

* Dynamic Programming :-

* Why it come to picture?

Ans: Recursion या Backtracking का time complexity exponential है आहा या like (a^n) or 2^n . ताकि अगर कोई तरीके हो पिछसे हम recursion का use करते हाला - बार-बार calculation को जटी store कर पाएं तो जो time complexity हो जाएगा फिर को linear हो जाएगा like ($O(n)$ or $O(n^2)$ type)

∴ Recursion / backtracking: O^n (exponential)

Dynamic Programming: n (linear)

* Dynamic programming का use कहाँ होता है

⇒ Condition for DP:

- ① Recursion होना चाहिए
 - ② Optimal Substructure
 - ③ Overlapping Subproblem
 - ↳ यदि highly तभी we होता था कि वार Recursion call हो रहा है तो उसकी जरूरी नहीं
- max m
min m
target etc.

* How do we solve D.P Problem.

↓
Tabulation
(Bottom-up) ↑

↓
Memorization
(Top-down) ↓

0-1 Knapsack Problem



Types of knapsack:

- Red of Knapsack:**

 - ① **Fractional Knapsack:**
दो दाता दिया गया हैं। Suppose 9 kg filled हैं और 2.2 kg के block हैं। तो 1 kg पूर्ण हुआ गया के लिए 2 kg के block की जावा कर के उत्तरी 1 kg के सकते हैं (greedy algo).
Capacity = 10 kg.
 - ② **0-1 Knapsack:**
Same Scenario but इसी दो या तो दूरा block में है या नहीं लेना है फल
Condition in bag में पहले 1 kg का बिगा जाएगा। Then 2kg वाला भी जाएगा।
 - ③ **Unbounded Knapsack:**
0-1 Knapsack में दो element पर बार दो ले सकते हैं. Criteria
fulfil करने के लिए but unbounded हैं दो multiple बार पर element
ले सकते हैं। यहाँ तो दूरा container दो Same element को
मर दो।

Here we are focusing in 0-1 Knapsack.

Given:

- Knapsack Problem:

 - ↳ weight and value of 'n'-item, Now we have to put these into a bag (knapsack) of capacity W, and then we have to find the max profit or value that can be achieved by putting 'W'-capacity item in bag.
 - ↳ इसे 2 array किया जाएगा: named
 - weight = []
 - value = []
 - ↳ और एक capacity 'W'.
 - ↳ अब 'W' एक constraint है जो मैं बताऊँगा तिक दूल्हा value की डालना नहीं bag के आदर.
 - ↳ अब इस एक-एक item की डालना शुरू करते हैं और check करते होते हैं कि max val क्या होगा। | फिलहाल यहाँ पर choice based Selection ही रहा ही।
 - ↳ ये यां Recursion से करने पर :- $O(2^n)$ लग जाएगा।
या d.P से करने पर :- $O(n * w)$ लग जाएगा।

Step 1 :- Always remember in d.P of knapsack our max profit is in the last index of 2-D array. And for each and every knapsack problem there are formula which implies and then the result will come in the last index.

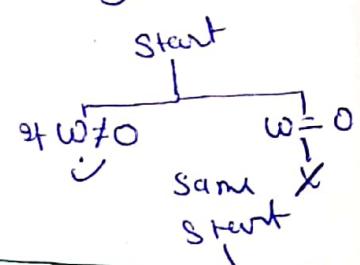
- ↳ This must be the approach to bring the answer (whether it is max or min) in the last index.
- ↳ First we define the function in which our (weight[], Val[], capacity, len of arr(val or weight)) passed. So that we performed our operation.

```
def Knapsack [ wt, Val, W, n ]:  
    :  
    :  
    :  
    return
```

Step 2

Then we have to create an 2-D array whose row and column is choosed by that element who is changing frequently. for that we can create choice diagram.

Clearly array ने कोई change नहीं होगा only weight and n की changes होगा। तो उसी को 2-D array बना ली।



```
def Knapsack (wt, val, W, n):  
    K = [[0 for x in range(w+1)] for y in range(n+1)]  
    :  
    return
```

अहम पर 1-element बनाए जाएगी है

Step-3: Main logic (Backbone):-

- ↳ आब अपने पास array बन गया है
- ↳ इसे D.P. का use करके इस तरह भरना है कि पर Consolidated Sheet होयाए हो जाए।
- ↳ हमें दो 100P की जोकरत पड़ेगी 1st loop 100P से data 30एगा 2nd loop column की data 30एगा।
- ↳ होशा याद रखे जो capacity है वो weight वा के element से ज्ञा होगा।
- ↳ आब 1st loop 100P की indicate को (weight of sack)
- ↳ 2nd loop द्वारा data को check कोगा। (element put on sack)
- ↳ हमारे code में लिखा है कि आज तक i=0 और j==0 हो तो पहले भवको 0 कर दो।
- ↳ उसके बाद हम 'j' loop को compare करते हैं wt के data से
 - आज data होता है तुला wt वाले से मध्ये उसकी पहले जोड़े किया जाता है। तब चुपचाप से उसे copy कर दें।
 - आज तुला data बड़ा हुआ हो तो उसके जो, जो element का value की combination होता है calculate किया जाया।
 - आज j का data wt[j] array के किसी भी element के उपर निकला है। वह से उसका value put कर देना है। wt[j] array से

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0

कृति
क्रिया
कानून

* Formula used

$$\text{अब } \text{wt}[i-1] \leq j \text{ से } \text{जैसे } \text{line } \text{की } \text{ऊपर } \text{ले } \text{जाए}$$

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

$$\text{अब } \text{wt}[i-1] > j \text{ की } \dots$$

$$K[i][j] = K[i-1][j]$$

↑
ये कोई रद्दा है जो j आज लेता है तो तीव्र
जूरे की राय है जो data है उसको लिख दो।

Implementation of KnapSack (0-1) problem:- (Max Profit)

def knapsack (wt, val, W, n):

K = [[0 for j in range (W+1)] for i in range (n+1)]

for i in range (n+1):

 for j in range (W+1):

 if i==0 or j==0:

 K[i][j] = 0

 else if (wt[i-1] <= j):

 K[i][j] = max(val[i-1] + K[i-1][j-wt[i-1]], K[i-1][j])

 else:

 K[i][j] = K[i-1][j]

return K[n][w]

* description

- जैसा कि हमें जा रहा है कि जब KnapSack का weight उसीके 100P के पास पहुंचता है।
- हम पहले ही जो val है को क्या consent ही भाल है।
- किस combination करने के लिए आपके निक त्रैयु वाले 100(i-1) से data लेकर दौरे तका भाल है तभी possible combination के data को गर सके।

If __name__ == '__main__':

 val = [10, 15, 40]

 wt = [1, 2, 3]

 W = 6

 print (knapsack (wt, val, W, len(val)))

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	10	10	10	10	10	10
2	0	10	15	25	25	25	25
3	0	10	15	40	50	55	65

40+10, 40+15, 40+25

$W = 6$ ← container size
 $wt = [1, 2, 3]$ ← weight
 $val = [10, 15, 10]$ ← value
 $n = \text{len}(val) = 3$

$i=0$	0	1	2	3	4	5	6
$j=0$	0	0	0	0	0	0	0
$j=1$	1	10	10	10	10	10	10
$j=2$	2	0	10	15			
$j=3$	3						

$i=0$ के लिए
 $j=0, j=1, j=2, j=3, j=4, j=5, j=6$ →
 $\underline{i=1}$ के लिए
 $K[0][1] = 0, K[0][2] = 0, \dots, K[0][6] = 0$
 $j=0, i=0 \text{ or } j=0$
 $K[j][j] = 0$

$\underline{j=1}$
 $wt[1-1] \leq 1$ True
 $\Rightarrow wt[0] \leq 1$ ∵ $wt[0] = 1$ है।

$$\begin{aligned}
 K[1][1] &= \max(val[0] + K[0][1-wt[0]], K[0][1]) \\
 &\Rightarrow \max(10 + K[0][0], K[0][1]) \\
 &\Rightarrow \max(10 + 0, 0) = 10.
 \end{aligned}$$

$\underline{j=2}$
 $wt[0] \leq 2$
 $1 \leq 2$ True.
 $K[1][2] = \max(val[0] + K[0][2-1], K[0][2])$
 $= \max(10 + 0, 0) = 10$

$\underline{j=3}$
 $wt[0] \leq 3$
 $1 \leq 3$ True.
 $K[1][3] = \max(val[0] + K[0][3-1], K[0][3])$
 $= \max(10 + 0, 0) = 10$
 Similarly $j=4, j=5, j=6 = 10$ हैं।

$\underline{i=2}$
 $\underline{j=1}$ $wt[2-1] \leq 1$ False.
 $wt[2] \leq 1$
 $K[2][1] = K[2-1][1] \text{ or } K[1][2], 0 \text{ or } = 10$
 $\therefore K[2][1] = 10.$

$\underline{j=2}$ $wt[2-1] \leq 2$ or $2 \leq 2$ True.
 $\hookrightarrow K[2][2] = \max(val[1] + K[1][2-2], K[1][2])$
 $= \max(15 + K[1][0], K[0][2])$
 $= (15 + 0, 10)$
 $= 15$

i=2
j=3

$wt[1] \leq 3$ True.

$$K[2][3] = \max [val[2-1] + K[2-1][3-2], K[2-1][3]]$$

$$K[2][3] = \max [15 + K[1][1], K[1][3]]$$

$$= \max [15 + 10, 10]$$

$$= \max [25, 10] = \underline{25}$$

Similarly

$j=4, 5, 6$ False

		1	2	3	4	5	6
		0	0	0	0	0	0
i	0	10	10	10	10	10	25
	1	20	10	15	25	25	55
	2	30	10	15	40	50	65

i=3 $j=0$ के लिए 0 ही दैवा।

j=1 $wt[3-1] \leq 1$ $3 \leq 1$ False.

$$\therefore K[3][1] = K[2][1] = \underline{10}.$$

j=2 $wt[3-1] \leq 2$
 $3 \leq 2$ False

$$K[3][2] = K[2][2] = \underline{15}.$$

j=3 $wt[3-1] \leq 3$
 $3 \leq 3$ True.

$$K[3][3] = \max [val[3-1] + K[3-1][3-wt[3-1]], K[3-1][3]]$$

$$= \max [40 + 0, 25]$$

$$= \max [40, 25] = \underline{40}$$

j=4 $wt[3-1] \leq 4$
 $3 \leq 4$ True.

$$K[3][4] = \max [val[2] + K[2][1], K[2][4]]$$

$$= \max [40 + 10, 25]$$

$$= \max [50, 25] = \underline{50}$$

j=5 Skipped = Ans = 55

j=6 $wt[3-1] \leq 6$ $3 \leq 6$ True.

$$K[3][6] = \max [val[2] + K[2][6-wt[2]], K[2][6]]$$

$$= \max [40 + K[2][6-3], K[2][6]]$$

$$= \max [40 + 25, 25]$$

$$= \max [65, 25]$$

65

Ans
65
65
65

* KnapSack(0-1) (extended) :- (Memorization technique) :-

=) 2nd recursion or extended version.

↳ इस technique की ओरें की तीव्र recursion का पूरा code आना चाहिए।

↳ Then हम क्या करें? इस recursive step को DP table में store कर देंगे।

First we see the recursive approach of KnapSack Problem:

Code : Using Recursion

```
def knapsack (wt, val, w, n, res):
    if n==0 or w==0:
        return res
    if (wt[n-1] > w):
        return knapsack(wt, val, w, n-1, res)
    else:
        res = max(val[n-1] + knapsack(wt, val, w-wt[n-1], n-1, res),
                  knapsack(wt, val, w, n-1, res))

```

else:

return max(val[n-1] + knapsack(wt, val, w-wt[n-1], n-1, res),

knapsack(wt, val, w, n-1, res))

if __name__ == '__main__':

wt = [1, 2, 3]

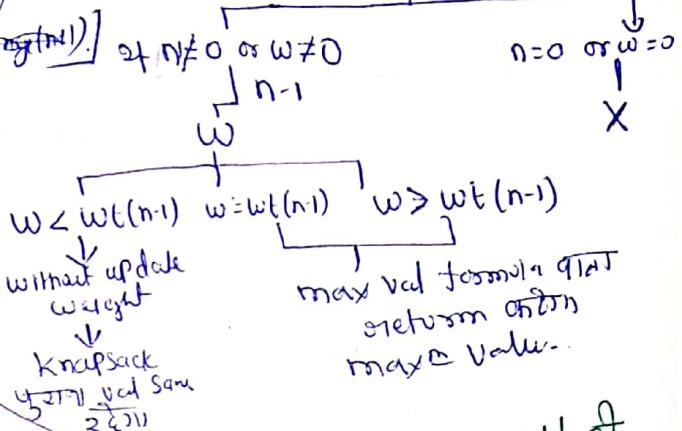
val = [10, 15, 40]

w = 6

print(knapsack(wt, val, w, len(val), 0))

choice diagram :

Start(KnapSack())



means जब weight of
एक वज़ाहत वज़ाहत
कुछ मत नहीं

जब weight array का
weight नहीं समान है तो
max find करें।

* Enhancing the recursive code using memorization.

