Problem 3562: Maximum Profit from Trading Stocks with Discounts

Problem Information

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

Problem Description

You are given an integer

n

, representing the number of employees in a company. Each employee is assigned a unique ID from 1 to

n

, and employee 1 is the CEO. You are given two

1-based

integer arrays,

present

and

future

, each of length

n

, where:

present[i]

represents the

current

price at which the

i

th

employee can buy a stock today.

future[i]

represents the

expected

price at which the

i

th

employee can sell the stock tomorrow.

The company's hierarchy is represented by a 2D integer array

hierarchy

, where

hierarchy[i] = [u

i

, v

i

]

means that employee

u

i

is the direct boss of employee

v

i

.

Additionally, you have an integer

budget

representing the total funds available for investment.

However, the company has a discount policy: if an employee's direct boss purchases their own stock, then the employee can buy their stock at

half

the original price (

floor(present[v] / 2)

).

Return the

maximum

profit that can be achieved without exceeding the given budget.

Note:

You may buy each stock at most

once

.

You

cannot

use any profit earned from future stock prices to fund additional investments and must buy only from

budget

.

Example 1:

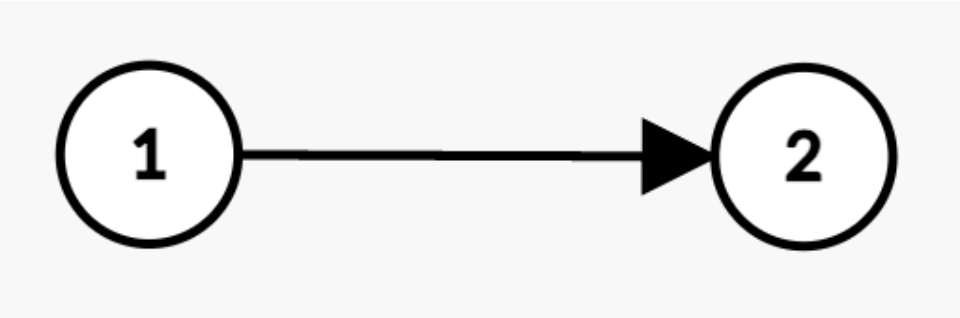
Input:

n = 2, present = [1,2], future = [4,3], hierarchy = [[1,2]], budget = 3

Output:

5

Explanation:



Employee 1 buys the stock at price 1 and earns a profit of

4 - 1 = 3

.

Since Employee 1 is the direct boss of Employee 2, Employee 2 gets a discounted price of

floor(2 / 2) = 1

.

Employee 2 buys the stock at price 1 and earns a profit of

3 - 1 = 2

.

The total buying cost is

1 + 1 = 2 <= budget

. Thus, the maximum total profit achieved is

3 + 2 = 5

.

Example 2:

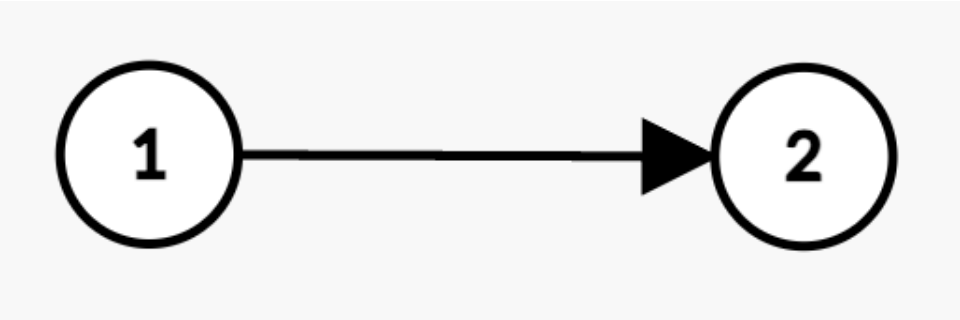
Input:

n = 2, present = [3,4], future = [5,8], hierarchy = [[1,2]], budget = 4

Output:

4

Explanation:



Employee 2 buys the stock at price 4 and earns a profit of

8 - 4 = 4

.

