

Notes Made By

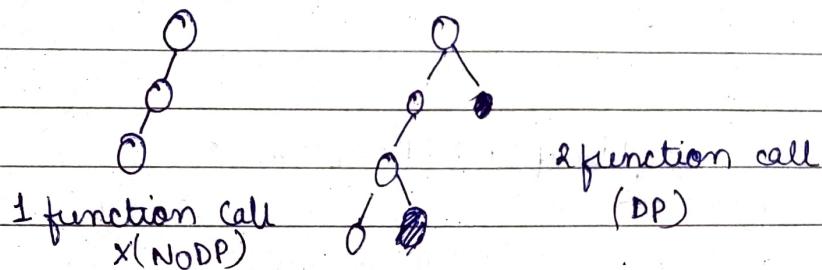
RITI KUMARI

Dynamic Programming (Aditya Verma)

DP = enhanced recursion

How to identify DP problem (2 cases)

- 1) where there is recursion, DP is used (for overlapping problem)
 - a) choice



- b) Optimal - min, max, largest

How to write DP code?

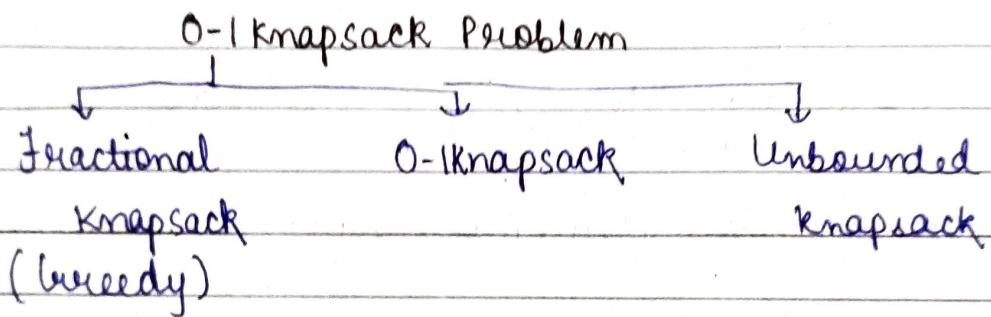
Recursive solution \rightarrow memoization \rightarrow Top down approach

Questions on DP.

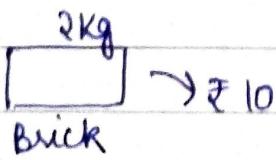
- 1) 0-1 knapsack (6)
- 2) Unbounded knapsack (5)
- 3) Fibonacci (7)
- 4) LCS (15) (longest common subsequence)
- 5) LIS (10) (longest increasing subsequence)
- 6) Kadane's Algorithm (6)
- 7) Matrix chain multiplication (7)
- 8) DP on grid (4)
- 9) DP on grid (14)
- 10) Others (5)

Types of knapsack

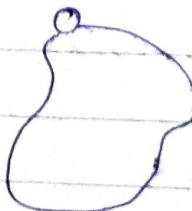
1. Subset sum
2. Equal sum partition
3. Count of subset sum
4. Minimum subset diff
5. Target sum.
- 6.



0-1 Knapsack \rightarrow We are given some weight & some value. Then a max weight w . pick items so that the profit is maximum. And the weight has a given bound w .



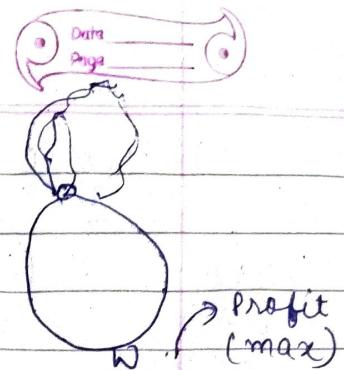
	I_1	I_2	I_3	I_4	\dots
$wt[] =$	1	3	4	5	
$val[] =$	1	4	5	7	



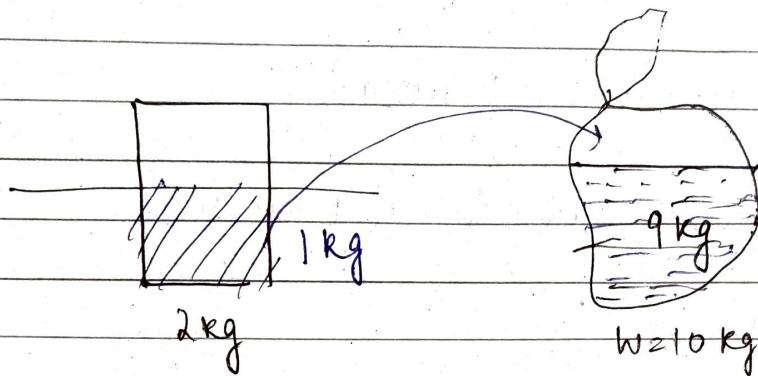
$w=7\text{kg}$.

Max Profit = ?

$P_1 \quad P_2 \quad P_3 \quad P_4$
 $w_1 \quad w_2 \quad w_3 \quad w_4$



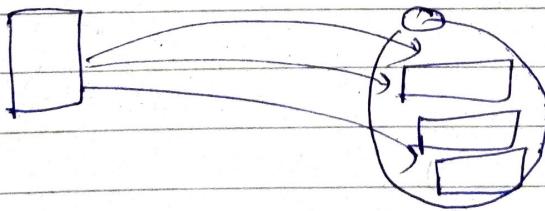
Fractional knapsack



Greedy approach

Unbounded knapsack

unlimited supply of every item



0-1 knapsack

i) How to identify

$wt[] : 1 \quad 3 \quad 4 \quad 5$ $W : 7 \text{ kg}$
 $val[] : 1 \quad 4 \quad 5 \quad 7$

$\% : \text{max profit}$

1) Choice
↓

(either
put in
knapsack)

2) Optimal
(max profit)

(either
don't
put in
knapsack)

DP: Recursive → Memoization → Top down
(DP) (DP)

DP → recursion storage

0-1 knapsack Recursive

Identify

DP → Recursive → DP (topdown)
→ DP (memoization)

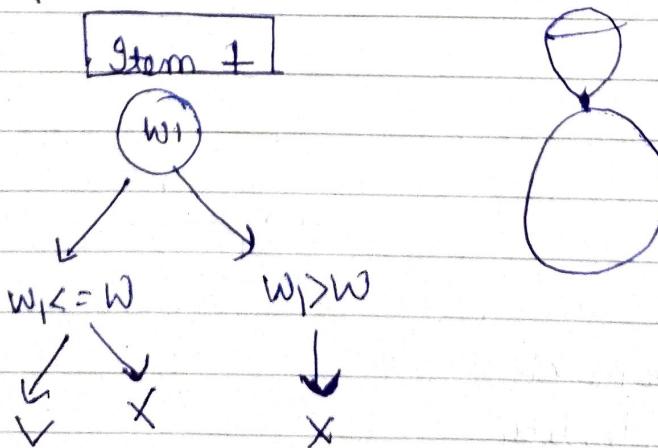
I/P $wt[] = [1|3|4|5]$

$val[] = [1|4|5|7]$

O/p → Max Profit

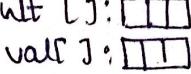
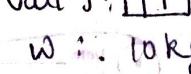
(Capacity of
knapsack) $W = 7 \text{ kg}$

choice
diagram



We have to return the max profit so return type would
be int.

Base condⁿ → think of the smallest valid ip.

IP wt[]: ] → n → 0.
val[]: 

W: 10 kg → 0 kg



max profit

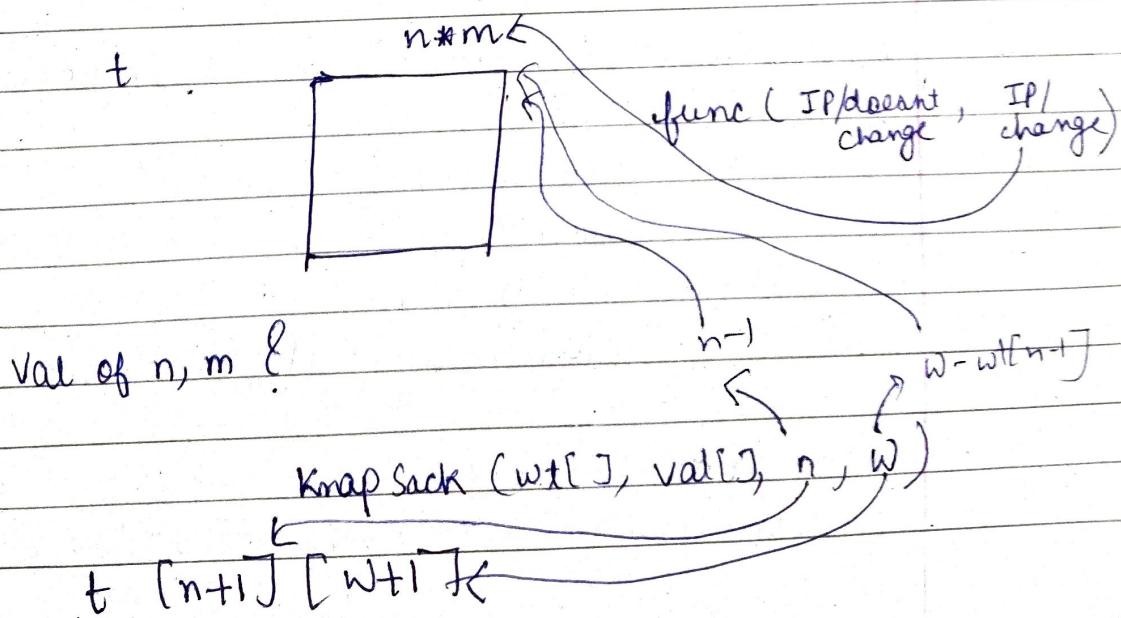
```
int Knapsack( int wt[], int val[], int w, int n ) {  
    // base condition  
    if (n == 0 || w == 0)  
        return 0;
```

// choice diagram

if (wt[n-1] <= w) { → weight vs gain
 return max{val[n-1] + Knapsack(wt, val, w - wt[n-1]),
 Knapsack(wt, val, w, n-1)}
}
else if (wt[n-1] > w) { weight less hai,
 but dont include
 return Knapsack(wt, val, w, n-1);
}

Q1 Knapsack Memoization

Memoization = Recursive + 2 lines



	-1	-1	+	-1	-1
	-1	-1	-1	-1	-1
n+1	-1	-1	-2	-1	-1
	-1	-1	-1	-1	-1
	-1	-1	-1	-1	-1

if (-1) is not present then val exists so, return

initialise this matrix with -1.

```
int t[n+1][w+1]
memset(t, -1, sizeof(t))
```

Change

Now declare the matrix globally.

```
int static t[102][1002]; constraint
n <= 100
w <= 1000
memset(t, -1, sizeof t)
```

```
int knapsack (int wt[], int val[], int w, int n)
{
```

if (n == 0 || w == 0)

return 0;

if (t[n][w] != -1)

return t[n][w];

if (wt[n-1] <= w)

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

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

else if ($wt[n-1] > w$)

return $t[n][w] = \text{Knapsack}(wt, val, w, n-1)$;

}

The complexity of top down & memoization remains same but the problem with memoization is the stack gets full due to repeated funcn' calls.

(0-1 Top Down
Knapsack)

Real DP

Recursive \rightarrow Memoize \rightarrow Top down \rightarrow 6 Problems.

\downarrow

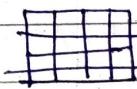
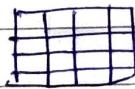
\downarrow

\downarrow

BC + recursive
calls

RC +
table

only
Table

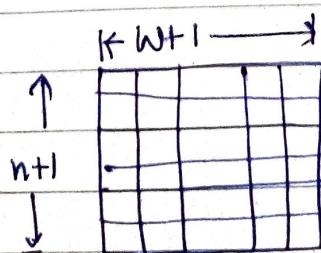


Recursive

Memoization

Initialising
matrix with -1

Top-down (totally omit the
recursive call &
use the table only)



We only make table for
the ip which is
changing

2 steps to make table for top down

Step1: Initialization

Step2: Recursive code changes Iterative code

Step1: Initialize

$$w = 7$$

$$n = 4$$

$$wt[] = [1 3 4 5]$$

$$val[] = [1 4 5 7]$$

w → (j)

		0	1	2	3	4	5	6	7
val	wt								
1	1	0							
4	3	1	0						
5	4	2	1	0					
7	5	3	2	1	0				
		4	3	2	1	0			

t[n][w]

→ it will give
ans

w=3

$$wt[] = [1 3 4 5] \quad wt[] = [1 3]$$

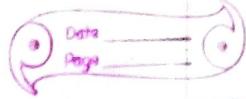
$$val[] = [1 4 5 7] \quad val[] = [1 4]$$

$$w = 3$$

$$wt[] = [1 3 4]$$

$$val[] = [1 4 5]$$

$$w = 6$$



RC + table \longrightarrow table

Base condⁿ \longrightarrow Initialization

Base condⁿ RC
 $\text{if } (n == 0 \text{ || } w == 0)$
 $\quad \downarrow$ between 0;

table

$\text{for (int } i=0; i < n+1; i++) \{$

$\quad \text{for (int } j=0; j < w+1; j++) \{$

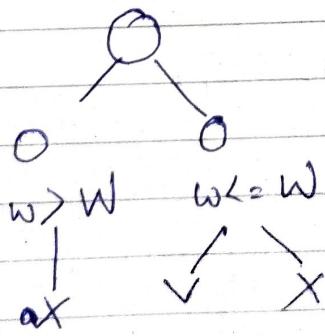
$\quad \quad \text{if } (i == 0 \text{ || } j == 0)$

$\quad \quad \quad t[i][j] = 0;$

}

w\o	0	1	2	...
0	0	0	0	0
1	0			
2	0			
3	0			
4	0			

Choice
diagram



$n, w \rightarrow i, j$

$dp[i][j]$

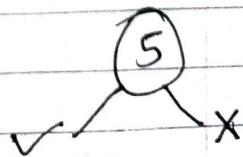
RC

$$wt = 13 + 5$$

$$W = 7$$

$$\sqrt{ad} = 14 \ 5$$

$\max(\text{val}[n-1] + \text{dp}[i-1][j - w[i-1]],$
 $\text{dp}[i-1][j])$



Date _____
Page _____

<p>Recursive</p> <pre> if (wt[n-1] <= w) return max(val[n-1] + knapsack(wt, val, w-wt[n-1], n), knapsack(wt, val, w, n-1)) else if (wt[n-1] > w) return knapsack(wt, val, w, n-1) </pre>	<p>Top down</p> <pre> if (wt[n-1] <= w) t[n][w] = max(val[n-1] + t[w-wt[n-1]] , t[n-1][w]) else t[n][w] = t[n-1][w] </pre>
---	---

<p>pseudo code</p> <pre> int t[n+1][w+1]; for (int i=1; i < n+1; i++) for (int j=1; j < w+1; j++) if (wt[i-1] <= j) t[i][j] = max(val[i-1] + t[i-1][j-wt[i-1]], t[i-1][j]) else t[i][j] = t[i-1][j] return t[n][w]; </pre>	<p>Top - down approach</p> <p style="text-align: center;"> $t[1][1] = \max(0, 1)$ $t[1][2] = \max(1, 1)$ $t[1][3] = \max(2, 1)$ $t[1][4] = \max(3, 1)$ $t[1][5] = \max(4, 1)$ $t[1][6] = \max(5, 1)$ $t[1][7] = \max(6, 1)$ $t[2][1] = \max(1, 1)$ $t[2][2] = \max(2, 2)$ $t[2][3] = \max(3, 2)$ $t[2][4] = \max(4, 2)$ $t[2][5] = \max(5, 2)$ $t[2][6] = \max(6, 2)$ $t[2][7] = \max(7, 2)$ $t[3][1] = \max(1, 1)$ $t[3][2] = \max(2, 2)$ $t[3][3] = \max(3, 3)$ $t[3][4] = \max(4, 3)$ $t[3][5] = \max(5, 3)$ $t[3][6] = \max(6, 3)$ $t[3][7] = \max(7, 3)$ $t[4][1] = \max(1, 1)$ $t[4][2] = \max(2, 2)$ $t[4][3] = \max(3, 3)$ $t[4][4] = \max(4, 4)$ $t[4][5] = \max(5, 4)$ $t[4][6] = \max(6, 4)$ $t[4][7] = \max(7, 4)$ $t[5][1] = \max(1, 1)$ $t[5][2] = \max(2, 2)$ $t[5][3] = \max(3, 3)$ $t[5][4] = \max(4, 4)$ $t[5][5] = \max(5, 5)$ $t[5][6] = \max(6, 5)$ $t[5][7] = \max(7, 5)$ $t[6][1] = \max(1, 1)$ $t[6][2] = \max(2, 2)$ $t[6][3] = \max(3, 3)$ $t[6][4] = \max(4, 4)$ $t[6][5] = \max(5, 5)$ $t[6][6] = \max(6, 6)$ $t[6][7] = \max(7, 6)$ $t[7][1] = \max(1, 1)$ $t[7][2] = \max(2, 2)$ $t[7][3] = \max(3, 3)$ $t[7][4] = \max(4, 4)$ $t[7][5] = \max(5, 5)$ $t[7][6] = \max(6, 6)$ $t[7][7] = \max(7, 7)$ </p>
---	--

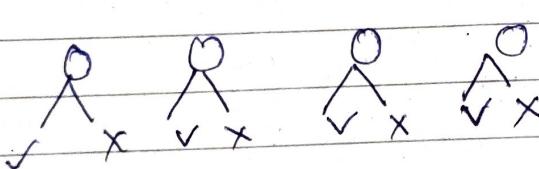
Identification of Knapsack Problem

- 1) Subset sum problem
- 2) Equal sum partition
- 3) Count of subset sum.
- 4) Minimum subset sum diff
- 5) Target sum
- 6) No of subset with a given diff.

