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.

Solution Approach

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

Algorithm Steps

1. Create a 2D array `dp` with `n+1` rows and 2 columns.
2. Set `dp[i][0] = i` for all `0 <= i <= n`.
3. Set `dp[1][1] = 1` for a single floor.
4. For `2 <= i <= n`, if the previous egg was dropped from floor `j (1 <= 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

```

python
class Solution:
    def twoEggDrop(self, n: int) -> int:
        def drop(k: int, N: int) -> int:
            if k == 0:
                return 0
            if k == 1:
                return N
            if N == 0:
                return 0
            if N == 1:
                return 1
            if dp[k][N] != -1:
                return dp[k][N]

            l, r = 1, N + 1

            while l < r:
                m = (l + r) // 2
                broken = drop(k - 1, m - 1)
                unbroken = drop(k, N - m)
                if broken >= unbroken:
                    r = m
                else:
                    l = m + 1

            dp[k][N] = 1 + drop(k - 1, l - 1)
            return dp[k][N]

        dp = [[-1] * (n + 1) for _ in range(3)]
        return drop(2, n)
```

Java Solution

```

java
class Solution {
    public int twoEggDrop(int n) {
        int[][] dp = new int[3][n + 1];

        for (int i = 0; i <= n; i++) {
            dp[0][i] = 0;
            dp[1][i] = i;
            dp[2][i] = -1;
        }

        for (int i = 1; i <= n; i++) {
            int minMoves = Integer.MAX_VALUE;
            for (int j = 1; j < i; j++) {
                int moves = Math.max(dp[0][j - 1], dp[1][i - j]) + 1;
                minMoves = Math.min(minMoves, moves);
            }
            dp[2][i] = minMoves;

            if (dp[1][i] == dp[2][i]) {
                break;
            }

            dp[1][i] = dp[2][i];
        }

        return dp[2][n];
    }
}
```

JavaScript Solution

```

javascript
class Solution {
    twoEggDrop(n) {
        const dp = Array.from({length: 3}, () => Array(n + 1).fill(-1));

        for (let i = 0; i <= n; i++) {
            dp[0][i] = 0;
            dp[1][i] = i;
        }

        for (let i = 1; i <= n; i++) {
            dp[2][i] = dp[1][i];
            for (let j = 1; j < i; j++) {
                const moves = Math.max(dp[0][j - 1], dp[1][i - j]) + 1;
                dp[2][i] = Math.min(dp[2][i], moves);
            }

            if (dp[1][i] === dp[2][i]) {
                break;
            }

            dp[1][i] = dp[2][i];
        }

        return dp[2][n];
    }
}
```

C++ Solution

```

cpp
class Solution {
public:
    int twoEggDrop(int n) {
        vector<vector<int>> dp(3, vector<int>(n + 1, -1));

        for (int i = 0; i <= n; i++) {
            dp[0][i] = 0;
            dp[1][i] = i;
        }

        for (int i = 1; i <= n; i++) {
            dp[2][i] = dp[1][i];
            for (int j = 1; j < i; j++) {
                int moves = max(dp[0][j - 1], dp[1][i - j]) + 1;
                dp[2][i] = min(dp[2][i], moves);
            }

            if (dp[1][i] == dp[2][i]) {
                break;
            }

            dp[1][i] = dp[2][i];
        }

        return dp[2][n];
    }
};
```

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];
        }

        for (int i = 0; i <= n; i++) {
            dp[0][i] = 0;
            dp[1][i] = i;
            dp[2][i] = -1;
        }

        for (int i = 1; i <= 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;
                minMoves = Math.Min(minMoves, moves);
            }
            dp[2][i] = minMoves;

            if (dp[1][i] == dp[2][i]) {
                break;
            }

            dp[1][i] = dp[2][i];
        }

        return dp[2][n];
    }
}
```

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.

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