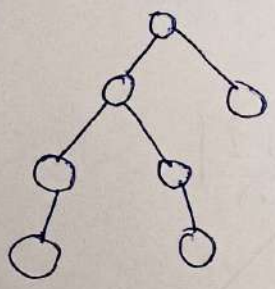
Dynamic Programming (DP)

Parent of DP is Recursion.

→ Recursion को enhance करना ही DP है।

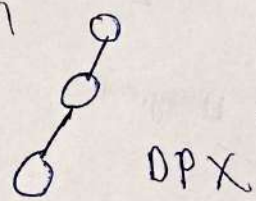
Identification of DP

1. Choice
2. optimal (find maximum, min^m, largest etc.)
3. Recursion calls two times



Parent question

of recursion calls one times then DP नहीं लगेगा।



DPX

1) 0-1 Knapsack (6)
2) Unbounded Knapsack (5)
3) fibonacci
4) LCS (15)
5) LIS (10)
6) Kadane's Algorithm (6)
7) matrix chain multiplication (7)
8) DP ~~and~~ on tree (4)
9) DP on Grid (17)
10) others (5)

0/1 Knapsack

यह DP है

1. choice है
2. optimization है।

IP

wt[ ] :	1	3	4	7
val[ ] :	1	4	3	8
W : 7kg				

n

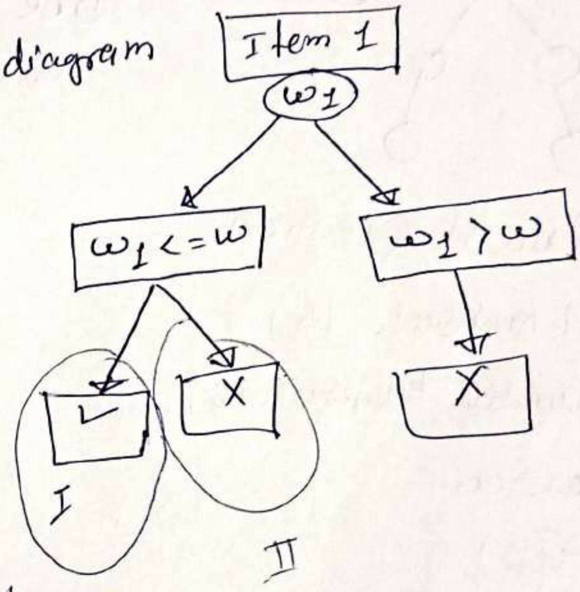
O/P : max^m Profit.

DP: Recursive soln. → memorization → Top-down (DP)

DP = Recursion + Storage.