Since both employees cannot buy together, the maximum profit is 4.

Example 3:

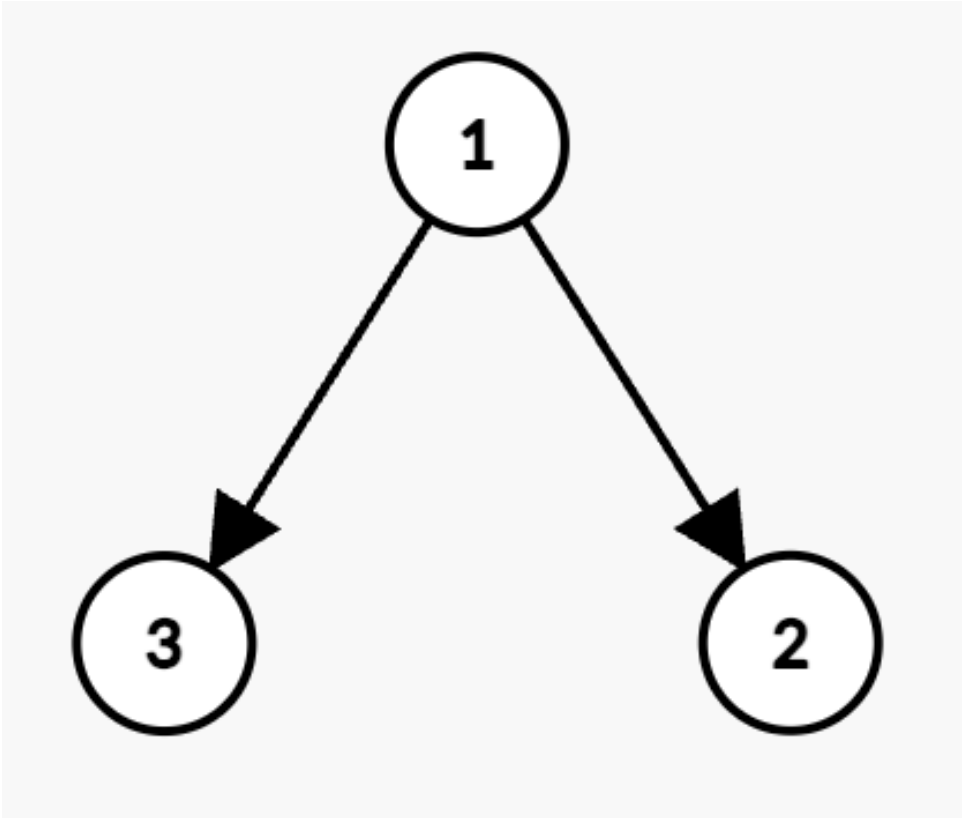
Input:

n = 3, present = [4,6,8], future = [7,9,11], hierarchy = [[1,2],[1,3]], budget = 10

Output:

10

Explanation:



Employee 1 buys the stock at price 4 and earns a profit of

7 - 4 = 3

.

Employee 3 would get a discounted price of

floor(8 / 2) = 4

and earns a profit of

11 - 4 = 7

.

Employee 1 and Employee 3 buy their stocks at a total cost of

4 + 4 = 8 <= budget

. Thus, the maximum total profit achieved is

3 + 7 = 10

.

Example 4:

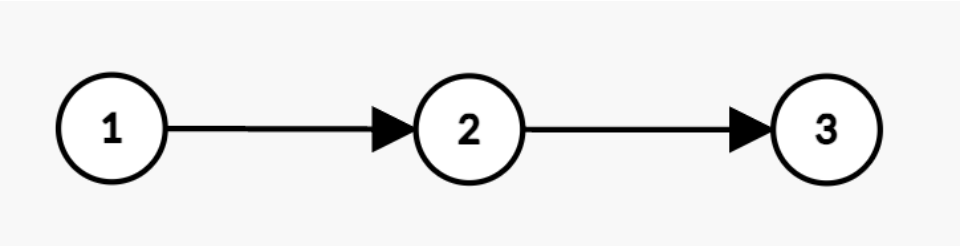
Input:

n = 3, present = [5,2,3], future = [8,5,6], hierarchy = [[1,2],[2,3]], budget = 7

Output:

12

Explanation:



Employee 1 buys the stock at price 5 and earns a profit of

8 - 5 = 3

.