I_p : Item array :

--	--	--	--

w : Capacity



1. Subset Sum problem

arr [] : 2 3 7 8 10

sum =

- a) Problem statement
- b) Similarity with knapsack
- c) Code variation

D) Problem statement : find if there is a subset present in an array with given sum.

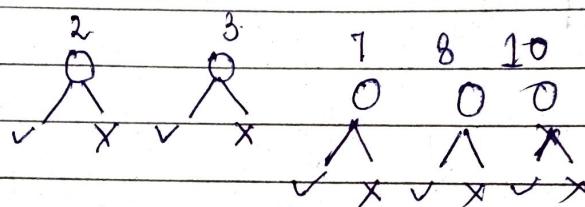
arr []: 2 3 7 8 10

Sum : 11

2) Similarity

item array [] : → 2 3 7 8 10

weight : capacity → 11



3) Code variation

$t[n+1][w+1]$

$t[5+1][11+1]$

sum

$t[6][12]$

Initialization:

0	1	2	3	4	5	6	7	8	9	10	11
T	F	F	F	F	F	F	F	F	F	F	F
T											
T											
T											
T											
T											
T											
T											
T											
T											
T											

$\text{if } \rightarrow \text{True/False}$

arr []:
sum : 0
arr []: 1
sum = 0

arr []: 2
sum = 0

arr []:
sum = 2

when array is empty sum can't be anything

arr[]: no elements

sum: 1 → not possible

$t[n+1][sum+1]$

Initialisation: $\text{for } (\text{int } i = 1 \dots n+1)$
 $\quad \text{for } (\text{int } j = 1 \dots m+1)$

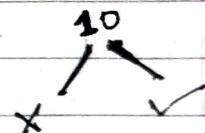
$\text{if } (i == 0)$

$t[i][j] = \text{false}$.

2378!0

$\text{if } (j == 0)$

$t[i][j] = \text{True}$



knapSack → arr

$\text{if } (\text{cost}[i-1] \leq j)$

$t[i][j] = \max(\text{val}[i-1] + t[i-1]$

↓
there is
no max
in true
or false.

else

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

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

$= t[i][0 - \text{arr}[0]]$
 $* [0][0]$
 $* [i-2][1] * [0][0]$
 $= t[i-2][1] * [0][0]$

$= t[i][j]$

2, 3, 8

$\left\{ \begin{array}{l} \{3, 8\} \rightarrow \text{true } \checkmark \\ \{3\} \rightarrow \text{false } \times \end{array} \right.$

Subset sum

$\text{if } (\text{arr}[i-1] \leq j)$

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

||

$t[i-1][j]$

else

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

return $t[n][sum]$;

2. Equal Sum Partition Problem

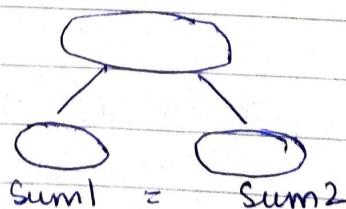
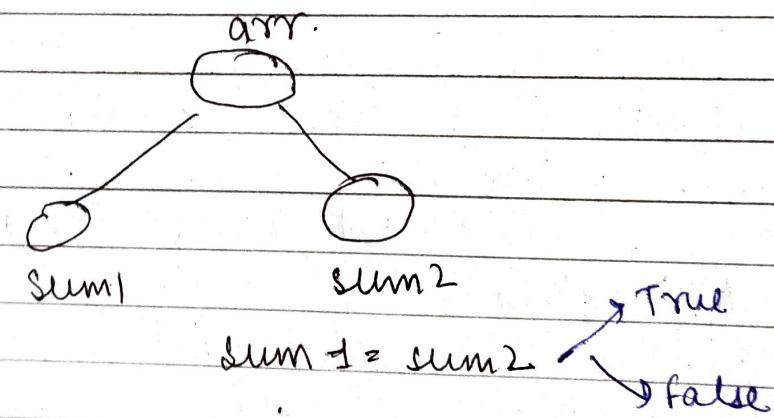
- 1) Problem statement
- 2) Subset sum similarity
- 3) Odd / Even significance
- 4) Code variation

1) Problem statement

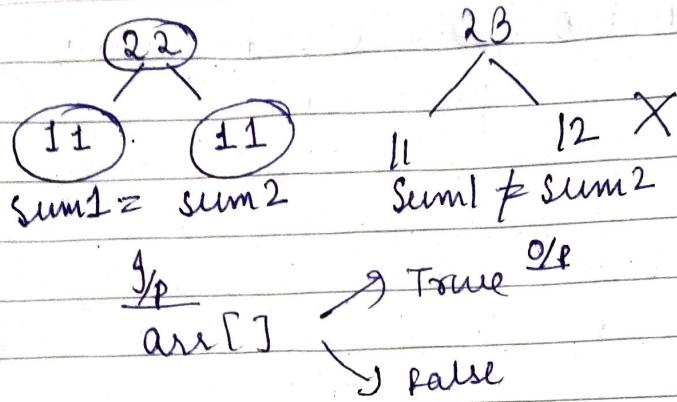
$$\text{arr}[] = \{1, 5, 11, 5\}$$

∴ O/P : T/F

Is it possible to divide the array such that both the subset gives an equal sum.



$\text{sum1} = \text{sum2} \rightarrow$ It is equal when the no is even. Sum of all the array elements should be even to change it into equal parts.



```
for (int i = 0; i < size; i++) {
```

{1, 5, 11, 5}

Subset even Subset odd

even sum=22 odd sum=11

Subset = 11 Subset 11

```
    sum = sum + arr[i];
```

}

if ($\text{sum} \% 2 \neq 0$) (sum is odd)
 return false

else if ($\text{sum} \% 2 == 0$)

*→ we need to find one subset with sum 11 the
 next subset would automatically be 11.

return subsetsum(arr, sum/2);

3. Code:

```
ip → arr[], n.
```

```
int sum = 0;
```

```
for (int i = 0; i < n; i++)
```

```
    sum += arr[i];
```

```
if ( $\text{sum} \% 2 \neq 0$ )
```

```
    return false
```

```
else
```

```
    return subsetsum(arr, sum/2);
```

3. Count of Subsets sum with a given sum.

S/p:

$arr[] = 2 \ 3 \ 5 \ 6 \ 8 \ 10$

sum = 10.

Flow

- 1) Problem Statement
- 2) Similarity to subset sum
- 3) Code variation

✓ ↘

Initialisation Code

- 4) Return Type.

- 1) Problem Statement

$arr[] = 2 \ 3 \ 5 \ 6 \ 8 \ 10$

Sum : 10

O/p = 3.

$\{2, 8\} \rightarrow$ Yes/True (in subset sum)

$$\left. \begin{array}{l} \{2, 8\} = 10 \\ \{5, 2, 3\} = 10 \\ \{10\} = 10 \end{array} \right\} \text{count} = 3$$

- 2) Similarity

2, 8

Yes No

return count

3. Code variation

Subset sum

Count
int

due to ↙ bool
T/F

False → 0 (no of subset
0)

True → null
subset (no of subset
1)

0	0	0	0	0/0	-
1					
1					

if ($arr[i-1] \leq j$) +
 $dp[i][j] = dp[i-1][j] + t[i-1]$
 [j-arr[i-1]]

else

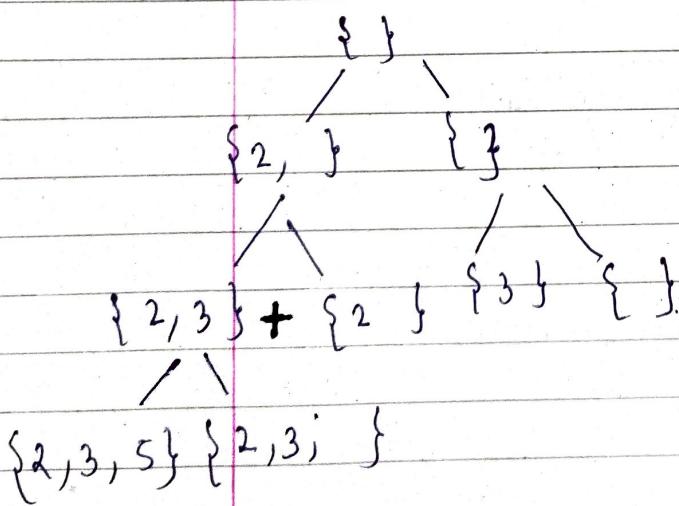
$dp[i][j] = dp[i-1][j]$

we will add all the subset
 so arr would be changed
 to + - True & false case
 we can use ~~arr~~ arr.

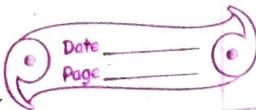
if ($arr[i-1] \leq j$)
 $dp[i][j] = dp[i-1][j-1] + dp[i-1]$
 [j-arr[i-1]]

else

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



if ($\text{arr}[i-1] \leq j$)
 $\text{dp}[i][j] = \text{dp}[i-1][j - \text{arr}[i-1]] + \text{dp}[i-1][j]$
 else
 $\text{dp}[i][j] = \text{dp}[i-1][j]$



$\text{arr}[] = [3 5 6 8 10]$

Sum = 10

O/p = 3

1, 3, 2, 5 5

O/p $\rightarrow \{3, 2\}$
 $\{5\}$

$\text{arr}[0] \leq 2^1$

~~10000~~

$$\text{dp}[N+1][\text{sum}+1] = \text{dp}[6+1][10+1]$$

$$= \text{dp}[7][11]$$

$i \in [1, 2, 3, 4]$

Sum \rightarrow

$+1 \quad \{ \quad \}$
 $-1 \quad \{ \quad \}$
 $+3 \quad \{ 1 \quad \}$
 $-3 \quad \{ \quad \}$
 $+3 \quad \{ 1 \quad \}$
 $-3 \quad \{ 3 \quad \}$
 $\{ 1, 3 \} \quad \{ 1 \} \quad \{ 3 \} \quad \{ \}$

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

Minimum $\sum_{i=1}^n |S_i - S_j|$

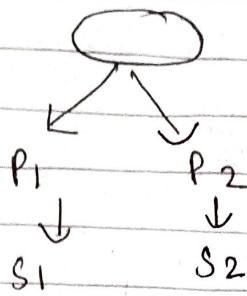
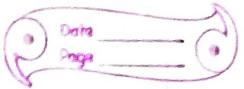
- 1) Problem statement
- 2) Similarity
- 3) Solve using its previous concept

+

- 1) Problem statement

$\text{arr}[] : [16 11 5]$

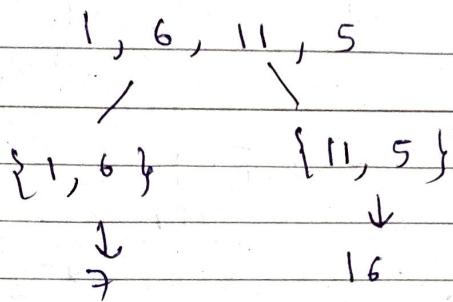
O/P : 1



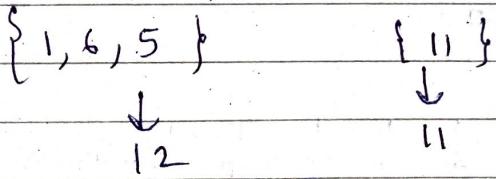
Equal sum: $S_1 - S_2 = 0$

Num^m subset: $S_1 - S_2 = \min$

$\text{abs}(S_1 - S_2) = \min$ (should be min)



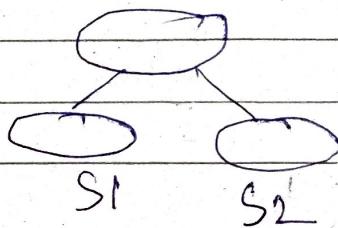
$16 - 7 = 9 \rightarrow$ minimize of
find could
it be done
in a better
way.



+ $12 - 11 = 1 \rightarrow$ Can't be minimized further
→ O/P

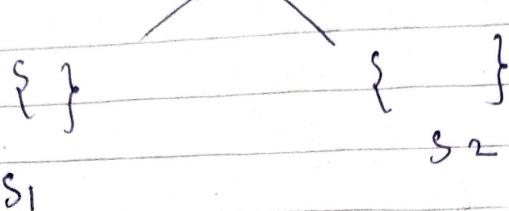
2) Similarity

It is similar to equal sum partition.



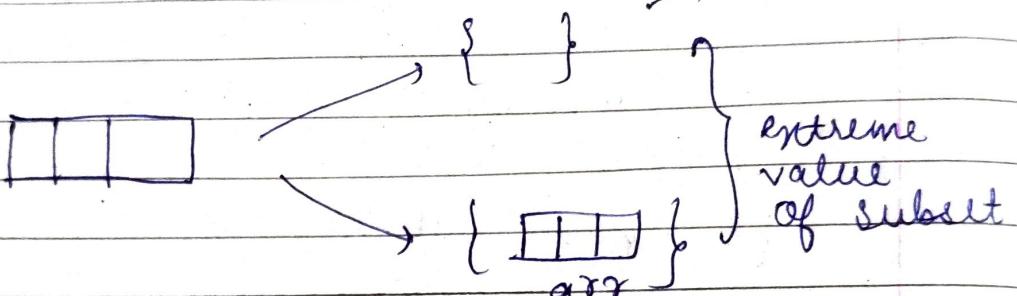
We need to find s_1 & s_2 .

$\text{arr}[] [1 | 6 | 11 | 5]$



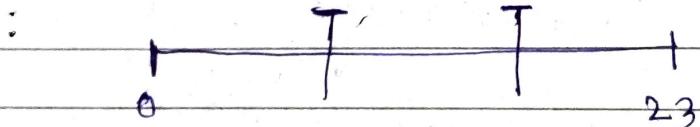
We can find the range of s_1 & s_2 .

$$s_1 = 0 \quad (0+0+0+0)$$

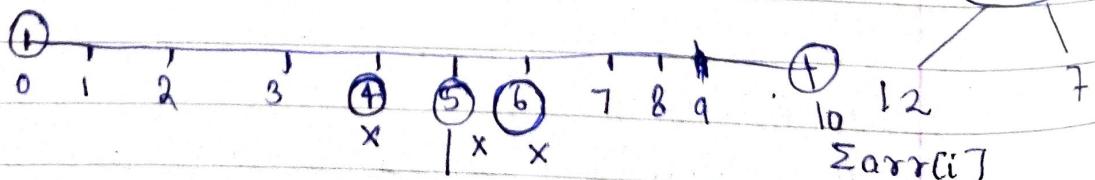
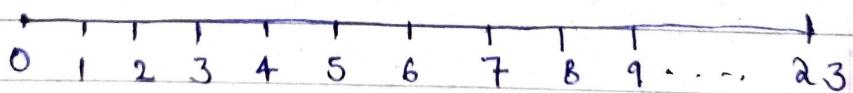


$$s_1 \quad s_2$$

$$s_2 = 23 \quad (1+6+11+5)$$



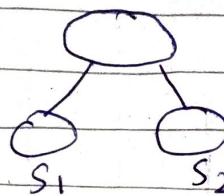
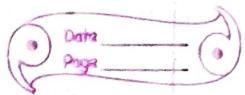
$\text{arr}[] : \{1, 2, 7\}$



s_1 / s_2
can't be

5

$$s_1 / s_2 = \{0, 1, 2, 3, 7, 8, 9\}$$

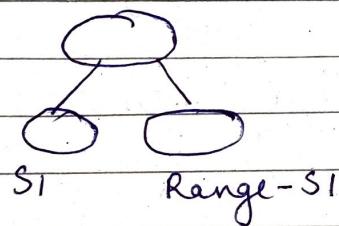


$$S1 + S2 = \sum arr[i]$$

$$S1 / S2 = \{ 0, 1, 2, 3, | 7, 8, 9, 10 \}$$

S_1 Range - S_1
 (S_1) (S_2)

$$S1 - S2 = \text{minimize}$$

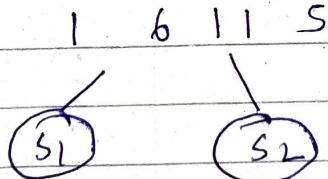


$$\text{abs}(.S_2 - S_1) \text{ or } \text{abs}(S_1 - S_2)$$

$$\text{abs}(S_2 - S) = \text{minimize} \quad \rightarrow \text{minimize}$$

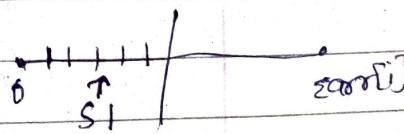
$$\text{abs}(\text{Range}-S_1 - S_1) = \text{minimize. abs}(\text{Range} - 2S_1)$$

Sum up

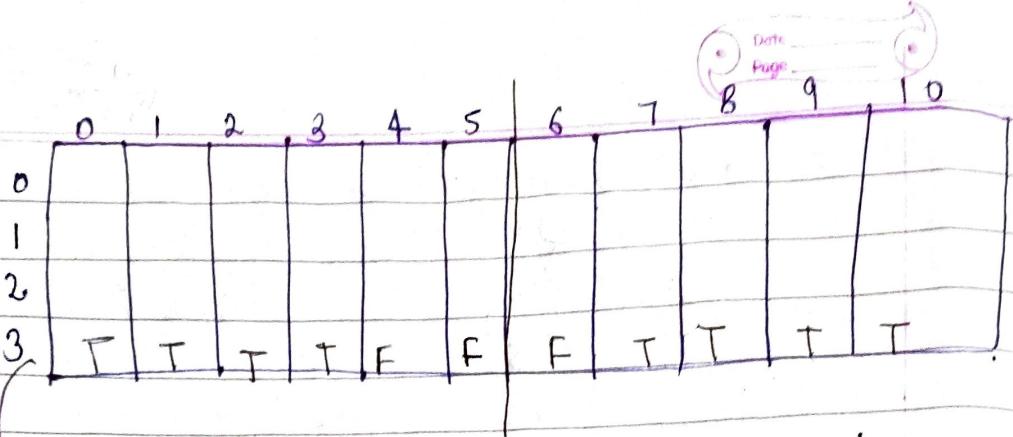


$$\left. \begin{array}{l} S_1 - S_2 \\ S_2 - S_1 \end{array} \right\} \text{minimize}$$

$$\begin{array}{l} \downarrow \\ ((\text{Range} - S_1) - S_1) \\ (\text{Range} - 2S_1) \end{array}$$



Range/2



→ we will push the last row in vector till half way.

