Date
Page No.

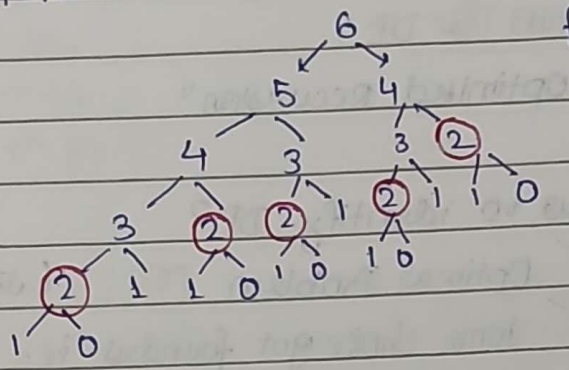
DYNAMIC PROGRAMMING

- **Fibonacci No.**

n =	0	1	2	3	4	5	6	...
	0	1	1	2	3	5	8	...

In recursion →

```
psvm{
int n=5;
sout(fib(n));
}
static int fib(int n){
if(n<2){
return n;
}
return fib(n-1)+fib(n-2);
}
```



fib(2) → repeating 5 times

fib(3) → n 3 times

inefficient

How to solve this problem of inefficiency?

⇒ 1. Store it in an array i.e. saare fib(2), fib(3) ke values ek array me store.

int[]f = new int[n+1]; → initially 0,0,0,0..

↳ ∵ starting from 0th fibonacci to nth

e.g. n=6

0	1	1	2	3	5	8	
0	1	2	3	4	5	6	

2. If fib(n) is already calculated

∴ (f[n] != 0) ↳ return f[n]

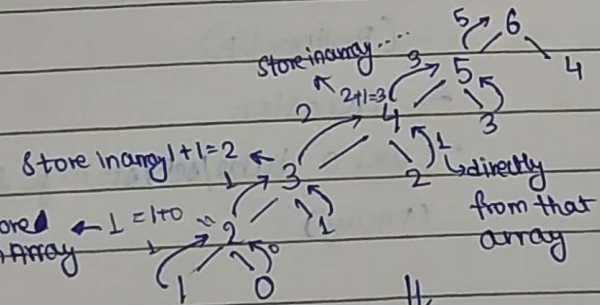
3. Storing continuously → storing these into f[n]

```
f[n] = fib(n-1,f) + fib(n-2,f);
return f[n]
```

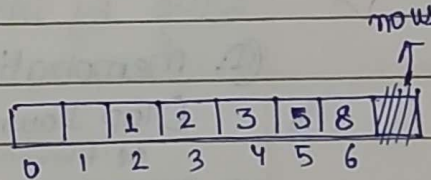
Hence,

earlier Time complexity was exponential $O(2^n)$ but now it's linear → O(n)

How?



∴ Skewed Tree (ek taraf jhuka hua rev. BT)



If we count operations here, for how many nos. fibo. was calc.

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

n+1

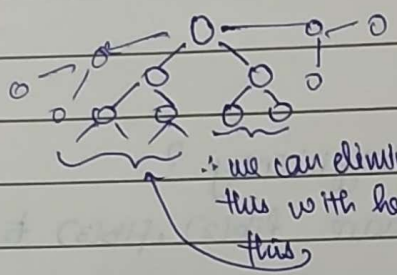
1

O(n)

This whole process of storage of memory for optimization for other recursive calls (for reuse) is called Memoization.

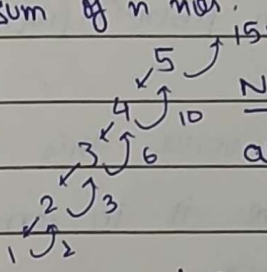
- What is DP?
  "Optimized Recursion"

★ How to identify DP?
a. Optimal Problem (like we did with Greedy)
b. Some choice got founded ie. which forms multiple branches in rec. tree.



∴ we can eliminate this with help of this.

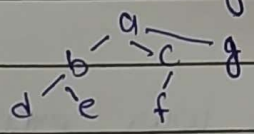
eg. sum of n nos.



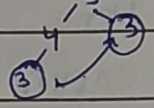
NOT DP
already optimized linearly
↓
better optimization = $\frac{n(n+1)}{2}$.

★ Definition: DP is a technique in programming that helps to efficiently solve a class of problem that have overlapping subproblems and optimal substructure property.

choices given



optimal

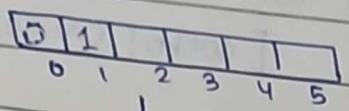


Ways of DP

① Memoization (Top Down)	② Tabulation (Bottom Up)
eg. fibonacci	
→ Recursion	→ Iteration
→ storage of subproblems for reuse.	→ uses a table/set/storage for iteration (tricky)

Note:- Tabulation is more efficient in comparison to Memoization.

Tabulation Process →

dp: 

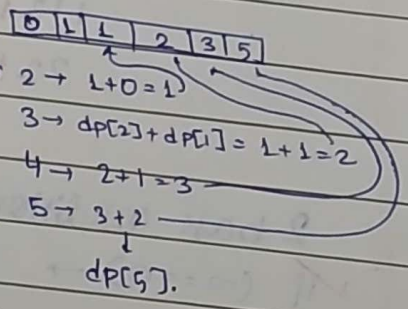
dp[i] → ith fib.
dp[0] → 0th fib
dp[n] → nth fib.

⇒ dp[n+1]

```
for (int i=2 ; i<=n ; i++){
    dp[i] = dp[i-1] + dp[i-2]
}
ans. = dp[n]
```

T.C here also → O(1)

*** steps :- (1) initialisation ↑ already known values size 0,1.
(2) meaning
(3) filling : from small to large



5/08/24

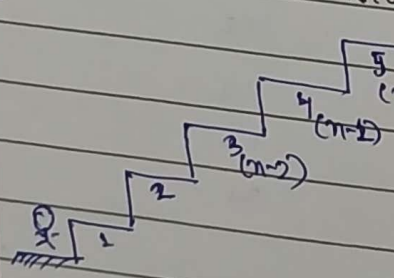
7 important Questions/Concepts

- Fibonacci
- Unbounded knapsack
- 0-1 knapsack
- LCS
- Kadane's Algo.
- Catalan No.
- DP on Grid (2D Arrays)

Climbing Stairs → [concept of Fibonacci no.]

Q. Count ways to reach the nth stair. The person can climb either 1 stair or 2 stairs at a time.

→



ways for 3
111
12
21

ways 4
1111
121
211
112
22

Therefore, ways for 5 will be same as (ways of 4)+1
*and (ways for 3)+2 [socho! v.v.easy]

∴ ways for 5

112	11111	n=5
122	1211	∴ 8 ways
212	2111	
	1121	Hence, ways(n) = way(n-1) + way(n-2)
	221	1step 2step

(Don't need to add step as we're just calc. no. of ways not steps)

2 base condition:-
if (n==0) → return 1; (Ground to Ground)
if (n<0) → return 0;
∵ if n=1 → ways(1) = ways(0) + ways(-1)
out of bound if no condition

Now memoization :- O(n)

(analyze krdo & using skewed tree & storing in array)
(same as fibo. nos.) ⇒ check code!

Ily, if asked we can climb 1, 2 & 3 stairs :-

ways for 2	ways for 3	ways for 4
11	111	1111
2	12	121
	21	211
	3	112
		22
		13
		31

∴ ways of 5 = ways(2) + ways(3) + ways(4)
2 steps 1 step2 1 step3
but ways will be same

Hence,
ways(n) = way(n-1) + ways(n-2) + ways(n-3)

Ily other problems.

