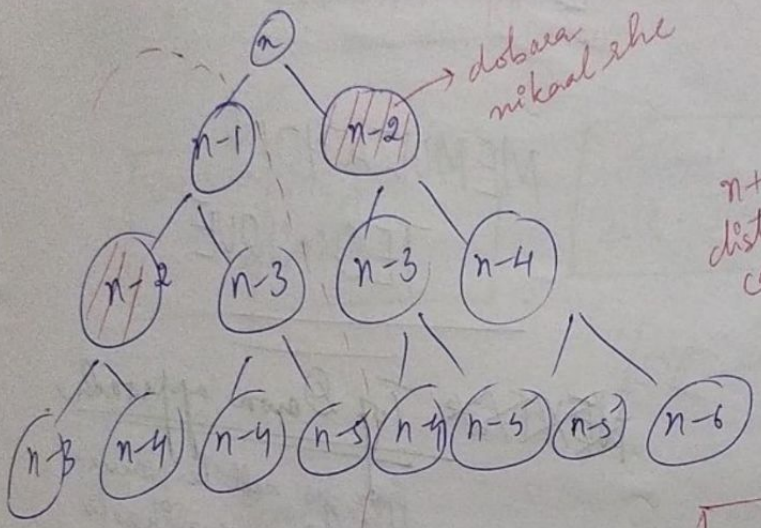
# Dynamic Programming

(Enhanced Recursion)

Fibonacci

$T(n) = T(n-1) + T(n-2) + k$

recurrence relation




dobara nikaal rhe

n+1 distinct calls

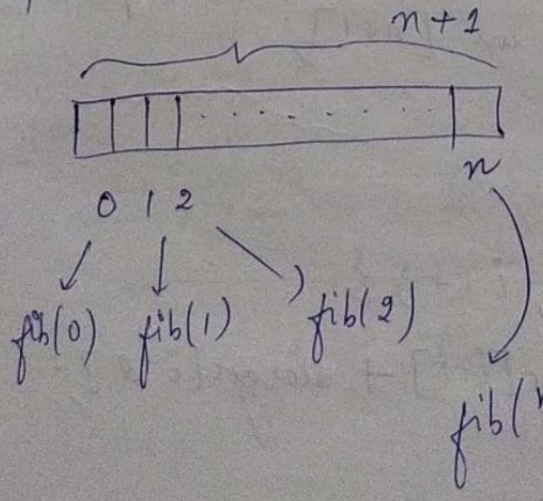
fib(n)
fib(n-1)
fib(n-2)
.
.
fib(0)

Height = n

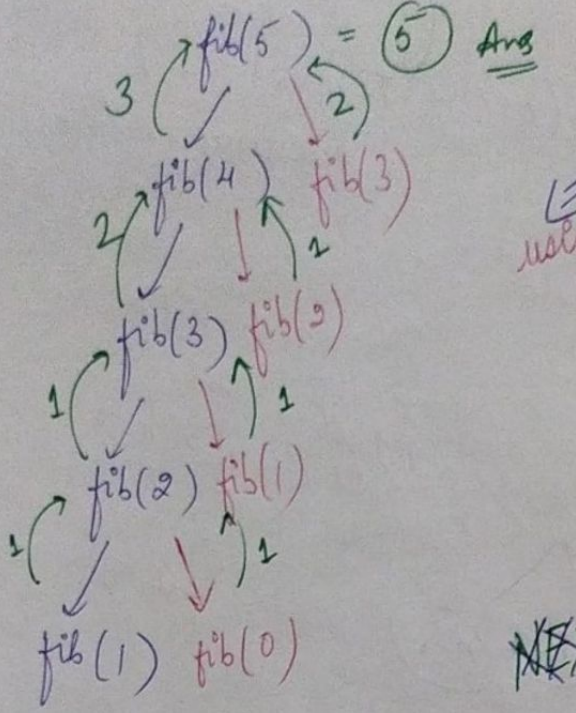
$O(2^n)$

repeating same same kaam

* We are actually making too many redundant calls
O(n+1) me kaam ho jaana chahiye tha but O(n) me kar rhe.
* So, we'll store the results to use it whenever required



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




5+4 = 9 steps

$O(2n-1) = O(n)$

Bottom-Up Approach

using DP

MEMOIZATION TECHNIQUE

~~NEXT~~ Top Down approach.

jab jis cell ki value chahiye use nikalo aur use karo

Dynamic prog says → if you can eliminate recursion
↓
you should do

Memoization
Top Down Approach ⇒ DP (Bottom up Approach)

```
int storage[] = new int[n+1];
storage[0] = 0
storage[1] = 1
for (int i=2; i<=n; i++) {
    storage[i] = storage[i-1] + storage[i-2];
}
return storage[n];
```

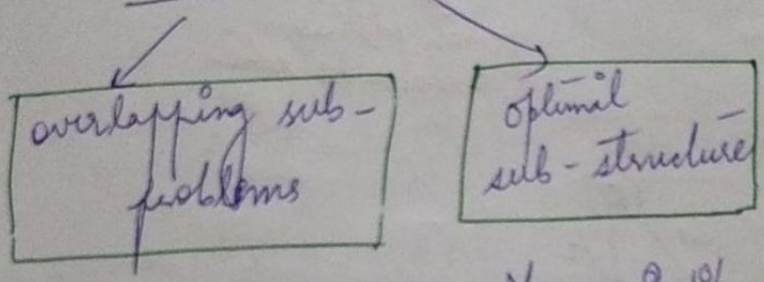
Fib recursive

↓

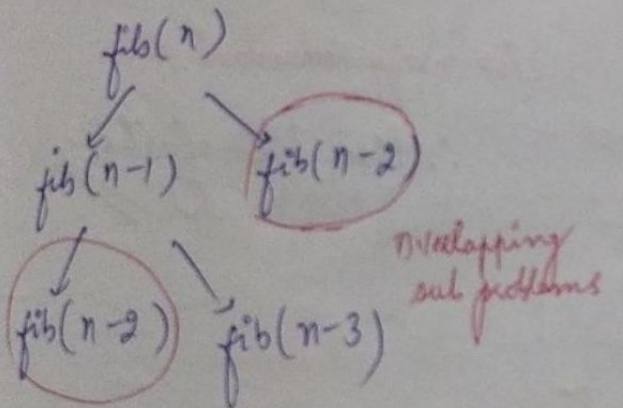
Fib Memoization → Top Down approach

↓

Fib DP → Bottom Up Approach




Yes // in fibonacci ques → { min, max, largest, greatest }



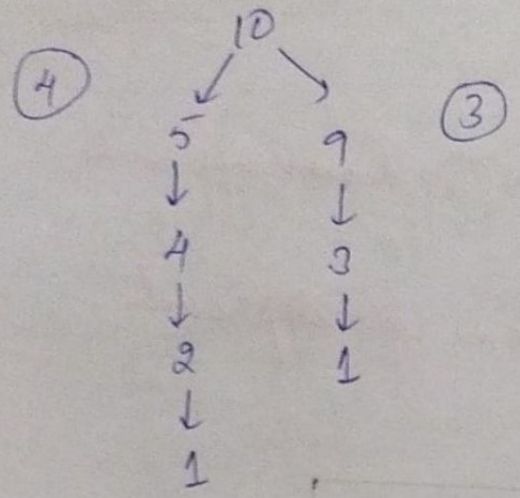

redundancy!!

Q. Min steps to 1

1) Subtract 1 from n
2) Divide n by 2
3) Divide n by 3

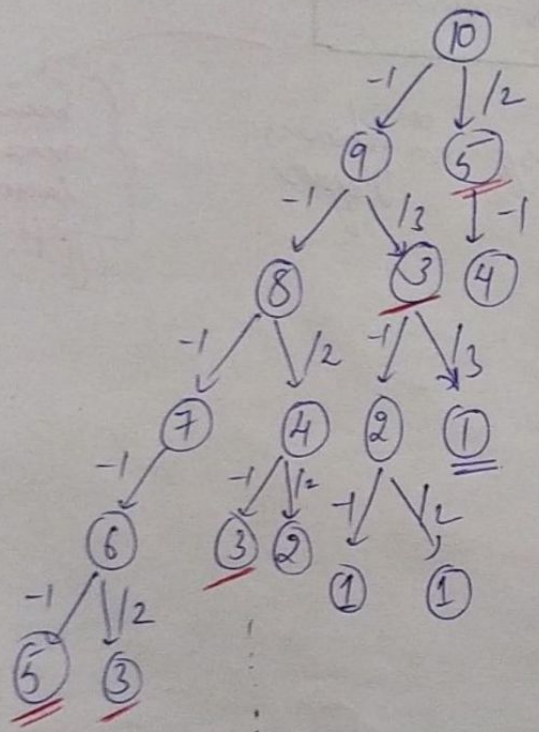
5 $\xrightarrow{-1}$ 4 $\xrightarrow{/2}$ 2 $\xrightarrow{/2}$ 1

3 steps



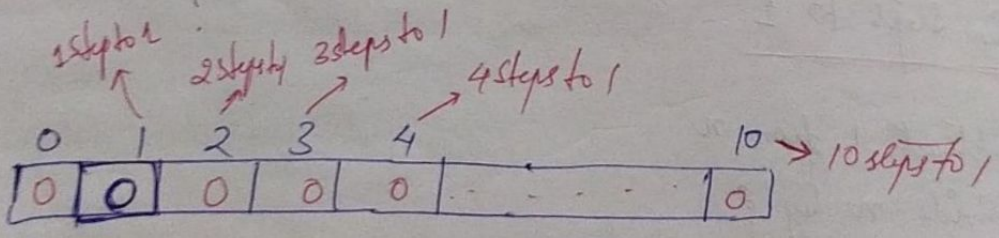

op1 → div by 2
op2 → div by 3
op3 → subtract 1

ye toh pakka perform krr sake




Too many redundant calls

∴ Use Memoization




→ Bottom Up Approach – DP

Q. Minimum Count of Squares

$n = 2 \rightarrow 1^2 + 1^2$ — Ans = 2

$n = 3 \rightarrow 1^2 + 1^2 + 1^2$ — Ans = 3

$n = 4 \rightarrow 1^2 + 1^2 + 1^2 + 1^2$

$2^2$ — Ans = 1

$n = 5 \rightarrow 1^2 + 1^2 \ldots + 1^2$

$2^2 + 1^2$ — Ans = 2

$n = 6 \rightarrow 1^2 + \ldots + 1^2$

$2^2 + 1^2 + 1^2$ — Ans = 3

Chain of squares

$n = x^2 + (y^2 + z^2 + \ldots)$

$i = 1 \quad = 1^2 + \underbrace{n - 1^2}$

1 + Minimum int. required — Ans

N = 10

$i = 1^2 +$ (9 → $n-1$)

$i = 2^2 +$ (6 → $n-2^2$)

$i = 3^2 +$ (1 → $n-3^2$)

