Problem 1999: Smallest Greater Multiple Made of Two Digits

Problem Information

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

Problem Description

Given three integers,

k

,

digit1

, and

digit2

, you want to find the

smallest

integer that is:

Larger

than

k

,

A

multiple

of

k

, and

Comprised of

only

the digits

digit1

and/or

digit2

.

Return

the

smallest

such integer. If no such integer exists or the integer exceeds the limit of a signed 32-bit integer (

2

31

- 1

), return

-1

.

Example 1:

Input:

k = 2, digit1 = 0, digit2 = 2

Output:

20

Explanation:

20 is the first integer larger than 2, a multiple of 2, and comprised of only the digits 0 and/or 2.

Example 2:

Input:

k = 3, digit1 = 4, digit2 = 2

Output:

24

Explanation:

24 is the first integer larger than 3, a multiple of 3, and comprised of only the digits 4 and/or 2.

Example 3:

Input:

k = 2, digit1 = 0, digit2 = 0

Output:

-1

Explanation:

No integer meets the requirements so return -1.

Constraints:

1 <= k <= 1000

0 <= digit1 <= 9

0 <= digit2 <= 9

Code Snippets

C++:

```
class Solution {
public:
int findInteger(int k, int digit1, int digit2) {

}
};
```

Java:

```
class Solution {
public int findInteger(int k, int digit1, int digit2) {

}
}
```

Python3:

```
class Solution:
def findInteger(self, k: int, digit1: int, digit2: int) -> int:
```

Python:

```
class Solution(object):
def findInteger(self, k, digit1, digit2):
"""
:type k: int
:type digit1: int
:type digit2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} k
* @param {number} digit1
* @param {number} digit2
* @return {number}
*/
var findInteger = function(k, digit1, digit2) {

};
```

TypeScript:

```
function findInteger(k: number, digit1: number, digit2: number): number {

};
```

C#:

```
public class Solution {
public int FindInteger(int k, int digit1, int digit2) {

}
}
```

C:

```
int findInteger(int k, int digit1, int digit2) {

}
```

Go:

```
func findInteger(k int, digit1 int, digit2 int) int {

}
```

Kotlin:

```
class Solution {
fun findInteger(k: Int, digit1: Int, digit2: Int): Int {

}
}
```

Swift:

```
class Solution {
func findInteger(_ k: Int, _ digit1: Int, _ digit2: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_integer(k: i32, digit1: i32, digit2: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} k
# @param {Integer} digit1
# @param {Integer} digit2
# @return {Integer}
def find_integer(k, digit1, digit2)

end
```

PHP:

```
class Solution {

/**
* @param Integer $k
* @param Integer $digit1
* @param Integer $digit2
* @return Integer
*/
function findInteger($k, $digit1, $digit2) {

}
}
```

Dart:

```
class Solution {
int findInteger(int k, int digit1, int digit2) {

}
}
```

Scala:

```
object Solution {
def findInteger(k: Int, digit1: Int, digit2: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_integer(k :: integer, digit1 :: integer, digit2 :: integer) ::
integer
def find_integer(k, digit1, digit2) do

end
end
```

Erlang:

```
-spec find_integer(K :: integer(), Digit1 :: integer(), Digit2 :: integer())
-> integer().
find_integer(K, Digit1, Digit2) ->
.
```

Racket:

```
(define/contract (find-integer k digit1 digit2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Greater Multiple Made of Two Digits
* Difficulty: Medium
* 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:
int findInteger(int k, int digit1, int digit2) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Greater Multiple Made of Two Digits
* Difficulty: Medium
* 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 int findInteger(int k, int digit1, int digit2) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Greater Multiple Made of Two Digits
Difficulty: Medium
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 findInteger(self, k: int, digit1: int, digit2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findInteger(self, k, digit1, digit2):
"""
:type k: int
:type digit1: int
:type digit2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Smallest Greater Multiple Made of Two Digits
```

```
* Difficulty: Medium
* 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} digit1
* @param {number} digit2
* @return {number}
*/
var findInteger = function(k, digit1, digit2) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Greater Multiple Made of Two Digits
* Difficulty: Medium
* 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 findInteger(k: number, digit1: number, digit2: number): number {

};
```

C# Solution:

```
/*
* Problem: Smallest Greater Multiple Made of Two Digits
* Difficulty: Medium
* 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 int FindInteger(int k, int digit1, int digit2) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Greater Multiple Made of Two Digits
* Difficulty: Medium
* 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
*/

int findInteger(int k, int digit1, int digit2) {

}
```

Go Solution:

```
// Problem: Smallest Greater Multiple Made of Two Digits
// Difficulty: Medium
// 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 findInteger(k int, digit1 int, digit2 int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findInteger(k: Int, digit1: Int, digit2: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findInteger(_ k: Int, _ digit1: Int, _ digit2: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Smallest Greater Multiple Made of Two Digits
// Difficulty: Medium
// 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 find_integer(k: i32, digit1: i32, digit2: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} k
# @param {Integer} digit1
# @param {Integer} digit2
# @return {Integer}
def find_integer(k, digit1, digit2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $k
* @param Integer $digit1
* @param Integer $digit2
* @return Integer
*/
function findInteger($k, $digit1, $digit2) {

}
}
```

Dart Solution:

```
class Solution {
int findInteger(int k, int digit1, int digit2) {

}
}
```

Scala Solution:

```
object Solution {
def findInteger(k: Int, digit1: Int, digit2: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_integer(k :: integer, digit1 :: integer, digit2 :: integer) ::
integer
def find_integer(k, digit1, digit2) do

end
end
```

Erlang Solution:

```
-spec find_integer(K :: integer(), Digit1 :: integer(), Digit2 :: integer())
-> integer().
find_integer(K, Digit1, Digit2) ->
.
```

Racket Solution:

```
(define/contract (find-integer k digit1 digit2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```