DATE 06/06/24
PAGE

* 0-1 Knapsack

- Types of knapsack problems
  - → Fractional knapsack
  - → 0-1 knapsack
  - → Unbounded ".
- e.g. of 0-1 knapsack Problem :-

	value	weight
fridge →	100	10
laptop →	50	5
phone →	30	2
tablet →	25	1
pencil →	5	0.5
book →	15	1

W = 10 ,

In this problem we select max. valued item for 10 kg knapsack.
[In case of 0-1 knapsack, apart from greedy (frac. knapsack) we don't take fractional values to achieve max. value for full 10kg, rather here either we take it or ignore it for max. weight reached.

"0-1"

∴

case 1	case-2
fridge (full) val = 100	laptop, phone, tablet, pencil, book → 9.5kg value = 135

e.g. for Unbounded knapsack

case-1 "

case-2 "

case-3
↳ 10x tablets → wt = 10 kg
value = 250.

Q. val[] = 15, 14, 10, 45, 30
wt[] = 2, 5, 1, 3, 4.
W (total allowed weight) = 7
max Profit?

→ How DP Ques.?

⇒ 1. Choice (selecting numerous & putting in knapsack)

2. Optimal → when we need to find best soln from feasible soln possible.

Therefore :- item (val, wt)

valid → Include (1)

non-valid → exclude (0)

wt <= W

include, exclude (for selecting max. value)

Base condition → W capacity = 0 , index at n to 0

∴ if wt. = 0 so no items can be added so, ans = 0.

at item index = 0 ∴ no item left now so ans = 0

Hence, → (recursion)

```
knapsack (val[], wt[], W, n)
    if (W == 0 || n == 0)
        return 0;
    if ( wt <= W) → valid
    → include ⇒ W - wt , i - 1
    → exclude ⇒ W , i - 1
    else → not valid
    → exclude ⇒ W , i - 1
```

Date : 7/02/24

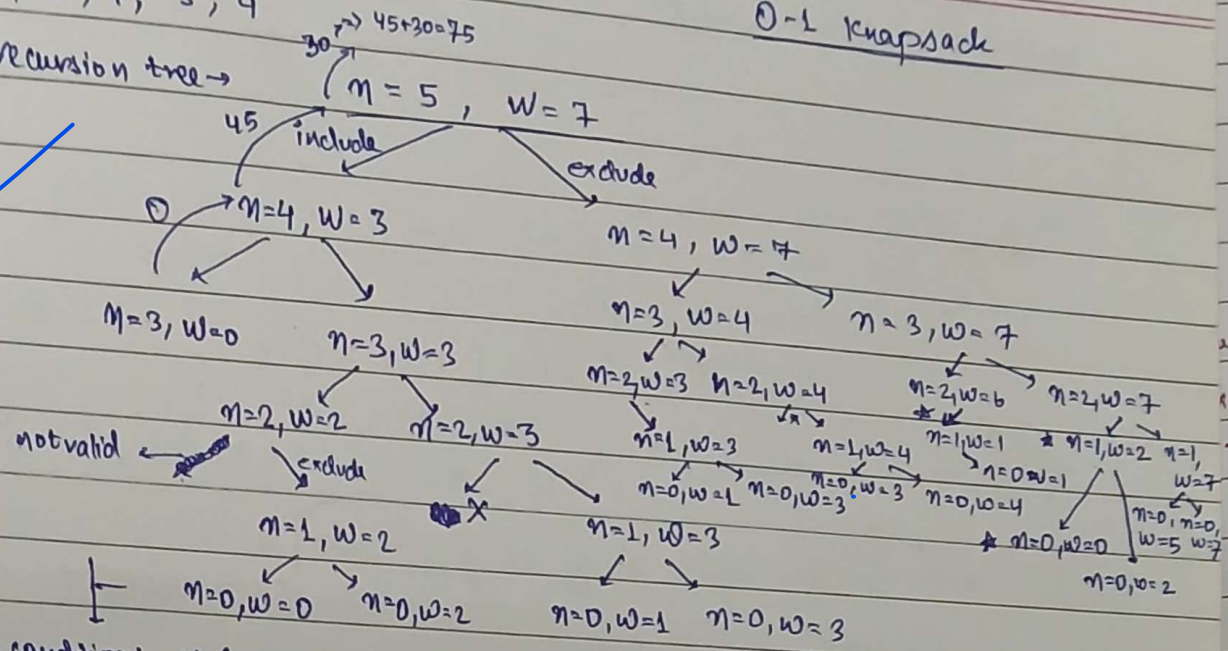
0-1 Knapsack

[] val = 15, 14, 10, 45, 30

[] wt = 2, 5, 1, 3, 4

W = 7

recursion tree →



base condition: if (n==0 || w==0)
return 0;

In case of valid use Math.max for include & exclude at every step return call.

- **Memoization →** val[], wt[] → fixed ✓
  w, n → variables in recursion → ∴ 2D array ✓
  n → wt.length

  array { from 0 to n+1 items
  and 0 to w+1 weights

  int[] dp = new int[n+1][w+1]

  initialise all with -1 ✓

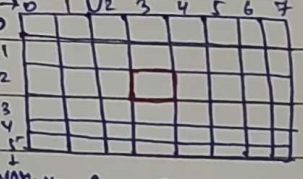
  if (dp[n][W] != -1) → filled/already stored
  ∴ return dp[n][W]

  Ith matrix dp[i][j]
  Knapsack (w) = j
  items = 0 to i
  for last row, last col → i = n
  j = W

  → check code

  → Time complexity : O(n * w) ∵ size of array

- **Tabulation →** ~~(1)~~. Create a Table : → (n+1) * (w+1) →



  Dp[][] (i,j) → i = items + j = W (knapsack size)

  (n+1) → size वाला array होगा तभी n → achieve होगा ★★★

  (1) Meaning ★ initialize

  at index i,j let (2,3) → n=2, W=3 → smaller problem of (15,2) & (14,5).

initialize → (Base case) → W = 0 ∴ profit = 0 ✓

n = 0 ∴ " " ✓

↳ (no item)

(5) Filling (bottom up)

all 0th row & 0th col → 0 for using Base cond^n

now → for (int i = 1 to n+1) { → item no.

for (int j = 1 to W+1) { → allowed weight (capacity) → isme galti kabhi mt karna!

val[i-1] ← (val, wt) → wt[i-1]

if (wt <= j) // valid, since j → capacity at that cell or,

now for this inside array early 2nd index is now at 1st index.

↳ at (i,j) weight capacity is j

I. Valid case 1 : include

value of that item + val[i-1] + dp[i-1][j-wt] ← now to calc max. profit of

val[i-1] → of pr

dp[i-1][j-wt] → stored val

j-wt

wt

15	14	10	45	30
2	5	1	3	4

∵ e.g. if (3,3) → i = 3, W = 3 kg

I_1(15, 2 kg), I_2 (14, 5), I_3 (10, 1)

① I_3 → selected → wt. = 1 kg ∴ left 3-1 = 2 kg → [j-wt]

② Now check for filling in I_1 & I_2 → dp[i-1]

case 2 : exclude

dp[i-1][j] (simple)

∴ dp[i][j] = math.max (case 1, case 2) → store in dp.

II. Invalid → dp[i][j] = dp[i-1][j]

final ans return → dp[n][W] which means all items considered under capacity.

since last wala hi saara can consider karega

dp[1][1] → dp[1][1] ⇒ for this W = 1 kg, n = 1 item only. → check dp array

For better understanding watch how dp array filled in video.

Target Sum Subset

Variation of 0-1 Knapsack

numbers[] = 4,2,7,1,3

Target Sum = 10 → True (can be achievable)

∴ subsets ⇒ {7,3}, {7,2,1}, {4,2,3,1}, ...

How DP? → (1). Choice of elements (p,wp)

(2). limit on max. allowed capacity

(3). val = wt

val = wt

target Sum → max. allowed capacity

How of knapsack

Using Tabulation:- i j (target sum)

1. Table. → dp[n+1][sum+1]

2. meaning + initialize

3. bottom up manner (Filling) → small to large.