$\vdots$

$i = n/2$

OR

$\sqrt{n}$

minimum

Ans

① Recursively
② Memoization
③ DP

N=10 $i = n/2 = (5)$ $i = \sqrt{n} = (3)$

$i=1$ $1^2 + 1^2 + \cdots + 1^2$ Ans = 10

$2^2 + 2^2 + 1^2 + 1^2$ Ans = 4

$3^2 + 1^2$ Ans = 2

$(i <= 3)$

10

$op = 1^2 + countSq(10-1^2)$

$op = 1 + countSq(9)$

$1 + countSq(8)$

$1 + (7)$

$1 + (6)$

$1 + (5)$

$2^2 + (6)$ → $3^2 + 1$ → 1

$1 + (5)$

$1 + (4)$

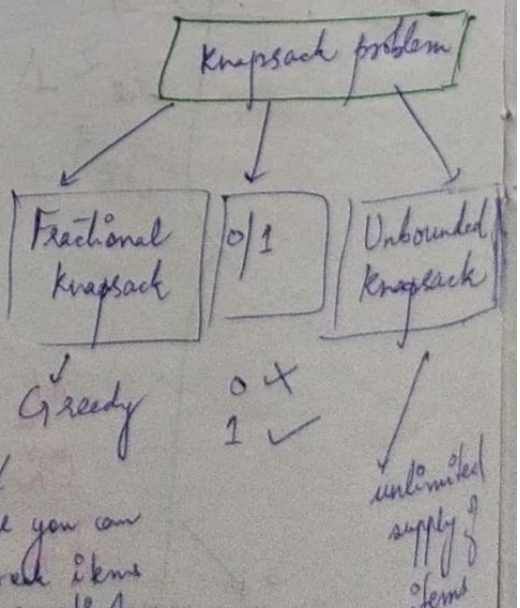
indiabix

Repetition

use Memoization

** 0-1 Knapsack problem

1) Subset Sum
2) Equal Sum Partition
3) Count of subset Sum
4) Minimum Subset Sum Diff
5) Target Sum
6) # of subset with given diff



1. Choice ✓
2. Optimal ✓

DP

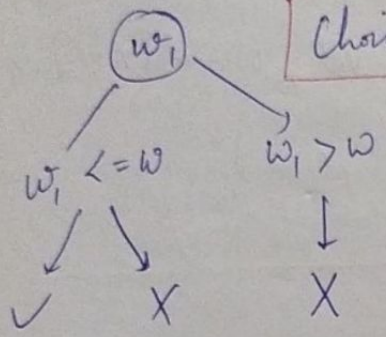
Constraints
$n <= 100$
$W \leq 1000$

wt[] : 1 3 4 5
val[] : 1 4 5 7
W : 7kg
o/p : max profit

Recursive Solution

DP : recursion + storage

Choice diagram




Base Cond$^n$ : smallest valid i/p

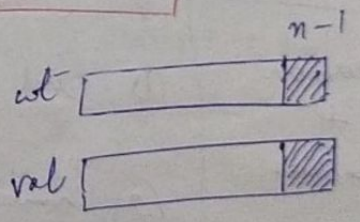
size of array → n → 0
W → 0 valid

```
if( n == 0 || w == 0 ){
    return 0;
}
```

→ profit will be zero

⇒ Base Cond$^n$.

From choice diagram




```
if( wt[n-1] <= w ) {
    return (val[n-1] +
        max of  knapsack(wt, val, w - wt[n-1], n-1)
    and ( knapsack(wt,val,w,n-1) )
}
```

left node

```
if( wt[n-1] > w ){
    return knapsack(wt, val, w, n-1)
}
```

→ $w_1 > w$ | X

right node

```
int knapsack (int wt[], int val[], int W, int n) {
    if (n == 0 || W == 0)
        return 0;
    if (wt[n-1] <= W)
        return Math.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 knapsack (wt, val, W, n-1);
```

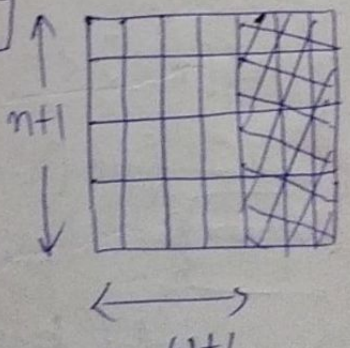
## memoize

* jo variables change ho rhe hain → unki hi matrix bnegi (Yahan W & n)

Recursion + 4 times

① int t[n+1][W+1]

② initialise with -1 every element




③ if value == -1 then call recursion fn & Save value

④ agar -1 nhi hai, then return krdo value (kyunki saved thi na)

UP NEXT :- RC

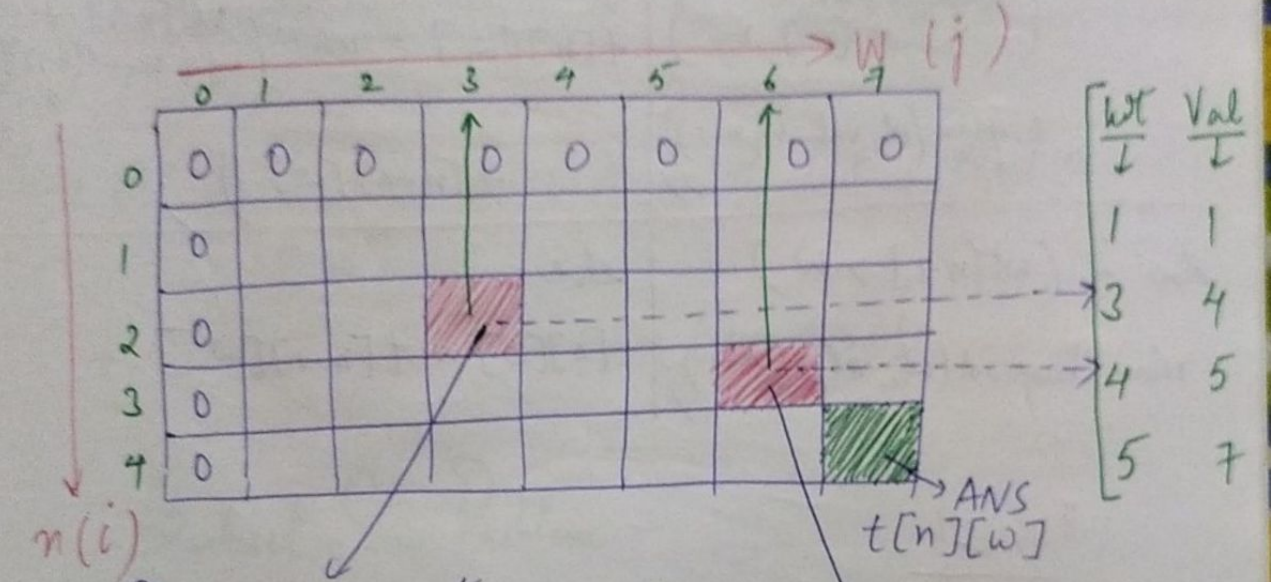
RC + 


Iterative

# DP Code — Knapsack Tabulation

I/P
wt[] = [1 3 4 5]
val[] = [1 4 5 7]
W = 7
n = 4

t[n+1][w+1]  t[5][8]




Initialize with 0

max profit

sub problem: wt[] = [1 3], val[] = [1 4], W = 3

sub problem: wt[] = [1 3 4], val[] = [1 4 5], W = 6

Smaller problems → Sub problems
(every box is a smaller problem)

RC → BaseCond" ⟶ DP me convert to Initialization

if(n==0 || w==0) return 0;  ← n=0 or W=0 in table

↳ Initialization

**Recursive**

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

if (wt[n-1] <= w)
return 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 knapsack(wt, val, w, n-1);
```

**Iterative**

```
for(int i=0; i<n+1; i++)
    for(int j=0; j<w+1; j++)
        if(i==0 || j==0)
            t[i][j] = 0;

if (wt[n-1] <= w)
t[n][w] = max(val[n-1] + t[w-wt[n-1]][n-1],
              t[n-1][w]);

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

FINALLY

```
for(int i=0; i<n+1; i++){
    for(int j=0; j<w+1; j++){
        if(i==0 || j==0)
            t[i][j] = 0;
    }
}
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]);
```

Initialisation

```
else
    t[i][j] = t[i-1][j];
}
return t[n][W];
}
```

Sum up

→ init t[n+1][W+1]
→ Initialisation → acc to Base Condⁿ in Recursive Code
→ Looping through & filling cells value
→ We know that Ans is in the last cell i.e.
t[n][W]
→ We will return t[i][j] → o/p

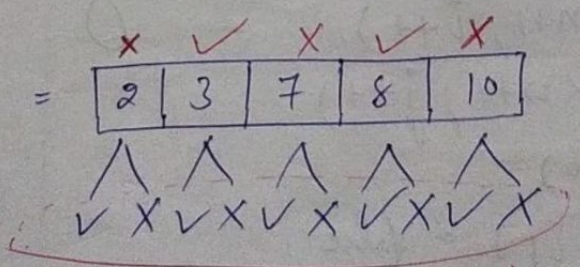
# Subset Sum Problem

arr[] : 2 3 7 8 10
Sum = 11

Q. If subset exists ? Sum = 11
[3, 8] True

Q. Sum = 14 False

1. Problem Statement
2. Similarity
3. Code variations
   - Code
   - Initialization




Sum = 11 W

{3, 8} → True

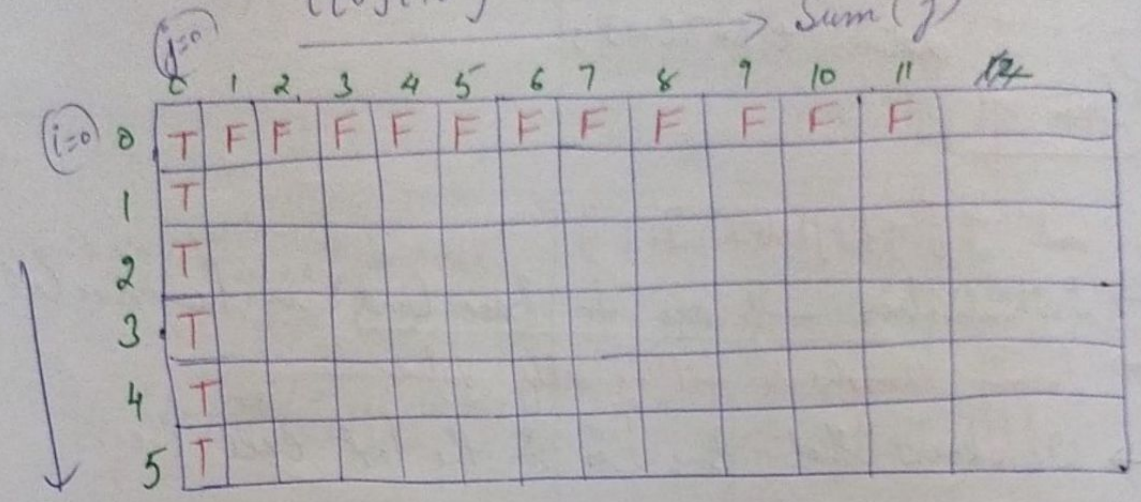
knapsack ki yaad aani chahiye

arr[] : 2 3 7 8 10
Sum : 11

t[n+1][w+1]

t[5+1][11+1]

t[6][12]




n (arr size + 1)

(i) arr

arr[] : Sum : 0 → { } True

arr[] : 2 Sum : 0 → { } True

arr[] : 2 3 Sum = 0 → { } True

arr[] : Sum = 1 → False

arr[] : 2 Sum = 2. → False

Initialization of matrix

① Code - Initialization

```
for(int i=0; i<n+1; i++)
    for(int j=0; j<w+1; j++)
        if (i==0)
            t[i][j] = false;
        if (j==0)
            t[i][j] = true;
```

## Compared to knapsack :-

wt → arr

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

max → OR (∵ boolean)

t[i][j] = t[i][j - arr[i-1]] || t[i-1][j] ;

(val) OR

else

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

(3) return [n][sum] ;

## Equal Sum Partition Problem

arr[] : {1 5 11 5}

o/p : T/F

→ 2 partitions of array with equal sum
Is this possible ?
T/F.

