Application of Dynamic Programming to Solve Egg Dropping Puzzle

Girvin Junod - 13519096
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519096@std.stei.itb.ac.id

***Abstract*—The Egg Dropping Puzzle is a puzzle that involves eggs and a building with multiple floors. The puzzle asks how many acts of egg dropping is required to get a guaranteed answer on which floors on the building are safe to drop an egg from. This puzzle can be solved using dynamic programming. Dynamic programming is a method to solve problems by breaking down the solution to the problem into a group of stages so that the solution to the problem is a series of interconnected decisions to the sub-problems. Dynamic programming is as a method usually used to solve optimization problems. This paper discusses the application of dynamic programming to solve the egg dropping puzzle.**

Keywords—dynamic programming; egg dropping puzzle; puzzle

I. INTRODUCTION

There are many types of puzzles. One type of puzzles is known as logic puzzles. They are called logic puzzles because they are derived from mathematical deductions. This means that the solution to the puzzle requires using deductive reasoning to reach a logical conclusion. The egg dropping puzzle is an example of these logic puzzles.

Fig. 1. Illustration of the egg dropping puzzle (medium.com/@parv51199/egg-drop-problem-using-dynamic-programming-e22f67a1a7c3, accessed on May 10, 2021)

The egg dropping puzzle is a logical problem involving E amount of eggs and a building with F number of floors. Suppose that we wish to know which story in the F-story building that are safe to drop an egg from, meaning that if the egg is dropped from that story, the egg wouldn't break when it lands on the ground. The effect of the fall is the same for all eggs. It's important to note that if an egg breaks after being dropped from a certain floor, then the egg will also break if dropped from floors higher than that floor. Vice versa, if an egg doesn't break after being dropped from a certain floor, then the egg will also not break if dropped from floors lower than that floor. A broken egg can't be used in another trial, but an egg that survives a fall can be reused in another trial. The question is, with E amount of eggs, what is the minimum number of egg droppings that is needed to guarantee which floors are safe to drop eggs from. The question asks for a guaranteed answer meaning that luck and probability are not part of the answer and whether the egg breaks or not when dropped ultimately doesn't matter for the solution.

The solution to the egg dropping puzzle can be found in many ways. One can simply use an algorithm to go through every single possible iteration of the trials and then pick the one with the lowest amount of egg drops. But this is inefficient as the number of iterations increases massively with every additional floor. The egg dropping puzzle is an optimization problem, which means that optimization algorithms such as the greedy algorithm can be used to reach the solution. But the greedy algorithm is not reliable in reaching the optimal solution. Dynamic programming in the other hand can reach the optimal solution in an optimization problem like the egg dropping puzzle much more reliably as it checks for more than one possible solution.

II. THEORETICAL FRAMEWORK

A. Recursion

Recursion in computer science occurs when a function calls upon itself during its execution. In other words, recursion is when the function being defined is applied in its own definition so that the function will call upon itself during the execution of the function.

A function calling upon itself will create a loop. In order to stop this loop, in every recursive function exist something called a base case, a terminating scenario that does not use recursion to produce an answer. The result of that base case will then be used to produce results for cases that's not the base through recursion. A recursive case is a scenario that uses recursion to produce results, meaning that it calls upon function being defined so that

a recursion occurs. A recursive function is made up of base cases and recursive cases.

Recursion is used in many ways in computer science. It's used to create algorithms like depth-first search algorithms and algorithms using dynamic programming. Recursion is useful for breaking down complex problems into smaller simpler sub-problems. However, recursion is generally not that efficient when used without optimization methods. This is because recursion can result in recalculating sub-problems that has been solved and this incurs extra computation time that makes the function inefficient.

B. Dynamic Programming

Dynamic programming is a method in mathematics and computer science used to find and optimize solutions. Dynamic programming refers to simplifying a complex problem into simpler sub-problems in a recursive manner. Dynamic programming breaks down the solution to the problem into a group of stages so that the solution to the problem can be seen as a series of interconnected decisions or solutions to the sub-problems. It's similar to the greedy algorithm as it also sees the solution as a series of interconnected decisions. The difference between greedy algorithms and dynamic programming is that in dynamic programming, more than one series of decisions are considered for the final solution.

