Egg Dropping Problem

average-problem
Aliza Saleem Lakhani-al05435
Hafsa Irfan-hi05946
Shayan Ahmed Shariff-ss05583

Github link: https://github.com/CS-412-Spring-2022/p–p-average-problem

CS-412
Habib University

Table of Contents

Introduction

1.1 Problem Statement

You are given n eggs and a building with k floors. Dropping an egg from a floor higher than the threshold floor will break the egg. A drop from a floor equal to or lower than the threshold floor conserves the egg. *Given that an unbroken egg can be dropped from any floor, what is the minimum number of egg drops D needed in order to find the threshold floor in the worst case?*

Constraints:

1. An egg that survives a fall can be used again.
2. A broken egg must be discarded.
3. The effect of a fall is the same for all eggs.
4. If an egg breaks when dropped, then it would break if dropped from a higher floor.
5. If an egg survives a fall then it would survive a shorter fall

We will be comparing three approaches to solve this problem.

- Recursive
- Dynamic Programming
- Binomial Coefficients With Binary Search

1.2 Applications

1. Minimizing cache misses when retrieving data from the cache [1]
2. Performing a large number of queries on a database [1]
3. Avoiding a call to the HDD which is slower [1]

Recursive Approach

2.1 Approach

We can generalize the problem with n eggs and k floors. The solution is to do a trial by dropping an egg at every floor from floor 1 to floor k and recursively calculate the minimum number of trials/drops required in the worst case. The floor that gives the minimum number of trials/drops in the worst case will be part of the solution.
When we drop an egg from a floor x, there are two possibilities:

1. The egg breaks:
   If the egg breaks when dropped from xth floor, then we only need to check for floors below xth floor with leftover eggs since we know that the egg will break for floors above x. Hence, we need to find a floor below floor x where the egg will not break. So our problem reduces to $n-1$ eggs and $k-1$ floors.

2. The egg does not break:
   If the egg breaks when dropped from xth floor, then we only need to check for floors below xth floor with leftover eggs since we know that the egg will break for floors above x. Hence, we need to find a floor below floor x where the egg will not break. So our problem reduces to n eggs and $k-1$ floors.

In the worst case, we need to minimize the number of drops, so we take the maximum of the above two possibilities and then choose the minimum from these worsts from 1 to k to get minimum number of drops. [3]

2.2 Python Code

```python
### Modification to code taken from https://www.geeksforgeeks.
    org/egg-dropping-puzzle-dp-11/ ###

import sys


def eggDropRecursive(eggs, floors):
    ''' Function to get minimum number of trials
        needed in worst case with some eggs and floors '''

    # no floors / one floor
    if (floors == 1 or floors == 0):
        return floors

    # k trials for one egg
    if (eggs == 1):
        return floors

    minValue = sys.maxsize

    # from 1st floor to kth floor
    for x in range(1, floors + 1):

        res = max(eggDropRecursive(eggs - 1, x - 1),
                  eggDropRecursive(eggs, floors - x))
        if (res < minValue):
            minValue = res

    return minValue + 1
```

Listing 2.1: Recursive

2.3 Time Complexity

The recursive approach is time consuming since it checks recursively on all the floors. Hence, this approach solves the overlapping sub-problems many times.
Below is the recurrence relation:

$$T(n,k) = T(n-1,k) + T(n-1,k-1)$$

Each function calls itself twice. Using a recurrence tree, number of immediate children per node is 2, and length of tree is k floors, therefore total time complexity is 2^k.

Dynamic Programming Approach

3.1 Approach

Dynamic programming approach uses the same idea as the recursive approach, but instead of solving the sub-problems again and again, it uses a two-dimension array to store the values. So, if it encounters a sub-problem again, it would only need a look up from the array which will take constant time. Hence, much faster.
We select max from two possibilities of whether egg breaks or does not break, and since we are making a drop on the particular floor too, therefore 1 is added with the above solution. [3]

```
(1 + max( table[i-1][j-1], table[i][j-x] ))
```

