Theory Assignment-2: ADA Winter-2024

Aditya Kumar (2022033) — Nishant Kumar (2022326)

February 11, 2024

Assumptions:

In the pseudo code, the array has index 0 to $n - 1$.

Pseudo Code:

```
function maxChickens(A):

    1. n = length of A
    //create a 2D int array chicken with dimensions n x 6 (memoization)
    2. int[][] chicken = new int[n][6]

    3. // Handle the base case when i is 0
        chicken[0][0] = A[0];
        chicken[0][1] = A[0];
        chicken[0][2] = A[0];
        chicken[0][3] = -A[0];
        chicken[0][4] = -A[0];
        chicken[0][5] = -A[0];

    4. for i from 1 to n - 1:
        // Update chicken[i][0] (consecutive Ring = 1)
        5. chicken[i][0] = A[i] + max(getChicken(chicken, i-1, 3),
        getChicken(chicken, i-1, 4), getChicken(chicken, i-1, 5)))
        // Update chicken[i][1] (consecutive Ring = 2)
        6. chicken[i][1] = A[i] + chicken[i-1][0]
        // Update chicken[i][2] (consecutive Ring = 3)
        7. chicken[i][2] = A[i]+ chicken[i-1][1]

        // Update chicken[i][3] (consecutive Ding = 1)
        8. chicken[i][3] = -A[i] + max(getChicken(chicken, i-1, 0),
        getChicken(chicken, i-1, 1), getChicken(chicken, i-1, 2)))
        // Update chicken[i][4] (consecutive Ding = 2)
        9. chicken[i][4] = -A[i] + chicken[i-1][3]
        // Update chicken[i][5] (consecutive Ding = 3)
        10. chicken[i][5] = -A[i]+ chicken[i-1][4]

    // Find the maximum chickens earned from all the cases in the last row
    11. maxChickens = MIN_VALUE
    12. for j from 0 to 5:
        13. maxChickens = max(maxChickens, chicken[n - 1][j])

    14. return maxChickens

//helper function to avoid index out of bound
15. function getChicken(chicken, i, j):
    16. if i < 0 or j < 0 or j >= columns of chicken:
        17. return 0
    18. return chicken[i][j]
```

Algorithm 1 Maximize Chickens :

```
function MAXIMIZECHICKENS($A$)
    $n$ = length of $A$
    Create a 2D array $chicken$ with dimensions $n \times 6$
    Initialise Chicken[0], base case
    for $i$ from 1 to $n$ do
        $chicken[i][1] = A[i] + \max(chicken[i-1][4], \max(chicken[i-1][5], chicken[i-1][6]))$
        $chicken[i][2] = A[i] + chicken[i-1][1]$
        $chicken[i][3] = A[i] + chicken[i-1][2]$
        $chicken[i][4] = A[i] + \max(chicken[i-1][1], \max(chicken[i-1][2], chicken[i-1][3]))$
        $chicken[i][5] = A[i] + chicken[i-1][4]$
        $chicken[i][6] = A[i] + chicken[i-1][5]$
    end for
    for $j$ from 1 to 6 do
        $maxChickens = \max(maxChickens, chicken[n][j])$
    end for
    return $maxChickens$
end function
```

1. Subproblem definition

- Given an array A representing the number of chickens at each booth, where the number of booths is n, the subproblem involves calculating all possible cases of reward and penalty for Mr. Fox at the i-th booth while adhering to the given rules. This involves considering different combinations of

"RING" and "DING" choices at each booth up to the i-th booth, ensuring that Mr. Fox does not violate any rule.

- The state of the subproblem can be defined by the parameters i and j, where:
  - i represents the current booth being considered (ranging from 0 to $n-1$).
  - j represents the number of consecutive "RING" or "DING" choices made by Mr. Fox up to the i-th booth. The objective is to compute the maximum number of chickens earned up to the i-th booth with j consecutive "RING" or "DING" choices.

2. Recurrence of the subproblem

At each booth i, there are six possible states for Mr. Fox, corresponding to the number of consecutive "RING" or "DING" choices made up to that point. These states are represented by the columns of the chicken array. For each booth i (starting from the second booth since the base case is handled separately), the maximum number of chickens earned depends on the choices made at the previous booth $i-1$ and the current booth i. The recurrence relation for calculating $chicken[i][j]$ is as follows:

- For $j = 0$ (consecutive "RING" choices = 1):
  - Mr. Fox can either choose "RING" at the i-th booth, adding $A[i]$ chickens to the total, or he can transition from a consecutive "DING" state at the previous booth $(i-1)$ to the current "RING" state at booth i.
  - Therefore, $chicken[i][0]$ is calculated as $A[i]+\max(getChicken(chicken, i-1, 3), \max(getChicken(chicken, i-1, 4), getChicken(chicken, i-1, 5)))$.
- For $j = 1$ (consecutive "RING" choices = 2) and $j = 2$ (consecutive "RING" choices = 3):
  - Mr. Fox can choose "RING" at the i-th booth, adding $A[i]$ chickens to the total, or he can transition from a consecutive "RING" state at the previous booth $(i-1)$ to the current "RING" state at booth i.
  - Therefore, $chicken[i][1]$ is calculated as $A[i] + chicken[i-1][0]$, and $chicken[i][2]$ is calculated as $A[i] + chicken[i-1][1]$.
- For $j = 3$ (consecutive "DING" choices = 1):
  - Mr. Fox can either choose "DING" at the i-th booth, adding $-A[i]$ chickens to the total, or he can transition from a consecutive "RING" state at the previous booth $(i-1)$ to the current "DING" state at booth i.