- ↳ कमी कमी ही DP का top-down approach (memorization) करते ही भी जापदा होगा कि ज्यादा Code बढ़ाने का उत्तर नहीं।
- ↳ वर्स एक 2-D array का लेना है और Code को भी बढ़ाना है कि इसका करने से पहले array में check करने का आगे कुछ पहुँच ही हो पड़ी से उस value को लेकर return आ जाए। further recursion की execution करने के अपराध नहीं है।

Code

```
def knapsack( wt, val, W, n):
    if n==0 or w==0
        return 0
    if t[n][w] != -1
        return t[n][w]
    if (wt[n-1] > w):
        t[n][w] = knapsack( wt, val, w, n-1)
    else:
        t[n][w] = max( val[n-1] + knapsack( wt, val, w-wt[n-1], n-1),
                      knapsack( wt, val, w, n-1))

    return t[n][w]
```

↑ ही सबसे पहली base case
जोचे। (function n = max पर चुस्त हुआ)
↑ n = min पर चुस्त होगा।
Nia-Versa

मैं line के रहे हैं कि आज तो वह
मैं 2D array में पुष्ट data है तो उसे
return कर दो तो recursion call
न कर।

if __name__ == '__main__':
 wt = [1, 2, 3]

val = [10, 15, 40]

W = 6

n = len(val)

t = [[-1 for j in range(W+1)] for i in range(n+1)]

Knapsack(wt, val, W, n)

यहाँ पर -1 की initialise करती है
(0 की नहीं करते)

② Subset Sum problem (using D.P) (with reference to Knapsack problem).

2.1 Detect whether a given sum is present on any subset of Given set (return True or False):

Given: a) एक Set या array होगा (+ve numbers) N, no

b) एक Sum = value होगा जिसे होगा

c) Time Complexity $O(N * \text{Sum})$

Task If there is any subset with given sum present in set then return True else return False.

Logic: a) क्या कुछ same है Knapsack की तरह एक weight array नहीं है।

b) D.P से लेने के लिए same 2-D array बनाना है

c) इसकी कोई subset point जैसे जरवान है तो logic Backtracking के Accept / Reject की ही तरफ़ियाँ बदलें।

Before Starting this: First we code this program using recursion:-

```
def Subset(arr, sum, n):
    if sum == 0:
        return True
    if n == 0 and sum != 0:
        return False
    if arr[n-1] > sum:
        return Subset(arr, sum, n-1)
    else:
        return Subset(arr, sum - arr[n-1], n-1) or Subset(arr, sum, n-1)
```

if __name__ == '__main__':
 arr = [1, 2, 3, 4, 5, 6, 7]
 sum = 17
 n = len(arr)

Point(Subset(arr, sum, n)).

7.9ms

* Implementation Of Subset Of Sum having sum equal to sum of any subset of sum (Present | absent).

Recursion वाला Code 10000 Iteration में भी जाने हैं ही डिशनों
optimised wing. Bottom up (Tabulation) method से करना better है।

False का मतलब: कोई ऐसा subset
नहीं जिसमें एक भी element न
हो जो उसको sum 0 से बड़ा बना।

```
def isSubset(arr, sum, n, t):
    # initialization V.V.G part for this ques!
```

For i in range (1, sum+1):
 t[0][i] = False.

For j in range (n+1):
 t[j][0] = True.

For i in range (1, n+1):
 For j in range (1, sum+1):

if arr[i-1] ≤ j:

t[i][j] = t[i-1][j-arr[i-1]] or t[i-1][j]

else if (arr[i-1] > j):

t[i][j] = t[i-1][j]

return t[n][sum]:

in initialization

0	T	F	F	F	F	F
1	T					
2	T					
3	T					
4	T					

True का मतलब: sum यादों ही
जोकि उसके element बढ़ा जाए
1 elem = []
2 elem = [] + [] } empty element
3 elem = [] + [] + [] } अपना बढ़ावा

logically भी यहीं
जाए ताकि] छोटा है
ही उससे कुछ भी जाए
कर्म पर ऐसे उससे का
है ans -ve ताकि
यही Array bound error हो।

If __name__ == '__main__':

arr = [1, 2, 3, 4]

sum = 6

n = len(arr)

t = [[False for i in range(sum+1)] for j in range(n+1)]

print(isSubset(arr, sum, n, t))

Greedy

initialization.

Putting $n = 0$ to $n+1 = \text{True}$.

Putting $\text{sum} = 1$ to $\text{sum}+1 = \text{False}$.

Now iteration start from

$i = (1, n+1)$ & $j = (1, \text{sum}+1)$

	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
1	T	T	F	F	F	F	F
2	T	T	T	F	F	F	F
3	T						
4	T						

for $j = 1$

$j=1$: $\text{arr}[1-1] > j$ or $\text{arr}[0] > j$
 $1 > 1$ (False)

$$t[i][j] = t[i-1][j - \text{arr}[i-1]] \text{ or } t[i-1][j]$$

$$\text{or } t[i][j] = t[0][0] \text{ or } t[0][1]$$

$$T \text{ or } F = T$$

Cond¹
 $\exists \text{arr}[i-1] > j$:
 यहीं पर्याप्त है कि $\text{arr}[i-1] > j$ हो।
 $\exists \text{arr}[i-1] \leq j$:
 यहीं $\text{arr}[i-1] \leq j$ हो।

$j=2$: $\text{arr}[1-1] > 2$ or $\text{arr}[0] > 2$
 $1 > 2$ (False)

$$t[1][2] = t[0][2-1] \text{ or } t[0][2]$$

$$= t[0][1] \text{ or } t[0][2]$$

$$F \text{ or } F = F$$

$j=3$: $\text{arr}[0] > 3$ (False)
 $1 > 3$

$$t[1][3] = t[0][3-1] \text{ or } t[0][3]$$

$$\Rightarrow F \text{ or } F = F$$

Similarly $j=4, j=5, j=6$,

जबकि $\text{arr}[j] > 1$ होता है तो $t[i][j]$ का value F होता है।
 simple long j : t element जो कि t है का value
 max sum 1 से होता होता करी 2, 3, 4, ..., 16 होता है।

For $j=2$

$j=1$

$\text{arr}[2-1] > 1$

$\text{arr}[1] > 1$

$2 > 1$ (True).

$$\therefore t[2][1] = t[1][1] = T$$

$j=2$

$\text{arr}[2-1] > 2$ or $\text{arr}[1] > 2$

$2 > 2$ False means $\text{arr}[1] \leq 2$

$$t[2][2] = t[1][2-\text{arr}[1]] \text{ or } t[1][2]$$

$$\Rightarrow t[1][2-2] \text{ or } t[1][2]$$

$$\Rightarrow t[1][0] \text{ or } t[1][2] = T \text{ or } F = T$$

$j=3$

$\text{arr}[2-1] > 3$ or $\text{arr}[1] > 3$ or $2 > 3$ (False)

$$t[2][3] = t[2-1][3-\text{arr}[2-1]] \text{ or } t[2-1][3]$$

$$\Rightarrow t[1][2] \text{ or } t[1][3] \text{ or } T \text{ or } F = T$$

Rest of code $j=4, j=5, j=6$

= False. यहीं $\text{arr}[i-1] <= 4$ होना चाहिए ताकि $t[i][j]$ का value हो।

	0	1	2	3	4	5
0	T	F	F	F	F	F
1	T	T	F	F	F	F
2	T	T	T	T	F	F
3	T	T	T	T	T	T
4	T	T	T	T	T	T

For $i = 3$

$J = 1$

$$arr[3-1] > 1$$

$$arr[2] > 1 \text{ or}$$

$$3 > 1 \text{ (true)}$$

$$t[3][1] = t[2][1] = T$$

$J = 2$

$$arr[3-1] > 2$$

$$3 > 2 \text{ (true)}$$

$$t[3][2] = t[2][2] = T$$

$J = 3$

$$arr[3-1] > 3$$

$$3 > 3 \text{ false.}$$

$$t[3][3] = t[2][3 - arr[3-1]] \text{ or } t[3-1][3]$$

$$\Rightarrow t[2][0] \text{ or } t[2][3]$$

$$\Rightarrow T \text{ or } T = T$$

$J = 4$

$$arr[3-1] > 4$$

$$3 > 4 \text{ false.}$$

$$t[3][4] = t[2][4 - arr[3-1]] \text{ or } t[3-1][4]$$

$$= t[2][4-3] \text{ or } t[2][4] \text{ or } t[2][1] \text{ or } t[2][0]$$

$$\Rightarrow F \text{ or } F = T$$

$J = 5$

$$arr[3-1] > 5$$

$$3 > 5$$

$$t[3][5] = t[2][5 - arr[3-1]] \text{ or } t[3-1][5]$$

$$= t[2][5-3] \text{ or } t[2][5]$$

$$\Rightarrow t[2][2] \text{ or } t[2][5] \text{ or } T \text{ or } F = T$$

$J = 6$

$$arr[3-1] > 6 \text{ false}$$

$$t[3][6] = t[2][6 - arr[3-1]] \text{ or } t[3-1][6]$$

$$\Rightarrow t[2][3] \text{ or } t[2][6]$$

$$\Rightarrow T \text{ or } F = T$$

For $i = 4$ इसी तरह शाखा true होगा.

इसे 4 iteration तक पहल थी क्या यह है.

1, 2 already हैं तो उनसे 1, 2, 3 एवं संकेत ही इसे लिये

3 तक ही सब true हैं जैसे ही 3 add किये subset होंगे तो क्या

पहल 3 single ही क्या संकेत हैं

3+1 ही 4 एवं संकेत हैं som ∴ true.

3+2 ही 5 एवं संकेत हैं som ∴ true.

3+2+1 ही 6 एवं संकेत हैं som ∴ true.

2.2. Subset of Sum Variation:

To count how many Subsets of a given set whose sum is equal to given Sum -

Given : $\{ \text{One Set} \}$ or array = [1, 2, 3, 4, 5, 6]

$$b) \quad \text{Sum} = 5.$$

task : Count the number of subset whose sum = 6.

$$\underline{\text{like}} \quad \{1, 2, 3\} = 6$$

$$\{4, 2\} = 6$$

$$25 + 2 \cdot 10 = 55$$

$$\$ 5,13 = 6$$

Logic. L) Similar to previous program where we are detecting that a given sum present in any subset or not

or not

↳ असे Change में ज्ञाना है कि जहाँ छो जे या कारके boolean
गलतना या रहे थे पहाँ पर count को गलतना करेंगे,
means जैसे ही detect हुआ की sum वह रहा है पहाँ पर
वी increment ही जारी flag या पिं +[0][sum]
ताकि यो कोई another element add हो तो वह sum के
ले first की increment कर कर ही tree का record
रख पाए की इससे पहाँ जो set लिया गया उसके सी
ऐसा element का subset है जो sum देने में capable
हो।

* Implementation of Count of Subset of given Sum.

```
def SubSetCount (arr, sum, n, t)
    # initialization of the 0th index of each.
```

Initialization
(start v.0th)
& first step
For i in range (1, sum+1):
 t[0][i] = False.
For j in range (n+1):
 t[j][0] = True.

For i in range (1, n+1):
 for j in range (1, sum+1):
 if arr[i-1] > j:
 t[i][j] = t[i-1][j]
 elif arr[i-1] ≤ j:
 t[i][j] = t[i-1][j-arr[i-1]] + t[i-1][j]

return t[n][sum].