{ } | { }

1, 5, 5 | 11 → equal sum

Subset sum → se related hai //.

22 | 23

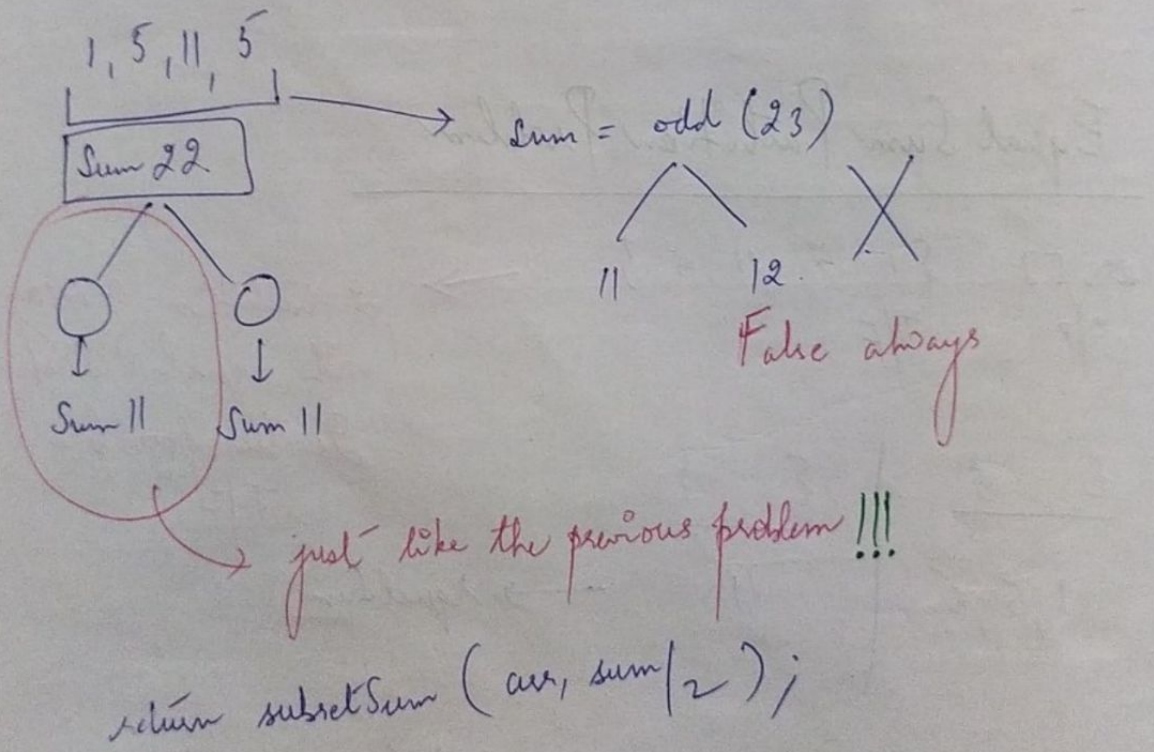
11 11 | 11 12.

Even | Odd

arr[ ] → given

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

if(sum % 2 != 0)
    return false
else {
    think
}
```




return subsetSum ( arr, sum/2 );

CODE:

```
input: arr[], n

int sum = 0
for(int i=0; i<n; i++){
    sum = sum + arr[i];
}
if (sum % 2 != 0) {
        return false
}
else{
    return subsetSum(arr, sum/2);
}
```

↓ last question code.

## Count of Subsets with a Given Sum

IP: arr[] : 2 3 5 6 8 10
Sum : 10

{2,8}, {10}, {2,3,5}  count = 3 Ans//

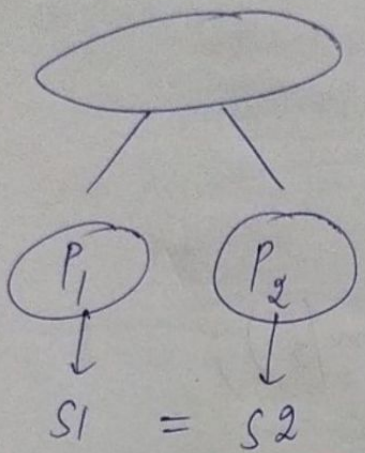
t[6+1][10+1] => t[7][11]

```
if (arr[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];
```

# Minimum Subset Sum Difference

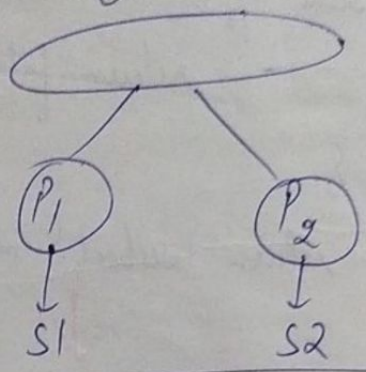
arr[]: | 1 | 6 | 11 | 5 | → ip

op : 1 → op




Equal partition problem

S1 - S2 = 0




S1 - S2 = minimum ★

{1, 6, 5} {11}

12 11

12 - 11 = 1

Minimum

(This problem is close to equal partition problem)

S1 S2

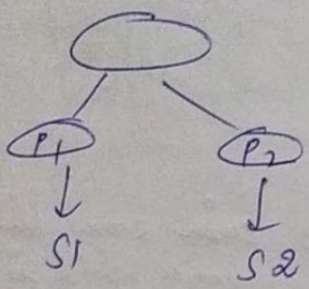
clueless

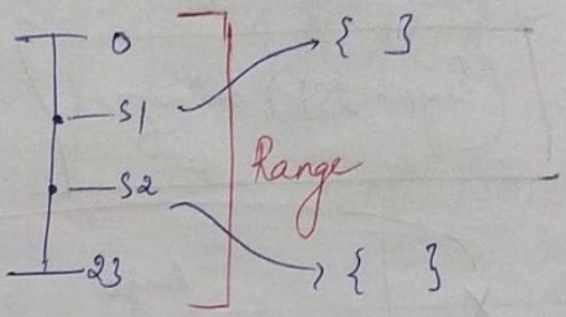
So, we try to find the range //

| 1 | 6 | 11 | 5 |
|---|---|---|---|

$P_1$ → $\{\ \}$ → $S_1 = 0$

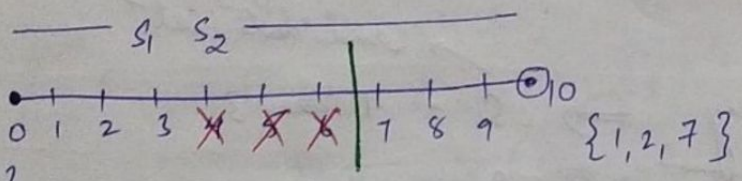
$P_2$ → $\{1, 6, 11, 5\}$ → $S_2 = 23$




ase : | 1 | 2 | 7 |

$10 - 0 \Rightarrow$ max




$\{1, 2, 7\}$

$\{\ \}$

↓

S1/S2 candidates

$\{0, 1, 2, 3, 7, 8, 9, 10\}$

$S1 + S2 = Range$

S1 ✓

S2 aaram se nikal jaayega (Range − S1)

(S1 − S2) minimize

$S_2 - S_1$

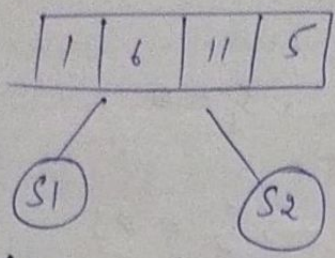
Range − S1 − S1

| Range − 2S1 | minimize

* S1 ko chhota maana hai, toh Range/2 = 5

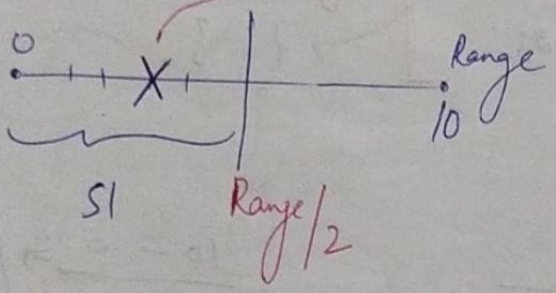
5 se zyada S1 ko nhi hone dena hai

Sum up




$|S2 - S1|$ minimize

(Range - 2S1) minimize




aisa sum bhi hoga which is not possible

2n [1|2|7] array ⟹ out of Candidate set

S1 = 1 2 3

| Range − 2S1 | |
|---|---|
| S1 = 1 | 10 − 2 = 8 |
| S1 = 2 | 10 − 4 = 6 |
| S1 = 3 | 10 − 6 = 4 |

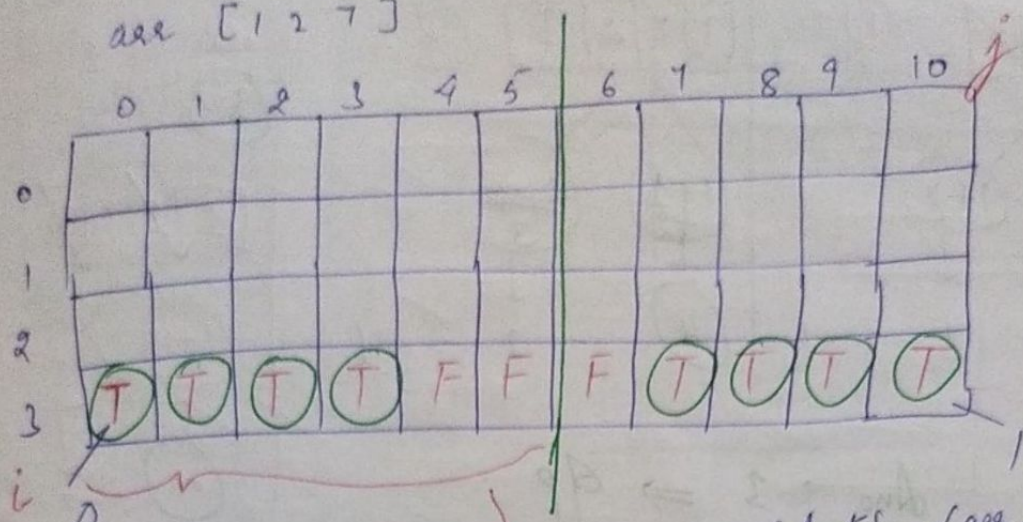
minimum answer hoga

arr : [1 2 7]
4
↓
False

→ Ye toh Subset Sum problem ki tarh hai

→ isi matrix ki Last Row nikaalni hai

Socho!!!




Range

SubsetSum (arr, sum) {

vector me last wale daalo (T) wali

SI → smaller

v: | 0 | 1 | 2 | 3 |

3 candidates aa gye

