Problem 3211: Generate Binary Strings Without Adjacent Zeros

Problem Information

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

Problem Description

You are given a positive integer

n

.

A binary string

x

is

valid

if all

substrings

of

x

of length 2 contain

at least

one

"1"

.

Return all

valid

strings with length

n

,

in

any

order.

Example 1:

Input:

n = 3

Output:

["010","011","101","110","111"]

Explanation:

The valid strings of length 3 are:

"010"

,

"011"

,

"101"

,

"110"

, and

"111"

.

Example 2:

Input:

n = 1

Output:

["0","1"]

Explanation:

The valid strings of length 1 are:

"0"

and

"1"

.

Constraints:

1 <= n <= 18

Code Snippets

C++:

```
class Solution {
public:
vector<string> validStrings(int n) {

}
};
```

Java:

```
class Solution {
public List<String> validStrings(int n) {

}
}
```

Python3:

```
class Solution:
def validStrings(self, n: int) -> List[str]:
```

Python:

```
class Solution(object):
def validStrings(self, n):
"""
:type n: int
:rtype: List[str]
"""
```

JavaScript:

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

};
```

TypeScript:

```
function validStrings(n: number): string[] {

};
```

C#:

```
public class Solution {
public IList<string> ValidStrings(int n) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** validStrings(int n, int* returnSize) {

}
```

Go:

```
func validStrings(n int) []string {

}
```

Kotlin:

```
class Solution {
fun validStrings(n: Int): List<String> {

}
}
```

Swift:

```
class Solution {
func validStrings(_ n: Int) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn valid_strings(n: i32) -> Vec<String> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {String[]}
def valid_strings(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return String[]
*/
function validStrings($n) {

}
}
```

Dart:

```
class Solution {
List<String> validStrings(int n) {

}
}
```

Scala:

```
object Solution {
def validStrings(n: Int): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec valid_strings(n :: integer) :: [String.t]
def valid_strings(n) do

end
end
```

Erlang:

```
-spec valid_strings(N :: integer()) -> [unicode:unicode_binary()].
valid_strings(N) ->
.
```

Racket:

```
(define/contract (valid-strings n)
(-> exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Generate Binary Strings Without Adjacent Zeros
* Difficulty: Medium
* Tags: string, tree
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<string> validStrings(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Generate Binary Strings Without Adjacent Zeros
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public List<String> validStrings(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Generate Binary Strings Without Adjacent Zeros
Difficulty: Medium
Tags: string, tree

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""
```

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

Python Solution:

```
class Solution(object):
def validStrings(self, n):
"""
:type n: int
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Generate Binary Strings Without Adjacent Zeros
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number} n
* @return {string[]}
*/
var validStrings = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Generate Binary Strings Without Adjacent Zeros
* Difficulty: Medium
* Tags: string, tree
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function validStrings(n: number): string[] {

};
```

C# Solution:

```
/*
* Problem: Generate Binary Strings Without Adjacent Zeros
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public IList<string> ValidStrings(int n) {

}
}
```

C Solution:

```
/*
* Problem: Generate Binary Strings Without Adjacent Zeros
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
```

```
* Note: The returned array must be malloced, assume caller calls free().
*/
char** validStrings(int n, int* returnSize) {

}
```

Go Solution:

```
// Problem: Generate Binary Strings Without Adjacent Zeros
// Difficulty: Medium
// Tags: string, tree
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func validStrings(n int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun validStrings(n: Int): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func validStrings(_ n: Int) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Generate Binary Strings Without Adjacent Zeros
// Difficulty: Medium
// Tags: string, tree
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn valid_strings(n: i32) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {String[]}
def valid_strings(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return String[]
*/
function validStrings($n) {

}
}
```

Dart Solution:

```
class Solution {
List<String> validStrings(int n) {

}
}
```

Scala Solution:

```
object Solution {
def validStrings(n: Int): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec valid_strings(n :: integer) :: [String.t]
def valid_strings(n) do

end
end
```

Erlang Solution:

```
-spec valid_strings(N :: integer()) -> [unicode:unicode_binary()].
valid_strings(N) ->
.
```

Racket Solution:

```
(define/contract (valid-strings n)
(-> exact-integer? (listof string?))
)
```