Problem 241: Different Ways to Add Parentheses

Problem Information

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

Problem Description

Given a string

expression

of numbers and operators, return

all possible results from computing all the different possible ways to group numbers and operators

. You may return the answer in

any order

.

The test cases are generated such that the output values fit in a 32-bit integer and the number of different results does not exceed

10

4

.

Example 1:

Input:

expression = "2-1-1"

Output:

[0,2]

Explanation:

((2-1)-1) = 0 (2-(1-1)) = 2

Example 2:

Input:

expression = "2*3-4*5"

Output:

[-34,-14,-10,-10,10]

Explanation:

(2*(3-(4*5))) = -34 ((2*3)-(4*5)) = -14 ((2*(3-4))*5) = -10 (2*((3-4)*5)) = -10 (((2*3)-4)*5) = 10

Constraints:

1 <= expression.length <= 20

expression

consists of digits and the operator

'+'

,

'-'

, and

'*'

.

All the integer values in the input expression are in the range

[0, 99]

.

The integer values in the input expression do not have a leading

'-'

or

'+'

denoting the sign.

Code Snippets

C++:

```
class Solution {
public:
vector<int> diffWaysToCompute(string expression) {

}
};
```

Java:

```
class Solution {
public List<Integer> diffWaysToCompute(String expression) {
}
}
```

Python3:

```
class Solution:
def diffWaysToCompute(self, expression: str) -> List[int]:
```

Python:

```
class Solution(object):
def diffWaysToCompute(self, expression):
"""
:type expression: str
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string} expression
* @return {number[]}
*/
var diffWaysToCompute = function(expression) {

};
```

TypeScript:

```
function diffWaysToCompute(expression: string): number[] {

};
```

C#:

```
public class Solution {
public IList<int> DiffWaysToCompute(string expression) {

}
}
```

C:

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

}
```

Go:

```
func diffWaysToCompute(expression string) []int {

}
```

Kotlin:

```
class Solution {
fun diffWaysToCompute(expression: String): List<Int> {

}
}
```

Swift:

```
class Solution {
func diffWaysToCompute(_ expression: String) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn diff_ways_to_compute(expression: String) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {String} expression
# @return {Integer[]}
```

```
def diff_ways_to_compute(expression)

end
```

PHP:

```
class Solution {

/**
* @param String $expression
* @return Integer[]
*/
function diffWaysToCompute($expression) {

}
}
```

Dart:

```
class Solution {
List<int> diffWaysToCompute(String expression) {

}
}
```

Scala:

```
object Solution {
def diffWaysToCompute(expression: String): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec diff_ways_to_compute(expression :: String.t) :: [integer]
def diff_ways_to_compute(expression) do

end
end
```

Erlang:

```
-spec diff_ways_to_compute(Expression :: unicode:unicode_binary()) ->
[integer()].
diff_ways_to_compute(Expression) ->
.
```

Racket:

```
(define/contract (diff-ways-to-compute expression)
(-> string? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Different Ways to Add Parentheses
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
vector<int> diffWaysToCompute(string expression) {

}
};
```

Java Solution:

```
/**
* Problem: Different Ways to Add Parentheses
* Difficulty: Medium
* Tags: string, dp, math
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public List<Integer> diffWaysToCompute(String expression) {

}
}
```

Python3 Solution:

```
"""
Problem: Different Ways to Add Parentheses
Difficulty: Medium
Tags: string, dp, math

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

```
class Solution(object):
def diffWaysToCompute(self, expression):
"""
:type expression: str
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Different Ways to Add Parentheses
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} expression
* @return {number[]}
*/
var diffWaysToCompute = function(expression) {

};
```

TypeScript Solution:

```
/**
* Problem: Different Ways to Add Parentheses
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function diffWaysToCompute(expression: string): number[] {

};
```

C# Solution:

```
/*
* Problem: Different Ways to Add Parentheses
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public IList<int> DiffWaysToCompute(string expression) {

}
}
```

C Solution:

```
/*
* Problem: Different Ways to Add Parentheses
* Difficulty: Medium
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Different Ways to Add Parentheses
// Difficulty: Medium
// Tags: string, dp, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func diffWaysToCompute(expression string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun diffWaysToCompute(expression: String): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func diffWaysToCompute(_ expression: String) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Different Ways to Add Parentheses
// Difficulty: Medium
// Tags: string, dp, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn diff_ways_to_compute(expression: String) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String} expression
# @return {Integer[]}
def diff_ways_to_compute(expression)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $expression
* @return Integer[]
*/
function diffWaysToCompute($expression) {

}
}
```

Dart Solution:

```
class Solution {
List<int> diffWaysToCompute(String expression) {

}
}
```

Scala Solution:

```
object Solution {
def diffWaysToCompute(expression: String): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec diff_ways_to_compute(expression :: String.t) :: [integer]
def diff_ways_to_compute(expression) do

end
end
```

Erlang Solution:

```
-spec diff_ways_to_compute(Expression :: unicode:unicode_binary()) ->
[integer()].
diff_ways_to_compute(Expression) ->
.
```

Racket Solution:

```
(define/contract (diff-ways-to-compute expression)
(-> string? (listof exact-integer?))
)
```