```
int min = Integer.MAX_VALUE;
for (int i=0; i < v.size(); i++) {
    min = Math.min(min, Range - 2*v[i]);
}
return min;
```

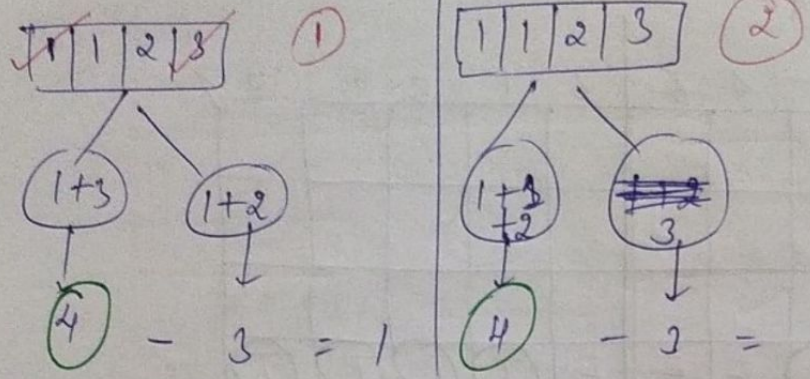
# Count the # of subset with given difference

arr[]: | 1 | 1 | 2 | 3 |

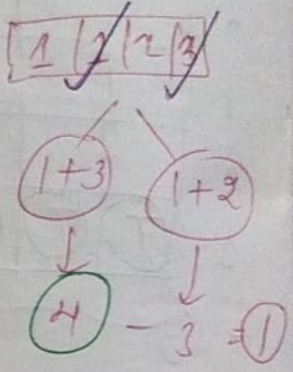
Diff : 1

Input

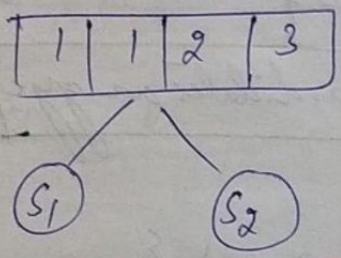
o/p = 3




kyunki 2 alag alg 1 give 3 hain !




Ans : 3 ⇒ qp




→ Sum(S1) − Sum(S2) = diff — (1)

→ Sum(S1) + Sum(S2) = Sum(array) — (2)

(1) + (2)

2 Sum(S1) = diff + Sum(arr)

$$Sum(S1) = \frac{diff + Sum(arr)}{2}$$

$$Sum(S1) = \frac{1+7}{2} = \frac{8}{2} = 4$$

Sum(S1) = 4

) ye toh count subset sum jaisa hi hai !!!

count # of subset with given diff → Reduced to → count of subset sum

$$Sum(S1) = \frac{diff + sum(arr)}{2}$$

# Target Sum

arr : | 1 | 1 | 2 | 3 |

sum : 1

op : 3

+/− signs assign karne hain

Ques says

+1 −1 −2 +3

−1 +1 −2 +3

+1 +1 +2 −3

## BASICALLY

This ques is exactly the same as —

Count the # of subset with given difference

+1 −1 −2 +3

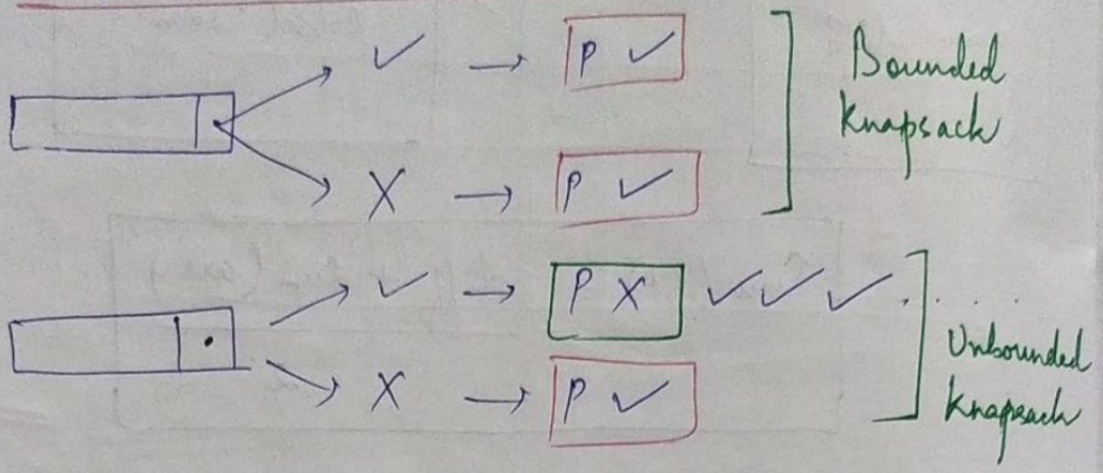
+1 +3 | −1 −2

(1, 3) − (1, 2)

$S1 - S2 = diff$

eventually kar toh yehi rhe hain

# UNBOUNDED KNAPSACK




## Code

### Bounded Knapsack

```
if (wt[i-1] <= j){
  t[i][j] = Math.max(
      val[i-1] + t[i-1][j-wt[i-1]],
      t[i-1][j]);
}
else {
   t[i] = t[i-1][j];
}
```

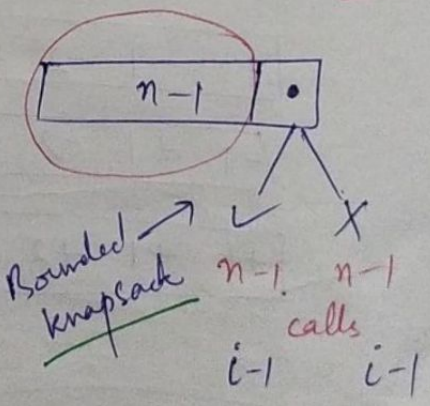
Code diff (Unbounded)

```
t[i]
```

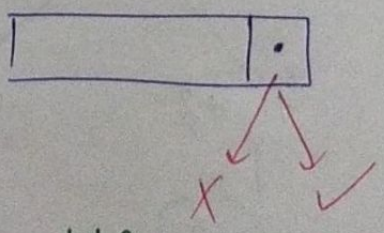
### Unbounded Knapsack

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | | | | | |
| 0 | | | | | |
| 0 | | | | | |

Initialization → Same




But in unbounded




because ye kitni bhi baar liya jaa sakta hai

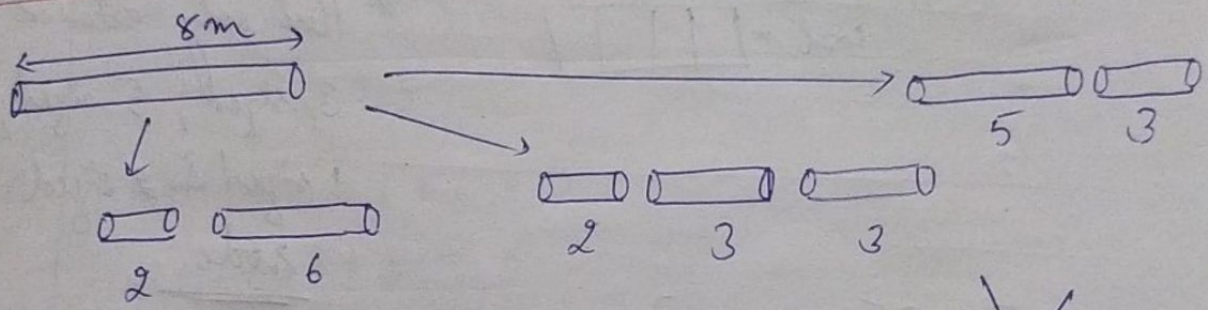
# Rod Cutting Problem

| Length[] : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Price[] : | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 |

N : 8

→ 1 to N

Input




5 + 17 = (22) → maximize

profit

~~Rotation~~

## Matching

wt[] → length []

val[] → price []

W → N

0-1 knapsack (Bounded)

Unbounded knapsack ] → multiple times le lena item ko

flexibility

```
if(length[i-1] <= j){
t[i][j] = Math.max(price[i-1] + t[i][j - length[i-1]],
                   t[i-1][j])
}
else {
 t[i][j] = t[i-1][j];
}
```

* jaroori nhi hai length array given ho
But obvious to take → 1 to N

* Sometimes they can give length array as not 1 to N (1,2,3,4,5,6,7,8)
(if N=8)

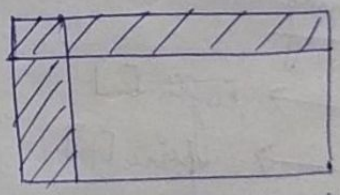
length = | 3 | 5 | 1 | 2 | → size (of array here)

cost = | | | | |

→ Here only allowed 3 sized, 5 sized, 1 sized & 2 sized rods

* t[n+1][length+1]
t[n+1][N+1] length → N (in most cases)

* otherwise
t[size+1][N+1]



0 se initialization
(Why 0 → same reason !!)

# Coin Change I - MAX^m No. OF WAYS

Unbounded knapsack

coins[] : | 1 | 2 | 3 |

Sum : 5

- choice combination

| 1 | 2 | 3 |
|---|---|---|
| x | ✓ | ✓ |

5 ways

- 2 + 3 = 5
- 1 + 2 + 2 = 5
- 1 + 1 + 3 = 5
- 1 + 1 + 1 + 1 + 1 = 5
- 1 + 1 + 1 + 2 = 5

Matching

wt[] → coins[]

W → Sum

agar max^m no. of ways puche ⇒ ~~max~~ + lagao

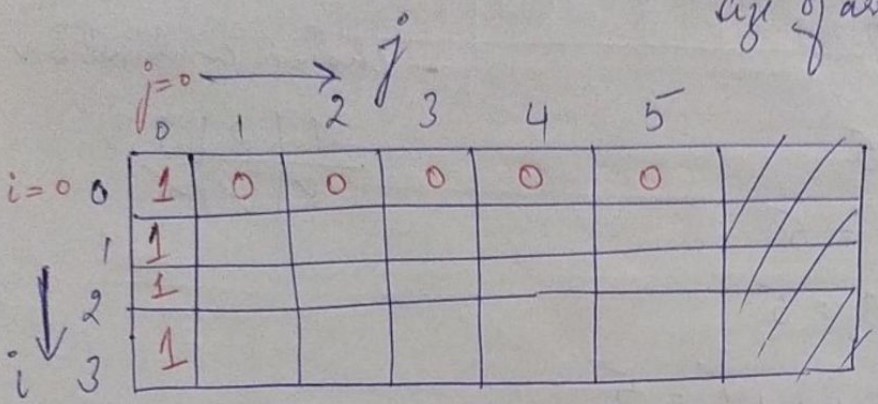
```
if (wt[i-1] <= j) {
    t[i][j] = Math.max(~~val[i-1]~~ + t[i-1][j-wt[i-1]],
                       t[i-1][j])
}
else {
    t[i][j] = t[i-1][j];
}
```

(wt → coins; unbounded → t[i]; coins)

## Initialization

t[n+1][W+1] ⟶ t[n+1][sum+1]

↓ size of array




horizontal wala

arr[] = { }

Sum = 0

arr[] ;

sum = 1

arr[] ;

Sum = 2.

vertical wala

arr : [1]

Sum : 0 → { } ✓

↓ Sum = 0

arr : [1 2]

sum : 0 → Yes ✓

{ , }

↙

Sum = 0

# Coin Change II : Minimum # of coins

coins[] = {1, 2, 3}

Sum : 5

Op : 2

$2 + 3 = 5 \rightarrow (2)$

$1 + 1 + 3 = 5 \rightarrow 3$

$1 + 2 + 2 = 5 \rightarrow 3$

$1 + 1 + 1 + 2 = 5 \rightarrow 4$

$1 + 1 + 1 + 1 + 1 = 5 \rightarrow 5$

## Initialization

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

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

$t[4][6]$

arr :
Sum : 1

$\infty$

INT_MAX - 1

arr :
Sum : 2.

$\infty$

INT_MAX - 1

arr : [1]
Sum = 0

0

arr : [1|2]
Sum = 0

0

