: Hard
```

```
* Tags: array, 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 int[] SumOfDistancesInTree(int n, int[][] edges) {

}
}
```

### C Solution:

```
/*
* Problem: Sum of Distances in Tree
* Difficulty: Hard
* Tags: array, 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
*/

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

}
```

### Go Solution:

```
// Problem: Sum of Distances in Tree
// Difficulty: Hard
// Tags: array, 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 sumOfDistancesInTree(n int, edges [][]int) []int {

}
```

### Kotlin Solution:

```
class Solution {
fun sumOfDistancesInTree(n: Int, edges: Array<IntArray>): IntArray {

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Sum of Distances in Tree
// Difficulty: Hard
// Tags: array, 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 sum_of_distances_in_tree(n: i32, edges: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

### Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer[]}
def sum_of_distances_in_tree(n, edges)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Integer[]
*/
function sumOfDistancesInTree($n, $edges) {

}
}
```

**Dart Solution:**

```
class Solution {
List<int> sumOfDistancesInTree(int n, List<List<int>> edges) {

}
}
```

**Scala Solution:**

```
object Solution {
def sumOfDistancesInTree(n: Int, edges: Array[Array[Int]]): Array[Int] = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec sum_of_distances_in_tree(n :: integer, edges :: [[integer]]) ::
[integer]
```

```
def sum_of_distances_in_tree(n, edges) do

end
end
```

### Erlang Solution:

```
-spec sum_of_distances_in_tree(N :: integer(), Edges :: [[integer()]]) ->
[integer()].
sum_of_distances_in_tree(N, Edges) ->
.
```

### Racket Solution:

```
(define/contract (sum-of-distances-in-tree n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```