

" Those who can not remember
Past, are condemned to repeat
it ! "

Dynamic programming

—By steiver.



Dynamic Programming Playlist | Interview Questions | Recursion | Tabulation | Striver | C++ | Java | DSA | Placements

take U forward

The playlist aims to teach you Dynamic Programming in depth. The focus of the playlist is to cov...[MORE](#)



PLAY

SHUFFLE

57 videos

Description

The playlist aims to teach you Dynamic Programming in depth. The focus of the playlist is to cover all the concepts, and then follow it up with a lot of problems so that the concepts go into your head and stay there.

The focus is on logic, so no matter in which language you code, you can easily convert it into code, as we will be writing the pseudocode while teaching.

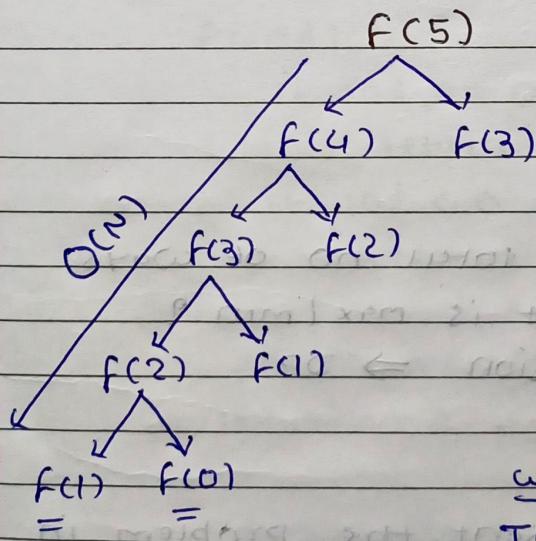
You can also find notes in the description of all the videos so that you can easily revise.

1) Tabulation - Bottom up

2) memoization - top down.

Overlapping subproblem :- It is part of our DP problem and need to compute again and again. like for $\text{fibonacci}(5)$ we will need $\text{fibonacci}(2)$ and then also for $\text{fibonacci}(6)$ we will be needing $\text{fibonacci}(2)$, so we will compute once and will store it. that is called memoization.

(1) Fibonacci number :-



with recursion :-

If we use memoization then we will not recalculate some already calculated values so,

TC: $O(N)$

SC: $O(N) + O(M)$

Rec.
stack

Array of DP.

Iterative method:-

then SC will be $O(N)$ only
 cause there will not be any recursion
 stuck.

This is promethod

$\text{prev2} = 0$ And $\text{SC} = O(1)$

$\text{prev1} = 1$

for $i=2$ to $i=n$

$\text{curr} = \text{prev1} + \text{prev2}$

$\text{prev2} = \text{prev1}$

$\text{prev1} = \text{curr}$

print(prev1)

(*) Short tricks:-

when, count total no of ways

so what is max/min?

will use Recursion \Rightarrow DP

How?

\rightarrow Try to represent the problem in terms of index

\rightarrow Do all possible stuff on index according to problem.

\rightarrow count all way = sum of all stuff

find min/max = min/max of all stuff

*) Max sum of non-adjacent elements.

Input : { 5, 5, 10, 100, 10, 5 }

Output : 110

How ?

We are not allowed to choose two adjacent elements

so to get max, we will select { 5, 100, 5 }
index 2 4 6

Approach 1 :-

Layout all possible sub-sequence.

F(ind)

if ind == 0

return a[ind]

} means we haven't
pick ind I so
pick at index 0

if ind < 0

return 0

Pick = a[ind] + F(ind-2)

} can't pick adj
so ind-2

not_Pick = a[i] + F(ind-1)

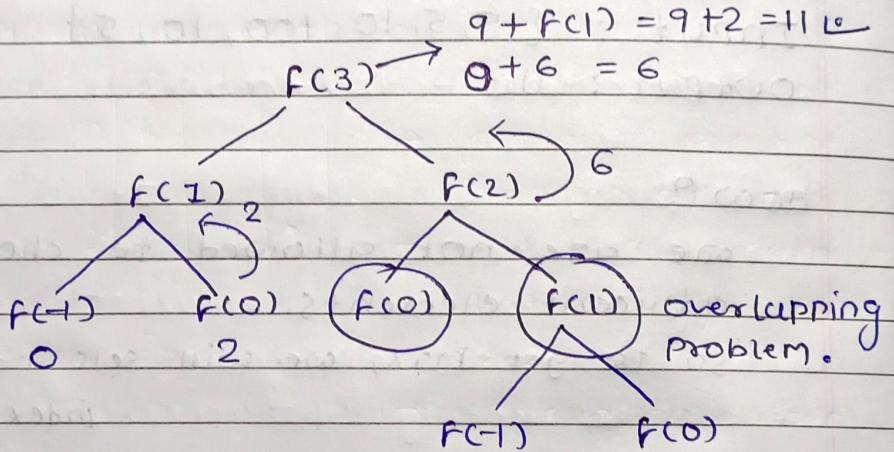
} can pick adj
so ind-1

return cmax(pick, not_Pick) } returning
max of it.

→ F(i) says that max sum from index 0 to i
if you do not pick up any adjacent

lets take example :-

$$a = \{2, 1, 4, 9\}$$



for $f(2)$

$$\begin{aligned} \text{pick} &= 4 + f(0) = 4 + 2 = 6 \\ \text{not-pick} &= 0 + f(1) = 2 \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{Myx} \rightarrow 6$$

so answer is 11

at $f(3)$ we have two option,

if we pick at k index 3

$$\text{then } 9 + f(1) = 9 + 2 = 11$$

else we do not

$$\text{then } 0 + f(2) = 0 + 6 = 6$$

and it is myx here.

$$\text{so } TC := O(n)$$

$$SC := O(n) + O(n)$$

It's taking $O(n)$ space for recursive stack,
let's do tabulation.

some(a)

$$n = a.size$$

dp[n]

$$dp[0] = a[0]$$

for $i=1$ to $i=n-1$

$$take = a[i]$$

if $i > 1$

$$take = take + dp[i-2]$$

$$not-take = dp[i-1]$$

$$dp[i] = \max(take, not-take)$$

return $dp[n-1]$

Optimized the best approach

Here we can see that we just need the last two state ($dp[i-1], dp[i-2]$) we can store this in 2 variables and can omit the $O(n)$ space.

* DP on Grids :-

Total unique path :-

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | * | | |
| 1 | | | |
| 2 | | | * |

→ You are present standing at (0,0)

→ Find unique path that leads you to (2,2)

→ You can only move to downwards or rightwards.

How?

1. Represent index in terms of row no and col. no $\Rightarrow (i,j)$
2. Explore all path
3. Do action (sum, min, max etc) as per question.

Let's solve question now :

$f(i,j) \Rightarrow$ from cell (i,j) how many paths are there possible to reach $(0,0)$

So let's code

$f(i, j)$

if ($i == 0$, and $j == 0$)

return 1

if ($i < 0$ or $j < 0$)

return 0

$TC = O(2^{M \times N})$

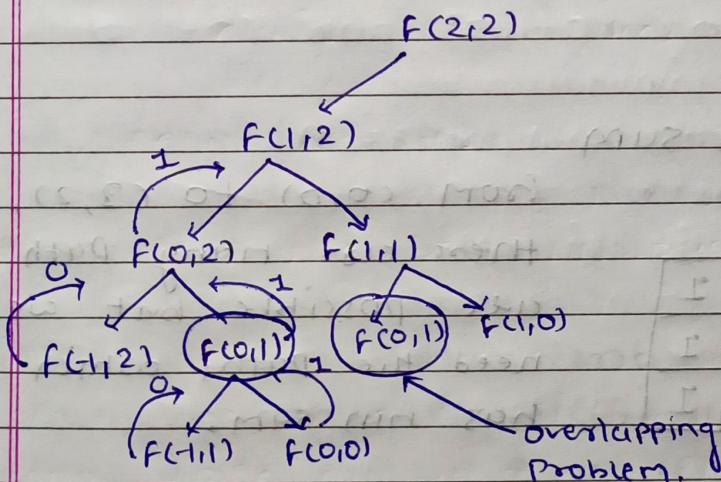
$UP = f(i-1, j)$

$SC = O(\max \text{ path length})$

$left = f(i, j-1)$

return $UP + left$

To improve TC we need to find that if there is overlapping problem or not?



So we have state i and j which will be useful for applying DP.

so we will just add $dP[i][j]$ to recursion

Memoization to tabulation :

STEPS:-

1. Declare base case
2. Express all states in loop
3. Copy recurrence relation

so

for $i=0$ to $i=m-1$

 for $j=0$ to $j=n-1$

 if $i=0$ and $j=0$

$dp[0][0] = 1$

 else if $i > 0$ and $j > 0$

$dp[i][j] = dp[i-1][j] + dp[i][j-1]$

*) Min path sum

from $(0,0)$ to $(2,2)$,
 there are many path
 are possible but we
 need the path, which
 has min sum.

| | | |
|---|---|---|
| 1 | 3 | 1 |
| 2 | 5 | 1 |
| 4 | 2 | 1 |

obviously it's grid DP as we can see

NOTE:- Here we are not counting the path
 instead of we are finding min/max
 value, so on out of bound index case
 there will be different logic.

solve(i, j, dp, grid)

if $i == 0$ and $j == 0$

return $grid[0][0]$

if $i < 0$ or $j < 0$

return INT_MAX

if $dp[i][j] \neq -1$

return $dp[i][j]$

int left = $grid[i][j] + solve(i, j-1, dp, grid)$

int up = $grid[i][j] + solve(i-1, j, dp, grid)$

return min(left, up)

\Rightarrow tabulation is easy system two for loops

\Rightarrow To optimize space, we just need to use two array of cols size prev and curr.