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

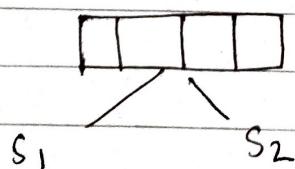
int min = INT_MAX;

for (int i = 0; i < v.size(); i++) {

} mn = min (mn, Range - 2v[i])

return mn;

Concept

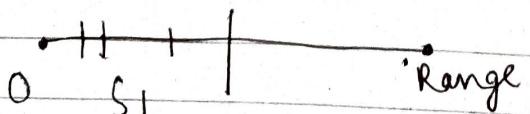


$$(S_2 - S_1) = \min$$

↓
smaller

$$(S_1) \quad (\text{Range} - S_1)$$

$$\text{Range} - 2S_1 \rightarrow \text{Min}^m$$



Code

subset sum(int arr[], int range)

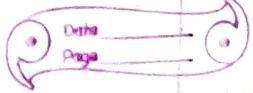
{

last row filled in vec till

0	1	0	0
---	---	---	---

halfway

}



```
int mindiff (int arr[], int n) {
```

```
    int sum = 0;
```

```
    for (int i = 0; i < n; i++) {
```

```
        sum += arr[i];
```

```
}
```

```
    bool dp[n+1][sum+1];
```

```
    for (int i = 0; i < n+1; i++) {
```

```
        for (int j = 0; j < sum+1; j++) {
```

```
            if (i == 0) dp[i][j] = false;
```

~~```
 if (j == 0) dp[i][j] = true;
```~~

```
}
```

```
}
```

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

```
 for (int j = 1; j < sum+1; j++) {
```

```
 if (arr[i-1] <= j)
```

```
 dp[i][j] = dp[i-1][j - arr[i-1]] || dp[i-1][j];
```

```
 else
```

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

```
}
```

```
}
```

```
 int diff = INT_MAX;
```

```
 for (int j = sum/2; j >= 0; j--) {
```

```
 if (dp[n][j] == 2 * true) {
```

```
 diff = min(sum - 2 * j, diff)
```

~~```
        }
```~~

```
}
```

```
    return diff;
```

```
}
```

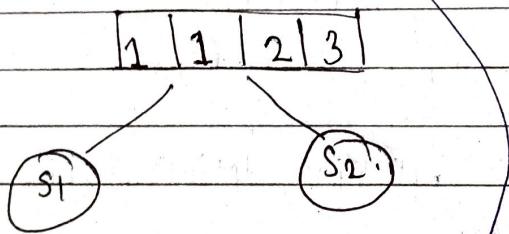
Count the number of subset with
a given diff

- 1) Problem statement
- 2) will try to reduce the actual statement
- 3) solve it using already solved problem.

D) Problem statement

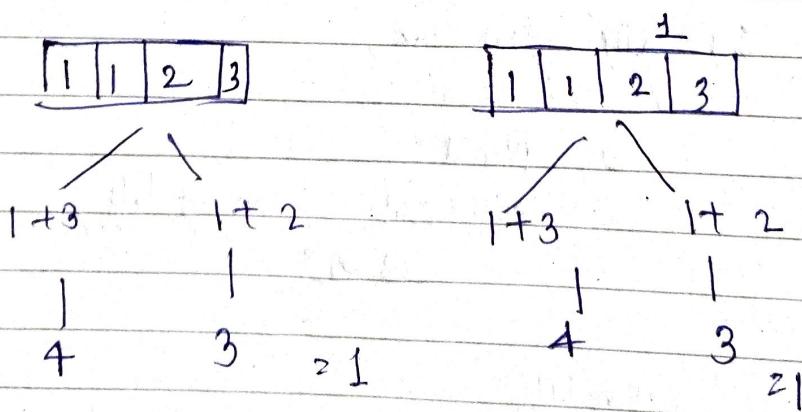
arr[] : [1 1 2 3]

Diff : 1



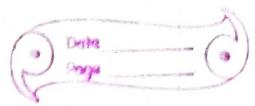
$$S1 - S2 = \text{diff}$$

return the count of
subset with diff^n



$$\begin{array}{|c|c|c|c|} \hline 1 & 1 & 2 & 3 \\ \hline \end{array}$$

$1+1+2 - 3 = 1$



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

S_1 S_2

(diff) $\rightarrow S = \text{sum of arr}$

$$\text{sum}(S_1) - \text{sum}(S_2) = \text{diff}$$

$$\text{sum}(S_1) + \text{sum}(S_2) = S$$

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

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

$$= \frac{1+7}{2} = \frac{8}{2} = 4$$

$$S_1 = 4$$

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

$$= 4 - 1$$

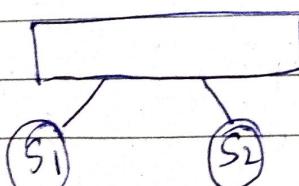
$$\text{Count} = ?$$

$$= 3.$$

① Find count when sum of $S_1 = 4$

no of count of \rightarrow count of
Subset with subset
given diff sum.

Sum up.



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

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

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

$$S_1 + S_2 = \text{sum(arr)}$$

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

return countofsubsetsum(arr, sum);

int countofsubsetwithdiff (int arr[], int n) {
 int sum = $\frac{diff + sum(arr)}{2}$; int diff
 arrsum = arr[0];
 for (int i=1; i<n; i++) {
 arrsum += arr[i];
 }
 if (arrsum < sum) return 0;

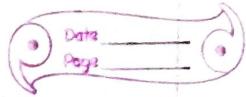
int arrsum = 0;
 for (int i=0; i<n; i++) {
 arrsum += arr[i];
 }

int sum = 0;

Sum = $\frac{diff + arrsum}{2}$.

return countofsubsetsum(arr, sum, n);
 }

int countofsubsetsum (int arr[], int sum, int n) {
 int dp[n+1][sum+1];
 for (int i=0; i<n+1; i++) {
 for (int j=0; j<sum+1; j++) {
 if (i == 0) dp[i][j] = 0;
 if (j == 0) dp[i][j] = 1;
 }
 }
}



```

for (int i = 0; i < n+1; i++) {
    for (int j = 1; j < sum+1; j++) {
        if (arr[i-1] <= j) {
            dp[i][j] = dp[i-1][j - arr[i-1]] + dp[i-1][j];
        } else {
            dp[i][j] = dp[i-1][j];
        }
    }
    return dp[n][sum];
}

```

Target Sum.

arr :

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

sum : 1

+/- :

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

target = 1

%p : 3

Target value = 2

$$+1 -1 -2 +3 = 1$$

$$-1 +1 -2 +3 = 1 \quad \{0, 2\}$$

$$+1 +1 +2 -3 = 1$$

(+, 2) (-, 2)

arr [] \rightarrow

| | | | |
|---|---|---|---|
| + | - | + | + |
|---|---|---|---|

count = 2
 $\{0, 0, 2\}$



$\{+0, +0, 2\} \quad \{+0, -0, 2\} \quad \{0, 0, 2\}$

S1 S2

$\{-0, -0, 2\}$

$S1 - S2 = \text{diff}$

= 4 (count)

So answer is increased
by a power of 2
no of zeros
no of zeros

$$\begin{array}{cccc}
 + & - & - & + \\
 1 & 1 & 2 & 3 \\
 / & & \backslash & \\
 +1+3 & & -1-2 &
 \end{array}$$

$$(1+3) - (1+2)$$

$S_1 - S_2 \rightarrow$ count of
subset with
given diff.

Count Subset Difference

```
int findTargetSumWays(vector<int>& nums, int s) {
```

```
    int cnt = 0, sum = 0;
```

```
    int n = nums.size();
```

```
    for (int i = 0; i < nums.size(); i++) {
```

```
        sum = sum + nums[i];
```

```
        if (nums[i] == 0)
```

```
            cnt = cnt + 1;
```

Cnt the
no. of zeroes
in subset

```
s = abs(s);
```

```
if (s > sum) || (s + sum) % 2 != 0)
```

```
return 0;
```

```
int s = (s + sum) / 2;
```

```
int dp[n+1][s+1];
```

```
for (int i = 0; i < n+1; i++)
```

```
    for (int j = 0; j < s+1; j++)
```

```
        if (i == 0) dp[i][j] = 0;
```

```
        if (j == 0) dp[i][j] = 1;
```

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

```
    for (int j=1; j<s+1; j++) {
```

if ($\text{num}[i-1] \leq 0$)

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

else if ($\text{num}[i-1] > j$)

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

else

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

};

return $\text{dp}[n][s];$

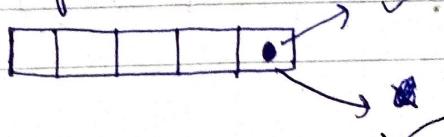
}.

13. Unbounded Knapsack

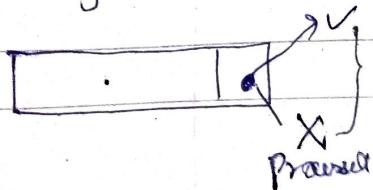
Related Problems

- 1) Road cutting
 - 2) Coin change I (Max no of ways)
 - 3) Coin change II (Min no of ways)
 - 4) Max m: Ribbon cut
- (Variations of unbounded knapsack)

Unbounded
(multiple occurrence
of same item)

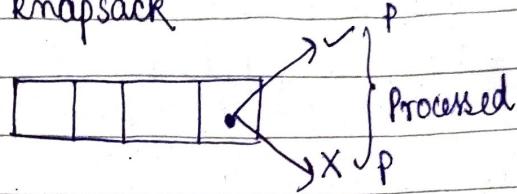


Knapsack
(only one occurrence)

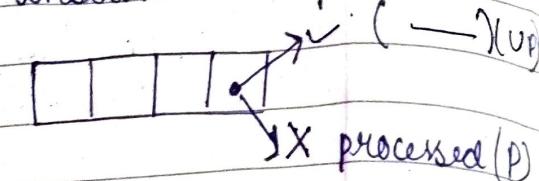


Prasad

knapsack



Unbounded knapsack



Multiple occurrences

(can exist many times)

Comparison betw.

Knapsack

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

Unbounded

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

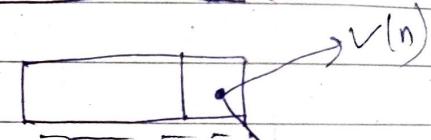
$t[n+1][w+1]$

if ($wt[i-1] \leq j$) {

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

else {

$t[i][j] = t[i-1][j];$
 $}.$



if ($wt[i-1] \leq j$)

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

else

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

14. Rod cutting Problem

length [] :

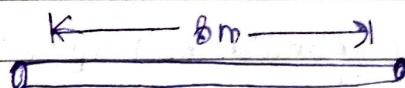
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|

price [] :

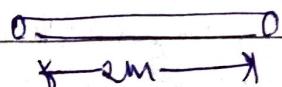
| | | | | | | | |
|---|---|---|---|----|----|----|----|
| 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |
|---|---|---|---|----|----|----|----|

N : 8

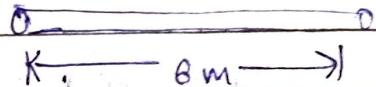
1) Problem statement



length of rod is given. we need to cut it in such a way the profit is max^m.



Rs 5



Rs 17

$$\text{Total} = 5 + 17$$

$$= \text{Rs } 22$$

knapsack

| | |
|-----|--|
| wt | |
| val | |
| W | |

length = 1 to N

Price =

| | | | |
|--|--|--|--|
| | | | |
|--|--|--|--|

N = 8

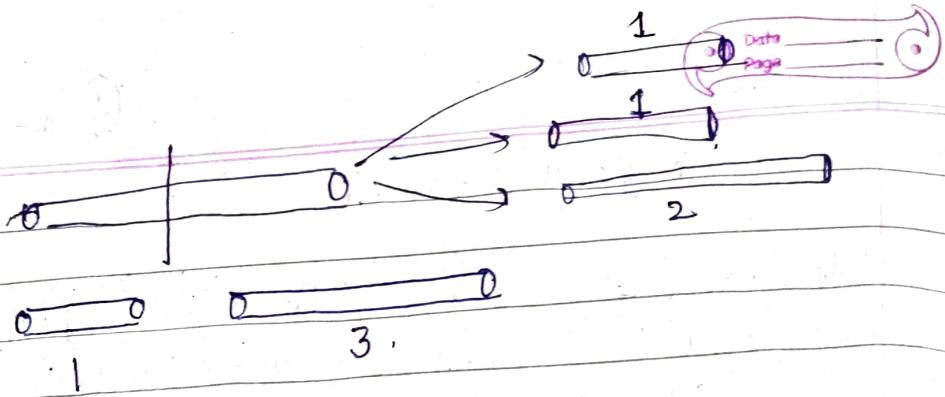
if length array is not give
make it & fill it will
1 to N.

knapsack
~~bounded~~
~~0-1~~

val []
wt []
W

unbounded knapsack
~~0-1~~
~~multiple~~

price []
length []
N



Code variation

$t[N+1][N+1]$

if ($\text{length}[i-1] \leq j$)

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

else

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

Sometimes our size changes therefore we need to find the size of array.

$dp[\text{size}+1][N+1].$

| | | length → | | | | | |
|-----------|---|----------|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 |
| ↓
size | 0 | | | | | | |
| | 0 | | | | | | |

Coin change Problem

Max^m no of
ways

Min^m no
of coins

Max^m no of ways.

Problem

statement

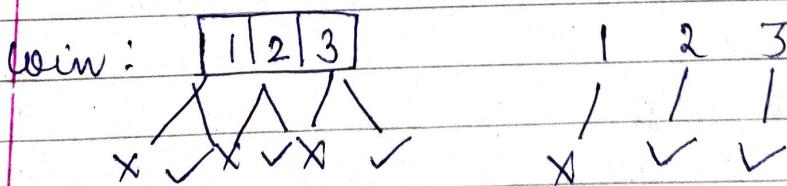
coin [] : [1 | 2 | 3] → (unlimited)
sum: 5.

Use the coin array unlimited time & get the sum 5. Now find the max^m no of ways in which we can get 5.

| | OP |
|---------|-----------|
| 5 ways. | { 2+3 |
| | 1+2+2 |
| | 1+1+3 |
| | 1+1+1+1+1 |
| | 1+1+1+2 |

Q why knapsack

→ Every coin has a choice to include or not



wt []

item

val [] x (when one array is given)

Matching

$wt[] \rightarrow coin[]$
 $w \rightarrow sum$

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

sum 5 : $(\textcircled{1}) + (\textcircled{1}) + 3$

one item used
many times

If taking one item many times does the sum allows then it is unbounded knapsack.

