

# Leetcode 887: Super Egg Drop

107703006 資科四 楊永靖

# Problem

- 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  $0 \leq f \leq n$  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  $1 \leq x \leq n$ ). 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.

# Solution 1 (DP)

- $k$  eggs,  $n$  floor
- Drop egg from  $i$  floor.
  - Broken  $\Rightarrow$  test floors under  $i$  ( $k-1$  eggs remain)
  - Unbroken  $\Rightarrow$  test floors above  $i$  ( $n-i$  floors in total,  $k$  eggs remain)

# Recurrence Relation

$$\begin{array}{l} k \text{ eggs, } n \text{ floors} \\ \text{Base case } \left\{ \begin{array}{ll} k = 0 & \text{return } 0 \\ k = 1 & \text{return } n \\ n \leq 1 & \text{return } n \end{array} \right. \end{array}$$

$$ans = \min(ans, 1 + \max(dp(k - 1, i - 1), dp(k, n - i)))$$

Broken  $\Rightarrow dp(k - 1, i - 1)$   $\Rightarrow (k - 1)$  eggs and  $(i - 1)$  floors to check.

Unbroken  $\Rightarrow dp(k, n - i)$   $\Rightarrow (k)$  eggs and  $(n - i)$  floors to check.

$$D(2,6) = 3$$

$$\textcircled{1} \lambda = 1$$

$$1 + \text{Max}(D(1,0), D(2,5)) = 4$$

$$\textcircled{16} \lambda = 2$$

$$1 + \text{Max}(D(1,1), D(2,4)) = 4$$

$$\textcircled{17} \lambda = 3$$

$$1 + \text{Max}(D(1,2), D(2,3)) = 3$$

$$\textcircled{18} \lambda = 4$$

$$1 + \text{Max}(D(1,3), D(2,2)) = 4$$

$$\textcircled{19} \lambda = 5$$

$$1 + \text{Max}(D(1,4), D(2,1)) = 5$$

$$\textcircled{20} \lambda = 6$$

$$1 + \text{Max}(D(1,5), D(2,0)) = 6$$

$$D(2,5) = 3$$

$$\textcircled{2} \lambda = 1$$

$$1 + \text{Max}(D(1,0), D(2,4)) = 4$$

$$\textcircled{12} \lambda = 2$$

$$1 + \text{Max}(D(1,1), D(2,3)) = 3$$

$$\textcircled{13} \lambda = 3$$

$$1 + \text{Max}(D(1,2), D(2,2)) = 3$$

$$\textcircled{14} \lambda = 4$$

$$1 + \text{Max}(D(1,3), D(2,1)) = 4$$

$$\textcircled{15} \lambda = 5$$

$$1 + \text{Max}(D(1,4), D(2,0)) = 5$$

$$D(2,4) = 3$$

$$\textcircled{3} \lambda = 1$$

$$1 + \text{Max}(D(1,0), D(2,3)) = 3$$

$$\textcircled{9} \lambda = 2$$

$$1 + \text{Max}(D(1,1), D(2,2)) = 3$$

$$\textcircled{10} \lambda = 3$$

$$1 + \text{Max}(D(1,2), D(2,1)) = 3$$

$$\textcircled{11} \lambda = 4$$

$$1 + \text{Max}(D(1,3), D(2,0)) = 4$$

$$D(2,3) = 2$$

$$\textcircled{4} \lambda = 1$$

$$1 + \text{Max}(D(1,0), D(2,2)) = 3$$

$$\textcircled{7} \lambda = 2$$

$$1 + \text{Max}(D(1,1), D(2,1)) = 2$$

$$\textcircled{8} \lambda = 3$$

$$1 + \text{Max}(D(1,2), D(2,0)) = 3$$

$$D(2,2) = 2$$

$$\textcircled{5} \lambda = 1$$

$$1 + \text{Max}(D(1,0), D(2,1)) = 2$$

$$\textcircled{6} \lambda = 2$$

$$1 + \text{Max}(D(1,1), D(2,0)) = 2$$

# Code

```
int superEggDrop(int K, int N) {  
    // Init table for record  
    vector<vector<int>> table(K+1, vector<int>(N+1, INT_MAX));  
    // Recursion (lambda expression)  
    function<int(int, int)> dp = [&](int k, int n) {  
        // Base case  
        if(k == 0) return 0;  
        else if(k == 1) return n;  
        else if(n <= 1) return n;  
        int& ans = table[k][n];  
        if(ans != INT_MAX) return ans; // dp(k, n) has done  
        for(int i=1; i<=n; i++) { // Drop from i floor  
            ans = min(ans, 1 + max(dp(k-1, i-1), dp(k, n-i)));  
        }  
        return ans;  
    };  
    return dp(K, N);  
}
```

# Complexity (TLE)

# of subproblems  $O(KN)$ , complexity of each take  $O(N)$

Time complexity  $O(KN^2)$

Space complexity  $O(KN)$



## Solution 2 (DP)

- $f(x)$  is monotonic increase.
- $g(x)$  is monotonic decrease.
- Binary search.