* Triangle Path Problem.

2

given a triangle

3 9

find min path from top

6 5 7

to bottom

4 1 8 3

at ith index, there will be ith elements.

You can move down to next row or right diagonally to next row.

that is,
can move to index i or $i+1$
into next row.

⇒ let's start.

Here greedy solution fails
because value distribution is
not uniform (sorted).

so we will tryout all path and
find the minimum.

→ so we can start from $(0,0)$
go to $(0+1, 0)$ and $(0+1, 0+1)$
take minimum
base case will be $i = n$

solve $(i, j, \text{tei}, \text{dp})$

if $(i == n-1)$ return $\text{tei}[i][j]$

if $(\text{dp}[i][j]) (= -1)$ return $\text{dp}[i][j]$

down = $\text{tei}[i][j] + \text{solve}(i+1, j, \text{tei}, \text{dp})$

diag = $\text{tei}[i][j] + \text{solve}(i+1, j+1, \text{tei}, \text{dp})$

return $\text{dp}[i][j] = \min(\text{down}, \text{diag})$

==

Tabulation

last row is our base case so will start from that as $DP[n-1] = TC[n-1]$

then

for $i = n-2$ to $i = 0$

$j = i + 1$ to $j = 0$

down = $TC[i][j]$

+ $DP[i+1][j]$

diag = $TC[i][j]$

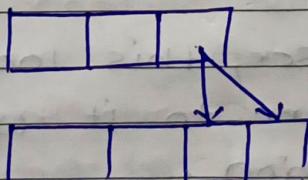
+ $DP[i+1][j+1]$

$DP[i][j] = \min(\text{down}, \text{diag})$

and our answer will be at $DP[0][0]$.

Space Optimization

If we notice we just need curr row that is for present process.



and pre. row to look up so we can do with just one array and one temp array.

* Maximum Path sum :-

$$\begin{matrix} 10 & 2 & 3 \\ 3 & 7 & 2 \\ 8 & 1 & 5 \end{matrix} \quad \text{and} \quad 10 \rightarrow 7 \rightarrow 8 \quad = 25 \text{ is max}$$

You are at first row, can start from from a cell you can move any column.

down - arrow

down left diagonal

down right diagonal

Find out the max path.

\Rightarrow As usual greedy will not work because values are not sorted.

So we need to try out all path and find the max one.

If we found sub problem overlapping then we need DP.

\Rightarrow As we can see, this looks like path sum, we can do it with recursion and memoization.

so we will start with) starting (3)

$f(i, j)$ ^{max} path from (i, j) cell to
last row

so base case : $i == \text{last row}$

out of $j < 0$ or $j > n$.

bound case

so,

$f(i, j, n, \text{grid}, \text{dp})$

if $j > n$ or $j < 0$ return -10^9

if $i == n$ return 0

if $\text{dp}[i][j] \neq -1$ return $\text{dp}[i][j]$

down = $\text{grid}[i][j] + f(i+1, j, n, \text{grid}, \text{dp})$

d-left = $\text{grid}[i][j] + f(i+1, j-1, n, \text{grid}, \text{dp})$

d-right = $\text{grid}[i][j] + f(i+1, j+1, n, \text{grid}, \text{dp})$

return $\text{dp}[i][j] = \max(\{\text{down}, \text{d-left}, \text{d-right}\})$

NOTE: Here starting point is not fixed so,

for $i=0$ to $i=\text{cols}$

$f(0, i, n, \text{grid}, \text{dp})$

* chocolate pickup (3D DP)

problem:-

you have $R \times C$ grid, having chocolates at each cell, two friend start picking chocolate from $(0,0)$ and $(0,C)$ cell. you want to pick max chocolate, find the max no of chocolates they can pick by reaching to last row.

= conditions:-

they can move to

right down

left down

right down

If both are in same cell then only one will pick the chocolate.

Intuition:-

As we can see it's path problem
grid DP will be solution for this.

we will proceed both friend simultaneously because we need to check that if both are picking from same cell then they are only one allowed to take chocolate.

Here for simultaneously movement

when A friend is moving down, B can move to down, left-down or right down

this is how for 1 movement of A, B can make 3 movement.

because we are not sure that both are making same movement.

but we are sure that both will have same row at a time.

so let's start :-

$f(i, j_A, j_B)$:- max chocolate can obtain when friend A is in (i, j_A) cell and friend B is in (i, j_B) cell.

base case :- $i == n - 1$

if $j_A == j_B$

return $\text{grid}[i][j_A]$

else

return $\text{grid}[i][j_A] + \text{grid}[i][j_B]$

so, it will go like this:

$f(i, j_A, j_B, \epsilon, c, \text{grid}, dp)$

if $j_A < 0$ or $j_B < 0$ or

$j_A > c$ or $j_B > c$

return -10^8

if $i == \sigma - 1$

if $j_A == j_B$

return $\text{grid}[i][j_A]$

else

return $\text{grid}[i][j_A]$

+ $\text{grid}[i][j_B]$

if $dp[i][j_A][j_B] \neq -1$ return $dp[i][j_A][j_B]$

int maxI = -10^8

for $j_1 = -1$ to $j_1 = 1$

for $j_2 = -1$ to $j_2 = 1$

if $j_A == j_B$

value = $\text{grid}[i][j_A]$

else

value = $\text{grid}[i][j_A]$

+ $\text{grid}[i][j_B]$

value += $f(i+1, j_A + j_1, j_B + j_2, \epsilon, c,$
 $\text{grid}, dp)$

maxI = $\max(maxI, value)$

return \maxI $dp[i][j_A][j_B] = \maxI$

Note :-

Here we can go to,
 down : $(i, j) \Rightarrow (i+1, j+0)$
 down-left : $(i, j) \Rightarrow (i+1, j-1)$
 down-right : $(i, j) \Rightarrow (i+1, j+1)$

so we need for loop of $i = 1$ to n .

* DP on subset / subsequences :-

* subset sum (equal to k) :-

You are given an array and integer k , you need to check that is there exists a subset of array, with sum equal to k .

Eg:- $\{1, 2, 3, 4\}$ $k = 4$

yes:- $\{1, 3\}$ and $\{4\}$

Output: true.

Let's start:-

First approach come in mind is generate all the subsequences.

We can do using power set or recursion

Here powerset generates all, but we need only one subset so we will follow the recursion.

so we can start like this.

- (1) $f(\text{ind}, \text{target})$
- (2) EXPLORE all possibility.
 - (i) ind is part of subseq.
 - (ii) ind is not part of subseq.
- (3) return T/F.

Here $f(n-1, \text{target})$ says that in the entire array till $n-1$ from 0, subset sum target exists or not.

Base case :-

- $\text{target} == 0$ return true;
- if $\text{ind} == 0$
 - return $\text{target} == \text{arr}[0]$
- we are decreasing target so
- $f(0, \text{target})$ says that till 0 is target sum exists or not.

Exploring possibilities:-

- take = false

- if target <= arr[инд]

- take = F(инд-1, target - arr[инд])

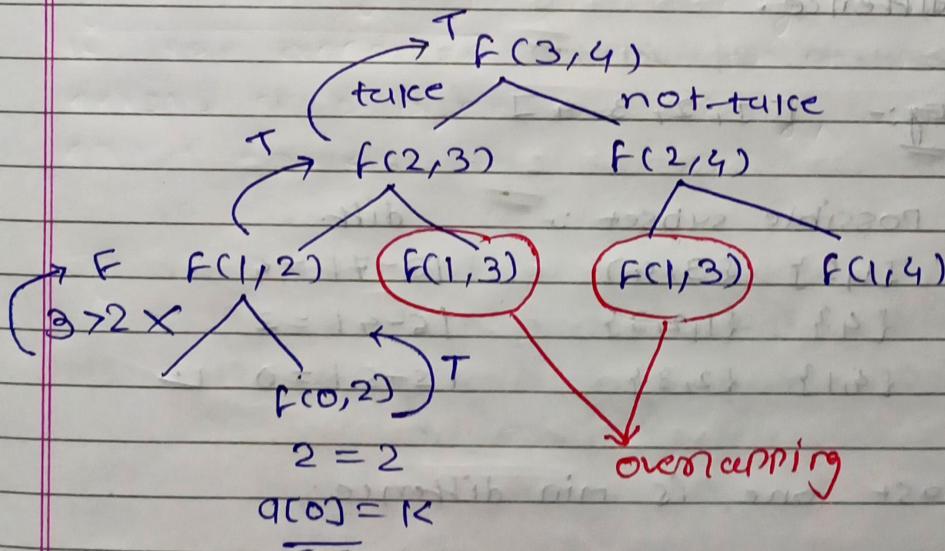
- not-take = F(инд-1, target)

returning:-

- return take or not-take

⇒ Now let's draw Rec-tree:-

2, 3, 1, 1
[1, 2, 3, 4] K = 4



For Recursion:-

Tc : $O(2^n)$ Recursion

Sc : $O(n)$ ← stack space

But we can apply memoization
states : ind, target.

so TC :- $O(N \times \text{target})$

SC :- $O(N) + \underline{O(N \times \text{target})}$

$\frac{O(N \times \text{target})}{\approx}$

(*) Partition a set into two subsets
such that difference of subset
is minimum.

You just need to find the minimum
difference.

