|  |  |
| --- | --- |
| **Egg drop C++** | |
| #include <iostream>  #include <climits>  using namespace std;  int eggDrop(int n, int k) {  // Initialize a 2D array for DP table  int dp[n + 1][k + 1]; // Array with (n + 1) rows and (k + 1) columns  for (int i = 0; i <= n; i++) {  for (int j = 0; j <= k; j++) {  dp[i][j] = 0;  }  }  // Fill the DP table  for (int i = 1; i <= n; i++) {  for (int j = 1; j <= k; j++) {  if (i == 1) {  dp[i][j] = j; // If only one egg is available, we need j trials  } else if (j == 1) {  dp[i][j] = 1; // If only one floor is there, one trial needed  } else {  int minDrops = INT\_MAX;  // Check all floors from 1 to j to find the minimum drops needed  for (int floor = 1; floor <= j; floor++) {  int breaks = dp[i - 1][floor - 1]; // Egg breaks, check below floors  int survives = dp[i][j - floor]; // Egg survives, check above floors  int maxDrops = 1 + max(breaks, survives); // Maximum drops needed in worst case  minDrops = min(minDrops, maxDrops); // Minimum drops to find the critical floor  }  dp[i][j] = minDrops;  }  }  }  return dp[n][k]; // Return the minimum drops needed  }  int main() {  int n = 4; // Number of eggs  int k = 2; // Number of floors  cout << eggDrop(n, k) << endl; // Output the minimum drops required  return 0;  } | **Step 1: Understanding the DP State**  * dp[i][j] = **Minimum number of trials** needed to find the critical floor with i eggs and j floors. * If we have **1 egg**, we must check **each floor one by one** → dp[1][j] = j * If we have **1 floor**, only **1 trial** is needed → dp[i][1] = 1  **Step 2: Dry Run for** n = 4 **(eggs),** k = 2 **(floors)** We build the **DP table** from dp[1][1] up to dp[4][2]. ****Step 2.1: Initialize Base Cases****  | **dp[i][j]** | **0 Floors** | **1 Floor** | **2 Floors** | | --- | --- | --- | --- | | **0 Eggs** | 0 | 0 | 0 | | **1 Egg** | 0 | 1 | 2 | | **2 Eggs** | 0 | 1 | ? | | **3 Eggs** | 0 | 1 | ? | | **4 Eggs** | 0 | 1 | ? |  ****Step 2.2: Fill DP Table Using Recurrence**** For dp[i][j], we check all floors f from 1 to j, and take the worst-case minimum:  dp[i][j]=1+min⁡∀f(max⁡(dp[i−1][f−1],dp[i][j−f])) ****Filling for**** dp[2][2]  * Try dropping from **floor 1**:   + If **breaks**, check below: dp[1][0] = 0   + If **survives**, check above: dp[2][1] = 1   + **Max** → max(0,1) + 1 = 2 * Try dropping from **floor 2**:   + If **breaks**, check below: dp[1][1] = 1   + If **survives**, check above: dp[2][0] = 0   + **Max** → max(1,0) + 1 = 2 * **Final Result:** dp[2][2] = min(2,2) = 2  ****Filling for**** dp[3][2]  * Try dropping from **floor 1**:   + If **breaks**, check below: dp[2][0] = 0   + If **survives**, check above: dp[3][1] = 1   + **Max** → max(0,1) + 1 = 2 * Try dropping from **floor 2**:   + If **breaks**, check below: dp[2][1] = 1   + If **survives**, check above: dp[3][0] = 0   + **Max** → max(1,0) + 1 = 2 * **Final Result:** dp[3][2] = min(2,2) = 2  ****Filling for**** dp[4][2]  * Try dropping from **floor 1**:   + If **breaks**, check below: dp[3][0] = 0   + If **survives**, check above: dp[4][1] = 1   + **Max** → max(0,1) + 1 = 2 * Try dropping from **floor 2**:   + If **breaks**, check below: dp[3][1] = 1   + If **survives**, check above: dp[4][0] = 0   + **Max** → max(1,0) + 1 = 2 * **Final Result:** dp[4][2] = min(2,2) = 2  **Final DP Table**  | **dp[i][j]** | **0 Floors** | **1 Floor** | **2 Floors** | | --- | --- | --- | --- | | **0 Eggs** | 0 | 0 | 0 | | **1 Egg** | 0 | 1 | 2 | | **2 Eggs** | 0 | 1 | 2 | | **3 Eggs** | 0 | 1 | 2 | | **4 Eggs** | 0 | 1 | 2 |  **Step 3: Final Answer** dp[4][2] = 2  Thus, the **minimum trials needed** to determine the critical floor with **4 eggs** and **2 floors** is **2**. |
| Output:- 2 | |