Problem 564: Find the Closest Palindrome

Problem Information

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

Problem Description

Given a string

n

representing an integer, return

the closest integer (not including itself), which is a palindrome

. If there is a tie, return

the smaller one

.

The closest is defined as the absolute difference minimized between two integers.

Example 1:

Input:

n = "123"

Output:

"121"

Example 2:

Input:

n = "1"

Output:

"0"

Explanation:

0 and 2 are the closest palindromes but we return the smallest which is 0.

Constraints:

1 <= n.length <= 18

n

consists of only digits.

n

does not have leading zeros.

n

is representing an integer in the range

[1, 10

18

- 1]

.

Code Snippets

C++:

```
class Solution {
public:
string nearestPalindromic(string n) {

}
};
```

Java:

```
class Solution {
public String nearestPalindromic(String n) {

}
}
```

Python3:

```
class Solution:
def nearestPalindromic(self, n: str) -> str:
```

Python:

```
class Solution(object):
def nearestPalindromic(self, n):
"""
:type n: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} n
* @return {string}
*/
var nearestPalindromic = function(n) {

};
```

TypeScript:

```
function nearestPalindromic(n: string): string {

};
```

C#:

```
public class Solution {
public string NearestPalindromic(string n) {

}
}
```

C:

```
char* nearestPalindromic(char* n) {

}
```

Go:

```
func nearestPalindromic(n string) string {

}
```

Kotlin:

```
class Solution {
fun nearestPalindromic(n: String): String {

}
}
```

Swift:

```
class Solution {
func nearestPalindromic(_ n: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn nearest_palindromic(n: String) -> String {

}
}
```

Ruby:

```
# @param {String} n
# @return {String}
def nearest_palindromic(n)

end
```

PHP:

```
class Solution {

/**
* @param String $n
* @return String
*/
function nearestPalindromic($n) {

}
}
```

Dart:

```
class Solution {
String nearestPalindromic(String n) {

}
}
```

Scala:

```
object Solution {
def nearestPalindromic(n: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec nearest_palindromic(n :: String.t) :: String.t
def nearest_palindromic(n) do

end
end
```

Erlang:

```
-spec nearest_palindromic(N :: unicode:unicode_binary()) ->
unicode:unicode_binary().
nearest_palindromic(N) ->
.
```

Racket:

```
(define/contract (nearest-palindromic n)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Closest Palindrome
* 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 nearestPalindromic(string n) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Closest Palindrome
* 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 nearestPalindromic(String n) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Closest Palindrome
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 nearestPalindromic(self, n: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def nearestPalindromic(self, n):
"""
:type n: str
:rtype: str
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Closest Palindrome
* 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 {string} n
* @return {string}
*/
var nearestPalindromic = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Closest Palindrome
* 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 nearestPalindromic(n: string): string {

};
```

C# Solution:

```
/*
* Problem: Find the Closest Palindrome
* 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 NearestPalindromic(string n) {

}
}
```

C Solution:

```
/*
* Problem: Find the Closest Palindrome
* 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* nearestPalindromic(char* n) {

}
```

Go Solution:

```
// Problem: Find the Closest Palindrome
// 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 nearestPalindromic(n string) string {

}
```

Kotlin Solution:

```
class Solution {
fun nearestPalindromic(n: String): String {

}
}
```

Swift Solution:

```
class Solution {
func nearestPalindromic(_ n: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Find the Closest Palindrome
// 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 nearest_palindromic(n: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} n
# @return {String}
def nearest_palindromic(n)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $n
* @return String
*/
function nearestPalindromic($n) {

}
}
```

Dart Solution:

```
class Solution {
String nearestPalindromic(String n) {

}
}
```

Scala Solution:

```
object Solution {
def nearestPalindromic(n: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec nearest_palindromic(n :: String.t) :: String.t
def nearest_palindromic(n) do

end
end
```

Erlang Solution:

```
-spec nearest_palindromic(N :: unicode:unicode_binary()) ->
unicode:unicode_binary().
nearest_palindromic(N) ->
.
```

Racket Solution:

```
(define/contract (nearest-palindromic n)
(-> string? string?)
)
```