Problem 2467: Most Profitable Path in a Tree

Problem Information

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

Problem Description

There is an undirected tree with

n

nodes labeled from

0

to

n - 1

, rooted at node

0

. 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.

At every node

i

, there is a gate. You are also given an array of even integers

amount

, where

amount[i]

represents:

the price needed to open the gate at node i

, if

amount[i]

is negative, or,

the cash reward obtained on opening the gate at node

i

, otherwise.

The game goes on as follows:

Initially, Alice is at node

0

and Bob is at node

bob

.

At every second, Alice and Bob

each

move to an adjacent node. Alice moves towards some

leaf node

, while Bob moves towards node

0

.

For

every

node along their path, Alice and Bob either spend money to open the gate at that node, or accept the reward. Note that:

If the gate is

already open

, no price will be required, nor will there be any cash reward.

If Alice and Bob reach the node

simultaneously

, they share the price/reward for opening the gate there. In other words, if the price to open the gate is

c

, then both Alice and Bob pay

$c / 2$

each. Similarly, if the reward at the gate is

c

, both of them receive

$c / 2$

each.

If Alice reaches a leaf node, she stops moving. Similarly, if Bob reaches node

0

, he stops moving. Note that these events are

independent

of each other.

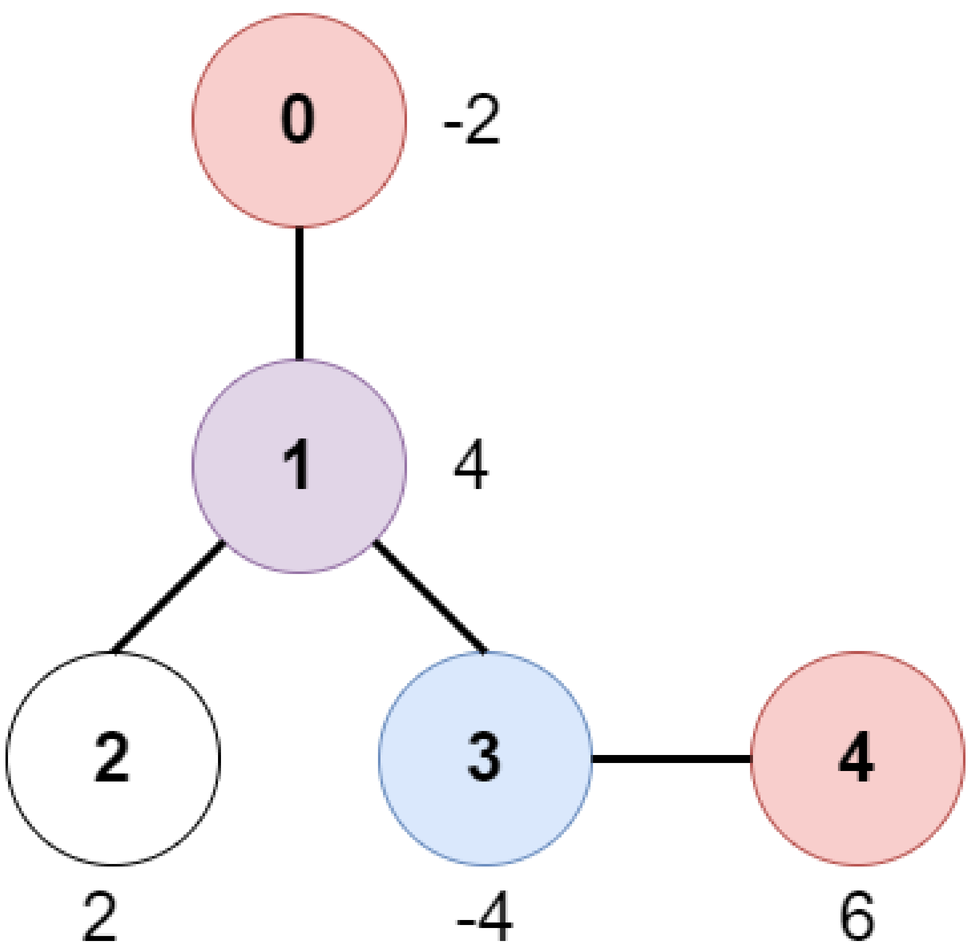
Return

the

maximum

net income Alice can have if she travels towards the optimal leaf node.

Example 1:



Input:

edges = [[0,1],[1,2],[1,3],[3,4]], bob = 3, amount = [-2,4,2,-4,6]

Output:

6

Explanation:

The above diagram represents the given tree. The game goes as follows: - Alice is initially on node 0, Bob on node 3. They open the gates of their respective nodes. Alice's net income is now -2. - Both Alice and Bob move to node 1.   Since they reach here simultaneously, they open the gate together and share the reward.   Alice's net income becomes -2 + (4 / 2) = 0. - Alice moves on to node 3. Since Bob already opened its gate, Alice's income remains unchanged.   Bob moves on to node 0, and stops moving. - Alice moves on to node 4 and opens the gate there. Her net income becomes 0 + 6 = 6. Now, neither Alice nor Bob can make any further moves, and the game ends. It is not possible for Alice to get a higher net income.

Example 2:

Input:

edges = [[0,1]], bob = 1, amount = [-7280,2350]

Output:

-7280

Explanation:

Alice follows the path 0->1 whereas Bob follows the path 1->0. Thus, Alice opens the gate at node 0 only. Hence, her net income is -7280.

Constraints:

2 <= n <= 10

5

edges.length == n - 1

edges[i].length == 2

0 <= a

i

, b

i

< n

a

i

!= b

i

edges

represents a valid tree.

