Problem 1376: Time Needed to Inform All Employees

Problem Information

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

Problem Description

A company has

n

employees with a unique ID for each employee from

0

to

n - 1

. The head of the company is the one with

headID

.

Each employee has one direct manager given in the

manager

array where

manager[i]

is the direct manager of the

i-th

employee,

manager[headID] = -1

. Also, it is guaranteed that the subordination relationships have a tree structure.

The head of the company wants to inform all the company employees of an urgent piece of news. He will inform his direct subordinates, and they will inform their subordinates, and so on until all employees know about the urgent news.

The

i-th

employee needs

informTime[i]

minutes to inform all of his direct subordinates (i.e., After informTime[i] minutes, all his direct subordinates can start spreading the news).

Return

the number of minutes

needed to inform all the employees about the urgent news.

Example 1:

Input:

n = 1, headID = 0, manager = [-1], informTime = [0]

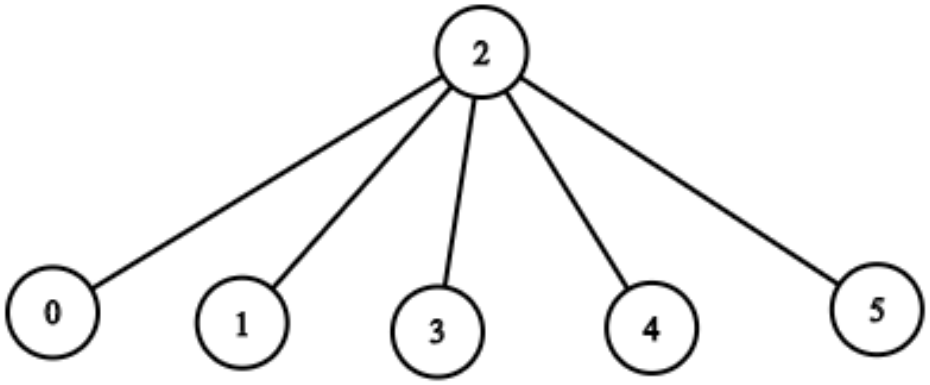
Output:

0

Explanation:

The head of the company is the only employee in the company.

Example 2:



Input:

n = 6, headID = 2, manager = [2,2,-1,2,2,2], informTime = [0,0,1,0,0,0]

Output:

1

Explanation:

The head of the company with id = 2 is the direct manager of all the employees in the company and needs 1 minute to inform them all. The tree structure of the employees in the company is shown.

Constraints:

1 <= n <= 10

5

0 <= headID < n

manager.length == n

0 <= manager[i] < n

manager[headID] == -1

informTime.length == n

0 <= informTime[i] <= 1000

informTime[i] == 0

if employee

i

has no subordinates.

It is

guaranteed

that all the employees can be informed.

Code Snippets

C++:

```
class Solution {
public:
int numOfMinutes(int n, int headID, vector<int>& manager, vector<int>&
informTime) {

}
};
```

Java:

```
class Solution {
public int numOfMinutes(int n, int headID, int[] manager, int[] informTime) {

}
}
```

Python3:

```
class Solution:
def numOfMinutes(self, n: int, headID: int, manager: List[int], informTime:
List[int]) -> int:
```

Python:

```
class Solution(object):
def numOfMinutes(self, n, headID, manager, informTime):
"""
:type n: int
:type headID: int
:type manager: List[int]
:type informTime: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} headID
* @param {number[]} manager
* @param {number[]} informTime
* @return {number}
*/
var numOfMinutes = function(n, headID, manager, informTime) {

};
```

TypeScript:

```
function numOfMinutes(n: number, headID: number, manager: number[],
informTime: number[]): number {

};
```

C#:

```
public class Solution {
public int NumOfMinutes(int n, int headID, int[] manager, int[] informTime) {

}
}
```

C:

```
int numOfMinutes(int n, int headID, int* manager, int managerSize, int*
informTime, int informTimeSize) {

}
```

Go:

```
func numOfMinutes(n int, headID int, manager []int, informTime []int) int {

}
```

Kotlin:

```
class Solution {
fun numOfMinutes(n: Int, headID: Int, manager: IntArray, informTime:
IntArray): Int {

}
}
```

Swift:

```
class Solution {
func numOfMinutes(_ n: Int, _ headID: Int, _ manager: [Int], _ informTime:
[Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_of_minutes(n: i32, head_id: i32, manager: Vec<i32>, inform_time:
```

```
Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} head_id
# @param {Integer[]} manager
# @param {Integer[]} inform_time
# @return {Integer}
def num_of_minutes(n, head_id, manager, inform_time)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $headID
* @param Integer[] $manager
* @param Integer[] $informTime
* @return Integer
*/
function numOfMinutes($n, $headID, $manager, $informTime) {

}
}
```