$D(2,5) = 3$

②  $\lambda=1$   
 $1 + \text{Max}(D(1,0), D(2,4)) = 4$   
 $\begin{array}{cc} \text{"} & \text{"} \\ 0 & 3 \end{array}$

③  $\lambda=2$   
 $1 + \text{Max}(D(1,1), D(2,3)) = 3$   
 $\begin{array}{cc} \text{"} & \text{"} \\ 1 & 2 \end{array}$

⑬  $\lambda=3$   
 $1 + \text{Max}(D(1,2), D(2,2)) = 3$   
 $\begin{array}{cc} \text{"} & \text{"} \\ 2 & 2 \end{array}$

⑭  $\lambda=4$   
 $1 + \text{Max}(D(1,3), D(2,1)) = 4$   
 $\begin{array}{cc} \text{"} & \text{"} \\ 3 & 1 \end{array}$

⑮  $\lambda=5$   
 $1 + \text{Max}(D(1,4), D(2,0)) = 5$   
 $\begin{array}{cc} \text{"} & \text{"} \\ 4 & 0 \end{array}$

$f(x)$        $g(x)$



# Recurrence Relation

$k$  eggs,  $n$  floors

$$\text{Base case } \begin{cases} k = 0 & \text{return 0} \\ k = 1 & \text{return } n \\ n \leq 1 & \text{return } n \end{cases}$$

Search  $i$  with binary search.

$$ans = \min(ans, 1 + \max(dp(k - 1, i - 1), dp(k, n - i)))$$

# Code

```
int superEggDrop(int K, int N) {
    vector<vector<int>> table(K+1, vector<int>(N+1, INT_MAX));
    function<int(int, int)> dp = [&](int k, int n) {
        if(k == 0) return 0;
        else if(k == 1) return n;
        else if(n <= 1) return n;
        int& ans = table[k][n];
        if(ans != INT_MAX) return ans; // dp(k, n) has done
        int left = 1, right = n + 1;
        while(left < right) { // Binary search
            int i = left + (right - left) / 2; // Drop from i floor
            if(dp(k-1, i-1) >= dp(k, n-i)) right = i;
            else left = i + 1;
        }
        ans = 1 + max(dp(k-1, left-1), dp(k, n-left));
        return ans;
    };
    return dp(K, N);
}
```

# Complexity

# of subproblems  $O(KN)$ , complexity of each take  $O(\log N)$

Time complexity  $O(KN \log N)$

Space complexity  $O(KN)$

# More on binary search ...

Handwritten notes showing recursive calculations for a binary search problem, likely related to the edit distance or a similar dynamic programming problem. The calculations are organized into columns, each representing a different value of  $k$  (1, 2, 3, 4, 5, 6).

**Column 1:  $k=1$**

- $D(2, 6) = 3$
- ①  $\lambda=1$   
 $1 + \text{Max}(D(1, 0), D(2, 5)) = 4$
- ⑥  $\lambda=2$   
 $1 + \text{Max}(D(1, 1), D(2, 4)) = 4$
- ⑦  $\lambda=3$   
 $1 + \text{Max}(D(1, 2), D(2, 3)) = 3$
- ⑧  $\lambda=4$   
 $1 + \text{Max}(D(1, 3), D(2, 2)) = 4$
- ⑨  $\lambda=5$   
 $1 + \text{Max}(D(1, 4), D(2, 1)) = 5$
- ⑩  $\lambda=6$   
 $1 + \text{Max}(D(1, 5), D(2, 0)) = 6$

**Column 2:  $k=2$**

- $D(2, 5) = 3$
- ②  $\lambda=1$   
 $1 + \text{Max}(D(1, 0), D(2, 4)) = 4$
- ⑦  $\lambda=2$   
 $1 + \text{Max}(D(1, 1), D(2, 3)) = 3$
- ⑬  $\lambda=3$   
 $1 + \text{Max}(D(1, 2), D(2, 2)) = 3$
- ⑭  $\lambda=4$   
 $1 + \text{Max}(D(1, 3), D(2, 1)) = 4$
- ⑮  $\lambda=5$   
 $1 + \text{Max}(D(1, 4), D(2, 0)) = 5$