Subset sum

| | | | |
|--------|-------------|---|---|
| 1 | 2 | 3 | 5 |
| Sum: 8 | → T
→ F. | | |

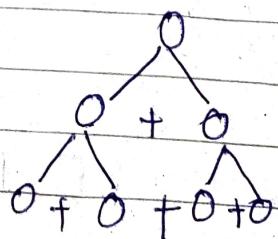
Subset sum

```

if curr[i-1] <= j {
    t[i][j] = t[i-1][j] || t[i-1][j - arr[i-1]]
} else {
    t[i][j] = t[i-1][j];
}
for count
we remove
|| by +.

```

count / no of ways



We need to find maxim. no of ways.

Matching \rightarrow Knapsack $\xrightarrow{0-1}$
 $\xrightarrow{\text{Unbounded}}$

wt \rightarrow coin

W \rightarrow sum.

if ($\text{coin}[i-1] \leq j$)

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

else

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

sum \rightarrow

| | 0 | 1 | 2 | 3 | ... |
|-------------------|---|---|---|---|-----|
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | | | | |
| 2 | 1 | | | | |
| size \downarrow | 3 | 1 | | | |

Coin change-II (Min^m no of coins)

$$\text{coin}[] = [1 \ 2 \ 3]$$

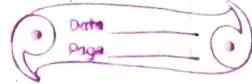
$$\text{sum} = 5$$

$$\begin{aligned}
 2+3 &\rightarrow 5 & \rightarrow 2 \text{ coins} \\
 1+2+2 &\rightarrow 5 & \rightarrow 3 \text{ coins} \\
 1+1+1+2 &\rightarrow 5 & \rightarrow 4 \text{ coins} \\
 1+1+3 &\rightarrow 5 & \rightarrow 3 \text{ coins} \\
 1+1+1+1+1 &\rightarrow 5 & \rightarrow 5 \text{ coins}
 \end{aligned}
 \left. \begin{array}{l} \text{find min}^m \\ \text{no. of coins} \\ \text{to make} \\ \text{sum as } 5 \end{array} \right\}$$

coin[] = [1 2 3]

Sum = 5

O/p : 2 (2+3 = 5)



Initialisation: $t[n+1][w+1]$

\downarrow

$t[n+1][sum+1]$

wt \rightarrow coin[] = n

val \rightarrow x

w \rightarrow sum.

$n^2 3$
sum = 5

size
(n)

sum = 3
coin[] = 1

Initialisation



+ Twist

coin[]: empty

sum : 1

\rightarrow INT_MAX (oo coins)

coin[]: empty

sum : 0

\rightarrow INTMAX - 1



coin[]: 1

sum: 0

} 0 coins

coin[]: 1 2

sum: 0

} 0 coins

for guess
 $\text{sum} = 5$
 $\text{arr} = [3, 5, 2]$

when
 $\text{arr}[3] = 4$
 $\text{sum} = 4$

$$\frac{3}{3} = 1$$

$$\frac{4}{3} = \text{INT_MAX}$$

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

$$\frac{4}{3} = \text{INT_MAX}$$

$$\frac{j}{\text{arr}[0]} = \frac{3}{3} = 1$$

for second row

```
for(int i=1; j < sum+1; j++) {
    if (j % arr[0] == 0)
        t[i][j] = j / arr[0]
    else
```

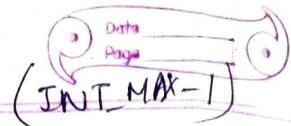
$t[i][j] = \text{INT_MAX-1}$

$$\frac{j}{\text{arr}[0]} = \frac{3}{3} = 1$$

$$\frac{4}{3} = \text{INT_MAX}$$

Code variation

1st row \rightarrow INT-MAX 1st col \rightarrow ~~INT-MAX~~ 0

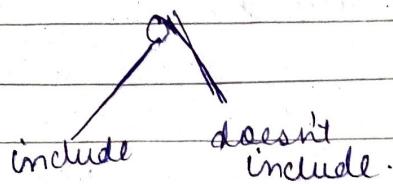


if (~~coins[i-1]~~ $\leq j$)

$t[i][j] = \min(t[i-1][j], t[i][j - \text{coins}[i-1]] + 1)$

else

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



```
int minCoins( int coins[], int M, int V ) {
```

 M = arrsize V = sum.

```
    int dp [M+1][V+1];
```

```
    for (int i=0; i < M+1; i++) {
```

```
        for (int j=0; j < V+1; j++) {
```

 if (j == 0) $dp[i][j] = 0;$

 if (i == 0) $dp[i][j] = INT_MAX-1;$

 }

```
    for (int i=1, j=1; j < V+1; j++) {
```

 if (j * coins[0] == 0) $dp[i][j] = j / coins[0];$

 else

$dp[i][j] = INT_MAX-1;$

```
    for (int i=2; i < M+1; i++) {
```

```
        for (int j=1; j < V+1; j++) {
```

 if (coins[i-1] $\leq j$)

$dp[i][j] = \min(dp[i-1][j], 1 + dp[i-1][j - \text{coins}[i-1]]);$

 else

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

 if ($dp[M][V] == INT_MAX-1$) return -1;

 return dp[M][V];

}



Longest Common Subsequence.

- 1) Longest common substring
- 2) Print LCS
- 3) Shortest common supersequence
- 4) Print SCS
- 5) Min^m no of insertion and deletion $a \rightarrow b$
- 6) Largest repeating subsequence
- 7) length of largest subsequence of a which is a substring is b .
- 8) Subsequence pattern matching
- 9) Count how many times a appear subsequence in b
- 10) largest Palindromic subsequence
- 11) largest palindromic substring
- 12) Count of palindromic substring
- 13) minimum no of deletion in a string to make it a palindrome
- 14) Minimum no of insertion in a string to make it a palindrome

Longest Common Subsequence (Recursive)

Problem Statement

X : @⑥ c⑦ d g⑧ h
Y : @⑨ b e⑩ d f⑪ h u.

String X = abcdg

String Y = abedfhu

\Rightarrow abdh
 \downarrow 4 (length of string)

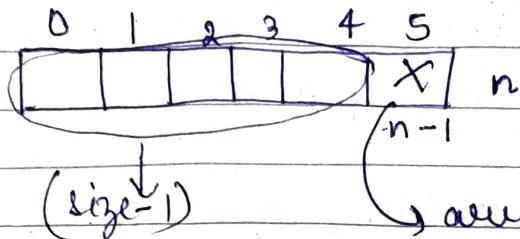
diffn. \rightarrow longest common subsequence - abdh
contr. \rightarrow longest common substring - ab

Recursive approach:

Base condⁿ + Choice diagram + i/p small

↓

(x, y)



fun(x, y)

{

fun(x, y)

and smaller
x

Base condⁿ: Think of the smallest valid input.

X: → n
Y: → m.

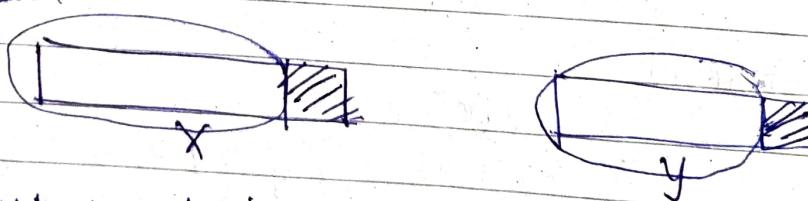
n = 0 m = 0 LCS = 0
(empty string)

if (n = 0 || m = 0)
 between 0

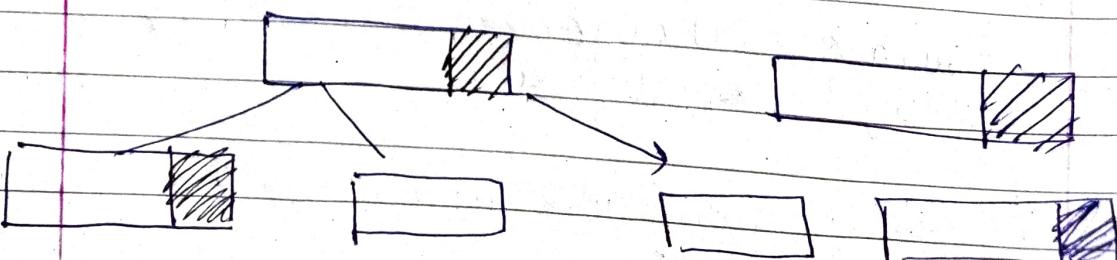
Choice diagram:

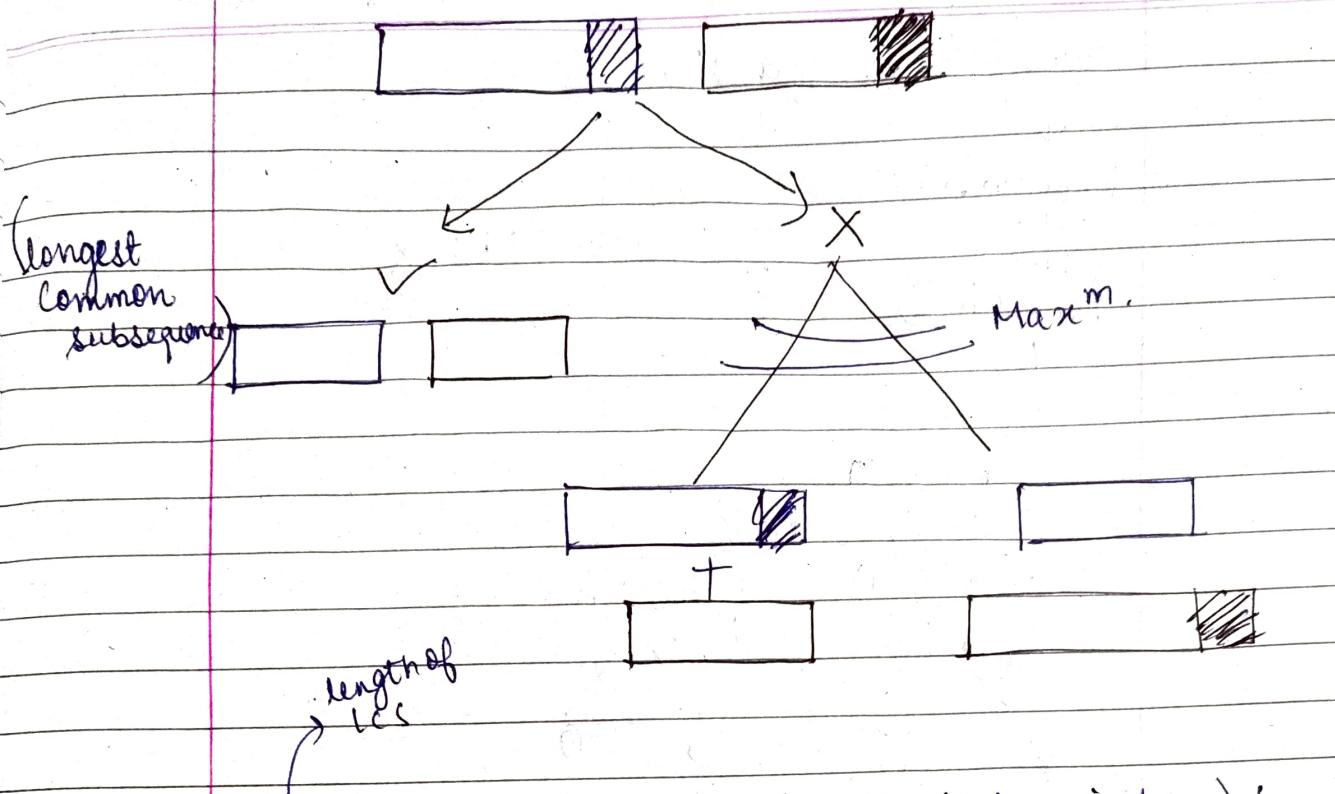
X : abc dgh
Y : abedf hei

① when last char matches



② when last char don't match





```
int lcs (string X, string Y, int n, int m) {
```

```
    if (n==0 || m==0) return 0;
```

```
    if (X[n-1] == Y[m-1])
```

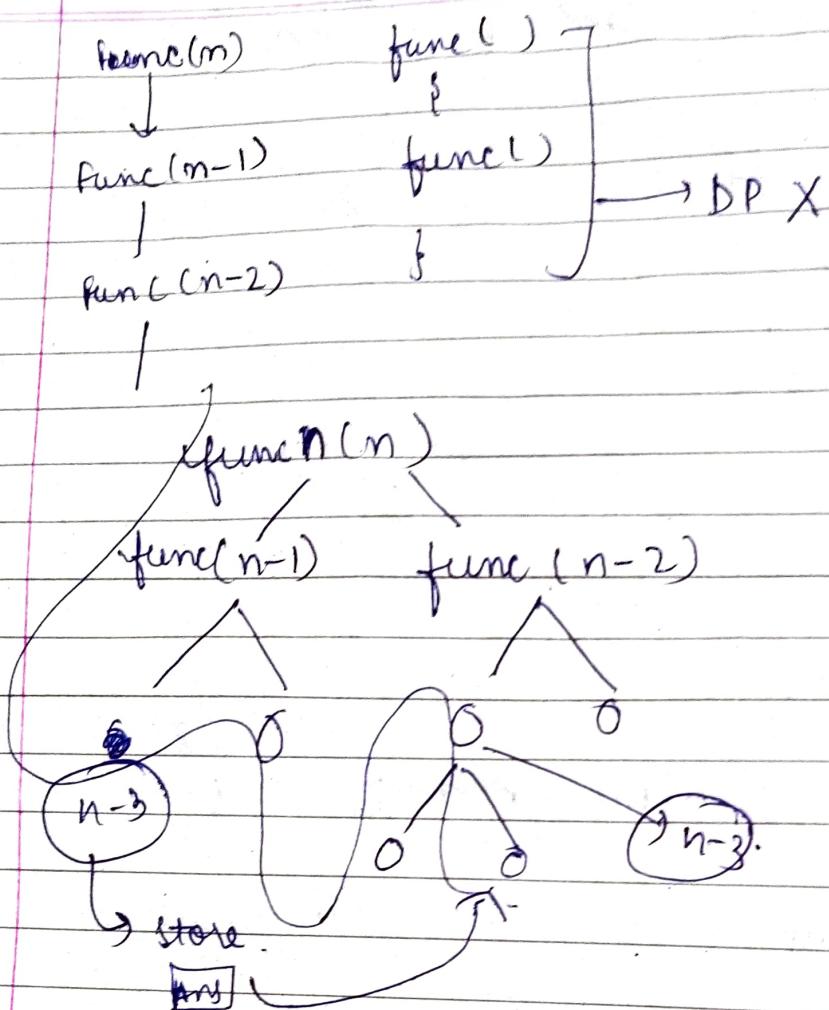
```
        return 1 + lcs(X, Y, n-1, m-1);
```

```
    else
```

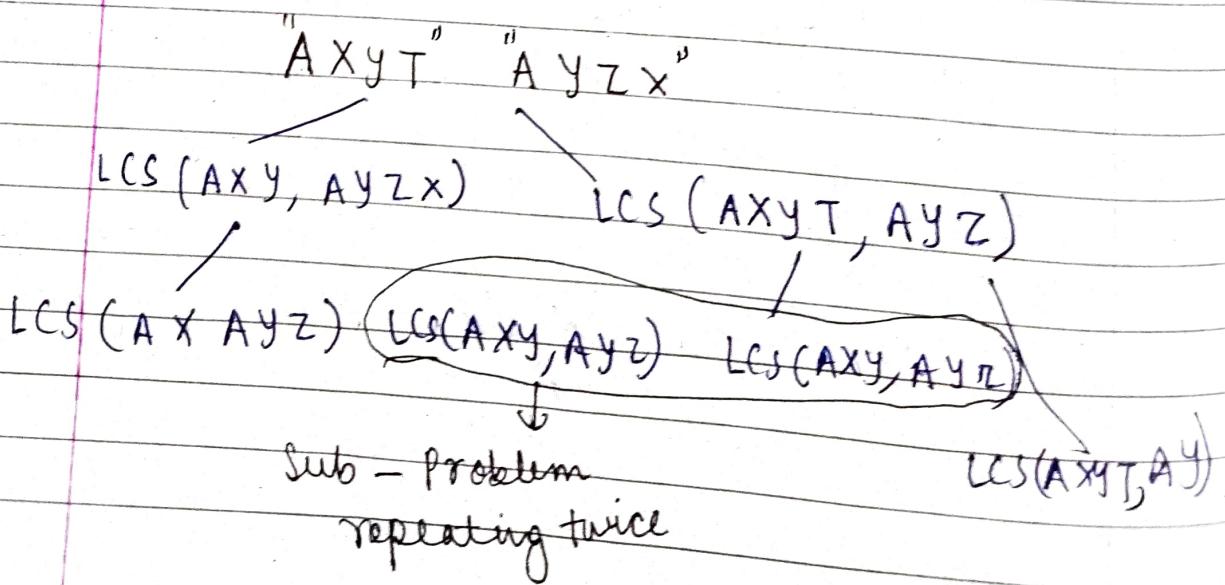
```
        return max (lcs(X, Y, n, m-1),  
                    lcs(X, Y, n-1, m));
```

```
}
```