3.2 Python Code

```

### Modification to code taken from https://www.geeksforgeeks.
    org/egg-dropping-puzzle-dp-11/ ###

from decimal import FloatOperation
INT_MAX = 32767
def eggDropDynamic(eggs, floors):
    '''
    Function to get minimum number of trials needed in worst
    case with n eggs and k floors with dynamic approach
    '''

    eggFloor = [[0 for x in range(floors + 1)] for x in range(
    eggs + 1)]
    # A 2D table with eggFloor[i][j] representing minimum number of trials

    for i in range(1, eggs + 1):
        eggFloor[i][1] = 1   # 1 trial for 1 floor
        eggFloor[i][0] = 0   # 0 trials for 0 floor

    for j in range(1, floors + 1):
        eggFloor[1][j] = j  # j trials for one egg and j floors.

    for i in range(2, eggs + 1):
        for j in range(2, floors + 1):
            eggFloor[i][j] = INT_MAX
            for x in range(1, j + 1):
                res = 1 + max(eggFloor[i-1][x-1], eggFloor[i][j-x])
                if res < eggFloor[i][j]:
                    eggFloor[i][j] = res

    return eggFloor[eggs][floors]  # eggFloor[n][k] holds the result
```

3.3 Time Complexity

To fill in the values for eggFloor[i][j] in the table for the base-cases, two for loops are used: one with n (eggs) iterations and the other with k (floors) iterations. So, it takes $n + k$ time.
The three nested loops where the first two loops iterate over eggs(n) and floors(k) respectively, and then the third loop again traverses all the floors at max. Hence, it takes $n.k.k$ time.
So, the total time complexity is $n + k + nk^2$. Since nk^2 is the dominant term, asymptotic time complexity for dynamic approach is $O(nk^2)$

Binomial Coefficients And Binary Search

4.1 Approach

The binomial approach is more dependent on number of trials/drops left. So, lets say the first egg broke, now we have only one egg and that leads us to the one-egg solution i.e. to do a trial at each floor.
When we begin with n eggs and d trials/drops, after the first trial [2]:

1. Either we have n eggs if the egg did not break.
2. If the egg breaks, we have $n - 1$ eggs.

Hence, the total number of floors we will be able to cover is

$$f(d, n) = 1 + f(d - 1, n - 1) + f(d - 1, n)$$

Let's define a function f(d, n) that represents the number of floors we can cover using n eggs and no more than d remaining drops in the worst case. We simply have to find a value for d such that:

$$f(d, n) \geq k$$

$$\sum_{n}^{i=1} \binom{d}{i} \geq k$$

Binary search can be used to find value for d as it is way faster than linear search. [1]

4.2 Python Code

```python
### Modification to code taken from https://www.geeksforgeeks.
    org/eggs-dropping-puzzle-binomial-coefficient-and-binary-
    search-solution/ ###

def binomialCoeff(trials, eggs, floors):
    '''
    Calculate the sum of the binomial coefficients
    '''

    ans = 0
    term = 1
    i = 1
    while(i <= eggs and ans < floors):
        term *= trials - i + 1
        term /= i
        ans += term
        i += 1
    return ans


def eggDropBinomial(eggs, floors):
    '''
    Function to get minimum number of trials needed in worst
    case with n eggs and k floors with binomial approach
    '''

    # first floor - f
    # last floor  - l
    f = 1
    l = floors

    # binary search, for each pivot find the
    # sum of binomial coefficient and check if < floors
    while (f < l):

        mid = int((f + l) / 2)

        if (binomialCoeff(mid, eggs, floors) < floors):
            f = mid + 1
        else:
            l = mid

    return int(f)

```

4.3 Time Complexity

Binary search used divides the floors into half each time, we know the time complexity of binary search is $O(logk)$ where $k = floors$. The function binomialCoeff makes n (eggs) iterations to find sum of binomial coefficients. Hence, the total time complexity is $O(nlogk)$.

Comparison

The recursive solution is very slow, and the same function is called more than once due to overlapping sub-problems and so the time complexity is exponential ie $O(2^k)$ where k = total floors. Dynamic Programming improves this time complexity by avoiding recalculation of the same sub-problems by storing the sub-problems in a two-dimensional array. This reduces time complexity to $O(nk^2)$ where n = eggs and k = total floors. This is a great improvement. Binomial with binary search improves this further to $O(nlogk)$. So recursion performs the worst, dynamic programming performs better and binomial with binary search performs the best.

Empirical Analysis

6.1 System Specifications

Processor: Intel Core i5
RAM: 8GB
Language: Python 3.8.0

6.2 Results And Analysis

6.2.1 Comparision Of All Three Algorithms

Experiment Design:

- Eggs kept constant at 2
- Floors Increased
- Average Runtime calculated on 5 runs

