Theory Assignment-1: ADA Winter-2024

Niteen Kumar (2022336) Shubham (2022488)

1 Assumptions

0-based indexing, array indices start from 0.

Operations like arithmetic (addition, division, subtraction, multiplication), comparison between two elements take constant time.

We are assuming that there will be no overflows.

2 Preprocessing

A 3D vector dp is created for dynamic programming. This vector has dimensions (n + 1) * 2 * 4 to account for the varying states involved in the problem. Each element of this vector is initialized to INT_MIN to signify an unset value. dp[0][0][0], and dp[0][1][0] are set to 0.

3 Subproblem

The specific subproblem(s) that solve the actual problem:

Booth decision:

At each booth i, the algorithm makes a choice: say RING or DING.

Consecutive word restriction:

The choice needs to respect the rule that saying the same word cannot exceed three consecutive times. Therefore, for each word choice (RING or DING), the algorithm considers three subproblems:

1. **Subproblem 1.1 (j=1):** Saying the chosen word for the first time at booth i. This involves considering possible transitions from previous states (either RING or DING) depending on the chosen word.
2. **Subproblem 1.2 (j=2):** Saying the chosen word for the second consecutive time at booth i. This subproblem only applies if the algorithm already said the same word at the previous booth $(i - 1)$, and it involves considering continuing this sequence or shifting from previous states with the same word.
3. **Subproblem 1.3 (j=3):** Saying the chosen word for the third consecutive time at booth i. Similar to Subproblem 1.2, this only applies if the word was said twice consecutively before and involves considering continuation or shifting.

Maximizing chicken count:

For each subproblem (1.1, 1.2, 1.3), the algorithm calculates the maximum achievable chicken count by considering:

- Transition from previous states: The algorithm explores possible transitions from previous states (RING or DING) considering the word difference (chicken gain/loss) and the maximum value among different previous state options.

- Continuing the same word sequence (if allowed): If allowed by the consecutive word rule, the algorithm considers continuing the same word sequence by adding the current booth value and adjusting based on the word difference.
- Shifting from previous states with the same word (if allowed): For consecutive occurrences beyond the first two ($j = 2, 3$), the algorithm explores shifting from different previous occurrences of the same word to the current sequence length, again adjusting values based on the word difference.

4 Algorithm Description

Initialization:

- Create a 3D dynamic programming table dp with dimensions $(n + 1) \times 2 \times 4$, where:
  - 1st state: Represents booth numbers from 0 to n (0 for initialization).
  - 2nd state: Represents the current word (0: RING, 1: DING).
  - 3rd state: Represents the consecutive occurrences of the current word (0 to 3).
- Initialize all values in dp to negative infinity (INT_MIN) except:
  - $dp[0][0][0] = 0$: Represents no words said initially.
  - $dp[0][1][0] = 0$: Represents no words said initially.

Iteration:

- For each booth i from 1 to n:
  - **RING case:**
    - For each consecutive count j from 1 to 3:
      - Consider previous DING states:
      - Calculate $\max(dp[i-1][1][k]+A[i])$ for k from 0 to 3, which represents the maximum achievable by saying RING next, considering possible previous DING states with different consecutive counts (k).
      - Consider previous RING state (if $j > 1$):
      - Calculate $dp[i - 1][0][j - 1] + A[i]$, which represents the maximum achievable by continuing the RING sequence.
      - Consider shifting from previous RING states with different consecutive counts (if $j > 1$):
      - Calculate $dp[i - 1][0][k] + A[i]$ for k from 0 to $j - 2$, which represents shifting from different previous RING states to a RING sequence of length j.
      - Choose the maximum among all calculated values and store it in $dp[i][0][j]$.
  - **DING case:**
    - For each consecutive count j from 1 to 3:
      - Consider previous RING states:
      - Calculate $\max(dp[i-1][0][k]-A[i])$ for k from 0 to 3, which represents the maximum achievable by saying DING next, considering possible previous RING states with different consecutive counts (k).
      - Consider previous DING state (if $j > 1$):
      - Calculate $dp[i - 1][1][j - 1] - A[i]$, which represents the maximum achievable by continuing the DING sequence.
      - Consider shifting from previous DING states with different consecutive counts (if $j > 1$):
      - Calculate $dp[i - 1][1][k] - A[i]$ for k from 0 to $j - 2$, which represents shifting from different previous DING states to a DING sequence of length j.
      - Choose the maximum among all calculated values and store it in $dp[i][1][j]$.

5 Recurrence Relation

Iterative solution, so no recurrence relation.

6 Time Complexity Analysis

The code has a loop iterating over the booth number n. Each iteration involves constant-time calculations and comparisons. And there are 6 operations inside the loop which take constant time.

6.1 Overall Time Complexity

Total time complexity: $O(6 \cdot n)$ which simplifies to $O(n)$

7 Space Complexity Analysis

7.1 Auxiliary Space

3D vector dp: This is the key element storing state information. Its size is (n+1) x 2 x 4, which simplifies to $O(n)$.

7.2 Input Space

Stores booth values, size $O(n)$ but not relevant to dynamic space growth. Loop variables: Constants like i that don't contribute significantly

7.3 Overall Space Complexity

The overall space complexity is the sum of the auxiliary space and input space complexities: $O(n)$ (auxiliary space) + $O(n)$ (input space) = $O(n)$. Therefore, the overall space complexity of the provided code is $O(n)$.