Ex :- [1, 2, 3, 4]

Possible subset :- diff.

$$\{1, 2\} \quad \{3, 4\} \quad |3 - 7| = 4$$

$$\{4\} \quad \{1, 2, 3\} \quad |6 - 4| = 2$$

$$\{4, 1\} \quad \{2, 3\} \quad |5 - 5| = 0 \quad \checkmark$$

last one is min difference

so output :- 0.

For this we will have s_1 and s_2 and we want to make $|s_1 - s_2|$ as minimum as possible.

So, we will find all possible s_1 and $s_2 = \text{total} - s_1$ and find the min difference.

so we need to have $\text{dp}[n][\text{total}]$
here $\text{total} = q_1 + q_2 + \dots + q_n$.

so,

for $i=0$ to $i=\text{totalSum}/2$

if $\text{dp}[n-1][i] == \text{true}$

$\text{ans} = \min(\text{ans}, \text{abs}(s_1 - (\underline{\underline{\text{total}}} - s_1)))$

Here we take $\text{totalSum}/2$ because
for $\{1, 2, 3, 4\}$

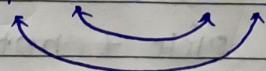
$$\text{total sum} = 10$$

so

s_1 1 2 3 4 5 6 7 8 9 10

s_2 10 9 8 7 6 5 4 3 2 1 0

$|s_1 - s_2|$ 10 8 6 4 2 0 2 4 6 8 10



same. so,

Here $\text{dp}[n-1][i]$ is that from 0 to $n-1$ is subset sum = i exist or not?

Note :-

while writing recurrence if we told to find count or something then,

i) return 1 for right case

return 0 for fail case

ii) sum of all possible operation

like,

if $\text{ind} == 0$

if $\text{sym} == \text{arr}[0]$

return 1

return 0

if $\text{sym} == 0$

return 1

$\text{pick} = \text{solve}(\text{ind}-1, \text{sym} - \text{arr}[\text{ind}])$

$\text{notpick} = \text{solve}(\text{ind}-1, \text{sym})$

return $\text{pick} + \text{notpick}$

* 0/1 knapsack

You are given weight and value of items and knapsack capacity, find the max value can get.

Eg:- wt :- 3 4 5 capacity = 8
 value :- 30 50 60

$$\begin{array}{r} \text{item selected :- } \\ \begin{array}{r} 3 \ 5 \\ 30 \ 60 \end{array} \end{array} = \frac{8 < 9}{30 + 60 = 90}$$

We can try out all possible ways and select the max.

so $f(\text{ind}, w)$:- till index ind what the max value you will get with weight w remaining)

→ and we will proceed with take and not take,

→ while taking the item, we need to check either item weight is less or equal to bag capacity.

→ we will start from $f(n-1, w)$ so base case:
 if $\text{ind} == 0$

if $\text{weight}[0] \leq w$
 return $\text{value}[0]$

return 0;

Here we will do recursion that will take $O(2^n)$ time

Now, we can optimize it using memoization

our changing state is

- ind
- weight

Recursive DP code is easy
for that:

Time complexity :- $O(n \times w)$

Space complexity :- $O(n \times w) + \underline{O(n)}$
space

so let's do tabulation:-

1. we will have $dP[n][w] = \{0\}$

2. Fill value for base case.

base case. { for every $w \leq \text{capacity}$ and $w \geq w[0]$
 $dP[0][i] = val[0]$

3. do nested loop for states.

4. copy or write recurrence.

so let's code it:-

knapsack(n, maxw, weight[], value[])

$$dp[n][maxw+1] = \{0\}$$

for $i = weight[0]$ to $i = maxw$,

$$dp[0][i] = value[0]$$

for $i = 1$ to $i = n-1$

for $w = 0$ to $w = maxw$

$$notTake = dp[i-1][w]$$

$$take = -10^8$$

if $w \geq weight[i]$

$$take = value[i] + dp[i-1]$$

$[w - weight[i]]$

$$dp[i][w] = \max (notTake, take)$$

return $dp[n-1][maxw]$

so If we talk about

time complexity :- $O(n \times maxw)$

space complexity :- $O(n \times maxw)$

we can omit the space complexity to $O(maxw)$ by using 1D DP, 2 array, prev and curr.

* coin change:-

We are given array of some distinct coins, every coin has infinite supply and given target, we need to use minimum no of coin as possible and print that coin number count.

Eg:- $\{1, 2, 3\}$ target = $\{7\}$

Ans:- $\{3, 3, 1\} \Rightarrow 3$

Here greedy comes for many of case but fails sometime.

Eg: $\{9, 5, 6, 1\}$ target = 11

greedy = $\{9, 1, 1\} \Rightarrow 3$

but right is 2 $\Rightarrow \{5, 6\}$.

so we will try out all possible path and choose the minimum one.

so, $f(ind, target)$:- How many coins required to form 'target' using '0 to ind ' coins.

so possible cases:-

tuice = INT_MAX

if coins[ind] <= target

 tuice = i + f(ind, target - coins[ind])

not tuice = f(ind-1, T)

returning required value :-

return min(tuice, nottuice)

base case:-

we will start with $f(n-1, \text{target})$

so

if ind == 0

 if (target % coins[ind] == 0)

 return target / coins[ind]

 else

 return INT_MAX

It is like we have $\text{coin}[0] = 4$

and $\text{target} = 12$

then we need $12/4 = 3$ coin

this possible because we can make 12 using 4 coin.

but if $\text{target} = 11$

then we can not so return INT_MAX

let's solve it:-

solve c coins, x)

$n = \text{coins.size}()$

$dP[n][x+1] = \{0\}$

for $t=0$ to $t=x$

if $t < \text{coins}[0] == 0$

$dP[0][t] = t / \text{coins}[0]$

else

$dP[0][t] = 10^9$

for $\text{ind}=1$ to $\text{ind}=n-1$

for $\text{target}=0$ to $\text{target}=x$

$\text{truce} = dP[\text{ind}-1][\text{target}]$

$\text{nottrue} = 10^9$

if $\text{target} \geq \text{coins}[\text{ind}]$

$\text{nottrue} = 1 + dP[\text{ind}]$

$[\text{target} - \text{coins}[\text{ind}]$

$dP[i][\text{target}] = \min(\text{truce},$

$\text{nottrue})$

return $dP[n-1][x]$

so time complexity = $O(x) + O(n \times x)$

space complexity = $O(n \times x)$

* Unbounded Knapsack:-

Given Profit and weight arrays, capacity you need to get max profit using this capacity but here you have unlimited source of item, that is you can pick any item, any time you want.

Same as previous knapsack problem but here we have unlimited resource of item so there will be slight change in base case and recursion.

base case:-

if $cind == 0$

return $(\omega / \text{weight}[0]) * \text{profit}[0]$

recursion

$\text{fclce} = \text{profit}[cind] + f[cind, \underline{\omega - \text{weight}[cind]}]$

This is same as coin change as we had unlimited coins.

so when we have unlimited supply, they while taking don't change index.

(*) Rod cutting

you are given an rod length and price for each length piece. you need to get max possible price.

$$\text{Eg:- } N = 5, \{2, 5, 7, 8, 10\}$$

$$\text{Ans} = \{2, 3\} \Rightarrow 5 + 8 = 12 \text{ is max.}$$

| | | |
|---|-----|-------------|
| 2 | 3 | 1 |
| 5 | + 7 | <u>= 12</u> |

It looks somewhat same as knapsack, where we have weights of 1 to N and want to make $w = N$, using this weight which gives max value.

also we have infinite supply. (of rod lengths)

Solve(ind, N)

if ind == 0

return price[0]

notTake = solve(ind-1, N)

take = -10^9

if ind+1 <= N

take = price[ind] + solve(ind, N-(ind+1))

return max(take, notTake)

Note:- Here we have/can take rod of same length for more than one time.
so while taking we are not doing $ind-1$.

$$\overbrace{abcd}^{\text{one rod}} \leftarrow \overbrace{ab\overset{c}{|}d}^{\text{one rod}} \rightarrow \overbrace{a\overset{b}{|}cd}^{\text{one rod}}$$

$$E = H_2 + \frac{1}{2}$$

$$\overbrace{abcd}^{\text{one rod}} \leftarrow \overbrace{a\overset{b}{|}cd}^{\text{one rod}}$$

$$2H_1 b \text{ rods}$$

$$\overbrace{abcd}^{\text{one rod}} \leftarrow \overbrace{a\overset{b}{|}cd}^{\text{one rod}}$$