Results

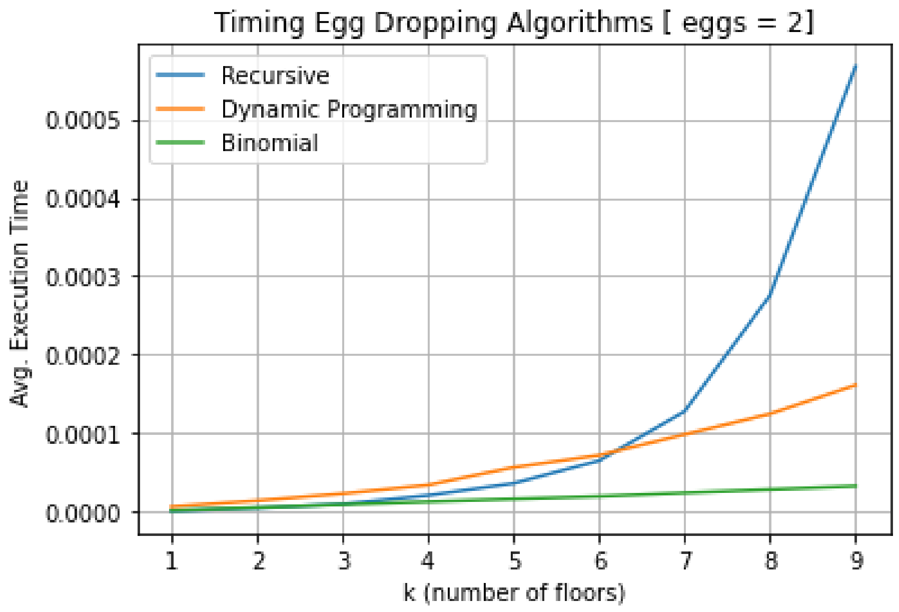

Figure 6.1: All three approaches

In the Figure 6.1, for small values of eggs(n) and floors(k), all the algorithms are almost at par with each other. But as the floors increased, the time of recursion spiked up, with the dynamic approach taking much less time and the binomial approach the least. These agree with the trend observed in theoretical time complexities; where recursive had exponential complexity and binomial coefficients' approach had the least of all three approaches of O(nlogk).

6.2.2 Base-Cases

Experiment Design:

1. Figure 6.2
    - Eggs kept constant at 1
    - Floors Increased
    - Average Runtime calculated on 5 runs
2. Figure 6.3
    - Floors kept constant at 1