DP → Recursion → Choice. ~~Diagram~~ ✓

Choice diagram



Base condition → think of the smallest valid IP

IP

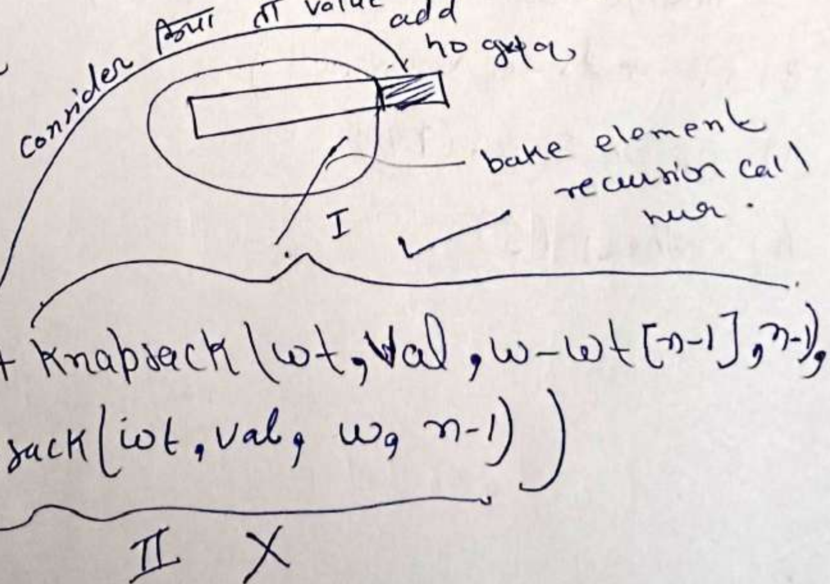
wt[ ]	X X X
val[ ]	
w : 10kg	

n → 0

smallest = 0

code

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



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

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

01 Knapsack TOP Down

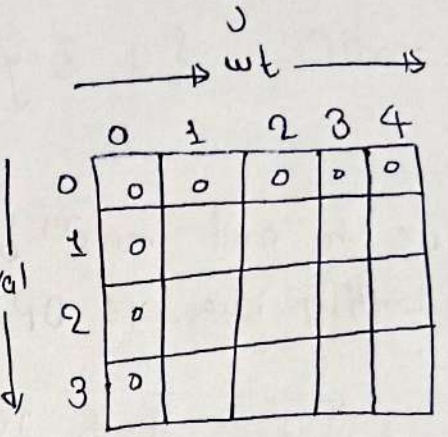
Recursion

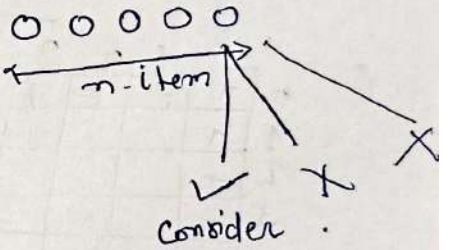
```
if(n==0 || w==0)
return 0;
```

Top-Down

```
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
```

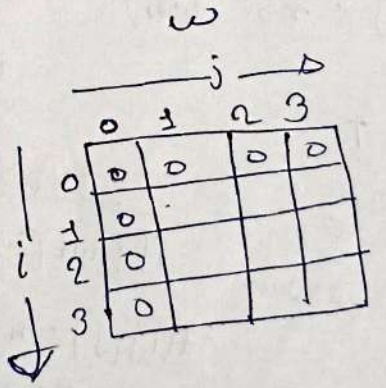
BC





Induction step

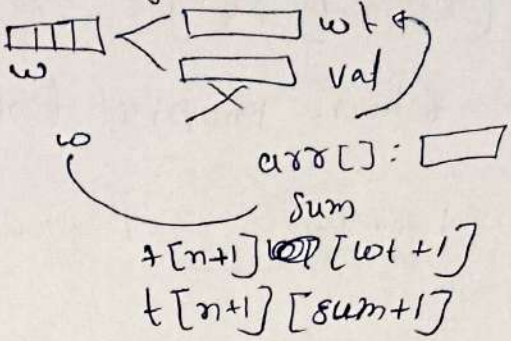
```
if (wt[n-1] <= w){
t[n][W] = max( val[n-1] + t[n-1][w-wt], t[n-1][w] )
}
else
    t[n][w] = t[n-1][W]
```




```
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]
```

Subset sum Problem

matching with 0/1 knapsack.



arr[] = | 2 | 3 | 7 | 8 | 10 |

Sum = 11

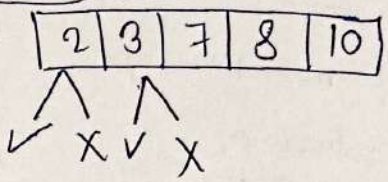
~~Output~~

O/P: True {3, 8}

Sol.

