Problem 728: Self Dividing Numbers

Problem Information

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

Problem Description

A

self-dividing number

is a number that is divisible by every digit it contains.

For example,

128

is

a self-dividing number

because

128 % 1 == 0

,

128 % 2 == 0

, and

128 % 8 == 0

.

A

self-dividing number

is not allowed to contain the digit zero.

Given two integers

left

and

right

, return

a list of all the

self-dividing numbers

in the range

[left, right]

(both

inclusive

).

Example 1:

Input:

left = 1, right = 22

Output:

[1,2,3,4,5,6,7,8,9,11,12,15,22]

Example 2:

Input:

left = 47, right = 85

Output:

[48,55,66,77]

Constraints:

1 <= left <= right <= 10

4

Code Snippets

C++:

```
class Solution {
public:
vector<int> selfDividingNumbers(int left, int right) {

}
};
```

Java:

```
class Solution {
public List<Integer> selfDividingNumbers(int left, int right) {

}
}
```

Python3:

```
class Solution:
def selfDividingNumbers(self, left: int, right: int) -> List[int]:
```

Python:

```
class Solution(object):
def selfDividingNumbers(self, left, right):
"""
:type left: int
:type right: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} left
* @param {number} right
* @return {number[]}
*/
var selfDividingNumbers = function(left, right) {

};
```

TypeScript:

```
function selfDividingNumbers(left: number, right: number): number[] {

};
```

C#:

```
public class Solution {
public IList<int> SelfDividingNumbers(int left, int right) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* selfDividingNumbers(int left, int right, int* returnSize) {

}
```

Go:

```
func selfDividingNumbers(left int, right int) []int {

}
```

Kotlin:

```
class Solution {
fun selfDividingNumbers(left: Int, right: Int): List<Int> {

}
}
```

Swift:

```
class Solution {
func selfDividingNumbers(_ left: Int, _ right: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn self_dividing_numbers(left: i32, right: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} left
# @param {Integer} right
# @return {Integer[]}
def self_dividing_numbers(left, right)

end
```

PHP:

```
class Solution {

/**
* @param Integer $left
* @param Integer $right
* @return Integer[]
*/
function selfDividingNumbers($left, $right) {

}
}
```

Dart:

```
class Solution {
List<int> selfDividingNumbers(int left, int right) {

}
}
```

Scala:

```
object Solution {
def selfDividingNumbers(left: Int, right: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec self_dividing_numbers(left :: integer, right :: integer) :: [integer]
def self_dividing_numbers(left, right) do

end
end
```

Erlang:

```
-spec self_dividing_numbers(Left :: integer(), Right :: integer()) ->
[integer()].
```

```
self_dividing_numbers(Left, Right) ->
.
```

Racket:

```
(define/contract (self-dividing-numbers left right)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Self Dividing Numbers
* Difficulty: Easy
* 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:
vector<int> selfDividingNumbers(int left, int right) {

}
};
```

Java Solution:

```
/**
* Problem: Self Dividing Numbers
* Difficulty: Easy
* 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 List<Integer> selfDividingNumbers(int left, int right) {

}
}
```

Python3 Solution:

```
"""
Problem: Self Dividing Numbers
Difficulty: Easy
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 selfDividingNumbers(self, left: int, right: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def selfDividingNumbers(self, left, right):
"""
:type left: int
:type right: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Self Dividing Numbers
* Difficulty: Easy
* 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} left
* @param {number} right
* @return {number[]}
*/
var selfDividingNumbers = function(left, right) {

};
```

TypeScript Solution:

```
/**
* Problem: Self Dividing Numbers
* Difficulty: Easy
* 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 selfDividingNumbers(left: number, right: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Self Dividing Numbers
* Difficulty: Easy
* 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 IList<int> SelfDividingNumbers(int left, int right) {

}
}
```

C Solution:

```
/*
* Problem: Self Dividing Numbers
* Difficulty: Easy
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* selfDividingNumbers(int left, int right, int* returnSize) {

}
```

Go Solution:

```
// Problem: Self Dividing Numbers
// Difficulty: Easy
// 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 selfDividingNumbers(left int, right int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun selfDividingNumbers(left: Int, right: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func selfDividingNumbers(_ left: Int, _ right: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Self Dividing Numbers
// Difficulty: Easy
// 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 self_dividing_numbers(left: i32, right: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} left
# @param {Integer} right
# @return {Integer[]}
def self_dividing_numbers(left, right)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $left
* @param Integer $right
* @return Integer[]
*/
function selfDividingNumbers($left, $right) {

}
}
```

Dart Solution:

```
class Solution {
List<int> selfDividingNumbers(int left, int right) {

}
}
```

Scala Solution:

```
object Solution {
def selfDividingNumbers(left: Int, right: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec self_dividing_numbers(left :: integer, right :: integer) :: [integer]
def self_dividing_numbers(left, right) do

end
end
```

Erlang Solution:

```
-spec self_dividing_numbers(Left :: integer(), Right :: integer()) ->
[integer()].
self_dividing_numbers(Left, Right) ->
```

```
.
```

Racket Solution:

```
(define/contract (self-dividing-numbers left right)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```