1884. Egg Drop With 2 Eggs and N Floors

Leetcode Link

Problem Explanation:

In this problem, we are given two identical eggs and a building with n floors (numbered 1 to n). We must find the 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. The goal is to find the minimum number of moves required to determine the value of f with certainty. To solve this problem, we'll be using dynamic programming.

Let's work through an example:

Example

```
Input: n = 2
```

Output: 2

For this example, we have 2 floors. We can drop the first egg from floor 1 and the second egg from floor 2:

- 1. If the first egg breaks, we know that f = 0.
- 2. If the second egg breaks but the first egg didn't, we know that f = 1.
- 3. Otherwise, if both eggs survive, we know that f = 2.

We only needed 2 moves to determine the value of f, so the output is 2.

The solution approach for this problem includes using dynamic programming and a 2D array dp with n + 1 rows and 2 columns,

Solution Approach

where dp[i][j] represents the minimum number of moves required to determine the value of f with i floors and j + 1 eggs. We will loop over all possible values of j and get the minimum possible value of dp[i][1].

1. Create a 2D array dp with n+1 rows and 2 columns.

Algorithm Steps

- 2. Set dp[i][0] = i for all $0 \le i \le n$.
- 3. Set dp[1][1] = 1 for a single floor.
- 4. For 2 \ll i \ll n, if the previous egg was dropped from floor j (1 \ll j < i), then dp[i][1] = max(dp[j 1][0], dp[i j] [1]) + 1.5. Loop over all possible values of j and get the minimum possible value of dp[i][1].
- 6. Finally, return dp[n][1].

Python Solution

```
2 python
   class Solution:
        def twoEggDrop(self, n: int) -> int:
            def drop(k: int, N: int) -> int:
                if k == 0:
 6
                    return 0
 8
                if k == 1:
                    return N
 9
10
                if N == 0:
11
                    return 0
12
                if N == 1:
13
                    return 1
14
                if dp[k][N] != -1:
15
                    return dp[k][N]
16
17
                l, r = 1, N + 1
18
19
                while l < r:
20
                    m = (l + r) // 2
21
                    broken = drop(k - 1, m - 1)
22
                    unbroken = drop(k, N - m)
                    if broken >= unbroken:
23
24
                         r = m
25
                    else:
26
                        l = m + 1
27
28
                dp[k][N] = 1 + drop(k - 1, l - 1)
29
                return dp[k][N]
30
31
            dp = [[-1] * (n + 1) for _ in range(3)]
32
            return drop(2, n)
```

Java Solution

```
2 java
   class Solution {
        public int twoEggDrop(int n) {
            int[][] dp = new int[3][n + 1];
            for (int i = 0; i \le n; i++) {
                dp[0][i] = 0;
                dp[1][i] = i;
 9
                dp[2][i] = -1;
10
11
12
13
            for (int i = 1; i \le n; i++) {
                int minMoves = Integer.MAX_VALUE;
14
                for (int j = 1; j < i; j++) {
15
                    int moves = Math.max(dp[0][j - 1], dp[1][i - j]) + 1;
16
17
                    minMoves = Math.min(minMoves, moves);
18
                dp[2][i] = minMoves;
19
20
21
                if (dp[1][i] == dp[2][i]) {
22
                    break;
23
24
25
                dp[1][i] = dp[2][i];
26
27
28
            return dp[2][n];
29
30 }
```

class Solution { twoEggDrop(n) {

2 javascript

JavaScript Solution

```
const dp = Array.from(\{length: 3\}, () => Array(n + 1).fill(-1));
 6
            for (let i = 0; i \le n; i++) {
                dp[0][i] = 0;
                dp[1][i] = i;
10
11
12
            for (let i = 1; i \le n; i++) {
13
                dp[2][i] = dp[1][i];
                for (let j = 1; j < i; j++) {
14
                    const moves = Math.max(dp[0][j - 1], dp[1][i - j]) + 1;
15
                    dp[2][i] = Math.min(dp[2][i], moves);
16
17
18
19
                if (dp[1][i] === dp[2][i]) {
20
                    break;
21
22
23
                dp[1][i] = dp[2][i];
24
25
26
            return dp[2][n];
27
28 }
C++ Solution
```

int twoEggDrop(int n) { vector<vector<int>> dp(3, vector<int>(n + 1, −1));

class Solution {

срр

public:

```
for (int i = 0; i \le n; i++) {
 8
                dp[0][i] = 0;
 9
                dp[1][i] = i;
10
11
13
            for (int i = 1; i \le n; i++) {
14
                dp[2][i] = dp[1][i];
                for (int j = 1; j < i; j++) {
15
                    int moves = \max(dp[0][j-1], dp[1][i-j]) + 1;
16
                    dp[2][i] = min(dp[2][i], moves);
17
18
19
20
                if (dp[1][i] == dp[2][i]) {
21
                    break;
22
23
24
                dp[1][i] = dp[2][i];
25
26
27
            return dp[2][n];
28
29 };
C# Solution
   csharp
   public class Solution {
       public int TwoEggDrop(int n) {
            int[][] dp = new int[3][];
            for (int i = 0; i < 3; i++) {
                dp[i] = new int[n + 1];
```

9 10 for (int i = 0; $i \le n$; i++) {

```
dp[0][i] = 0;
11
                dp[1][i] = i;
12
13
                dp[2][i] = -1;
14
15
16
            for (int i = 1; i \le n; i++) {
                int minMoves = int.MaxValue;
                for (int j = 1; j < i; j++) {
                    int moves = Math.Max(dp[0][j-1], dp[1][i-j]) + 1;
19
                    minMoves = Math.Min(minMoves, moves);
20
21
                dp[2][i] = minMoves;
22
23
24
                if (dp[1][i] == dp[2][i]) {
25
                    break;
26
27
28
                dp[1][i] = dp[2][i];
30
            return dp[2][n];
31
32
33 }
```

Got a question? Ask the Teaching Assistant anything you don't understand.

In the solutions above, the logic behind the code in each language is explained step-by-step in the algorithm steps section, as we perform dynamic programming to build up the dp array and find the minimum number of moves required to determine the value of f. The solutions above for Python, Java, JavaScript, C++, and C# define the standard dynamic programming method for solving the egg drop problem with two eggs and n floors. These solutions have a time complexity of O(n^2) and O(n) space complexity. You can try implementing the given solutions in your preferred programming language to understand how to solve this problem effectively.

process of problem-solving using dynamic programming. This helps us break down complex problems into simpler sub-problems and solve them in a systematic manner for efficient solutions. Besides, practicing with multiple languages will help you become a better programmer in general, as it strengthens your ability to adapt and learn different programming paradigms. In conclusion, the egg drop problem is a classic question in dynamic programming that can be solved using various techniques. The

methods provided above are well-explained and optimized to solve this problem for given constraints. Implementing the solutions in

different programming languages will help you enhance your programming skills and strengthen your understanding of dynamic

By following the algorithm steps mentioned above and analyzing the code provided for each language, we can better understand the