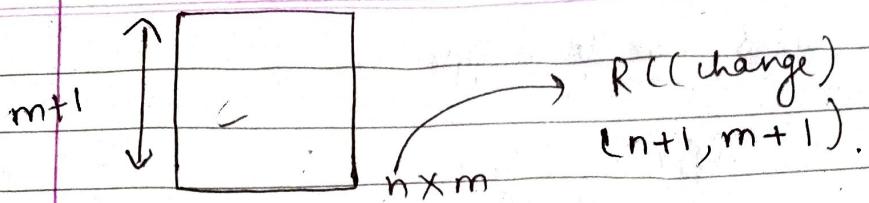
LCS memoization (bottom up approach)



R.C + table
 $(\boxed{\quad \quad \quad})$



5 Date _____
Page _____



int t[100][100];

int lcs()

int main()

memset(t, -1, sizeof(t));

lcs()

}

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

int lcs(string x, string y, int m, int n)

if (n == 0 || m == 0)

return 0;

if (t[m][n] != -1)

return t[m][n];

if (x[m-1] == y[n-1])

return t[m][n] = 1 + lcs(x, y, m-1, n-1);

else

return t[m][n] = max(lcs(x, y, m, n-1),

lcs(x, y, m-1, n))

:

exponential $\rightarrow O(n^2)$

LCS Top-down Approach

$x : @ \textcircled{b} \textcircled{c} d a \textcircled{f}$
 $y : @ c \textcircled{b} \textcircled{c} f$

$\text{O/P} \rightarrow 4 \text{ (abcf)}$

| | | length Y | | | | | |
|---------------|---|----------|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 |
| length X
↓ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | | | | | |
| | 2 | 0 | | | | | |
| | 3 | 0 | | | | | |

Base cond'n → initialization
 (R.S) (Top down)

int LCS(string X, string Y, int n, int m){
~~if (n <= 0 || m <= 0)~~
 dp[m+1][n+1];

for (int i=0; i<m+1; i++)

for (int j=0; j<n+1; j++)

if (X[i-1] == Y[j-1])

dp[i][j] = 1

for (int i=1; i<m+1; i++)

for (int j=1; j<n+1; j++)

if (X[i-1] == Y[j-1])

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

else

dp[i][j] = max(dp[i-1][j], dp[i][j-1])

between $dp[m][n]$;

Largest common Substring

I/p a : $\rightarrow \underline{a}bcde$
 b : $\rightarrow abf\cancel{c}e$

O/p $\rightarrow 2 (ab) \quad ab, c, e$

longest = ab

$\begin{matrix} \downarrow & \downarrow & \downarrow & \downarrow \\ a & b & c & d & e \\ a & b & f & c & e \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \end{matrix}$

* ② \rightarrow length = 2

m → length = 0

| | | | | | | |
|-----|---|---|---|---|--|--------------------|
| | 0 | 0 | 0 | 0 | | dis continuity = 0 |
| n ↓ | 0 | | | | | |
| | 0 | | | | | |
| | 0 | | | | | |

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

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

else

$$dp[i][j] = 0$$

// for max^m val.

int maxi = 0;

for (i = 0 → n+1)

 for (j = 0 → m+1)

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

return maxi;

Print LCS b/w 2 string

a : @ c b @ f
 b : @ b c d a f

O/P \rightarrow abc f

~~abc f~~

| | | | | | |
|---|---|---|---|---|---|
| a | b | c | d | a | f |
| @ | | | | | |
| c | | | | | |
| b | | | | | |
| a | | | | | |
| f | | | | | |

| | Ø | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|
| Ø | 0 | 0 | 0 | 0 | 0 | 0 | b |
| a | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| c | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| b | 0 | 1 | 2 | 2 | 2 | 2 | 2 |
| d | 0 | 1 | 2 | 3 | 3 | 3 | 3 |
| f | 0 | 1 | 2 | 3 | 3 | 3 | 4 |

Now, we need to print LCS.

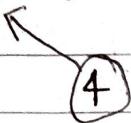
"f c b a"
 abc f ← reverse

if equal $i, j \rightarrow (i--, j--)$

if not equal $i, j \rightarrow \max((i-1, j), (i, j-1))$

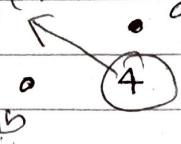
when equal

$(i--, j--)$



when not equal

$(\max \text{ between } a \& b)$



add in string

nothing to be added

int $i = m, j = n;$

String $s = " "$;

while ($i > 0 \&& j > 0$)

{
 a b
 if ($t[i-1] == t[j-1]$)
 {

$s.push_back(t[i-1])$

$i--;$

$j--;$

}

else {

 if ($t[i][j-1] > t[i-1][j]$)

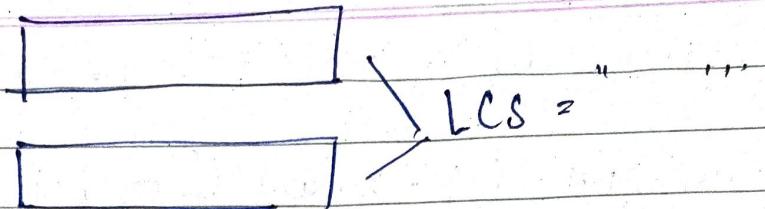
$j--;$

else

$i--;$

}

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



if ($a[i-1] == b[j-1]$)
 S.push_back ($a[i-1]$)
 $i--$
 $j--$

else

None in the direction of maximum
 reverse (s.begin(), s.end())

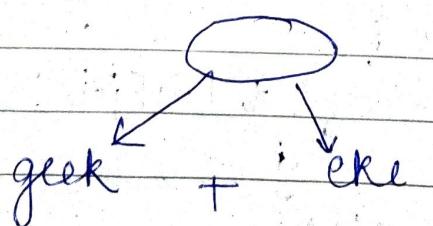
24. Shortest Common Supersequence

a: "geek."

b: "eke"

i) Problem statement :

→ 2 strings are given. We need to merge the strings such that we get both the subsequences. Mix 2 sequence to make it a supersequence.



Op

(geek)eke, geek(e)

find shortest

Subsequence \rightarrow sequence of events

$s_1 \quad s_2 \quad s_3$

Order should be maintained
but don't need to be
continuous always.

a : A G G T A B

b : G X T X A Y B

Super sequence : A G I G I T G I X A B T X A Y B

sequence

A G I G I X T X A Y B \rightarrow shortest supersequence

\swarrow \searrow
(A G I G I T A B) (G I X T X A Y B)

Give the length in O/P.

The one letter which is common write once.

A G G I T A B
G I X T X A Y B

G I T A B is common in both
i.e LCS.

Now thinking the brute force approach we
can merge both the string & then subtract
G I T A B.

a: AGIGTAB

b: GXTXAYB

Worst case: $a+b = AGIGTAB GXTXAYB$

length of S

$a+b$

↓
AGIGTABGXTXAYB - GTAB



A GIG X TX A Y B

Shortest length = $(m+n) - \text{LCS}$

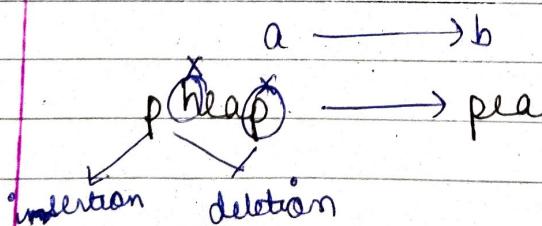
25. Minimum no of insertion & deletion to convert a string a to string b.

a: heap

$\%p = 1$ (Insertion)

b: pea

$\%d$ (deletion)



When to use LCS.

I/P

Q

%P

LCS

a:

~~LCS~~

%t

] → Pattern

Given Ques

b:

~~LCS~~

%t

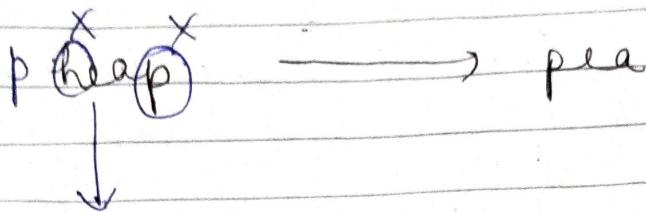
matching
algorithm

a:

insertion/
deletion

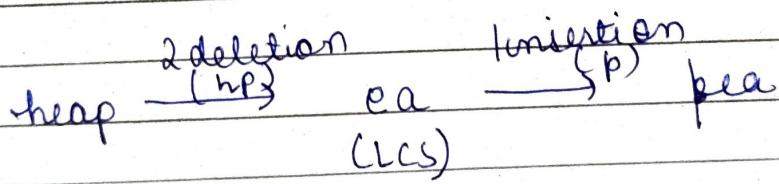
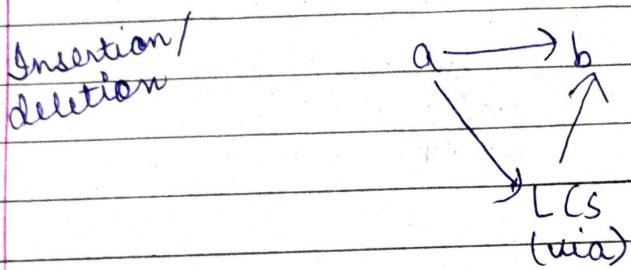
%t

Convert heap to pea.



"ea" is the LCS of both string

heap \rightarrow pea



$$\text{No of deletion} = \text{a length} - \text{LCS}$$

$$\text{No of insertion} = \text{b length} - \text{LCS}$$

26. Longest Palindromic Subsequence

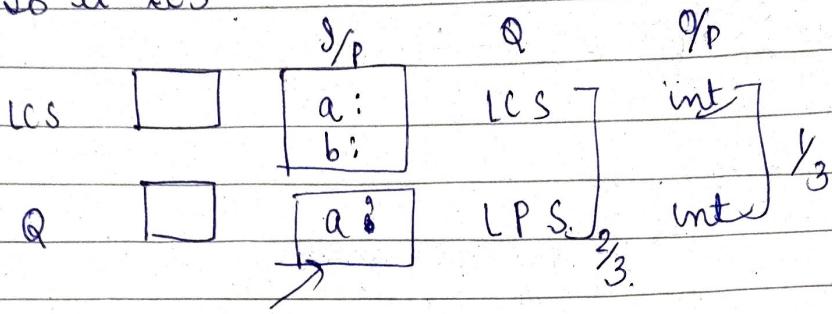
Problem Statement : S : a g b c b a

$$O/P = 5$$

$S \leftarrow$ longest [abcba
 bcb
 b] ✓

O/p \rightarrow 5 (abcba)

2) Is it LCS?



LCS S/p a, b
 LPS S/p a b = func(a)
 (hidden
 string/redundant)

"agbcba"
 ↓

LPS



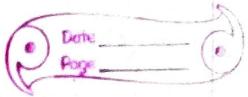
a \rightarrow agbcba

b \rightarrow reverse(a) abc bga.

↓
 LCS

@bCba@ @gbcba@ \rightarrow abcba

$LPS(a) \equiv LCS(a, \text{reverse}(a))$
 $LPS(agbcba) \rightarrow \text{return}(agbcba,$



28. Minimum no. of deletion in a string to make it palindrome

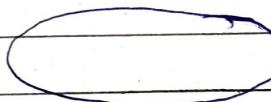
a : "agbcba"

$$Q_p = 1$$

S : agbcba

| | | | | | |
|---|---|---|---|---|---|
| a | g | b | c | b | a |
|---|---|---|---|---|---|

↓ No of deletions (minimize)



→ New string

(palindrome)

agbcba
bcb (palindrome) agcba (notpalindrome)
3

agbcba
x bcb x agbcba x agbcba
bcb (3) C (5) abcba (1)
(LPS) (LPS) LPS
Min^m.

\uparrow length of LPS \propto no of deletions \downarrow



LPS

\downarrow min^m no of deletions.

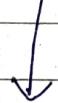


→ Palindromic subsequence.

\uparrow length of LPS \propto $\frac{1}{\text{no of deletions}}$

(longest palindromic subsequence)

agbcba



LCS(s, reverse(s))



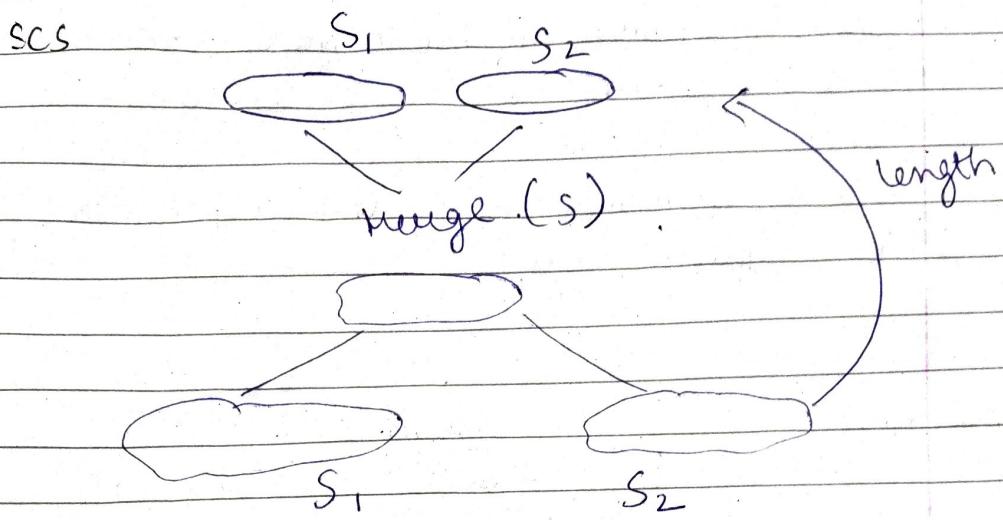
abcbab (LPS)

$$\text{Min no of deletion} = S - \underset{\text{(length)}}{\text{LPS(length)}}$$

29. Print shortest common subsequence

i/p a: a cbcaf
b: abcdaf

o/p: acbcdaf



Worst case acbcaf abcda^f

acbcda^f → point the whole string

mtn
acbcaf abcda^f
↓
now
mtn - LCS.

LCS → SCS
Point LCS Point SCS

| \emptyset | a | b | c | d | a | t | \rightarrow LCS |
|-------------|---|---|---|---|---|---|-------------------|
| \emptyset | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| c | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| b | 0 | 1 | 2 | 2 | 2 | 2 | 2 |
| c | 0 | 1 | 2 | 3 | 3 | 3 | 3 |
| f | 0 | 1 | 2 | 3 | 3 | 3 | 4 |



LCS → common ni hai to move kar jao
common hai to print karo.

SCS → common hai to ek bar print kro
else print karo.

LCS.

```

String s = " ";
while (i > 0 && j > 0) {
    if (a[i-1] == b[j-1]) {
        s.push_back(a[i])
        i--;
        j--;
    } else {
        if (t[i][j-1] > t[i-1][j])
            j--;
        else if (t[i-1][j] > t[i][j-1])
            i--;
    }
}

```

SCS

```

String s = " ";
while (i > 0 && j > 0) {
    if (a[i-1] == b[j-1]) {
        s.push_back(a[i])
        i--;
        j--;
    } else {
        if (t[i][j-1] > t[i-1][j])
            s.push_back(b[j-1]);
        else
            j--;
    }
    while (i > 0)
        s.push_back(a[i-1]);
    i--;
    while (j > 0)
        s.push_back(b[j-1]);
    j--;
}

```

- i) Now in SCS code we include the letter on moving at $i-1$ & $j-1$.
- ii) Now in $(i > 0 \&\& j > 0)$ we need to change because in case of LCS we don't need to stop at the topmost but in SCS we need to.

In case
of LCS

| | ϕ | a | b | f |
|--------|--------|---|---|---|
| ϕ | 0 | 0 | 0 | 0 |
| a | 0 | | | |
| c | 0 | | | |
| b | 0 | | | |
| d | 0 | | | |

| | ϕ | a | b | f |
|--------|--------|---|---|---|
| ϕ | 0 | 0 | 0 | 0 |
| a | 0 | | | |
| f | 0 | | | |
| d | 0 | | | |

Some
can't
stop
here.

ac,] lcs (" ")

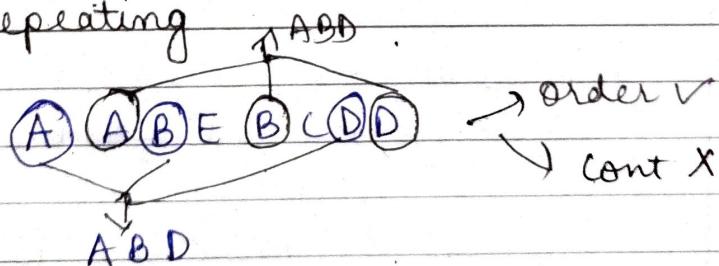
ac,] scs (ab)

longest repeating subsequence

Str = "A A B E B C D D"

%p = 3

Problem Statement - find a subsequence which is repeating



ABD (2x)] longest \rightarrow "ABD"
AB (2x) %p = 3

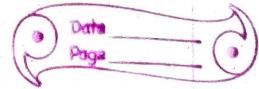
S = A A B E B C D D

E → 3
C → 5

A ~~K~~ 1 2 3 4 5 6 6
A K B E B C D D \rightarrow LCS = A A B (E) B C D D.