Unlike in o/l knapsack we were storing quantities as max Profit but here we have to check True/False

* meaning → (boolean type T/F)

ans → n items ⇒ subset sum = target? T/F

(smaller problem) * dp(i,j) → i item ⇒ subset sum = j? T/F

eg. dp(3,5) → 3 items subset sum = 5 (j)

using [4,2,7] we can't make 5 ∴ False.

{}, {4}, {4,2}, {7}, ..

initialise (base case) → 1. sum = 0 ∴ True using {}/ϕ.

(j=0)

2. items=0 (i=0) → sum > 0 ∴ False always

(j=1 to sum)

∴ 0th row from index 1 → all False

0th col → all True

Note:- in boolean type method, ~~initialised~~ array is initialised using False only. ∴ No need to declare this.

Filling :→ small → large

i↓ j↓ i↑ j↑

clock → Note: val[i-1] was used bcoz at base condt row & col there's an extra col.

Date | 18 | 08 | 24
Page No.

* Rod Cutting -

Given a rod of length n inches and an array of prices that includes prices of all pieces of size smaller than n. Determine the max. value obtainable by cutting up the rod and selling the pieces.

length = 1 2 3 4 5 6 7 8
price = 1 5 8 9 10 17 17 20
rod length = 8 .

case 1 → 4 , 4 → 8 + 8 = 16

case 2 → 4 , 1 → 17 + 1 = 18

case 3 → 2 , 6 → 5 + 17 = 22

∴ same as knapsack problem. "for max. Profit".

"Unbounded knapsack"

price = val
length = wt
total rod = W

code :-

```
for (int i=1; i<n+1; i++){
    "  "  j  "  "<W+1; "  "){
    if( wt[i-1] <= j){ //valid
        dp[i][j] = Math.max(val[i-1]+dp[i][j-
                                wt[i-1]],
                            dp[i-1][j]);
    }
    else{ //invalid .
        dp[i][j] = dp[i-1][j];
    }
}
```

Here, dp(i,j) = for 'j' total rod length how many i pieces do we need to get for max. Profit. .

Teacher's Signature.....

Date | Page No.

** Longest Common Subsequence (LCS)

imp.

A subsequence of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of remaining characters.

eg. abc and ac

abc
ac
ab
bc
a
c
""

ac
a
c
""

∴ from both common subsequences one with longest (length) size is ac

∴ ans = 2.

eg. abcde and ace ⇒ Ans. = 3

recursion approach :- (1) Base condition

(2) Bigger to smaller problem.

str1 = "abcde" (m), str2 = "ace" (n)

ans = 3 "ace"

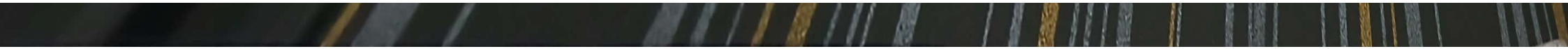
for ~~to~~ checking in strings of sizes m & n first find for smaller problem.

∴ str1 → n-2 , str2 → n-2

str1 → n-1 , str2 → n-1

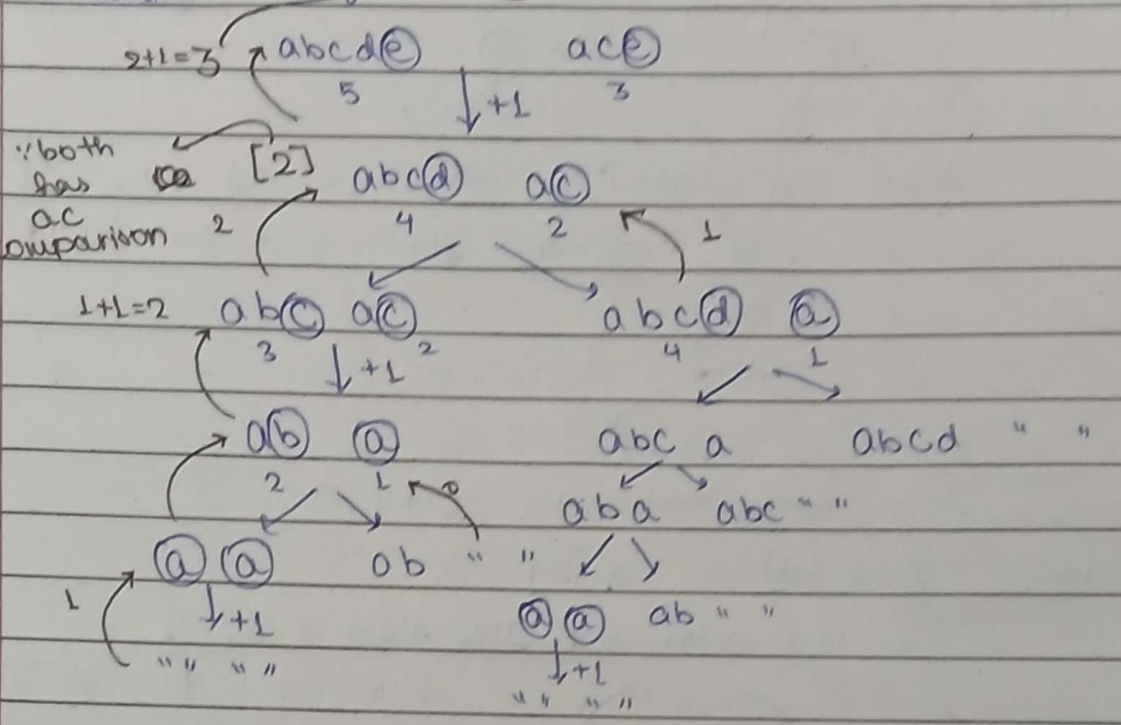
str1 → n , str2 → n

Teacher's Signature.....



Date
Page No.

Recursive tree :- → (3) ans.



∴ Concept "code" ⇒ lcs (str1, str2, n, m) {

Base cond :→ if (n == 0 || m == 0)
return 0;

same → if (str1(n-1) == str2(m-1))
return lcs (str1, str2, n-1, m-1) + 1

Different → else ans1 = lcs (str1, str2, n-1, m)
ans2 = lcs (str1, str2, n, m-1)
return Math.max (ans1, ans2)
}

- Memoization approach

⇓

Because of overlapping subproblems
∴ We make a 2D array since 2 variables are changing (n and m) after every recursion call.

Teacher's Signature.....

Date
Page No.

for → str1 = "ABCD" n=4 str2 = "ACEB" m=4

∴ n+1 → rows
m+1 → cols.
dp[n+1][m+1]

meaning → at (i,j) which string will be stored :-
→ eg. abc (n=3), ce (m=2) ∴ at (3,2)
eg. ABC (n=3); ACE (m=3) ∴ at (3,3).

∴ (i,j) → str1(i), str2(j) → LCS

for final ~~ans~~ i=n & j=m (last row, last col).

Hence, fill all rows & col with -1.

if (dp[n][m] != -1)
return dp[n][m]
else → LCS → same / diff.

Time complexity : O(n*m).

- **Tabulation approach**

↳ helps in preventing from recursion, because of stack overflow issue.

3 steps :- (1). table create → dp[n+1][m+1]
(2). meaning + initialize ↓ (base condition)
(3). fill (bottom up)
↓
from small to large → solve for smaller problem ∴ eventually can solve for larger problems.

for eg. → str1 = "abcde" str2 = "ace"

str1(n), str2(m)
i, j

meaning → for (i,j) → str1(i), str2(j) → LCS ?

eg. (3,2) → i=3 "abc", j=2 "ac" → LCS ? = ac ∴ @(3,2) "ac" is stored

Teacher's Signature.....