- Eggs Increased
- Average Runtime calculated on 5 runs

Results

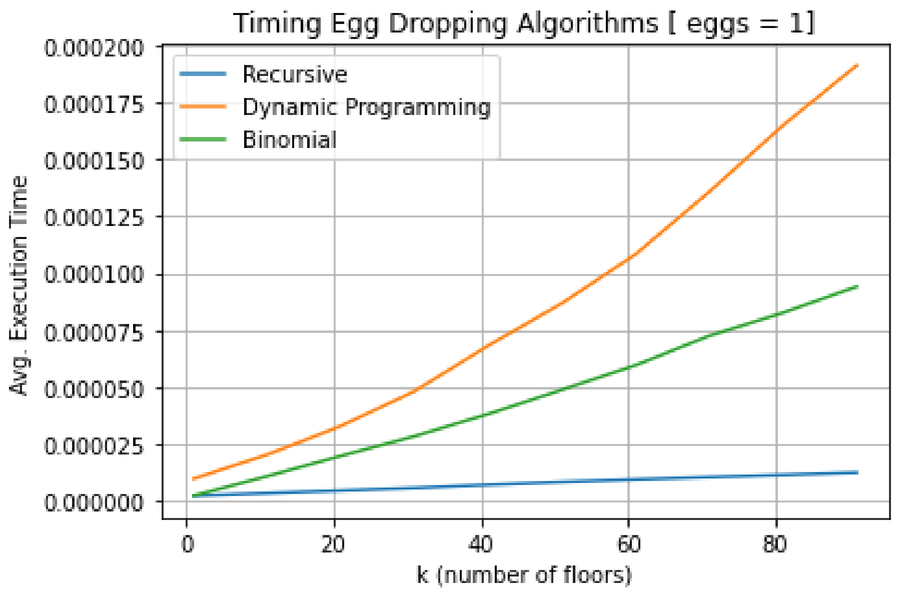

Figure 6.2: Eggs = 1

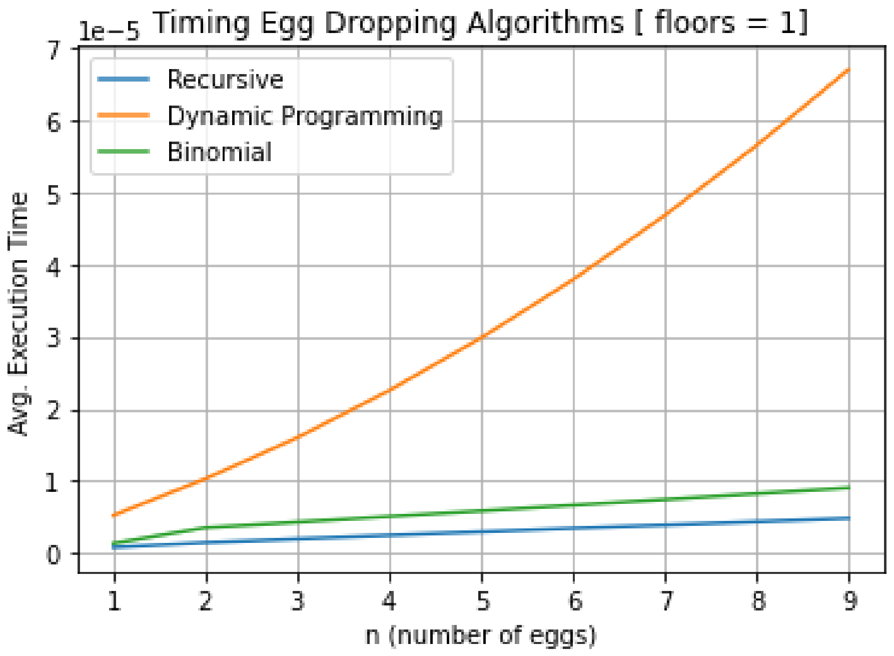

Figure 6.3: Floors = 1

In Figure 6.2, when number of eggs are kept constant at 1, we can't take risk of the egg breaking when dropping the egg from a random floor, because if the egg breaks once, we won't be able to find the threshold floor. So one has to start checking floors from the bottom one by one . In the worst case, number of drops = numbers of floors. Recursion, performs the best. This is because the base cases are checked at the start in the recursion approach so the algorithm returns early. In Figure 2, execution time for dynamic and binomial increases as usual, if we look at a particular k for example k = 20, binomial performs better since input size is getting halved each time,(outer loop runs $logk$ times), and dynamic a bit worse because of it having to solve all sub-problems and so the dominant k^2 term takes over the execution time.

Similar is the case when floors = 1 in Figure 6.3. Recursion performs best due to the base-case checked at the beginning. Binomial, dynamic follow the usual trend where binomial is much faster than dynamic. In dynamic, as eggs increase, even if floor = 1, DP table eggFloor[egg][1] has to be filled for all eggs. And so execution time of dynamic approach increases as number of eggs increase.

6.2.3 Large Input Size

Experiment Design:

1. Figure 6.4
    - Eggs kept constant at 50
    - Floors Increased
    - Average Runtime calculated on 5 runs
2. Figure 6.5
    - Floors kept constant at 50
    - Eggs Increased
    - Average Runtime calculated on 5 runs