```
if( coins[i-1] <= j )
    t[i][j] = Math.min( t[i][j-coins[i-1]] + 1
                        t[i-1][j] );
else
    t[i][j] = t[i-1][j];
```

ek coin add kiya hoga +1

isi +1 ke wajah se we would get INT_MAX - 1

kyunki agar value INT_MAX + 1 ho jaati toh overflow ho jaata
∴ cannot be stored in 'int' datatype

# LONGEST COMMON SUBSEQUENCE

I/P: X : a b c d g h
Y : a b e d f h r

O/P: 4 (abdh)

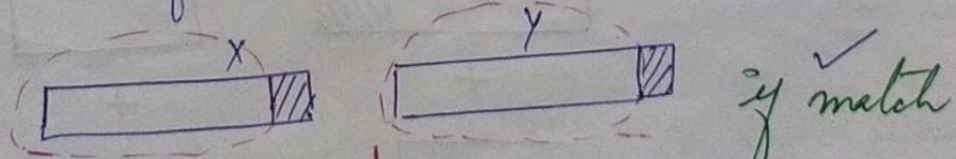
Length of LCS.

| Subsequence | V/s | Substring |
|---|---|---|
| Yahan chod chod ke bhi le sakte hain (discontinous) | | Yahan continous honi chahiye (continous) |

Base Cond$^n$ — $n \to 0$, $m \to 0$

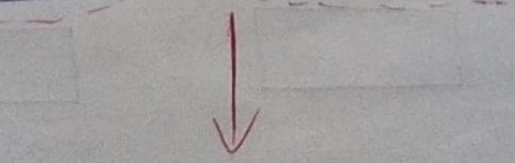
```
if(n==0 || m==0){
    return 0;
}
```

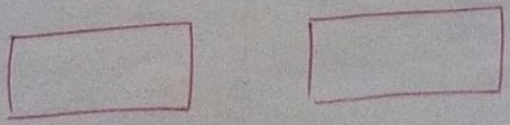
X : a b c d g h
Y : a b e d f h r

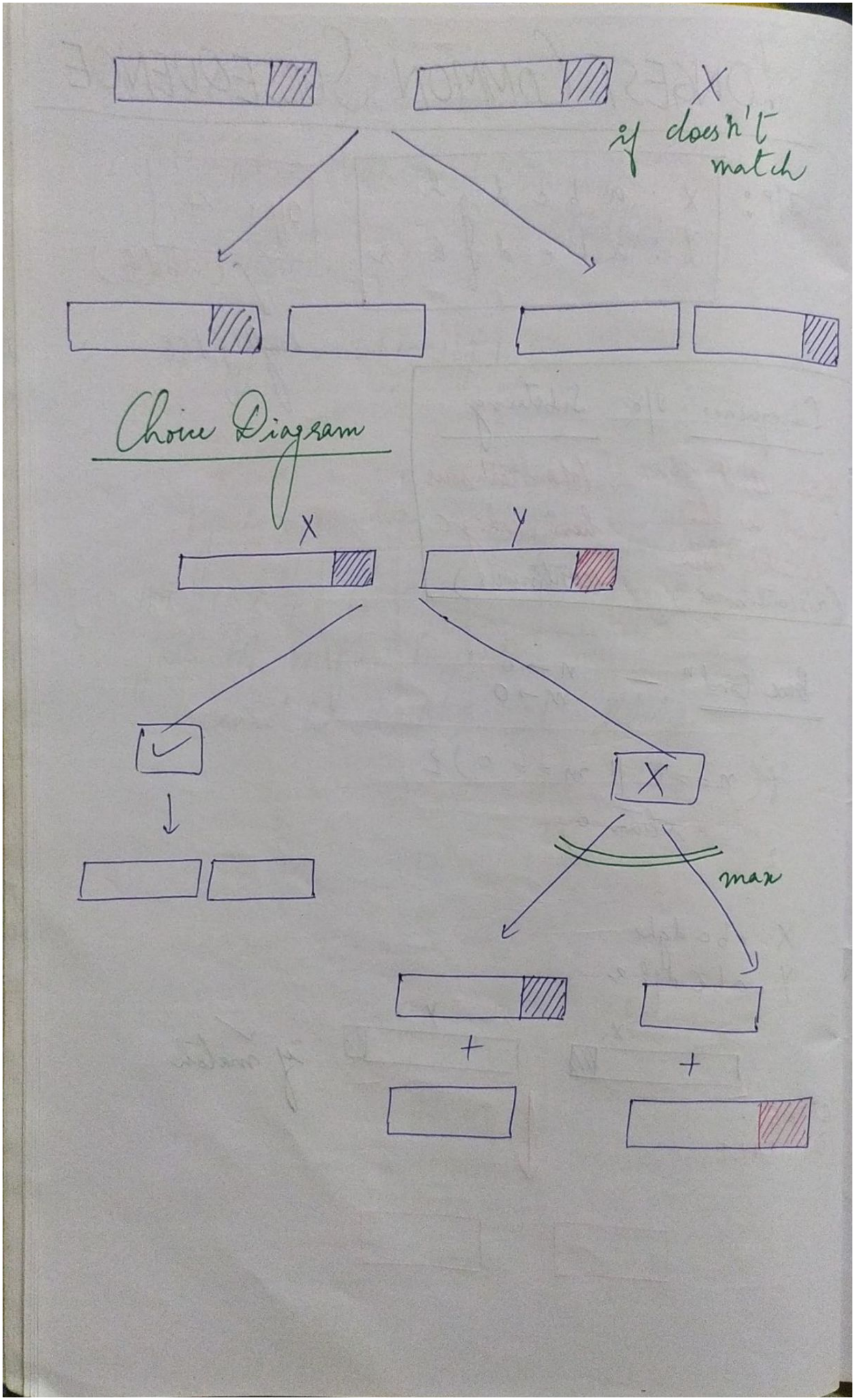



if match ✓

CHOICE DIAGRAM






X
if doesn't match
Choice Diagram
X
Y
X
max
+
+

```
int LCS ( String X, String Y, int n, int m) {
```

Base Cond^n

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

Choice Diagram

```
if ( X.charAt(n-1) == Y.charAt(n-1) )
    return 1 + LCS ( X, Y, n-1, m-1);
```

1 → 1 char same matched

```
else
    return Math.max (LCS (X, Y, n, m-1),
                     LCS (X, Y, n-1, m))
```

```
}
```

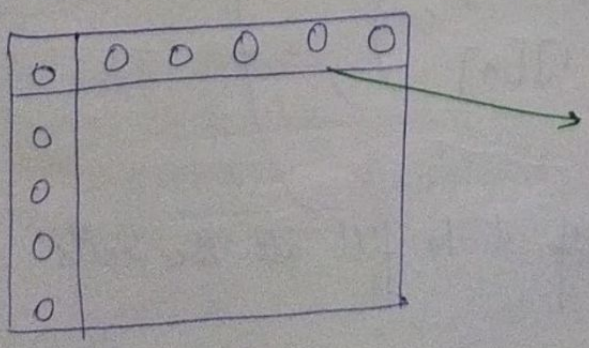
## Memoization

* Check constraints — Given $\Rightarrow X, Y < 10^3$

```
int t[1001][1001]
```

## DP (Tabulation Method)

* n, m → changing variables
* Base Cond^n in recursive code → initialization in table



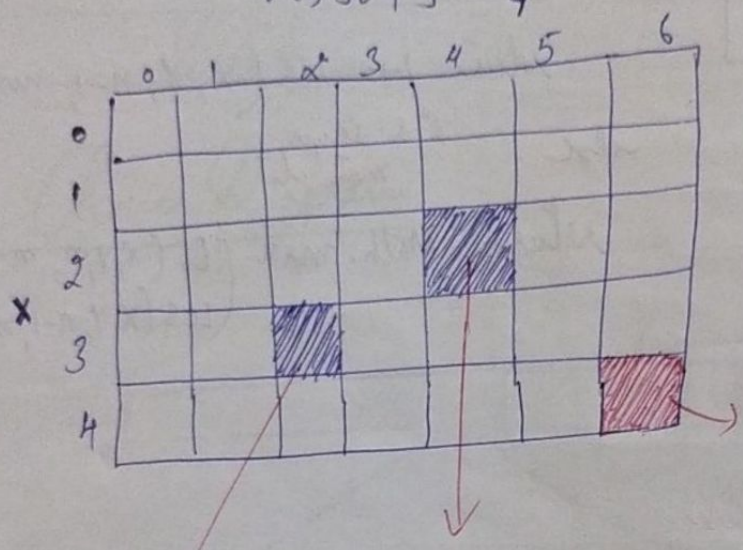
Eg. X: abcf → m = 4

Y: abcdaf → n = 6

t[m+1][n+1]

t[4+1][6+1]

t[5][7]




blocks are subproblems

ANSWER
t[m][n]

X: abc
Y: ab
LCS value
t[3][2]

X: ab
Y: abcd
LCS = 2 That value gets stored in the cell t[2][4]

```
if(X[m-1] == Y[n-1])
t[m][n] = 1 + t[m-1][n-1];
else
return max( t[m][n-1] , t[m-1][n] );
```

Now loop it to fill all the cells

```
int t[m+1][n+1]
for (int i=0; i<m+1; i++)
    for(int j=0; j<n+1; j++)
        if (i==0 || j==0)
            t[i][j] = 0

for (int i=1; i<m+1; i++){
    for(int j=1; j<n+1; j++){
        if (x[i-1] == Y[j-1])
            t[i][j] = 1 + t[i-1][j-1]

