Problem 2920: Maximum Points After Collecting Coins From All Nodes

Problem Information

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

Problem Description

There exists an undirected tree rooted at node

0

with

n

nodes labeled from

0

to

n - 1

. You are given a 2D

integer

array

edges

of length

n - 1

, where

edges[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

in the tree. You are also given a

0-indexed

array

coins

of size

n

where

coins[i]

indicates the number of coins in the vertex

i

, and an integer

k

.

Starting from the root, you have to collect all the coins such that the coins at a node can only be collected if the coins of its ancestors have been already collected.

Coins at

node

i

can be collected in one of the following ways:

Collect all the coins, but you will get

coins[i] - k

points. If

coins[i] - k

is negative then you will lose

abs(coins[i] - k)

points.

Collect all the coins, but you will get

floor(coins[i] / 2)

points. If this way is used, then for all the

node

j

present in the subtree of

node

i

,

coins[j]

will get reduced to

floor(coins[j] / 2)

.

Return

the

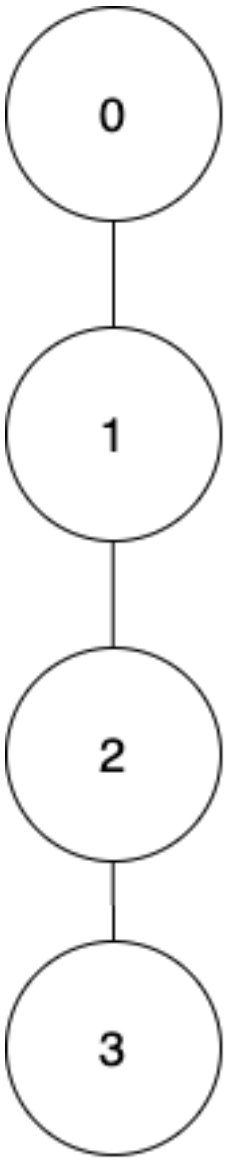
maximum points

you can get after collecting the coins from

all

the tree nodes.

Example 1:



Input:

edges = [[0,1],[1,2],[2,3]], coins = [10,10,3,3], k = 5

Output:

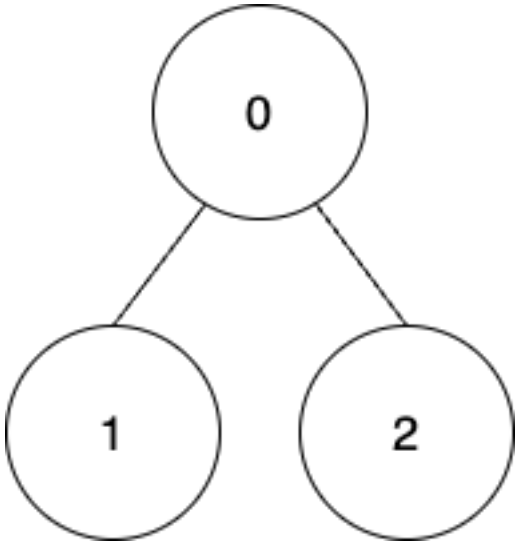
11

Explanation:

Collect all the coins from node 0 using the first way. Total points = 10 - 5 = 5. Collect all the coins from node 1 using the first way. Total points = 5 + (10 - 5) = 10. Collect all the coins from node 2 using the second way so coins left at node 3 will be floor(3 / 2) = 1. Total points = 10 +

floor(3 / 2) = 11. Collect all the coins from node 3 using the second way. Total points = 11 + floor(1 / 2) = 11. It can be shown that the maximum points we can get after collecting coins from all the nodes is 11.

Example 2:



Input:

edges = [[0,1],[0,2]], coins = [8,4,4], k = 0

Output:

16

Explanation:

Coins will be collected from all the nodes using the first way. Therefore, total points = (8 - 0) + (4 - 0) + (4 - 0) = 16.