Employee 2 would get a discounted price of

floor(2 / 2) = 1

and earns a profit of

5 - 1 = 4

.

Employee 3 would get a discounted price of

floor(3 / 2) = 1

and earns a profit of

6 - 1 = 5

.

The total cost becomes

5 + 1 + 1 = 7 <= budget

. Thus, the maximum total profit achieved is

3 + 4 + 5 = 12

.

Constraints:

1 <= n <= 160

present.length, future.length == n

1 <= present[i], future[i] <= 50

hierarchy.length == n - 1

hierarchy[i] == [u

i

, v

i

]

1 <= u

i

, v

i

<= n

u

i

!= v

i

1 <= budget <= 160

There are no duplicate edges.

Employee 1 is the direct or indirect boss of every employee.

The input graph

hierarchy

is

guaranteed

to have no cycles.

Code Snippets

C++:

```
class Solution {
public:
int maxProfit(int n, vector<int>& present, vector<int>& future,
vector<vector<int>>& hierarchy, int budget) {

}
};
```

Java:

```
class Solution {
public int maxProfit(int n, int[] present, int[] future, int[][] hierarchy,
int budget) {

}
}
```

Python3:

```
class Solution:
def maxProfit(self, n: int, present: List[int], future: List[int], hierarchy:
List[List[int]], budget: int) -> int:
```

Python:

```
class Solution(object):
def maxProfit(self, n, present, future, hierarchy, budget):
"""
:type n: int
:type present: List[int]
:type future: List[int]
:type hierarchy: List[List[int]]
:type budget: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} present
* @param {number[]} future
* @param {number[][]} hierarchy
```

```
* @param {number} budget
* @return {number}
*/
var maxProfit = function(n, present, future, hierarchy, budget) {

};
```

TypeScript:

```
function maxProfit(n: number, present: number[], future: number[], hierarchy:
number[][], budget: number): number {

};
```

C#:

```
public class Solution {
public int MaxProfit(int n, int[] present, int[] future, int[][] hierarchy,
int budget) {

}
}
```

C:

```
int maxProfit(int n, int* present, int presentSize, int* future, int
futureSize, int** hierarchy, int hierarchySize, int* hierarchyColSize, int
budget) {

}
```

Go:

```
func maxProfit(n int, present []int, future []int, hierarchy [][]int, budget
int) int {

}
```

Kotlin:

```
class Solution {
fun maxProfit(n: Int, present: IntArray, future: IntArray, hierarchy:
```

```
Array<IntArray>, budget: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxProfit(_ n: Int, _ present: [Int], _ future: [Int], _ hierarchy:
[[Int]], _ budget: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_profit(n: i32, present: Vec<i32>, future: Vec<i32>, hierarchy:
Vec<Vec<i32>>, budget: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} present
# @param {Integer[]} future
# @param {Integer[][]} hierarchy
# @param {Integer} budget
# @return {Integer}
def max_profit(n, present, future, hierarchy, budget)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $present
* @param Integer[] $future
```

```
* @param Integer[][] $hierarchy
* @param Integer $budget
* @return Integer
*/
function maxProfit($n, $present, $future, $hierarchy, $budget) {

}
}
```

Dart:

```
class Solution {
int maxProfit(int n, List<int> present, List<int> future, List<List<int>>
hierarchy, int budget) {

}
}
```

Scala:

```
object Solution {
def maxProfit(n: Int, present: Array[Int], future: Array[Int], hierarchy:
Array[Array[Int]], budget: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_profit(n :: integer, present :: [integer], future :: [integer],
hierarchy :: [[integer]], budget :: integer) :: integer
def max_profit(n, present, future, hierarchy, budget) do

end
end
```

Erlang:

```
-spec max_profit(N :: integer(), Present :: [integer()], Future ::
[integer()], Hierarchy :: [[integer()]], Budget :: integer()) -> integer().
max_profit(N, Present, Future, Hierarchy, Budget) ->
```

```
.
```

