Theory Assignment-2: ADA Winter-2024

Himanshu Raj (2022216) Tanish Verma (2022532)

February 11, 2024

1 Preprocessing

Given an input array A of size n (greater than 0), construct a two-dimensional dp array of size nx2 (n arrays of size 2). Initial values of dp array don't matter as we will update the entire array during the execution of our algorithm. For Ring, we'll use A[i] and for Ding, -A[i].
We will follow 0-based indexing, i.e. A[0,1,...,n-1] and the indexes of dp array are as follows

$$\begin{bmatrix} (0,0) & (0,1) \\ (1,0) & (1,1) \\ . & . \\ . & . \\ (n-1,0) & (n-1,1) \end{bmatrix}$$

2 Subproblem Definition

dp[i][0] →The maximum amount of chickens in possession of Mr. Fox till the i^{th} week, where Mr. Fox says Ring in the i^{th} week.
dp[i][1]→The maximum amount of chickens in possession of Mr. Fox till the i^{th} week, where Mr. Fox says Ding in the i^{th} week.
Note: The second parameter's value refers to Ring for that week if it is 0, or Ding if it is 1.

3 Recurrence Relation

Base Cases
if (n > 0):
dp[0][0] = A[0]
dp[0][1] = -A[0]

if (n > 1):
dp[1][0] = A[1] + max(dp[0][0] , dp[0][1])
dp[1][1] = -A[1] + max(dp[0][0] , dp[0][1])

if (n > 2):
dp[2][0] = A[2] + max(dp[1][0] , dp[1][1])
dp[2][1] = -A[2] + max(dp[1][0] , dp[1][1])

Recurrence of the Subproblem for i = 3 to n-1

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

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

Note: The second parameter's value refers to Ring for that week if it is 0, or Ding if it is 1.
The value taken for Ring at i^{th} week is A[i].

The value taken for Ding at i^{th} week is -A[i].
So, the negative symbol represents Ding for that week.

4 Subproblem that solves the actual problem

max (dp[n-1][0], dp[n-1][1]) → returns the largest number of chickens that Mr. Fox earns by running in the obstacle course.

5 Pseudocode

```
1:  function CHICKENSOLVER(A, n, dp)
2:      if (n>0):
3:          dp[0][0] = A[0]
4:          dp[0][1] = -A[0]
5:
6:      if (n>1):
7:          dp[1][0] = A[1] + max(dp[0][0], dp[0][1])
8:          dp[1][1] = -A[1] + max(dp[0][0], dp[0][1])
9:
10:     if (n>2):
11:         dp[2][0] = A[2] + max(dp[1][0], dp[1][1])
12:         dp[2][1] = -A[2] + max(dp[1][0], dp[1][1])
13:
14:     for (i : 3 to n-1):
15:
16:         dp[i][0] = A[i] + max (dp[i-1][1], A[i-1]+dp[i-2][1], A[i-1]+A[i-2]+dp[i-3][1])
17:
18:         dp[i][1] = -A[i] + max (dp[i-1][0], -A[i-1]+dp[i-2][0], -A[i-1]-A[i-2]+dp[i-3][0])
19:
20:     return max (dp[n-1][0], dp[n-1][1])
21: end function
```

6 Algorithm Description

The primary objective is to calculate the maximum amount of chickens Mr. Fox can get at the end of the path, while keeping in mind the constraints. For this we will use the concept of dynamic programming.
The dp array stores the maximum chickens at the i^{th} iteration for both cases, that is, when i^{th} choice is Ring or Ding. Note that in the dp array, dp[i][0] is the value in the array when i^{th} entry is a Ring and dp[i][1] is the value when it is a Ding.
The first three entries of the dp array correspond directly to the base cases of the dynamic programming problem, the explanation of base cases is as follows:
For i = 0, the entry for the Ring column is just A[i] and for Ding column is just -A[i].
For i = 1, it is A[i] + the maximum of the two columns in the previous entry, which is maximum of dp[0][0] and dp[0][1] in Ring case. For Ding case, it is similar but A[i] is replaced with -A[i].
For i = 2, it is A[i] + the maximum of the two columns in the previous entry, that is maximum of dp[1][0] and dp[1][1] in Ring case. For Ding, we use -A[i] instead of A[i].

Now,
for i→3 to n-1
we use the Recurrence defined above to calculate the dp[i][0]th and dp[i][1]th entries in the array.
To calculate dp[i][0] and dp[i][1], there are three cases each to ensure the given constraints are not violated while calculating maximum chickens, that is, at most three Rings or Dings are allowed in a row.
Now, for dp[i][0], the cases are as follows:
If the i^{th} choice is a Ring,
then i-1th choice can be a Ding, so we take dp[i-1][1]

If the i-1th choice is also a Ring, then i-2th choice can be a Ding, so we take dp[i-2][1] + A[i-1]
If the i-2th choice is also another Ring, then i-3th choice can only be a Ding, so we take dp[i-3][1] + A[i-2] + A[i-1]
Finally for dp[i][0] we take the maximum value of these three cases and add the value of A[i].

For dp[i][1], the cases are as follows:
If the ith choice is a Ding,
then i-1th choice can be a Ring, so we take dp[i-1][0]
If the i-1th choice is also a Ding, then i-2th choice can be a Ring, so we take dp[i-2][0] - A[i-1]
If the i-2th choice is also a Ding, then i-3th choice has to be a Ring, so we take dp[i-3][0] - A[i-2] - A[i-1]
Finally, for dp[i][1] we take the maximum value of these three cases and add -A[i] to it.

Finally, we return the maximum of the values in dp[n-1][0] and dp[n-1][1].

7 Complexity Analysis

7.1 Time complexity

During the algorithm, we are iterating over the array A once, and in each iteration we compute values for the dp array by taking the maximum from 3 values twice, and the values are taken from the previous indexes of the dp array. Both these operations of getting values and computing their maximum take constant time. So running time complexity of our algorithm is O(n).

7.2 Space complexity

We are using an additional dp array of size 2n, so additional space complexity of our algorihtm is O(2n).