Theory Assignment-2: ADA Winter-2024

Shobhit Raj (2022482) Vashu (2022606)

1 Subproblem Definition

- $dp[CR][CD][RING][i] = \text{boothDoor}[i] + \max(dp[CR][CD][RING][i-1], dp[CR][CD][DING][i-1])$

  This stores the maximum score achievable up to the i^{th} booth with CR consecutive "Ring!" actions, CD consecutive "Ding!" actions, and the current word being "Ring!". The score is calculated by adding the score obtained at the i^{th} booth (boothDoor[i]) to the maximum of the scores achievable at the previous booth with either "Ring!" or "Ding!" actions.

- $dp[CR][CD][DING][i] = -\text{boothDoor}[i] + \max(dp[CR][CD][RING][i-1], dp[CR][CD][DING][i-1])$

  This stores the maximum score achievable up to the i^{th} booth with CR consecutive "Ring!" actions, CD consecutive "Ding!" actions, and the current word being "Ding!". The score is calculated by subtracting the negative score obtained at the i^{th} booth ($-$boothDoor[i]) from the maximum of the scores achievable at the previous booth with either "Ring!" or "Ding!" actions.

CR - Number of Consecutive Rings i.e. CR = 0,1,2,3
CD - Number of Consecutive Dings i.e. CD = 0,1,2,3
CURR - Current Word Mr. Fox says at the i^{th} booth i.e CURR = 0,1 0 is for ring 1 is for ding

2 Recurrence relation for the subproblem

$$maxScore[CR][CD][CURR][i] = \begin{cases} \max(dp[0][1][0][i-1], dp[0][1][1][i-1]) - \text{boothDoor}[i] & \text{if } CR = 3 \\ \max(dp[1][0][0][i-1], dp[1][0][1][i-1]) + \text{boothDoor}[i] & \text{if } CD = 3 \\ \text{score} + \max\Big(maxScore[CR+1][CD+1][0][i-1], \\ \qquad maxScore[CR+1][CD+1][1][i-1]\Big) & \text{otherwise} \end{cases}$$

Notations:
CR - No of Consecutive Rings
CD - No of Consecutive Dings
$CURR = 0$ - Ring said at the i^{th} booth
$CURR = 1$ - Ding said at the i^{th} booth
score: boothScore[i] if $CURR = 0$ (RING), else $-$boothScore[i] if $CURR = 1$ (DING)

3 Specific Subproblem that Solves the Final Problem

- maxChickens = $\max(dp[0][0][0][\text{boothDoor.size()} - 1], dp[0][0][1][\text{boothDoor.size()} - 1])$

  Solving this subproblem gives the final score that represents the maximum number of chickens Mr. Fox can earn by running through the entire obstacle course while adhering to the rules. As it considers possibilities of both the words at the last booth (n^{th} booth here represented by [boothDoor.size() − 1)] element in the boothDoor input array) and selects the one that results in the highest score by taking the maximum of these two values.