$$5 \text{ rods}$$

$$H_1 p - 1 \text{ rod}$$

$$-1$$

$$5 \text{ rods} - 1 \text{ rod}$$

$$4 \text{ rods}$$

$$5 \text{ rods} - 1 \text{ rod}$$

$$-1$$

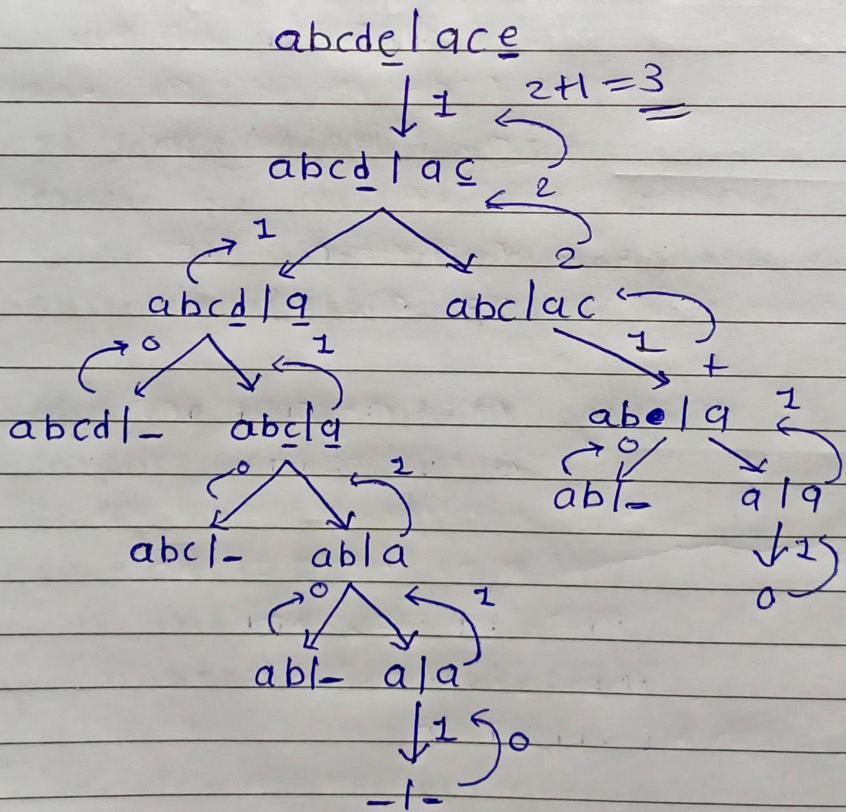
So we can add all rods

DP on string :-

longest common sub-sequence.

Given two strings you need to print the string that is longest and sub-sequence in both.

Eg:- abcde \Rightarrow ace.



this is how we will do.

start comparing from last
if char is same at that index
go to next/prev index for both
and add 1

else
first go for one string I prev.
then go for another string I prev
return max of them.

so we will have two index:

ind1 { this is state for
ind2 { our DP

so

```
f(ind1, ind2)
    if ind1 < 0 or ind2 < 0
        return 0
    if s1[ind1] == s2[ind2]
        return 1 + f(ind1-1, ind2-1)
```

else

```
    OP1 = f(ind1-1, ind2)
    OP2 = f(ind1, ind2-1)
    return max(OP1, OP2)
```

now we can add DP for our states
ind1, ind2.

For tabulation we will need to do index shifting.

because in recursion base case is for $\underline{\text{ind} < 0}$, so -1, but table can not have minus index

so,

base case will be $\underline{\text{ind} = 0}$

so,

SOLVE ($S1, S2$)

$$\text{dp } n = S1.\text{length}()$$

$$m = S2.\text{length}()$$

$$\text{dp}[n][m] = \{0\}$$

for $i=0$ to $i=n$

$$\text{dp}[i][0] = 0$$

} base case

for $j=0$ to $j=m$

$$\text{dp}[0][j] = 0$$

for $i=1$ to $i=n$

for $j=1$ to $j=m$

if $S1[i] = S2[j]$

$$\text{dp}[i][j] = i + \text{dp}[i-1][j-1]$$

else

$$\text{dp}[i][j] = \max(\text{dp}[i-1][j], \text{dp}[i][j-1])$$

return $\text{dp}[n][m]$

Note:- Here we have discussed about subsequence, but if we want to do same for substring then second case will be unusable.

so,

for $i=1$ to $i=n$

for $j=1$ to $j=m$

if $s1[i] == s2[j]$

$dP[i][j] = 1 + dP[i-1][j-1]$

else

$dP[i][j] = 0$

$ans = \max(ans, dP[i][j])$

now to print LCS, we need dP array
then,

$i=n, j=m, ans = ""$

while ($i>0$ and $j>0$)

if $s1[i-1] == s2[j-1]$

$ans = ans + s1[i-1]$

else if $dP[i-1][j] > dP[i][j-1]$

$i--$

else

$j--$

$\text{reverse}(ans)$

$\text{print } ans$

* Longest palindromic subsequence

Given a string, find out the longest palindromic subsequence.

Eg:- bbbab \Rightarrow 4 (bbbb)

Eg:- cbbd \Rightarrow 2 (bb)

If we observe ans is LCS of s and reverse of s
that is,

LCS of { bbbab \Rightarrow 4 (bbbb)
 babb } \leftarrow reverse of s.

so,
solve c s)

string $s_1 = s$

reverse(s.begin(), s.end())

string $s_2 = s$

return $\text{LCS}(s_1, s_2)$;

=.

* Minimum insertion required to make string Palindrome:

Given a string s , you need to make it Palindrome with minimum insertion of alphabets.

Eg:- $\underline{abcba} \Rightarrow 2(99bcba9)$

So we will make longest Palindrome constant and make some insertion for remaining.

So for $abcba$ = longest Palindrome is $a\bar{c}a$

so,

a b c q q we need pair for 'b' and 'q'

a b c bqq inserted 'b'

a q b c b a q inserted 'q'

aa b c b a q now,
 ← Palindrome.

this is how we need to add

$\text{length}(s) - \text{LCS}(s, \text{reverse}(s))$

characters.

=

(*) Shortest common supersequence:-

You are given two strings s_1 and s_2 .
You need to construct the shortest string that contains s_1 and s_2 as subsequences.

Eg:- $s_1 = \text{blue}$

$s_2 = \text{root}$

Output:- $\text{b} \circ \text{g} \circ \text{r} \circ \text{o} \circ \text{t} \circ \text{e}$

It contains both as ^{sub-}sequence

So what we will do is

i) find out LCS

ii) write down s_1 by substituting LCS from it

iii) do same for s_2

iv) now write LCS.

let's make LCS DP table

| | | | | | |
|---|---|---|---|---|---|
| | g | x | o | o | t |
| o | 0 | 0 | 0 | 0 | 0 |
| b | 0 | 0 | 0 | 0 | 0 |
| g | 0 | 0 | 1 | 1 | 1 |
| y | 0 | 0 | 1 | 1 | 1 |
| t | 0 | 0 | 1 | 1 | 1 |
| e | 0 | 0 | 1 | 1 | 1 |

so, $etoxgozb^g$

- If cell has same value write once, else write both.
- Reverse it to get original string (\Rightarrow)
- $\Rightarrow gbozoyte \Leftarrow$

so,

solve (a, b)

LCS (a, b)

DP[n+1][m+1]

$i = n, j = m$
ans = " "

while $i > 0$ and $j > 0$

if $a[i-1] == b[j-1]$

ans = ans + $a[i-1]$

else if $dP[i-1][j] > dP[i][j-1]$

$\text{ans} = \text{ans} + a[i=1]$

$i--$

else

$\text{ans} = \text{ans} + b[j-1]$

$j--$

while ($i > 0$) $\text{ans} += a[i-1]$, $i--$

while ($j > 0$) $\text{ans} += b[j-1]$, $j--$

reverse (ans)

return ans .

(*) Distinct subsequences.

given two strings s_1 and s_2 , want to know how many distinct subsequences of s_2 are present in s_1 .

Eg.: $s_1 = \text{babgbag}$

$s_2 = \text{bag}$

Output :- 5

babgbug

babgbug

babgbag

babgbug

babgbug

so

we will start from tail

if $s1[ind] == s2[ind]$

then both pointer will be decreased
but one time $s2$ pointer will be same.

else

one time decrease one pointer of $s1$.

one time decrease other pointer.

base case will be

if $s1$ is exhausted return 0

if $s2$ is exhausted,

means we have found $s2$ in $s1$

so return 1.

