Problem 2232: Minimize Result by Adding Parentheses to Expression

Problem Information

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

Problem Description

You are given a

0-indexed

string

expression

of the form

"<num1>+<num2>"

where

<num1>

and

<num2>

represent positive integers.

Add a pair of parentheses to

expression

such that after the addition of parentheses,

expression

is a

valid

mathematical expression and evaluates to the

smallest

possible value. The left parenthesis

must

be added to the left of

'+'

and the right parenthesis

must

be added to the right of

'+'

.

Return

expression

after adding a pair of parentheses such that

expression

evaluates to the

smallest

possible value.

If there are multiple answers that yield the same result, return any of them.

The input has been generated such that the original value of

expression

, and the value of

expression

after adding any pair of parentheses that meets the requirements fits within a signed 32-bit integer.

Example 1:

Input:

expression = "247+38"

Output:

"2(47+38)"

Explanation:

The

expression

evaluates to 2 * (47 + 38) = 2 * 85 = 170. Note that "2(4)7+38" is invalid because the right parenthesis must be to the right of the

'+'

. It can be shown that 170 is the smallest possible value.

Example 2:

Input:

expression = "12+34"

Output:

"1(2+3)4"

Explanation:

The expression evaluates to 1 * (2 + 3) * 4 = 1 * 5 * 4 = 20.

Example 3:

Input:

expression = "999+999"

Output:

"(999+999)"

Explanation:

The

expression

evaluates to 999 + 999 = 1998.

Constraints:

3 <= expression.length <= 10

expression

consists of digits from

'1'

to

'9'

and

'+'

.

expression

starts and ends with digits.

expression

contains exactly one

'+'

.

The original value of

expression

, and the value of

expression

after adding any pair of parentheses that meets the requirements fits within a signed 32-bit integer.

Code Snippets

C++:

```
class Solution {
public:
string minimizeResult(string expression) {

}
};
```

Java:

```
class Solution {
public String minimizeResult(String expression) {

}
}
```

Python3:

```
class Solution:
def minimizeResult(self, expression: str) -> str:
```

Python:

```
class Solution(object):
def minimizeResult(self, expression):
"""
:type expression: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} expression
* @return {string}
*/
var minimizeResult = function(expression) {

};
```

TypeScript:

```
function minimizeResult(expression: string): string {

};
```

C#:

```
public class Solution {
public string MinimizeResult(string expression) {

}
}
```

C:

```
char* minimizeResult(char* expression) {

}
```

Go:

```
func minimizeResult(expression string) string {

}
```

Kotlin:

```
class Solution {
fun minimizeResult(expression: String): String {

}
}
```

Swift:

```
class Solution {
func minimizeResult(_ expression: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn minimize_result(expression: String) -> String {

}
}
```

Ruby:

```
# @param {String} expression
# @return {String}
def minimize_result(expression)

end
```

PHP:

```
class Solution {

/**
* @param String $expression
* @return String
*/
function minimizeResult($expression) {

}
}
```

Dart:

```
class Solution {
String minimizeResult(String expression) {

}
}
```

Scala:

```
object Solution {
def minimizeResult(expression: String): String = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec minimize_result(expression :: String.t) :: String.t
def minimize_result(expression) do

end
end
```

Erlang:

```
-spec minimize_result(Expression :: unicode:unicode_binary()) ->
unicode:unicode_binary().
minimize_result(Expression) ->
.
```

Racket:

```
(define/contract (minimize-result expression)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize Result by Adding Parentheses to Expression
* Difficulty: Medium
* 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:
```

```
string minimizeResult(string expression) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize Result by Adding Parentheses to Expression
* Difficulty: Medium
* 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 String minimizeResult(String expression) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize Result by Adding Parentheses to Expression
Difficulty: Medium
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 minimizeResult(self, expression: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimizeResult(self, expression):
"""
:type expression: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Minimize Result by Adding Parentheses to Expression
* Difficulty: Medium
* 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} expression
* @return {string}
*/
var minimizeResult = function(expression) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize Result by Adding Parentheses to Expression
* Difficulty: Medium
* 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 minimizeResult(expression: string): string {

};
```

C# Solution:

```
/*
* Problem: Minimize Result by Adding Parentheses to Expression
* Difficulty: Medium
* 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 string MinimizeResult(string expression) {

}
}
```

C Solution:

```
/*
* Problem: Minimize Result by Adding Parentheses to Expression
* Difficulty: Medium
* 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
*/

char* minimizeResult(char* expression) {

}
```

Go Solution:

```
// Problem: Minimize Result by Adding Parentheses to Expression
// Difficulty: Medium
// 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 minimizeResult(expression string) string {

}
```

Kotlin Solution:

```
class Solution {
fun minimizeResult(expression: String): String {

}
}
```

Swift Solution:

```
class Solution {
func minimizeResult(_ expression: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Minimize Result by Adding Parentheses to Expression
// Difficulty: Medium
// 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 minimize_result(expression: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} expression
# @return {String}
def minimize_result(expression)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $expression
* @return String
*/
function minimizeResult($expression) {

}
}
```

Dart Solution:

```
class Solution {
String minimizeResult(String expression) {

}
}
```

Scala Solution:

```
object Solution {
def minimizeResult(expression: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimize_result(expression :: String.t) :: String.t
def minimize_result(expression) do

end
```

```
end
```

Erlang Solution:

```
-spec minimize_result(Expression :: unicode:unicode_binary()) ->
unicode:unicode_binary().
minimize_result(Expression) ->
.
```

Racket Solution:

```
(define/contract (minimize-result expression)
(-> string? string?)
)
```