Results

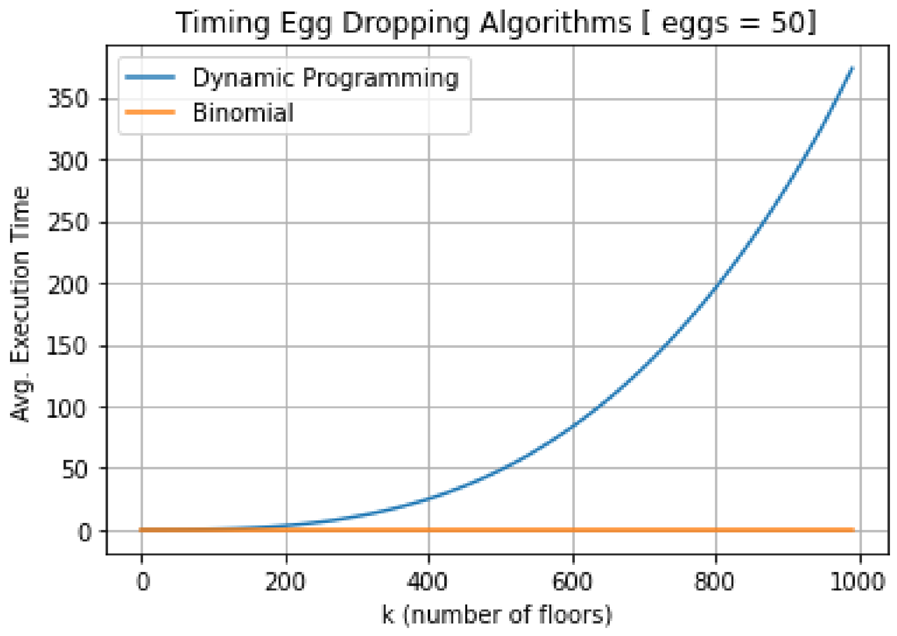

Figure 6.4: Eggs constant, Floors increased

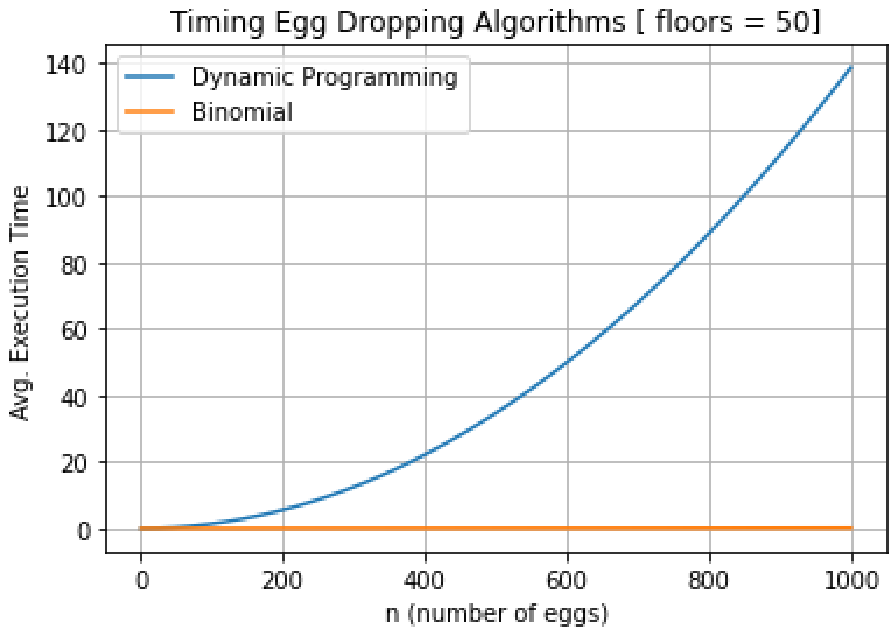

Figure 6.5: Floors = constant, Eggs increased

With much larger values of eggs and floors, the difference between dynamic and binomial is visible as in Figures 6.4 and 6.5. For the binomial with binary search approach, input size is divided with into half each time and so the problem size becomes smaller and smaller, and so we can observe how the execution time is close to zero, very low compared to Dynamic. For dynamic approach as floors or eggs increase, the input size becomes larger and so will have to fill the n * k table for all those values. The graph is a proof as to how dynamic starts performing way worse when eggs/floors increase compared to binomial. Time complexity will seem to appear close to exponential if we represent number as bits for a problem with a building with large number of floors. And so dynamic does not performs will large problem sets.

Conclusion

We've looked at three approaches to solve Egg Dropping Problem; recursion, dynamic programming and binomial coefficients with binary search. The time complexities and their comparison led us to the conclusion that recursion is the worst of all, dynamic programming approach performs much better and binomial with binary search performs the best.

Empirical results, agreed with the trend observed with theoretical time complexities. However, we observed some corner-cases e.g eggs/floors = 1 allow recursion to perform the best because bases-cases are handled at beginning of recursive function with if conditions. Moreover, large input size, ie if we are to solve the problem for a building with large number of floors or if we possess a large number of eggs (less of a possibility), dynamic starts performing really bad and can be represented as exponential time complexity. But, binomial with binary search performs really well, even in those cases. So overall, binomial with binary search seems to perform best for the egg dropping problem.

Bibliography

[1] Alves, G., Pilling, G., Innes, J., Barl, R., Quang, N., Silverman, J., Kau, A. and KhIm, J., n.d. Egg Dropping | Brilliant Math Science Wiki. Brilliant. Retrieved May 1, 2022 from `https://brilliant.org/wiki/egg-dropping/`

[2] Michael Boardman. 2004. The Egg-Drop Numbers. Mathematics Magazine 77, 5 (2004), 368-372. DOI:https://doi.org/10.1080/0025570x.2004.11953281

[3] Egg Dropping Puzzle — DP-11 - GeeksforGeeks. GeeksForGeeks. Retrieved May 1, 2022 from `https://www.geeksforgeeks.org/egg-dropping-puzzle-dp-11/`

[4] Eggs dropping puzzle (Binomial Coefficient and Binary Search . GeeksForGeeks. Retrieved May 1, 2022 from `https://www.geeksforgeeks.org/eggs-dropping-puzzle-binomial-coefficient-and-binary-search-solution/`