Theory Assignment-2: ADA Winter-2024

Himang Chandra Garg (2022214) Nishil Agarwal (2022334)

1 Preprocessing

Not Applicable in this Algorithm.

2 Subproblem

We calculate these 2 values at each step of loop :

dp[i][0] = The maximum number of chickens Mr.Fox has collected till ith booth if he says Ring at ith booth.

dp[i][1] = The maximum number of chickens Mr.Fox has collected till ith booth if he says Ding at ith booth.

Here, the indexing starts from 0.

3 Reccurrence of Subproblem

$$dp[i][0] = \max \left\{ \begin{array}{l} dp[i-1][1] + v[i] \\ dp[i-2][1] + v[i-1] + v[i] \\ dp[i-3][1] + v[i-2] + v[i-1] + v[i] \end{array} \right\}$$

$$dp[i][1] = \max \left\{ \begin{array}{l} dp[i-1][0] - v[i] \\ dp[i-2][0] - v[i-1] - v[i] \\ dp[i-3][0] - v[i-2] - v[i-1] - v[i] \end{array} \right\}$$

4 Base Case

$dp[0][0] \leftarrow v[0]$

$dp[0][1] \leftarrow -v[0]$

$$dp[1][0] \leftarrow \max \left\{ \begin{array}{l} v[0] + v[1] \\ -v[0] + v[1] \end{array} \right\}$$

$$dp[1][1] \leftarrow \max \left\{ \begin{array}{l} -v[0] - v[1] \\ v[0] - v[1] \end{array} \right\}$$

$$dp[2][0] \leftarrow \max \left\{ \begin{array}{l} v[0] + v[1] + v[2] \\ dp[0][1] + v[1] + v[2] \\ dp[1][1] + v[2] \end{array} \right\}$$

$$dp[2][1] \leftarrow \max \left\{ \begin{array}{l} -v[0] - v[1] - v[2] \\ dp[0][0] - v[1] - v[2] \\ dp[1][0] - v[2] \end{array} \right\}$$

5 Subproblem that solves the final problem

$$\max \left\{ \begin{array}{l} dp[n-1][0] \\ dp[n-1][1] \end{array} \right\}$$

6 Algorithm Description

We have used tabularization or a bottom-up approach of Dynamic Programming using a 2-D array.

Here, the ith element of our dp vector is a pair of elements where the first element represents the maximum number of chickens Mr.Fox currently has if he chooses Ring on the ith booth and similarly, the second element represents the maximum number of chickens he currently has if he chooses Ding on the ith booth.

Let us suppose, he chooses Ring at the ith booth, then one of these 3 cases must have arisen:

1. he had chosen Ding at i-1 booth.
2. if not, then he had chosen Ding at the i-2 booth.
3. if not, then he had chosen Ding at the i-3 booth.

We choose the maximum of these 3 cases by using our previous results of Ding stored in the dp vector and storing that in the current entry of Ring. We use self-written max functions for this.

Similarly, we can calculate the current entry if he had chosen Ding instead.

We continue this process till the whole given vector is traversed. Then, finally, we return the larger of the 2 elements of dp at n-1th index, which is either when Mr. Fox will Ring or Ding at the last booth.

7 Runtime Complexity Analysis

The time complexity of this Algorithm is

$$O(n)$$

Here, n is the size of the initial vector given to us.

We came to this result since our code contains 3 base cases which takes O(1). The max function we have used for comparing 2/3 elements at a time computes result in O(1) time. Finally, a single for loop has been used which iterates from 3 to n-1 which gives us the total time complexity of O(n).

8 Pseudocode

Algorithm 1 Max of Two

```
1: function maxOfTwo(a, b)
2:     if a > b then
3:         return a
4:     else
5:         return b
6:     end if
7: end function
```

Algorithm 2 Max of Three

```
function maxOfThree(a, b, c)
    if a > b then
        if a > c then
            return a
        else
            return c
        end if
    else
        if b > c then
            return b
        else
            return c
        end if
    end if
end function
```

Algorithm 3 Solve

```
function solve(n, v)
    Initialize dp as an array of size n × 2, filled with zeros
    dp[0][0] ← v[0]
    dp[0][1] ← −v[0]
    dp[1][0] ← maxOfTwo(v[0] + v[1], −v[0] + v[1])
    dp[1][1] ← maxOfTwo(−v[0] − v[1], v[0] − v[1])
    dp[2][0] ← maxOfThree(v[0] + v[1] + v[2], dp[0][1] + v[1] + v[2], dp[1][1] + v[2])
    dp[2][1] ← maxOfThree(−v[0] − v[1] − v[2], dp[0][0] − v[1] − v[2], dp[1][0] − v[2])
    for i ← 3 to n − 1 do
        dp[i][0] ← maxOfThree(dp[i−1][1]+v[i], dp[i−2][1]+v[i−1]+v[i], dp[i−3][1]+v[i−2]+v[i−1]+v[i])
        dp[i][1] ← maxOfThree(dp[i−1][0]−v[i], dp[i−2][0]−v[i−1]−v[i], dp[i−3][0]−v[i−2]−v[i−1]−v[i])
    end for
    return max(dp[n − 1][0], dp[n − 1][1])
end function
```