Problem 690: Employee Importance

Problem Information

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

Problem Description

You have a data structure of employee information, including the employee's unique ID, importance value, and direct subordinates' IDs.

You are given an array of employees

employees

where:

employees[i].id

is the ID of the

i

th

employee.

employees[i].importance

is the importance value of the

i

th

employee.

employees[i].subordinates

is a list of the IDs of the direct subordinates of the

i

th

employee.

Given an integer

id

that represents an employee's ID, return

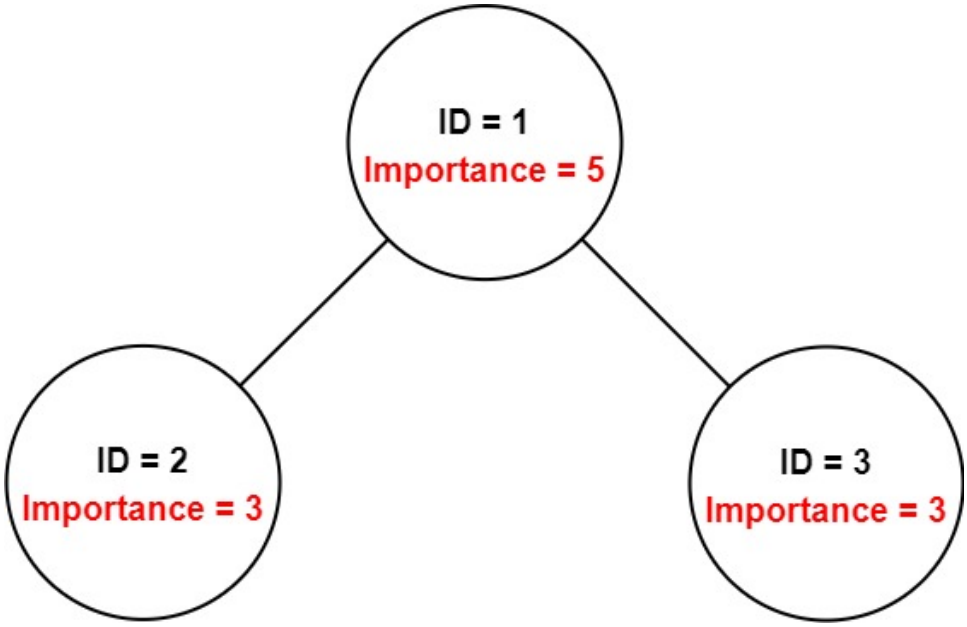
the

total

importance value of this employee and all their direct and indirect subordinates

.

Example 1:



Input:

employees = [[1,5,[2,3]],[2,3,[]],[3,3,[]]], id = 1

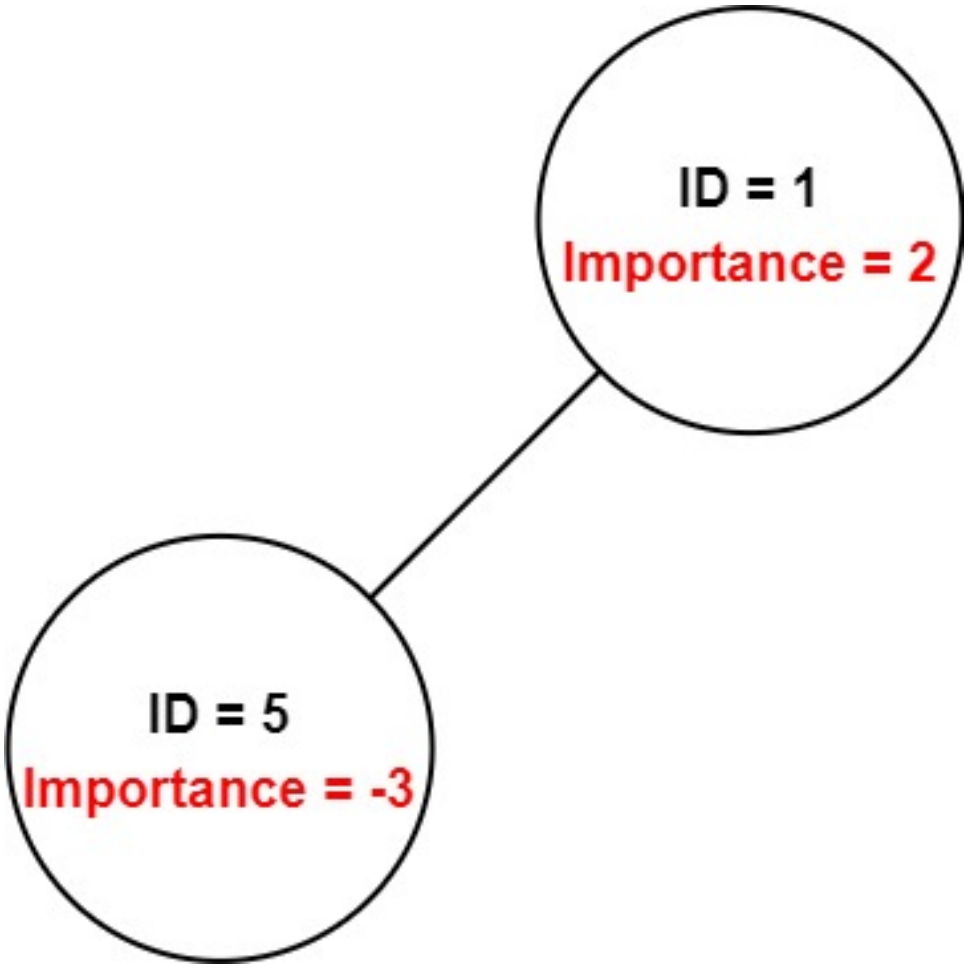
Output:

11

Explanation:

Employee 1 has an importance value of 5 and has two direct subordinates: employee 2 and employee 3. They both have an importance value of 3. Thus, the total importance value of employee 1 is 5 + 3 + 3 = 11.

Example 2:



Input:

employees = [[1,2,[5]],[5,-3,[]]], id = 5

Output:

-3

Explanation:

Employee 5 has an importance value of -3 and has no direct subordinates. Thus, the total importance value of employee 5 is -3.

Constraints:

1 <= employees.length <= 2000

1 <= employees[i].id <= 2000

All

employees[i].id

are

unique

.

-100 <= employees[i].importance <= 100

One employee has at most one direct leader and may have several subordinates.

The IDs in

employees[i].subordinates

are valid IDs.

Code Snippets

C++:

```
/*
// Definition for Employee.
class Employee {
public:
int id;
int importance;
vector<int> subordinates;
};
*/

class Solution {
public:
int getImportance(vector<Employee*> employees, int id) {

}
```

```
};
```

Java:

```
/*
// Definition for Employee.
class Employee {
public int id;
public int importance;
public List<Integer> subordinates;
};
*/

class Solution {
public int getImportance(List<Employee> employees, int id) {

}
}
```

Python3:

```
"""
# Definition for Employee.
class Employee:
def __init__(self, id: int, importance: int, subordinates: List[int]):
self.id = id
self.importance = importance
self.subordinates = subordinates
"""

class Solution:
def getImportance(self, employees: List['Employee'], id: int) -> int:
```

Python:

```
"""
# Definition for Employee.
class Employee(object):
def __init__(self, id, importance, subordinates):
#################
:type id: int
:type importance: int
```

```
:type subordinates: List[int]
##################
self.id = id
self.importance = importance
self.subordinates = subordinates
"""

class Solution(object):
def getImportance(self, employees, id):
"""
:type employees: List[Employee]
:type id: int
:rtype: int
"""
```