	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
1	T	1	0	0	0	0	0
2	T	1	1	1	0	0	0
3	T	1	1	2	1	1	1
4	T	1	1	2	2	2	2
5	T	1	1	2	2	3	3
6	T	1	1	2	2	3	4

if __name__ == '__main__':

arr = [1, 2, 3, 4, 5, 6]

sum = 6

n = len(arr).

t = [[For i in range(sum+1)] for j in range(n+1)]

Count of Subset of Sum (Optimized approach)

Time Complexity: $O(N^*S)$

Space Complexity: - $O(S)$
Highly reduced.

- ↳ We know this logic by $t[2][sum+1]$
- ↳ ये पाया गया कि निकसी रखी रास्तों में data गरजे के लिए पूरी हीं एक previous रास्ता है। इसे उत्तर द्यते हैं।
- ↳ यहाँ इस modulus operator का use करते हैं।
- ↳ इस logic में alternate रास्ता का use करते हैं।
 - कभी 1st रास्ता current, 2nd रास्ता previous
 - कभी 1st रास्ता previous, 2nd रास्ता current

Code:

```
def SubsetCount (arr, sum, n, t):
```

```
    for j in range(n+1):
```

```
        for i in range(sum+1):
```

else का रास्ता है।

```
        if [j == 0]:  
            t[i][j] = True.  
        elif (i == 0):  
            t[i][j] = False.
```

पहले j आने सम
तक की रास्ते।

then
element (n) का रास्ता
के लिए रास्ते।

Initialisation
on set tech

```
        elif (arr[n-1] > j):
```

Reject करना है।
 $t[i][j] = t[i][j]$

```
        elif (arr[i-1] ≤ j):
```

Accept करना है।
 $t[i][j] = t[(i-1)][j - arr[i-1]] + t[(i-1)][j]$



```
if __name__ == '__main__':
```

```
arr = [1, 2, 3, 4, 5, 6]
```

```
sum = 6
```

```
n = len(arr)
```

```
t = [[False for i in range(sum+1)] for j in range(3)]
```

```
print(subset(arr, sum, n, t)).
```

Time Complexity
 $O(Sum)$

2.3

Equal Sum Partition Problem:-

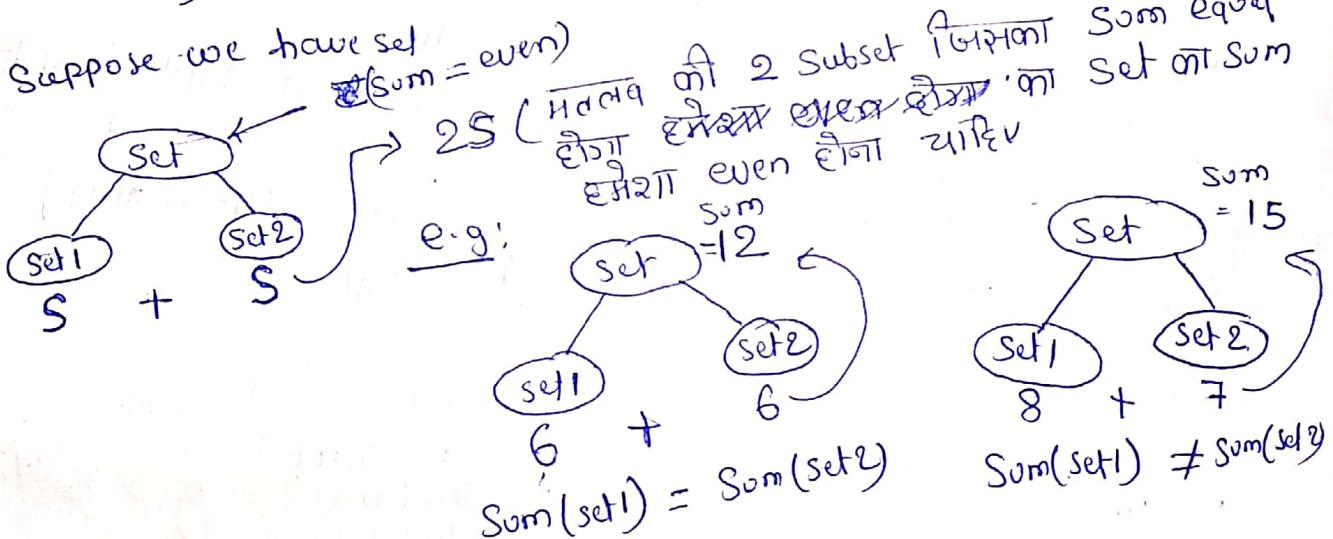
whether a given set can be partitioned into 2 subset such that the sum of element in both subset is equal.

Given: One array or set having +ve number.

Task: To find whether the given set can be parted into 2 set or not. with equal sum

- Logic:
- ↳ Once read करते के fact फैल ही रहा हींगा की पहले सबसे Possible Combination हुई तो then sum करके Check करें
 - ↳ catch: कार्य Combination हुई मी ले ही तो ये मुख्यल दृष्टि की 2 set में किसे separate करें कि किनीं set हींगा की 2 set में किसे separate करें कि किनीं set में unique element आए mean दृष्टि में किसी same element नहीं होना चाहिए ($Set_1 \cap Set_2 = \emptyset$)
 - ↳ ऐसा नहीं करना है।

↳ Let us see some logic.



↳ 1st logic के अद्वी बन जाया कि sum check को 30% even है तो उसके का काम odd & ही नहीं होता false

↳ यदि sum even निकला तो Simple Subset sum एवं function चला देना है नियरम (या) $S = \frac{\text{sum}}{2}, \text{if } \text{sum} \neq 0, \text{else}$

↳ we are forgetting to find only one subset the other subset will automatically covered or no need to find it

* Implementation of equal sum Partition problem:
Recursion vs Dynamic programming.

Recursive Approach :-

```

def Add (arr)
    S = 0
    for i in range (len(arr))
        S = S + arr[i]

```

```

def check(S):
    if S[1] * 2 == 0:
        return True.
    else:
        return False.

```

$\text{arr} = [1, 5, 5, 11]$
 $n = \text{len}(\text{arr})$
 $S = \text{Add}(\text{arr})$
if ($\text{check}(S)$):
 Sum = $S / 2$
if ($\text{Subset}(\text{arr}, \text{sum}, n)$):
 Point ("Possible")
else: Point ("Not Possible").
Sum on half
on $\frac{n}{2}$ subset
me call
arr

else:
Point (" Not Possible since sum is
Odd")

Dynamic Approach:

```

def odd(avrl):
    same
    } Step 1

def check(avrl):
    same
    }

def subset(avrl, sum, n, t):
    For i in range(n+1):
        For j in range(sum+1):
            If (j == 0):
                t[i][j] = True.
            Else if (i == 0):
                t[i][j] = False
            Else:
                t[i][j] = t[i-1][j] + t[i-1][j-1]
    exit (avrl[i-1] > j):
        t[i][j] = t[i-1][j]

```

मर्यादा रात्रि
प्रतिक्रिया देखने की
मुद्रण प्रक्रिया

Else:

$$t[i+1..2][j] = t[(i+1)..2][j] + t[i+1..2][j-1]$$

+ OR +

$$t[(i+1)..2][j-1] - \text{min}[i-1]$$

accept

return $t[n/2][\text{sum}]$

main step

if `-- name --` = '`-- main --`':
 $\text{arr} = [1, 5, 5, 11]$
 $n = \text{len}(\text{arr})$
 $s = \text{add}(\text{arr})$

`if (check(S))
S = S[1:2]
J = [[False for i in range(S[i+1])] for j in range(2)]`

$\text{g4}(\text{Subset}(\text{arr}, s, n, t))$:

Point ("Possible")

else: Point ("Not Possible")

Point (Not) Only

- clc:
Point ("sum is odd Not Possible")

Minimum Subset Sum difference (\therefore Subset sum का) :-

Given

एक array दिया है। Say arr = [1, 6, 11, 5] whose length $\Rightarrow n = \text{len}(arr)$.

Task

we have to find the minimum difference b/w subsets.

e.g. Subsets of [1, 6, 11, 5] are:-

$\Rightarrow \{\}, (1, 6), (1, 5), (11, 6, 5), (6, 5), (1, 11, 5), (1, 11, 6), (1, 6, 5)$

$(11, 6), (11, 5), (1), (6), (5), \dots$ and so on.

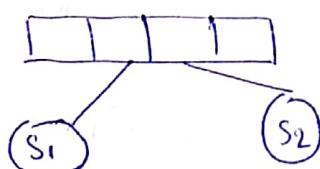
$\Rightarrow (1, 6, 5), (11)$ की दीपक का diff $= 12 - 11 = 1$ इसकी तरफ
जब उनकी sum odd है तो 0 नहीं हो सकता।

Output ने '1' जिलगा चाहिए।
गल्ती Half तक ही data लेगा।

T	F	F	F	F	F	F	E
T							
T							
T							
T							
T							
T							
T							

Logic

दो array दिया हुआ है। Let's break it into 2 Partition

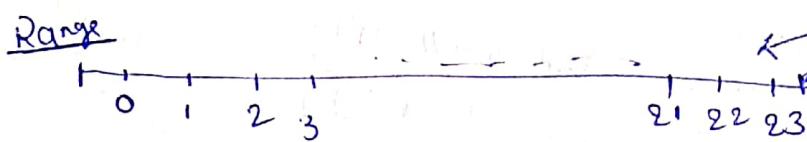


$S_1 < S_2$ (for absolute Ans)
calculation

better understanding
comes
from
examples

l) We have to find the sum of array? ऐसा कौन?

अंगठे द्वारे पास S_1 और S_2 की $S_2 = \text{Sum} - S_1$ भी आवश्यक है।
माना है (Sum को as Range treat करें)।



दो उपर्युक्त subsets की sum जूँच values की दीपक lies करें।

l) Note Range के उपर के सारे value subset की sum की नहीं है जबकि तो हम केवल उनी सम की ज्ञेय बर्तावी।

Subset की sum की दीपक द्वारा subset of sum की अनुरूप बर्तावी।

V.V. 29 इस काम के लिए दो subset of sum की अनुरूप बर्तावी।

Subset of sum की sum के Range की पास करना है।

l) Subset of sum की 2D array का last column Pick करें।
उपर loop लगा करकी 2D array का append करें।
loop छुड़ा से Range 1/2 तक का data की array 2 का append करें।

* Implementation of minimum subset sum difference:

Class Solution:

```
def minDifference(self, arr, n):
```

```
    def add(arr):
```

```
        temp = 0
```

```
        for i in range(n):
```

```
            temp += arr[i]
```

} Sum find करने का Temp

```
    def subset(arr, sum, n, t):
```

```
        for x in range(1, sum+1):
```

```
            t[0][x] = False.
```

```
        for x in range(n+1):
```

```
            t[x][0] = True.
```

```
        for i in range(1, n+1):
```

```
            for j in range(1, sum+1):
```

```
                if arr[i-1] > j:
```

```
                    t[i][j] = t[i-1][j]
```

```
                else:
```

```
                    t[i][j] = t[i-1][j - arr[i-1]] or t[i-1][j].
```

```
Sum = add(arr)
```

$t = [\text{False for } i \in \text{range}(sum+1)] \text{ for } j \in \text{range}(n+1)$

```
subset(arr, sum, n, t)
```

```
m = 0
```

```
If (len(arr) == 1)
```

```
return arr[0]
```

```
else:
```

```
m = 99999
```

For $i \in \text{range}(1, \lfloor \frac{sum}{2} + 1 \rfloor)$:

```
m = min(m, sum - (2 * i))
```

```
return m.
```

loop sum की आधी होती रहती है

मिनीमल

minimize करें

sum - (2 * i) करता है

```
> If __name__ == '__main__':
```

```
    n = int(input())
```

```
arr = [1, 6, 11, 5]
```

```
obj = Solution()
```

```
ans = obj.minDifference(arr, N)
```

```
print(ans).
```

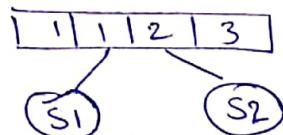
Count the number of Subsets with a given difference:-

Given :-
 b) One array is given like $\text{arr} = [1, 1, 2, 3]$
 b) and difference = 1 दिया है।

Task : दिए गए अवय से 2 Subsets निकली जाना difference find करो
 और उनमें दिए गए difference को जातना है और Counter की Count
 कर दो +1.

e.g.: $\text{arr} = [1, 1, 2, 3]$
 $\text{diff} = 1$
 $O/P = \begin{matrix} S_1 & S_2 \\ \{1\} & \{2\} \\ \{1, 2\} & \{3\} \\ \{1, 3\} & \{2\} \\ \{1, 2, 3\} & \emptyset \end{matrix}$
 $\frac{S_1}{2} = 1, \frac{S_2}{2} = 1, \frac{S_1 + S_2}{2} = 1$
 $\frac{S_1}{4} = 1, \frac{S_2}{4} = 1, \frac{S_1 + S_2}{4} = 1$

logic.



$$\text{we have } S_1 - S_2 = \text{diff}$$

$$\text{and we know } S_1 + S_2 = \text{Sum(arr)}$$

$$2S_1 = \text{diff} + \text{sum}(arr)$$

$S_1 = \frac{\text{diff} + \text{sum}(arr)}{2}$

मेरे पास Sum of 1st Subset है।
 इसका अर्थ है कि मेरे पास subtraction करने के लिए पास है।

Ques Sum of Subset (S_1) निकालो कैसे?