As dynamic programming is an optimization method, the series of interconnected decisions that we want to get is the one that leads to the optimal solution. In order to get the optimal solution, the principle of optimality is used. The principle of optimality or Bellman's principle of optimality states that if the final solution is optimal, then parts of the solution leading to the final solution is also optimal. This show that in order for dynamic programming to get an optimal solution, the problem needs to have an optimal substructure which means that the optimal solution can be constructed from the optimal solution of its sub-problems.

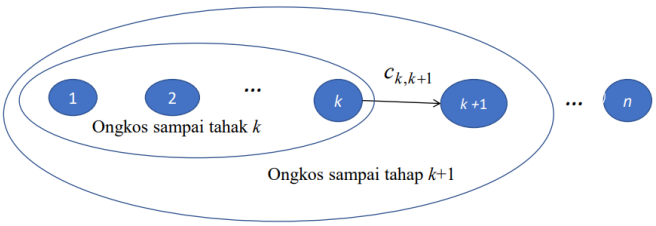

Fig. 2. Illustration of the principle of optimality (informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf, accessed on May 10, 2021)

On figure 2, it can be seen that the cost for stage k+1 is the cost for stage k + cost from stage k to k+1. This means that to get an optimal solution for stage k+1, the optimal solution for stage k is also needed. This means that if we're working to find the optimal solution for stage k+1 from stage k, we can simply use the optimal solution for stage k without redoing the calculations. This is an example of the principle of optimality.

There are certain characteristics that show that a certain problem can be solved with dynamic programming. By fulfilling all seven of the characteristics, a problem can then be solved using dynamic programming.

The first characteristic is that the problem can be divided into several stages and every stage can be solved with a single decision.

The second characteristic is that every stage consists of several states that is related to the stage. Usually the states of a stage are all the possible inputs for that stage.

The third characteristic is that for every decision taken for every state in a stage, the decision is then transformed to be used by the next state in the next stage.

The fourth characteristic is that the cost of a stage increases steadily with every additional stage. This is with the assumption that every cost is positive and so with every additional stage, the cost will only increase.

The fifth characteristic is that the cost of a stage is dependent on the cost of previous stages and the cost from that stage to the next one.

The sixth characteristic is that identifies the optimal solution for every state in a stage so that the optimal solution for every state in the next stage can be identified.

The seventh characteristic is that the principle of optimality can be applied to the problem.

There are two approaches in solving problems through dynamic programming. The first one is the top-down or forward approach. In the top-down approach, the calculation starts from the first stage and then continues until it reaches the end. The second one is the bottom-up or backward approach. In the bottom-up approach, the calculation starts from the last stage and then proceeds backward until the first stage.

The general steps to develop an algorithm using dynamic programming are usually the same for every algorithm. The first of these general steps is to characterize the structure of the optimal solution. This means to characterize the stages, states, etc. The second step is to define recursively the optimal solution to a stage by connecting the optimal solution of the previous stage to the current one. The third step is to calculate the final optimal solution using the forward or backward approach. In order to not recalculate all the optimal solution for all stages and states, the optimal solution for all stages and states are recorded in a table so that further use of that optimal solution can simply just use the value from the table so that it doesn't need to recalculate. The fourth step is an optional one which is to reconstruct the final optimal solution so that it can be shown clearly all the decisions taken in every stage.

C. Egg Dropping Puzzle

Egg dropping puzzle involves E amount of eggs and a F-story building and it asks what is the minimum amount of egg dropping needed to guarantee which floors in the building are safe to drop an egg from. There some rules to the puzzle. The rules are:

1. The effect of a fall is the same for all eggs.

2. An egg that breaks from a fall can't be reused in another trial.
3. An egg that survives a fall can be reused in another trial.
4. If an egg breaks from a fall from a certain floor, the egg will also break if dropped from a higher floor.
5. If an egg doesn't break from a fall from a certain floor, the egg will also not break if dropped from a lower floor.

