**Thumb Rule :**

* If a problem is solved by Dp approach , then it is guaranteed that you make Choice at each step and it is best for now and the future . Then use greedy method.
* If there are multiple choice at each step and you don’t know which of the option give the best answer then use DP.

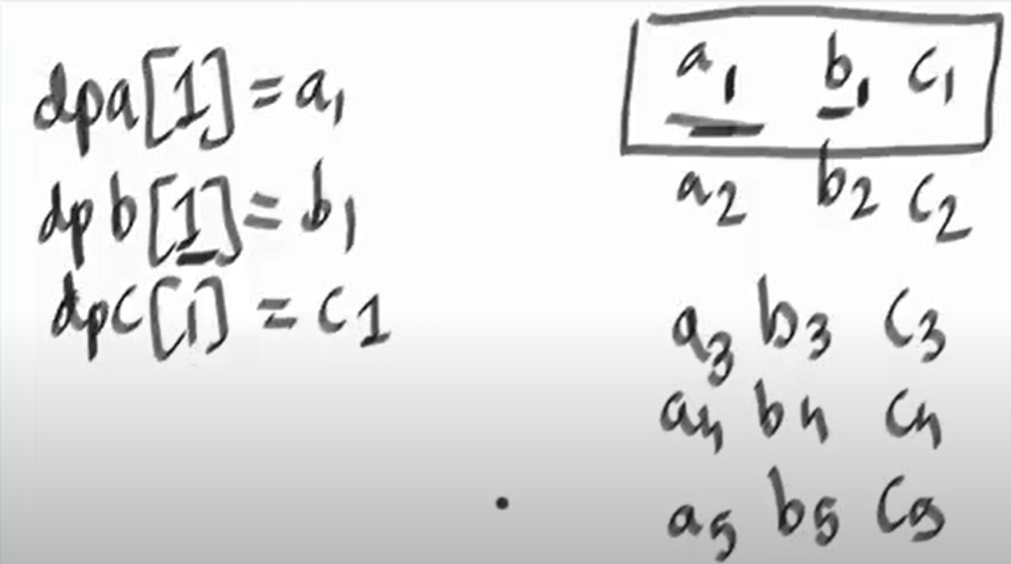
**Qn Link :** <https://atcoder.jp/contests/dp/tasks/dp_c>

**Question Summary :**

* You re given a 2D array , each row has 3 values , let the column b be a , b , c.
* At each row you should select the maximum , but for the next row you can’t select the same column again .
* Suppose if you re selecting column c value at row i , then you should select only from column A or B in the i +1 th row.

Observation :

* We can select any of the three elements , because we don’t know that if the selected options will lead to maximum sum
* Like selecting maximum At row 0 , and at row 1 from remaining 2 , …. Will it lead to maximum sum ? ANS : Don’t know
* SOLN : create 3 Dp array and keep track of all possible paths and return the maximum.



Step 1: Create 3 array dpa[] , dpb[] , dpc[]

* dpa[i] = best possible answer if we choose a[i] at the index i
* dpb[i] = best possible answer if we choose b[i] at the index i
* dpc[i] = best possible answer if we choose c[i] at the index i

Step 2: Init the DP arrays by

* dpa[0] = a[0]
* dpb[1] = b[0]
* dpc[0] = c[0]

Step 3 : For i = 1

* dpa[1] max (dpb[0] , dpc[0]) + a[1]
* dpb[1] max (dpa[0] , dpc[0]) + b[1]
* dpc[1] max (dpa[0] , dpb[0]) + b[1]

Step 4 : For each iteration use the recurrence formula .

Step 5 : return the max (dpa[n] , dpb[n] ,dpc[n])

**Recurrence Relation :**

* dpa[i] = max (dpb[i - 1] , dpc[i - 1]) + a[i]
* dpb[i] = max (dpa[i - 1] , dpc[i - 1]) + b[i]
* dpc[i] = max (dpa[i - 1] , dpb[i - 1]) + c[i]

**Code :**

class Solution {

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

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

return a;

}else if ( b > c){

return b;

}

return c;

}

public int maxHappiness(int [][] nums) {

int n = nums.length;

int [] dpa = new int[n];

int [] dpb = new int[n];

int [] dpc = new int[n];

dpa[0] = nums[0][0];

dpb[0] = nums[0][1];

dpc[0] = nums[0][2];

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

dpa[i] = Math.max(dpb[i - 1] , dpc[i - 1]) + nums[i][0];

dpb[i] = Math.max(dpa[i - 1] , dpc[i - 1]) + nums[i][1];

dpc[i] = Math.max(dpa[i - 1] , dpb[i - 1]) + nums[i][2];

}

return max (dpa[n - 1] , dpb[ n - 1] , dpc[n - 1]);

}

}

**Follow Up :** Select upto 2 consecutive Elements

**Recurrence relation :**

Dpa[i] = Math.max (dpb[i - 1] , dpc[i - 1) + nums[i][0] ; (selecting from prev row)

//getting max of i- 1 & i - 2 th index

Dpa[i] = math.max(dpa[i] ,

// selecting from i - 2 th row because we can select till 2 consec indexes

nums[i][0] + nums[i - 1] [0] + Math.max(dpb[i - 2] , dpc[i - 2]))