states:- both pointer

so,

$solve(i, j, s1, s2, dp)$

if $i < 0$ return 0

if $j < 0$ return 1

if $dp[i][j] \neq -1$

return $dp[i][j]$

if $s1[i] == s2[j]$

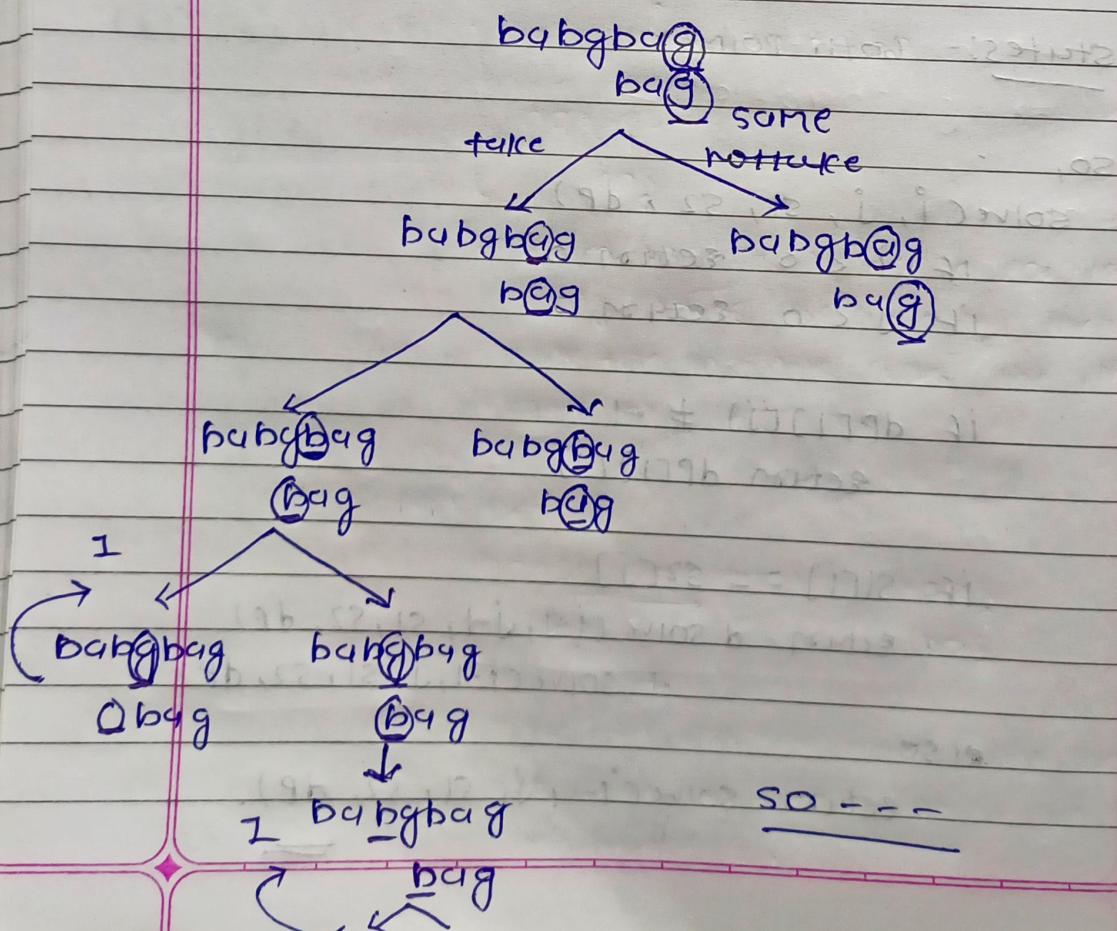
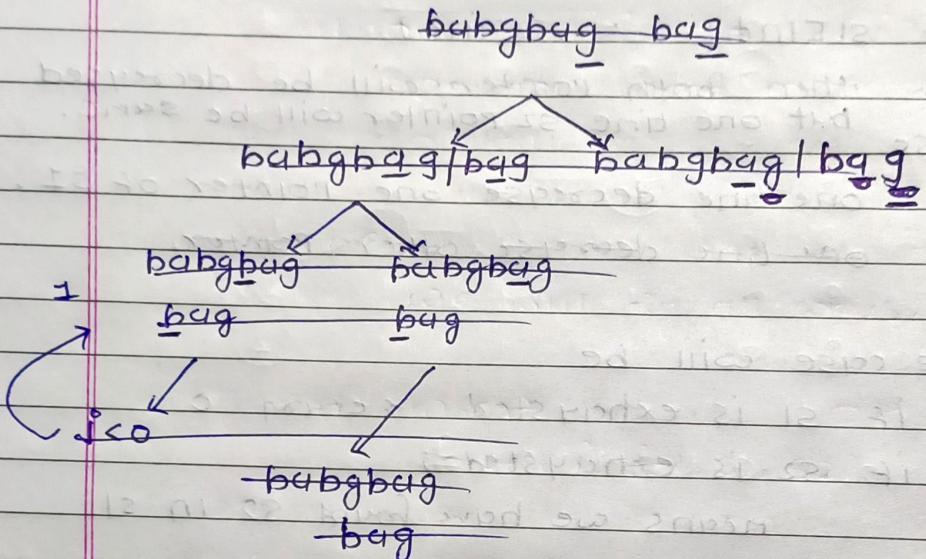
return $dp[solve(i-1, j-1, s1, s2, dp)]$

+ $solve(i-1, j, s1, s2, dp)$

else

return $solve(i-1, j, s1, s2, dp)$.

Eg:- babgbag bag



(*) Edit distance:-

We are given string s_1 and s_2 . We need to convert s_1 into s_2 . Following three operation is allowed

- i) Deletion of character
- ii) Replacement of character with other character
- iii) Insertion of character.

We have to return minimum number of O/P.

→ So we have 3 operation, we will try all combination and then will find min.

So it's about recursion.

Eg:- $s_1 = \text{horse}$ OP:- 3
 $s_2 = \text{ros}$

$\text{horse} \rightarrow \text{orse} \rightarrow \text{ose} + \text{ros}$

So we will follow three step:-

(i) Express everything in terms of index.

so,

$f(i, j) \Rightarrow$ min no of operation required
to convert $s_1[0:i]$ to $s_2[0:j]$

(ii) Tryout all possible way :-

(i) when $s1[i] == s2[j]$

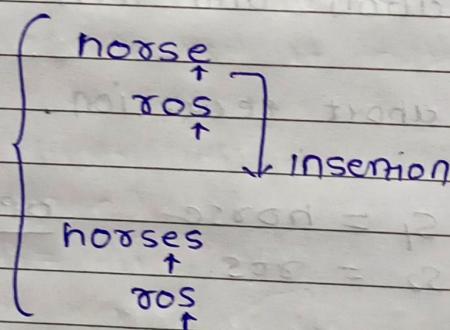
we don't need to perform any operation.

else,

we will try all ways,

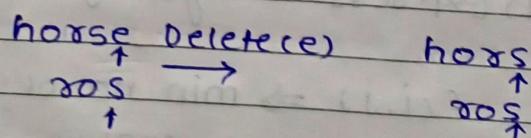
(i) insert

some ($i, j-1, s1, s2$)



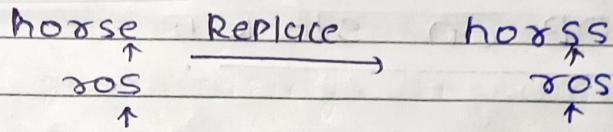
→ After insertion second pointer only moves, because hypothetically we are inserting after the i th pointer.

(ii) Delete



→ here i th pointer moves only because we have no match for s , we are still finding s .

(iii) Replace



We will replace with desired char
and move both pointer

so,

sove $C[i-1, j-1, s_1, s_2]$

(iii) return min or all.

now base case:-

(i) when $i < 0$

so s_1 has got exhausted.

hotos we have as s_1

$s_2 = \text{horoso}$

we will need to insert need character,
as our source has got empty now.

so return $j+1$ (oth index so +1)

(ii)

when $j < 0$

we got our s_2 in s_1 , but still having
some extra char in s_1 , we need
to delete them

so return $i+1$

(*) DP on stock 1:

(i) you are given prices, you can perform one transaction only, find-out the max profit you can make.

Eg:- [7, 1, 5, 3, 6, 4] Opt - 5 (-6)

we can max the profit by buying at min price and sell it after max price so,

solve (price)

$$\min P = \text{Price}[0]$$

$$\text{Profit} = 0$$

for $i=1$ to $i < \text{Price.size()}$

$$\text{cost} = \text{Price}[i] - \min P$$

$$\text{Profit} = \max(\text{Profit}, \text{cost})$$

$$\min P = \min(\min P, \text{Price}[i])$$

return profit.

=

(ii) on each day you can buy or sell, but can not sell before buy or can not buy after buy immediately. you have to sell before buy.

Here we have many t_i available and we have one buy-sell constraint.

so we will try all possible combination of buy and sell then find out max.

see while buying our profit will be

$$\Rightarrow -\text{Price}[i]$$

STEPS:-

$f(\text{ind}, \text{buy}) \Rightarrow$ max profit we can make at till index ind and can buy or not on curr day.

if $\text{buy} = 1$, we have to buy or not buy, but can not sell for sure

when $\text{buy} = 0$, we have to sell, we can not buy. we can also hold though.

(iii) All possibility:-

if (buy == 1)

$$\text{buy} = \text{solve}(i+1, 0) - \text{price}[i]$$

$$\text{notbuy} = \text{solve}(i+1, 1)$$

else

$$\text{sell} = \text{solve}(i+1, 1) + \text{price}[i]$$

$$\text{notsell} = \text{solve}(i+1, 0)$$

(iv) find out max profit

$$\text{profit} = \max(\text{buy}, \text{notbuy})$$

$$\text{profit} = \max(\text{sell}, \text{notsell}).$$

base case:-

when $i == n$

return 0;

now we do not have any day left so,
0.

(v) Almost two transaction:-

add one more state to the previous question that is cap = 2.

while selling decrease it by 2, because we have completed one transaction

and add one more byte case:-

if cur == 0 return 0;
that's it.

[101, F, E, S] → -10H

[81, F, E, S]

-10H → 00H

00H → -10H

00H → -10H

↓

addition

10H + 10H = 20H → 00H

00H + 10H = 10H → 00H

(*) DP on longest Increasing subsequence

(i) longest increasing subsequence.

Given an array, you need to return
max length of increasing subsequence.

Eg:- [10, 9, 2, 5, 3, 7, 101, 18]

Ans:- 4 [2, 3, 7, 101]

or

[2, 3, 7, 18]

so, subsequence



try all path



recursion



memoization

Here while constructing LIS, we look to previous guy, that is previous guy is smaller than current then add curr to our answer.

so its states are:-

current index

prev index.

so $F(ind, prev_ind)$

\hookrightarrow Length of LIS starting from ind , where the last index that is part of LIS is $prev_ind$.

Now we will explore all possibility:-

that is take, nottake

(i) If we does not take curr index as our subsequence part, then our previous index will not change.

so, $F(ind+1, prev_ind)$

(ii) when we can take curr index?

if $a[curr_index] > a[prev_index]$

our starting $prev_index$ will be -1 when.

so, if $prev_ind == -1$ or

$a[ind] > a[prev_ind]$

$takce = 1 + F(ind+1, ind)$

Here our length will increase by one, and our previous index will be update so.

Now return max of these.

Now, base case will be when $ind == a.size()$

so IF $ind == a.size()$
 return 0;

Now for memoization we need to take

$dp[n][n+1]$

$n = a.size()$

Here $n+1$, because -1 can't be index
 so we will add $+1$ to it always.