  - Therefore, $chicken[i][3]$ is calculated as $-A[i]+\max(getChicken(chicken, i-1, 0), \max(getChicken(chicken, i-1, 1), getChicken(chicken, i-1, 2)))$.

- For $j = 4$ (consecutive "DING" choices = 2) and $j = 5$ (consecutive "DING" choices = 3):
  - Mr. Fox can choose "DING" at the i-th booth, adding $-A[i]$ chickens to the total, or he can transition from a consecutive "DING" state at the previous booth $(i-1)$ to the current "DING" state at booth i.
  - Therefore, $chicken[i][4]$ is calculated as $-A[i] + chicken[i-1][3]$, and $chicken[i][5]$ is calculated as $-A[i] + chicken[i-1][4]$.

This recurrence relation efficiently computes the maximum number of chickens earned up to the i-th booth for all possible combinations of consecutive "RING" and "DING" choices.

3. The specific subproblems that solve the actual problem

Subproblem 1: For each booth i, the subproblem involves calculating the maximum number of chickens earned up to that booth considering all possible combinations of consecutive "RING" and "DING" choices made up to that point.

Subproblem 2: The state at each booth i is defined by six different scenarios:

- $chicken[i][0]$: Mr. Fox has said "RING" once consecutively.
- $chicken[i][1]$: Mr. Fox has said "RING" twice consecutively.
- $chicken[i][2]$: Mr. Fox has said "RING" thrice consecutively.
- $chicken[i][3]$: Mr. Fox has said "DING" once consecutively.
- $chicken[i][4]$: Mr. Fox has said "DING" twice consecutively.
- $chicken[i][5]$: Mr. Fox has said "DING" thrice consecutively.

Subproblem 3: The maximum number of chickens earned at each booth depends on the choices made at the previous booth $(i-1)$ and the rewards/penalties associated with the current booth i.

Subproblem 4: By efficiently computing the maximum chickens earned at each booth while considering the constraints, we can find the overall maximum number of chickens earned by Mr. Fox when he completes the entire obstacle course, i.e., when he reaches the nth booth.

Therefore, the specific subproblems involve determining the maximum chickens earned at each booth i for all possible combinations of consecutive "RING" and "DING" choices, leading to the solution of the overall problem of maximizing Mr. Fox's chicken count throughout the obstacle course.

4. Algorithm description

Initialization: Given an array A of length n, initialize a 2D array *chicken* of dimensions $n \times 6$ to store the maximum chickens earned considering different consecutive "RING" and "DING" choices.

Base Case: Initialize the first row of the *chicken* array with the rewards/penalties of the first booth.

Dynamic Programming: Iterate over the array A from the second element ($i = 1$) to the last element ($i = n - 1$).

- Update $chicken[i][0] - chicken[i][5]$: For each booth i, update the maximum chickens earned for different consecutive "RING" and "DING" choices based on the rewards/penalties at the current booth and the choices made at the previous booth.

Find Maximum: After updating all elements of the *chicken* array, traverse the last row to find the maximum chickens earned considering all possible combinations of consecutive "RING" and "DING" choices.

Return Maximum: Return the maximum chickens earned found in the last row, which represents the maximum number of chickens Mr. Fox can earn while following the rules.

5. Explanation of the running time of the algorithm

Time Complexity:

1. Let n be the length of the input array A.

2. The algorithm iterates through the array once, performing constant-time operations at each iteration from lines 5 to 10.

3. Each iteration involves several constant-time operations, including comparisons and array accesses. Therefore, the time complexity per iteration is $O(1)$.

4. The total number of iterations is n, as the algorithm traverses the entire array.

5. Hence, the time complexity of the loop from lines 5 to 10 is $n \times O(1)$, which simplifies to $O(n)$.

6. Line 11 performs a constant-time comparison operation 6 times, as it compares the elements in the last row of the 2D array. Since the number of comparisons is constant and independent of the input size n, the time complexity for this part is $6 \times O(1)$, which also simplifies to $O(1)$.

7. Similarly, the helper function (lines 14-17) executes constant-time operations, as it only involves basic comparisons and array accesses. Thus, its time complexity is also $O(1)$.

Combining these complexities, the overall time complexity is the sum of the complexities of each part:

$$\text{Time complexity} = n \times 6 \times O(1) + O(1) = O(n)$$

Therefore, the total time complexity of the algorithm is $O(n)$.