Ans: S_2 को निकालने का ज़ंकरत वर्षी क्या है? यह पास है कि subtraction
 2 pairs के लिये ही हवा है तो अब S_1 को subset S_1 हवा हवा है।
 कि automatic S_2 भी बन हवा है। rather than कि है complex
 Calculation में पड़ कर time waste को simply उस subset की
 Count कर दें है जिसका $\frac{S_1}{2} = S_1$ के equivalent है।

Sum निकालने के लिए formula -

$$\text{new-sum} = \frac{\text{diff} + \text{sum}(arr)}{2}$$

on use in then Count Subset of
 given sum &
 count करो तो है।

Code:

```

def Subset(arr, sum, n, t):
    for i in range(n+1):
        for j in range(sum+1):
            if i == 0:
                t[i][j][0] = True
            elif i == 0:
                t[i][j][0] = False
            else:
                if arr[i-1] > j:
                    t[i][j][0] = t[i-1][j][0]
                else:
                    t[i][j][0] = t[i-1][j][0] + t[i-1][j-arr[i-1]][0]
            else:
                t[i][j][0] = t[i-1][j][0] + t[i-1][j-arr[i-1]][0] + t[i-1][j-arr[i-1]-1][0]
    return t[n][sum][0]
  
```

If --name-- = '--main--'
 $arr = [1, 1, 2, 3]$
 $n = \text{len}(arr)$
 $\text{diff} = 1$
 $S_1 = \text{sum}(arr)$
 $\text{new-sum} = \frac{S_1 + \text{diff}}{2}$

This is the extra element in
 subset sum problem.

$t = [\text{False for } i \text{ in range(newsum+1)}]$
 $\text{for } j \text{ in range(2)}]$

Point ('Count': "", subset(arr, new-sum, n, t))

* Target Sum Problem. (Knapsack Part):

Q.4. Given:- ↳ An array is given with some elements.

↳ Sum = value फिया जाएगा।
↳ '+' और '-' sign को इस प्रकार से इस array element
में set लेना है तिक उसका sum given sum के
बराबर हो।

Task :-

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

and sum = 1

+1	+1	+2	-3
----	----	----	----

Sum = 1

+1	-1	-2	+3
----	----	----	----

Sum = 1

+1	+1	-2	+3
----	----	----	----

Sum = 1

$$\text{Total} = 1+1+1 = 3$$

3 Possible Combinations

Logic:

+1	+1	+2	-3
----	----	----	----

+1	-1	-2	+3
----	----	----	----

-1	+1	-2	+3
----	----	----	----

अब ये कुछ similar Count the number of subset with given difference
जैसा ही हो जाया है। इसके बास पर +ve no को & -ve no को separate
करके लिख दिया है। but याम वही हो रहा है।

Note: इस क्षमता के लिए हमारा जाया है but actual में coding
में को difference नहीं क्योंकि sum करने पर भी difference पाला
Value हो आए है (like 4-3 लिखें या +4 + (-3) same)

Ques यहाँ confuse करने की टिप्पणी की गयी है +, - की combination

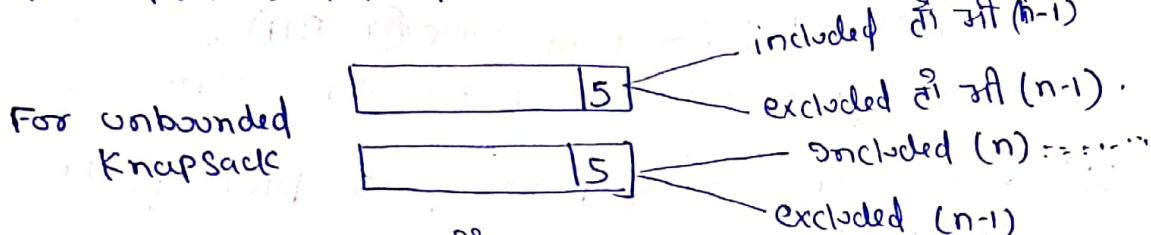
$$\hookrightarrow \text{logic नहीं है: } \text{new_sum} = \frac{\text{diff} + \text{sum}}{2}$$

Rest code are same mentioned in previous page.

* Unbounded knapsack

↳ It is similar to the previous knapsack but with only one difference that is in 0-1 knapsack we once we include one element it can't be further included while in unbounded one element can be included many times.

↳ Form Previous 0 knapsack



↳ Coding is same as Previous knapsack but

↳ How to recognise that the question is based on
0-1 knapsack: No Repetition of items.

unbounded Knapsack: Repetition of item allowed.

Q. We have a knapsack with weight 'W', and 'n'-items,
and an weight array = wt[] and value array = val[]
Now we are allowed to use unlimited no of instance of an item

$$\text{e.g.: } N=2 \quad W=3$$

$$\text{val} = [1, 1]$$

$$\text{wt} = [2, 1]$$

Code for unbounded at O(Sum) space.

def knapsack(val, wt, n, w, t):

 for i in range(n+1):

 for j in range(w+1):

 if (j==0 or i==0):

$$t[i][j] = 0$$

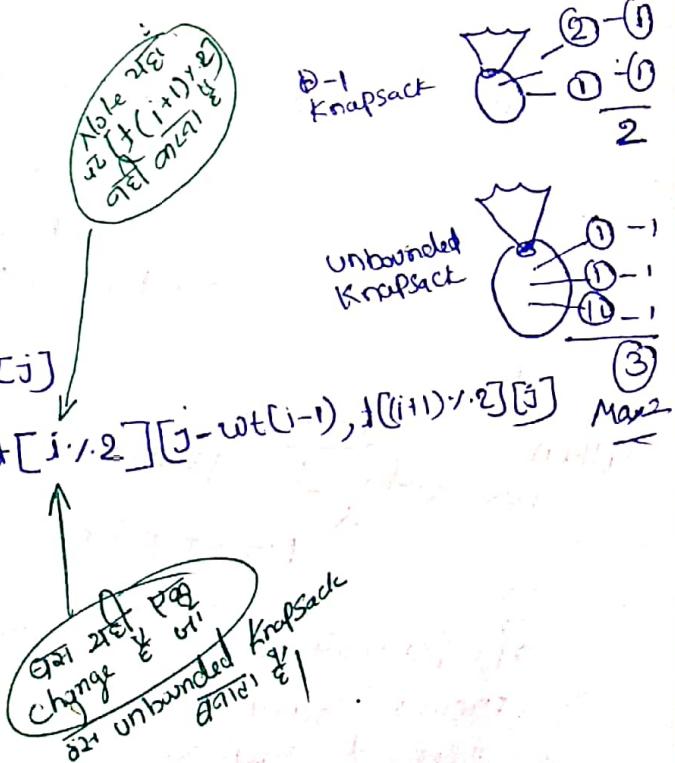
 elif (wt[i-1]>j):

$$t[i][j] = t[i-1][j]$$

 else:

$$t[i][j] = \max \left(\text{val}[i-1] + t[i-1][j-wt[i-1]], t[i-1][j] \right)$$

 return t[n][w]



* Cutting Rod Problem:-

Given:- ↳ One array having lengths of all pieces of rod or the size of rod 'N'.
 means हमें इस तार व्याय पिया है तो यह किसी तरीके से बांटना चाहते हैं। e.g.,
 rod of length = 8 के के लिए

$$\text{len} = [i \text{ for } i \text{ in range}(1, n+1)]$$

$$\text{len} = [1, 2, 3, 4, 5, 6, 7, 8] \text{ जब जानेगा}$$

↳ One array having price of all pieces of the rod length like

$$\text{Price} = [3, 5, 8, 9, 10, 17, 17, 20]$$

$$\text{len} = [1, 2, 3, 4, 5, 6, 7, 8]$$

Task इसलिए को maximize करना है obtained value को like knap के question में

first we take 8, then result = 20

then we take 7 and we have 1 choice to take 1 $\Rightarrow 20$

we take 6 we have 2 choice

$$6 + 2 = 17 + 5 = 22$$

$$6 + 1 + 1 = 17 + 3 + 3 = 23$$

we take 5 we have 3 choice.

$$5 + 3 = 10 + 8 = 18$$

$$5 + 2 + 1 = 10 + 5 + 3 = 18$$

$$5 + 1 + 1 + 1 = 10 + 3 + 3 + 3 = 19$$

When we take only 1 then we have choice:

$$1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 3 \times 8 = 24$$

max

2nd Possible तरीकी ही बात है उत्तर एवं combination one means Knapsack के concept अपेक्षा कोई भिन्नता नहीं एक element repeat ही रहा है ही unbounded दौड़ा।

Implementation of Rod cutting Problem:-

जब Price array और length array दिया है (कोई specific array of length नहीं)

e.g.; len = 8

Price = [3, 5, 8, 9, 10, 17, 17, 20] का output = 24.

For Time complexity $O(N \cdot W)$
Space complexity $O(N \cdot W)$

```
def rod(len, price, n, s, t):
    for i in range(0, n+1):
        for j in range(s+1):
            if i == 0 and j == 0:
                t[i][j] = 0
            elif len[i-1] > j:
                t[i][j] = t[i-1][j]
            else:
                t[i][j] = max(price[i-1] + t[i-1][j], t[i][j-1])
    return t[n][s]
```

Space Complexity
 $O(N \cdot S)$

For Time complexity — Same ($O(N \cdot W)$)
but Space complexity $O(W)$.

def rod optimised (len, price, n, s, t):

```
    for i in range(0, n+1):
        for j in range(s+1):
            if i == 0 and j == 0:
                t[i][j] = 0
            elif len[i-1] <= j:
                t[i][j] = max(price[i-1] + t[i-1][j], t[i][j-1])
            else:
                t[i][j] = t[i-1][j]
```

return t[n][s]

Space Complexity

$O(S)$

If __name__ == '__main__':

N = 8

price = [3, 5, 8, 9, 10, 17, 17, 20]

len = [i for i in range(1, N+1)]

t = [[0 for i in range(N+1)] for j in range(N+1)]

Point(rod(len, price, N, N, t))

K = [[0 for i in range(N+1)] for j in range(2)]

Point(rodoptimised(len, price, N, N, K))

* Cutting Rod Problem:- (Variation):

Cutting Rod Problem:- (Variation):

- ⇒ Maximise number of cuts in a rod if it can be cut only in given 3 sizes.

Here ; Given : Rod of length : N

Rod cut in size A, B and C

Task: Maximise the number of cuts in a rod.
उत्तर गमिष्ठ ही है - 1 point का देना)

logic:

- (i) क्षेत्रफल परिमाय को assume करना आवश्यक है। length N दिया है मतलब road का size = $N^{\frac{1}{2}}$ हो इसी
1, 2, 3, 4, 5, 6, 7 रिसर्सों में कार जा सकता है।

(ii) Then कृदि के logic लगाना है जो कि हर A, B, C के
पास कट के indicate करने वाला हो जो Possible हो
है उसे ट्रॉक्स करता हो।
पार्ट-अप 2 के example में 5, 5, 2 में छंट 1, 3, का कोई भी
part नहीं बचा सकता। एक array बनाते हैं जिसके size [0 से $N+1$] तक

(iii) एक loop और array में len पाला Parameter की array
जो index होगा।

v) len 100 का था और array में 100 पाला Parameter की array
का index होगा।

vi) len 100 का था और चलाते हैं जो दिया हुआ length से data तापा।
और कि check करें कि यह array में जो भी length है वहाँ को
possible है यहाँ जब road के section-cut से।

vii) उत्तर की possible है तो next के check करें। possible cut
repeat की वही न कर रहा तो उसका कारण होता है कि उसका
उस index के respect में cuts की agreement कर दी।
उस वर्ग की 0, 1 को किसी फॉर्म देना है।

viii) 1 कट से खासा में होगा की उत्तर की combination का
एवं कुते के cut में include हो जाए। e.g.
1 का 1 cut possible है।
2 का 1 cut possible है।
5 का 1 cut possible है।
7 का 1 cut possible है।
3 का 2 cut possible है।
last we get.