The question of the puzzle can be reworded to finding out the least amount of egg drops needed to determine the threshold floor, namely the floor from which the egg breaks if dropped. This means that the egg will break if dropped from the threshold floor or higher and will not break if dropped from floors lower than the threshold floor. By finding out the threshold floor, the floors that are safe to drop an egg from will also be known, as it's all the floors below the threshold floor. It's also important to note that the focus of the question is the minimum number of steps needed to guarantee which floor is the threshold floor, not determining which floor is the threshold floor. Because of that, the solution to the problem will not be affected by factors such as the durability of the egg or the height of the building as it's purely a matter of logical reasoning.

III. Implementation of Dynamic Programming to Solve The Egg Dropping Puzzle

In order to apply dynamic programming in solving the egg dropping puzzle, first we must characterize the optimal structure of the solution. For this example, we'll go with the forward or top-down approach. Then, we can identify that the stage for the dynamic programming of the egg dropping puzzle is the process of dropping an egg from a floor. We can also identify the states of the stages to be the number of eggs.

After identifying the structure of the optimal solution, we must identify the base cases and the recursive cases for the egg dropping puzzle so that it can be solved recursively. In other words, we must identify the recursive relation for the optimal solution.

In order to find the base case, first we must look at the case with only 0 or 1 number of floors. If the building has 0 floors, that means we don't need to do any trials at all so we get the answer 0. If the building has 1 floor, then we only need to test for the first floor meaning that the answer is 1.

The other base case is for every case with only 1 egg. With only 1 egg, the worst-case scenario will always be that we need to test for every single floor in order to find the threshold floor as it's possible that the egg doesn't break when dropped from the highest floor. As the puzzle asks for the answer that works for every single case, that means we will always take the worst-case scenario as the answer. This means that for every case that has 1 egg and F number of floors, the answer will always be F.

In order to find the recursive case, we must take a look at what happens if we drop an egg on a random X floor. There are two possibilities, the first one is that the egg breaks, and the second one is that egg doesn't break. In the case where the egg breaks, this means that the egg will also break for every single floor higher than X. This means that all the floors above X cannot be the threshold floor as the egg already breaks when dropped from X. So, we only need to check for the floors lower than X to see if X or the floors lower than it is the threshold floor. It's also important to note that the number of eggs decreases by one as one egg is already broken from the first fall.

In the case where the egg doesn't break, that means that X and all the floors lower than it can't be the threshold floor. Therefore, we only need to check for the floors higher than X. As the egg doesn't break, this means that the egg can be reused in further trials and the number of eggs doesn't decrease.

As we are looking for the answer for all cases, that means we have to look for the answer for worst-case scenario. This means that out of the answers from the two possibilities, we have to take the maximum one. This is done for every single possible value of X which is in the range 1 to F. Of all the possible answer for the worst-case scenario for every X, we want the answer with the minimum value as the puzzles asks for the least amount of trials. As we are doing an egg drop for every time we check whether an egg breaks or not when dropped from X, we add 1 to the answer every time we enter the recursive case.

$$eggDrop(e, f) = \begin{cases} 0 & \text{if } f = 0 \\ 1 & \text{if } f = 1 \\ f & \text{if } e = 1 \\ 1 + min(max(eggDrop(e-1, f-1), eggDrop(e, f-x)), x \; in \; 1:f) & \text{else} \end{cases}$$

Fig. 3. Recursive function for the egg dropping puzzle

The recursive function for the egg dropping puzzle can be found on Figure 3. The function eggDrop(e,f) has 3 base cases and 1 recursive case. The 3 base cases are for the cases where the number of floors is 0 or 1 and where the number of eggs is 1. Normally, we can simply solve the problem by using this function in a recursive method, but there will be repetitive function calls or the same function being recalculated multiple times. This can be solved by solving the problem through dynamic programming which is why the solving method is more than just the recursive function.