Solve(ind , $prev_ind$, a , dp)

if $ind == a.size$

return 0

if $dp[ind][prev_ind+1] != -1$

return $dp[ind][prev_ind+1]$

nottake = solve($ind+1$, $prev_ind$, a , dp)

take = 0

if ($prev_ind == -1$ or

$a[ind] > a[prev_ind]$

take = 1 + solve($ind+1$, ind , a , dp)

return $dp[ind][prev_ind+1] = \max(take,$

nottake)

Here we have done $i \rightarrow 0, 0 \rightarrow 1, 1 \rightarrow 2, \dots, n-1 \rightarrow n$ increment change for prev-ind that is called co-ordinate change.

$$TC := O(n \times n)$$

$$SC := O(n \times (n+1)) + O(n)$$

We can do tabulation by,

$$dp[n+1][n+1]$$

for $i = n-1$ to $i = 0$

for prev = $i-1$ to $i = -1$

{copy recurrence.}

$$\text{return } dp[0][0]$$

After this space optimization will be there

that is,

using two array of

after[n+1]

prev[n+1]

But we can do it in 1D DP also
let's see it.

* Largest Divisible subset

Given a set of distinct positive integers, return the largest subset such that for every pair (i, j)

$$\text{ans}[i] \times \text{ans}[j] = 0$$

or

$$\text{ans}[i] \times \text{ans}[j] = 0$$

Eg:- $\{1, 16, 7, 8, 4\}$

$$\text{Ans} = \{1, 16, 8, 4\}$$

so we will have to sort it

as $1 \ 4 \ 7 \ 8 \ 16$

so, if we pick $\{1, 4, \dots\}$

and looking for $\{8\}$
then,

if 8 is divisible by 1st (1)

it is divisible by all others too

so we will loop through array,

actually this is same as LIS with some changes, we need to use tabulation method of LIS to solve this.

Solve(arr , n)

$$\text{dp}[n] = \{1\}, \text{hash}[n] = \{0\}$$

sort(arr)

for $i=0$ to $i=n$, $\text{hash}[i] = i$

for $j=0$ to $j < i$

if $\text{arr}[i] \times \text{arr}[j] == 0$

and

$$\text{dp}[i] < \text{dp}[j] + 1$$

$$\text{dp}[i] = \text{dp}[j] + 1$$

$$\text{hash}[i] = j$$

int ans = -1

int lastIndex = -1

for $i=0$ to $i < n$

if $\text{dp}[i] > \text{ans}$

$$\text{ans} = \text{dp}[i]$$

$\text{lastIndex} = i$

$\text{ans} = \{ \}$

$\text{ans.push_back}(\text{arr}[\text{lastIndex}])$

while $\text{hash}[\text{lastIndex}] (= \text{lastIndex})$

$\text{lastIndex} = \text{hash}[\text{lastIndex}]$

$\text{ans.push_back}(\text{arr}[\text{lastIndex}])$

return ans.

*) Longest string chain

you are given array of words return the length of longest chain.

SO IF there is w_1, w_2 two word then if we can insert any one char in w_1 and can make w_2 then it is chain.

Eg: a, b, ba, bca, bda, bdcg

OUTPUT:- 4

a, ba, bca, bdcg.

Here that is not necessary to have subsequence so will sort it based on length

then just adding one more condition to last problem and removing mod condition we are good to go

like,

we will take

'a', now if diff between (a, b) = 1 then take it else not

to check if the different is one or not

bca bdcg \Rightarrow same so both pointer
 \uparrow \uparrow ++

bca bdcg \Rightarrow not same then increment the largest pointer
 \uparrow \uparrow

bca bdcg on 1, found 201
 \uparrow \uparrow

bca bdcg
 \uparrow \uparrow

If they reach end at the same time then
possible else not

big small
 \downarrow \downarrow

so check (S1, S2)

$n = S1.size()$

$m = S2.size()$

if $n \neq m + 1$ return false

first = 0, second = 0

while (first < n)

if second & second $S1[first] == S2[second]$

first++

second++

else

first++

if first == n and second == m

return true

return false.

* longest bitonic subsequence :-

Given an array of integer, you need to print the longest length of bitonic subsequence.

Here bitonic means increasing and then decreasing.

Eg:- {1, 11, 2, 10, 4, 5, 2, 1}

Ans. \Rightarrow 6 {1, 2, 10, 4, 2, 1}

Here, It can be decreasing only or increasing only.

so we will find,

for index i

$I_DP[i]$:- how many number are in $0 \dots i-1$ which can be taken for LIS

$D_DP[i]$:- how many number are in $i+1 \dots n$ which can be taken for LIS.

now find max for $\forall i$

$$\max(D_DP[i] + I_DP[i] - 1)$$

= .

For I-dP[], we know the process

so for D-dP[]

$$\left\{
 \begin{array}{l}
 \text{for } i = n-1 \text{ till } i = 0 \\
 \text{for } j = i+1 \text{ to } n \\
 D-dP[i] = \max(D-dP[i], \\
 \quad \quad \quad \underline{I + D-dP[j]}) \\
 \end{array}
 \right.$$

(*) NO OF longest Increasing subsequences
 Given an array, return the number of LIS in that array.

Eg:- [1, 3, 5, 4, 7]

Answer :- 2 = (D + max)

How?

[1, 3, 4, 7] ↗

[1, 3, 5, 7] ↗

let's try for this below example using previous LIS problem knowledge.

| | | | | | | | | | | | | |
|---------|---|---|---|---|---|---|---|---|---|----|---|---|
| | 1 | 5 | 4 | 3 | 2 | 8 | 2 | 6 | 7 | 10 | 8 | 9 |
| dP[] | 1 | 2 | 2 | 2 | 2 | | | 2 | 3 | | | |
| COUNT[] | 1 | 1 | 1 | 1 | 1 | | | 2 | 4 | | | |

so on--

5
↗

there is 5 LIS that ends with 6

Here $dP[i]$:- longest LIS length
 $count[i]$:- no of LIS of $dP[i]$ length.

Solve arr, n)

$$dP[n] = \{1\}$$

$$count[n] = \{1\}$$

$$maxi = 1$$

for $i = 0$ to $n - 1$

for $j = 0$ to $i - 1$

if $arr[j] < arr[i]$

and $dP[i] < 1 + dP[j]$

$$dP[i] = 1 + dP[j]$$

$$count[i] = count[j]$$

else if $arr[j] < arr[i]$

and $dP[i] == 1 + dP[j]$

$$count[i] = count[i] + count[j]$$

$$maxi = \max(maxi, dP[i])$$

$$\text{int ans} = 0$$

for $i = 0$ to $n - 1$

if $dP[i] = maxi$

$$ans = ans + count[i]$$

return ans.

(*) Partition DP

Whenever we need to find the answer to a large problem such that the problem can be broken into subproblems and the final answer varies due to the order in which subproblems are solved, we can think in terms of partition DP.

1. Matrix chain multiplication :-

Problem :-

Given n matrix dimensions, return the min cost to multiply them to a single matrix.

$$\text{Eg:- } A = 10 \times 30$$

$$B = 30 \times 5$$

$$C = 5 \times 60$$

$$\text{so } (AB)C \Rightarrow 10 \times 30 \times 5 + 10 \times 5 \times 60 \\ = 4500$$

$$A(BC) \Rightarrow 10 \times 30 \times 60 + 30 \times 5 \times 60 \\ = 27000$$

$$\text{so, Ans} = \underline{\underline{4500}}$$

$ABCD$,

- $(AB)(CD)$ possible
- $(A(BC))D$ possible
- $A(B(CD))$ possible

Given $[10, 20, 30, 40, 50]$

so,

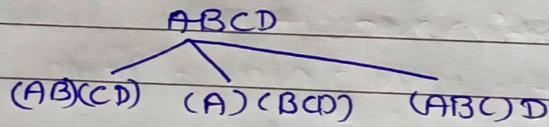
1st matrix : 10×20

2nd matrix : 20×30

ith matrix : $A[i-1] \times A[i]$

Rules :-

1. start with entire block/array.



i : start point.

j : end point.

2. Try all partitions.

$$ABCD \rightarrow (AB)(CD)$$

$$A|BCD \rightarrow (A)(BCD)$$

$$AB|CD \rightarrow (AB)(CD)$$

$$ABC|D \rightarrow (ABC)D$$

3. Return the best possible two partition.

$f(i, j)$:- Return the min multiplication required to multiply matrix i to matrix j chain.

i, j will shrink.

so base case:

if $i == j$ return 0

→ when i will be equal to j , there will be only one matrix, so we required 0 operations only.

Now to try all partition,

for $k = i$ to $k = j-1$ // Here $j-1$ because $f(i, k), f(k+1, j)$ there will be only one matrix at last.

so,

$$\text{steps} = A[i-1] * A[k] * A[j] + f(i, k) + f(k+1, j)$$

$$\text{mini} = \min(\text{mini}, \text{steps})$$

like,

10 20 30 40 50
 i, k j

$$\begin{aligned} \text{so } & \frac{(10, 20 \times 20, 30)}{\downarrow 1} \times (30, 40 \\ & = 10 \times 30 \quad \downarrow \times 40, 50) \\ & \quad \quad \quad \downarrow \\ & \quad \quad \quad 10 \times 30 \times 50 \end{aligned}$$

so,

solve($i, j, \text{arr}, DP[i][j]$)

if $i == j$

return 0

if $DP[i][j] != -1$

return $DP[i][j]$

int mini = 10^9

for $k = i + 1$ to $k = j - 1$

steps = $\text{arr}[i-1] \times \text{arr}[k] \times \text{arr}[j]$

+ solve(i, k, arr, DP)

+ solve($i+1, j, \text{arr}, DP$)

mini = min(mini, steps)

return $DP[i][j] = mini$

Time complexity : $O(N^3)$

2. Minimum cost to cut the streak (stick)

We are given a stick of length N and a cuts array. The stick has markings as shown and the cuts array depicts the marking at which the stick need to

be cut.

