Theory Assignment-2: ADA Winter-2024

Ritika Thakur (2022408) Sidhartha Garg(202499)

1 Subproblem definition.

Let's define the subproblem in terms of the function ans(index, array, repRing, repDing, dpArr): The problem is broken down into further subproblems which are so that, each subproblem is the max chickens that the fox can get till that index, taking care of not using "Ring" or "Ding" consecutively more than 3 times in a row.

2 Recurrence of the subproblem.

$$\text{ans}(N, R, D) = \max \begin{cases} \text{ans}(N-1, 0, D+1) - \text{array}[i] & \text{if } Ring >= 3 \\ \text{ans}(N-1, R+1, 0) + \text{array}[i] & \text{if } Ding >= 3 \\ \text{ans}(N-1, R+1, 0) + \text{array}[i] & \text{continuing with "Ring"} \\ \text{ans}(N-1, 0, D+1) - \text{array}[i] & \text{continuing with "Ding"} \end{cases}$$

3 The specific subproblem(s) that solves the actual problem.

The specific subproblem being solved is finding the combination of the number of consecutive "Ring" repetitions and the number of consecutive "Ding" repetitions so the chickens are the maximum at the current index.

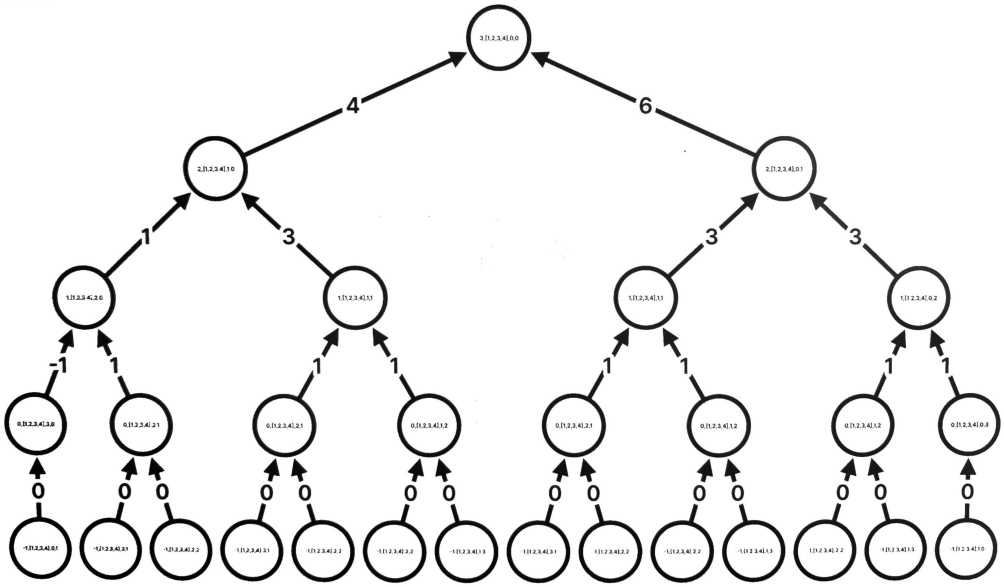

Figure 1: Recursion Tree(created using : https://recursion.vercel.app)

4 Algorithm description.

The algorithm employs dynamic programming with a three-dimensional array to efficiently solve the problem. The three dimensions of the array correspond to the current index in the input array, the count of consecutive

"Ring" repetitions ('repRing'), and the count of consecutive "Ding" repetitions ('repDing'). This allows the algorithm to store and retrieve previously computed results, facilitating memoization and avoiding redundant calculations.

The `ans` function, the core of the algorithm, takes the following inputs:

- `index`: The current index being processed in the input array.
- `array`: The input array containing the values representing the rewards or penalties at each index.
- `repRing`: The count of consecutive "Ring" repetitions encountered so far.
- `repDing`: The count of consecutive "Ding" repetitions encountered so far.
- `dpArr`: The three-dimensional array used for memoization, storing previously computed results.

The function computes the maximum value achievable by exploring various choices at each index of the input array. It handles termination when the index becomes negative, effectively handling the base case for index = 0. Recursively, it considers different scenarios, including cases where the count of consecutive "Ring" or "Ding" repetitions exceeds three, and explores options to continue or switch between the sequences. The top-down approach starts the computation from the last index of the input array and works its way backward.

The `main` function serves as the entry point of the program. It first reads the input size `n` and the elements of the array. Then, it initializes the `dpArr` with a value of INT_MIN. Afterward, it calls the `ans` function to compute the maximum value achievable and outputs the result.