JavaScript:

```
/**
* Definition for Employee.
* function Employee(id, importance, subordinates) {
* this.id = id;
* this.importance = importance;
* this.subordinates = subordinates;
* }
*/

/**
* @param {Employee[]} employees
* @param {number} id
* @return {number}
*/
var GetImportance = function(employees, id) {

};
```

TypeScript:

```
/**
* Definition for Employee.
* class Employee {
* id: number
* importance: number
```

```
* subordinates: number[]
* constructor(id: number, importance: number, subordinates: number[]) {
* this.id = (id === undefined) ? 0 : id;
* this.importance = (importance === undefined) ? 0 : importance;
* this.subordinates = (subordinates === undefined) ? [] : subordinates;
* }
* }
*/

function getImportance(employees: Employee[], id: number): number {

};
```

C#:

```
/*
// Definition for Employee.
class Employee {
public int id;
public int importance;
public IList<int> subordinates;
}
*/

class Solution {
public int GetImportance(IList<Employee> employees, int id) {

}
}
```

Go:

```
/**
* Definition for Employee.
* type Employee struct {
* Id int
* Importance int
* Subordinates []int
* }
*/

func getImportance(employees []*Employee, id int) int {
```

```
}
```

Kotlin:

```
/*
* // Definition for Employee.
* class Employee {
* var id:Int = 0
* var importance:Int = 0
* var subordinates:List<Int> = listOf()
* }
*/

class Solution {
fun getImportance(employees: List<Employee?>, id: Int): Int {

}
}
```

Swift:

```
/**
* Definition for Employee.
* public class Employee {
* public var id: Int
* public var importance: Int
* public var subordinates: [Int]
* public init(_ id: Int, _ importance: Int, _ subordinates: [Int]) {
* self.id = id
* self.importance = importance
* self.subordinates = subordinates
* }
* }
*/

class Solution {
func getImportance(_ employees: [Employee], _ id: Int) -> Int {

}
}
```

Ruby:

```
=begin
# Definition for Employee.
class Employee
attr_accessor :id, :importance, :subordinates
def initialize( id, importance, subordinates)
@id = id
@importance = importance
@subordinates = subordinates
end
end
=end

# @param {Employee} employees
# @param {Integer} id
# @return {Integer}
def get_importance(employees, id)

end
```

PHP:

```
/**
* Definition for Employee.
* class Employee {
* public $id = null;
* public $importance = null;
* public $subordinates = array();
* function __construct($id, $importance, $subordinates) {
* $this->id = $id;
* $this->importance = $importance;
* $this->subordinates = $subordinates;
* }
* }
*/

class Solution {
/**
* @param Employee[] $employees
* @param Integer $id
* @return Integer
*/
```

```
function getImportance($employees, $id) {

}
}
```

Scala:

```
/*
// Definition for Employee.
class Employee() {
var id: Int = 0
var importance: Int = 0
var subordinates: List[Int] = List()
};
*/

object Solution {
def getImportance(employees: List[Employee], id: Int): Int = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Employee Importance
* Difficulty: Medium
* Tags: array, tree, 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
*/

/*
// Definition for Employee.
class Employee {
public:
```

```
int id;
int importance;
vector<int> subordinates;
};
*/

class Solution {
public:
int getImportance(vector<Employee*> employees, int id) {

}
};
```

Java Solution:

```
/**
* Problem: Employee Importance
* Difficulty: Medium
* Tags: array, tree, 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
*/

/*
// Definition for Employee.
class Employee {
public int id;
public int importance;
public List<Integer> subordinates;
};
*/

class Solution {
public int getImportance(List<Employee> employees, int id) {

}
}
```