Dart:

```
class Solution {
int numOfMinutes(int n, int headID, List<int> manager, List<int> informTime)
{

}
}
```

Scala:

```
object Solution {
def numOfMinutes(n: Int, headID: Int, manager: Array[Int], informTime:
Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_of_minutes(n :: integer, head_id :: integer, manager :: [integer],
inform_time :: [integer]) :: integer
def num_of_minutes(n, head_id, manager, inform_time) do

end
end
```

Erlang:

```
-spec num_of_minutes(N :: integer(), HeadID :: integer(), Manager ::
[integer()], InformTime :: [integer()]) -> integer().
num_of_minutes(N, HeadID, Manager, InformTime) ->
.
```

Racket:

```
(define/contract (num-of-minutes n headID manager informTime)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Time Needed to Inform All Employees
* Difficulty: Medium
* Tags: array, tree, graph, 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 numOfMinutes(int n, int headID, vector<int>& manager, vector<int>&
informTime) {

}
};
```

Java Solution:

```
/**
* Problem: Time Needed to Inform All Employees
* Difficulty: Medium
* Tags: array, tree, graph, 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 numOfMinutes(int n, int headID, int[] manager, int[] informTime) {

}
}
```

Python3 Solution:

```
"""
Problem: Time Needed to Inform All Employees
Difficulty: Medium
Tags: array, tree, graph, 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 numOfMinutes(self, n: int, headID: int, manager: List[int], informTime:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numOfMinutes(self, n, headID, manager, informTime):
"""
:type n: int
:type headID: int
:type manager: List[int]
:type informTime: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Time Needed to Inform All Employees
* Difficulty: Medium
* Tags: array, tree, graph, 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} n
* @param {number} headID
* @param {number[]} manager
* @param {number[]} informTime
* @return {number}
*/
var numOfMinutes = function(n, headID, manager, informTime) {

};
```

TypeScript Solution:

```
/**
* Problem: Time Needed to Inform All Employees
* Difficulty: Medium
* Tags: array, tree, graph, 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 numOfMinutes(n: number, headID: number, manager: number[],
informTime: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Time Needed to Inform All Employees
* Difficulty: Medium
* Tags: array, tree, graph, 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 NumOfMinutes(int n, int headID, int[] manager, int[] informTime) {

}
}
```

C Solution:

```
/*
* Problem: Time Needed to Inform All Employees
* Difficulty: Medium
* Tags: array, tree, graph, 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
*/

int numOfMinutes(int n, int headID, int* manager, int managerSize, int*
informTime, int informTimeSize) {

}
```

Go Solution:

```
// Problem: Time Needed to Inform All Employees
// Difficulty: Medium
// Tags: array, tree, graph, 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 numOfMinutes(n int, headID int, manager []int, informTime []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numOfMinutes(n: Int, headID: Int, manager: IntArray, informTime:
IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func numOfMinutes(_ n: Int, _ headID: Int, _ manager: [Int], _ informTime:
[Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Time Needed to Inform All Employees
// Difficulty: Medium
// Tags: array, tree, graph, 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 num_of_minutes(n: i32, head_id: i32, manager: Vec<i32>, inform_time:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} head_id
# @param {Integer[]} manager
# @param {Integer[]} inform_time
# @return {Integer}
def num_of_minutes(n, head_id, manager, inform_time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $headID
* @param Integer[] $manager
* @param Integer[] $informTime
* @return Integer
*/
function numOfMinutes($n, $headID, $manager, $informTime) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int numOfMinutes(int n, int headID, List<int> manager, List<int> informTime)
{

}
}
```

Scala Solution:

```
object Solution {
def numOfMinutes(n: Int, headID: Int, manager: Array[Int], informTime:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_of_minutes(n :: integer, head_id :: integer, manager :: [integer],
inform_time :: [integer]) :: integer
def num_of_minutes(n, head_id, manager, inform_time) do

end
end
```

Erlang Solution:

```
-spec num_of_minutes(N :: integer(), HeadID :: integer(), Manager ::
[integer()], InformTime :: [integer()]) -> integer().
num_of_minutes(N, HeadID, Manager, InformTime) ->
.
```

Racket Solution:

```
(define/contract (num-of-minutes n headID manager informTime)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
```

```
exact-integer?) exact-integer?)
)
```