Eg:- CUTS: [1, 3, 4, 5]

| | | | | | | | |
|---|---|---|---|---|---|---|----------|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | CUTS[0] |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | cost = 7 |

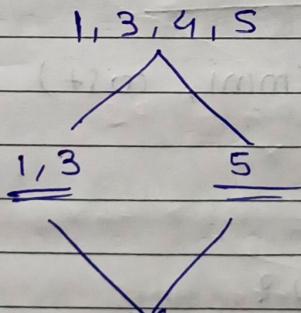
| | | | | | | | |
|---|---|---|---|---|---|---|------------|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | cost + = 6 |
|---|---|---|---|---|---|---|------------|

| | | | | | | | |
|---|---|---|---|---|---|---|------------|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | cost + = 4 |
|---|---|---|---|---|---|---|------------|

| | | | | | | | |
|---|---|---|---|---|---|---|-------------|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | cost + = 3. |
|---|---|---|---|---|---|---|-------------|

$$= 20.$$

this is brute force approach of cutting
we need to minimise it.



suppose we cut at & 5

this both state is independent from each other.

because

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|

X.

0 1 3 4 5 7
i j

cost :- length of stick to be cut

cost :- $\text{cuts}[j+1] - \text{cuts}[i-1]$

so,

$f(i, j)$

if $i > j$ return 0

$\text{mini} = 10^9$

for $\text{ind} = i$ to $\text{ind} = j$

$\text{cost} = \text{cuts}[j+1] - \text{cuts}[i-1]$

+ $f(i, \text{ind}-1)$ } Remaining
+ $f(\text{ind}+1, j)$ } path.

$\text{mini} = \min(\text{mini}, \text{cost})$

return mini ;

so DP:- states ??
i, j

$\text{dp}[c+1][c+1]$

c is size of cuts;
=.

Time complexity :-

$O(c^3)$ { two changing parameter and one for loop }.

Space complexity :-

$O((c+1)^2)$.

Now for tabulation, we need to think bottom up;

50:

for $i = c$ till $i >= 1$

 for $j = 1$ to $j <= c$

 if $i > j$ continue

 // copy sequence

return $dp[i][c]$.

3. Burst Balloons:-

You are given n balloons, painted with some number on it. You asked to burst all them If you burst i th balloon then you will get $\text{num}[i-1] \times \text{num}[i] \times \text{num}[i+1]$ cost. You are asked to maximise this cost as possible.

Eg:-

$$\begin{array}{r} 3 \quad 1 \quad 5 \quad 8 \\ \times \\ \hline \end{array} \quad \text{cost} = 3 \times 1 \times 5$$

$$\begin{array}{r} 3 \quad 5 \quad 8 \\ \times \\ \hline \end{array} \quad + 3 \times 5 \times 8$$

$$\begin{array}{r} 3 \quad 8 \\ \times \\ \hline \end{array} \quad + 1 \times 3 \times 8$$

$$\begin{array}{r} 8 \\ \times \\ \hline \end{array} \quad + 1 \times 8 \times 1$$

$$= 167.$$

Here if we choose one index and then there will be two sub problem but they are not independent

like

$$\begin{array}{r} b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5 \\ \times \\ \hline \end{array}$$

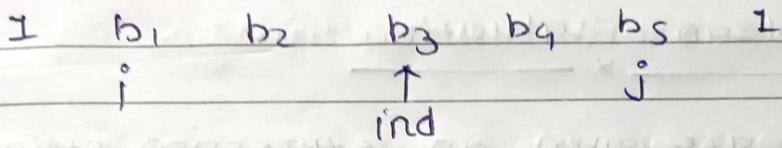
$$\begin{array}{r} b_1 \quad b_2 \quad b_4 \quad b_5 \\ \times \\ \hline \end{array}$$

$$= b_2 \times b_4 \times 5$$

we need this

from other part.

so we will start from down side:-



$$\text{cost} = \text{num}[i] \times \text{num}[i-1] \\ \times \text{num}[j+1]$$

for all $\text{ind} = i$ to $\text{ind} = j$

and

$$+ f(i, \text{ind}-1)$$

$$+ f(\text{ind}+1, j)$$

and we will maximise it.

so

$$f(i, j)$$

$$\text{if } i > j$$

return 0

for $\text{ind} = i$ to $\text{ind} = j$

$$\text{cost} = \text{num}[i-1] \times \text{num}[i] \\ \times \text{num}[j+1]$$

$$+ f(i, \text{ind}-1)$$

$$+ f(\text{ind}+1, j)$$

$$\text{maxi} = \max(\text{maxi}, \text{cost})$$

return maxi

to memoize we will have i and j as states.

4. Boolean Evaluation :-

You are given an expression string, where operands will be T and F, operator will be AND, OR, XOR.

You need to find the number of ways we can parenthesize the expression such that it evaluate to True.

Eg:- FIT^F

$$(F|T)^F \Rightarrow T \wedge F \Rightarrow T$$

$$F|(T \wedge F) \Rightarrow F|T \Rightarrow T$$

So output :- 2

so

$$T \wedge F | T \not\Rightarrow F^T$$

$$(T^F) | T \not\Rightarrow F^T$$

or I can,

$$(I^F) | (T \wedge F^T)$$

or I can

$$(T^F | T \wedge F) ^ T$$

Many partition possible

so pattern is breaking down at operand.

so,

$$T \wedge F \mid T \& F \wedge T \mid F$$

↑ ↑ ↑ ↑ ↑
i +2 +2 +2 j

partition can be at

let's say we have & has at middle index

left & right

~~xways for true~~ ~~yways for true~~ so = $x_1 \times x_2$ ~~only and only~~
~~both are true~~ ~~both are false~~ ~~total and false~~ ~~so both are true~~.

for OR (gives T if one is true)

left & right

$$T = x_1 \quad T = x_2$$

$$F = x_3 \quad F = x_4$$

$$so \ total = x_1 \times x_2 \quad (both \ true)$$

$$+ x_3 \times x_2 \quad (one \ is \ true)$$

$$+ x_4 \times x_2 \quad (one \ is \ true)$$

for XOR (gives T when both are not same)

left & right

$$T = x_1 \quad T = x_2$$

$$F = x_3 \quad F = x_4$$

$$total = x_1 \times x_4 \quad (so both are not same)$$

$$+ x_3 \times x_2 \quad (so both are not same)$$

5 *) palindrome partitioning:-

Given string s, partition s such that every substring of the partition is a palindrome.

Let min cuts needed.

Eg:- bab|abcba|d|c|e

$\Rightarrow 4$.

Front partitioning :-

\rightarrow start from front

\rightarrow check if I can do partition or not,
based on question condition

\rightarrow baba|bebe

yes (b is palindrome)
 $\Rightarrow 1 + (bab|bebe)$

\rightarrow b|bab|bebe

NO (ba is not palindrome)

\rightarrow baba|bebe

yes (bab is palindrome)

$\Rightarrow 1 + (bab|bebe)$

\rightarrow so on...

\rightarrow a|bab|bcba

yes

\rightarrow a|b|ab|beba

no yes.

so on...

bababcba d c e d e

↳ do :- b | (ababcba d c e d e)

don't :- bal --

not possible as "ba" is not
palindrome

↳ go to next

do bubl --
yes we can.

this is front partition

so now we can implement this:-

$f(i) \text{ :- min partition required to cut string } s[i \dots n]$.

so,

$f(i, s)$

if $i == s.size()$

return 0

$mini = 10^9$

for $ind = i$ to $ind < s.size$

if $c isPulindrome(i, ind, s)$

$cuts = 1 + f(ind+1, s)$

$mini = \min(mini, cuts)$

return $mini$

=====

6 *) PARTITION ARRAY FOR MAX SUM

Given array, partition the array into subarrays of max k length, after partitioning each subarray changed to the max of that partition.

return the max sum after partition.

Eg:- $[1, 15, 7, 9, 2, 5, 10]$, $k=3$

$[1, 15, 7, | 9, | 2, 5, 10]$

↓

$[15, 15, 15, 9, 10, 10, 10]$

gives 84

Rules or steps for partition discussion :-

1. express everything in terms of index
2. Layout all possible partition from that index.
3. choose the best from them

$F(O)$:- max sum if we have array from $[0 \dots n]$

Here we can use concept of front partitioning.

Here we are allowed the max size of partition is k , that is our condition

solve(i , arr, n , k)
if $i == n$
return 0

int len = 0

int maxi = -10^9

int ans = -10^9

for $j=i$ to $j = \min(i+k, n)$

len = len + 1

maxi = max(maxi, arr[j])

ans = maxi * len + solve($j+1$, arr, n , k)

return ans

Here i is state for our dp solution, so we can memoize it using dp array of n size.