        else
            t[i][j] = max( t[i-1][j] , t[i][j-1] )
    }
}
return t[m][n]
```

## Longest Common Substring

X: a b c d e

Y: a b f c e

→ ab
→ c
→ e

} Longest → (ab) Longest Common Substring

2 // Ans

* Variation of LCS
* Initialization → with 0
* if ( x[i-1] == y[j-1] )

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

  else

  t[i][j] = 0;

## Print LCS b/w 2 strings

| x : a c b c f |
|---|
| y : a b c d a f |

i/p

| a b c f |
|---|

o/p

x : a c b c f → m (5)

y : a b c d a f → n (6)

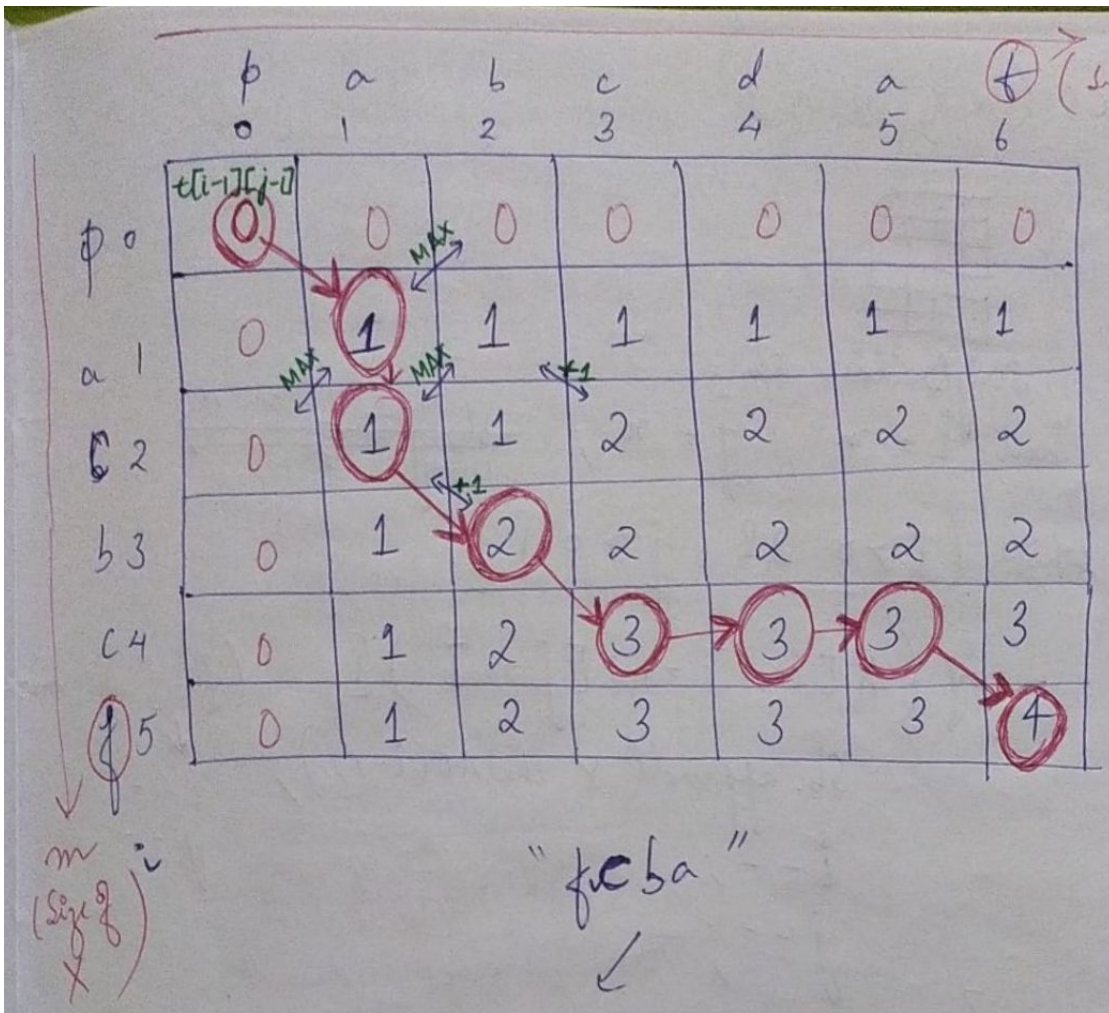
LCS t[m+1][n+1] = t[6][7]

## How to create table ?

| a | b |
|---|---|
| c | d |

if same → d = a+1

if not → d = max (b, c)




"fcba"

↙

reverse

"abcf" → Answer

* agar equal mile to letter ko add kate rahungi and then age badhungi

Otherwise

* max towards the maximum

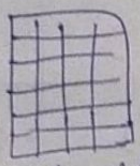
** if if same mile the →

$i, j \rightarrow i--, j--$

nhi toh

$i, j \rightarrow \max\begin{pmatrix}(i-1, j)\\(i, j-1)\end{pmatrix}$

LCS ( X, Y, m, n )

t 

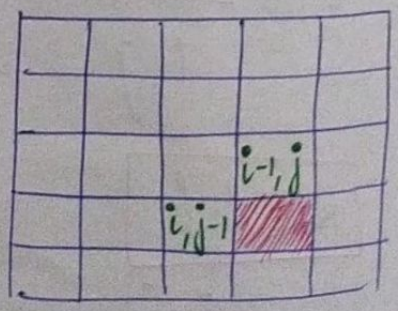
StringBuilder sb = "" ;

int i = m , j = n ; ~~String s = "";~~

```
while ( i > 0 && j > 0 ) {
    if ( X[i-1] == Y[j-1] ) {
        sb.append( X.charAt(i-1) );
        i--;
        j--;
    }
    else {
        if ( t[i][j-1] > t[i-1][j]
            j--;
        else
            i--;
    }
}
sb.reverse();
return sb.toString();
```

(f, f) ↓ i-- j--

(c, a) not equal
① i-1, j
② i, j-1




o/p → abcf

# Shortest Common Supersequence

Eg. a: "geek"
b: "eke"

↓ merge

→ geekeke

→ geeke → shortest supersequence

Eg. X : A G G T A B
Y : G X T X A Y B

i/p

A G G X T X A Y B

Shortest Supersequence

o/p : 9

A G G X T X A Y B

Common

G T A B → Write Only Once

↳ Yeh hai kya? LCS!!!

Worst Case : X.length() + Y.length()
6 + 7 = 13

Common part → G T A B → LCS

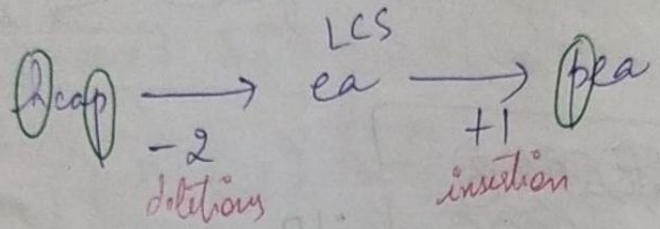
13 − length("GTAB") = 13 − 4 = 9

# Minimum # of insertion & deletion

x: heap
y: pea

o/p: Ins: 1
Del: 2.

~~heap~~




$$\#\text{ of del} = X.\text{length}() - \text{length}(LCS)$$
$$\#\text{ of ins} = Y.\text{length}() - \text{length}(LCS)$$

# Longest Palindromic Subsequence

x: agbcba

LPS.

Matching algorithm

LCS [a: b:] LCS o/p int

LPS [a:] LPS o/p int

LCS laga sakte h.
we need 2 strings usko liye.

2/3 matching

X: agbcba

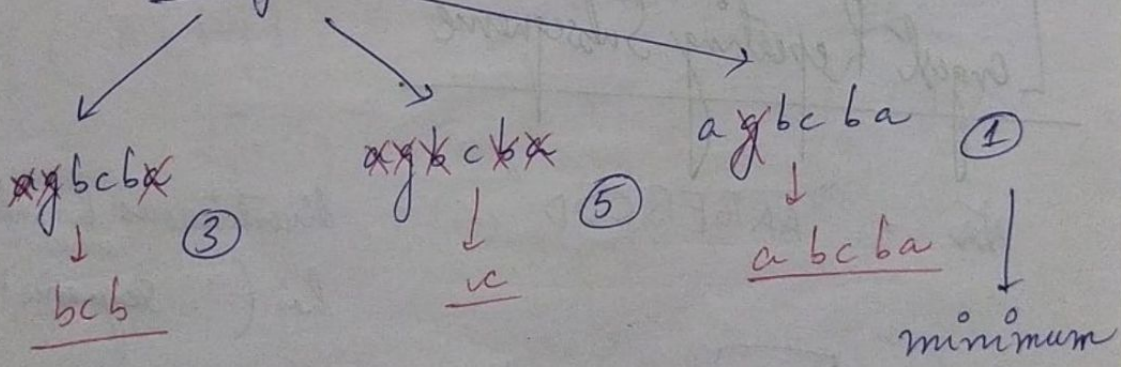
reverse X: abcbga

LCS of above two strings ⇒ abcba

length = 5 Ans

## Minimum # of deletion in a string to make it a palindrome

X: agbcba




↑ Length of LPS ∝ 1 / No. of deletion ↓

LPS (X) → LCS (X, reverse (X))

Min^m # of deletions ⇒ X.length() − LPS

6 − 5 = 1 Ans//

# Printing Shortest Common Supersequence

@ a b c f
@ b c d a f

Same (hence write only once)

Code variation
* else cond^n badlegi (refer Print LCS)
* 2 while loop add hoge

ac b c d a f

GITHUB → Sol^n tested

ac
" "
LCS → " " empty

SCS
"ac"

combine

# Longest Repeating Subsequence

str = "AABEBCDD"

discontinuous ho sakta hai (∵ sequence!!!)

ABD → 2 X times
AD
AB

longest

o/p → 3

LCS ( A A B E B C D D
A A B E B C D D
0 1 2 3 4 5 6 7 )

↓

AABEBCDD

0 1 2 3 4 5 6 7
A A B E B C D D

→ AABBDD

Once

a = S
b = S

Same String in a, b

E → 3
C → 5

i == j ✗

A → 0, 1
A → 0, 1

C → 5 E → 3
C → 5 E → 3

B → 2, 4
B → 2, 4

i ≠ j Imp!!!

```
if ( a[i-1] == b[j-1] && i != j ) {
    t[i][j] = 1 + t[i-1][j-1];
}
else {
    t[i][j] = max( t[i-1][j], t[i][j-1] );
```

## Sequence Pattern Matching

X: "AXY"
Y: "ADXCPY"

O/P: True

O/P: boolean T/F

LCS [a, b] Seabsq int

[a, b] Subsq boolean

I/P Q O/P

2/3 matching

LCS se ho sakta hai!!

a : AXY
b : ADXCPY

LCS : AXY

a : AXYZ
b : ADXCPY

LCS : AXY

if (LCS == a.length())
return true;

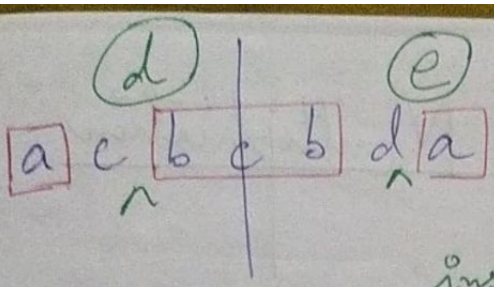
else
return false;

LCS → 0 to min(m, n)

Minimum # of insertions in a string to make it a palindrome

I/P : "aebcbda"

adebcbeda → palindrome




insert d, e to make it a palindrome

Imp!!

**# of insertions = # of deletions**

Socha kaise?

I/p

# of deletions [a] → min # → o/p int

# of insertions [a] → min # → int

(3/3 matching)

means # of deletions socho & then think # of ins. ko kaise relate kiya jaaye.

# Matrix Chain Multiplication

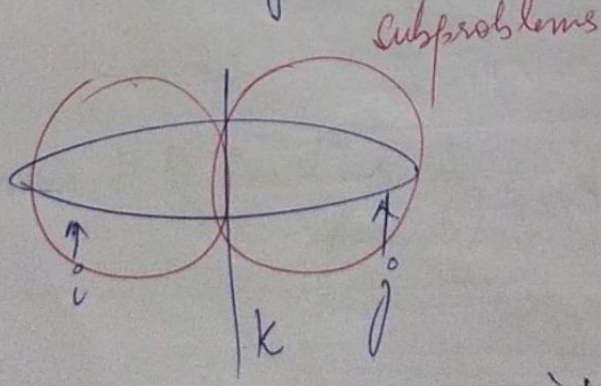
1) MCM
2) Printing MCM
3) Evaluate Expr$^{n}$ to True / Boolean Parenthesization
4) Min / max value of an expression
5) Palindrome Partitioning
6) Scramble string
7) Egg Dropping Problem.

Nutanix → MCM, variations.

base 22 LPA

**Identification + Formal**

MCM < string / array




'k' se break krne ki feel aa rhi hai

temp ans $i - j$ temp ans

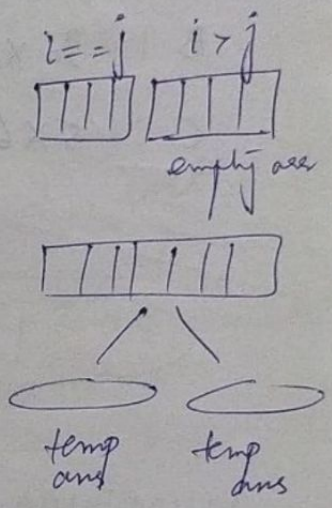
$i, k$ $\quad$ $k+1, j$

Base Cond$^{n}$ : Think of 1st invalid i/p

FORMAT