Now we can start calculating the optimal solution with dynamic programming. For this example, we shall calculate the value of eggDrop(3, 6) or the egg dropping puzzle with 3 eggs and 8 floors. We can see the structure of the optimal solution through the recursive function. We shall label the states as s and the stages as k with the value of s = 2, 3 and k = 2, 3, 4, 5, 6. There is no s = 1, k = 0, and k= 1 because they are all covered by the base cases. First, we shall look at the base cases of the recursion. From the base cases, we get the following tables.

s	eggDrop(s, 0)
1	0
2	0
3	0

Fig. 4. Table for the case with 0 floor

s	eggDrop(s,1)
1	1
2	1
3	1

Fig. 5. Table for the case with 1 floor

k	eggDrop(1, k)
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

Fig. 6. Table for the case with 1 egg

After covering all the base cases, we can start calculating the solution for stage k = 2. After covering all the base cases, we can start calculating the solution for stage k = 2. The calculation for eggDrop(s,k) will be by the recursive base as shown in Figure 3 with x = 1, 2.

s	max(eggDrop(s-1, x-1), eggDrop(s, k-x))		min	eggDrop(s,2)
	x			
	1	2		
2	1	1	1	2
3	1	1	1	2

Fig. 7. Table for stage k = 2

As we can see, in calculating for the optimal solution for stage k = 2, we're using the optimal solution that was calculated in the base cases. For example, the value of eggDrop(2,1) that was used in Figure 7 is already calculated in Figure 5, therefore no recalculation is done as we can simply take the value from previous tables. This is the difference between solving the problem with dynamic programming and solving the puzzle with basic recursion. If we simply solve the problem with a normal recursive function then a lot of recalculation will be done and the algorithm will not be that efficient. The following tables are the solution for k = 3, 4, 5, 6.

s	max(eggDrop(s-1, x-1), eggDrop(s, k-x))			min	eggDrop(s,3)
	x				
	1	2	3		
2	2	1	2	1	2
3	2	1	2	1	2

Fig. 8. Table for stage k = 3

s	max(eggDrop(s-1, x-1), eggDrop(s, k-x))				min	eggDrop(s,4)
	x					
	1	2	3	4		
2	2	2	2	3	2	3
3	2	2	2	2	2	3

Fig. 9. Table for stage k = 4

s	max(eggDrop(s-1, x-1), eggDrop(s, k-x))					min	eggDrop(s,5)
	x						
	1	2	3	4	5		
2	3	2	2	3	4	2	3
3	3	2	2	2	3	2	3

Fig. 10. Table for stage k = 5

s	max(eggDrop(s-1, x-1), eggDrop(s, k-x))						min	eggDrop(s,6)
	X							
	1	2	3	4	5	6		
2	3	3	2	3	4	5	2	3
3	3	3	2	2	3	3	2	3

Fig. 11. Table for stage k = 6

As we can see in Figure 11, we get the value of eggDrop(3,6) as 3. This means for the egg dropping puzzle with 3 eggs and 6 floors, the optimal solution is 3 moves. In other words, the threshold floor of the building can be determined with 3 moves or less. As the puzzle is solved using dynamic programming, then all the optimal solutions for every combination of amount of eggs and floors that was calculated before eggDrop(3,6) is also stored inside the tables. Although some of those answers are not used in calculating eggDrop(3,6), they may be used if we ever want to extend the stages to solve the egg dropping puzzle with a different number of eggs and floors. Ultimately, using dynamic programming allows us to not repeat calculations that have been done so that the algorithm is more efficient and so more time will be saved.

The solution to the egg dropping puzzle using dynamic programming can be confirmed by solving the problem using logical deduction. With 6 floors, pick the 3rd floor to drop an egg from. If the egg breaks then the threshold floor is between the 1st and the 3rd floor. The worst-case scenario would then be needing to check both the 1st floor and the 2nd floor with the egg not

breaking when dropped on the 1st floor therefore needing a third try. So, the minimum number of trials for that scenario is 3. If the egg doesn't break when dropped from the 3rd floor, then the threshold floor is between the 4th floor and the 6th floor. Drop the egg on the 5th, if it breaks then check the 4th floor, if it doesn't break then check the 6th floor. So, the minimum number of trials in this scenario is also 3. It's impossible to get a solution lower than 3 for the egg dropping puzzle with 3 eggs and 6 floors so we can say the using dynamic programming we have successfully found the optimal solution for the egg dropping puzzle. Of course, the method above can also be used to solve the egg dropping puzzle with a different amount of eggs and floors. Therefore, we can say that dynamic programming can be used to solve the egg dropping puzzle effectively.

