Problem 2081: Sum of k-Mirror Numbers

Problem Information

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

Problem Description

A

k-mirror number

is a

positive

integer

without leading zeros

that reads the same both forward and backward in base-10

as well as

in base-k.

For example,

9

is a 2-mirror number. The representation of

9

in base-10 and base-2 are

9

and

1001

respectively, which read the same both forward and backward.

On the contrary,

4

is not a 2-mirror number. The representation of

4

in base-2 is

100

, which does not read the same both forward and backward.

Given the base

k

and the number

n

, return

the

sum

of the

n

smallest

k-mirror numbers

.

Example 1:

Input:

k = 2, n = 5

Output:

25

Explanation:

The 5 smallest 2-mirror numbers and their representations in base-2 are listed as follows: base-10 base-2 1 1 3 11 5 101 7 111 9 1001 Their sum = 1 + 3 + 5 + 7 + 9 = 25.

Example 2:

Input:

k = 3, n = 7

Output:

499

Explanation:

The 7 smallest 3-mirror numbers are and their representations in base-3 are listed as follows: base-10 base-3 1 1 2 2 4 11 8 22 121 11111 151 12121 212 21212 Their sum = 1 + 2 + 4 + 8 + 121 + 151 + 212 = 499.

Example 3:

Input:

k = 7, n = 17

Output:

20379000

Explanation:

The 17 smallest 7-mirror numbers are: 1, 2, 3, 4, 5, 6, 8, 121, 171, 242, 292, 16561, 65656, 2137312, 4602064, 6597956, 6958596

Constraints:

2 <= k <= 9

1 <= n <= 30

Code Snippets

C++:

```
class Solution {
public:
long long kMirror(int k, int n) {

}
};
```

Java:

```
class Solution {
public long kMirror(int k, int n) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function kMirror(k: number, n: number): number {

};
```

C#:

```
public class Solution {
public long KMirror(int k, int n) {

}
}
```

C:

```
long long kMirror(int k, int n) {

}
```

Go:

```
func kMirror(k int, n int) int64 {

}
```

Kotlin:

```
class Solution {
fun kMirror(k: Int, n: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn k_mirror(k: i32, n: i32) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int kMirror(int k, int n) {

}
}
```

Scala:

```
object Solution {
def kMirror(k: Int, n: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec k_mirror(k :: integer, n :: integer) :: integer
def k_mirror(k, n) do

end
end
```

Erlang:

```
-spec k_mirror(K :: integer(), N :: integer()) -> integer().
k_mirror(K, N) ->
```

```
.
```

Racket:

```
(define/contract (k-mirror k n)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of k-Mirror Numbers
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
long long kMirror(int k, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of k-Mirror Numbers
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public long kMirror(int k, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of k-Mirror Numbers
Difficulty: Hard
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of k-Mirror Numbers
* Difficulty: Hard
* Tags: math
*
```

```
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Sum of k-Mirror Numbers
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function kMirror(k: number, n: number): number {

};
```

C# Solution:

```
/*
* Problem: Sum of k-Mirror Numbers
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
public class Solution {
public long KMirror(int k, int n) {

}
}
```

C Solution:

```
/*
* Problem: Sum of k-Mirror Numbers
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

long long kMirror(int k, int n) {

}
```

Go Solution:

```
// Problem: Sum of k-Mirror Numbers
// Difficulty: Hard
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func kMirror(k int, n int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun kMirror(k: Int, n: Int): Long {
```

```
}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of k-Mirror Numbers
// Difficulty: Hard
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn k_mirror(k: i32, n: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $k
* @param Integer $n
* @return Integer
*/
function kMirror($k, $n) {

}
}
```

Dart Solution:

```
class Solution {
int kMirror(int k, int n) {

}
}
```

Scala Solution:

```
object Solution {
def kMirror(k: Int, n: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec k_mirror(k :: integer, n :: integer) :: integer
def k_mirror(k, n) do

end
end
```

Erlang Solution:

```
-spec k_mirror(K :: integer(), N :: integer()) -> integer().
k_mirror(K, N) ->
.
```

Racket Solution:

```
(define/contract (k-mirror k n)
(-> exact-integer? exact-integer? exact-integer?)
)
```