Problem 1884: Egg Drop With 2 Eggs and N Floors

Problem Information

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

Problem Description

You are given

two 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

.

In 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:

n = 2

Output:

2

Explanation:

We can drop the first egg from floor 1 and the second egg from floor 2. If the first egg breaks, we know that f = 0. If the second egg breaks but the first egg didn't, we know that f = 1. Otherwise, if both eggs survive, we know that f = 2.

Example 2:

Input:

n = 100

Output:

14

Explanation:

One optimal strategy is: - Drop the 1st egg at floor 9. If it breaks, we know f is between 0 and 8. Drop the 2nd egg starting from floor 1 and going up one at a time to find f within 8 more drops. Total drops is 1 + 8 = 9. - If the 1st egg does not break, drop the 1st egg again at floor 22. If it breaks, we know f is between 9 and 21. Drop the 2nd egg starting from floor 10 and going up one at a time to find f within 12 more drops. Total drops is 2 + 12 = 14. - If the 1st egg does not break again, follow a similar process dropping the 1st egg from floors 34, 45, 55, 64, 72, 79, 85, 90, 94, 97, 99, and 100. Regardless of the outcome, it takes at most 14 drops to determine f.

Constraints:

1 <= n <= 1000

Code Snippets

C++:

```
class Solution {
public:
int twoEggDrop(int n) {

}
};
```

Java:

```
class Solution {
public int twoEggDrop(int n) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function twoEggDrop(n: number): number {

};
```

C#:

```
public class Solution {
public int TwoEggDrop(int n) {

}
}
```

C:

```
int twoEggDrop(int n) {

}
```

Go:

```
func twoEggDrop(n int) int {

}
```

Kotlin:

```
class Solution {
fun twoEggDrop(n: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int twoEggDrop(int n) {

}
}
```

Scala:

```
object Solution {
def twoEggDrop(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec two_egg_drop(n :: integer) :: integer
def two_egg_drop(n) do

end
end
```

Erlang:

```
-spec two_egg_drop(N :: integer()) -> integer().
two_egg_drop(N) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Egg Drop With 2 Eggs and N Floors
* Difficulty: Medium
* Tags: dp, math
*
* 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 twoEggDrop(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Egg Drop With 2 Eggs and N Floors
* Difficulty: Medium
* Tags: dp, math
*
* 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 twoEggDrop(int n) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Egg Drop With 2 Eggs and N Floors
Difficulty: Medium
Tags: dp, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Egg Drop With 2 Eggs and N Floors
* Difficulty: Medium
* Tags: dp, math
*
* 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} n
* @return {number}
*/
var twoEggDrop = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Egg Drop With 2 Eggs and N Floors
* Difficulty: Medium
* Tags: dp, math
*
* 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 twoEggDrop(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Egg Drop With 2 Eggs and N Floors
* Difficulty: Medium
* Tags: dp, math
*
* 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 TwoEggDrop(int n) {

}
}
```

C Solution:

```
/*
* Problem: Egg Drop With 2 Eggs and N Floors
* Difficulty: Medium
* Tags: dp, math
*
* 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 twoEggDrop(int n) {

}
```

Go Solution:

```
// Problem: Egg Drop With 2 Eggs and N Floors
// Difficulty: Medium
// Tags: dp, math
//
// 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 twoEggDrop(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun twoEggDrop(n: Int): Int {

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Egg Drop With 2 Eggs and N Floors
// Difficulty: Medium
// Tags: dp, math
//
// 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 two_egg_drop(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int twoEggDrop(int n) {

}
}
```

Scala Solution:

```
object Solution {
def twoEggDrop(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec two_egg_drop(n :: integer) :: integer
def two_egg_drop(n) do

end
end
```

Erlang Solution:

```
-spec two_egg_drop(N :: integer()) -> integer().
two_egg_drop(N) ->
.
```

Racket Solution:

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