4 Algorithm Description

4.1 Pseudocode

Algorithm 1 Dynamic Programming Algorithm

INPUT: Array boothDoor of size n representing the numbers on the booth's door
OUTPUT: Maximum score achievable by Mr. Fox by running through the entire obstacle course

```
Assumption - 0 based indexing
Initialize a 4-D table $dp[][][][]$ with dimensions $4 \times 4 \times 2 \times (N)$

for $CR \leftarrow 0$ to 3 do
    for $CD \leftarrow 0$ to 3 do
        for $CURR \leftarrow 0$ to 1 do
            for $i \leftarrow 0$ to $N-1$ do Initialize $dp[R][D][RD][i] \leftarrow 0$

// Iterative bottom-up approach to fill up the table
    for $i \leftarrow 0$ to $N-1$ do
        for $CURR \leftarrow 0$ to 1 do
            for $CR \leftarrow 0$ to 3 do
                for $CD \leftarrow 0$ to 3 do // Compute score based on current action and booth number
                    if $CURR == 0$ then $score \leftarrow$ boothDoor[i]
                    else $score \leftarrow -$ boothDoor[i]
                       // Handle the cases when R or D reaches 3
                        if $CR == 3$ then $dp[CR][CD][CURR][i] \leftarrow -$ boothDoor[i] +
$\max(dp[0][1][0][i-1], dp[0][1][1][i-1])$
                            if $CD == 3$ then $dp[CR][CD][CURR][i] \leftarrow$ boothDoor[i] +
$\max(dp[1][0][0][i-1], dp[1][0][1][i-1])$
                            else // Calculate the maximum score achievable
                                if $CURR == 0$ then $maxScore \leftarrow \max(dp[R+1][0][0][i-1], dp[R+1][0][1][i-1])$
                                else $maxScore \leftarrow \max(dp[0][D+1][0][i-1], dp[0][D+1][1][i-1])$
                            $dp[CR][CD][CURR][i] \leftarrow score + maxScore$
// Get the maximum score from the last index i.e. the last booth
return $\max(dp[0][0][0][N-1], dp[0][0][1][N-1])$
```

4.2 Detailed Description

This algorithm uses Tabulation Method i.e. filling up of the table in bottom-up approach. It solves the problem by iteratively filling up a table, starting from the smallest subproblems and building up to the original problem. This ensures that each subproblem is solved only once, and its solution is used in solving larger subproblems.

In this dynamic programming algorithm, we use a 4-D table which will be storing the maximum score achievable under different combinations of consecutive actions (choosing the word) and booth positions while adhering to the rules. The dimensions of the table represent the counts of consecutive "Ring!" and "Ding!" actions, as well as the current word spoken by Mr.Fox at the i^{th} door of the booth.

We first initialize the table with zeros. Then, we iterate through each booth in the obstacle course and update the table based on the recurrence relations described earlier. Finally, we return the maximum score achievable at the last booth, considering both possibilities for the last action.

This algorithm ensures that Mr. Fox maximizes his score by choosing the optimal action at each booth, considering the constraints imposed by the rules of the obstacle course.

Score Calculation:

- The algorithm calculates the score for the current booth based on the value of $CURR$ (the last action taken).
- If $CURR$ is 0, it means the last action was "RING", so the score is positive (the door's value).
- If $CURR$ is not 0, it means the last action was "DING", so the score is negative (the negative of the door's value).

Handling Cases for CR and CD reaching 3:

- If CR reaches 3, it means that three consecutive "RING" actions have been taken. In this case, the algorithm maximizes the score by assuming the next action is a "RING" (0) or "DING" (1) and considers the scores from the (index $-$ 1).
- Similarly, if CD reaches 3, it means that three consecutive "DING" actions have been taken. The algorithm maximizes the score by assuming the next action is a "RING" (0) or "DING" (1) and considers the scores from the (index $-$ 1).

Handling Other Cases:

- If neither CR nor CD has reached 3, the algorithm calculates the maximum score achievable by considering both "RING" and "DING" actions.
- It computes the maximum score (maxScore) by checking whether the last action was a "RING" or "DING" and then considering the possibilities for the next action.
- The maximum score is calculated based on whether the last action was "RING" or "DING" and by looking at the next possible actions.

Updating the DP Table:

- Finally, the algorithm updates the DP table for the current R, D, and RD values at the current index with the calculated score (score) and the maximum achievable score (maxScore).

5 Run time analysis

The Algorithm uses a Tabulation approach in which we are filling a table starting from the base case till the top. In the end we return maximum of the case when we choose Ding/Ring on the nth booth.
The filling of the DP table also involves four nested loops iterating over indexes i, CURR, CR, and CD. At each iteration, the algorithm calculates the maximum score based on the choices made at the previous door. This computation of the maximum value takes O(1)-time. The time complexity for filling up the DP table is O(N * 2 * 4 * 4) = O(32N) $\approx O(N)$.

Overall, the runtime complexity of the provided code is dominated by the iterative bottom-up approach, which is $O(N)$.