Python3 Solution:

```
"""
Problem: Employee Importance
Difficulty: Medium
Tags: array, tree, 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
"""

"""
# Definition for Employee.
class Employee:
def __init__(self, id: int, importance: int, subordinates: List[int]):
self.id = id
self.importance = importance
self.subordinates = subordinates
"""

class Solution:
def getImportance(self, employees: List['Employee'], id: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
"""
# Definition for Employee.
class Employee(object):
def __init__(self, id, importance, subordinates):
#################
:type id: int
:type importance: int
:type subordinates: List[int]
#################
self.id = id
self.importance = importance
self.subordinates = subordinates
"""

class Solution(object):
def getImportance(self, employees, id):
```

```
"""
:type employees: List[Employee]
:type id: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Employee Importance
* Difficulty: Medium
* Tags: array, tree, 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
*/

/**
* Definition for Employee.
* function Employee(id, importance, subordinates) {
* this.id = id;
* this.importance = importance;
* this.subordinates = subordinates;
* }
*/

/**
* @param {Employee[]} employees
* @param {number} id
* @return {number}
*/
var GetImportance = function(employees, id) {

};
```

TypeScript Solution:

```
/**
* Problem: Employee Importance
* Difficulty: Medium
```

```
* Tags: array, tree, 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
*/

/**
* Definition for Employee.
* class Employee {
* id: number
* importance: number
* subordinates: number[]
* constructor(id: number, importance: number, subordinates: number[]) {
* this.id = (id === undefined) ? 0 : id;
* this.importance = (importance === undefined) ? 0 : importance;
* this.subordinates = (subordinates === undefined) ? [] : subordinates;
* }
* }
*/

function getImportance(employees: Employee[], id: number): number {

};
```

C# Solution:

```
/*
* Problem: Employee Importance
* Difficulty: Medium
* Tags: array, tree, 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
*/

/*
// Definition for Employee.
class Employee {
public int id;
```

```
public int importance;
public IList<int> subordinates;
}
*/

class Solution {
public int GetImportance(IList<Employee> employees, int id) {

}
}
```

Go Solution:

```
// Problem: Employee Importance
// Difficulty: Medium
// Tags: array, tree, 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

/**
* Definition for Employee.
* type Employee struct {
* Id int
* Importance int
* Subordinates []int
* }
*/

func getImportance(employees []*Employee, id int) int {

}
```

Kotlin Solution:

```
/*
* // Definition for Employee.
* class Employee {
* var id:Int = 0
* var importance:Int = 0
```

```
* var subordinates:List<Int> = listOf()
* }
*/

class Solution {
fun getImportance(employees: List<Employee?>, id: Int): Int {

}
}
```

Swift Solution:

```
/**
* Definition for Employee.
* public class Employee {
* public var id: Int
* public var importance: Int
* public var subordinates: [Int]
* public init(_ id: Int, _ importance: Int, _ subordinates: [Int]) {
* self.id = id
* self.importance = importance
* self.subordinates = subordinates
* }
* }
*/

class Solution {
func getImportance(_ employees: [Employee], _ id: Int) -> Int {

}
}
```

Ruby Solution:

```
=begin
# Definition for Employee.
class Employee
attr_accessor :id, :importance, :subordinates
def initialize( id, importance, subordinates)
@id = id
@importance = importance
```

```
@subordinates = subordinates
end
end
=end

# @param {Employee} employees
# @param {Integer} id
# @return {Integer}
def get_importance(employees, id)

end
```

PHP Solution:

```
/**
* Definition for Employee.
* class Employee {
* public $id = null;
* public $importance = null;
* public $subordinates = array();
* function __construct($id, $importance, $subordinates) {
* $this->id = $id;
* $this->importance = $importance;
* $this->subordinates = $subordinates;
* }
* }
*/

class Solution {
/**
* @param Employee[] $employees
* @param Integer $id
* @return Integer
*/
function getImportance($employees, $id) {

}
}
```

Scala Solution:

```
/*
// Definition for Employee.
class Employee() {
var id: Int = 0
var importance: Int = 0
var subordinates: List[Int] = List()
};
*/

object Solution {
def getImportance(employees: List[Employee], id: Int): Int = {

}
}
```