1 <= bob < n

amount.length == n

amount[i]

is an

even

integer in the range

[-10

4

, 10

4

]

.

Code Snippets

C++:

```
class Solution {
public:
int mostProfitablePath(vector<vector<int>>& edges, int bob, vector<int>&
amount) {

}
};
```

Java:

```
class Solution {
public int mostProfitablePath(int[][] edges, int bob, int[] amount) {

}
}
```

Python3:

```
class Solution:
def mostProfitablePath(self, edges: List[List[int]], bob: int, amount:
List[int]) -> int:
```

Python:

```
class Solution(object):
def mostProfitablePath(self, edges, bob, amount):
"""
:type edges: List[List[int]]
:type bob: int
:type amount: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} edges
* @param {number} bob
* @param {number[]} amount
* @return {number}
*/
var mostProfitablePath = function(edges, bob, amount) {

};
```

TypeScript:

```
function mostProfitablePath(edges: number[][], bob: number, amount:
number[]): number {

};
```

C#:

```
public class Solution {
public int MostProfitablePath(int[][] edges, int bob, int[] amount) {

}
}
```

C:

```
int mostProfitablePath(int** edges, int edgesSize, int* edgesColSize, int
bob, int* amount, int amountSize) {

}
```

Go:

```
func mostProfitablePath(edges [][]int, bob int, amount []int) int {

}
```

Kotlin:

```
class Solution {
fun mostProfitablePath(edges: Array<IntArray>, bob: Int, amount: IntArray):
Int {

}
}
```

Swift:

```
class Solution {
func mostProfitablePath(_ edges: [[Int]], _ bob: Int, _ amount: [Int]) -> Int
{

}
}
```

Rust:

```
impl Solution {
pub fn most_profitable_path(edges: Vec<Vec<i32>>, bob: i32, amount: Vec<i32>)
-> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer} bob
# @param {Integer[]} amount
# @return {Integer}
def most_profitable_path(edges, bob, amount)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer $bob
* @param Integer[] $amount
* @return Integer
*/
function mostProfitablePath($edges, $bob, $amount) {

}
}
```

Dart:

```
class Solution {
int mostProfitablePath(List<List<int>> edges, int bob, List<int> amount) {

}
}
```

Scala:

```
object Solution {
def mostProfitablePath(edges: Array[Array[Int]], bob: Int, amount:
Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec most_profitable_path(edges :: [[integer]], bob :: integer, amount ::
[integer]) :: integer
def most_profitable_path(edges, bob, amount) do

end
end
```

Erlang:

```
-spec most_profitable_path(Edges :: [[integer()]], Bob :: integer(), Amount
:: [integer()]) -> integer().
most_profitable_path(Edges, Bob, Amount) ->
.
```

Racket:

```
(define/contract (most-profitable-path edges bob amount)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Most Profitable Path in a Tree
* 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 mostProfitablePath(vector<vector<int>>& edges, int bob, vector<int>&
amount) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Most Profitable Path in a Tree
* 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 mostProfitablePath(int[][] edges, int bob, int[] amount) {

}
}
```

Python3 Solution:

```
"""
Problem: Most Profitable Path in a Tree
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 mostProfitablePath(self, edges: List[List[int]], bob: int, amount:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mostProfitablePath(self, edges, bob, amount):
"""
:type edges: List[List[int]]
:type bob: int
:type amount: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Most Profitable Path in a Tree
* 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[][]} edges
* @param {number} bob
* @param {number[]} amount
* @return {number}
*/
var mostProfitablePath = function(edges, bob, amount) {

};
```

TypeScript Solution:

```
/**
* Problem: Most Profitable Path in a Tree
* 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 mostProfitablePath(edges: number[][], bob: number, amount:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: Most Profitable Path in a Tree
* 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 MostProfitablePath(int[][] edges, int bob, int[] amount) {

}
}
```

C Solution:

```
/*
* Problem: Most Profitable Path in a Tree
* 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 mostProfitablePath(int** edges, int edgesSize, int* edgesColSize, int
bob, int* amount, int amountSize) {

}
```

Go Solution:

```
// Problem: Most Profitable Path in a Tree
// 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 mostProfitablePath(edges [][]int, bob int, amount []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun mostProfitablePath(edges: Array<IntArray>, bob: Int, amount: IntArray):
Int {

}
}
```

Swift Solution:

```
class Solution {
func mostProfitablePath(_ edges: [[Int]], _ bob: Int, _ amount: [Int]) -> Int
{

}
}
```

Rust Solution:

```
// Problem: Most Profitable Path in a Tree
// 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 most_profitable_path(edges: Vec<Vec<i32>>, bob: i32, amount: Vec<i32>)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer} bob
# @param {Integer[]} amount
# @return {Integer}
def most_profitable_path(edges, bob, amount)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer $bob
* @param Integer[] $amount
* @return Integer
*/
function mostProfitablePath($edges, $bob, $amount) {

}
}
```

Dart Solution:

```
class Solution {
int mostProfitablePath(List<List<int>> edges, int bob, List<int> amount) {

}
}
```

Scala Solution:

```
object Solution {
def mostProfitablePath(edges: Array[Array[Int]], bob: Int, amount:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec most_profitable_path(edges :: [[integer]], bob :: integer, amount ::
[integer]) :: integer
def most_profitable_path(edges, bob, amount) do

end
end
```

Erlang Solution:

```
-spec most_profitable_path(Edges :: [[integer()]], Bob :: integer(), Amount
:: [integer()]) -> integer().
most_profitable_path(Edges, Bob, Amount) ->
.
```

Racket Solution:

```
(define/contract (most-profitable-path edges bob amount)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?)
exact-integer?)
)
```