0	1	2	3	4	5	6	7	
0	X	1	X	2	1	3	2	4

यह cut करते ही
उसका length
का respect हो।

* Implementation of Rod cutting Problem: (Variation)

When specified cut is given (प्रत्येक सीधे cut निश्चिद न हो) और उसके सीधे variable मिला है और पूछा जाए max किरण तो तो तो सकता है।

```
def maximumcut (arr, N):
```

$$t = [0] * (N+1)$$

For i in range

$$t[i] = -99999$$

For i in range (3):

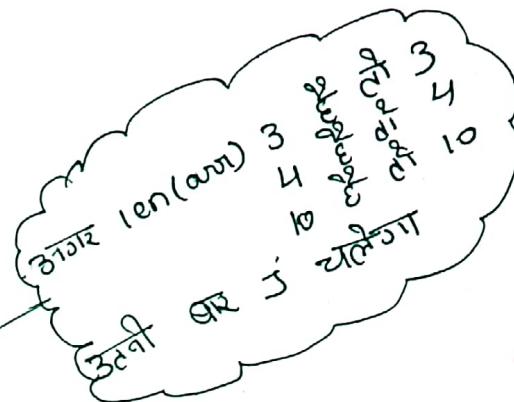
If ($i - arr[j] \geq 0$ and $t[i - arr[j]] == -99999$):

$$t[i] = \max(t[i - arr[j]] + 1, t[i])$$

If $t[n] == -99999$:

$$t[n] = 0$$

return t[n]



उत्तीर्ण ग्र ज्युलन्ड्र
possible नहीं हुआ array के कोई cut

```
if __name__ == '__main__':
```

$$arr = [2, 1, 1]$$

$$N = 4$$

```
Point(maximumcut(arr, N))
```

$$arr = [5, 5, 2]$$

$$N = 7$$

वहाँ output को आएगा क्योंकि जब भावे की तीक्ष्ण
max^म 2-cut i.e. 5, 2 हैं क्योंकि सकारात्मक है।

Tressing

For $i = 0, 1$

$j = 0$

$$\begin{aligned} 1 - arr[0] &\geq 0 \\ 1 - 5 &\geq 0 \quad (\text{False}) \end{aligned}$$

$j = 1$

$$\begin{aligned} 1 - arr[1] &\geq 0 \\ 1 - 5 &\geq 0 \quad (\text{False}) \end{aligned}$$

$j = 2$

$$\begin{aligned} 1 - arr[2] &\geq 0 \\ 1 - 2 &\geq 0 \quad (\text{False}) \end{aligned}$$

For $i = 2$,

$j = 0$

$$\begin{aligned} 2 - arr[0] &\geq 0 \\ 1 - 5 &\geq 0 \quad (\text{False}) \end{aligned}$$

$j = 1$

$$\begin{aligned} 2 - arr[1] &\geq 0 \\ 1 - 5 &\geq 0 \quad (\text{False}) \end{aligned}$$

0	1	2	...	7
0	-9999			

0	1	2	3	...	7
0	-9999	-9999	-9999		

इसका मतलब यह है
कि 2 section का
rod काटा जा सकता है
available rod में
काटा जा सकता है

$j = 2$

$$\begin{aligned} 2 - arr[2] &\geq 0 \\ 2 - 2 &\geq 0 \quad (\text{True}) \end{aligned}$$

$$\text{Then, } DP[2-2]! = -9999$$

$$\text{or } DP[0]! = -9999 \quad (\text{True})$$

$$\therefore DP[2] = \max(DP[i - arr[2], +1], DP[i])$$

$$DP[2] = \max(DP[0] + 1, -9999)$$

$$DP[2] = 1$$

0	1	2	3	4	...	7
0	-9999	1	-9999			

For $i = 3$

$j = 0$

$$\begin{aligned} 3 - arr[0] &\geq 0 \\ 3 - 5 &\geq 0 \quad (\text{False}) \end{aligned}$$

$j = 1$

$$\begin{aligned} 3 - arr[1] &\geq 0 \\ 3 - 5 &\geq 0 \quad (\text{False}) \end{aligned}$$

$j = 2$

$$\begin{aligned} 3 - arr[2] &\geq 0 \\ 3 - 2 &\geq 0 \quad (\text{True}) \end{aligned}$$

Next cond² check

0	1	2	3	4	5	...	7
0	-9999	1	-9999	-9999			

For $i = 4$

$j = 0$

$$\begin{aligned} 4 - arr[0] &\geq 0 \\ 4 - 5 &\geq 0 \quad (\text{False}) \end{aligned}$$

$j = 1$

$$\begin{aligned} 4 - arr[1] &\geq 0 \\ 4 - 5 &\geq 0 \quad (\text{False}) \end{aligned}$$

$j = 2$

$$\begin{aligned} 4 - arr[2] &\geq 0 \\ 4 - 2 &\geq 0 \quad (\text{True}) \end{aligned}$$

Next cond² check

$$\begin{aligned} DP[4 - arr[2]]! &= -9999 \\ DP[4 - 2]! &= -9999 \quad (\text{True}) \end{aligned}$$

$$\begin{aligned} DP[4] &= \max(DP[4 - arr[2]] + 1, DP[i]) \\ DP[4] &= \max(2, -9999) = DP[4] = 2 \end{aligned}$$

for i=5

j=0

$$5 - arr[0] \geq 0$$

$5 - 5 \geq 0$ True. Then $dp[5-5]! = -9999$

$$dp[0]! = -9999 \text{ (True)}$$

Then

$$dp[5] = \max(dp[5-5]+1, dp[5])$$

$$= \max(dp[0]+1, -9999)$$

$$= \max(1, -9999) \therefore dp[5] = 1$$

$$\Rightarrow 1$$

j=L

$$5 - arr[1] \geq 0$$

$5 - 5 \geq 0$ (True) Then $dp[5-5]! = -9999$

$$dp[0]! = -9999 \text{ (True)}$$

Then

$$dp[5] = \max(dp[5-5]+1, -9999)$$

$$= \max(1, -9999)$$

$dp[5] = 1$. (already ~~not~~ override ~~not~~ ~~True~~)

j=2

$$5 - arr[2] \geq 0$$

$5 - 2 \geq 0$ (True) Then $dp[5-2]! = -9999$

$$dp[3]! = -9999 \text{ (False)}$$

0	1	2	3	4	5	6	7
0	-9999	1	-9999	2	1	3	

for i=6

$6 - arr[0] \geq 0$ (True) Then $dp[6-5]! = -9999$

$$dp[1]! = -9999 \text{ (False)}$$

j=1

$$6 - arr[1] \geq 0$$

$6 - 5 \geq 0$ (True) Then $dp[6-5]! = -9999$

$$dp[1]! = -9999 \text{ (False)}$$

j=2

$6 - arr[2] \geq 0$ Then $dp[6-2]! = -9999$

$$dp[4]! = -9999 \text{ (True)}$$

Then $dp[6] = \max(dp[6-2]+1, -9999) = dp[6] = 3$

$$dp[6] = \max(dp[4]+1, -9999) = dp[6] = 3$$

0	1	2	3	4	5	6	7
0	-9999	1	-9999	2	1	3	-9999

for i=7

j=0

$7 - arr[0] \geq 0$ Then $dp[7-arr[0]]! = -9999$

$7 - 5 \geq 0$ True. $dp[2]! = -9999$ True. $\Rightarrow dp[7] = \max(dp[7-5]+1, -9999)$

$dp[2] = 2$

Similarly j=1 $\Rightarrow 2$ & 3 & 4

j=2

$7 - arr[2] \geq 0$ (True) $dp[7-arr[2]]! = -9999$ $\Rightarrow dp[7] = \max(dp[7-2]+1, 2)$

$dp[2] = 2$ $\Rightarrow dp[7] = 2$

Coin Change Problem:

Given: Value of N , find the number of ways to make change for ' N '-cents, if we have infinite supply of each $S = \{S_1, S_2, \dots, S_m\}$ valued coins.

Means \rightarrow एक Value of N परिया होगा जो कि nett amount / sum हो।
 \rightarrow एक Coin on array के होगा

Task:- Find करते हैं total no of ways जो कि coin on combination का sum के equal हो।

E.g: Total = 4 Coin = $\{1, 2, 3\}$

$$\begin{aligned} \{1, 1, 1, 1\} &= 4 \\ \{1, 1, 2\} &= 4 \\ \{2, 2\} &= 4 \\ \{1, 3\} &= 4 \end{aligned}$$

Total outcome = 4.

- Logic:
- \rightarrow Clearly it's knapsack problem on variant of knapsack where we have choice either to take a particular coin or not.
 - \rightarrow Since we have infinite supply of coin so it is definitely unbounded knapsack problem.
 - \rightarrow Now we have to count the possible subset whose sum is equal to given sum.
 - \rightarrow This question is 100% similar to Subset of Sum problem where we have a given sum and a set and we have to find the subset whose sum is equal to given sum.

So No more discussion

See Subset of Sum problem where we count the subset whose sum is equal to given sum.

* Implementation of Coin change Problem

(counting no of possible way in which a dukandar give change to customer)

```
-def subsetCount(avl, n, sum, t):
```

```
    for i in range(n+1):
```

```
        for j in range(sum+1):
```

```
            if j == 0:
```

```
                t[i][j] = 1
```

```
            elif i == 0:
```

```
                t[i][j] = 0
```

```
            elif (avl[i-1] <= j):
```

```
                t[i][j] = t[i-1][j-avl[i-1]] + t[i-1][j]
```

```
            else:
```

```
                t[i][j] = t[i-1][j]
```

```
return t[n][sum]
```

```
def coin(avl, total):
```

```
n = len(avl)
```

```
sum = total
```

```
t = [[0 for i in range(sum+1)] for j in range(2)]
```

```
Point ("Total no of ways : ", subsetCount(avl, n, sum, t)).
```

```
If __name__ == '__main__':
```

```
    avl = [2, 3, 5, 6]
```

```
Total = 10
```

```
coin(avl, Total)
```

unbounded knapsack
 $t[i][j] = t[i-1][j-avl[i-1]] + t[i-1][j]$

Time Complexity :- $O(N^{\text{sum}})$
 Space Complexity :- $O(\text{sum})$.

Coin Change Problem (Variation) :-

(Find the min no of coin that give the total)

Use: If customer demand the amount in such a way that make convenience for him than obviously less number of note variation is convenient.

$$\Rightarrow \text{mean} \quad \begin{array}{r} 100, 100, 100, 100, 100 \\ \hline 500 \end{array} = 500 \text{ रुपये } \text{ एक नोट } \& \\ = 500 \text{ रुपये } \text{ single note } \text{ convinience } \forall .$$

Given: \hookrightarrow Coins on Pan array के पास उपलब्ध (Infinite number).
 \hookrightarrow Total Sum / amount के पास उपलब्ध

Task: \hookrightarrow To find fewest number of coin that is needed to make up that amount.

e.g: From previous example:-

Coins: [1, 2, 3]

and amount = 4

Possible ways

$$\begin{aligned} 1 + 1 + 1 + 1 &= 4 \\ 1 + 1 + 2 &= 4 \\ 1 + 3 &= 4 \\ 2 + 2 &= 4 \end{aligned}$$

Then we have 2 convenience way to keep the change in such a way that the coin is minimum i.e. $\{1+3\}$ both only 2 coins needed.

Logic: \hookrightarrow We have a amount and a array from which we have to find min number of coin.

\hookrightarrow Since we have to find min number of coin means it is an optimisation problem and definitely we have to use KnapSack. (जब तकी min होता है कि subset तक जरूर होता है KnapSack का basic fonda use करना होता)

\hookrightarrow One array is given with random coins (Similar to maximize rod cutting problem when particular section is given).