Racket:

```
(define/contract (max-profit n present future hierarchy budget)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?) (listof
(listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Profit from Trading Stocks with Discounts
* Difficulty: Hard
* Tags: array, tree, graph, 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 maxProfit(int n, vector<int>& present, vector<int>& future,
vector<vector<int>>& hierarchy, int budget) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Profit from Trading Stocks with Discounts
* Difficulty: Hard
* Tags: array, tree, graph, 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 maxProfit(int n, int[] present, int[] future, int[][] hierarchy,
int budget) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Profit from Trading Stocks with Discounts
Difficulty: Hard
Tags: array, tree, graph, 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 maxProfit(self, n: int, present: List[int], future: List[int], hierarchy:
List[List[int]], budget: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxProfit(self, n, present, future, hierarchy, budget):
"""
:type n: int
:type present: List[int]
:type future: List[int]
:type hierarchy: List[List[int]]
:type budget: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Profit from Trading Stocks with Discounts
* Difficulty: Hard
* Tags: array, tree, graph, 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} n
* @param {number[]} present
* @param {number[]} future
* @param {number[][]} hierarchy
* @param {number} budget
* @return {number}
*/
var maxProfit = function(n, present, future, hierarchy, budget) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Profit from Trading Stocks with Discounts
* Difficulty: Hard
* Tags: array, tree, graph, 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 maxProfit(n: number, present: number[], future: number[], hierarchy:
number[][], budget: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Profit from Trading Stocks with Discounts
* Difficulty: Hard
* Tags: array, tree, graph, 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 MaxProfit(int n, int[] present, int[] future, int[][] hierarchy,
int budget) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Profit from Trading Stocks with Discounts
* Difficulty: Hard
* Tags: array, tree, graph, 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 maxProfit(int n, int* present, int presentSize, int* future, int
futureSize, int** hierarchy, int hierarchySize, int* hierarchyColSize, int
budget) {

}
```

Go Solution:

```
// Problem: Maximum Profit from Trading Stocks with Discounts
// Difficulty: Hard
// Tags: array, tree, graph, 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 maxProfit(n int, present []int, future []int, hierarchy [][]int, budget
int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxProfit(n: Int, present: IntArray, future: IntArray, hierarchy:
Array<IntArray>, budget: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxProfit(_ n: Int, _ present: [Int], _ future: [Int], _ hierarchy:
[[Int]], _ budget: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Profit from Trading Stocks with Discounts
// Difficulty: Hard
// Tags: array, tree, graph, 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 max_profit(n: i32, present: Vec<i32>, future: Vec<i32>, hierarchy:
Vec<Vec<i32>>, budget: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} present
# @param {Integer[]} future
# @param {Integer[][]} hierarchy
# @param {Integer} budget
# @return {Integer}
def max_profit(n, present, future, hierarchy, budget)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $present
* @param Integer[] $future
* @param Integer[][] $hierarchy
* @param Integer $budget
* @return Integer
*/
function maxProfit($n, $present, $future, $hierarchy, $budget) {

}
}
```

Dart Solution:

```
class Solution {
int maxProfit(int n, List<int> present, List<int> future, List<List<int>>
hierarchy, int budget) {

}
}
```

Scala Solution:

```
object Solution {
def maxProfit(n: Int, present: Array[Int], future: Array[Int], hierarchy:
Array[Array[Int]], budget: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_profit(n :: integer, present :: [integer], future :: [integer],
hierarchy :: [[integer]], budget :: integer) :: integer
def max_profit(n, present, future, hierarchy, budget) do

end
end
```

Erlang Solution:

```
-spec max_profit(N :: integer(), Present :: [integer()], Future ::
[integer()], Hierarchy :: [[integer()]], Budget :: integer()) -> integer().
max_profit(N, Present, Future, Hierarchy, Budget) ->
.
```

Racket Solution:

```
(define/contract (max-profit n present future hierarchy budget)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?) (listof
(listof exact-integer?)) exact-integer? exact-integer?)
)
```