Problem 2117: Abbreviating the Product of a Range

Problem Information

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

Problem Description

You are given two positive integers

left

and

right

with

left <= right

. Calculate the

product

of all integers in the

inclusive

range

[left, right]

.

Since the product may be very large, you will

abbreviate

it following these steps:

Count all

trailing

zeros in the product and

remove

them. Let us denote this count as

C

.

For example, there are

3

trailing zeros in

1000

, and there are

0

trailing zeros in

546

.

Denote the remaining number of digits in the product as

d

. If

d > 10

, then express the product as

<pre>...<suf>

where

<pre>

denotes the

first

5

digits of the product, and

<suf>

denotes the

last

5

digits of the product

after

removing all trailing zeros. If

d <= 10

, we keep it unchanged.

For example, we express

1234567654321

as

12345...54321

, but

1234567

is represented as

1234567

.

Finally, represent the product as a

string

"<pre>...<suf>eC"

.

For example,

12345678987600000

will be represented as

"12345...89876e5"

.

Return

a string denoting the

abbreviated product

of all integers in the

inclusive

range

[left, right]

.

Example 1:

Input:

left = 1, right = 4

Output:

"24e0"

Explanation:

The product is 1 × 2 × 3 × 4 = 24. There are no trailing zeros, so 24 remains the same. The abbreviation will end with "e0". Since the number of digits is 2, which is less than 10, we do not have to abbreviate it further. Thus, the final representation is "24e0".

Example 2:

Input:

left = 2, right = 11

Output:

"399168e2"

Explanation:

The product is 39916800. There are 2 trailing zeros, which we remove to get 399168. The abbreviation will end with "e2". The number of digits after removing the trailing zeros is 6, so we do not abbreviate it further. Hence, the abbreviated product is "399168e2".

Example 3:

Input:

left = 371, right = 375

Output:

"7219856259e3"

Explanation:

The product is 7219856259000.

Constraints:

1 <= left <= right <= 10

4

Code Snippets

C++:

```
class Solution {
public:
string abbreviateProduct(int left, int right) {

}
};
```

Java:

```
class Solution {
public String abbreviateProduct(int left, int right) {

}
}
```

Python3:

```
class Solution:
def abbreviateProduct(self, left: int, right: int) -> str:
```

Python:

```
class Solution(object):
def abbreviateProduct(self, left, right):
"""
:type left: int
:type right: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number} left
* @param {number} right
* @return {string}
*/
var abbreviateProduct = function(left, right) {

};
```

TypeScript:

```
function abbreviateProduct(left: number, right: number): string {

};
```

C#:

```
public class Solution {
public string AbbreviateProduct(int left, int right) {

}
}
```

C:

```
char* abbreviateProduct(int left, int right) {

}
```

Go:

```
func abbreviateProduct(left int, right int) string {

}
```

Kotlin:

```
class Solution {
fun abbreviateProduct(left: Int, right: Int): String {

}
}
```

Swift:

```
class Solution {
func abbreviateProduct(_ left: Int, _ right: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn abbreviate_product(left: i32, right: i32) -> String {

}
}
```

Ruby:

```
# @param {Integer} left
# @param {Integer} right
# @return {String}
def abbreviate_product(left, right)

end
```

PHP:

```
class Solution {

/**
* @param Integer $left
* @param Integer $right
* @return String
*/
function abbreviateProduct($left, $right) {

}
}
```

Dart:

```
class Solution {
String abbreviateProduct(int left, int right) {

}
}
```

Scala:

```
object Solution {
def abbreviateProduct(left: Int, right: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec abbreviate_product(left :: integer, right :: integer) :: String.t
def abbreviate_product(left, right) do
```

```
end
end
```

Erlang:

```
-spec abbreviate_product(Left :: integer(), Right :: integer()) ->
unicode:unicode_binary().
abbreviate_product(Left, Right) ->
.
```

Racket:

```
(define/contract (abbreviate-product left right)
(-> exact-integer? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Abbreviating the Product of a Range
* 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:
string abbreviateProduct(int left, int right) {

}
};
```

Java Solution:

```
/**
* Problem: Abbreviating the Product of a Range
* 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 String abbreviateProduct(int left, int right) {

}
}
```

Python3 Solution:

```
"""
Problem: Abbreviating the Product of a Range
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 abbreviateProduct(self, left: int, right: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def abbreviateProduct(self, left, right):
"""
:type left: int
:type right: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Abbreviating the Product of a Range
* 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 {number} left
* @param {number} right
* @return {string}
*/
var abbreviateProduct = function(left, right) {

};
```

TypeScript Solution:

```
/**
* Problem: Abbreviating the Product of a Range
* 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 abbreviateProduct(left: number, right: number): string {

};
```

C# Solution:

```
/*
* Problem: Abbreviating the Product of a Range
* 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 string AbbreviateProduct(int left, int right) {

}
}
```

C Solution:

```
/*
* Problem: Abbreviating the Product of a Range
* 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
*/

char* abbreviateProduct(int left, int right) {

}
```

Go Solution:

```
// Problem: Abbreviating the Product of a Range
// 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 abbreviateProduct(left int, right int) string {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun abbreviateProduct(left: Int, right: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func abbreviateProduct(_ left: Int, _ right: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Abbreviating the Product of a Range
// 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 abbreviate_product(left: i32, right: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer} left
# @param {Integer} right
# @return {String}
def abbreviate_product(left, right)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $left
* @param Integer $right
* @return String
*/
function abbreviateProduct($left, $right) {

}
}
```

Dart Solution:

```
class Solution {
String abbreviateProduct(int left, int right) {

}
}
```

Scala Solution:

```
object Solution {
def abbreviateProduct(left: Int, right: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec abbreviate_product(left :: integer, right :: integer) :: String.t
def abbreviate_product(left, right) do

end
end
```

Erlang Solution:

```
-spec abbreviate_product(Left :: integer(), Right :: integer()) ->
unicode:unicode_binary().
abbreviate_product(Left, Right) ->
.
```

Racket Solution:

```
(define/contract (abbreviate-product left right)
(-> exact-integer? exact-integer? string?)
)
```