Problem 465: Optimal Account Balancing

Problem Information

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

Problem Description

You are given an array of transactions

transactions

where

transactions[i] = [from

i

, to

i

, amount

i

]

indicates that the person with

ID = from

i

gave

amount

i

$

to the person with

ID = to

i

.

Return

the minimum number of transactions required to settle the debt

.

Example 1:

Input:

transactions = [[0,1,10],[2,0,5]]

Output:

2

Explanation:

Person #0 gave person #1 $10. Person #2 gave person #0 $5. Two transactions are needed. One way to settle the debt is person #1 pays person #0 and #2 $5 each.

Example 2:

Input:

transactions = [[0,1,10],[1,0,1],[1,2,5],[2,0,5]]

Output:

1

Explanation:

Person #0 gave person #1 $10. Person #1 gave person #0 $1. Person #1 gave person #2 $5. Person #2 gave person #0 $5. Therefore, person #1 only need to give person #0 $4, and all debt is settled.

Constraints:

1 <= transactions.length <= 8

transactions[i].length == 3

0 <= from

i

, to

i

< 12

from

i

!= to

i

1 <= amount

i

<= 100

Code Snippets

C++:

```
class Solution {
public:
int minTransfers(vector<vector<int>>& transactions) {

}
};
```

Java:

```
class Solution {
public int minTransfers(int[][] transactions) {

}
}
```

Python3:

```
class Solution:
def minTransfers(self, transactions: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minTransfers(self, transactions):
"""
:type transactions: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} transactions
```

```
* @return {number}
*/
var minTransfers = function(transactions) {

};
```

TypeScript:

```
function minTransfers(transactions: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinTransfers(int[][] transactions) {

}
}
```

C:

```
int minTransfers(int** transactions, int transactionsSize, int*
transactionsColSize) {

}
```

Go:

```
func minTransfers(transactions [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minTransfers(transactions: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minTransfers(_ transactions: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_transfers(transactions: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} transactions
# @return {Integer}
def min_transfers(transactions)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $transactions
* @return Integer
*/
function minTransfers($transactions) {

}
}
```

Dart:

```
class Solution {
int minTransfers(List<List<int>> transactions) {

}
}
```

Scala:

```
object Solution {
def minTransfers(transactions: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_transfers(transactions :: [[integer]]) :: integer
def min_transfers(transactions) do

end
end
```

Erlang:

```
-spec min_transfers(Transactions :: [[integer()]]) -> integer().
min_transfers(Transactions) ->
.
```

Racket:

```
(define/contract (min-transfers transactions)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Optimal Account Balancing
* Difficulty: Hard
* Tags: array, dp
*
* 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 minTransfers(vector<vector<int>>& transactions) {

}
};
```

Java Solution:

```
/**
* Problem: Optimal Account Balancing
* Difficulty: Hard
* Tags: array, dp
*
* 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 minTransfers(int[][] transactions) {

}
}
```

Python3 Solution:

```
"""
Problem: Optimal Account Balancing
Difficulty: Hard
Tags: array, dp

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 minTransfers(self, transactions: List[List[int]]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def minTransfers(self, transactions):
"""
:type transactions: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Optimal Account Balancing
* Difficulty: Hard
* Tags: array, dp
*
* 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[][]} transactions
* @return {number}
*/
var minTransfers = function(transactions) {

};
```

TypeScript Solution:

```
/**
* Problem: Optimal Account Balancing
* Difficulty: Hard
* Tags: array, dp
*
* 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 minTransfers(transactions: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Optimal Account Balancing
* Difficulty: Hard
* Tags: array, dp
*
* 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 MinTransfers(int[][] transactions) {

}
}
```

C Solution:

```
/*
* Problem: Optimal Account Balancing
* Difficulty: Hard
* Tags: array, dp
*
* 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 minTransfers(int** transactions, int transactionsSize, int*
transactionsColSize) {

}
```

Go Solution:

```
// Problem: Optimal Account Balancing
// Difficulty: Hard
// Tags: array, dp
//
// 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 minTransfers(transactions [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minTransfers(transactions: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minTransfers(_ transactions: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Optimal Account Balancing
// Difficulty: Hard
// Tags: array, dp
//
// 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 min_transfers(transactions: Vec<Vec<i32>>) -> i32 {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer[][]} transactions
# @return {Integer}
def min_transfers(transactions)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $transactions
* @return Integer
*/
function minTransfers($transactions) {

}
}
```

Dart Solution:

```
class Solution {
int minTransfers(List<List<int>> transactions) {

}
}
```

Scala Solution:

```
object Solution {
def minTransfers(transactions: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_transfers(transactions :: [[integer]]) :: integer
def min_transfers(transactions) do

end
end
```

Erlang Solution:

```
-spec min_transfers(Transactions :: [[integer()]]) -> integer().
min_transfers(Transactions) ->
.
```

Racket Solution:

```
(define/contract (min-transfers transactions)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```