Date
Page No.

initialise → base condition → same as of recursion → that is length of str1 or 2 is zero ∴ return 0.

⇓

@ i=0 and j=0

∴ 1st row & 1st col will be zero.

filling → (small to large) (1 → n+1)

```
for (int i=1 to n+1) {
    for (int j=1 to m+1) {
        if (str1(i-1) == str2(j-1)) {
```

comparing last character of both strings

Now in rec →	n-1 , m-1
	same ∴ +1
here →	i-1 , j-1
	same ∴ +1

```
∴ dp[i][j] = dp[i-1][j-1] + 1.
        }
        else {
            ans1 = dp[i-1][j]
            ans2 = dp[i][j-1]
            dp[i][j] = max(ans1, ans2).
        }
```

- Time Complexity: O(n*m).

Teacher's Signature.....

Date 19/08/24
Page No.

* Longest common substring :-

Note :- a substring is a contiguous sequence of characters within a string.

e.g. S1 = "ABCDE" , S2 = "ABACE"

Longest common subsequence = ABCE

" " substring = AB.

*eg.2 S1 = "ABDEFG" , S2 = "ABCDEFGH"

L.C. substring = EFG.

- Tabulation approach :-

① 2D array (n+m) ⇒ characters same → length+1

② meaning, initialise ⇒ " different → 0

③ filling up. str1, n str2, m substring

str1, n-1 ↓ str2, m-1 substring.

1. Now → rows :- 0 - n+1 } +1 since calculating from 0 to n
   cols :- 0 - m+1 } 0 to m

2. meaning → (i,j) → str1(i) , str2(j)
   longest common substring will be stored
   eg. i=1 (A)
   j=2 (AB)
   ∴ at (1,2) → A → 1

3. initialisation → n=0, m=0 → ∴ 0.

substring (n,m)

same
∴ (dp(n-1), (m-1)) + 1
dp[i][j]

different
dp[i][j] → 0
counter resets

Teacher's Signature.....

Date | Page No.

Yeah

Note:- here it's not certain that the final answer will be at (i=n, j=m), here it can be present at any cell not considering full string length.

∴ ans=0

ans = max(ans, dp[i][j])

Time complexity: O(n*m)

20/08/24

* Longest increasing subsequence

⇓

arr[] = {50, 3, 10, 7, 40, 80}

Length of LIS = 4.

meaning longest ascending (sorted) → [3, 10, 40, 80] → (4) longest

7, 40, 80

3, 10

10, 40, 80

eg. arr[] = {5, 1, 6}.

LIS = 2 ∴ [5, 1, 6, 56, (16), 56, 516]

all sub-sequences

longest sorted increasing subsequence

Solving approach: eg. {50, 3, 10, 7, 40, 80} — A

sorted ⇒ {3, 7, 10, 40, 50, 80} — B

If we'll find the LCS of A and B we'll get:-

⇒ 3, 10, 40, 80 } i.e. 4 ⇒ Ans.
3, 7, 40, 80 }

Hence, LIS = Longest common sorted unique subsequence.

↓

since if we've got any duplicate elements then take only 1.

e.g. 50,10 ≡ 50 ✓

Teacher's Signature.....

"Benefit of Hashset is that it stores only unique elements."

Date | Page No.

∴ Steps :- ① Hashset → used to find unique elements.

② arr2[] (m) — length of array

arr1[] (n) → LCS ⇒ LIS

→ m = n when elements were already unique.

• Time Complexity : O(n*m).

code ⇒ public static int lis (int[] arr1) {

→ Hashset<Integer> set = new Hashset<>();

for (int i = 0 to arr1.length) {

set.add(arr1[i]);

int[] arr2 = new int[set.size()]; → sorted unique elements wala array.

int i = 0;

for all numbers in the set ⇒ for (int num : set) {

arr2[i] = num; → storing those nos. in arr2

i++;

}

Arrays.sort(arr2); → ascending sorted

return lcs(arr1, arr2);

lcs wala code ⇒ static int lcs (int[] arr1, int[] arr2) {

(for arrays)

↓

Concept samjho + concept revise

kaise karoge toh hi yaad rahega.

Koi ek sheet select karo and use lagao.

*** Imp. for placements :→ ① DSA ki sheets (Ques.) e.g. 450 Que.

② Web dev karo + projects banao

③ OS + DBMS

Teacher's Signature.....

Date | Page No.

20/08/24

** Edit Distance

↓

* Given 2 string word1 and word2, return the min. no. of operations required to convert word1 to word2.

Using :- (1). Insert a character
(2). Delete " "
(3). Replace " "

Process of converting WORD1 into WORD2

eg. word1 = "intention", word2 = "execution"

∴ intention —remove 't'→ inention ①

② ↓ replace 'i' with 'e'

exention ③ ←replace 'n' with 'x'— enention

④ replace ↓ 'n' with 'c'

exection —insert 'u' ⑤→ execution

dis. = 5

Similarity from LCS :-

word1 : abcdef (n)
word2 : bdeg (m)

① . Same (if last character of both words are same.

② Different

(a). Add

Suppose we added 'g' in word1.

counting of operation ↓

∴ abcdefⓖ → str1(n) str2(m-1) + 1

bdeⓖ

Since, now comparing b/w n → abcdef and (m-1) → bde

Teacher's Signature.....

(b) Delete

word1: a b c d e ✗
word2: b d e g

∴ now compare in :-
str1(n-1) str2(m) +1

v. easy cichud x socho why

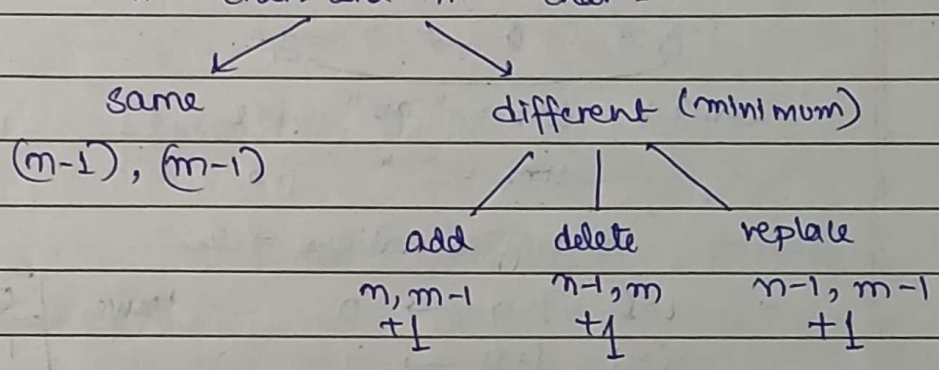
(c). Replace

1: a b c d e (f) → replace with 'g'
2: b d e g

now compare in :-
str1(n-1) str2(m-1) +1

Ans. for different will be minimum of (a), (b), (c).

• Tree :-



Hence, Similarity :-

LCS	Edit Distance
2 strings	2 strings
output: int (length of LCS)	output: int (no. of operations)

• ∴ Tabulation approach :-

(1). 2D array : dp[n+1][m+1]

(2) meaning : eg. "abc" "sbd"
n=3 m=3

∴ in 2D array eg. (1,2) → i=1, j=2

dimag ka use karo ∴ [at (1,2) => 2 will be stored] a sb
↓ replace with 's'
s → add 'b' → sb ∴ 2 operations

Teacher's Signature.....

Date
Page No.

initialisation: if ~~one~~ of string 1 is empty " " then the no. of operations will be just as the length of str2 (Jitne elements hai in str2 usko ek-ek karke add karde in str1 or viceversa → remove)

eg.

ie. case 1: str1 = " " i=0

str2 = "sba" j=3

∴ no. of operations = 3.

