Problem 2310: Sum of Numbers With Units Digit K

Problem Information

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

Problem Description

Given two integers

num

and

k

, consider a set of positive integers with the following properties:

The units digit of each integer is

k

.

The sum of the integers is

num

.

Return

the

minimum

possible size of such a set, or

-1

if no such set exists.

Note:

The set can contain multiple instances of the same integer, and the sum of an empty set is considered

0

.

The

units digit

of a number is the rightmost digit of the number.

Example 1:

Input:

num = 58, k = 9

Output:

2

Explanation:

One valid set is [9,49], as the sum is 58 and each integer has a units digit of 9. Another valid set is [19,39]. It can be shown that 2 is the minimum possible size of a valid set.

Example 2:

Input:

num = 37, k = 2

Output:

-1

Explanation:

It is not possible to obtain a sum of 37 using only integers that have a units digit of 2.

Example 3:

Input:

num = 0, k = 7

Output:

0

Explanation:

The sum of an empty set is considered 0.

Constraints:

0 <= num <= 3000

0 <= k <= 9

Code Snippets

C++:

```
class Solution {
public:
```

```
int minimumNumbers(int num, int k) {

}
};
```

Java:

```
class Solution {
public int minimumNumbers(int num, int k) {

}
}
```

Python3:

```
class Solution:
def minimumNumbers(self, num: int, k: int) -> int:
```

Python:

```
class Solution(object):
def minimumNumbers(self, num, k):
"""
:type num: int
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minimumNumbers(num: number, k: number): number {

};
```

C#:

```
public class Solution {
public int MinimumNumbers(int num, int k) {

}
}
```

C:

```
int minimumNumbers(int num, int k) {

}
```

Go:

```
func minimumNumbers(num int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minimumNumbers(num: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimumNumbers(_ num: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_numbers(num: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num
# @param {Integer} k
# @return {Integer}
def minimum_numbers(num, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $num
* @param Integer $k
* @return Integer
*/
function minimumNumbers($num, $k) {

}
}
```

Dart:

```
class Solution {
int minimumNumbers(int num, int k) {

}
}
```

Scala:

```
object Solution {
def minimumNumbers(num: Int, k: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec minimum_numbers(num :: integer, k :: integer) :: integer
def minimum_numbers(num, k) do

end
end
```

Erlang:

```
-spec minimum_numbers(Num :: integer(), K :: integer()) -> integer().
minimum_numbers(Num, K) ->
.
```

Racket:

```
(define/contract (minimum-numbers num k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Numbers With Units Digit K
* Difficulty: Medium
* Tags: dp, greedy, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minimumNumbers(int num, int k) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Sum of Numbers With Units Digit K
* Difficulty: Medium
* Tags: dp, greedy, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimumNumbers(int num, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Numbers With Units Digit K
Difficulty: Medium
Tags: dp, greedy, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

```
class Solution(object):
def minimumNumbers(self, num, k):
"""
:type num: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Sum of Numbers With Units Digit K
* Difficulty: Medium
* Tags: dp, greedy, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} num
* @param {number} k
* @return {number}
*/
var minimumNumbers = function(num, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Numbers With Units Digit K
* Difficulty: Medium
* Tags: dp, greedy, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumNumbers(num: number, k: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Sum of Numbers With Units Digit K
* Difficulty: Medium
* Tags: dp, greedy, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinimumNumbers(int num, int k) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Numbers With Units Digit K
* Difficulty: Medium
* Tags: dp, greedy, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumNumbers(int num, int k) {

}
```

Go Solution:

```
// Problem: Sum of Numbers With Units Digit K
// Difficulty: Medium
```

```
// Tags: dp, greedy, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func minimumNumbers(num int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumNumbers(num: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumNumbers(_ num: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sum of Numbers With Units Digit K
// Difficulty: Medium
// Tags: dp, greedy, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_numbers(num: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num
# @param {Integer} k
# @return {Integer}
def minimum_numbers(num, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $num
* @param Integer $k
* @return Integer
*/
function minimumNumbers($num, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minimumNumbers(int num, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minimumNumbers(num: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_numbers(num :: integer, k :: integer) :: integer
def minimum_numbers(num, k) do

end
end
```

Erlang Solution:

```
-spec minimum_numbers(Num :: integer(), K :: integer()) -> integer().
minimum_numbers(Num, K) ->
.
```

Racket Solution:

```
(define/contract (minimum-numbers num k)
(-> exact-integer? exact-integer? exact-integer?)
)
```