Problem 779: K-th Symbol in Grammar

Problem Information

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

Problem Description

We build a table of

n

rows (

1-indexed

). We start by writing

0

in the

1

st

row. Now in every subsequent row, we look at the previous row and replace each occurrence of

0

with

01

, and each occurrence of

1

with

10

.

For example, for

n = 3

, the

1

st

row is

0

, the

2

nd

row is

01

, and the

3

rd

row is

0110

.

Given two integer

n

and

k

, return the

k

th

(

1-indexed

) symbol in the

n

th

row of a table of

n

rows.

Example 1:

Input:

n = 1, k = 1

Output:

0

Explanation:

row 1:

0

Example 2:

Input:

n = 2, k = 1

Output:

0

Explanation:

row 1: 0 row 2:

0

1

Example 3:

Input:

n = 2, k = 2

Output:

1

Explanation:

row 1: 0 row 2: 0

1

Constraints:

1 <= n <= 30

1 <= k <= 2

n - 1

Code Snippets

C++:

```
class Solution {
public:
int kthGrammar(int n, int k) {

}
};
```

Java:

```
class Solution {
public int kthGrammar(int n, int k) {

}
}
```

Python3:

```
class Solution:
def kthGrammar(self, n: int, k: int) -> int:
```

Python:

```
class Solution(object):
def kthGrammar(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function kthGrammar(n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int KthGrammar(int n, int k) {

}
}
```

C:

```
int kthGrammar(int n, int k) {

}
```

Go:

```
func kthGrammar(n int, k int) int {

}
```

Kotlin:

```
class Solution {
fun kthGrammar(n: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func kthGrammar(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn kth_grammar(n: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def kth_grammar(n, k)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function kthGrammar($n, $k) {

}
}
```

Dart:

```
class Solution {
int kthGrammar(int n, int k) {

}
}
```

Scala:

```
object Solution {
def kthGrammar(n: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec kth_grammar(n :: integer, k :: integer) :: integer
def kth_grammar(n, k) do

end
end
```

Erlang:

```
-spec kth_grammar(N :: integer(), K :: integer()) -> integer().
kth_grammar(N, K) ->
.
```

Racket:

```
(define/contract (kth-grammar n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: K-th Symbol in Grammar
* 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 kthGrammar(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: K-th Symbol in Grammar
* 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 kthGrammar(int n, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: K-th Symbol in Grammar
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 kthGrammar(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def kthGrammar(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: K-th Symbol in Grammar
* 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} n
* @param {number} k
* @return {number}
*/
var kthGrammar = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: K-th Symbol in Grammar
* 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 kthGrammar(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: K-th Symbol in Grammar
* 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 KthGrammar(int n, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: K-th Symbol in Grammar
* 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 kthGrammar(int n, int k) {

}
```

Go Solution:

```
// Problem: K-th Symbol in Grammar
// 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 kthGrammar(n int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun kthGrammar(n: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func kthGrammar(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: K-th Symbol in Grammar
// 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 kth_grammar(n: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def kth_grammar(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function kthGrammar($n, $k) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int kthGrammar(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def kthGrammar(n: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec kth_grammar(n :: integer, k :: integer) :: integer
def kth_grammar(n, k) do

end
end
```

Erlang Solution:

```
-spec kth_grammar(N :: integer(), K :: integer()) -> integer().
kth_grammar(N, K) ->
.
```

Racket Solution:

```
(define/contract (kth-grammar n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```