∴ if j=0 → i=0 → no. of operation = 0

j=1 → i=0 → " =1

j=2 → i=0 → " =2

j=3 → i=0 → " =3

case 2: str1 = "ab" i=2

str2 = " " j=0

∴ no of operations → 2.

∴ i=0 j=0 → 0

i=1 j=0 → 1

i=2 j=0 → 2

i=3 j=0 → 3

Hence.

	0	1	2	3
1				
2				
3				

pseudocode for initialisation:

```
for (int i=0 to n+1)
    for (int j=0 to m+1)
    if (i==0) → dp[i][j] = j
    if (j==0) → dp[i][j] = i
```

(B) Smaller to large "filling" :-

here i = length of string 1

j = " " " " 2

Same ⇒ if (str1(i-1) == str2(j-1)) {

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

} else

⇒ dp[i][j] = min.(dp[i][j-1] , dp[i-1][j] , dp[i-1][j-1]) + 1

}

add delete replace.

Teacher's Signature.....

Note (Reminder) :- We can't compare 3 numbers in min operation, therefore we it nested wise.

Time Complexity: $O(n * m)$.

→ code yourself → v.v.v. easily.

- (Variation) String Conversion

Convert string1 to string2 with only insertion & deletion.

"Not Directly" ⚠ → (normally gives ans. = 4) but using this approach ans. = 3.

e.g. str1 = "pear" str2 = "sea"

remove p, remove r, add s → sea

lcs

Note:- If we want to convert str1 to str2 :- indirect (quick) method

1. Find LCS (of both) and delete extra characters of string1.

str1 = abcdef → lcs = ace ∴ remove b, d, f from str1.

str2 = aceg ↓ now add 'g'

2. now add remaining.

1. No. of delete operations → length of str1 − length of LCS

6 − 3 = 3

2. No. of insert operations → length of str2 − length of LCS

4 (aceg) − 3 = 1

∴ 3+1 = 4 ans.

Code → yourself ✓ easy.

Teacher's Signature.....

21/08/24

Date | Page No.

Wildcard Matching

Given a text and a Wildcard Pattern, implement wildcard pattern matching algorithm that finds if wildcard is matched with text. (it must cover entire text not partial).
Wildcard include '?' and '*'.

'?' - matches any single character. (but not with ϕ)

eg. text = "ab" p = "a?"

replace ? with b

∴ "ab" ✓

'*' - matches any sequence of characters (including empty sequence)

eg. text = "aa" , p = "*"

replace * with aa ✓

⇓

If matching possible ≡ True

not possible ≡ False.

e.g.1 Text = "baaabab"

Pattern = "*****ba*****ab"

output = true. (***** → replace with ϕ; ***** → aab; ***** → ϕ)

e.g. Text = "baaabab"

Pattern = "a*ab"

o/p = false.

Variation of LCS

Teacher's Signature.....

Date
Page No.

① n, m = length of both strings

② n↓, m↓ ⇒ smaller problems → s = "abcdef" (n), p = "ab*?d" (m)

1. Create dp table of 2D array. ∴ n, m
   (n-1) s = "abcde" p = "ab*?" (m-1)
   ⋮
   (n=0) s = "" p = "" (m=0)

2. assign meaning + initialise
   dp(i, j) → s(i), p(j) → T/F ~~maths~~ ⇒ check at (i,j) that it's possible or not.
   eg: i=1, j=4 ⇒ s="a", p="**?b"
   not possible
   ∴ at (1,4) → False will be stored.

initialise — (1). s=0, p=0
Now since length of s → i
and " " p → j
∴ at i=0 and j=0
and s="", p="" → True ∴ at (0,0) → True

(2). for s≠0, p=0 ∴ Khali ~~str~~ pattern kabhi kisi string se match nhi kar skti (whose length ≠0)
e.g. s="aa", p="" → False. (can't make aa from empty string)
∴ for j=0, i≠0 → False

(3) for s=0, p≠0
- s="" p="*" → True, p="ab*" → False (ab converts to ∅)
- s="" p="?" → False
- s="" p="b" → False

but what about these

Teacher's Signature.....

Here, s = "" p = "*a(*)"

j^th character → ? or char → False

j^th character → '*' ∴ check for dp[0][j-1]

eg. s = "" , p = "**" → True

s = "" , p = "a*" → False.

pseudocode →
```
for (int j = 1 to m)
    if (p.char(j-1) == '*'){
        dp(0,j) = dp(0,j-1)
    }
```

test

{ Jaise * mile in 'p' check for last filled block jo usme hoga wohi usme hoga.

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

s = "abc" → n = 3
p = "**?b*" → m = 5

(1). 0,1 → s = "", p = "*"
(2). (0,2) → s = "" p = "**" star ∴ check for prev. block as (0,1) (left) T check prev.
(3) (0,3) → s = "" p = "**?" ?∴ false True ∴ True
(4) (0,4) → p = "**?b" → False
(5) (0,5) → p = "**?b*" last j = * ∴ check previous False ∴ False

3. Filling (bottom up)
1st case: a. eg. s = "abc" , p = "adc"
i^th → same ← j^th
Moving in array ← ∴ dp[i][j] = dp[i-1][j-1]

b. eg. s = "abd" , p = "ab?"
"?" can be replaced with 1 char.

∴ if (s.charAt (i-1) == p.charAt(j-1)) || p.charAt(j-1) == '?'){
    dp[i][j] = dp[i-1][j-1].
}

II nd case * → s = "abcd"   p = "aab*"

* → ignore ↓ * == "" ∴ s = i , P = j-1
* → case s = i-1 , p = j ↓ * compared now since it can take multiple items ∴ It will check also for next item.

if any of them gives true ∴ True ∴ use 'OR'

Therefore → if (p.charAt (j-1) == '*'){
    dp[i][j] = dp[i][j-1] || dp[i-1][j].

III rd case dp[i][j] == false.

Time complexity = O(n * m).

* Catalan's Number ★★★

Fixed ⇒ [$C_0 = 1$, $C_1 = 1$]   { same as fibo. }

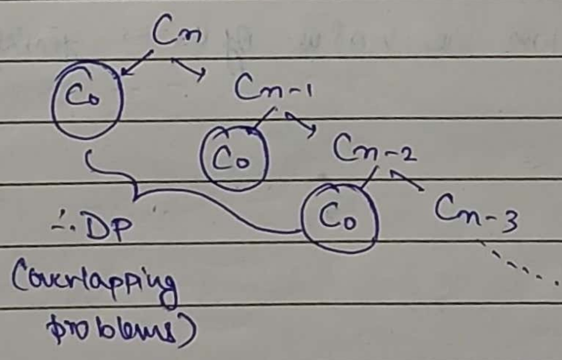
$C_2 = C_0 C_1 + C_1 \cdot C_0 = 2$

∴ $C_3 = C_0 \cdot C_2 + C_1 \cdot C_1 + C_2 \cdot C_0 = 5$

∴ [$C_n = C_{n-1} \cdot C_0 + C_{n-2} \cdot C_1 + C_{n-3} C_2 + \ldots\ldots + C_0 C_{n-1}$] ★

eg. $C_4 = C_3 C_0 + C_2 C_1 + C_1 C_2 + C_0 C_3 = 5 + 2 + 2 + 5 = 14$.

Why DP?:-



Teacher's Signature.................................

Date | Page No.

