Problem 2673: Make Costs of Paths Equal in a Binary Tree

Problem Information

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

Problem Description

You are given an integer

n

representing the number of nodes in a

perfect binary tree

consisting of nodes numbered from

1

to

n

. The root of the tree is node

1

and each node

i

in the tree has two children where the left child is the node

2 * i

and the right child is

2 * i + 1

.

Each node in the tree also has a

cost

represented by a given

0-indexed

integer array

cost

of size

n

where

cost[i]

is the cost of node

i + 1

. You are allowed to

increment

the cost of

any

node by

1

any

number of times.

Return

the

minimum

number of increments you need to make the cost of paths from the root to each

leaf

node equal

.

Note

:

A

perfect binary tree

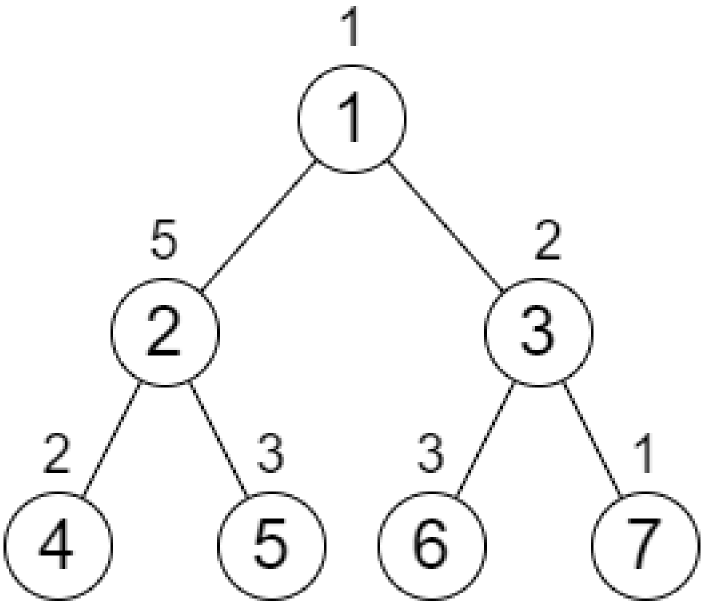
is a tree where each node, except the leaf nodes, has exactly 2 children.

The

cost of a path

is the sum of costs of nodes in the path.

Example 1:



Input:

n = 7, cost = [1,5,2,2,3,3,1]

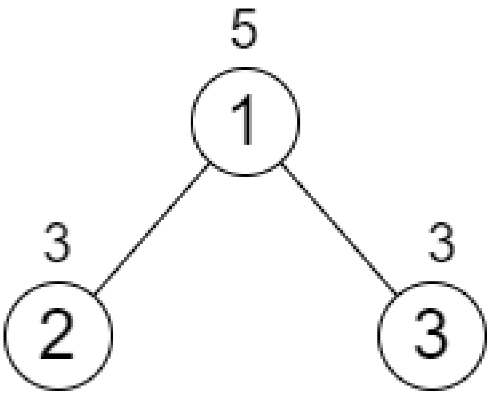
Output:

6

Explanation:

We can do the following increments: - Increase the cost of node 4 one time. - Increase the cost of node 3 three times. - Increase the cost of node 7 two times. Each path from the root to a leaf will have a total cost of 9. The total increments we did is 1 + 3 + 2 = 6. It can be shown that this is the minimum answer we can achieve.

Example 2:



Input:

n = 3, cost = [5,3,3]

Output:

0

Explanation:

The two paths already have equal total costs, so no increments are needed.

Constraints:

3 <= n <= 10

5

n + 1

is a power of

2

cost.length == n

1 <= cost[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int minIncrements(int n, vector<int>& cost) {

}
};
```

Java:

```
class Solution {
public int minIncrements(int n, int[] cost) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} n
* @param {number[]} cost
* @return {number}
*/
```

```
var minIncrements = function(n, cost) {

};
```

TypeScript:

```
function minIncrements(n: number, cost: number[]): number {

};
```

C#:

```
public class Solution {
public int MinIncrements(int n, int[] cost) {

}
}
```

C:

```
int minIncrements(int n, int* cost, int costSize) {

}
```

Go:

```
func minIncrements(n int, cost []int) int {

}
```

Kotlin:

```
class Solution {
fun minIncrements(n: Int, cost: IntArray): Int {

}
}
```

Swift:

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


}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} cost
# @return {Integer}
def min_increments(n, cost)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $cost
* @return Integer
*/
function minIncrements($n, $cost) {

}
}
```

Dart:

```
class Solution {
int minIncrements(int n, List<int> cost) {

}
}
```

Scala:

```
object Solution {
def minIncrements(n: Int, cost: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_increments(n :: integer, cost :: [integer]) :: integer
def min_increments(n, cost) do

end
end
```

Erlang:

```
-spec min_increments(N :: integer(), Cost :: [integer()]) -> integer().
min_increments(N, Cost) ->
.
```

Racket:

```
(define/contract (min-increments n cost)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make Costs of Paths Equal in a Binary Tree
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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 minIncrements(int n, vector<int>& cost) {

}
};
```

Java Solution:

```
/**
* Problem: Make Costs of Paths Equal in a Binary Tree
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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 minIncrements(int n, int[] cost) {

}
}
```

Python3 Solution:

```
"""
Problem: Make Costs of Paths Equal in a Binary Tree
Difficulty: Medium
Tags: array, tree, dp, greedy

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 minIncrements(self, n: int, cost: List[int]) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make Costs of Paths Equal in a Binary Tree
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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[]} cost
* @return {number}
*/
var minIncrements = function(n, cost) {

};
```

TypeScript Solution:

```
/**
* Problem: Make Costs of Paths Equal in a Binary Tree
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
```

```
* 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 minIncrements(n: number, cost: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Make Costs of Paths Equal in a Binary Tree
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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 MinIncrements(int n, int[] cost) {

}
}
```

C Solution:

```
/*
* Problem: Make Costs of Paths Equal in a Binary Tree
* Difficulty: Medium
* Tags: array, tree, dp, greedy
*
* 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
*/

int minIncrements(int n, int* cost, int costSize) {
}
```

Go Solution:

```
// Problem: Make Costs of Paths Equal in a Binary Tree
// Difficulty: Medium
// Tags: array, tree, dp, greedy
//
// 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 minIncrements(n int, cost []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minIncrements(n: Int, cost: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Make Costs of Paths Equal in a Binary Tree
// Difficulty: Medium
// Tags: array, tree, dp, greedy
//
// 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 min_increments(n: i32, cost: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} cost
# @return {Integer}
def min_increments(n, cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $cost
* @return Integer
*/
function minIncrements($n, $cost) {

}
}
```

Dart Solution:

```
class Solution {
int minIncrements(int n, List<int> cost) {

}
}
```

Scala Solution:

```
object Solution {
def minIncrements(n: Int, cost: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_increments(n :: integer, cost :: [integer]) :: integer
def min_increments(n, cost) do

end
end
```

Erlang Solution:

```
-spec min_increments(N :: integer(), Cost :: [integer()]) -> integer().
min_increments(N, Cost) ->
.
```

Racket Solution:

```
(define/contract (min-increments n cost)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```