Problem 1058: Minimize Rounding Error to Meet Target

Problem Information

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

Problem Description

Given an array of

prices

[p

1

,p

2

...,p

n

]

and a

target

, round each price

p

i

to

Round

i

(p

i

)

so that the rounded array

[Round

1

(p

1

),Round

2

(p

2

)...,Round

n

(p

n

)]

sums to the given

target

. Each operation

Round

i

(p

i

)

could be either

Floor(p

i

)

or

Ceil(p

i

)

.

Return the string

"-1"

if the rounded array is impossible to sum to

target

. Otherwise, return the smallest rounding error, which is defined as

Σ |Round

i

(p

i

) - (p

i

)|

for

i

from

1

to

n

, as a string with three places after the decimal.

Example 1:

Input:

prices = ["0.700","2.800","4.900"], target = 8

Output:

"1.000"

Explanation:

Use Floor, Ceil and Ceil operations to get (0.7 - 0) + (3 - 2.8) + (5 - 4.9) = 0.7 + 0.2 + 0.1 = 1.0
.

Example 2:

Input:

prices = ["1.500","2.500","3.500"], target = 10

Output:

"-1"

Explanation:

It is impossible to meet the target.

Example 3:

Input:

prices = ["1.500","2.500","3.500"], target = 9

Output:

"1.500"

Constraints:

1 <= prices.length <= 500

Each string

prices[i]

represents a real number in the range

[0.0, 1000.0]

and has exactly 3 decimal places.

0 <= target <= 10

6

Code Snippets

C++:

```
class Solution {
public:
string minimizeError(vector<string>& prices, int target) {

}
};
```

Java:

```
class Solution {
public String minimizeError(String[] prices, int target) {

}
}
```

Python3:

```
class Solution:
def minimizeError(self, prices: List[str], target: int) -> str:
```

Python:

```
class Solution(object):
def minimizeError(self, prices, target):
"""
:type prices: List[str]
:type target: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string[]} prices
* @param {number} target
* @return {string}
*/
var minimizeError = function(prices, target) {

};
```

TypeScript:

```
function minimizeError(prices: string[], target: number): string {

};
```

C#:

```
public class Solution {
public string MinimizeError(string[] prices, int target) {

}
}
```

C:

```
char* minimizeError(char** prices, int pricesSize, int target) {

}
```

Go:

```
func minimizeError(prices []string, target int) string {

}
```

Kotlin:

```
class Solution {
fun minimizeError(prices: Array<String>, target: Int): String {

}
}
```

Swift:

```
class Solution {
func minimizeError(_ prices: [String], _ target: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn minimize_error(prices: Vec<String>, target: i32) -> String {

}
}
```

Ruby:

```
# @param {String[]} prices
# @param {Integer} target
# @return {String}
def minimize_error(prices, target)

end
```

PHP:

```
class Solution {

/**
* @param String[] $prices
```

```
* @param Integer $target
* @return String
*/
function minimizeError($prices, $target) {

}
}
```

Dart:

```
class Solution {
String minimizeError(List<String> prices, int target) {

}
}
```

Scala:

```
object Solution {
def minimizeError(prices: Array[String], target: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimize_error(prices :: [String.t], target :: integer) :: String.t
def minimize_error(prices, target) do

end
end
```

Erlang:

```
-spec minimize_error(Prices :: [unicode:unicode_binary()], Target ::
integer()) -> unicode:unicode_binary().
minimize_error(Prices, Target) ->
.
```

Racket:

```
(define/contract (minimize-error prices target)
(-> (listof string?) exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize Rounding Error to Meet Target
* Difficulty: Medium
* Tags: array, string, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
string minimizeError(vector<string>& prices, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize Rounding Error to Meet Target
* Difficulty: Medium
* Tags: array, string, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String minimizeError(String[] prices, int target) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimize Rounding Error to Meet Target
Difficulty: Medium
Tags: array, string, greedy, math, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def minimizeError(self, prices: List[str], target: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimizeError(self, prices, target):
"""
:type prices: List[str]
:type target: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Minimize Rounding Error to Meet Target
* Difficulty: Medium
* Tags: array, string, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
/**
* @param {string[]} prices
* @param {number} target
* @return {string}
*/
var minimizeError = function(prices, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize Rounding Error to Meet Target
* Difficulty: Medium
* Tags: array, string, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function minimizeError(prices: string[], target: number): string {

};
```

C# Solution:

```
/*
* Problem: Minimize Rounding Error to Meet Target
* Difficulty: Medium
* Tags: array, string, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string MinimizeError(string[] prices, int target) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimize Rounding Error to Meet Target
* Difficulty: Medium
* Tags: array, string, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* minimizeError(char** prices, int pricesSize, int target) {

}
```

Go Solution:

```
// Problem: Minimize Rounding Error to Meet Target
// Difficulty: Medium
// Tags: array, string, greedy, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func minimizeError(prices []string, target int) string {

}
```

Kotlin Solution:

```
class Solution {
fun minimizeError(prices: Array<String>, target: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func minimizeError(_ prices: [String], _ target: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Minimize Rounding Error to Meet Target
// Difficulty: Medium
// Tags: array, string, greedy, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn minimize_error(prices: Vec<String>, target: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {String[]} prices
# @param {Integer} target
# @return {String}
def minimize_error(prices, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $prices
* @param Integer $target
* @return String
*/
function minimizeError($prices, $target) {
```

```
}
}
```

Dart Solution:

```
class Solution {
String minimizeError(List<String> prices, int target) {

}
}
```

Scala Solution:

```
object Solution {
def minimizeError(prices: Array[String], target: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimize_error(prices :: [String.t], target :: integer) :: String.t
def minimize_error(prices, target) do

end
end
```

Erlang Solution:

```
-spec minimize_error(Prices :: [unicode:unicode_binary()], Target ::
integer()) -> unicode:unicode_binary().
minimize_error(Prices, Target) ->
.
```

Racket Solution:

```
(define/contract (minimize-error prices target)
(-> (listof string?) exact-integer? string?)
)
```