IV. Implementation of Solver using Python

The following is the implementation of dynamic programming to solve the egg dropping puzzle using the programming language Python. The solver program using Python is created following the same steps that can be seen on the previous chapter.

First, we need to prepare the table of optimal solution for every single combination of eggs and floors. This allows the program to simply reuse the optimal solutions of previous stages without needing to recalculate anything. Then, we need to fill the table with the base cases. The base cases are for when the number of eggs is equal to 1 and when the number of floors is equal to 0 or 1. This is achieved through the Python function in Figure 12.

```
def makeEggDropArray(e, f):
    arrEggDrop = [[-1 for j in range(f+1)] for i in range(e)]
    for i in range(f+1):
        arrEggDrop[0][i] = i
    for i in range(len(arrEggDrop)):
        arrEggDrop[i][0] = 0
        arrEggDrop[i][1] = 1
    return arrEggDrop
```

Fig. 12. Python function to create the table of optimal solutions and fill it with the base cases

As we can see in Figure 12, the Python function makeEggDropArray(e,f) receives the variable e and f with e being the number of eggs and f being the number of floors. A matrix is created with the initial values of -1 so that we can tell that an index of matrix has not been filled yet. Then we fill the matrix with the values got from the base cases as shown in Figure 3.

After we create the table of optimal solutions, then we can start calculating the optimal solution through dynamic programming. For this example, we are using the forward or top-down approach. The python function to solve the egg dropping puzzle can be seen in Figure 13.

```
def eggDrop(e, f, arrEggDrop):
    if (arrEggDrop[e-1][f] != -1):
        return arrEggDrop[e-1][f]
    res = max(eggDrop(e-1,0, arrEggDrop), eggDrop(e, f-1, arrEggDrop))
    min = res
    for i in range(2, f+1):
        res = max(eggDrop(e-1,i-1, arrEggDrop), eggDrop(e, f-i, arrEggDrop))
        if (res < min):
            min = res
    answer = 1 + min
    arrEggDrop[e-1][f] = answer
    return answer

def solveEggDrop(e,f):
    arrEggDrop = makeEggDropArray(e,f)
    return eggDrop(e,f,arrEggDrop)
```

Fig. 13. Python function to solve the egg dropping puzzle

The Python function eggDrop(e,f,arrEggDrop) receives the variable e, f, and arrEggDrop with e being the number of eggs, f being the number of floors, and arrEggDrop being the table of optimal solution. First, the function will check whether the solution is already in the table of optimal solutions or not. If the solution already exists in the table, then the function will simply return the solution that is in the table so that it doesn't need to recalculate. If the solution doesn't exist yet, then it will calculate the optimal solution through the recursive base shown in Figure 3. After finding the optimal solution, then the function will store the answer in the table of optimal solutions and then returning the answer. The eggDrop function is a recursive function that will only calculate the optimal solution if the optimal solution hasn't already been calculated, reducing the number of times needed to recalculate solutions that has already been calculated. The Python function solveEggDrop(e,f) is a function made to combine the makeEggDropArray function and the eggDrop function so that we only need to call the solveEggDrop function. In order to use this Python function to solve the egg dropping puzzle, a simple main program is made with command-line input. This main program is shown in Figure 14.

```
#main
e = int(input("Enter the amount of eggs: "))
f = int(input("Enter the number of floors: "))

ans = "The minimum amount of trials for "
ans += str(e)
ans += " number of eggs and "
ans += str(f)
ans += " number of floors is "
ans += str(solveEggDrop(e,f))
ans += " trials"
print(ans)
```

Fig. 14. Main program for egg dropping puzzle solver with Python

The main program for the solver is a program that receives a command-line input for the amount of eggs and floors. Then, it will call upon the solveEggDrop(e,f) function with e and f being the number of eggs dan floors that have been inputted through the command-line. After getting the optimal solution through the solveEggDrop function, the main program then prints out the solution in a sentence to the command-line.