8 Pseudo code

Algorithm 1 Ring_or_Ding

```
function RING_OR_DING($n$, $arr$)
    // 3d vector to store which word was called and how many times
    $dp[0][0][0] \leftarrow 0$
    $dp[0][1][0] \leftarrow 0$
    for $i \leftarrow 1$ to $n$ do
        // if ring was called
        $dp[i][0][1] \leftarrow \max(\{dp[i-1][1][0], dp[i-1][1][1], dp[i-1][1][2], dp[i-1][1][3]\}) + arr[i-1]$
        $dp[i][0][2] \leftarrow dp[i-1][0][1] ==$ INT_MIN?INT_MIN : $dp[i-1][0][1] + arr[i-1]$
        $dp[i][0][3] \leftarrow dp[i-1][0][2] ==$ INT_MIN?INT_MIN : $dp[i-1][0][2] + arr[i-1]$
        // if ding was called
        $dp[i][1][1] \leftarrow \max(\{dp[i-1][0][0], dp[i-1][0][1], dp[i-1][0][2], dp[i-1][0][3]\}) - arr[i-1]$
        $dp[i][1][2] \leftarrow dp[i-1][1][1] ==$ INT_MIN?INT_MIN : $dp[i-1][1][1] - arr[i-1]$
        $dp[i][1][3] \leftarrow dp[i-1][1][2] ==$ INT_MIN?INT_MIN : $dp[i-1][1][2] - arr[i-1]$
    end for
    return $\max(\{dp[n][0][1], dp[n][0][2], dp[n][0][3], dp[n][1][1], dp[n][1][2], dp[n][1][3]\})$
end function
```

9 Proof of Correctness

Proof of Correctness

Optimal Substructure:

Let $OPT(i)$ denote the optimal solution at booth i for $i = 1, 2, ..., n$, where n is the total number of booths. We aim to maximize the total chicken count at the end of the process.

At each booth i, the algorithm makes a choice between saying "RING" or "DING". Denote the choice at booth i as $dp[i][\text{word}][\text{count}]$, where word represents whether "RING" or "DING" was said, and count represents the consecutive occurrences of the chosen word.

The optimal solution at booth i can be obtained by considering the optimal solutions at the previous booth $i - 1$, and then selecting the choice that maximizes the chicken count.

Therefore, we have:

$$OPT(i) = \max(dp[i-1][\text{RING}][\text{count}], dp[i-1][\text{DING}][\text{count}])$$

This demonstrates the optimal substructure property, as the optimal solution to the problem can be constructed from optimal solutions to its subproblems.

Overlapping Subproblems:

The algorithm uses dynamic programming to store the results of subproblems in the 3D vector dp. This allows the algorithm to avoid redundant calculations by storing and reusing intermediate results.

The subproblems are defined based on the number of booths and the choices made at each booth. Since the algorithm iterates through each booth and calculates the optimal solution based on previous subproblems, it is evident that there are overlapping subproblems.

Conclusion:

The provided algorithm exhibits both optimal substructure and overlapping subproblems, which are the key properties required for dynamic programming. Therefore, by correctly implementing the recurrence relation and ensuring that all subproblems are considered, the algorithm produces the correct result, maximizing the chicken count by choosing the optimal sequence of "RING" and "DING" calls.

10 Example

Dry Run of Ring_or_Ding Algorithm

Given $n = 4$ and arr $= \{2, -4, -5, -7\}$.

Initialization:

- Initialize dp as a 3D vector with dimensions $(5 \times 2 \times 4)$.
- Set $dp[0][0][1] = 0$, $dp[0][0][0] = 0$, $dp[0][1][1] = 0$, and $dp[0][1][0] = 0$.
- Initialize all other values in dp to INT_MIN.

Iteration:

- **Booth 1:**
  - If "RING" is called:

$$dp[1][0][1] = \max(\{0, 0, 0, 0\}) + 2 = 2$$
$$dp[1][0][2] = \text{INT\_MIN}$$
$$dp[1][0][3] = \text{INT\_MIN}$$

- If "DING" is called:

$$dp[1][1][1] = \max(\{0,0,0,0\}) - 2 = -2$$
$$dp[1][1][2] = \text{INT\_MIN}$$
$$dp[1][1][3] = \text{INT\_MIN}$$

- **Booth 2:**
  - If "RING" is called:

$$dp[2][0][1] = \max(\{-2,0,0,0\}) - 4 = -6$$
$$dp[2][0][2] = 2 - 4 = -2$$
$$dp[2][0][3] = \text{INT\_MIN}$$

  - If "DING" is called:

$$dp[2][1][1] = \max(\{0,0,0,0\}) + 4 = 6$$
$$dp[2][1][2] = 2 + 4 = 2$$
$$dp[2][1][3] = \text{INT\_MIN}$$

- **Booth 3:**
  - If "RING" is called:

$$dp[3][0][1] = \max(\{-2,4,4,0\}) - 5 = 1$$
$$dp[3][0][2] = -4 - 5 = -11$$
$$dp[3][0][3] = 2 - 5 = -7$$

  - If "DING" is called:

$$dp[3][1][1] = \max(\{2,2,0,0\}) + 5 = 3$$
$$dp[3][1][2] = 0 + 5 = 11$$
$$dp[3][1][3] = 6 + 5 = 7$$

- **Booth 4:**
  - If "RING" is called:

$$dp[4][0][1] = \max(\{7,-1,-9,-3\}) - 7 = 4$$
$$dp[4][0][2] = -2 - 7 = -6$$
$$dp[4][0][3] = -4 - 7 = -18$$

  - If "DING" is called:

$$dp[4][1][1] = \max(\{-4,4,4,0\}) + 7 = 8$$
$$dp[4][1][2] = 2 + 7 = 10$$
$$dp[4][1][3] = 11 + 7 = 18$$

Result:

The maximum chicken count at the end of the process is obtained from:

$$\max(\{4,-6,-18,8,10,18\}) = 18$$