Problem 964: Least Operators to Express Number

Problem Information

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

Problem Description

Given a single positive integer

x

, we will write an expression of the form

x (op1) x (op2) x (op3) x ...

where each operator

op1

,

op2

, etc. is either addition, subtraction, multiplication, or division (

+

,

-

,

*

, or

/)

. For example, with

x = 3

, we might write

3 * 3 / 3 + 3 - 3

which is a value of

3

.

When writing such an expression, we adhere to the following conventions:

The division operator (

/

) returns rational numbers.

There are no parentheses placed anywhere.

We use the usual order of operations: multiplication and division happen before addition and subtraction.

It is not allowed to use the unary negation operator (

-

). For example, "

x - x

" is a valid expression as it only uses subtraction, but "

-x + x

" is not because it uses negation.

We would like to write an expression with the least number of operators such that the expression equals the given

target

. Return the least number of operators used.

Example 1:

Input:

x = 3, target = 19

Output:

5

Explanation:

3 * 3 + 3 * 3 + 3 / 3. The expression contains 5 operations.

Example 2:

Input:

x = 5, target = 501

Output:

8

Explanation:

5 * 5 * 5 * 5 - 5 * 5 * 5 + 5 / 5. The expression contains 8 operations.

Example 3:

Input:

x = 100, target = 100000000

Output:

3

Explanation:

100 * 100 * 100 * 100. The expression contains 3 operations.

Constraints:

2 <= x <= 100

1 <= target <= 2 * 10

8

Code Snippets

C++:

```
class Solution {
public:
int leastOpsExpressTarget(int x, int target) {

}
};
```

Java:

```
class Solution {
public int leastOpsExpressTarget(int x, int target) {

}
}
```

Python3:

```
class Solution:
def leastOpsExpressTarget(self, x: int, target: int) -> int:
```

Python:

```
class Solution(object):
def leastOpsExpressTarget(self, x, target):
"""
:type x: int
:type target: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} x
* @param {number} target
* @return {number}
*/
var leastOpsExpressTarget = function(x, target) {

};
```

TypeScript:

```
function leastOpsExpressTarget(x: number, target: number): number {

};
```

C#:

```
public class Solution {
public int LeastOpsExpressTarget(int x, int target) {

}
}
```

C:

```
int leastOpsExpressTarget(int x, int target) {

}
```

Go:

```
func leastOpsExpressTarget(x int, target int) int {

}
```

Kotlin:

```
class Solution {
fun leastOpsExpressTarget(x: Int, target: Int): Int {

}
}
```

Swift:

```
class Solution {
func leastOpsExpressTarget(_ x: Int, _ target: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn least_ops_express_target(x: i32, target: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} x
# @param {Integer} target
# @return {Integer}
def least_ops_express_target(x, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer $x
* @param Integer $target
* @return Integer
*/
function leastOpsExpressTarget($x, $target) {

}
}
```

Dart:

```
class Solution {
int leastOpsExpressTarget(int x, int target) {

}
}
```

Scala:

```
object Solution {
def leastOpsExpressTarget(x: Int, target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec least_ops_express_target(x :: integer, target :: integer) :: integer
def least_ops_express_target(x, target) do
```

```

end
end
```

Erlang:

```
-spec least_ops_express_target(X :: integer(), Target :: integer()) ->
integer().
least_ops_express_target(X, Target) ->
.
```

Racket:

```
(define/contract (least-ops-express-target x target)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Least Operators to Express Number
* Difficulty: Hard
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int leastOpsExpressTarget(int x, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Least Operators to Express Number
* Difficulty: Hard
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int leastOpsExpressTarget(int x, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Least Operators to Express Number
Difficulty: Hard
Tags: dp, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def leastOpsExpressTarget(self, x: int, target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def leastOpsExpressTarget(self, x, target):
"""
:type x: int
:type target: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Least Operators to Express Number
* Difficulty: Hard
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} x
* @param {number} target
* @return {number}
*/
var leastOpsExpressTarget = function(x, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Least Operators to Express Number
* Difficulty: Hard
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function leastOpsExpressTarget(x: number, target: number): number {

};
```

C# Solution:

```
/*
* Problem: Least Operators to Express Number
* Difficulty: Hard
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int LeastOpsExpressTarget(int x, int target) {

}
}
```

C Solution:

```
/*
* Problem: Least Operators to Express Number
* Difficulty: Hard
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int leastOpsExpressTarget(int x, int target) {

}
```

Go Solution:

```
// Problem: Least Operators to Express Number
// Difficulty: Hard
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func leastOpsExpressTarget(x int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun leastOpsExpressTarget(x: Int, target: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func leastOpsExpressTarget(_ x: Int, _ target: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Least Operators to Express Number
// Difficulty: Hard
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn least_ops_express_target(x: i32, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} x
# @param {Integer} target
# @return {Integer}
def least_ops_express_target(x, target)
```

```

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @param Integer $target
* @return Integer
*/
function leastOpsExpressTarget($x, $target) {

}
}
```

Dart Solution:

```
class Solution {
int leastOpsExpressTarget(int x, int target) {

}
}
```

Scala Solution:

```
object Solution {
def leastOpsExpressTarget(x: Int, target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec least_ops_express_target(x :: integer, target :: integer) :: integer
def least_ops_express_target(x, target) do

end
end
```

Erlang Solution:

```
-spec least_ops_express_target(X :: integer(), Target :: integer()) ->
integer().
least_ops_express_target(X, Target) ->
.
```

Racket Solution:

```
(define/contract (least-ops-express-target x target)
(-> exact-integer? exact-integer? exact-integer?)
)
```