Problem 878: Nth Magical Number

Problem Information

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

Problem Description

A positive integer is

magical

if it is divisible by either

a

or

b

.

Given the three integers

n

,

a

, and

b

, return the

n

th

magical number. Since the answer may be very large,

return it modulo

10

9

+ 7

.

Example 1:

Input:

n = 1, a = 2, b = 3

Output:

2

Example 2:

Input:

n = 4, a = 2, b = 3

Output:

6

Constraints:

1 <= n <= 10

9

2 <= a, b <= 4 * 10

4

Code Snippets

C++:

```
class Solution {
public:
int nthMagicalNumber(int n, int a, int b) {

}
};
```

Java:

```
class Solution {
public int nthMagicalNumber(int n, int a, int b) {

}
}
```

Python3:

```
class Solution:
def nthMagicalNumber(self, n: int, a: int, b: int) -> int:
```

Python:

```
class Solution(object):
def nthMagicalNumber(self, n, a, b):
"""
:type n: int
:type a: int
:type b: int
```

```
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} a
* @param {number} b
* @return {number}
*/
var nthMagicalNumber = function(n, a, b) {

};
```

TypeScript:

```
function nthMagicalNumber(n: number, a: number, b: number): number {

};
```

C#:

```
public class Solution {
public int NthMagicalNumber(int n, int a, int b) {

}
}
```

C:

```
int nthMagicalNumber(int n, int a, int b) {

}
```

Go:

```
func nthMagicalNumber(n int, a int, b int) int {

}
```

Kotlin:

```
class Solution {
fun nthMagicalNumber(n: Int, a: Int, b: Int): Int {

}
}
```

Swift:

```
class Solution {
func nthMagicalNumber(_ n: Int, _ a: Int, _ b: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn nth_magical_number(n: i32, a: i32, b: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} a
# @param {Integer} b
# @return {Integer}
def nth_magical_number(n, a, b)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $a
* @param Integer $b
* @return Integer
*/
function nthMagicalNumber($n, $a, $b) {
```

```
}
}
```

Dart:

```
class Solution {
int nthMagicalNumber(int n, int a, int b) {

}
}
```

Scala:

```
object Solution {
def nthMagicalNumber(n: Int, a: Int, b: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec nth_magical_number(n :: integer, a :: integer, b :: integer) :: integer
def nth_magical_number(n, a, b) do

end
end
```

Erlang:

```
-spec nth_magical_number(N :: integer(), A :: integer(), B :: integer()) ->
integer().
nth_magical_number(N, A, B) ->
.
```

Racket:

```
(define/contract (nth-magical-number n a b)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Nth Magical Number
* Difficulty: Hard
* Tags: math, search
*
* 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 nthMagicalNumber(int n, int a, int b) {

}
};
```

Java Solution:

```
/**
* Problem: Nth Magical Number
* Difficulty: Hard
* Tags: math, search
*
* 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 nthMagicalNumber(int n, int a, int b) {

}
}
```

Python3 Solution:

```
"""
Problem: Nth Magical Number
Difficulty: Hard
Tags: math, search

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 nthMagicalNumber(self, n: int, a: int, b: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def nthMagicalNumber(self, n, a, b):
"""
:type n: int
:type a: int
:type b: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Nth Magical Number
* Difficulty: Hard
* Tags: math, search
*
* 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} n
* @param {number} a
* @param {number} b
```

```
* @return {number}
*/
var nthMagicalNumber = function(n, a, b) {

};
```

TypeScript Solution:

```
/**
* Problem: Nth Magical Number
* Difficulty: Hard
* Tags: math, search
*
* 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 nthMagicalNumber(n: number, a: number, b: number): number {

};
```

C# Solution:

```
/*
* Problem: Nth Magical Number
* Difficulty: Hard
* Tags: math, search
*
* 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 NthMagicalNumber(int n, int a, int b) {

}
}
```

C Solution:

```
/*
* Problem: Nth Magical Number
* Difficulty: Hard
* Tags: math, search
*
* 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 nthMagicalNumber(int n, int a, int b) {

}
```

Go Solution:

```
// Problem: Nth Magical Number
// Difficulty: Hard
// Tags: math, search
//
// 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 nthMagicalNumber(n int, a int, b int) int {

}
```

Kotlin Solution:

```
class Solution {
fun nthMagicalNumber(n: Int, a: Int, b: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func nthMagicalNumber(_ n: Int, _ a: Int, _ b: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Nth Magical Number
// Difficulty: Hard
// Tags: math, search
//
// 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 nth_magical_number(n: i32, a: i32, b: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} a
# @param {Integer} b
# @return {Integer}
def nth_magical_number(n, a, b)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $a
* @param Integer $b
* @return Integer
*/
function nthMagicalNumber($n, $a, $b) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int nthMagicalNumber(int n, int a, int b) {

}
}
```

Scala Solution:

```
object Solution {
def nthMagicalNumber(n: Int, a: Int, b: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec nth_magical_number(n :: integer, a :: integer, b :: integer) :: integer
def nth_magical_number(n, a, b) do

end
end
```

Erlang Solution:

```
-spec nth_magical_number(N :: integer(), A :: integer(), B :: integer()) ->
integer().
nth_magical_number(N, A, B) ->
.
```

Racket Solution:

```
(define/contract (nth-magical-number n a b)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```