I. Recursion Approach

∴ for 0 we have $n-1$ ∵ $i=0$ ∴ $n-0-1 = n-1$

1 ⟶ $n-2$ ∵ $i=1$ ∴ $n-1-1 = n-2$

2 ⟶ $n-3$ ∵ $i=2$ ∴ $n-i-1 = n-2-1 = 3$

i ⟶ $n-i-1$

for (int i=0 to n-1)

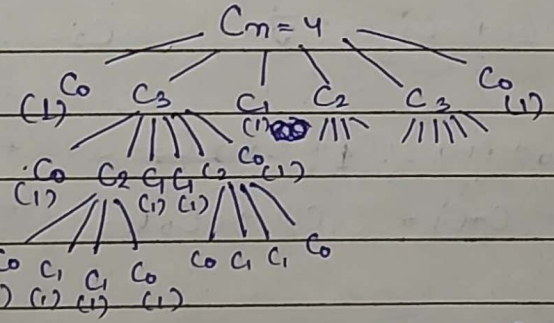
$$C_n \mathrel{+}= C_{n-i-1} + C_i$$

eg. $C_3 = C_2C_0 + C_1C_1 + C_0C_2$

Pseudo code* ⇒

```
catalan (n) {
if (n = 0 || n = 1){
    return 1
}
int ans = 0;
for ( int i=0 to n-1){
ans += catalan(i) * catalan(n-i-1)
}
return ans.
}
```

• rec. Tree ⇒



here if you will store the value of C_2 → you'll save 16 such calls!

Teacher's Signature.....

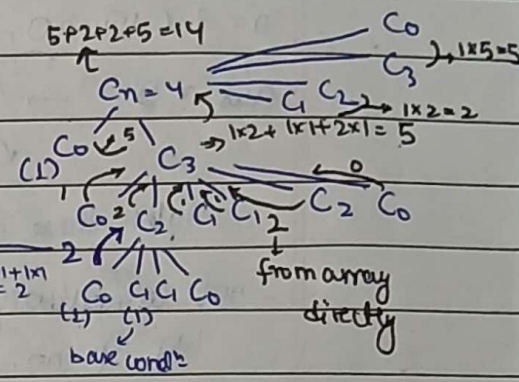
memoization Approach

(1D array) → ∴ "n"

∴ fill dp array (n+1) with -1. ∴ 0 → 4 ∴ (0 → n+1).

⇒

```
cat (n) {
if (n == 0 || n == 1)
    return 1;
if ( dp[n] != -1 )
    return dp[n]
for(int i=0 to n-1) {
}
return dp[n] = ans.
```



∴ dp

1	1	2	-1	-1
0	1	2	3	4

Tabulation Approach

1. table
   ∴ 1D array dp[n+1]
2. initialize
   dp[0] = dp[1] = 1

1	1			
0	1	2	3	4

3. filling

```
for(int i=2 to n) {          → for filling in dp array
    for (int j=0 to i-1){
    dp[i] += dp[j] * dp[i-j-1]   } for calculating Ci
    }
```

dry run & check for yourself.

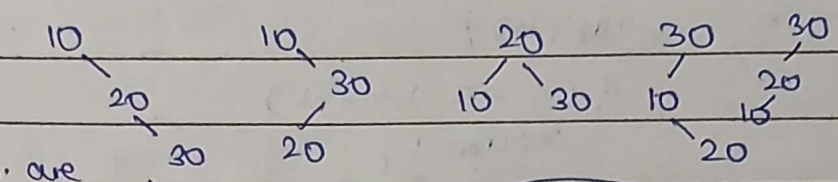
Date | Page No.

* Counting Trees [type of Catalan No. Ques.] → think wisely

Find no. of all possible BSTs with given n nodes.

n = 3 (10, 20, 30)

Ans = 5 →



note:- these nos. are not compulsory, agar koi bhi 3 no. le le tab bhi 5 hi aayega.

5.

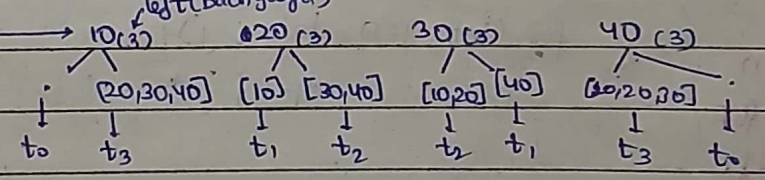
	BST	
n=0	1	(null node)
n=1 [10]	1	(10)
n=2 [10,20]	2	10 (20), 20 (10)
n=3 [10,20,30]	5	→ 10 ([20,30]), 20 ([10] [30]), 30 ([10,20])

left → t_0, right → t_2; t_1 t_1; t_2 t_0

$\Rightarrow t_0 \cdot t_2 + t_1 \cdot t_1 + t_2 \cdot t_0$

$= 1 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 = 5.$

n=4 [10,20,30,40] 14 →



$\Rightarrow 1 \cdot 5 + 1 \cdot 2 + 2 \cdot 1 + 5 \cdot 1 = 14.$

↓ expand the tree & verify.

Hence,

e.g. n = 5 ⇒ Catalan of 5 ⇒ No. of BSTs.

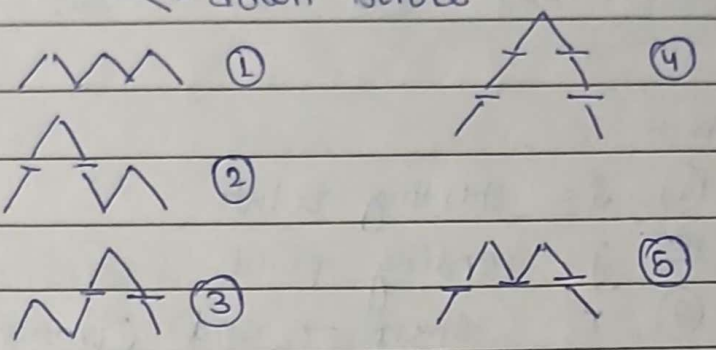
Teacher's Signature.....

- **Mountain Ranges**

at any moment the no. of down strokes cannot be more than no. of up strokes.

/ = up stroke \ = down stroke

pairs = 3 ⇒ 5 combinations



Hence,

∴ Pairs = n ⇒ Catalan of n ⇒ Catalan of n Combinations.

Ans

* Matrix Chain Multiplication → "DP on grid"

(7 Concepts in DP)

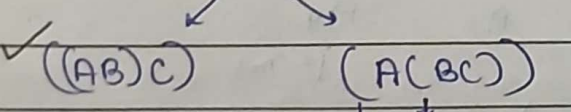
arr[] = {1, 2, 3, 4, 3}

Find min. cost.

matrix:	A	B	C	D
order:	1×2	2×3	3×4	4×3

✓ You firstly have to check the multiplication of ABCD is feasible or not then look for min. cost [cost ⇒ for A 1×2, B 2×3 ⇒ 1×2×3 ⇓ 6]

e.g. A 1×3, B 3×5, C 5×6



((AB)C): AB: result = 1×5, cost = 1×3×5 = 15 : A1

A1·C: result = 1×6, cost = 1×5×6 = 30 : A2

Total cost = 30+15 = 45

(A(BC)): BC: result = 3×6, cost = 3×5×6 = 90 : A1 (feasible multiplication)

A·A1: result = 1×6, cost = 1×3×6 = 18 : A2

Total cost = 90+18 = 108

min Cost ⇒ Ans.

Teacher's Signature.....

Date | Page No.

Now, from the Ques.

	(A_1)	(A_2)	(A_3)	(A_4)
~~arr[]~~	A	B	C	D
arr[] = {0, 1, 2, 3, 4, 3}	1×2	2×3	3×4	4×3

(((AB)C)D) (AB)(CD) (A(BC))D) .....

Therefore :-

1. i = starting point
2. j = ending point
3. k = start → end (traverse) → from i to j

$A_1 = arr[0] \times arr[1]$

$A_2 = arr[1] \times arr[2]$

$\therefore A_i = arr[i-1] \times arr[i]$.

① $\boxed{A_i = arr[i-1] \times arr[i]}$ {i → starting from 1} (obvious reasons)

Now dividing into subproblems for help with recursion

$\therefore A_1 A_2 A_3 A_4 \ldots A_{n-1}$

$A_1 \ldots A_k$ | $A_{k+1} \ldots A_{n-1}$

X | Y

mtlb chote problems ki madad se bade problems nikalo.