Choice h and max^m bola h so Identify hua ye OP se

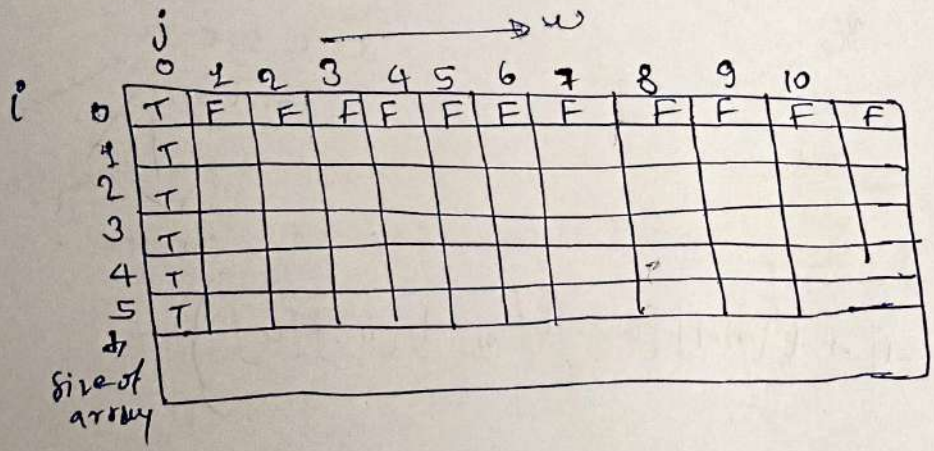
arr[] = | 2 | 3 | 7 | 8 | 10 |



Sum : 11

size of array

t[n+1][w+1]

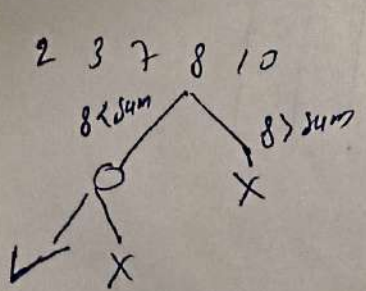



```
for(i=0; i<n+1; i++)
for(j=0; j<wt+1; j++)
  if(i==0){
    false }
  if(j==0){
    True }
```