5 Pseudocode

Algorithm 1 ans function

function ANS($index$, array, $repRing$, $repDing$, dpArr)
  if $index < 0$ **then**
    return 0
  end if
  if dpArr[$index$][$repRing$][$repDing$] $\neq INT\_MIN$ **then**
    return dpArr[$index$][$repRing$][$repDing$]
  end if
  if $repRing \geq 3$ **then**
    return ans($index - 1$, array, 0, $repDing + 1$, dpArr) $-$ array[$index$]
  else if $repDing \geq 3$ **then**
    return ans($index - 1$, array, $repRing + 1$, 0, dpArr) $+$ array[$index$]
  end if
  $Ring \leftarrow$ ans($index - 1$, array, $repRing + 1$, 0, dpArr) $+$ array[$index$]
  $Ding \leftarrow$ ans($index - 1$, array, 0, $repDing + 1$, dpArr) $-$ array[$index$]
  return dpArr[$index$][$repRing$][$repDing$] $\leftarrow \max(Ring, Ding)$
end function
function MAIN
  Read input size n
  Initialize array $array$ of size n with -1
  Initialize $dpArr$ as a 3D array of size $(n + 1) \times 4 \times 4$ with all elements set to $1e5$
  for i from 0 to $n - 1$ **do**
    Read $array[i]$
  end for
  Output ANS($n - 1, array, 0, 0, dpArr$)
end function

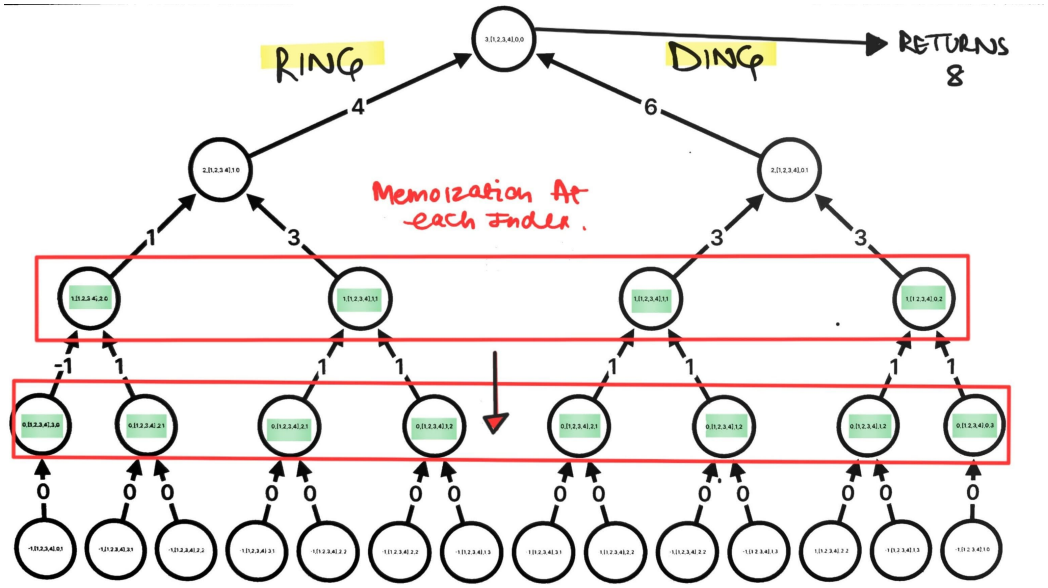

Figure 2: With Memoization(annotated by hand)

6 Complexity Analysis

Without Memoization:

In the case without memoization, the function makes redundant recursive calls, resulting in exponential time complexity. For each call, it will check 2 options that is either to "Ring" or "Ding", leading to further branching resulting in exponential time complexity. In the worst case, the function explores all possible combinations, leading to a time complexity of $O(2^n)$, where n is the value of index.

With Memoization:

With memoization, the function stores and reuses the results of previous computations, as it knows which combination of "Ring" and "Ding" results in max chickens. This significantly optimizes the time complexity.

If n is the value of index. Since each recursive call is memoized and only computed once, the time complexity becomes $O(n)$ for the entire recursion tree. This is because each subproblem is solved only once and the max is stored for that index.

In the first case, the space complexity is $O(n)$ due to the recursion depth and in memoization the space required for the memoization array is also included so the space complexity is $O(16n + n) = O(17n)$.