$\rightarrow (X \cdot Y)$ ✓

also for (AB)(CD) | A(BCD) ....

Set 1 Set 2 | Set 1 Set 2

Approach ⇒ ① $A_1 | A_2 A_3 A_4$.

$i=1$ $j=n-1$

$k=1$

(i,k) | $(k+1, j)$

$(2, 4)$

Set A_1 | Set $A_2 A_3 A_4$ = Y

$\therefore A_1 \cdot Y$ ✓

Teacher's Signature.....

last

Date | Page No.

② Now K=2 ∴ A_1 A_2 | A_3 A_4 (i=1, K=2, j=n-1)

(i,K) (K+1, j)

(1,2) (3,4)

A_1A_2 A_3A_4

$(A_1A_2)(A_3A_4)$ ✓

③ K=3 → i,K K+1,j

(1,3) (4,4)

$(A_1A_2A_3)$ (A_4)

$\therefore ((A_1A_2A_3)A_4)$ ✓

Hence, 'K' will traverse from i to j-1.

=> int mcm(arr[], int i (1), int j (n-1)) {

for (int K=i to j-1)

cost1 = mcm(arr, i, K);

cost2 = mcm(arr, K+1, j)

$A_i = arr[i-1] \times arr[i]$

$A_k = arr[k-i] \times arr[k]$ → result1 = [a (arr[i-1]) × b (arr[k])]

a: row of A_i (first matrix) b: col of A_k (last matrix)

$A_{k+1} = arr[k] \times arr[k+1]$

$A_j = arr[j-1] \times arr[j]$ → result2 = c (arr[k]) × d (arr[j])

also from this we get to know

Now if we see clearly multiplication of cost1 & cost2 is also valid. (b=c)

Therefore, ques. ki aise hi set karke diya hoga jisse cost1 + cost2 multiplication feasible ho.

∴ cost 3 = a×b×d = arr[i-1] × arr[k] × arr[j]

final cost = cost1 + cost2 + cost 3.

Teacher's Signature.....

Date
Page No.

✓ cost 1 : $A_i \cdots A_k$

✓ cost 2 : $A_{k+1} \cdots A_j$

✓ cost 3 : cost of $\underbrace{((A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j))}_{\text{cost 3}}$

make int ans = +∞ → i.e. (integer. max value)

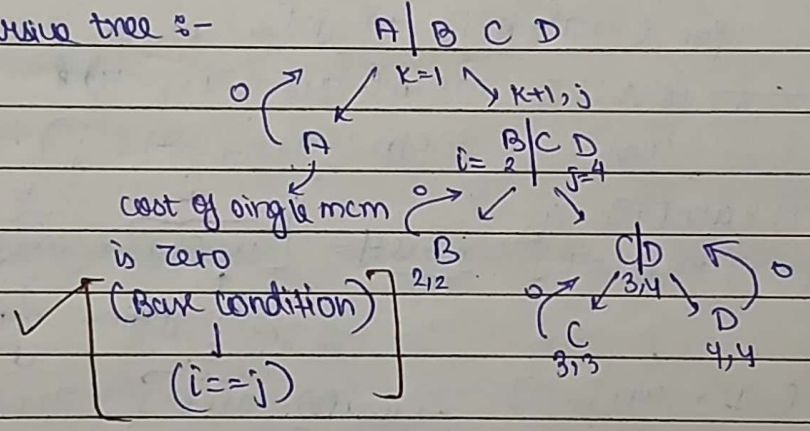
to check & compare from final cost

initialise ans ← ✓ ∴ in loop → ans = min (ans, final cost)

to ∞ so that to give ans. as final cost only after getting least value of final cost.

(no ∀ need to check for condition (here), because it'll already be make auto via Question).

recursive tree :-



Pseudo code ⇒

```
int mcm (arr, i, j) {
if (i==j){
return 0;
int ans = ∞
for (int k = i to j-1){
cost 1 = mcm (arr, i, k)
cost 2 = mcm (arr, k+1, j)
cost 3 = arr[i-1] x arr[k] * arr[j]
final cost = cost 1 + cost 2 + cost 3
ans = min. (ans, final cost)
}
return ans;
}
```

for first ∞, set as ∞ bcoz ans. hamesha final cost walabi aaye

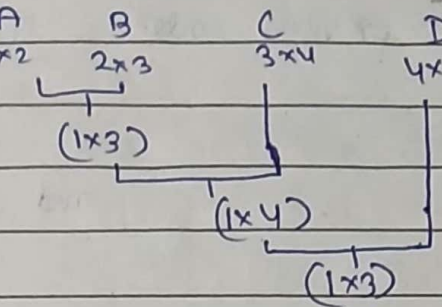
+∞, 108 → 108 is stored in ans

Now, (108, 45) → 45 stored in ans

Teacher's Signature.....

Date | Page No.

∴ ans = 30 ∵ A (1×2), B (2×3), C (3×4), D (4×3)

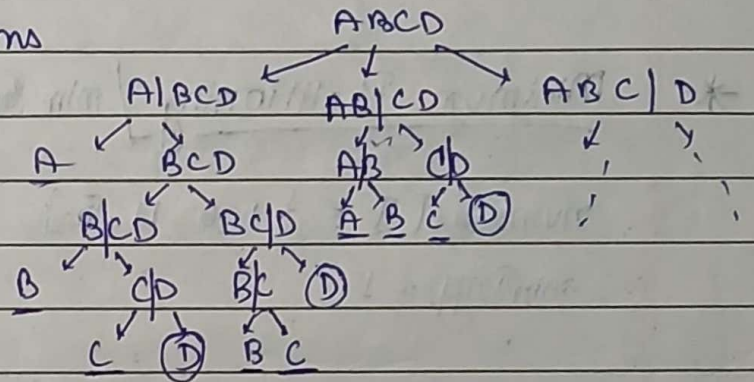


final result (does not matter).

∴ 1×2×3 = 6
1×3×4 = 12
1×4×3 = 12
30 Ans.

- **Memoization approach**

due to overlapping problems in recursion.



✓ Why DP?
→ ① choices
② overlapping substructure

✓ 2D Array since → i, j (1 to n) (n-1, 1) (1 to n-1).

∵ → A|BCD (i=1, j=n-1) (i,k) (k+1,j)
A (i=1,j=1) BCD (i=2,4)

✓ ∴ dp[n][n] & fill with -1.
i (1 to n) j (1 to n)

✓ arr[] = {1, 2, 3, 4, 3} → n=5 ∴ 5×5 matrix

✓ meaning of each block (i,j) → eg. (1,2)
1×2 (A1) 2×3 (A2) ∴ 1×2×3 = 6 will be stored

✓ Is bhi cell (i,j) pe main hun, agar mai i se j tak matrix ko multiply karu toh waha pe un matrix ka multiplication ka min cost kya hai.

Teacher's Signature.....

Date | Page No.

✓ changes from previous code :- if (dp[i][j] != -1)
return dp[i][j]
}

and store min. ans in dp[i][j]

✓ Time Complexity: $O(n^2)$ :: $(O(n*n))$.

- Tabulation Approach
↓
Intuitive
(:. leave for now)

* **Minimum Partitioning** / min. Subset sum Difference / Partitioning Subsets

numbers[] = {1, 6, 11, 5}
minDiff = 1.

⇒ aisa set choose karo jin do ke sum ke partition, ~~ka~~ difference min. aaye

{11, 5} ⟷ {1, 6}
set1 set2
↓ ↓
16 7 ∴ 16-7 = 9

Ily, {1, 11} = 12 {6, 5} = 11 ⇒ (1) → min. ans
{1, 5} = 6 {6, 11} = 17 ⇒ 11

→ minimum difference will be when sum of set 1 ≈ sum of set 2

$$|Sum2 - Sum1| = min.$$

Teacher's Signature.....

