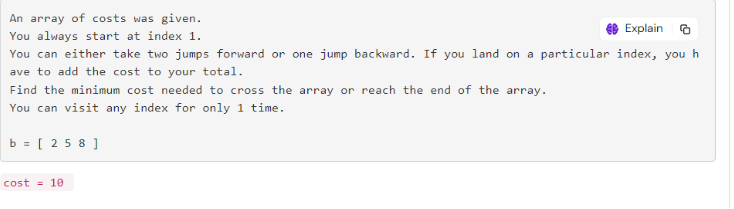
**Qn Link :**

<https://www.desiqna.in/13008/goldman-sachs-sde-oa-ctc-30-lac-2023-dp>

Question :



**Note :**

* We can either reach index “N” or out of index “N”.
* The jump allowed here is + 2 , so index “N - 1” has the potential to reach out of the index.
* So we need to consider min of (n-1 , n)

**Observation :**

* We can move backwards only one time , we can’t do backwards two times.
* Suppose we have index i , i + 1 , i + 2 , i + 3
* Now im at index i and i want to move to index i + 1 ,
  + Reach index i + 2 by + 2 jump .
  + Reach index i + 1 by -1 jump.
* But now from i + 1 you can’t go backwards , because we have already visited index “i”.

Can we maintain the state for both forward and backward jump in one array ?

* No , we can’t.
* Key Note : Which ever the thing creating problem for you , then create a separate state for it.
* So we maintain a 2D dp array.
  + One for forward move and another one for backward move

**What will be the value of dp[i] ?**

→ min (forward , backward) ;

**But how will you calculate the forward move and backward move ?**

→ forward move is easy simple dp[i - 2] + b[i]

→ but for backward move , we can’t directly reach desired index /

* for example , in the observation case we reached index i + 1 from i through i +2 then - 1.
* It is same for every backward move .
  + dp[i - 1] + b[i + 1] backward move + next index value.

**Recurrence Relation :**

dp[i][forward] = b[i] + min ( dp[i - 2][forward] , dp[i - 2] [backward])

dp[i][backward] = b[i] + b[i + 1] + dp[i - 2] [forward].

**Explanation :**

* For forward move , we can either come from forward move or backward move of the (i - 2) index .We need minimum of it.
* For backward move we only come from forward move.

Step 1 : create an 2D dp array of size n + 1.

Step 2 : for index 1 and index 2 , we can’t move backwards so fill it with INT\_MAX.

Step 3 : for index 3 to n , use the recurrence formula and find the index.

**Code :**

class Solution {

private int min(int a , int b , int c){

if(a < b && a < c){

return a;

}

if(b < c){

return b;

}

return c;

}

public int minOperation(int [] nums, int n){

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

//For index 1 and 2 ( 1 - based indexing)

//--> we can't reach index 1 by a forward move and

//--> we can't reach index 1 by a backward move from index 2

dp[1][1] = Integer.MAX\_VALUE;

dp[2][2] = Integer.MAX\_VALUE;

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

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

for(int i = 1 ; i < n- 1 ; i++){

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

dp[i][2] = nums[i] + nums[i + 1] + dp[i - 2][1];

}

dp[n - 1][1] = nums[n - 2] + Math.min(dp[n - 2 - 2][1] , dp[n - 2 - 2][2]);

dp[n - 1][2] = Integer.MAX\_VALUE; // WE can't reach index n- 1 from backwards because it is already visited

return min (dp[n - 1][1] , dp[n - 2][1] , dp[n - 2][1]);

}

}