Problem 3646: Next Special Palindrome Number

Problem Information

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

Problem Description

You are given an integer

n

.

A number is called

special

if:

It is a

palindrome

.

Every digit

k

in the number appears

exactly

k

times.

Return the

smallest

special number

strictly

greater than

n

.

Example 1:

Input:

n = 2

Output:

22

Explanation:

22 is the smallest special number greater than 2, as it is a palindrome and the digit 2 appears exactly 2 times.

Example 2:

Input:

n = 33

Output:

212

Explanation:

212 is the smallest special number greater than 33, as it is a palindrome and the digits 1 and 2 appear exactly 1 and 2 times respectively.

Constraints:

0 <= n <= 10

15

Code Snippets

C++:

```
class Solution {
public:
long long specialPalindrome(long long n) {

}
};
```

Java:

```
class Solution {
public long specialPalindrome(long n) {

}
}
```

Python3:

```
class Solution:
def specialPalindrome(self, n: int) -> int:
```

Python:

```
class Solution(object):
def specialPalindrome(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var specialPalindrome = function(n) {

};
```

TypeScript:

```
function specialPalindrome(n: number): number {

};
```

C#:

```
public class Solution {
public long SpecialPalindrome(long n) {

}
}
```

C:

```
long long specialPalindrome(long long n) {

}
```

Go:

```
func specialPalindrome(n int64) int64 {

}
```

Kotlin:

```
class Solution {
fun specialPalindrome(n: Long): Long {

}
}
```

Swift:

```
class Solution {
func specialPalindrome(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn special_palindrome(n: i64) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def special_palindrome(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function specialPalindrome($n) {

}

}
```

Dart:

```
class Solution {
int specialPalindrome(int n) {

}
}
```

Scala:

```
object Solution {
def specialPalindrome(n: Long): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec special_palindrome(n :: integer) :: integer
def special_palindrome(n) do

end
end
```

Erlang:

```
-spec special_palindrome(N :: integer()) -> integer().
special_palindrome(N) ->
.
```

Racket:

```
(define/contract (special-palindrome n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Next Special Palindrome Number
* Difficulty: Hard
* Tags: string
*
* 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:
long long specialPalindrome(long long n) {

}
};
```

Java Solution:

```
/**
* Problem: Next Special Palindrome Number
* Difficulty: Hard
* Tags: string
*
* 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 long specialPalindrome(long n) {

}
}
```

Python3 Solution:

```
"""
Problem: Next Special Palindrome Number
Difficulty: Hard
Tags: string
```

```
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 specialPalindrome(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def specialPalindrome(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Next Special Palindrome Number
* Difficulty: Hard
* Tags: string
*
* 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} n
* @return {number}
*/
var specialPalindrome = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Next Special Palindrome Number
* Difficulty: Hard
* Tags: string
*
* 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 specialPalindrome(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Next Special Palindrome Number
* Difficulty: Hard
* Tags: string
*
* 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 long SpecialPalindrome(long n) {

}
}
```

C Solution:

```
/*
* Problem: Next Special Palindrome Number
* Difficulty: Hard
* Tags: string
*
* 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
```

```
*/

long long specialPalindrome(long long n) {

}
```

Go Solution:

```
// Problem: Next Special Palindrome Number
// Difficulty: Hard
// Tags: string
//
// 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 specialPalindrome(n int64) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun specialPalindrome(n: Long): Long {

}
}
```

Swift Solution:

```
class Solution {
func specialPalindrome(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Next Special Palindrome Number
// Difficulty: Hard
// Tags: string
```

```
//
// 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 special_palindrome(n: i64) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def special_palindrome(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function specialPalindrome($n) {

}
}
```

Dart Solution:

```
class Solution {
int specialPalindrome(int n) {

}
}
```

Scala Solution:

```
object Solution {
def specialPalindrome(n: Long): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec special_palindrome(n :: integer) :: integer
def special_palindrome(n) do

end
end
```

Erlang Solution:

```
-spec special_palindrome(N :: integer()) -> integer().
special_palindrome(N) ->
.
```

Racket Solution:

```
(define/contract (special-palindrome n)
(-> exact-integer? exact-integer?)
)
```