Sum = 0
arr[] : no item
↳ { } T

arr[] = 2
sum = 0
↳ { } T

arr[] = ka size 0 ho
sum = 1
False



Knapsack

```
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]
```

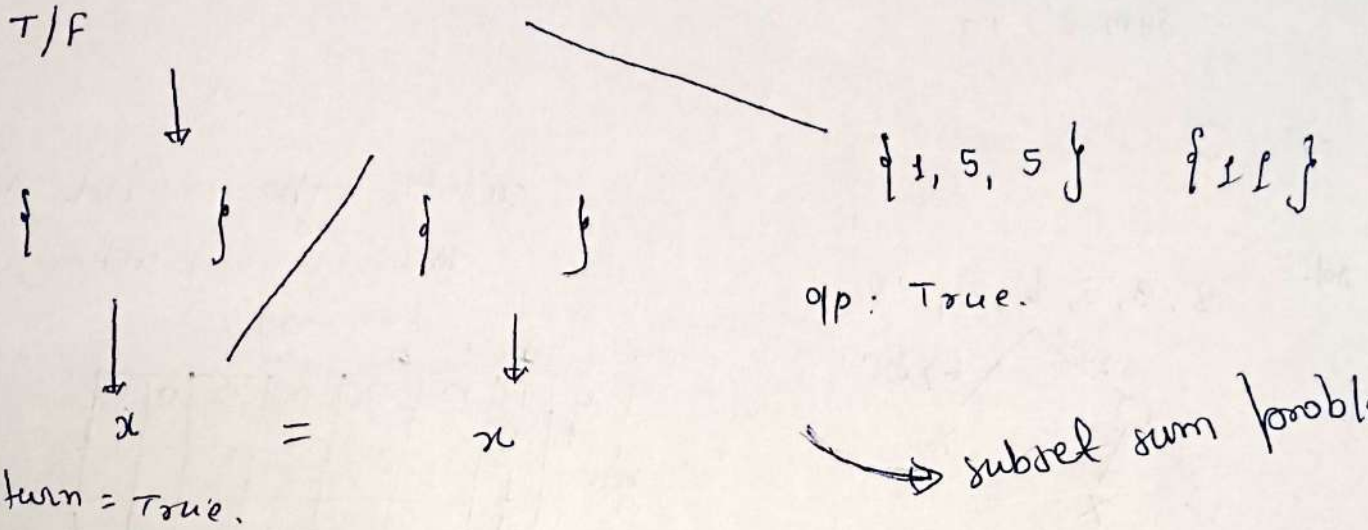
Subset sum

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

return t[n][sum]
```

Equal sum partition Problem.

Input: arr[] = { 1, 5, 11, 5 }, Sum = 11 (not given.)

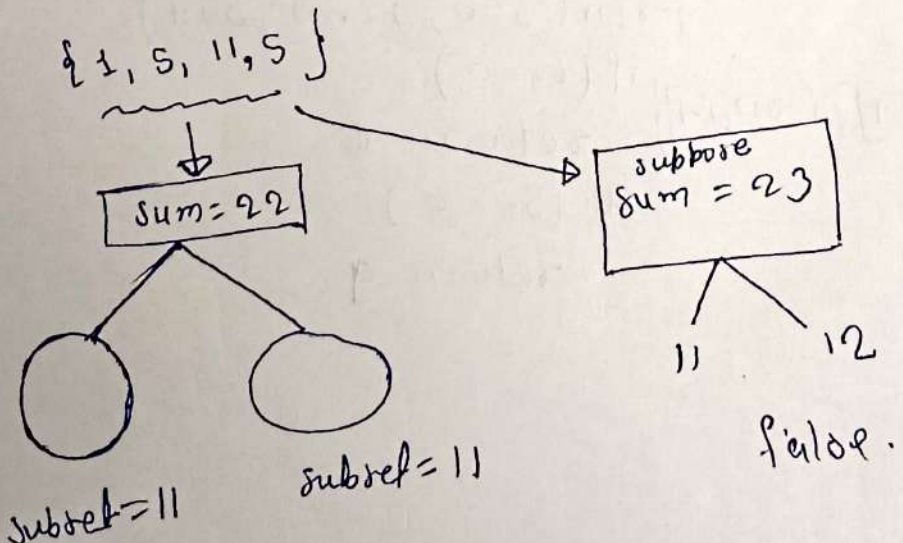
O/P: T/F



Sol.

```
boolean subsetsum (int arr[], int sum, int n){
```

}.




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

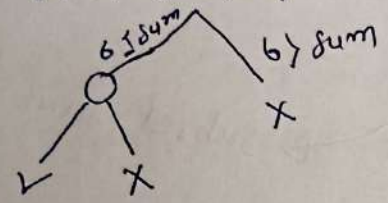
Count of subsets sum with a Given sum.

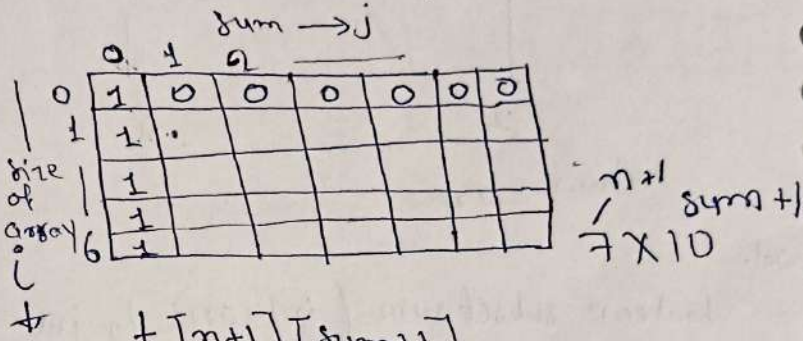
IP: arr[] = {2, 3, 5, 6, 8, 10}
sum = 10

O/P: 3

return type = int क्योंकि value (count) return करना है।

sol. 2, 3, 5, 6, 8, 10





t[n+1][sum+1]

wt ← arr

w ← sum

Similiarity with subset sum.

F → 0 consider नहीं करना है।

T → 1 consider करना है

means count 1

initialization.

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

Code

