**NOTE :**

**Toughest Problem in DP asked in OA ever [watch video again if you don’t get it white read the notes]**

**Qn Link :** <https://www.desiqna.in/12898/airbnb-oa-sde-set-3-dp-april-2023>

**Question Summary :**

You re given “N” stones and need to remove all the stones from the row.

You ve given an set of rules to remove the stone.

* If (i-1)th and (i+1)th stones are still present , then , cost of removing the ith stone is b[i]
* if either (i-1)th or (i+1)th stone is present , then cost of removing the ith stone is a[i].
* if neither (i-1)th nor (i+1)th stone is present , the cost of removing the ith stone is 0.

Find the minimum total cost of removing all the stones.

**Observation :**

Sample : A = [1 , 5 , 3 , 2] , B = [5 , 8 , 8 , 1]

Remove 3rd stone = 3

Remove 1st stone = 1

Remove 2nd stone = 0

Remove last stone = 0

* You can observe that , we don’t know the order of element we r picking .
* So we need to maintain a separate state for that .But what re those states ?
  + If the element is last → No left / No right element
  + If the element is secondLast → Have left and right element.
* But where is the state for no element ?

Let’s assume we are picking an element “i” which is last element , then there is no rightmost element.

Then the formula be a[i] + Best Answer dp[i-1]

Dp[i][last] = a[i] + min(dp[i - 1] [last] , dp[i - 1][second\_last])

If “i” is the last index , then i - 1 is the second last index , let the order of pick up be

**Order 1 :**

, i-1 → i → i - 2 [ one of possible pick up]

If i pick i - 1, then there is no element between right and left of i , so it is zero .

But there is a element on the left side , it is i - 2

Dp[i][second\_last] = b[i] + 0 (i) + min(dp[i - 2] [last] , dp[i - 2][second\_last])

**Order 2 :**

, i - 2 → i - 1 → i

Dp[i] [second\_last] = dp[i - 1] [second\_last] [2] + a[i]

**Explanation :**

**For first Equation :**

Assume that we re picking an element which is right most part

* Then there is no guy on the right , but there is guy in the left

, i- 3 , i - 2 , i - 1 , i

* I’m picking i , but i don;t know when i will pick i - 1 , but surely i will pick i - 1
* So the answer will be best (dp[i - 1] ) + a[i] (one guy)

**For Second Equation :**

* Assume we are picking the element where left and right element is present (i - 1)

i - 2 → i - 1 → i

* Now there is two possible order
  + , i - 2 → i -1 → i
  + , i - 2 → i -1 → i
* Formula for first order
  + We know i is the last element , but when picking i - 1 left and right element are present (b[i - 1])
  + After removing i - 1 , there is no element is present (left & right) for i
  + So cost to remove it is zero.
  + So cost to pick i - 1 has two possibilities
    - Either the i - 2 th guy is last element
    - Or i - 2 th guy is second last element

Value\_1 = b[i - 1] + 0 + min (dp[i -2][last] , dp[i - 2][second\_last)

* Formula for second order
  + Let’s assume we re picking i - 2 as first and i - 1 as second
  + If we picked i - 2 then there is only one element in the right of i - 1 that is i .So a[i - 1]
  + Plus , we re picking this (i - 1) as the second last element so dp[i -2][second\_last]

Dp[i] [second\_last] = min (formula 1 , formula 2)

**Recurrence Relation :**

Dp[i] [last ] = a[i] + min (dp[i -1][last] , dp[i - 1][second\_last)

V1 = b[i - 1] + 0 + min min (dp[i -2][last] , dp[i - 2][second\_last)

V2 = a[i - 1] + dp[i - 1] [second\_last]

Dp[i][second\_last] = min(v1 , v2)

**Base Case :**

When there is only one stone, then the cost to remove it will be 0

, dp[i] [last] = dp[i][second\_last] = 0

When there is 2 stone , then the cost to remove it will be

* dp[i][last] = a[2] [removed before 1]
* dp[i][second\_last] = a[1] + dp[1][2] → a[1] + 0 → a[1] [ removed after 1]

Code :

class Solution {

public int minSteps(int[] A, int [] B , int n) {

int [][] dp = new int[n + 1][3];

/\*

1 --> last

2 --> Second last

\*/

dp[1][1] = dp[1][2] = 0;

dp[2][1] = a[2];

dp[2][2] = a[1];

for(int i = 3 ; i < n ; i++){

dp[i][1] = a[i] + Math.min(dp[i - 1] [1] , d[i - 1][2]);

int v1 = b[i - 1] + 0 + Math.min(dp[i - 2][1] , dp[i - 2][2]);

int v2 = a[i - 1] + dp[i - 2][1];

dp[i][2] = Math.min(v1 , v2);

}

return Math.min(dp[n][1] , dp[n][2]);}}