A A B B D D
once ↙ → ans

E & C are occurring once.
letter at the same index don't take



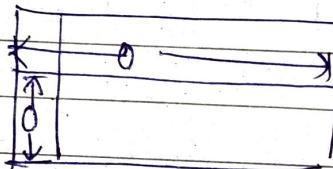
$E \rightarrow 3 \xrightarrow{i} j \quad i = j \times$
 $C \rightarrow 5$

$A \rightarrow 0, 1$
 $A \rightarrow 0, 1$ $B \rightarrow 2, 4$
same index diff index
 $B \rightarrow 2, 4$

Sum up:

$a = s$

$s \rightarrow \text{LCS}$ where $(i = j)$
 $b = s$



We can't take same letter for both the string during
LCS i.e. $(i = j)$

if $(a[i-1] = b[j-1] \text{ & } i = j)$

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

else

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

31. Sequence Pattern Matching

$a = "Axy"$
 $b = "ADXCpy"$

O/P \rightarrow T/F

Problem statement: Is string "A" a subsequence of "B"

a : AXY

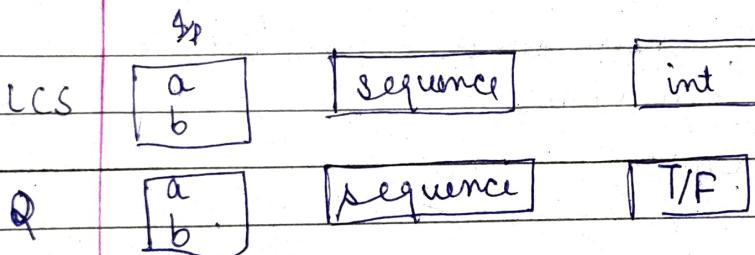
b : ADXCPY

b : A D X C P Y

a : AXY

O/P → True

LCS : AXY



S/P

b : ADXCPY

a : AXY

LCS : AXY (LCS == a)

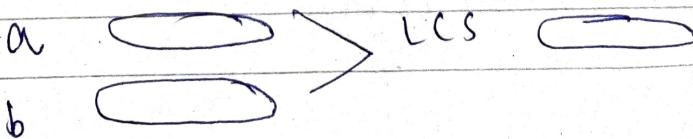
b : ADXCPY

a : AXYZ

LCS : A X Y (LCS != a)

if (LCS == A) return true
 else
 return false

Can it be done using length ?



a: Axy (3)

b: $ADXCPZ$ (6)

$LCS = 0 \text{ to } \min(m, n)$

m

n



$LCS = 2$

if ($LCS == a.length()$)

return true

else

return false.

3d. Min^m no of insertions in a string to make it palindrome

S/p: $s: "aebcbda"$

Q/p $\rightarrow 2$

$x a e b c b d a x$

↑ ↓ pc

adebcbeda. $x a d e b c b e d a x$

(2) (4)

Minimum = 2

Min^m no of deletion to make a string palindrome.

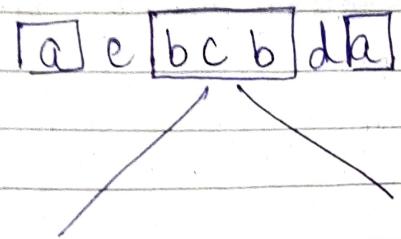
a e b e b d a

aea
(4)

abcba
(2)

$s.length() \rightarrow LPS.$ (deletion)

S : 'a e b c b d a'
 X X → Palindromic string



e x { delete a e ✓ { insert e & d
 d x { e & d d ✓ } & make their pair

no of insertions = no of deletions
 ||

$s.length - LPS.$

Deletion \rightarrow single (Pair) $\rightarrow \times$
 Insertion \rightarrow Single (Pair) $\rightarrow \checkmark$

33. MCM (Matrix chain Multiplication)

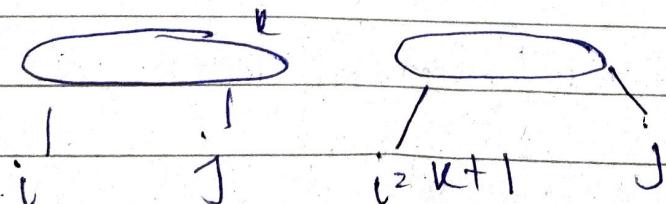
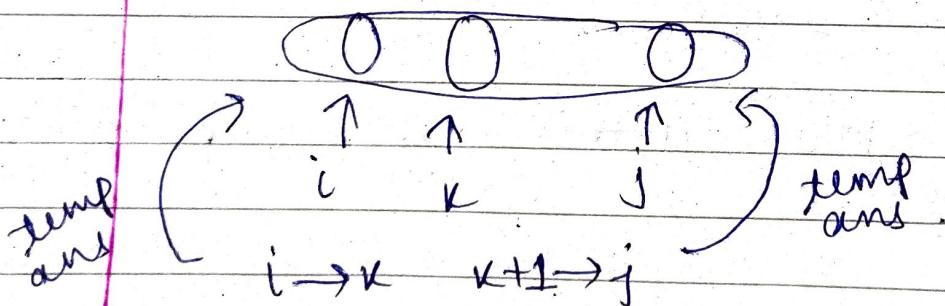
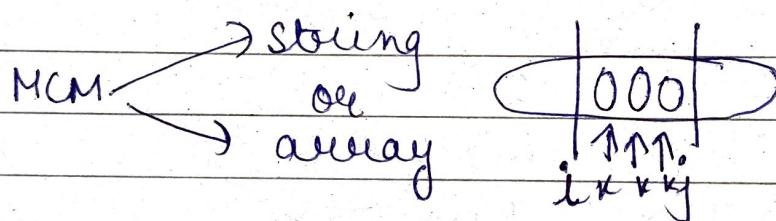
- 1) MCM
- 2) Painting MCM
- 3) Evaluate Exp. to True / Boolean Parenthesization
- 4) Min / Max value of an Expr.
- 5) Palindrome partitioning
- 6) Scramble string
- 7) Egg Dropping problem

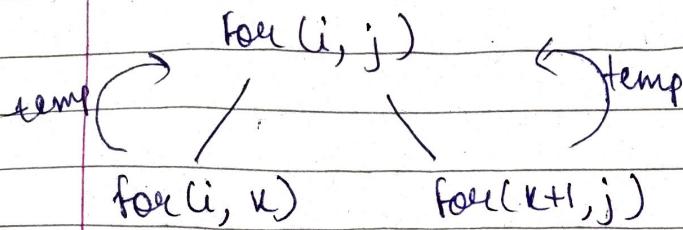
MCM format

- 1) MCM
- 2) Printing MCM
- 3) Evaluate Expr to True/ Boolean parenthesization
- 4) Min / Max value of an Expr
- 5) Palindrome partitioning
- 6) Scramble string
- 7) Egg dropping problem

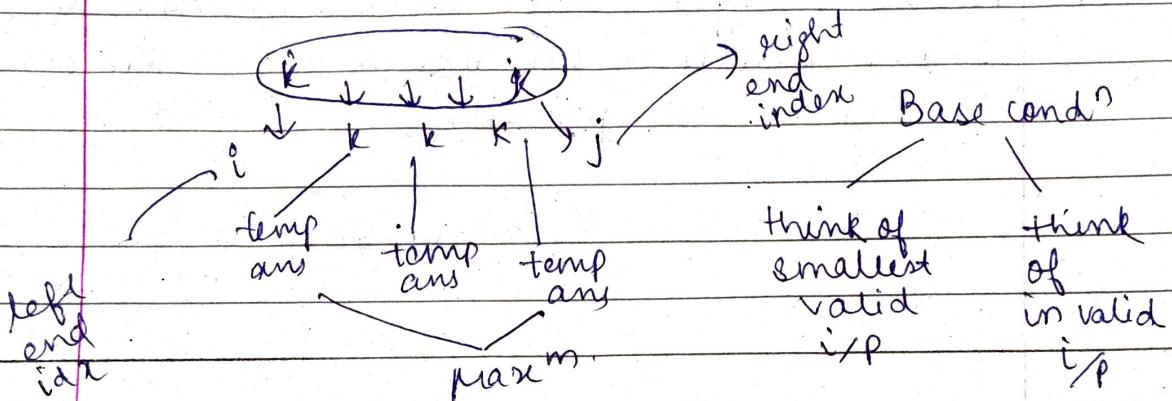
Identify \rightarrow MCM \rightarrow basic format

Identification + Format





$\text{ans} \leftarrow f(\text{temp ans})$



int solve (int arr[], int i, int j)
{

 if (i > j) → this may be diff accn'-le
 return 0;

 for (int k = i ; k < j ; k++)
{

 // Calc. temp ans .

 temp ans = solve(arr, i, k) +
 solve(arr, k+1, j);

 }

ans = func (temp ans)

MCM (Recursive)

Problem

Statement : $\text{arr}[] = \{ 40, 20, 30, 10, 30 \}$
 $\underbrace{A_1}_{m \times n}, \underbrace{A_2}_{m \times m}, \underbrace{A_3}_{n \times n}, A_4$

Some matrix are given like A_1, A_2, A_3, A_4 we have to multiply the matrix to reduce the cost (no of multiplications).

$b = c$ (then only we can multiply)

$$\begin{matrix} [] \\ a \times b \end{matrix} \quad \begin{matrix} [] \\ c \times d \end{matrix}$$

$$\begin{matrix} 2 \times 3 & 3 \times 6 \\ \downarrow & \downarrow \\ 2 & 3 & 6 \end{matrix} = 36 \text{ cost (no of multiplications)}$$

A_1, A_2, A_3, A_4

dimension $[] = \{x, y, z, w\}$

$$\begin{array}{cccc}
 A_1 & A_2 & A_3 & A_4 \\
 | & & \diagdown & \\
 (A_1 (A_2 A_3)) A_4 & ((A_1 A_2) (A_3 A_4)) & A_1 (A_2 (A_3 A_4))
 \end{array}$$

C_1 C_2 C_3

$$A \rightarrow 10 * 30$$

$$B \rightarrow 30 * 5$$

$$C \rightarrow 5 * 60$$

$$(A B) C = 10 * 30 * 30 * 5$$

\downarrow | $\quad \quad \quad$
~~10 * 5~~ ~~30 * 60~~ ~~10 * 30 * 5 * 60~~

~~300~~
~~500~~

$$(A B) C = (10 * 30 * 30 * 5) 5 * 60.$$

$A, A_2 A_3 A_4$

$$= 10 * 5 * 30 * 5 * 60$$

↓ do parenthesis
in such a way
cost is the least

$$= 10 * 30 * 5 * 60 * 10 * 5$$

$$= 4500.$$

$A_1 (A_2 A_3) A_4$ C_1
 $\cancel{(A_1 A_2) (A_3 A_4)}$ C_2
 $\cancel{A_1 (A_2 (A_3 A_4))}$ C_3

$\xrightarrow{\min \text{ cost}}$

$B C \rightarrow 30 * 5 * 60$

$A * B C \rightarrow 10 * 30 * 30 * 60$

$\xrightarrow{\text{Cost}}$

$10 * 30 * 60$

Bracket lagane par aleg aleg cost aa rahi
hai.

Date _____
Page _____

$$0 \quad 1 \quad 2 \quad 3 \quad 4$$
$$\text{arr}[] : \{ 40, 20, 30, 10, 30 \}$$

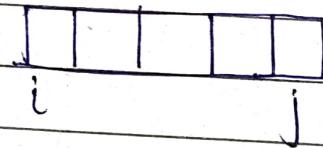
arr size n

n-1 matrix x.

$$\begin{aligned} A_1 &\rightarrow 40 * 20 \\ A_2 &\rightarrow 20 * 30 \\ A_3 &\rightarrow 30 * 10 \\ A_4 &\rightarrow 10 * 30 \end{aligned} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \begin{array}{l} A_1 * A_2 * A_3 * A_4 \\ \text{Cost (minim)} \end{array}$$

$$A_i \rightarrow \text{arr}[i-1] * \text{arr}[i] \quad (\text{no of multiplication should be less})$$
$$A[i] = \text{arr}[0] + \text{arr}[1]$$

Next step → Format



A₁ A₂ A₃ A₄

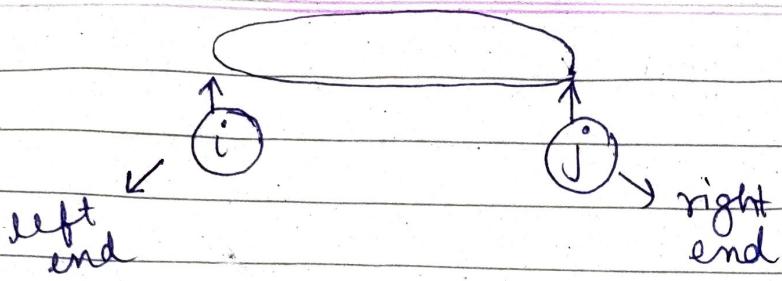
↓

(A₁) (A₂ A₃ A₄)

↑ ↓ temp ans
Min cost + Min cost

(A₁ A₂ A₃) (A₄)

put brackets on different places to get the answer
slide k at different positions.



| | | | | |
|----|----|----|----|----|
| 40 | 20 | 30 | 10 | 30 |
|----|----|----|----|----|

i i' j

$$A_i \rightarrow arr[i-1] * arr[i]$$

when $i=0$ $A_i \rightarrow arr[-1] * arr[0]$ X

when $i=1$ $A_i \rightarrow arr[0] * arr[1]$ ✓

$$A_j \rightarrow arr[j-1] * arr[j]$$

$$arr[3] * arr[4]$$

$i > j \rightarrow \text{size} = 0$

$i = j \rightarrow \text{size} = 1$

$\begin{pmatrix} i-1 \\ n-m \end{pmatrix}$

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

} return cost

$\rightarrow 1 \quad \quad \quad n-1$

int solve (int arr[], int i, int j)

{

if ($i > j$)

return 0;

int mn = INT_MAX;

for (int k = i ; k <= j-1 ; k++) {

int temp ans = solve (arr, i, k) +

solve (arr, k+1, j)

+ $arr[i-1] * arr[k] * arr[j]$

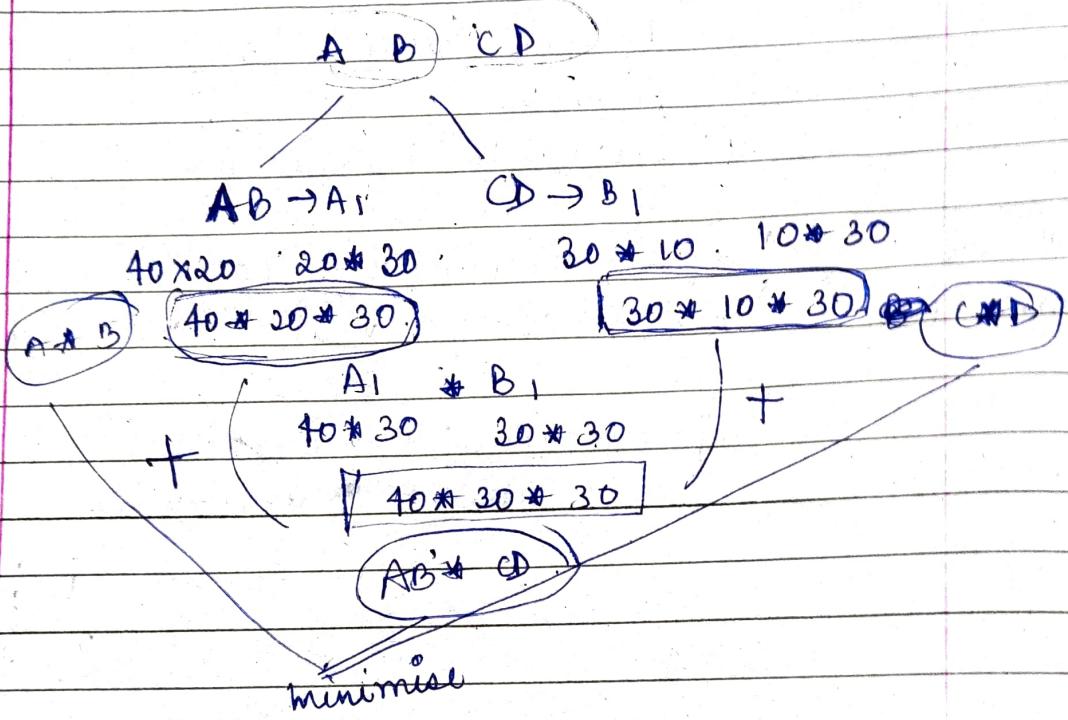
if (temp ans)

