Problem 887: Super Egg Drop

Problem Information

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

Problem Description

You are given

k

identical eggs and you have access to a building with

n

floors labeled from

1

to

n

.

You know that there exists a floor

f

where

0 <= f <= n

such that any egg dropped at a floor

higher

than

f

will

break

, and any egg dropped

at or below

floor

f

will

not break

.

Each move, you may take an unbroken egg and drop it from any floor

x

(where

1 <= x <= n

). If the egg breaks, you can no longer use it. However, if the egg does not break, you may

reuse

it in future moves.

Return

the

minimum number of moves

that you need to determine

with certainty

what the value of

f

is.

Example 1:

Input:

k = 1, n = 2

Output:

2

Explanation:

Drop the egg from floor 1. If it breaks, we know that f = 0. Otherwise, drop the egg from floor 2. If it breaks, we know that f = 1. If it does not break, then we know f = 2. Hence, we need at minimum 2 moves to determine with certainty what the value of f is.

Example 2:

Input:

k = 2, n = 6

Output:

3

Example 3:

Input:

k = 3, n = 14

Output:

4

Constraints:

1 <= k <= 100

1 <= n <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
.
```

Racket:

```
(define/contract (super-egg-drop k n)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Super Egg Drop
* Difficulty: Hard
* Tags: dp, math, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int superEggDrop(int k, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Super Egg Drop
* Difficulty: Hard
* Tags: dp, math, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

}
}
```

Python3 Solution:

```
"""
Problem: Super Egg Drop
Difficulty: Hard
Tags: dp, math, search

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Super Egg Drop
* Difficulty: Hard
* Tags: dp, math, search
*
```

```
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Super Egg Drop
* Difficulty: Hard
* Tags: dp, math, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function superEggDrop(k: number, n: number): number {

};
```

C# Solution:

```
/*
* Problem: Super Egg Drop
* Difficulty: Hard
* Tags: dp, math, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

}
}
```

C Solution:

```
/*
* Problem: Super Egg Drop
* Difficulty: Hard
* Tags: dp, math, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int superEggDrop(int k, int n) {

}
```

Go Solution:

```
// Problem: Super Egg Drop
// Difficulty: Hard
// Tags: dp, math, search
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func superEggDrop(k int, n int) int {

}
```

Kotlin Solution:

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

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Super Egg Drop
// Difficulty: Hard
// Tags: dp, math, search
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (super-egg-drop k n)
(-> exact-integer? exact-integer? exact-integer?)
)
```