The following are examples of the output of the main program of the egg dropping puzzle solver in Python as shown in Figure 15 and Figure 16.

```
PS E:\Coding\stima> python eggDrop.py
Enter the amount of eggs: 3
Enter the number of floors: 6
The minimum amount of trials for 3 number of eggs and 6 number of floors is 3 trials
PS E:\Coding\stima> python eggDrop.py
Enter the amount of eggs: 2
Enter the number of floors: 6
The minimum amount of trials for 2 number of eggs and 6 number of floors is 3 trials
PS E:\Coding\stima> python eggDrop.py
Enter the amount of eggs: 2
Enter the number of floors: 3
The minimum amount of trials for 2 number of eggs and 3 number of floors is 2 trials
```

Fig. 15. Output examples of the main program

As shown Figure 15, the output of the solver for the egg dropping puzzle with 3 eggs and 6 floors is 3 trials. This is the same result that we got in the previous chapter as shown in Figure 11. Furthermore, we can also see that the results that we got for the egg dropping problem for 2 eggs with 6 floors and 2 eggs with 3 floors are 3 trials and 2 trials. We can check this with the result that we got in Figure 8 and Figure 11. Therefore, we can safely say that the solver has successfully implemented the algorithm to solve the egg dropping puzzle with dynamic programming.

```
PS E:\Coding\stima> python eggDrop.py
Enter the amount of eggs: 2
Enter the number of floors: 8
The minimum amount of trials for 2 number of eggs and 8 number of floors is 4 trials
PS E:\Coding\stima> python eggDrop.py
Enter the amount of eggs: 2
Enter the number of floors: 36
The minimum amount of trials for 2 number of eggs and 36 number of floors is 8 trials
PS E:\Coding\stima> python eggDrop.py
Enter the amount of eggs: 2
Enter the number of floors: 100
The minimum amount of trials for 2 number of eggs and 100 number of floors is 14 trials
PS E:\Coding\stima> python eggDrop.py
Enter the amount of eggs: 3
Enter the number of floors: 100
The minimum amount of trials for 3 number of eggs and 100 number of floors is 9 trials
PS E:\Coding\stima> python eggDrop.py
Enter the amount of eggs: 2
Enter the number of floors: 500
The minimum amount of trials for 2 number of eggs and 500 number of floors is 32 trials
```

Fig. 16. More output examples of the main program

In Figure 16, we can see that the solver also works well for cases with a huge number of floors with numbers reaching 500 in the test case.

V. Conclusion

In conclusion, dynamic programming is an effective method to solve optimization problems. Not only that, dynamic programming is also an efficient method compared to other similar methods as it prevents recalculation of previously solved problems to reduce processing time. One of the applications of dynamic programming as seen in this paper is to solve logic puzzles such as the egg dropping puzzle. As shown in this paper, dynamic programming can be used as a method to find the minimum amount of egg droppings needed to guarantee which floors in a building are safe for an egg to be dropped from with certain amount of eggs and floors in a building. Therefore, the egg dropping puzzle can be solved using dynamic programming with the methods shown in this paper.

Dynamic programming has many uses beyond solving logic puzzles. There are many algorithms in the world today that uses dynamic programming. The author hopes that in the future more people will use methods such as dynamic programming to create more effective and efficient algorithms to solve all kinds of problems.

Video Link at Youtube

The following is a link to a Youtube video the author made to further explain solving the egg dropping puzzle using dynamic programming: https://youtu.be/EbQFTXGl1NU

Acknowledgment

The author would like to thank all the lecturers for the IF2211 Strategi Algoritma class who have taught and provided the author with the knowledge in algorithm strategies especially dynamic programming needed to write this paper.

References

[1] Munir, Rinaldi, informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf, accessed on May 10, 2021

[2] Munir, Rinaldi, informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian2.pdf, accessed on May 10, 2021

[3] Parv, Parikh, Egg Drop Problem Using Dynamic Programming, medium.com/@parv51199/egg-drop-problem-using-dynamic-programming-e22f67a1a7c3, accessed on May 10, 2021

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2021

Girvin Junod 13519096