Problem 2939: Maximum Xor Product

Problem Information

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

Problem Description

Given three integers

a

,

b

, and

n

, return

the

maximum value

of

(a XOR x) * (b XOR x)

where

0 <= x < 2

n

.

Since the answer may be too large, return it

modulo

10

9

+ 7

.

Note

that

XOR

is the bitwise XOR operation.

Example 1:

Input:

a = 12, b = 5, n = 4

Output:

98

Explanation:

For x = 2, (a XOR x) = 14 and (b XOR x) = 7. Hence, (a XOR x) * (b XOR x) = 98. It can be shown that 98 is the maximum value of (a XOR x) * (b XOR x) for all 0 <= x < 2

n

.

Example 2:

Input:

a = 6, b = 7 , n = 5

Output:

930

Explanation:

For x = 25, (a XOR x) = 31 and (b XOR x) = 30. Hence, (a XOR x) * (b XOR x) = 930. It can be shown that 930 is the maximum value of (a XOR x) * (b XOR x) for all 0 <= x < 2

n

.

Example 3:

Input:

a = 1, b = 6, n = 3

Output:

12

Explanation:

For x = 5, (a XOR x) = 4 and (b XOR x) = 3. Hence, (a XOR x) * (b XOR x) = 12. It can be shown that 12 is the maximum value of (a XOR x) * (b XOR x) for all 0 <= x < 2

n

.

Constraints:

0 <= a, b < 2

50

0 <= n <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:type b: int
:type n: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int maximumXorProduct(long long a, long long b, int n) {

}
```

Go:

```
func maximumXorProduct(a int64, b int64, n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn maximum_xor_product(a: i64, b: i64, n: i32) -> i32 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @param Integer $n
* @return Integer
```

```
*/
function maximumXorProduct($a, $b, $n) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-xor-product a b n)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Xor Product
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* 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 maximumXorProduct(long long a, long long b, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Xor Product
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* 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 maximumXorProduct(long a, long b, int n) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Xor Product
Difficulty: Medium
Tags: greedy, math

Approach: Greedy algorithm with local optimal choices
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maximumXorProduct(self, a: int, b: int, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Xor Product
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Xor Product
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function maximumXorProduct(a: number, b: number, n: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Xor Product
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* 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 MaximumXorProduct(long a, long b, int n) {
```

```
}
}
```

C Solution:

```c
/*
* Problem: Maximum Xor Product
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int maximumXorProduct(long long a, long long b, int n) {

}
```

Go Solution:

```go
// Problem: Maximum Xor Product
// Difficulty: Medium
// Tags: greedy, math
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func maximumXorProduct(a int64, b int64, n int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun maximumXorProduct(a: Long, b: Long, n: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Xor Product
// Difficulty: Medium
// Tags: greedy, math
//
// Approach: Greedy algorithm with local optimal choices
// 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 maximum_xor_product(a: i64, b: i64, n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $a
* @param Integer $b
* @param Integer $n
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-xor-product a b n)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```