* Implementation of Coin Change problem (Variation)

(min^m number of coin needed)

⇒ logic contn: जैसा एक array बना लेंगे जो कि total/sum तक का data hold करेगा e.g. $t=[0]*(\text{amount}+1)$

$t =$	0	1	2	3	4
	0				

अब इस array से data उठा कर check करेंगे कि जो यार्ड दिया है वो भी sum बनाने में capable है कि नहीं.

l) e.g.: If we have arr = [5, 2] amount = 7

$t =$	0	1	2	3	4	5	6	7
5 और 2 से कभी भी 1, 3, Sum नहीं बन सकता है तो इसे	X	V	X	V	V	V	V	V

neglect करना होगा।

Code.

```
def count(arr, N):
    t = [0] * (N+1).
```

```
for i in range(1, N+1):
    t[i] = 999999
```

```
for j in range(len(arr)):
```

```
if (i - arr[j] ≥ 0):
    t[i] = min (t[i - arr[j]] + 1, t[i])
```

```
If t[n] == 999999:
```

```
t[n] = -1
```

```
return t[n].
```

```
If __name__ == '__main__':
```

```
arr = [5, 2]
```

```
sum = 7
```

```
Point('coins', count(arr, sum))
```

Note यदि हर extra
का जरूर नहीं है

Accept $t[i-1]$ को

Reject case
 $t[i]$ is

यदि पर मी बुबुबुबु आया है
means को कभी नहीं लेंगा कि
जो coins की combination है

वाकी यदि पर जी कोई
digit है तो उसके min coins
use coins की संख्या जो सकता है

leetcode Random

arr = [5, 2]
sum = 7

arr =	0	1	2	3	4	5	6	7
	0	99999	1	99999	2	1	3	2

arr = [5, 5]
sum = 7

arr	0	9999	9999	9999	9999	1	9999	9999
	0	9999	9999	9999	9999	1	9999	9999

Generating!

* Longest Common Subsequence (LCS) :-

* Explanation : let us assume that we have 2 strings X, Y then

Longest Common Subsequence

$$X = \boxed{a} b c \boxed{d} g f \boxed{h}$$

$$Y = \boxed{a} b e \boxed{d} f \boxed{g} h q$$

Output 4 देगा

discontinuity नहीं हो सकता है।

V/S

Longest Common Substring

$$X = \boxed{a} b c \boxed{d} g f \boxed{h}$$

$$Y = \boxed{a} b e \boxed{d} f \boxed{g} h q$$

Output: 2 देगा।

Continuous होना पड़ेगा।

Logic: First we go with the Recursive approach:

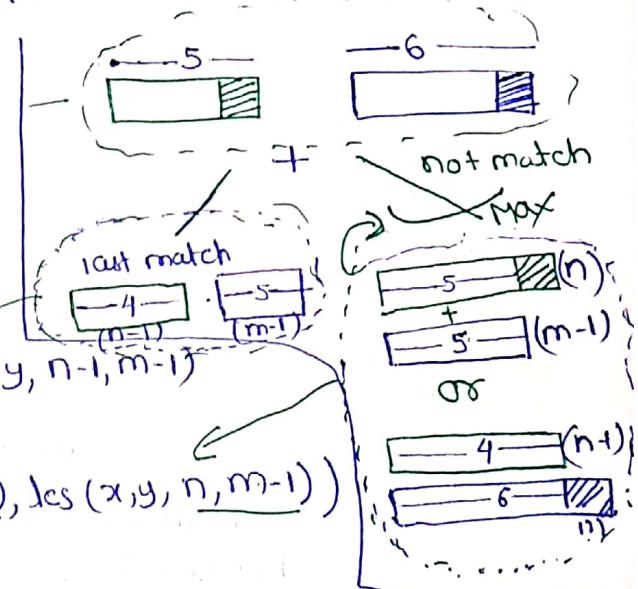
def lcscount(x, y, n, m):

If $n == 0$ or $m == 0$:
return 0.

If ($x[n-1] == y[m-1]$):
return 1 + lcscount($x, y, n-1, m-1$)

else:

return max(lcs($x, y, n-1, m$), lcs($x, y, n, m-1$))



If -- name -- = '-- main--':

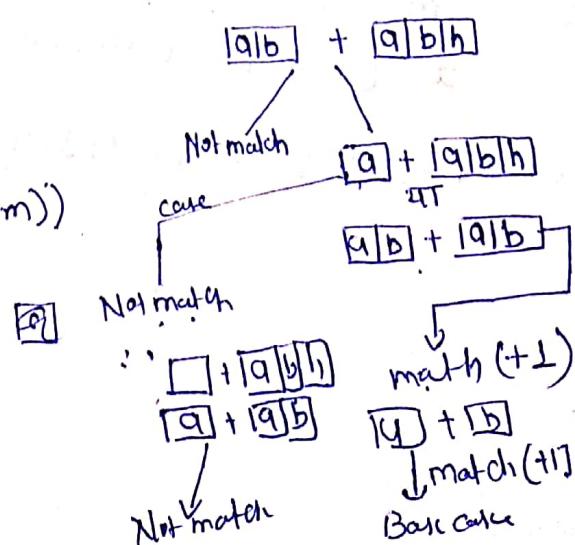
$$x = 'abcdh'$$

$$y = 'abedh\&$$

$n = \text{len}(x)$

$m = \text{len}(y)$

point("Count:", lcscount(x, y, n, m))



* Implementation of LCS (Longest Common Subsequence),
using Memoization (Enhanced Recursion means Recursion
with memory) -

```
def Lcscount(x, y, n, m, t):
```

If $n == 0$ or $m == 0$:
return 0.

if $t[n][m] != -1$
return $t[n][m]$

If ($x[n-1] == y[m-1]$):
 $t[n][m] = +1 + \text{Lcscount}(x, y, n-1, m-1, t)$

else:

$t[n][m] = \max(\text{Lcscount}(x, y, n-1, m, t),$
 $\text{Lcscount}(x, y, n, m-1, t))$

return $t[n][m]$

Don't forget that if the
digit letter are same then
we decrement size of
both strings

If --name-- = '--main--':

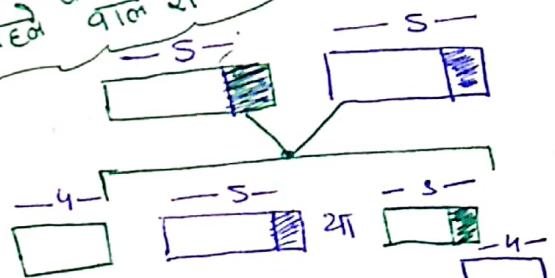
$X = \text{'Madam'}$,
 $y = \text{'Mad Pad'}$

$n = \text{len}(x)$

$m = \text{len}(y)$

$t = [-1 \text{ for } i \text{ in range}(m+1)] \text{ for } j \text{ in range}(n+1)$

Point ($\text{Lcscount}(x, y, n, m, t)$).



$t = \begin{matrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{matrix}$ $n \times m$
 6×7

Implementation of LCS using Tabulation (Non Recursive)

Fully DP based

def lcscount(x, y, n, m, t):

 For i in range (1, n+1):

 For j in range (m+1):

 if $i == 0$ or $j == 0$:

$t[i][j] = 0$.

 else if $x[i-1] == y[j-1]$:

$t[i][j] = t[i-1][j-1]$

 else:

$t[i][j] = \max(t[i-1][j], t[i][j-1])$

 return $t[n][m]$.

If __name__ == '__main__':

 x = 'abcdef',
 y = 'abedrt'

 n = len(x)

 m = len(y)

 t = [[0 for i in range(m+1)] for j in range(n+1)]

 print ("Count:", lcscount(x, y, n, m, t))

Note

Subsequence की view को लिए

e.g:-

x = a b c def
y = F e C ibg

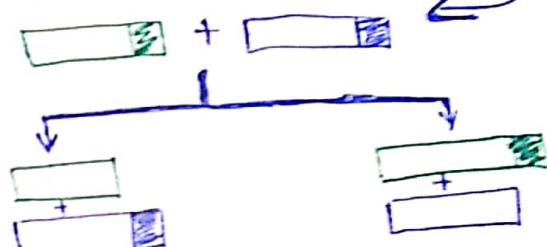
Output = 1

Don't do this

इसके कारण 1 नहीं आया
F के कारण आया है।

$x = \text{String 1}$
 $y = \text{String 2}$
 $n \leftarrow x$ की तो String 1 की उसकी length
 $m \leftarrow y$ की तो String 2 की उसकी length.

ये base case जैसी है
दूर छार नहीं हो तो यह जाए
आ जाएगा



* LCS Application :-

① Longest Common Substring :-

↳ It is a variation of Longest common Subsequence.

↳ Note: Here Substring is asked means Any sequence that is same in both the string but in continuous manner.

E.g:-

$$X = "ASF\boxed{Sam}d\&vTF" \\ Y = "QWXP\boxed{L}\boxed{Sam}T" \quad \text{Output} = 3$$

But for Subsequence. Output may be 4.

Logic:

↳ we know that it is similar to the substring as we done earlier

↳ only we have to remove that part which keep the date of non-contiguous letter.

↳ Means whenever the string is included our counter incremented and when it not matched then we start the counting from '0'.

e.g:- $X = P\boxed{Q}\boxed{S}am\&vT.V$

$Y = Q\boxed{A}F\boxed{S}am\&vT.V$

e.g. 2. $X = \text{Greek For Greeks}$
 $Y = \text{Greek For Freaks}$
Output: (Greek For) ^{on length} i.e. 8

but due to this out counter will again start from 0.

↳ There are possibilities that we have encounter more than one Substring but we have to return the length of max one.
So there must be a comparator that compare the max val.

Code:

Code of Subsequence :-

```
def lscount(x, y, n, m, t):  
    For i in range(1, n+1):  
        For j in range(m+1):  
            If i==0 or j==0:  
                t[i][j] = 0  
            Elif x[i-1] == y[j-1]:  
                t[i][j] = 1 + t[i-1][j-1]
```

else:
 $t[i][j] = \max(t[i-1][j], t[i][j-1])$

return t[n][m]

Change +

Code Of Substring :-

```
def SubstCount(x, y, n, m, t):  
    res = 0  
    min_max  
    For i in range(1, n+1):  
        For j in range(m+1):  
            If i==0 or j==0:  
                t[i][j] = 0  
            Elif x[i-1] == y[j-1]:  
                t[i][j] = 1 + t[i-1][j-1]  
                res = max(res, t[i][j])
```

$t[i][j] = 0$

return res

Change (Added)
New element.

* To Point the LCS (Substring को point करवाने के लिए).

Given: 2 Strings और उसका Subsequence point करवाना है।

e.g: $x = 'a'bcd\epsilon f'$
 $y = 'abfxed'$

Output: abfd

Logic: To understand this we have to go behind the Scene of LCS.

↳

		x = φ a b c d e f						
		φ	0	1	2	3	4	5
y =	φ	0	0	0	0	0	0	0
	a	0	1	1	1	1	1	1
b	0	1	2	2				
f	0							
x	0							

$t[i][j] = 0$
means.

Let us Check for

Checking for ab with abf

match		a	b	c
a	1	1	1	
b	1	2	2	

$x = 'abc'$
 $y = 'abfxed'$ iteration. which is subpart of our problem.

↳ 'b' और 'b' match हुआ ही दूर $t[i-1][j-1]$ करें हैं
 $t[i-1][j-1] = t[i-1][j-1] + 1$
 $t[i-1][j-1] = t[i-1][j-1] + 1 = 2$

'b' और 'c' match नहीं हुआ है
 $t[i][j] = \max[t[i-1][j], t[i][j-1]]$

'b' और 'c' match नहीं हुआ है
 $t[i][j] = \max[t[i-1][j], t[i][j-1]]$

↳ यदि match होता है तब उसके ठीक left upper diagonal की element की data $t[i][j]$ increment कर देंगी

↳ यदि match नहीं होता तब उसके right diagonal की data की check करके जो max होता होते box में डाल देंगी

Now after LCS completion we have.

