Problem 3260: Find the Largest Palindrome Divisible by K

Problem Information

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

Problem Description

You are given two

positive

integers

n

and

k

.

An integer

x

is called

k-palindromic

if:

x

is a

palindrome

.

x

is divisible by

k

.

Return the

largest

integer having

n

digits (as a string) that is

k-palindromic

.

Note

that the integer must

not

have leading zeros.

Example 1:

Input:

n = 3, k = 5

Output:

"595"

Explanation:

595 is the largest k-palindromic integer with 3 digits.

Example 2:

Input:

n = 1, k = 4

Output:

"8"

Explanation:

4 and 8 are the only k-palindromic integers with 1 digit.

Example 3:

Input:

n = 5, k = 6

Output:

"89898"

Constraints:

1 <= n <= 10

5

1 <= k <= 9

Code Snippets

C++:

```
class Solution {
public:
string largestPalindrome(int n, int k) {

}
};
```

Java:

```
class Solution {
public String largestPalindrome(int n, int k) {

}
}
```

Python3:

```
class Solution:
def largestPalindrome(self, n: int, k: int) -> str:
```

Python:

```
class Solution(object):
def largestPalindrome(self, n, k):
"""
:type n: int
:type k: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {string}
*/
var largestPalindrome = function(n, k) {

};
```

TypeScript:

```
function largestPalindrome(n: number, k: number): string {

};
```

C#:

```
public class Solution {
public string LargestPalindrome(int n, int k) {

}
}
```

C:

```
char* largestPalindrome(int n, int k) {

}
```

Go:

```
func largestPalindrome(n int, k int) string {

}
```

Kotlin:

```
class Solution {
fun largestPalindrome(n: Int, k: Int): String {

}
}
```

Swift:

```
class Solution {
func largestPalindrome(_ n: Int, _ k: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn largest_palindrome(n: i32, k: i32) -> String {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {String}
def largest_palindrome(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return String
*/
function largestPalindrome($n, $k) {

}
}
```

Dart:

```
class Solution {
String largestPalindrome(int n, int k) {
```

```
}
}
```

Scala:

```
object Solution {
def largestPalindrome(n: Int, k: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec largest_palindrome(n :: integer, k :: integer) :: String.t
def largest_palindrome(n, k) do

end
end
```

Erlang:

```
-spec largest_palindrome(N :: integer(), K :: integer()) ->
unicode:unicode_binary().
largest_palindrome(N, K) ->
.
```

Racket:

```
(define/contract (largest-palindrome n k)
(-> exact-integer? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Largest Palindrome Divisible by K
```

```
* Difficulty: Hard
* Tags: string, dp, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
string largestPalindrome(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Largest Palindrome Divisible by K
* Difficulty: Hard
* Tags: string, dp, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public String largestPalindrome(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Largest Palindrome Divisible by K
Difficulty: Hard
Tags: string, dp, greedy, math

Approach: String manipulation with hash map or two pointers
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def largestPalindrome(self, n: int, k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def largestPalindrome(self, n, k):
"""
:type n: int
:type k: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Largest Palindrome Divisible by K
* Difficulty: Hard
* Tags: string, dp, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @param {number} k
* @return {string}
*/
var largestPalindrome = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Largest Palindrome Divisible by K
* Difficulty: Hard
* Tags: string, dp, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function largestPalindrome(n: number, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Find the Largest Palindrome Divisible by K
* Difficulty: Hard
* Tags: string, dp, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public string LargestPalindrome(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the Largest Palindrome Divisible by K
* Difficulty: Hard
* Tags: string, dp, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

char* largestPalindrome(int n, int k) {

}
```

Go Solution:

```
// Problem: Find the Largest Palindrome Divisible by K
// Difficulty: Hard
// Tags: string, dp, greedy, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func largestPalindrome(n int, k int) string {

}
```

Kotlin Solution:

```
class Solution {
fun largestPalindrome(n: Int, k: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func largestPalindrome(_ n: Int, _ k: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Find the Largest Palindrome Divisible by K
// Difficulty: Hard
// Tags: string, dp, greedy, math
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn largest_palindrome(n: i32, k: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {String}
def largest_palindrome(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return String
*/
function largestPalindrome($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
String largestPalindrome(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def largestPalindrome(n: Int, k: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec largest_palindrome(n :: integer, k :: integer) :: String.t
def largest_palindrome(n, k) do

end
end
```

Erlang Solution:

```
-spec largest_palindrome(N :: integer(), K :: integer()) ->
unicode:unicode_binary().
largest_palindrome(N, K) ->
.
```

Racket Solution:

```
(define/contract (largest-palindrome n k)
(-> exact-integer? exact-integer? string?)
)
```