{

mn = temp ans;

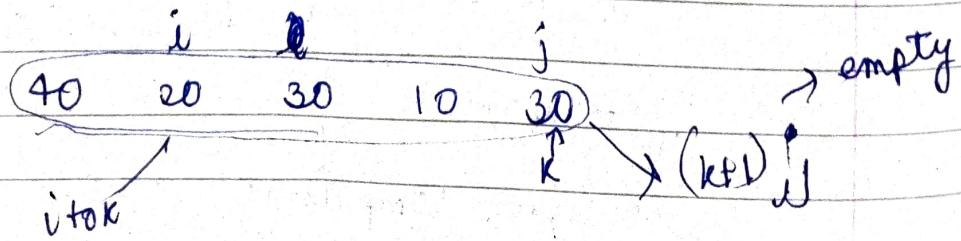
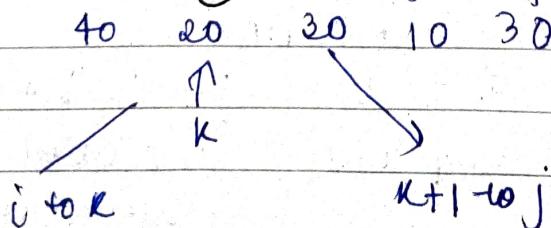
between mn

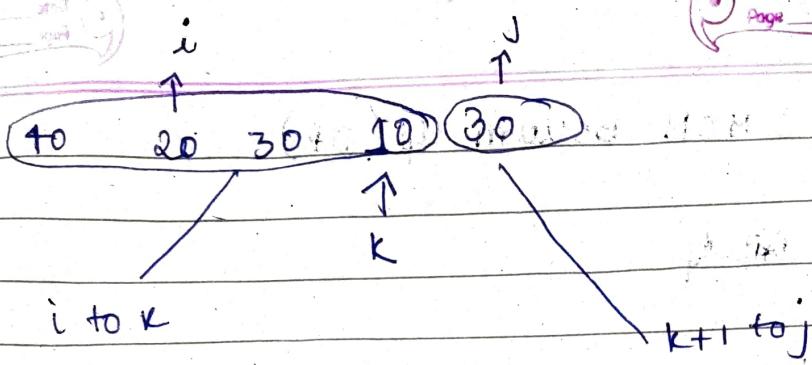
f



Steps for NCM

- 1) find i & j value
- 2) find eight base condn.
- 3) Move K \rightarrow i to j. (find K-loop scheme)
- 4) calculate cost for temp ans.



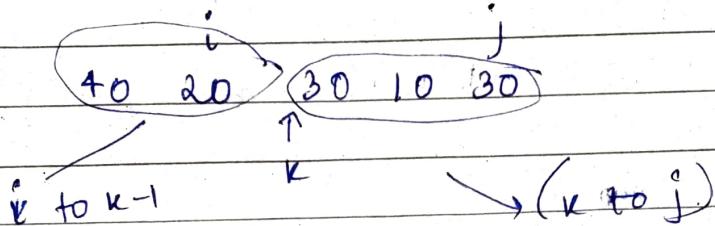


$40 * 20$

$20 * 30$

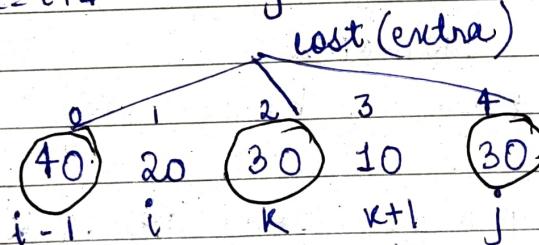
$30 * 10$

$$k = i \quad k = j - 1 \quad (i \rightarrow k \quad k+1 \rightarrow j)$$



2 schemes.

$$\begin{array}{l} k = i \rightarrow k = j - 1 \\ k = i + 1 \rightarrow k = j \end{array} \quad \begin{array}{l} i \rightarrow k \quad k+1 \rightarrow j \\ i \rightarrow k-1 \quad k \rightarrow j \end{array}$$



for ($i \rightarrow k$)

$40 * 20 * 20 * 30$

solve($i \rightarrow k$)

$40 * 20 * 30$

for ($k+1 \rightarrow j$)

$30 * 10 \quad 10 * 30$

\downarrow

$30 * 10 * 30$

solve ($k+1 \rightarrow j$)

$40 * 20 * 30 * 30$

\downarrow \rightarrow $40 * 20 * 30$ \rightarrow $aar[k]$

$aar[i-1]$

$\leftarrow 40 * 20 * 30 \rightarrow aar[j]$

dimension arr: []

MCM Bottom Up (DP)

~~Top Down~~

(dimension) arr[] : []

int dp[100][100]

memset (dp, -1, sizeof(dp))

int solve (int arr[], int i, int j)

if (i >= j)
~~dp[i][j] = 0~~
 return 0;

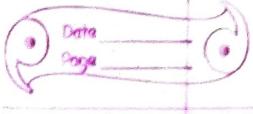
if (dp[i][j] == -1)
 return dp[i][j];

int mn = INT_MAX;
 for (int k = i ; k < j ; k++) {
 int temp_ans = solve (arr, i, k) +
 solve (arr, k+1, j) +
 arr[i-1] * arr[k] * arr[j];

if (temp_ans < mn)
 mn = temp_ans;
 !

dp[i][j] = mn;

return mn;



Pairwise Partitioning

```
int dp[100][100];
```

```
class Solution {
```

```
public:
```

```
int solve(int arr[], int i, int j) {
```

```
    if (i >= j)
```

```
        return 0;
```

```
    if (dp[i][j] == -1)
```

```
        return dp[i][j];
```

```
    int mn = INT_MAX;
```

```
    for (int k = i; k <= j - 1; k++) {
```

```
        int temp = solve(arr, i, k) + solve(arr, k + 1, j)
```

```
        + arr[i - 1] * arr[k] * arr[j]
```

```
        mn = min (temp, mn);
```

```
}
```

```
    dp[i][j] = mn;
```

```
    return dp[i][j];
```

```
}
```

```
int matrixmult (int N, int arr[]) {
```

```
    memset (dp, -1, sizeof (dp));
```

~~int ans =~~ solve (arr, 1, N - 1);

```
    return ans;
```

```
}
```

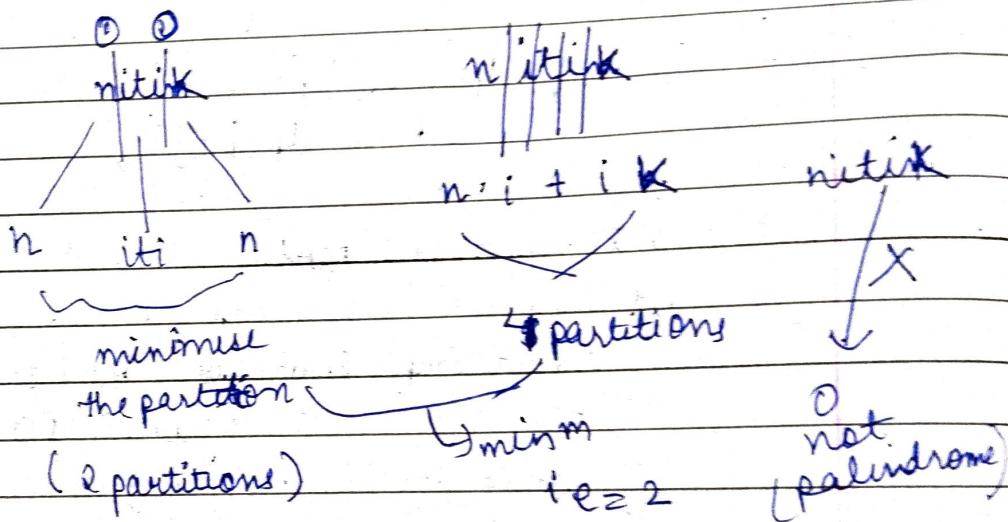
```
};
```

Palindrome partitioning

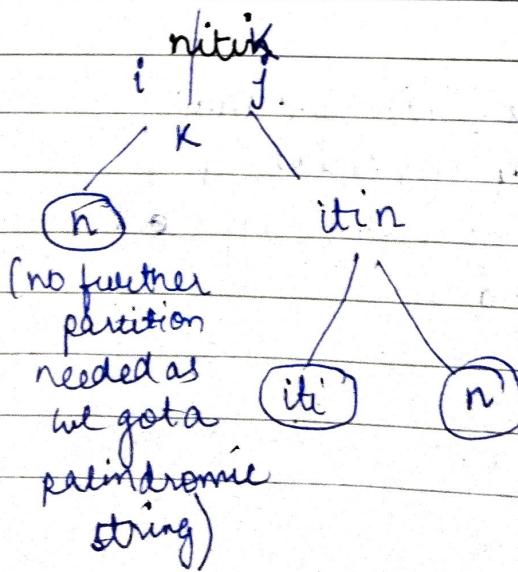
D) Problem statement

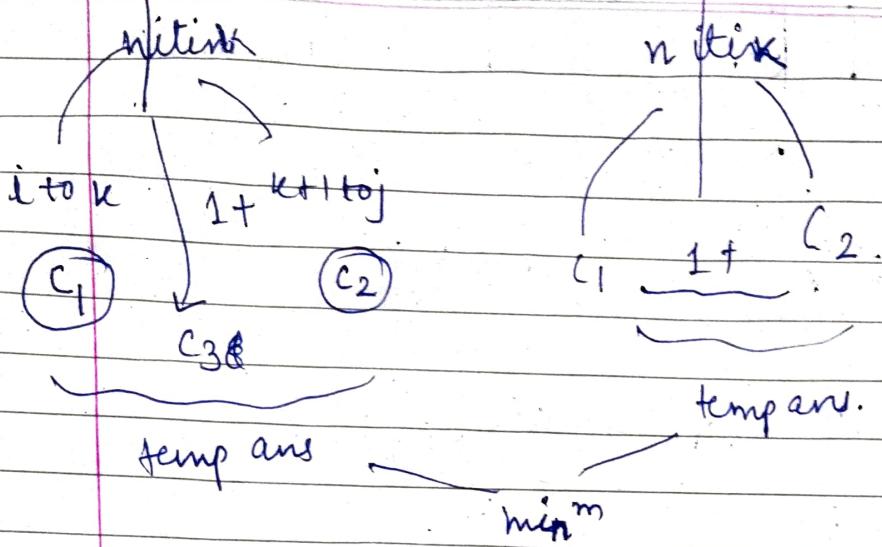
S: nitik → it is ~~not~~ as a whole is a palindrome
 Q: 2

divide string in such a way all the strings are palindrome.



worst case partition: ~~is~~ $n-1$





Format

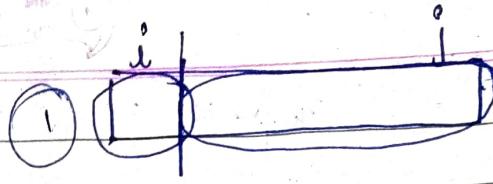
1. Find i & j
2. Find B.C.
3. Find K loop (scheme)

4. $\text{temp ans} \rightarrow \min^m$

Recursive code

$n \ i \ t \ i \ k$
 $i \ j$
 $i=0 \quad j=n-1$ (no need to start
i from 1 since
it doesn't require
i-1)
B.C $i=j \ size=1 \quad \{$
 $i>j \ size=0 \quad \}$ (if size is not given zero or equal
 no of partition would be
 0 as string is empty)
 is palindrome (s, i, j) ————— & if palindrome partition
 should be 0.

loop



$i \rightarrow k$

$k+1 \rightarrow j$

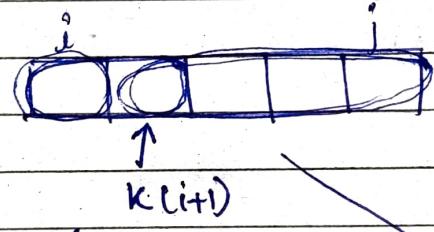
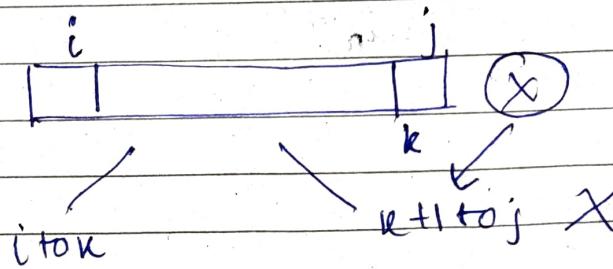
$$k=i \quad v=j-1$$

$i \rightarrow k$

$k+1 \rightarrow j$

$v \rightarrow i$

$k \rightarrow j$



$k(i+1)$

~~DECODE~~

$k \rightarrow j$

$$v = i+1$$

$$v = j$$

$$1) k=i$$

$$k=j-1$$

$i \rightarrow k$

$k+1 \rightarrow j$

$$2) k=i+1$$

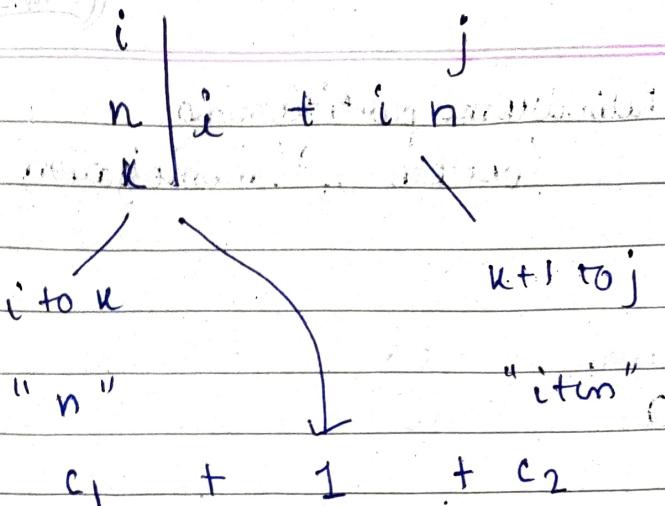
$$k=j$$

$i \rightarrow k-1$

$k \rightarrow j$

for (int $v = i$; $v = j-1$; $v++$)
{

 int temp = solve(s, i, v) + solve(s, v+1, j)



} ans = min (ans, temp)

} return ans;

Final code

```

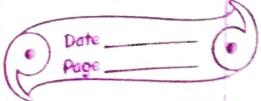
int solve (string s, int i, int j) {
    if (i >= j)
        return 0;
    if (isPalindrome (s, i, j) == True)
        return 0;
    int mn = INT-MAX;
    for (int k = i ; k <= j-1 ; k++) {
        int temp = 1 + solve (s, i, k) + solve (s, k+1, j);
        mn = min (mn, temp);
    }
}

```

~~mn~~ = min (mn, temp)

}
return mn;

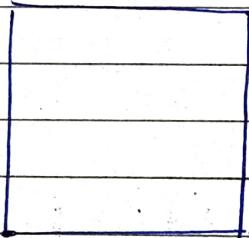
memset(-1) only allow 2 values



Palindrome partitioning (bottom up) (memoization)

$$\begin{array}{ll} i/p = \text{nitin} & \text{nitik} \\ o/p = 0 & 2 \end{array}$$

- 1) Initialise the matrix with -1.
- 2) Check if the value is $\{-1\}$ { value isn't evaluated
 $\neq -1$ value is evaluated
so we stored
return the value.



$n \times m$

✓

variables
which
changes
basically

i & j
changes.

```
int dp [100][100];
memset(dp, -1, sizeof(dp));
int solve ( string s, int i, int j ) {
    if ( i >= j )
        return 0;
    if ( is palindrome(s, i, j) )
        return 0;
```



```
if (dp[i][j] != -1)
    return dp[i][j];
int mn = INT_MAX;
for (int k = i ; k <= j-1 ; k++)
    int temp = solve(s, i, k) + solve(s, k+1, j) + 1
    mn = min (mn, temp);
}
dp[i][j] = mn;
return mn;
```

```
}
```

```
int palindromePart ( int string s) {
    int n = s.length() - 1;
    int ans = solve(s, 0, n);
    return ans;
```

```
bool ispalindrome (string s, int i, int j) {
```

```
if (i == j)
    return True;
```

```
if (i > j)
    return False;
```

```
while (i < j)
    if (s[i] != s[j])
        return False
```

```
else
```

```
i++;
```

```
j--;
```

GEEKSFORBEGEKS = ✓

INTERVIEWBIT = ✗

Further Optimization

Why the above code is not most optimized?

Since we are calling

$$\text{int temp} = \text{t} + \text{solve}(s, i, k) + \text{solve}(s, k+1, j)$$

^{RC}
(left) , ^{RL}
(right)

There is a possibility, might be one of the RC is solved or called.

