Problem 282: Expression Add Operators

Problem Information

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

Problem Description

Given a string

num

that contains only digits and an integer

target

, return

all possibilities

to insert the binary operators

'+'

,

'-'

, and/or

'*'

between the digits of

num

so that the resultant expression evaluates to the

target

value

.

Note that operands in the returned expressions

should not

contain leading zeros.

Note

that a number can contain multiple digits.

Example 1:

Input:

num = "123", target = 6

Output:

["1*2*3","1+2+3"]

Explanation:

Both "1*2*3" and "1+2+3" evaluate to 6.

Example 2:

Input:

num = "232", target = 8

Output:

["2*3+2","2+3*2"]

Explanation:

Both "2*3+2" and "2+3*2" evaluate to 8.

Example 3:

Input:

num = "3456237490", target = 9191

Output:

[]

Explanation:

There are no expressions that can be created from "3456237490" to evaluate to 9191.

Constraints:

1 <= num.length <= 10

num

consists of only digits.

-2

31

<= target <= 2

31

- 1

Code Snippets

C++:

```
class Solution {
public:
vector<string> addOperators(string num, int target) {

}
};
```

Java:

```
class Solution {
public List<String> addOperators(String num, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} num
* @param {number} target
```

```
* @return {string[]}
*/
var addOperators = function(num, target) {

};
```

TypeScript:

```
function addOperators(num: string, target: number): string[] {

};
```

C#:

```
public class Solution {
public IList<string> AddOperators(string num, int target) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** addOperators(char* num, int target, int* returnSize) {

}
```

Go:

```
func addOperators(num string, target int) []string {

}
```

Kotlin:

```
class Solution {
fun addOperators(num: String, target: Int): List<String> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param String $num
* @param Integer $target
* @return String[]
*/
function addOperators($num, $target) {

}
}
```

Dart:

```
class Solution {
List<String> addOperators(String num, int target) {
```

```
}
}
```

Scala:

```
object Solution {
def addOperators(num: String, target: Int): List[String] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec add_operators(Num :: unicode:unicode_binary(), Target :: integer()) ->
[unicode:unicode_binary()].
add_operators(Num, Target) ->
.
```

Racket:

```
(define/contract (add-operators num target)
(-> string? exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Expression Add Operators
```

```
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<string> addOperators(string num, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Expression Add Operators
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<String> addOperators(String num, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Expression Add Operators
Difficulty: Hard
Tags: string, math

Approach: String manipulation with hash map or two pointers
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Expression Add Operators
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} num
* @param {number} target
* @return {string[]}
*/
var addOperators = function(num, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Expression Add Operators
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function addOperators(num: string, target: number): string[] {

};
```

C# Solution:

```
/*
* Problem: Expression Add Operators
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<string> AddOperators(string num, int target) {

}
}
```

C Solution:

```
/*
* Problem: Expression Add Operators
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** addOperators(char* num, int target, int* returnSize) {

}
```

Go Solution:

```
// Problem: Expression Add Operators
// Difficulty: Hard
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func addOperators(num string, target int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun addOperators(num: String, target: Int): List<String> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Expression Add Operators
// Difficulty: Hard
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param String $num
* @param Integer $target
* @return String[]
*/
function addOperators($num, $target) {

}
}
```

Dart Solution:

```
class Solution {
List<String> addOperators(String num, int target) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def addOperators(num: String, target: Int): List[String] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec add_operators(Num :: unicode:unicode_binary(), Target :: integer()) ->
[unicode:unicode_binary()].
add_operators(Num, Target) ->
.
```

Racket Solution:

```
(define/contract (add-operators num target)
(-> string? exact-integer? (listof string?))
)
```