```
int solve(int arr[], int i, int j){
    if(i>j)
        return 0;          → BC
    for(int k=i; k<j; k++){
        tempans = solve(arr, i, k)
                + solve(arr, k+1, j)
        ans ← func^n(tempans)
    }
    return ans
}
```

→ question dependent




# MCM Recursive

arr[] = {40 20 30 10 30}

$A_1$ $A_2$ $A_3$ $A_4$ ...

$A_1$: 2×5 ; $A_2$: 30×20 ; $A_3$: 30×10

cost [ ] × [ ]

(2×3) (3×6)

⟶ 2×3×6 = 36

$(A_1(A_2\ A_3))A_4$ –

$(A_1\ A_2)(A_3\ A_4)$ –

$A_1\ (\ A_2(A_3 A_4)\ )$ –

min no. of cost → Answer

A : $10 \times 30$

B : $30 \times 5$

C : $5 \times 60$

$(A \cdot B)\ C$

$10 \times 30 \quad 30 \times 5$

$10 \times 30 \times 5 = 1500$

AB C

$(10 \times 5)\ (5 \times 60)$

$10 \times 5 \times 60 = 3000$

cost 4500

temp ans

A(BC)

↳ cost = $30 \times 5 \times 60$

\+

$10 \times 30 \times 60$

cost 27000

temp ans

minimum 4500

min (27000, 4500)

= ~~27000~~ 4500 Ans//

arr[] : 40 20 30 10 30

i: 0 1 2 3 4

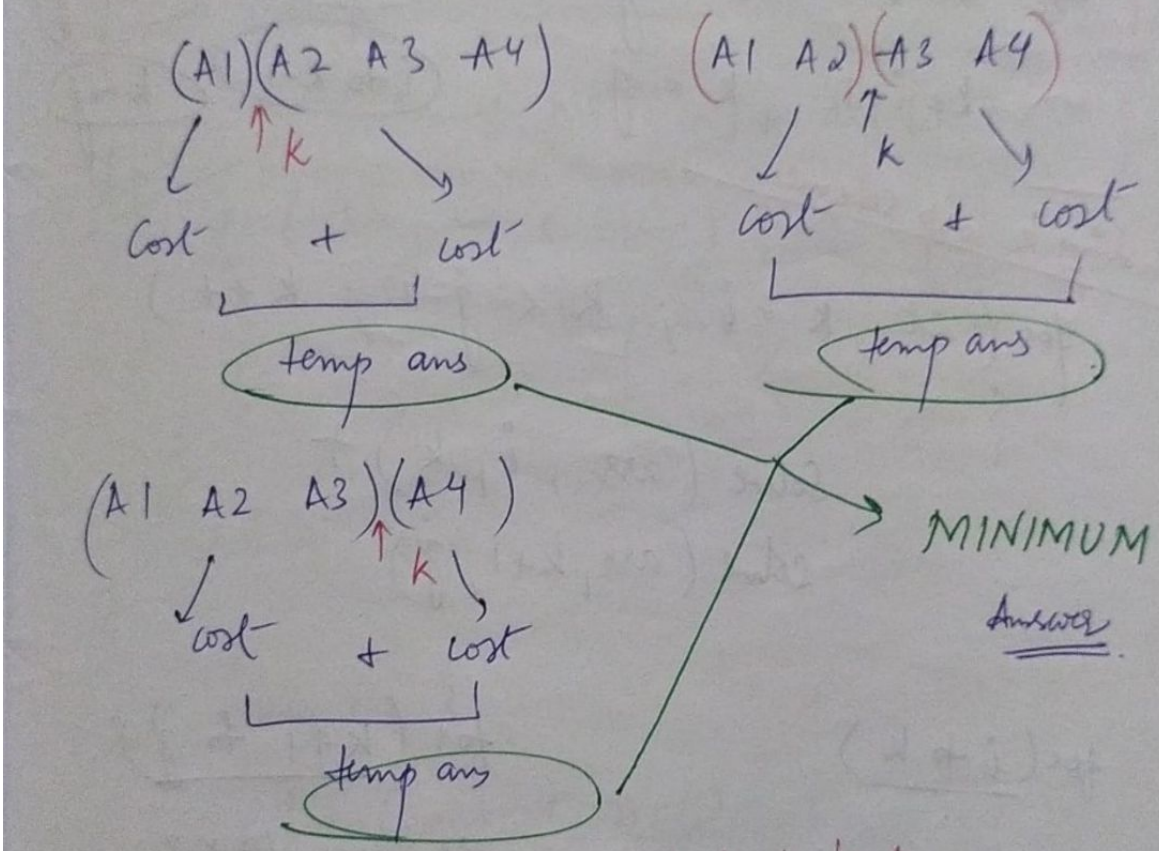
A1 → 40 × 20

A2 → 20 × 30

A3 → 30 × 10

A4 → 10 × 30

$A_i = arr[i-1] * arr[i]$




Step 1: Find i, j

$A_1 = arr[0] * arr[1]$ → i has to be 1; if it is 0 then i-1 = -ve value

∴ i = 1

$A_j = arr[j-1] * arr[j]$

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

$= 10 * 30$

i = 1

j = n − 1

Size of array

Step 2:

Size 0 array ✗

Size 1 array ✗

BC i >= j

Find BC

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

```
if ( i >= j )
    return 0;
```

BASE CONDITION

Step 3:

Schemes

| 40 | 20 | 30 | 10 | 30 |
|---|---|---|---|---|
| | ↓ i | | | ↓ j |

→ k = i  k = j-1  (i to k)  (k+1 to j)

→ k = i+1  k = j  (i to k-1)  (k to j)

Finding k loop scheme

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

for ( i to k )

40 x 20, 20 x 30

Cost { 40 x 20 x 30 }

for ( k+1 to j )

30 x 10, 10 x 30

Cost { 30 x 10 x 30 }

40 x 30 | 30 x 30

Cost { 40 x 30 x 30 }

arr[i-1] x arr[k] x arr[j]

```
int solve (int arr[], int i, int j) {
    if (i >= j)
        return 0;
    int min = Integer.MAX_VALUE;
    for (int k = i; k < j-1; k++){
        int tempans = solve (arr, i, k) + solve (arr,
                                                 k+1, j)
                    + arr[i-1] * arr[k] * arr[j]
```

Step 4 : Finding minimum value

```
        if (tempans < min) {
            min = tempans;
        }
    }
    return min
}
int main () {
    solve (arr, 1, n-1)
}
```

## MCM with Memoization

Change → i, j

t[ ][ ] → On the basis of constraints given

t[1001][1001]

→ Initialize the matrix by -1

→ Check if the value is -1 → if it is, that means the value is not stored yet

→ If it is not -1 → simply return the value (stored)

```
int solve ( int arr[] , int i , int j ) {
    int[][] t = new int[1001][1001];
    // Initialization
    for ( int i=0 ; i < 1001; i++ ) {
        for ( int j=0 ; j < 1001 ; j++ ) {
            t[i][j] = -1;
        }
    }
    if (i >= j) {
        return 0;
    }

    if ( t[i][j] != -1 ) {
        return t[i][j];
    }

    int min = Integer.MAX_VALUE;
    for ( int k = i ; k<j ; k++ ){

      int temp = solve (arr, i, k) + solve ( arr, k+1, j)
                    + arr[i-1]* arr[k] * arr[j];
```

```
if (temp < min) {
    min = temp;
}
}
t[i][j] = min;
return t[i][j];
}
```

# PALINDROME PARTITIONING

S : nitin  I/P

O/P : 0

already a palindrome

n | iti | n

n  iti  n

worst case

s.length - 1 ⇒ no. of partitions.

↳ MCM format

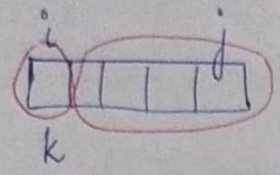
{ "" → empty string
"a" → length 1 string ] return 0;

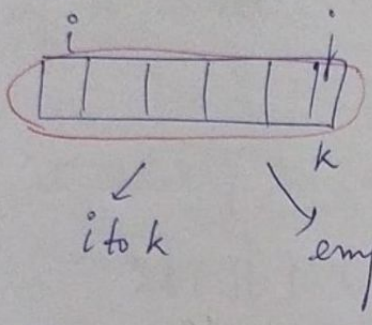
"abcba" → already a palindrome → 0 partitions required. return 0;

BC eg: if s = "nitik"
O/P = 2.

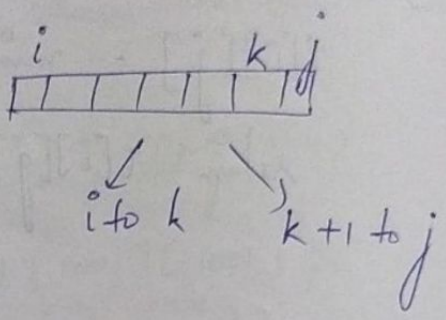
Loop partition kahan karun?

How to think of loop schemes?




X wrong scheme

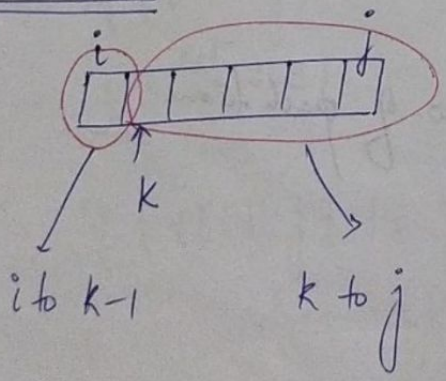


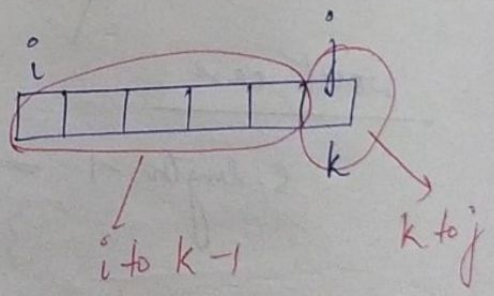

Schemes

1. k = i  k = j-1  (i to k)  (k+1 to j)

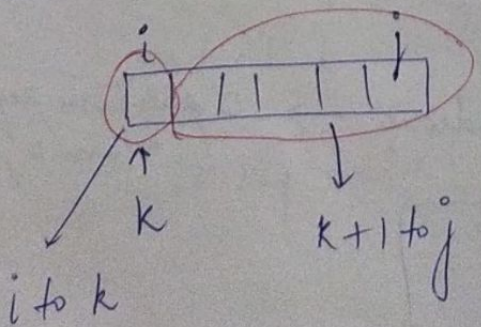
2. k = i+1  k = j  (i to k-1)  (k to j)

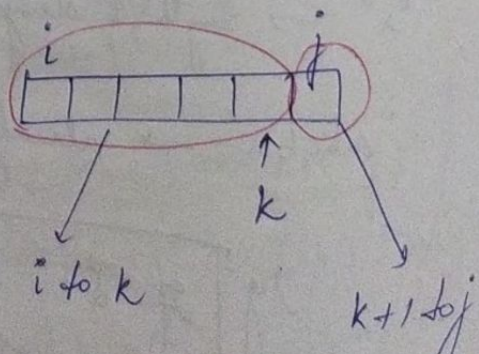
For 2nd scheme




For 1st scheme

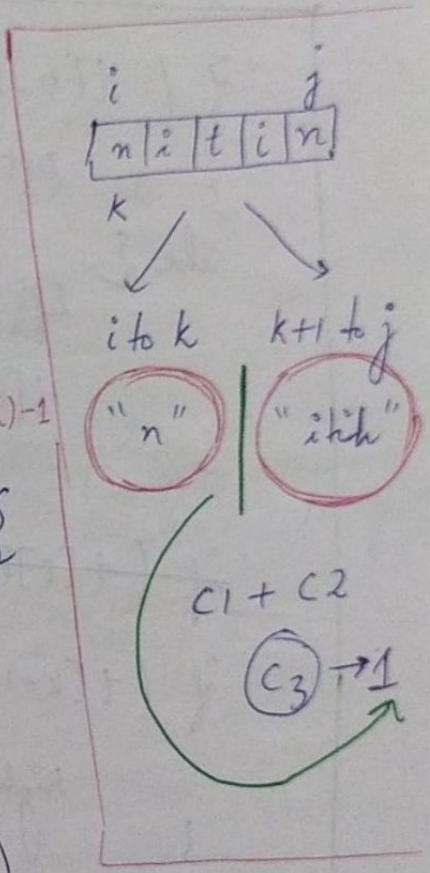



For loop ki koi si bhi scheme choose kelo

Lets go with first scheme