↳ A matrix as shown here

		φ	a	b	c	d	e	f
φ	φ	0	0	0	0	0	0	0
	a	0	1	1	1	1	1	1
b	0	1	2	2	2	2	2	2
f	0	1	2	2	2	2	3	3
x	0	1	2	2	2	2	3	3
e	0	1	2	2	2	3	3	3
d	0	1	2	2	3	3	3	3

↳ Form last box we traversed to the $t[e][d]$

↳ we will check the last element of each string
if it matched then we shift the string left upper diagonal

↳ if it does not match we have to choose between the upper or the left box whose value is max
then we move here at the same time when the data of strings matched we keep back in a int variable.

$\Rightarrow dba$

* Implementation of Lcs pointing the Subsequence. (Pointing the sequence Not the count)

```

def lcsCount(x, y, n, m, t):
    for i in range(1, n+1):
        for j in range(m+1):
            if i == 0 or j == 0:
                t[i][j] = 0
            elif x[i-1] == y[j-1]:
                t[i][j] = 1 + t[i-1][j-1]
            else:
                t[i][j] = max(t[i-1][j], t[i][j-1])
    return t[n][m]

```

```

def lcsPoint(x, y, n, m, t):
    i = n
    j = m
    res = ""
    while (i > 0 || j > 0):
        if x[i-1] == y[j-1]:
            res += x[i-1]
            i -= 1
            j -= 1
        else:
            if (t[i][j-1] > t[i-1][j]):
                j -= 1
            else:
                i -= 1
    return res

```

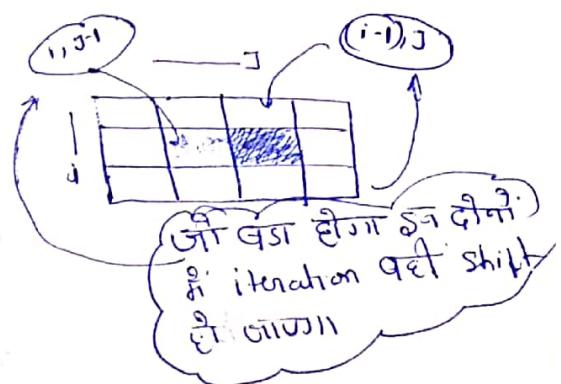
```

if __name__ == '__main__':
    x = 'abcdef'
    y = 'abfexed'
    n = len(x)
    m = len(y)
    t = [[0 for i in range(m+1)] for j in range(n+1)]

```

Point("Total no of Lcs:", lcsCount(x, y, n, m, t))

Point("The substring is:", lcsPoint(x, y, n, m, t))



② Shortest Common SuperSequence.

2 String को Combine कर
एक दूसरा String बनाना जिसमें
दीने गए String की Subsequence
हो।

Given : \hookrightarrow 2 String say x, y

$$x = 'A G G I T A B' - \text{len: } 6$$

$$y = 'G I X T X A Y B' - \text{len: } 7$$

Task \hookrightarrow we have to combine them in such a way that it will give a String for which we construct both the String (means जो String(new) formed होगा उसके लिए कौन सी String a Subsequence मिलेगी होना चाहिए।

Logic :- \hookrightarrow सबसे Worst Case होता है (Baby thinking). we add both the string and make them new string.

$$\text{like: } A G G I T A B G I X T X A Y B \quad \therefore \text{len: } (13) \text{ यह उत्तर है।}$$

Not Good

\hookrightarrow 3TG

$$x : A \textcircled{G} \textcircled{G} \textcircled{T} \textcircled{A} \textcircled{B}$$

$$y : \textcircled{G} \textcircled{I} \textcircled{X} \textcircled{T} \textcircled{A} \textcircled{Y} \textcircled{B}$$

LCS

G I T A B

$$\bar{A} \bar{G} \boxed{\bar{G}} \bar{X} \boxed{\bar{T}} \bar{X} \boxed{\bar{A}} \bar{Y} \boxed{\bar{B}}$$

len: 9

We approach like the letter which is present in both string can be written as one. So that the length of New String or Supersequence will be minimised.

\hookrightarrow we know: $x + \underline{\text{LCS}} + y + \underline{\text{LCS}}$ \leftarrow This repeated twice we don't need it

So our ans will be:

$$\boxed{x + \text{LCS} + y}$$

\hookrightarrow इसी LCS की ओरती: As we know it is optimal type question (means they are asking, largest, smallest, longer, shorter etc) and we know that the LCS repeated twice so we have to avoid this. A LCS has property to find the Subsequences.

\hookrightarrow At last in order to return the length of Super Sequence Simply in $\text{Lcs}(x, y, n, m, t)$:- return: $m + n - t[n][m]$

Only Print
Version

* (Q.9) Implementation of Shortest Common SuperSequence.
(only return the length of the superstring):-

L) This Code is Very much or exact Copy of LCS:-

```
def SCScount(x, y, n, m, t):
    For i in range(1, n+1):
        For j in range(m+1):
            If i == 0 and j == 0:
                t[i][j] = 0
            Else If (x[i-1] == y[j-1]):
                t[i][j] = 1 + t[i-1][j-1]
            Else:
                t[i][j] = max(t[i-1][j], t[i][j-1])
    return ((n+m) - t[n][m])
```

```
If __name__ == '__main__':
    x = 'AGGTAB'
    y = 'GXTXAYB'
    n = len(x)
    m = len(y)
    t = [[0 for i in range(m+1)] for j in range(n+1)]
    Point("length of Superstring", SCScount(x, y, n, m, t))
```

2.b. Implementation of pointing the Shortest Common SuperSequence (SCS):

Logic then Code.

- L) The code is similar to pointing the LCS but here the modification is only we have to include all the string part as well as of the LCS.
- b) means our super string consist of LCS & strings and the both the part of string.

e.g:

$x : A \underset{\boxed{G}}{G} \underset{\boxed{T}}{I} \underset{\boxed{A}}{T} \underset{\boxed{B}}{A} B$
 $y : \underset{\boxed{G}}{G} X \underset{\boxed{T}}{I} X \underset{\boxed{A}}{A} Y \underset{\boxed{B}}{B}$
Res: A G G X T X A Y B.

LCS: G T A B

Rest from A: A G I

Rest from B: X X Y

-def lcsCount(x, y, n, m, t):

 For i in range(1, n+1):

 For j in range(m+1):

 if i == 0 or j == 0:

 t[i][j] = 0

 elif x[i-1] == y[j-1]:

 t[i][j] = 1 + t[i-1][j-1]

 else: t[i][j] = max(t[i-1][j], t[i][j-1])

return 0

-def SCSpoint(x, y, n, m, t):

 i = n ; j = m ; res = "" ;

 while (i > 0 and j > 0):

 if (x[i-1] == y[j-1]):

 res = res + x[i-1]

 i = i - 1

 j = j - 1

 else: if t[i][j-1] > t[i-1][j]:

 res = res + y[j-1]

 j = j - 1

 else: res = res + x[i-1]

 i = i - 1

 while (i > 0):

 res = res + x[i-1]

 i = i - 1

 while (j > 0):

 res = res + y[j-1]

 j = j - 1

return res[::-1]

3. Find minimum number of insert and delete to convert String 'a' to 'b':

Given: \hookrightarrow 2 String दिया गया है। Say

$x = \text{`heap'}$

$y = \text{`Pea'}$

Task:

: $\hookrightarrow x$ की y String में बदलना है, heap की Pea में

\hookrightarrow min^m number of insert and delete के राह

Eg: \hookrightarrow heap - deleting 'h'

\hookrightarrow eap - inserting 'p' :- Peap.

Peap - deleting last 'p' ! Pea. \leftarrow String

Total 2 deletion and 1 insertion.

Insertion: 1

deletion: 2

Logic. Let us see the heap and Pea one again

heap : LCS मिल जाए. ea
Peap : LCS मिल जाए.

अब आरे String 'a' से तुम LCS को घट करे होंगे वो मिल जाएगा जो 'b' string में नहीं होता चाहिए 'heap' = 'ea' = 'hp'

Similarly आरे b से LCS को $4 - 2 = 2$

घट की होंगी वो मिल जाएगा जो b में चाहिए था पर 'a' में वही है

$$\begin{array}{r} \text{'Pea' - 'ea' = P} \\ 3 - 2 = 1 \end{array}$$

In order to convert Str1 to Str2;

for insertion : len(String2) - LCS (String1 and String2)

for deletion : len(String2) - LCS (String1 and String2)

Note //

आप इसे as a formula and भी की जाक मी only

Insertion & deletion को minimize करना है जिससे

(Str1) की (Str2) में बदला जा सके होंगे अप्रूप

Best

* Implementation to find minimum number of insertion and deletion to convert String 'a' to String 'b'
(only count the letters of insertion and deletion) :-

```

def lcsCount(x,y,n,m,t):
    for i in range(1,n+1):
        for j in range(m+1):
            if i==0 or j==0:
                t[i][j] = 0
            elif (x[i-1]==y[j-1]):
                t[i][j] = 1 + t[i-1][j-1]
            else:
                t[i][j] = max(t[i-1][j], t[i][j-1])
    return t[n][m]

```

	ϕ	H	e	j	a	p
ϕ	0	0	0	0	0	0
p	0		$\begin{bmatrix} 1-1 \\ \overline{1}-\overline{1} \end{bmatrix}$	$\begin{bmatrix} G-1 \\ \overline{G}-\overline{1} \end{bmatrix}$		
e	0		$\begin{bmatrix} G-1 \\ \overline{G}-\overline{1} \end{bmatrix}$			
j	0					

-2f -- name -- = '-- main --' :
 x = 'heap' { convert x into y }
 y = 'Pea'
 n = len(x)
 m = len(y)
 t = [[0 for j in range(m+1)] for i in range(n+1)]
 point ("insertion:", m - lcs(x, y, n, m, t))
 point ("deletion:", n - lcs(x, y, n, m, t))
 या एक Variable की (LCS) तो lcs(x, y, n, m, t) को store
 और कॉल करते होंगे 2 times for call की जरूरत
 पड़ा।

4. Find the palindromic (longest) String in a given string

Given

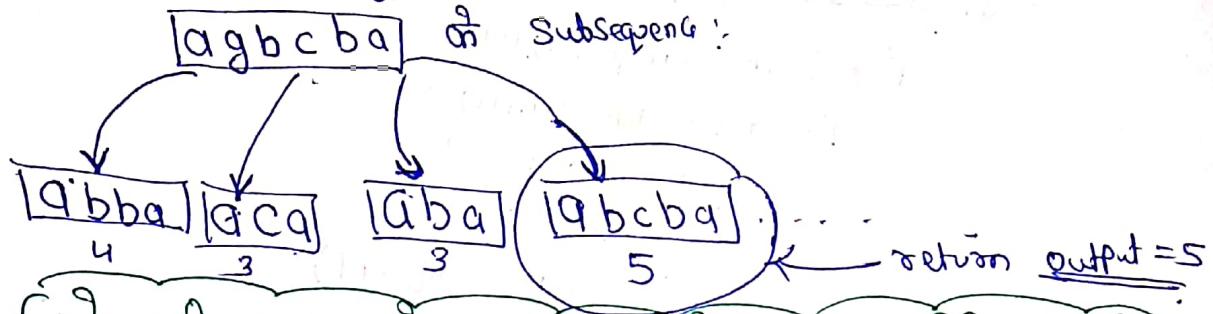
↳ Here only 1 string is given.

like

$$x = 'agbcba' \quad \underline{\text{len: 6}}$$

Task:

इसमें इस string में palindromic sequence कौनसा है?



Catch

ये भी LCS को through ही बनाए जाते हैं।
तो सर्वो एक ही String दिया है।

2nd String के लिए

$$a = agbcba$$

$$b = abcbgca \quad (\text{उल्टा कर ली)$$

Remember
this.

Longest palindromic String = LCS between (a, rev(a))

$$a = \boxed{a} \boxed{g} \boxed{b} \boxed{c} \boxed{b} \boxed{a}$$

$$b = \boxed{a} \boxed{b} \boxed{c} \boxed{b} \boxed{g} \boxed{a}$$

$\therefore \underline{abcba}$

इसे उस String को उल्टा करके उनके
बीच LCS निकालना है। then

Boom Ans आ जाएगा।

Implementation of longest Palindromic String from a given String:

(
↳ Only 1 String is given
↳ Printing both length and string module)

```
def palindromeCount(x,y,n,m,t):
    For i in range(1,n+1):
        For j in range(m+1):
            If (i==0 or j==0):
                t[i][j] = 0
            Else if (x[i-1] == y[j-1]):
                t[i][j] = 1 + t[i-1][j-1]
            Else:
                t[i][j] = max(t[i-1][j], t[i][j-1])
    return t[n][m]
```

def palindromePoint(x,y,n,m,t):

i = n
j = m
res = "

while (i > 0 and j > 0):

If x[i-1] == y[j-1]:
 res = res + x[i-1]
 i = i - 1
 j = j - 1

Else:

If (t[i-1][j] > t[i][j-1]):
 i = i - 1
Else:
 j = j - 1

return res.