The code will remain same but the diff is

$$\text{int temp} = \text{t} + \text{solve}(s, i, k) + \text{solve}(s, k+1, j)$$

$$\text{if } (+[i][k]) = -1$$

$$\text{left} = +[i][k]$$

else

$$\text{left} = \text{solve}(s, i, k)$$

$$+ [i][k] = \text{left}$$

$$\text{if } (+[k+1][j]) = -1$$

$$\text{right} = + [k+1][j]$$



else

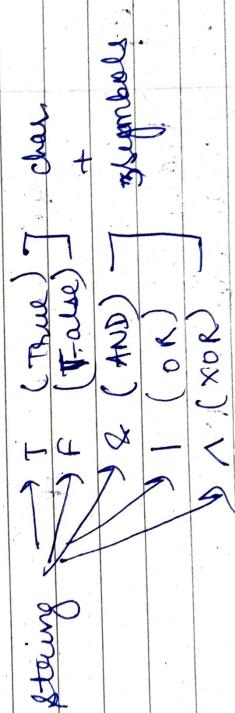
right = solve(s, k+1, j)
+ [k+1:j] = right.

int temp = 1 + left + right;

evaluate expression to true
Boolean parenthesization

String : True F and T.
Op : & ^

problem : A string is given . String might have some statement characters like T → true F → false

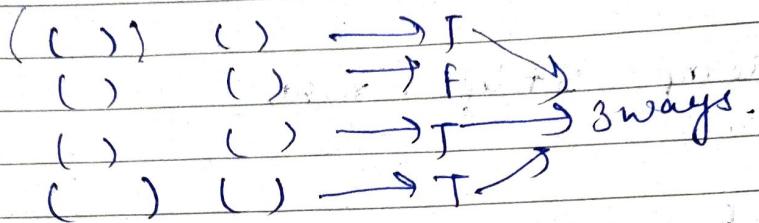


How to put bracket such that the expression evaluates to true.

Find the no of ways in which when bracket is put it evaluates to true.

Ques: "T | F & T ∨ F"

no of ways



In MCM we put brackets for min^m cost & in this also we do the same.

(T | F & T ∨ F)

$\xrightarrow{k-1} \xrightarrow{k} \xrightarrow{k+1}$

We need to break
bracket on
operator

Exprⁿ: operator Exprⁿ:

4 steps:

- 1) find i & j
- 2) find base condⁿ
- 3) Find k loop
- 4) temp ans & funcⁿ

↓
Main ans.

i j
T | F & T \wedge F

1) $i = 0$
 $j = \text{stacklength}() - 1$

2) BC i j
 $(T \text{ or } F \text{ and } T) \text{xor}(F)$
 ↓ ↓ ↓
 $i \text{ to } k-1$ K $k+1 \text{ to } j$

(left) Expreⁿ1 XOR Expreⁿ2 (right)
 ↓ ↑ ↓
 $i \text{ to } k-1$ K $k+1 \text{ to } j$

$$T \wedge T = \text{False}$$

$$F \wedge F = \text{False}$$

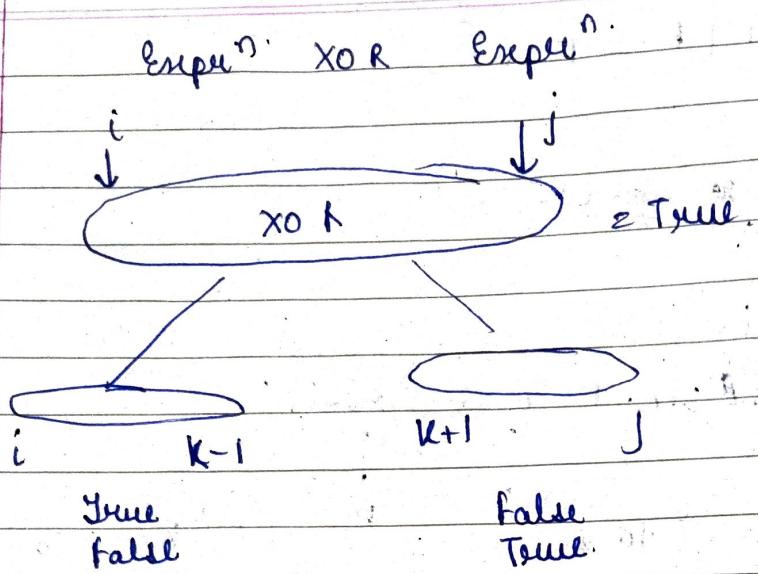
$$T \wedge F = \text{True}$$

$$F \wedge T = \text{True}$$

$$\text{no of true ways} = T \text{if } \text{left} + F \text{if } \text{right}$$

T \wedge F = T
 F \wedge T = T
 no of ways $\rightarrow 2$
 (True)

Expreⁿ XOR Expreⁿ
 ↓ ↓ ↓
 * * *
 + ← no of ways
 false



int solve (string s, int i, int j, boolean ^T
or
_{isFalse})

Base cond^n

i
T or F and T XOR F
T IF & T AF

boolean isTrue F

if ($i > j$) return false.

if ($i == j$)

if ($isTrue == \text{True}$)

return $s[i] == 'T'$

else

return $s[i] == 'F'$

loop

$$\begin{array}{c}
 i \quad j \\
 T \mid F \wedge T \wedge F \\
 \uparrow \quad \uparrow \quad \uparrow \\
 k = i+1 \quad k = j-1
 \end{array}$$

for (int k = i+1; k <= j-1; k = k+2)

 int ans = 0;

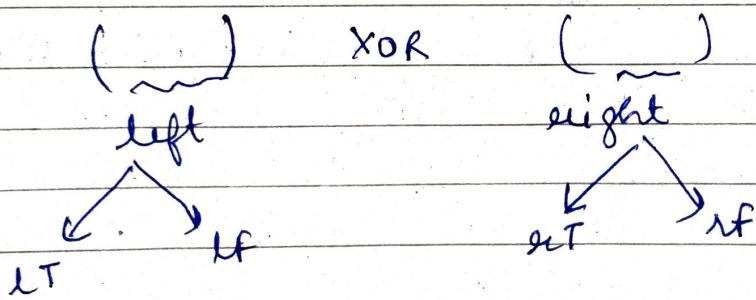
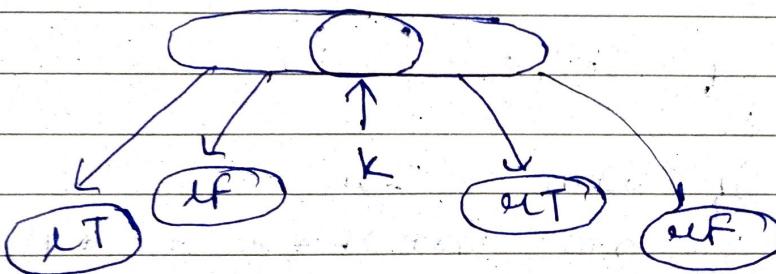
 int LT = (S, i, k-1, T);

 int LF = (S, i, k-1, F);

 int RT = (S, k+1, j, T);

 int RF = (S, k+1, j, F);

} temp ans.



$$ans += LT * LF + LF * RT$$

if ($s[k] = '8'$)

{

if (iTrue == True)

ans = ans + LT * aT

else {

ans = ans + LF * aT + LT * aF + LF * aF;

{

else if ($s[k] = '1'$) {

if (iTrue == True)

ans = ans + LT * aT + LT * aF +
LF * aLT..

else :

ans = ans + LF * aF.

{

else if ($s[k] = 'A'$)

{

if (iTrue == True)

ans = ans + LF * aT + LT * aF

else

ans = ans + LT * aT + LF * aF

{

return ans;

{

whole code

```
int solve (string s , int i , int j , bool isTrue ) {  
    if (i > j) {  
        return false ; }  
    if (i == j) {  
        if (isTrue == true)  
            return s[i] == 'T' ;  
        else  
            return s[i] == 'F' ;  
    }  
    for ( int k = i+1 ; k <= j-1 ; k+=2) {  
        int ans = 0 ;  
        int LT = solve (s , i , k-1 , T) ;  
        int LF = solve (s , i , k-1 , F) ;  
        int RT = solve (s , k+1 , j , T) ;  
        int RF = solve (s , k+1 , j , F) ;  
  
        if (s[k] == '&') {  
            if (isTrue == true)  
                ans = ans + LT * RT ;  
            else  
                ans = ans + LF * RT + LT * RF + LF * RF ;  
        }  
        else if ( s[k] == '|') {  
            if (isTrue == true)  
                ans += LT * RT + LT * RF + LF * RT ;  
            else  
                ans += LF * RF ;  
        }  
    }
```

```

else if ( S[k] == 'N' ) {
    if (isTrue == true)
        ans += LF * RT + LT * RF;
    else
        ans += LT * RT + LF * RF;
}
return ans;
}

```

Evaluate Expression to True Boolean
 Parenthesization = (memoization)
 (Bottom Up - DP)

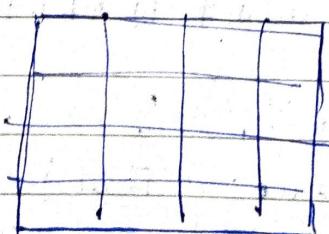
BD dp

Recursive \rightarrow Recursive call (R.C.)

Top down \rightarrow Table

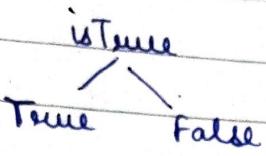
Bottom up \rightarrow R.C + table

P.S. \rightarrow put brackets in such a way that the expression evaluates to true.



$i * j * \text{isTrue}$
 (3d) C^3

int t [100][100][2]



More better way : We can use map

| map. | value |
|---|-------|
| $i, j, \text{ restore}$
$i + " " + j + " " + \text{restore}$ | |

unordered_map<string, int> mp;

```

int main() {
    mp.clear()
    solve()
}
  
```

After Base cond:-

```

string temp = to_string(i);
temp.push_back(' ');
temp.append(to_string(j));
temp.push_back(' ');
temp.append(to_string(isTrue));
  
```

```

if(mp.find(temp) != mp.end()){
    return mp[temp];
}
  
```

return mp[emp] = ans;

b.

Egg Dropping Problem

I/p: e = 3

f = 5

O/p: 3 → minimize no
of attempts
in worst
case.

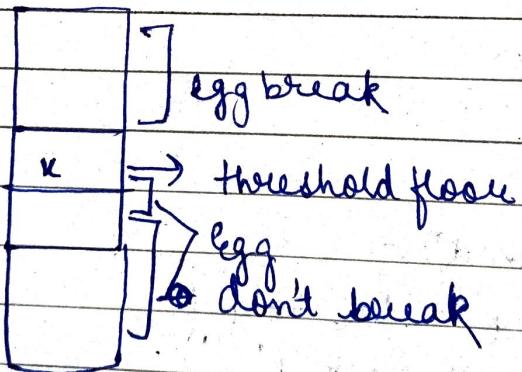
Problem

statement :



0 0 0

Beggs.



We are in a building, we need to find the
minⁿ no of attempts to find the critical
floor.

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

0 0 0

3 eggs

safe strategy \rightarrow worst case (1 egg) so drop from the last it won't break until threshold.

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

→ egg break (threshold floor)

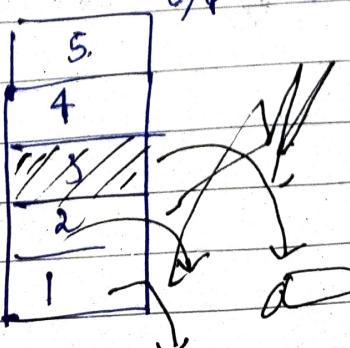
| |
|---|
| 7 |
| 6 |
| 5 |
| 4 |
| 3 |
| 2 |
| 1 |

worst-case \rightarrow minimize no of attempts to find threshold floor.

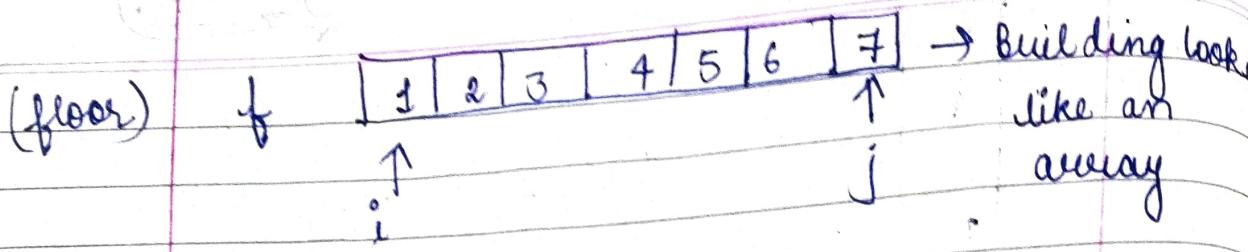
0 0 0
min
 $e = 3$

$$\frac{g}{f} e = 3 \\ f = 5$$

$0/f = 3$ attempts



egg break.

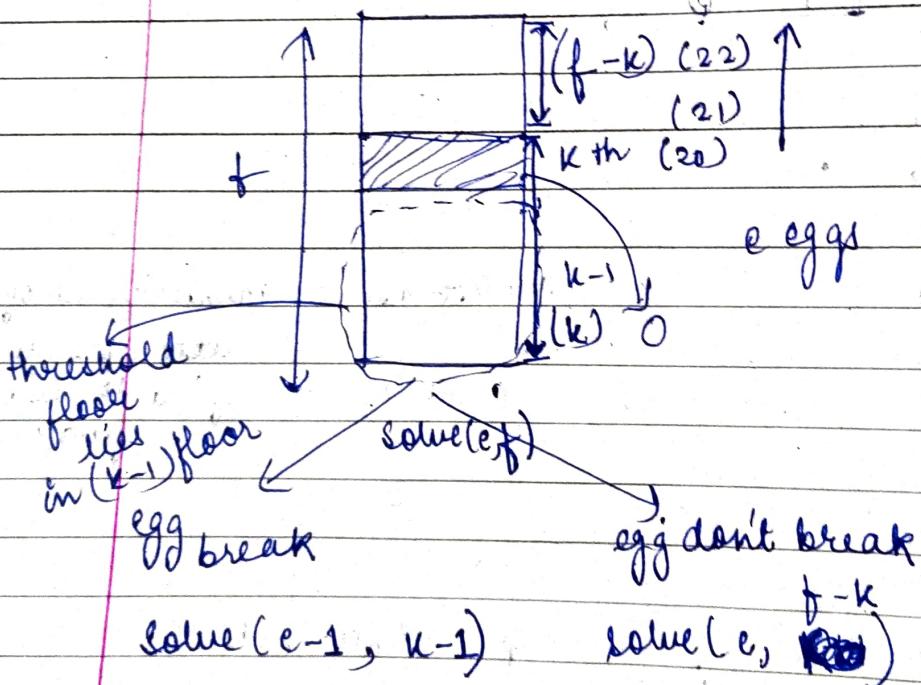


from where to take kth
 \rightarrow We will check for all k values.
 for $k=1$ $k=f$ $k++$

Base condⁿ. (smallest valid i/p)

$e = 1$ return f

$f = 0/1$ return f



Date _____
Page _____

```
int solve (int e, int f)
```

{

```
if (f == 0 || f == 1)  
    return f;
```

```
if (e == 1)  
    return f;
```

```
int mn = INT_MAX;
```

```
for (int k = 1; k <= f; k++)
```

{

for the worst
case

```
int temp = 1 + max ( solve (e - 1, k - 1),  
                     solve (e, f - k) );
```

```
mn = min (mn, temp);
```

}

```
return mn;
```

{

Egg Dropping Memoization

 $| \leq T \leq 30$ $| \leq e \leq 1000$ $| \leq f \leq 1000$

// globally
defined

```
int dp[100][100];
```

```
memset (dp, -1, sizeof(dp));
```

| | | | |
|---|---|---|---|
| - | - | - | - |
| - | - | - | - |
| - | - | - | - |
| - | - | - | - |

ex:

```
int solve (int e, int f) {
```

```
if (e == 1)
```

```
    return f;
```

```
if (f == 0 || f == 1)
```

```
    return f;
```

```

if ( $t[e][f] \neq -1$ )
    return  $t[e][f]$ ;
int mn = INT_MAX;
for (int k = i; k <= f; k++) {
    int temp =  $\max(\text{solve}(e-1, k-1),$ 
             $\text{solve}(e, f-k)) + 1$ ;
    mn = min(mn, temp);
}
return mn;  $t[e][f] = mn$ ;
}

```

Further Optimization

Inside the loop

```

if ( $t[e-1][k-1] \neq -1$ )
    int low =  $t[e-1][k-1]$ 
else
    low = solve(e-1, k-1)
     $t[e-1][k-1] = low$ ;

if ( $t[e][f-k] \neq -1$ )
    int high =  $t[e][f-k]$ ;
else
    high = solve(e, f-k);
     $t[e][f-k] = high$ ;

int temp =  $\max(\text{low}, \text{high}) + 1$ ;

```