Problem 1415: The k-th Lexicographical String of All Happy Strings of Length n

Problem Information

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

Problem Description

A

happy string

is a string that:

consists only of letters of the set

['a', 'b', 'c']

.

s[i] != s[i + 1]

for all values of

i

from

1

to

s.length - 1

(string is 1-indexed).

For example, strings

"abc", "ac", "b"

and

"abcbabcbcb"

are all happy strings and strings

"aa", "baa"

and

"ababbc"

are not happy strings.

Given two integers

n

and

k

, consider a list of all happy strings of length

n

sorted in lexicographical order.

Return

the kth string

of this list or return an

empty string

if there are less than

k

happy strings of length

n

.

Example 1:

Input:

n = 1, k = 3

Output:

"c"

Explanation:

The list ["a", "b", "c"] contains all happy strings of length 1. The third string is "c".

Example 2:

Input:

n = 1, k = 4

Output:

""

Explanation:

There are only 3 happy strings of length 1.

Example 3:

Input:

n = 3, k = 9

Output:

"cab"

Explanation:

There are 12 different happy string of length 3 ["aba", "abc", "aca", "acb", "bab", "bac", "bca", "bcb", "cab", "cac", "cba", "cbc"]. You will find the 9

th

string = "cab"

Constraints:

1 <= n <= 10

1 <= k <= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public String getHappyString(int n, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function getHappyString(n: number, k: number): string {

};
```

C#:

```
public class Solution {
public string GetHappyString(int n, int k) {

}
}
```

C:

```
char* getHappyString(int n, int k) {

}
```

Go:

```
func getHappyString(n int, k int) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return String
*/
function getHappyString($n, $k) {

}
}
```

Dart:

```
class Solution {
String getHappyString(int n, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec get_happy_string(n :: integer, k :: integer) :: String.t
def get_happy_string(n, k) do
```

```
end
end
```

Erlang:

```
-spec get_happy_string(N :: integer(), K :: integer()) ->
unicode:unicode_binary().
get_happy_string(N, K) ->
.
```

Racket:

```
(define/contract (get-happy-string n k)
(-> exact-integer? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The k-th Lexicographical String of All Happy Strings of Length n
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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 getHappyString(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: The k-th Lexicographical String of All Happy Strings of Length n
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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 getHappyString(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: The k-th Lexicographical String of All Happy Strings of Length n
Difficulty: Medium
Tags: string, graph, sort

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 getHappyString(self, n: int, k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: The k-th Lexicographical String of All Happy Strings of Length n
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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
* @param {number} k
* @return {string}
*/
var getHappyString = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: The k-th Lexicographical String of All Happy Strings of Length n
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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 getHappyString(n: number, k: number): string {

};
```

C# Solution:

```
/*
* Problem: The k-th Lexicographical String of All Happy Strings of Length n
* Difficulty: Medium
```

```
* Tags: string, graph, sort
*
* 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 GetHappyString(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: The k-th Lexicographical String of All Happy Strings of Length n
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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* getHappyString(int n, int k) {

}
```

Go Solution:

```
// Problem: The k-th Lexicographical String of All Happy Strings of Length n
// Difficulty: Medium
// Tags: string, graph, sort
//
// 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 getHappyString(n int, k int) string {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: The k-th Lexicographical String of All Happy Strings of Length n
// Difficulty: Medium
// Tags: string, graph, sort
//
// 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 get_happy_string(n: i32, k: i32) -> String {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return String
*/
function getHappyString($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
String getHappyString(int n, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_happy_string(n :: integer, k :: integer) :: String.t
def get_happy_string(n, k) do

end
end
```

Erlang Solution:

```
-spec get_happy_string(N :: integer(), K :: integer()) ->
unicode:unicode_binary().
get_happy_string(N, K) ->
.
```

Racket Solution:

```
(define/contract (get-happy-string n k)
(-> exact-integer? exact-integer? string?)
)
```