Problem 3348: Smallest Divisible Digit Product II

Problem Information

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

Problem Description

You are given a string

num

which represents a

positive

integer, and an integer

t

.

A number is called

zero-free

if

none

of its digits are 0.

Return a string representing the

smallest

zero-free

number greater than or equal to

num

such that the

product of its digits

is divisible by

t

. If no such number exists, return

"-1"

.

Example 1:

Input:

num = "1234", t = 256

Output:

"1488"

Explanation:

The smallest zero-free number that is greater than 1234 and has the product of its digits divisible by 256 is 1488, with the product of its digits equal to 256.

Example 2:

Input:

num = "12355", t = 50

Output:

"12355"

Explanation:

12355 is already zero-free and has the product of its digits divisible by 50, with the product of its digits equal to 150.

Example 3:

Input:

num = "11111", t = 26

Output:

"-1"

Explanation:

No number greater than 11111 has the product of its digits divisible by 26.

Constraints:

2 <= num.length <= 2 * 10

5

num

consists only of digits in the range

['0', '9']

.

num

does not contain leading zeros.

1 <= t <= 10

14

Code Snippets

C++:

```
class Solution {
public:
string smallestNumber(string num, long long t) {

}
};
```

Java:

```
class Solution {
public String smallestNumber(String num, long t) {

}
}
```

Python3:

```
class Solution:
def smallestNumber(self, num: str, t: int) -> str:
```

Python:

```
class Solution(object):
def smallestNumber(self, num, t):
"""
:type num: str
```

```
:type t: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} num
* @param {number} t
* @return {string}
*/
var smallestNumber = function(num, t) {

};
```

TypeScript:

```
function smallestNumber(num: string, t: number): string {

};
```

C#:

```
public class Solution {
public string SmallestNumber(string num, long t) {

}
}
```

C:

```
char* smallestNumber(char* num, long long t) {

}
```

Go:

```
func smallestNumber(num string, t int64) string {

}
```

Kotlin:

```
class Solution {
fun smallestNumber(num: String, t: Long): String {

}
}
```

Swift:

```
class Solution {
func smallestNumber(_ num: String, _ t: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn smallest_number(num: String, t: i64) -> String {

}
}
```

Ruby:

```
# @param {String} num
# @param {Integer} t
# @return {String}
def smallest_number(num, t)

end
```

PHP:

```
class Solution {

/**
* @param String $num
* @param Integer $t
* @return String
*/
function smallestNumber($num, $t) {

}
}
```

Dart:

```
class Solution {
String smallestNumber(String num, int t) {

}
}
```

Scala:

```
object Solution {
def smallestNumber(num: String, t: Long): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec smallest_number(num :: String.t, t :: integer) :: String.t
def smallest_number(num, t) do

end
end
```

Erlang:

```
-spec smallest_number(Num :: unicode:unicode_binary(), T :: integer()) ->
unicode:unicode_binary().
smallest_number(Num, T) ->
.
```

Racket:

```
(define/contract (smallest-number num t)
(-> string? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Divisible Digit Product II
* Difficulty: Hard
* Tags: string, greedy, 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 smallestNumber(string num, long long t) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Divisible Digit Product II
* Difficulty: Hard
* Tags: string, greedy, 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 smallestNumber(String num, long t) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Divisible Digit Product II
```

```
Difficulty: Hard
Tags: string, greedy, 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 smallestNumber(self, num: str, t: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def smallestNumber(self, num, t):
"""
:type num: str
:type t: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Smallest Divisible Digit Product II
* Difficulty: Hard
* Tags: string, greedy, 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} t
* @return {string}
*/
var smallestNumber = function(num, t) {
};
```

TypeScript Solution:

```
/**
* Problem: Smallest Divisible Digit Product II
* Difficulty: Hard
* Tags: string, greedy, 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 smallestNumber(num: string, t: number): string {

};
```

C# Solution:

```
/*
* Problem: Smallest Divisible Digit Product II
* Difficulty: Hard
* Tags: string, greedy, 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 SmallestNumber(string num, long t) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Divisible Digit Product II
* Difficulty: Hard
* Tags: string, greedy, 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* smallestNumber(char* num, long long t) {

}
```

Go Solution:

```
// Problem: Smallest Divisible Digit Product II
// Difficulty: Hard
// Tags: string, greedy, 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 smallestNumber(num string, t int64) string {

}
```

Kotlin Solution:

```
class Solution {
fun smallestNumber(num: String, t: Long): String {

}
}
```

Swift Solution:

```
class Solution {
func smallestNumber(_ num: String, _ t: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Smallest Divisible Digit Product II
// Difficulty: Hard
// Tags: string, greedy, 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 smallest_number(num: String, t: i64) -> String {


}
}
```

Ruby Solution:

```
# @param {String} num
# @param {Integer} t
# @return {String}
def smallest_number(num, t)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $num
* @param Integer $t
* @return String
*/
function smallestNumber($num, $t) {

}
}
```

Dart Solution:

```
class Solution {
String smallestNumber(String num, int t) {

}
}
```

Scala Solution:

```
object Solution {
def smallestNumber(num: String, t: Long): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec smallest_number(num :: String.t, t :: integer) :: String.t
def smallest_number(num, t) do

end
end
```

Erlang Solution:

```
-spec smallest_number(Num :: unicode:unicode_binary(), T :: integer()) ->
unicode:unicode_binary().
smallest_number(Num, T) ->
.
```

Racket Solution:

```
(define/contract (smallest-number num t)
(-> string? exact-integer? string?)
)
```