```
int min = Integer.MAX_VALUE;
for(int k = i; k <= j-1; k++){
    int temp = solve(s, i, k) +
               solve(s, k+1, j) + 1
    if(temp < min){
        min = temp
    }
}
return min;
```




# Code –

```
int solve(String s, int i, int j){
    if(i >= j)
        return 0;
    if(isPalindrome(s, i, j) == true)
        return 0;
    int min = Integer.MAX_VALUE;
    for(int k = i; k <= j-1; k++){
        int temp = solve(s, i, k) + solve(s, k+1, j) + 1
        if(temp < min){
            min = temp
        }
    }
    return min;
}
```

## Further Optimization

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

left

right

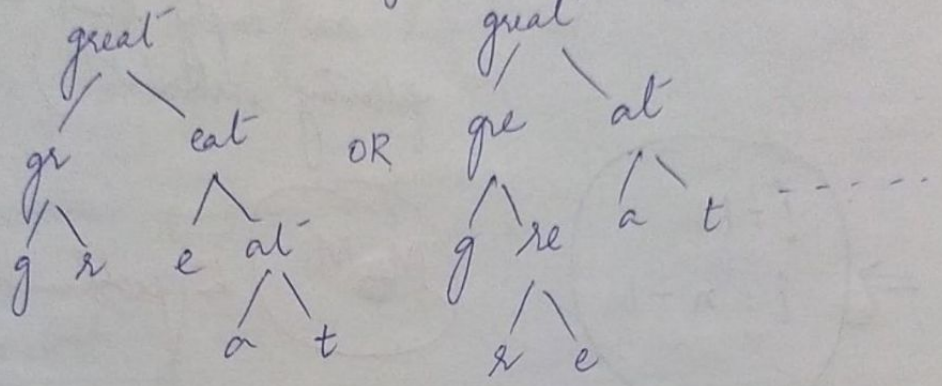
```
if ( t[i][k] != -1 ){
        left = t[i][k];
}
else {
        left = solve(s, i, k);
        t[i][k] = left;
}

~~if ( t[i][k]~~
if (t[k+1][j] != -1 ){
        right = t[k+1][j];
}
else {
        right = solve(s, k+1, j);
        ~~t[i][~~
        t[k+1][j] = right;
}
int temp = 1 + left + right;
```

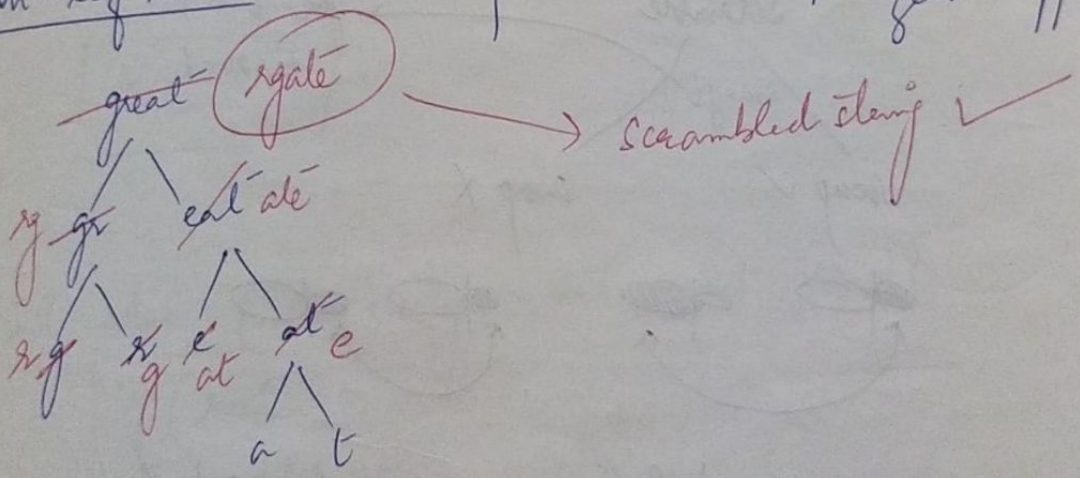
# Scrambled Strings

Constraints

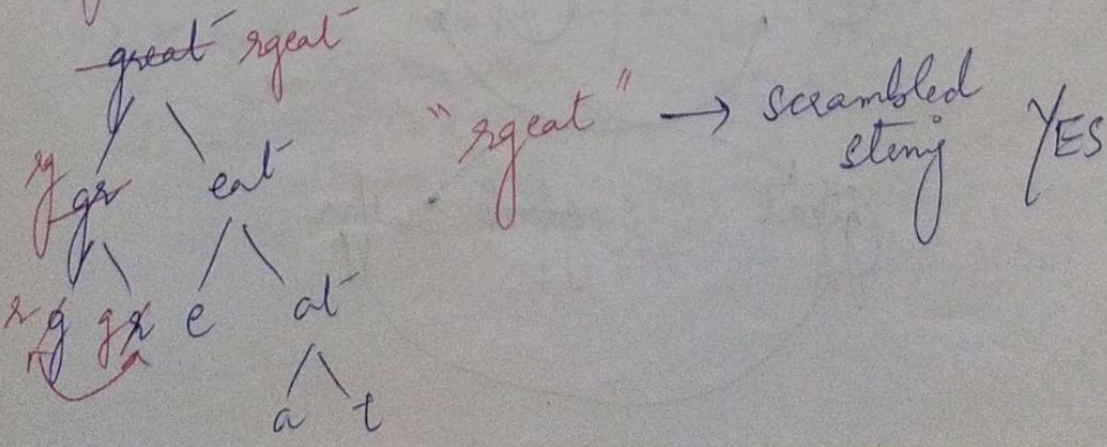
1. Binary Tree banana hai
2. Empty child nhi hona chahiye




Non-leaf nodes : can swap their children | Zero or non zero swapping




great
great
→ Yes, they are scrambled strings
(∵ Zero swapping)



How to break ?

great great great great

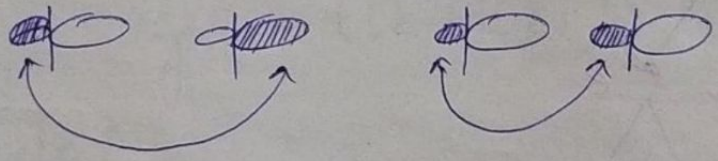
/great → NO (because left child would be an empty string → not following constraints)

⇒ i = 1
i = n - 1

MCM ki problem hai !!!

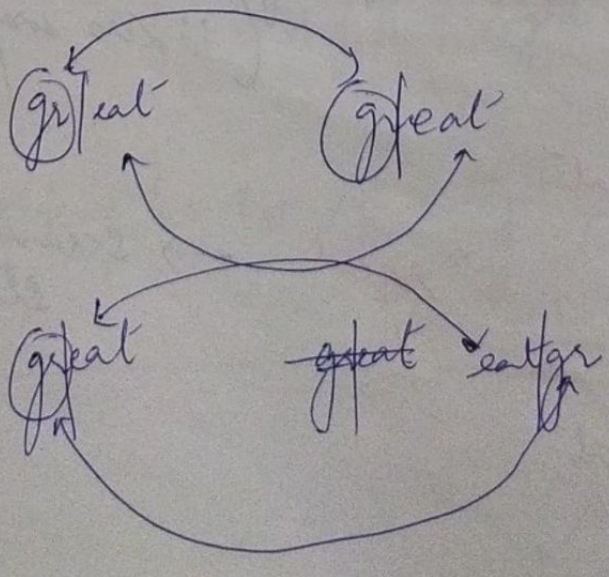
Scramble @any i → for i = 1 to n-1

swap ✓ swap X



a.first → b.last
a.last → b.first

a.first → b.first
a.last → b.last



# DP on Trees

1. General Syntax
2. How DP can be applied to Tree (Identification)
3. Diameter of a Binary Tree
4. Maximum Path Sum from any node to any
5. Maximum Path Sum from leaf to leaf
6. Diameter of N-ary Tree.

```
int funname (//---) {
    Base Cond^n
    HYPOTHESIS
    INDUCTION
}
```

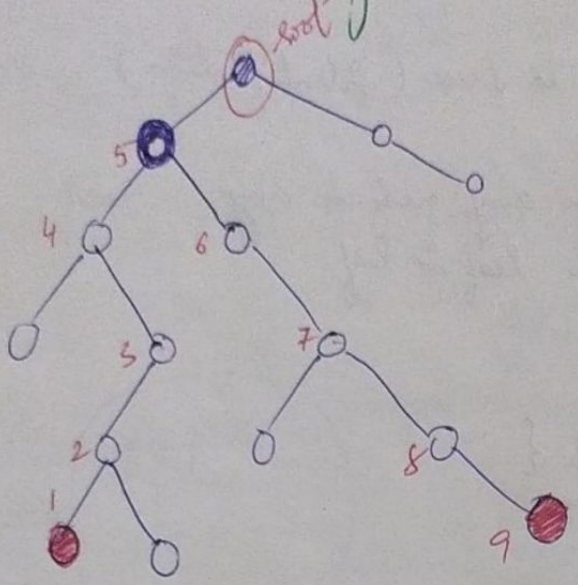
```
int solve (Node<Integer> root, int res) {
    if (root == null)
        return 0;                          BC

    int l = solve (root.left, res)
    int r = solve (root.right, res)        ~~relation~~

    int temp = calculate temp ans ( 1 + max(l, r) )
    int ans  = max ( temp, relation)
    res = max ( res, ans)

    return temp
}
```

# Diameter of a Binary Tree




* Not necessary ki longest path would pass through node.

9

```
int solve (Node<Integer> root, int res) {
    if(root == null) {
        return 0;
    }
    int l = solve(root.left, res);
    int r = solve(root.right, res);
    int temp = max(l, r) + 1
    int ans = max(temp, (1+l+r))
    res = max(res, ans)
    return temp;
}
int main() {
    int res = Integer.MIN_VALUE;
    solve(root, res)
    return res;
}
```