Date | Page No.

So in order to make this minimum we'd make sure to put sum/2 elements (or close to it) in each set.

Sum ⇒ sum of all nos. i.e. 1+6+11+5 = 23

∴ sum/2 ≃ 11.5

∴ sum/2 → Set 1 (11,1) ; sum/2 → Set 2 (6,5)

also if we select elements for set1 ∴ remaining will automatically goes to set2.

∴ fill set1 only & put rest in 2.

∴ capacity (W) = sum/2 ← max. weight

and select/ignore element in such a way to get min. Difference

Variation of 0/1 knapsack.

Valid: take it / ignore it ∴ (min-Diff.)

Invalid: e.g. W=7 and nos. = {1, 12, 3} — 12 is invalid

Also, when wt. & val are not given separately ∴ take val ⇒ wt.

Pseudocode ⇒ (steps)

1. find sum of all nos.
2. dp[n+1][W+1] (n+1 → nums.length; W+1 → sum/2) here dp[5][12] (12 → 11+1 → 23/2 = 11)
3. 0/1 knapsack
4. dp[n][W] = sum 1

sum 2 = sum − sum 1

min. Diff. = | sum1 − sum2 |

↓

Math.abs (sum1 − sum2).

Teacher's Signature.....

Date
Page No.

Code => psum 1

```
int[] nums = {1,6,11,5};

static int minPartition(int[] arr){
    int n = arr.length;
    int sum = 0;
    for(int i=0; i<arr.length; i++){
        sum += arr[i];
    }

    int W = sum/2;
    int dp = new int[n+1][W+1];

    for(int i=1; i<n+1; i++){
        for(int j=1; j<W+1; j++){
            if(arr[i-1] <= j){ // valid
                dp[i][j] = Math.max(arr[i-1] + dp[i-1][j-arr[i-1]],
                                    dp[i-1][j]);
            }
            else {
                dp[i][j] = dp[i-1][j];
            }
        }
    }

    int sum1 = dp[n][W];
    int sum2 = sum - sum1;
    return Math.abs(sum1 - sum2);
}
```

getSum

this (val = wt).

true

ignore

// Knapsack code ←

Kadane's Algo. : Max. Subarray Sum.

Date | Page No.

- **Minimum Array Jumps**

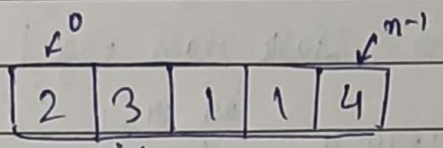
↓

to find min. jump to go from start index (0th) to end index (n-1).

∴ dp[i] : i^{th} index se n-1 tak min. jump.

int[] arr = {2, 3, 1, 1, 4}

min Jumps = 2.



choti problem (already calculated)

1 → 2 (step=1) jumps = dp[2]+1

1 → 3 (" =2) " dp[3]+1

1 → 4 (" =3) " dp[4]+1

∴ going from n-1 → n-1 } smallest problem

3 → n-1

⋮

0 → n-1 } longest problem

∴ using n-2 to 0 in for loop

∴ n-1 → n-1 = 0 steps (base) (already calculated)

ans = +∞

Pseudocode ⇒ for (int i = n-2 to 0)

int steps = arr[i] ; → Jo bhi us index ki value hogi utne steps liye jaa sakte hai.

j → traversal of steps till steps mentioned

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

if i=1 ∴ j = 2,3,4

i=1 + steps(3) = 4 ↓ max. of j can go

this condition also needs to be satisfied ∵ array ke bahar kaha jaoge.

i = Kaha se Jump lena hai

j = Kaha tak Jump lena hai

if (dp[j] != -1){

Also store -1 in Array if Jump is not feasible.

ans = min(ans, dp[j]+1)

dp[i] = ans (≠ ∞)

∴ if all indexes not feasibles ∴ store -1.

eg. [1|0|0] (0, 1, 2)

Jab tak call nhi hua tab tak already -1 hai

will be -1 since 2 ki kabhi pahuch hi nhi sakte.

return dp[0] → ∵ 0 se n-1 ka kitne min. steps.

Teacher's Signature.....

Date 27 09 24.
Page No.

GRAPHS - Important codes and points practice

refer theory notes from copy (main).

Store a graph → (1.) adjacency list 2. Adj. matrix 3. Edge list
Preferred ← 4. 2D matrix.
∵ optimized

Adjacent list ⇒ static class Edge { // static → to call in psvm without making its object.
↓
making array of ArrayList

```
int src;
" wt;
" dest;
```
} declarations

```
public Edge (int src, int dest, int wt){
    this.src = src;
    this.wt = wt;
    this.dest = dest;
}
```
} Constructor → to call it in a fixed manner.

```
psvm {
int V = 5;    // Vertices defined
ArrayList<Edge>[] graph = new ArrayList[V];   → Array of A.L. declaration
                          ↑ must define size
for (int i=0; i<graph.length; i++){
    graph[i] = new ArrayList<>();   → A.L. added to all index.
}
graph[0].add(new Edge(0, 1, 5));
              src  dest  wt
```

using for 5 vertices. . . .

to get neighbours ⇒ (e.g. 2)

```
for (int i=0; i<graph[2].length; i++)
    Edge e = graph[2].get(i);   // getting from Edge one by one
    sout (e.dest + " ");
}
```
length → since we're getting from an ArrayList.
size()

+ Do not forget to use util to get ArrayList.
⇒ import java.util.*;

Teacher's Signature.....

Date | Page No.

- In Adj matrix we put same node on rows & col to check connections (1) for each. Since elements repeated for no use ∴ not optimized + $O(V^2)$

Benefit → rather checking connections we can put weight there ✓

- Edge list → Edges = {{0,1,4}, {1,2,0}, {...}, {...}}
  ↳ used in Min. Spanning Tree
  e1 = 1st, e2

- Implicit → use rec. approach to fill matrix that contains info of graph.

- Graph Traversals

1. BFS → do horizontally level wise by checking neighbours first
2. DFS (wait)

eg. 0 → 1 → 3 can't access 3 before 2.
0 → 2

☆ NOTE:- Theres a theory of Wrapper class in BFS Graph Traversals.

- BFS ⇒ write boiler plate code for Graph (adj-list → same for all)
  rather doing all ~~declaration~~ creation initialization in psvm do it in ~~args~~ another function
  & create in psvm.

```
public static void bfs(ArrayList<Edge>[] graph){
    Queue<Integer> q = new LinkedList<>();
    boolean[] vis = new boolean[7]; graph.length ☆
    q.add(0);   starter
    while(!q.isEmpty()){
        int curr = q.remove();   // FIFO
        if(!vis[curr]){
            sout(curr + " ");
            vis[curr] = true;
            for(int i=0; i<graph[curr].size(); i++){
                Edge e = graph[curr].get(i);
                q.add(e.dest);
            }
        }
```

In psvm → declare graph → call created graph → call bfs.

Teacher's Signature.....

Date | Page No.

• 2. DFS → priority on the basis of neighbour not on level.

code → public static void dfs(ArrayList<Edge>[] graph, boolean[] vis, int curr){

always fixed ← { sout(curr + " "); vis[curr] = true;

```
for (int i = 0; i < graph[curr].size(); i++){
    Edge e = graph[curr].get(i);
    if (!vis[e.dest]){
        dfs(graph, e.dest, vis);
    }
```

• HasPath?

code → public static boolean hasPath(graph, src, dest, [] vis){

```
if (src == dest){
    return true
}
vis[src] = true;
if (!vis[e.dest] && hasPath(graph, e.dest, dest, [] vis)){
    return true;
}else{
    return false
}
```

Teacher's Signature.....