इस function का उत्तर सिंगल
नी point जैवाने की लिए कोरिक्स
fun दी 't'(डेवलप) की populate
जाएगा।

If __name__ == '__main__':

x = 'agbcba'

y = x[::-1]

n = len(x)

m = len(y)

t = [[0 for i in range(m+1)] for j in range(n+1)]

Point("Total len of Palindrome", PalindromeCount(x,y,n,m,t)))

Point("Palindrome String", PalindromePoint(x,y,n,m,t)))

* Minimum number of deletion in a string to make it Palindromic :

Given : एक String किया गया होगा।
Task : उसे इसमें कैसे delete करने की letter की इस प्रकार की जो बचा हुआ letter हो वो palindromic हो।
 At last उसे min number of deletion का count return करा हो।

Code : क्योंकि हमने इस type का Ques पहले किया है इसलिए logic Skipped and direct formula = $\text{len}(\text{String}) - \text{LPS}$.

```
def palindromicCount(x, y, n, m, t):
    For i in range(0, n+1):
        For j in range(m+1):
            If i == 0 & j == 0:
                t[i][j] = 0
            Else If (x[i-1] == y[m-j]):
                t[i][j] = 1 + t[i-1][j-1]
            Else:
                t[i][j] = max(t[i-1][j], t[i][j-1])
    return t[n][m]
```

If -- name -- = 'main':

x = 'agbcba'

y = x[::-1]

n = len(x)

m = len(y)

t = [[0 for i in range(m+1)] for j in range(n+1)]

res = palindromicCount(x, y, n, m, t).

Print("min no of deletion", res).

Print("min no of deletion", n - res).

Minimum number of insertion to make a string

Palindromic :- (Quite similar to deletion but concept अलग).

- Given :
- Given String के दिया जाएगा
 - पूछा जाएगा कि मिनीमल नो इंसर्टिंग करना है जिससे दिया हुआ String palindrome बन जाए.

Task :- Return the min no of count which is necessary to make a given String palindromic.

e.g. $x = 'aebc|bda'$

अब इसकी half में बाँट देते हैं।

इस logic का we करने के पता चल की 2 इंसर्टिंग होगी।

move
← move.
 $a|edb|c$

↑
Introducing
Introducing e
↓

$a e b d c b d e a$

Logic:

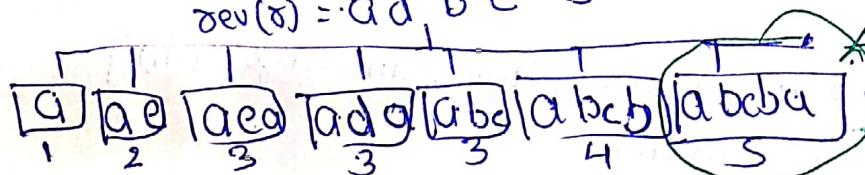
- Worst thing इस String के 1-1 करके, data pick करो, then check करो, for various pos में शर्हने पर क्या की Palindrome हो रहा है नहीं (Avoid this).
- Palindrome का होता है।
→ ऐसा Subsequence है जो forward as well as backward direction में same हो रहा है।

b) करके पहले दिये हुए String में हो Longest palindromic Subsequence को find करो।

e.g. $x = a e b | c b d a$
 $\text{rev}(x) = a d b c b e a$

बहुत भारी Subsequence
जिसका भारी है 1 4 2
समाप्त हो जाएगा।

This one is
the longest



Now we have abcba - Subsequence of Longest Palindromic

→ अब उच्चे हुए है 2' i.e. (e, d) इसको हमें पूछने पर ले String में इस प्रकार लगा देंगे कि कौन Palindromic बन जाए!

But wait a minute: 'e, d' & 'd' से याद आया कि यही ही थी दो वीज थे जिनको निकाले तो question का ही जारी

No of insertion = len(String) - Longest Palindromic Subsequence

* Implementation of Finding minimum number of insertion to make it palindrome:

(Code is similar to delete(min) to make Palindrome)! :)

```
def lcscount(x,y,n,m,t):
    for i in range(j, n+1):
        for j in range(m+1):
            if i == 0 or j == 0:
                t[i][j] = 0
            elif (x[i-1] == y[j-1]):
                t[i][j] = 1 + t[i-1][j-1]
            else:
                t[i][j] = max(t[i-1][j], t[i][j-1])
    return t[n][m]
```

```
If __name__ == '__main__':
```

```
x = "aebcbda"
```

```
y = x[::-1]
```

```
n = len(x)
```

```
m = len(y)
```

```
t = [[0 for i in range(m+1)] for j in range(n+1)]
```

```
res = lcscount(x,y,n,m,t)
```

```
print("minimum no of insertion:", n - res)
```

Pattern Matching :- (LCS का विवर)

Given: \hookrightarrow One string 'a' (पहले संख्याको match करवाना है)
 \hookrightarrow another string 'b' (दूसरी संख्या match करवाना है)

Task: \hookrightarrow we have to find that the letter of 'a' in sequence comes in 'b' or not.

\hookrightarrow जो letter वो 'a' की है वो 'b' में sequence की कौप ही होनी चाहिए और continuous manner में न रखी हो तो यहेगा लैसिट Order maintain होना चाहिए

e.g.: 1 a = Sam
 b = Samondon
 ↑
 Sam ondon
 : Matched.

e.g.: 2 a = 'Sam'
 b = PSothuqu
 ↓
 Continuous manner ही वही
 but Sequence ही नहीं
 ∴ Matched

e.g.: 3 a = Sam
 b = LSmrtu
 कॉमन के पास ही
 आएगा ∴ Not matched
 ∴ Not Matched

From above discussion we conclude that if the string is not in continuous manner then also it get matched.

Logic \hookrightarrow Best and worst thinking:- इसे Pn-एक करके letter चुनते हैं (a) जो उस से match होते जाएँ। Time complexity बहुत high हो जाएगा फॉर्म (exponential form में चल पाएगा)

$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$ $\therefore 2^n$ में चल जाए जहाँ $n=3$

s a m Sa am Sm Sam

\hookrightarrow इसके न की optimal Sol = पूँछी है न की possible cases ही करनी चाहिए ताकि आ अकल की ज्ञानों DP से उतारा हो। इसके आ जी ज्ञान की LCS की ओर सुधिकाल है इसलिए हमें Pattern matching जैसा with Subsequence then LCS may be a solution

Solve by
LCS

Pattern matching:

* Implementation of pattern matching using LCS:

We have to find: 'a' में जो भी string दिया हुआ है का
अकाल Subsequence 'b' ही है।

↪ इसके लिए हम 'a' को 'b' के बीच LCS find करना है तब
उसके Length 'a' के बराबर कहा mean 'a' का एक
letter 'b' के Subsequence में है। उस अदि concept है।

e.g.: $a = (\text{Sam})$
 $b = (\text{Pq}, \text{tS} \circ \text{qnpm})$

LCS = सबसे छोटी की
a के बराबर है।

def lcsCount(x, y, n, m, t):

for i in range(n+1):

 for j in range(m+1):

 if i == 0 or j == 0:

 t[i][j] = 0

 elif (x[i-1] == y[j-1]):

 t[i][j] = 1 + t[i-1][j-1]

 else:

 t[i][j] = max(t[i-1][j], t[i][j-1]).

return t[n][m]

If __name__ == '__main__':

 a = 'Sam'

 b = 'Pq, tS \circ qnpm'

 n = len(a)

 m = len(b)

 t = [[0 for i in range(m+1)] for j in range(n+1)]

 res = lcsCount(a, b, n, m, t)

 if (res == n):

 print("Matched")

 else:

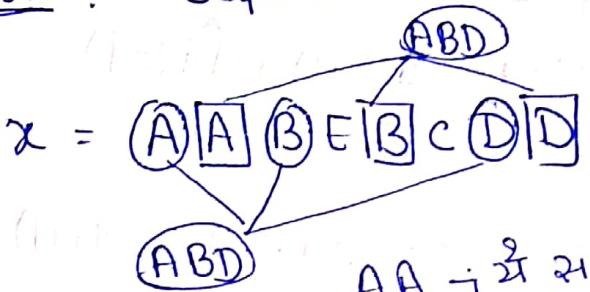
 print("Not-matched")

- Given:
- एक String के दिया जाएगा। (only one String)
 - और उसके लिये में उस Subsequence को निकालने के लिए जाए जाएगा जो इसके साथ संचारा अपेक्षित हुआ है।

Task: We have to return the length of the string which is repeated maxth time in given String.

Note: Sequence बना रहा चाहिए।

e.g:



ये Subsequence की बार
Repeat हुआ है।

V.V.9mD

AA ये सभी term
AB ये 2-2 बार repeat हुए हैं
but ये longest तो है
मतलब :: ABD return
होना चाहिए जो len(3)

Logic

- clearly 1 String दिया है और Subsequence का बात हो रही है।
- दो 1 और String चाहिए Comparison मारने के लिए।
- इस पर copy कर दें है पहले वाले string की दैरहत है।
- e.g. $x = \text{A A B E B C D D}$
 $y = \text{A A B E B C D D}$
- अब इसके बीच Substring Subsequence निकाले हैं और आ जाएगा ऐसे Subsequence equal to x हो जाएगा।
- यहाँ से constraint में लगता है कि Subsequence की letter include करो जो same index में न हो $i \neq j$ एवं



∴ Count = 3
String = ABD
आ जाएगा।

* Implementation of longest repeating Subsequence :-

- ↳ longest चीज़ रखा है (optimal sol) means dp used.
- ↳ asked for Subsequence means LCS use की जानकारी है।

```
def lcs_count(x, y, n, m, t):
```

For i in range (1, n+1):

 For j in range (m+1):

 if i == 0 or j == 0

 t[i][j] = 0

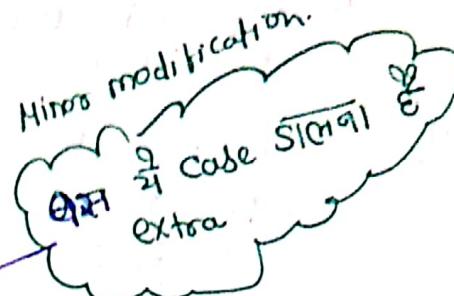
 elif (x[i-1] == y[j-1] and i != j)

 t[i][j] = 1 + t[i-1][j-1]

 else:

 t[i][j] = max(t[i-1][j], t[i][j-1]).

return t[n][m]



```
def lcs_point(x, y, n, m, t):
```

i = n

j = m

res = ""

while (i > 0 and j > 0):

 if (x[i-1] == y[j-1] and i != j)

 res = res + x[i-1]

 i = i - 1

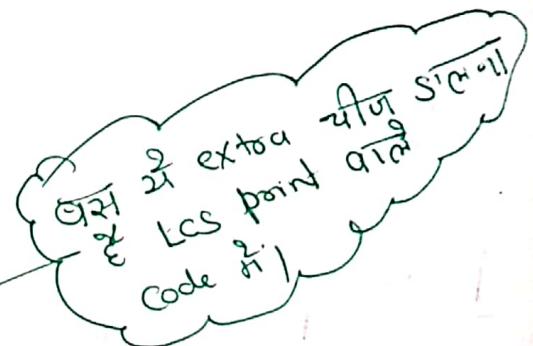
 j = j - 1

 else: if (t[i-1][j] > t[i][j-1]):

 i = i - 1

 else: j = j - 1

return res[::-1]



```
# -- name -- = __main__ :
```

x = 'AA BEBCDD'

y = x

n = m = len(x)

t = [[0 for i in range(m+1)] for j in range(n+1)]

Point("longest repeating subsequence length", lcscount(x, y, n, m, t))

Point("longest repeating subsequence", lcsPoint(x, y, n, m, t))