Problem 38: Count and Say

Problem Information

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

Problem Description

The

count-and-say

sequence is a sequence of digit strings defined by the recursive formula:

countAndSay(1) = "1"

countAndSay(n)

is the run-length encoding of

countAndSay(n - 1)

.

Run-length encoding

(RLE) is a string compression method that works by replacing consecutive identical characters (repeated 2 or more times) with the concatenation of the character and the number marking the count of the characters (length of the run). For example, to compress the string

"3322251"

we replace

"33"

with

"23"

, replace

"222"

with

"32"

, replace

"5"

with

"15"

and replace

"1"

with

"11"

. Thus the compressed string becomes

"23321511"

.

Given a positive integer

n

, return

the

n

th

element of the

count-and-say

sequence

.

Example 1:

Input:

n = 4

Output:

"1211"

Explanation:

countAndSay(1) = "1" countAndSay(2) = RLE of "1" = "11" countAndSay(3) = RLE of "11" = "21" countAndSay(4) = RLE of "21" = "1211"

Example 2:

Input:

n = 1

Output:

"1"

Explanation:

This is the base case.

Constraints:

1 <= n <= 30

Follow up:

Could you solve it iteratively?

Code Snippets

C++:

```
class Solution {
public:
string countAndSay(int n) {

}
};
```

Java:

```
class Solution {
public String countAndSay(int n) {

}
}
```

Python3:

```
class Solution:
def countAndSay(self, n: int) -> str:
```

Python:

```
class Solution(object):
def countAndSay(self, n):
"""
:type n: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {string}
*/
var countAndSay = function(n) {

};
```

TypeScript:

```
function countAndSay(n: number): string {

};
```

C#:

```
public class Solution {
public string CountAndSay(int n) {

}
}
```

C:

```
char* countAndSay(int n) {

}
```

Go:

```
func countAndSay(n int) string {

}
```

Kotlin:

```
class Solution {
fun countAndSay(n: Int): String {

}
}
```

Swift:

```
class Solution {
func countAndSay(_ n: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn count_and_say(n: i32) -> String {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {String}
def count_and_say(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return String
*/
function countAndSay($n) {

}
```

```
}
```

Dart:

```
class Solution {
String countAndSay(int n) {

}
}
```

Scala:

```
object Solution {
def countAndSay(n: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_and_say(n :: integer) :: String.t
def count_and_say(n) do

end
end
```

Erlang:

```
-spec count_and_say(N :: integer()) -> unicode:unicode_binary().
count_and_say(N) ->
.
```

Racket:

```
(define/contract (count-and-say n)
(-> exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count and Say
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
string countAndSay(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Count and Say
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String countAndSay(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Count and Say
Difficulty: Medium
Tags: string
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def countAndSay(self, n: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countAndSay(self, n):
"""
:type n: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Count and Say
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {string}
*/
var countAndSay = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Count and Say
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function countAndSay(n: number): string {

};
```

C# Solution:

```
/*
* Problem: Count and Say
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string CountAndSay(int n) {

}
}
```

C Solution:

```
/*
* Problem: Count and Say
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

char* countAndSay(int n) {

}
```

Go Solution:

```
// Problem: Count and Say
// Difficulty: Medium
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func countAndSay(n int) string {

}
```

Kotlin Solution:

```
class Solution {
fun countAndSay(n: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func countAndSay(_ n: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Count and Say
// Difficulty: Medium
// Tags: string
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn count_and_say(n: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {String}
def count_and_say(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return String
*/
function countAndSay($n) {

}
}
```

Dart Solution:

```
class Solution {
String countAndSay(int n) {

}
}
```

Scala Solution:

```
object Solution {
def countAndSay(n: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_and_say(n :: integer) :: String.t
def count_and_say(n) do

end
end
```

Erlang Solution:

```
-spec count_and_say(N :: integer()) -> unicode:unicode_binary().
count_and_say(N) ->
.
```

Racket Solution:

```
(define/contract (count-and-say n)
(-> exact-integer? string?)
)
```