**Column 3:  $k=3$**

- $D(2, 4) = 3$
- ③  $\lambda=1$   
 $1 + \text{Max}(D(1, 0), D(2, 3)) = 3$
- ⑧  $\lambda=2$   
 $1 + \text{Max}(D(1, 1), D(2, 2)) = 3$
- ⑩  $\lambda=3$   
 $1 + \text{Max}(D(1, 2), D(2, 1)) = 3$
- ⑪  $\lambda=4$   
 $1 + \text{Max}(D(1, 3), D(2, 0)) = 4$

**Column 4:  $k=4$**

- $D(2, 3) = 2$
- ④  $\lambda=1$   
 $1 + \text{Max}(D(1, 0), D(2, 2)) = 3$
- ⑦  $\lambda=2$   
 $1 + \text{Max}(D(1, 1), D(2, 1)) = 2$
- ⑧  $\lambda=3$   
 $1 + \text{Max}(D(1, 2), D(2, 0)) = 3$

**Column 5:  $k=5$**

- $D(2, 2) = 2$
- ⑤  $\lambda=1$   
 $1 + \text{Max}(D(1, 0), D(2, 1)) = 2$
- ⑥  $\lambda=2$   
 $1 + \text{Max}(D(1, 1), D(2, 0)) = 2$

The calculations are organized into columns, each representing a different value of  $k$  (1, 2, 3, 4, 5, 6). The calculations are organized into columns, each representing a different value of  $k$  (1, 2, 3, 4, 5, 6). The calculations are organized into columns, each representing a different value of  $k$  (1, 2, 3, 4, 5, 6).

# Solution 3 (DP)

- Inverted problem.
- $dp(i)$  is the number of floors that can be measured with  $i$  eggs.
- Maintain a counter to count the number of moves.
- Test  $i$  from  $k$  to  $1$ .

# Recurrence Relation

$res$  = number of all moves now

$for(i = k \text{ to } 1) :$

$$dp[i] = dp[i - 1] + dp[i] + 1$$

Broken  $\Rightarrow dp(i - 1)$   $\Rightarrow$  Use  $(i - 1)$  eggs and check floor in  $(res - 1)$  moves.

Unbroken  $\Rightarrow dp(i)$   $\Rightarrow$  Use  $(i)$  eggs and check floor in  $(res - 1)$  moves.

Current floor  $\Rightarrow +1$

Using '+' instead of 'max' is because  $dp()$  stands for "moves" not "floors".

# Code

```
int superEggDrop(int K, int N) {  
    // Init table for record  
    vector<int> dp(K + 1);  
  
    int res = 0;  
    for (; dp[K] < N; ++res) { // Max floor smaller than given => drop  
        for (int i = K; i > 0; --i) { // Use  
            dp[i] = dp[i] + dp[i - 1] + 1;  
        }  
    }  
  
    return res;  
}
```



# Complexity

# of subproblems  $O(\log N)$ , complexity of each take  $O(K)$

Time complexity  $O(K \log N)$

Space complexity  $O(K)$

# Solution 4 (Mathematical Thought)

- If we have K eggs on hand, and we can move n times.
- Then set the maximum number of floors we can test to fun(k,n), and we can calculate the fun(k,n) from the following formula:

$$fun(k, n) = n + n(n-1)/2! + n(n-1)(n-2)/3! + \dots + n(n-1)(n-2)\dots(n-k)/k!$$

We can slowly increase n, or we can do a binary search between n and N to find The smallest n, so that fun(k,n) is greater than or equal to N.

[reference](#)

# Code

```
class Solution {
public:
    int superEggDrop(int k, int N) {
        if (k == 1 || N < 3) return N;
        //n is the number of eggs we need when we do binary search
        double n = log(N) / log(2);
        // If the number of eggs is not enough to support our binary search,
        // slowly increase the number of moves n until the return value of the fun function is
        // greater than or equal to the target floor N
        if (k < n++)
            while (fun(k, n) < N) ++n;
        return n;
    }
private:
    //In the worst case,
    // the maximum number of floors that can be tested if
    // the number of moves is n and the number of eggs is k
    int fun(int k, int n) {
        int i=1, temp = 1, maxNumOfF = 0;
        while (i <= k) {
            temp = temp*(n--) / (i++);
            maxNumOfF += temp;
        }
        return maxNumOfF;
    }
}
```