Problem 753: Cracking the Safe

Problem Information

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

Problem Description

There is a safe protected by a password. The password is a sequence of

n

digits where each digit can be in the range

[0, k - 1]

.

The safe has a peculiar way of checking the password. When you enter in a sequence, it checks the

most recent

n

digits

that were entered each time you type a digit.

For example, the correct password is

"345"

and you enter in

"012345"

:

After typing

0

, the most recent

3

digits is

"0"

, which is incorrect.

After typing

1

, the most recent

3

digits is

"01"

, which is incorrect.

After typing

2

, the most recent

3

digits is

"012"

, which is incorrect.

After typing

3

, the most recent

3

digits is

"123"

, which is incorrect.

After typing

4

, the most recent

3

digits is

"234"

, which is incorrect.

After typing

5

, the most recent

3

digits is

"345"

, which is correct and the safe unlocks.

Return

any string of

minimum length

that will unlock the safe

at some point

of entering it

.

Example 1:

Input:

n = 1, k = 2

Output:

"10"

Explanation:

The password is a single digit, so enter each digit. "01" would also unlock the safe.

Example 2:

Input:

n = 2, k = 2

Output:

"01100"

Explanation:

For each possible password: - "00" is typed in starting from the 4

th

digit. - "01" is typed in starting from the 1

st

digit. - "10" is typed in starting from the 3

rd

digit. - "11" is typed in starting from the 2

nd

digit. Thus "01100" will unlock the safe. "10011", and "11001" would also unlock the safe.

Constraints:

1 <= n <= 4

1 <= k <= 10

1 <= k

n

<= 4096

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} n
* @param {number} k
```

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (crack-safe n k)
(-> exact-integer? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Cracking the Safe
* Difficulty: Hard
* Tags: string, graph, search
*
```

```
* 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 crackSafe(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Cracking the Safe
* Difficulty: Hard
* Tags: string, graph, search
*
* 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 crackSafe(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Cracking the Safe
Difficulty: Hard
Tags: string, graph, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Cracking the Safe
* Difficulty: Hard
* Tags: string, graph, search
*
* 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 crackSafe = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Cracking the Safe
```

```
* Difficulty: Hard
* Tags: string, graph, search
*
* 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 crackSafe(n: number, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Cracking the Safe
* Difficulty: Hard
* Tags: string, graph, search
*
* 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 CrackSafe(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Cracking the Safe
* Difficulty: Hard
* Tags: string, graph, search
*
* 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* crackSafe(int n, int k) {

}
```

Go Solution:

```
// Problem: Cracking the Safe
// Difficulty: Hard
// Tags: string, graph, search
//
// 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 crackSafe(n int, k int) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Cracking the Safe
// Difficulty: Hard
// Tags: string, graph, search
//
```

```
// 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 crack_safe(n: i32, k: i32) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (crack-safe n k)
(-> exact-integer? exact-integer? string?)
)
```