Problem 1625: Lexicographically Smallest String After Applying Operations

Problem Information

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

Problem Description

You are given a string

s

of

even length

consisting of digits from

0

to

9

, and two integers

a

and

b

.

You can apply either of the following two operations any number of times and in any order on

s

:

Add

a

to all odd indices of

s

(0-indexed)

. Digits post

9

are cycled back to

0

. For example, if

s = "3456"

and

a = 5

,

s

becomes

"3951"

.

Rotate

s

to the right by

b

positions. For example, if

s = "3456"

and

b = 1

,

s

becomes

"6345"

.

Return

the

lexicographically smallest

string you can obtain by applying the above operations any number of times on

s

.

A string

a

is lexicographically smaller than a string

b

(of the same length) if in the first position where

a

and

b

differ, string

a

has a letter that appears earlier in the alphabet than the corresponding letter in

b

. For example,

"0158"

is lexicographically smaller than

"0190"

because the first position they differ is at the third letter, and

'5'

comes before

'9'

.

Example 1:

Input:

s = "5525", a = 9, b = 2

Output:

"2050"

Explanation:

We can apply the following operations: Start: "5525" Rotate: "2555" Add: "2454" Add: "2353" Rotate: "5323" Add: "5222" Add: "5121" Rotate: "2151" Add: "2050" There is no way to obtain a string that is lexicographically smaller than "2050".

Example 2:

Input:

s = "74", a = 5, b = 1

Output:

"24"

Explanation:

We can apply the following operations: Start: "74" Rotate: "47" Add: "42" Rotate: "24" There is no way to obtain a string that is lexicographically smaller than "24".

Example 3:

Input:

s = "0011", a = 4, b = 2

Output:

"0011"

Explanation:

There are no sequence of operations that will give us a lexicographically smaller string than "0011".

Constraints:

2 <= s.length <= 100

s.length

is even.

s

consists of digits from

0

to

9

only.

1 <= a <= 9

1 <= b <= s.length - 1

Code Snippets

C++:

```
class Solution {
public:
string findLexSmallestString(string s, int a, int b) {

}
};
```

Java:

```
class Solution {
public String findLexSmallestString(String s, int a, int b) {

}
}
```

Python3:

```
class Solution:
def findLexSmallestString(self, s: str, a: int, b: int) -> str:
```

Python:

```
class Solution(object):
def findLexSmallestString(self, s, a, b):
"""
:type s: str
:type a: int
:type b: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} a
* @param {number} b
* @return {string}
*/
```

```
var findLexSmallestString = function(s, a, b) {

};
```

TypeScript:

```
function findLexSmallestString(s: string, a: number, b: number): string {

};
```

C#:

```
public class Solution {
public string FindLexSmallestString(string s, int a, int b) {

}
}
```

C:

```
char* findLexSmallestString(char* s, int a, int b) {

}
```

Go:

```
func findLexSmallestString(s string, a int, b int) string {

}
```

Kotlin:

```
class Solution {
fun findLexSmallestString(s: String, a: Int, b: Int): String {

}
}
```

Swift:

```
class Solution {
func findLexSmallestString(_ s: String, _ a: Int, _ b: Int) -> String {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn find_lex_smallest_string(s: String, a: i32, b: i32) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} a
# @param {Integer} b
# @return {String}
def find_lex_smallest_string(s, a, b)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $a
* @param Integer $b
* @return String
*/
function findLexSmallestString($s, $a, $b) {

}
}
```

Dart:

```
class Solution {
String findLexSmallestString(String s, int a, int b) {
```

```
}
}
```

Scala:

```
object Solution {
def findLexSmallestString(s: String, a: Int, b: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_lex_smallest_string(s :: String.t, a :: integer, b :: integer) ::
String.t
def find_lex_smallest_string(s, a, b) do

end
end
```

Erlang:

```
-spec find_lex_smallest_string(S :: unicode:unicode_binary(), A :: integer(),
B :: integer()) -> unicode:unicode_binary().
find_lex_smallest_string(S, A, B) ->
.
```

Racket:

```
(define/contract (find-lex-smallest-string s a b)
(-> string? exact-integer? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lexicographically Smallest String After Applying Operations
```

```
* Difficulty: Medium
* Tags: string, graph, search
*
* 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 findLexSmallestString(string s, int a, int b) {

}
};
```

Java Solution:

```
/**
* Problem: Lexicographically Smallest String After Applying Operations
* Difficulty: Medium
* Tags: string, graph, search
*
* 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 findLexSmallestString(String s, int a, int b) {

}
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Smallest String After Applying Operations
Difficulty: Medium
Tags: string, graph, search

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 findLexSmallestString(self, s: str, a: int, b: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findLexSmallestString(self, s, a, b):
"""
:type s: str
:type a: int
:type b: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Lexicographically Smallest String After Applying Operations
* Difficulty: Medium
* Tags: string, graph, search
*
* 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} s
* @param {number} a
* @param {number} b
* @return {string}
*/
var findLexSmallestString = function(s, a, b) {

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographically Smallest String After Applying Operations
* Difficulty: Medium
* Tags: string, graph, search
*
* 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 findLexSmallestString(s: string, a: number, b: number): string {

};
```

C# Solution:

```
/*
* Problem: Lexicographically Smallest String After Applying Operations
* Difficulty: Medium
* Tags: string, graph, search
*
* 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 FindLexSmallestString(string s, int a, int b) {

}
}
```

C Solution:

```
/*
* Problem: Lexicographically Smallest String After Applying Operations
* Difficulty: Medium
* Tags: string, graph, search
*
* 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* findLexSmallestString(char* s, int a, int b) {

}
```

Go Solution:

```
// Problem: Lexicographically Smallest String After Applying Operations
// Difficulty: Medium
// Tags: string, graph, search
//
// 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 findLexSmallestString(s string, a int, b int) string {

}
```

Kotlin Solution:

```
class Solution {
fun findLexSmallestString(s: String, a: Int, b: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func findLexSmallestString(_ s: String, _ a: Int, _ b: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Lexicographically Smallest String After Applying Operations
// Difficulty: Medium
// Tags: string, graph, search
//
// 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 find_lex_smallest_string(s: String, a: i32, b: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} a
# @param {Integer} b
# @return {String}
def find_lex_smallest_string(s, a, b)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $a
* @param Integer $b
* @return String
*/
function findLexSmallestString($s, $a, $b) {

}
}
```

Dart Solution:

```
class Solution {
String findLexSmallestString(String s, int a, int b) {

}
}
```

Scala Solution:

```
object Solution {
def findLexSmallestString(s: String, a: Int, b: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_lex_smallest_string(s :: String.t, a :: integer, b :: integer) ::
String.t
def find_lex_smallest_string(s, a, b) do

end
end
```

Erlang Solution:

```
-spec find_lex_smallest_string(S :: unicode:unicode_binary(), A :: integer(),
B :: integer()) -> unicode:unicode_binary().
find_lex_smallest_string(S, A, B) ->
.
```

Racket Solution:

```
(define/contract (find-lex-smallest-string s a b)
(-> string? exact-integer? exact-integer? string?)
)
```