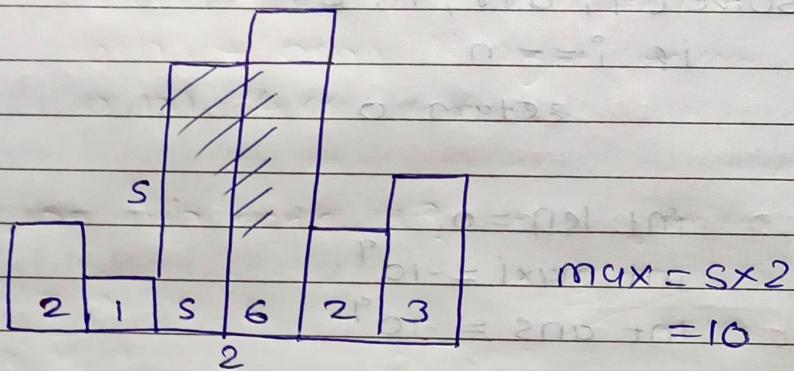
*) prerequisite for next que

7

*) largest rectangle in Histogram.

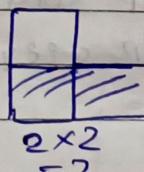
Given an array of integers heights
represent height of histogram's bar
width is 1 for every bar, return the
largest rectangle in the histogram.

Eg:- [2, 1, 5, 6, 2, 3]

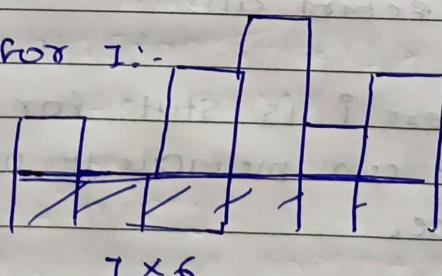


so here starting thought solution can
be take the curr bar as consideration
and look at left and right of it to make
area (rectangle)

so for 2:-



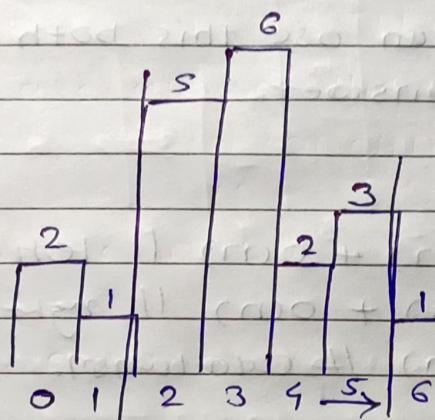
for 1:-



so on...

Here what we are doing is finding first smaller element than curr on both of the side. that will give us width and curr is our height for sure.

Eg:-



$$\text{for } curr = 2 \text{ (height)}$$

$$= (s-2+1) \times 2$$

$$= 8$$

It will give us $O(n^2)$ T.C.

so for optimisation we should use stack for next smaller elements.

left small.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 2 | 3 | 2 | 5 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

| | |
|---|---|
| 8 | 3 |
| 4 | 2 |
| 3 | 6 |
| 2 | 5 |
| 1 | x |
| 0 | 2 |

as, same as previous,

for right smaller we will start from last.

Just we need to minus 1, while hitting smaller on the stack instead of adding 2.

Now we can use this both array to compute answer.

Here TC:-

$O(n) + O(n)$ || left smaller

$O(n) + O(n)$ || right smaller

$O(n)$ || calculation.

How we can code it?

so,

for left:-

stack <int> st

left[n]

for $i=0$ to $i < n$

while stack is not empty

and

heights[$\text{top}(\text{stack})$] \geq heights[i]

st.pop()

if stack is not empty

left[i] = st.top() + 1

else

left[i] = 0

st.push(i)

for right:-

stack <int> st
right[n]

for $i = n-1$ to $i = 0$

while stack is not empty
and

$\text{heights}[\text{st.top}] \geq \text{heights}[i]$
st.pop()

if st is empty

right[i] = n-1

else

right[i] = st.top() - 1

st.push(i)

calculate ans:-

ans = 0

for $i = 0$ to n

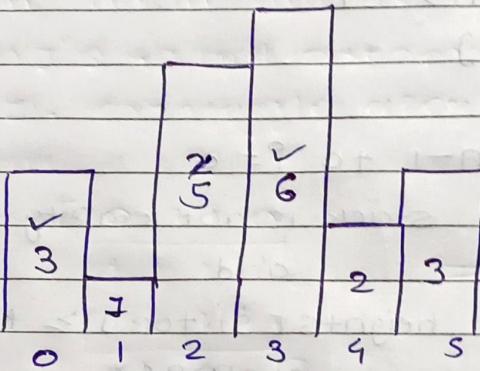
ans = max(ans,

$\text{heights}[i] \times (\underline{\text{right}[i]} - \underline{\text{left}[i]+1})$)

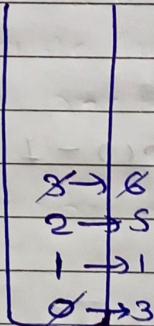
formula was:-

$(\text{right} - \text{left} + 1) \times \text{carr}$:

Can we do in one pass? Yes---



→ Now when our pointer came to 1 there is '3' on the stack that is greater than '1', so for '3' left smaller = right
is '1', pop stack



→ Now for left smaller we have to look into stack but it is empty so '0' is left smaller

→ So for 3, max area is $3 \times 1 = 3$

→ now push 1

→ now go ahead that is greater than top so push it.

→ Here we are trying to make increasing order into the stack

Now pointer came to $i = 4$ that is 2 and less than top.

so for top left smaller is at $\underline{\underline{4}}$ 2 (from stack)
right smaller is at $\underline{\underline{4}}$ (curr pointer)

\rightarrow pop it.

Now on top we have 5 that is greater than curr pointer 4 that is 2 value

$$1 \leftarrow 5 \rightarrow 4$$

$$= s \times (4 - 1 - 1) = 10$$

so on...

stack <int> st

ans = 0

for $i = 0$ to $i = n$

while stack is not empty

and

$i == n$ or $h[st.top()] \geq h[i]$

height = $h[st.top()]$

st.pop

if stack is empty $\omega = i$

else $\omega = i - st.top() - 1$

ans = max (ans, height $\times \omega$)

return ans

Max Rectangle area

Given $n \times m$ matrix formed of 0's and 1's. Find out max area formed by 1's. (rectangle area)

Eg:-

| | | | | | |
|-----|---|---|---|---|-----------------|
| $[$ | 0 | 1 | 1 | 0 | $]$ |
| | 1 | 1 | 1 | 1 | $\Rightarrow 8$ |
| | 1 | 1 | 1 | 1 | |
| | 0 | 1 | 1 | 0 | |

Here we will try to form histogram possible for every row and find the max rectangle area.

like for row 0:

| | | | |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

Here max area = $2 \times 1 = 2$

for row 1:

| | | | |
|---|---|---|---|
| 1 | 2 | 2 | 1 |
|---|---|---|---|

Here = $2 \times 2 = 4$

for row 2:-

| | | | |
|---|---|---|---|
| 2 | 3 | 3 | 2 |
|---|---|---|---|

Here $2 \times 4 = 8$

so on

for row 4:-

| | |
|---|---|
| | |
| 3 | 3 |

0 0

$$\text{here } 3 \times 2 = 6$$

out of all max is 8.

let's code it up

```
solveC matrix, n, m)
```

```
heights[m] = {0}
```

```
ans = 0
```

```
for i=0 to n
```

```
    for j=0 to m
```

```
        if matrix[i][j] == 1
```

```
            heights[i]++
```

```
        else
```

```
            heights[j] = 0
```

```
alpha = getMaxCRow(heights, m)
```

```
ans = max(ans, alpha)
```

return ans:

(*) count square submatrices with all ones.

Given $m \times n$ matrix of ones and zeros, return how many square submatrices have all ones.

Eg:-

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | $\Rightarrow 10 + 4 + 1 = \underline{\underline{15}}$ |
| 1 | 1 | 1 | 1 | ↑ ↑ ↑ |
| 0 | 1 | 1 | 1 | size size size |
| | | | | 1×1 2×2 3×3 |

Brute force can be stand at a index and try to expand it's size.

but for DP solution, we will make DP matrix of same size $m \times n$.

$dp[i][j] =$ how many rectangle ends at (i, j)
 that is how many square's right bottom is (i, j)

matrix

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

DP

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 2 | 2 |
| 1 | 2 | 3 |

Here what we are doing is finding min possible in previous neighbour and add one to it.

| Matrix | | | | dP | | | |
|--------|---|---|---|----|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| 1 | 1 | 1 | 1 | 1 | 2 | 3 | 3 |

$\underline{1+1} \quad \underline{2+1}$

like this.

Solve (matrix, m, n)

for i=0 to m

// Here m is row count

n is col count //

dP[m][n] = 0

for i=0 to m

dP[i][0] = matrix[i][0]

for i=0 to n

dP[0][i] = matrix[0][i]

for i=1 to m

for j=1 to n

if matrix[i][j] == 1

dP[i][j] = 1 + max(dP[i-1][j],

dP[i-1][j-1]

dP[i][j-1]

for i=0 to m

for j=0 to n

$$\text{ans} = \max(\text{ans}, \text{dp}[i][j])$$

return ans.

1

3 3 3 1 1 1 1 1 1 1

$$\begin{array}{r} 1+5 \\ \hline 6 \end{array}$$

Point 3)

(a.m. x(t) u(m)) w103

Mr. & Mrs. John

titles were set in a 20pt font.

11 1200 105 021 00

$\text{vol} = 60700796$

Monteiro

$\text{Corr}_{\text{X} \times \text{Y}} = \text{Corr}_{\text{X} \text{ & } \text{Y}}$

00101168

Thị trấn Giang Thành

Mat 1:1-20

Number of children

Lee Chinchin

600-000-2889 + 1-800-444-8888

卷之三

10. *What is the name of the author?*