ACM SUMMER CHALLENGE – 2K20

EDITORIAL

RECURSION AND DYNAMIC PROGRAMMING

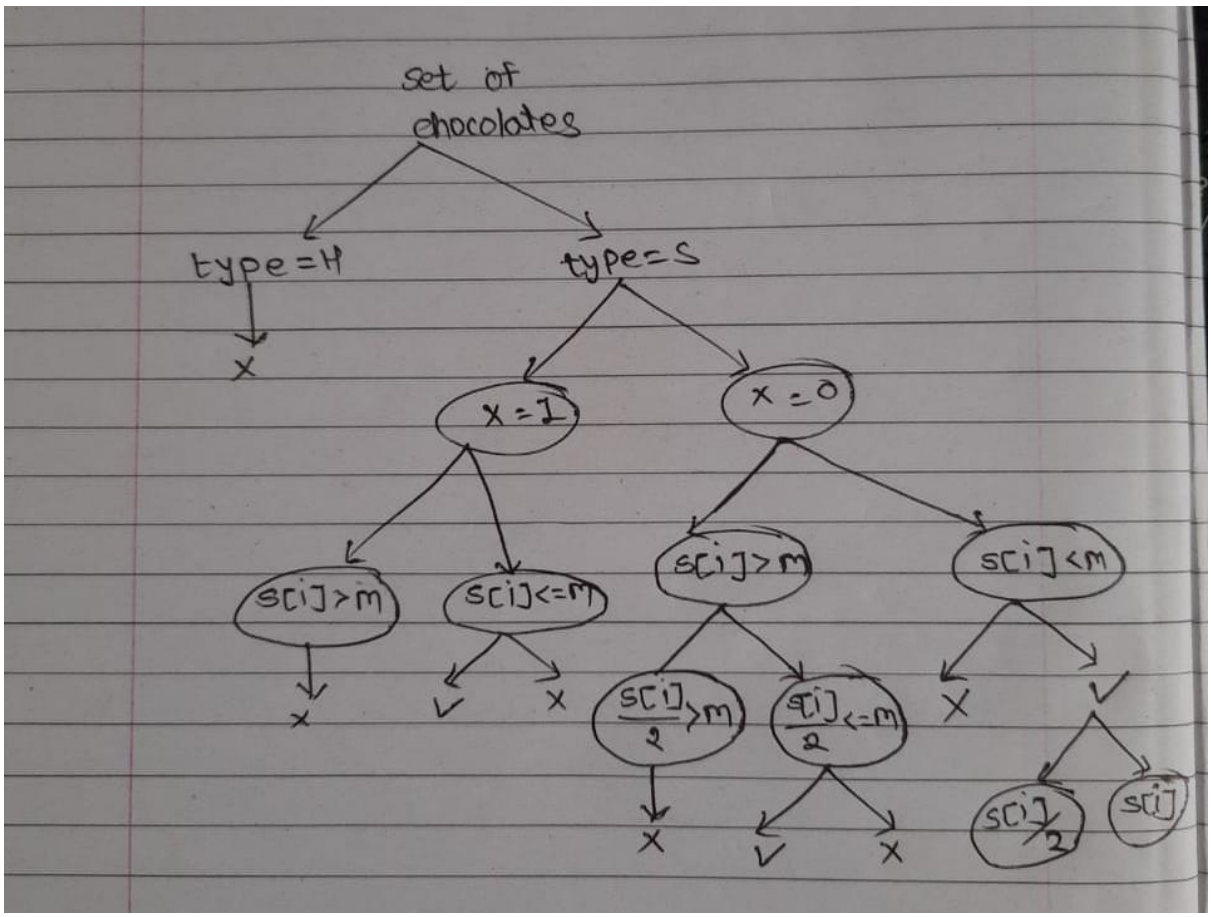
THE CHOCOLATE MAN:

Author : Jignesh

Motivation:

This problem is an extended version of 0-1 knapsack problem. You have to try all possibilities for choosing chocolates such that sum of sweetness shouldn't exceed M. So you can get idea that this can be solved using dp.

Idea:



Above picture represents the concept for selecting chocolates.

Here x is 1 if you already use the special operation to halve the sweetness, otherwise it is 0. You can easily understand how to code using this choice diagram.

LOKI AND TESSERACT:

Author : Shivangi

Motivation:

You can traverse right or down from any cell if that cell is possible. You can easily see that you have to remember just last trasntion of a[i-1][j] and a[i][j-1]. For that purpose you have to use dp.

Idea:

Let make one array say dp[n+1][m+1]. Initialize all elements of array with 0. Now start from (0,0). If a[0][0] is 0 then dp[0][0] is 0, otherwise it is 1.

Now for every i and j exclude(i = 0 && j =0), if a[i][j] = 1 then make dp[i][j] = 0 else if possible dp[i][j] = max({dp[i-1][j] + 1 , dp[i][j-1] + 1}).

Then traverse dp array and find the maximum value.

HARRY POTTER AND SPELLS:

Author : Mihir

Motivation:

You can check all possibilities. For every possible i, you can either multiply x[i] and x[i+1] with -1 or not. So you can use dp.

Idea:

let's solve this problem recursively. let (ind, invert) be the parameters of the recursive function. Where ind is the current index we are at. And invert will keep track weather the current ind has been inverted by ind -1th element or not.

So, we know weather the current index has been inverted or not ,i.e multiplied by -1 or not. We can call the next ind accordingly by removing and adding the necessary elements.

And at the end end we can use dp[ind][invert] to avoid repetitive calculation.

Time complexity is O(N)