Constraints:

n == coins.length

2 <= n <= 10

5

0 <= coins[i] <= 10

4

edges.length == n - 1

0 <= edges[i][0], edges[i][1] < n

0 <= k <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int maximumPoints(vector<vector<int>>& edges, vector<int>& coins, int k) {

}
};
```

Java:

```
class Solution {
public int maximumPoints(int[][] edges, int[] coins, int k) {

}
}
```

Python3:

```
class Solution:
def maximumPoints(self, edges: List[List[int]], coins: List[int], k: int) ->
int:
```

Python:

```
class Solution(object):
def maximumPoints(self, edges, coins, k):
"""
:type edges: List[List[int]]
:type coins: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} edges
* @param {number[]} coins
* @param {number} k
* @return {number}
*/
var maximumPoints = function(edges, coins, k) {

};
```

TypeScript:

```
function maximumPoints(edges: number[][], coins: number[], k: number): number
{

};
```

C#:

```
public class Solution {
public int MaximumPoints(int[][] edges, int[] coins, int k) {

}
}
```

C:

```
int maximumPoints(int** edges, int edgesSize, int* edgesColSize, int* coins,
int coinsSize, int k) {

}
```

Go:

```
func maximumPoints(edges [][]int, coins []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maximumPoints(edges: Array<IntArray>, coins: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maximumPoints(_ edges: [[Int]], _ coins: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_points(edges: Vec<Vec<i32>>, coins: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer[]} coins
# @param {Integer} k
# @return {Integer}
def maximum_points(edges, coins, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $coins
* @param Integer $k
* @return Integer
*/
function maximumPoints($edges, $coins, $k) {

}
}
```

Dart:

```
class Solution {
int maximumPoints(List<List<int>> edges, List<int> coins, int k) {

}
}
```

Scala:

```
object Solution {
def maximumPoints(edges: Array[Array[Int]], coins: Array[Int], k: Int): Int =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_points(edges :: [[integer]], coins :: [integer], k :: integer)
:: integer
def maximum_points(edges, coins, k) do

end
end
```

Erlang:

```
-spec maximum_points(Edges :: [[integer()]], Coins :: [integer()], K ::
integer()) -> integer().
maximum_points(Edges, Coins, K) ->
.
```

Racket:

```
(define/contract (maximum-points edges coins k)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Points After Collecting Coins From All Nodes
* Difficulty: Hard
* Tags: array, tree, 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 maximumPoints(vector<vector<int>>& edges, vector<int>& coins, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Points After Collecting Coins From All Nodes
* Difficulty: Hard
* Tags: array, tree, 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 maximumPoints(int[][] edges, int[] coins, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Points After Collecting Coins From All Nodes
Difficulty: Hard
Tags: array, tree, 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 maximumPoints(self, edges: List[List[int]], coins: List[int], k: int) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumPoints(self, edges, coins, k):
"""
:type edges: List[List[int]]
:type coins: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Points After Collecting Coins From All Nodes
* Difficulty: Hard
* Tags: array, tree, 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[][]} edges
* @param {number[]} coins
* @param {number} k
* @return {number}
*/
var maximumPoints = function(edges, coins, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Points After Collecting Coins From All Nodes
* Difficulty: Hard
* Tags: array, tree, 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 maximumPoints(edges: number[][], coins: number[], k: number): number
{

};
```

C# Solution:

```
/*
* Problem: Maximum Points After Collecting Coins From All Nodes
* Difficulty: Hard
```

```
* Tags: array, tree, 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 MaximumPoints(int[][] edges, int[] coins, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Points After Collecting Coins From All Nodes
* Difficulty: Hard
* Tags: array, tree, 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
*/

int maximumPoints(int** edges, int edgesSize, int* edgesColSize, int* coins,
int coinsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Points After Collecting Coins From All Nodes
// Difficulty: Hard
// Tags: array, tree, 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 maximumPoints(edges [][]int, coins []int, k int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun maximumPoints(edges: Array<IntArray>, coins: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumPoints(_ edges: [[Int]], _ coins: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Points After Collecting Coins From All Nodes
// Difficulty: Hard
// Tags: array, tree, 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 maximum_points(edges: Vec<Vec<i32>>, coins: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer[]} coins
# @param {Integer} k
# @return {Integer}
```

```
def maximum_points(edges, coins, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $coins
* @param Integer $k
* @return Integer
*/
function maximumPoints($edges, $coins, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maximumPoints(List<List<int>> edges, List<int> coins, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maximumPoints(edges: Array[Array[Int]], coins: Array[Int], k: Int): Int =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_points(edges :: [[integer]], coins :: [integer], k :: integer)
:: integer
```

```
def maximum_points(edges, coins, k) do

end
end
```

Erlang Solution:

```
-spec maximum_points(Edges :: [[integer()]], Coins :: [integer()], K ::
integer()) -> integer().
maximum_points(Edges, Coins, K) ->
.
```

Racket Solution:

```
(define/contract (maximum-points edges coins k)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?
exact-integer?)
)
```