```
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]
```

return t[n][sum]

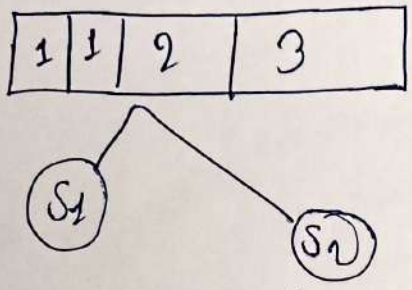
minimum subset sum Dif

Count the number of subset with a given difference

Input: arr[] = {1, 1, 2, 3}

diff : 1

Sol.



$S_1 - S_2 = diff$ — (I)

$S_1 + S_2 = sum$ — (II)

(I) + (II)

$2S_1 = diff + sum$

$S_1 = \frac{diff + sum}{2} = \frac{1+7}{2} = 4$

$sum = 1+1+2+3 = 7$

{1,1,2} - {3} (4 - 3)

{1,1,3} - {2} (4 - 3)

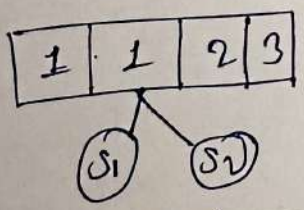
{1,2,3} - {1} (4 - 3)

{1,3} - {1,2} (4 - 3)

1.

int sum = $\frac{diff + sum\ of\ array}{2}$

return count of subset sum(arr, sum).



Count of subset of given diff → Count of subset of given sum.

Target sum

target = abs(target);
if (s < target || (sum + target) % 2 != 0)
return 0;

Ip: arr: | 1 | 1 | 2 | 3 |
sum : 1

→ +/- assign करना

op : 3

+1−1−2+3 = 1
−1+1−2+3 = 1
1+1+2−3 = 1

| + | − | − | + |

| +1 | −1 | −2 | +3 |

+1+3   −1−2

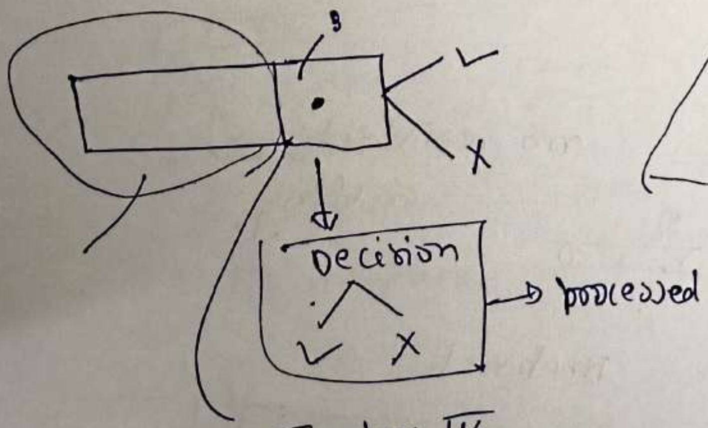
S_1 {1+3} − {1+2} S_2

(target sum) → (Count the subset of given diff.)

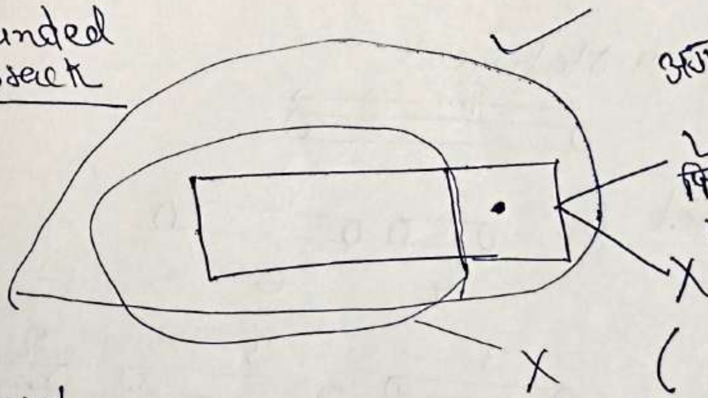
unbounded knapsack

Knapsack
- fractional greedy
- 0 1 knapsack
- Unbounded

0 1 knapsack   **unbounded knapsack**



इस Item पर decision le lia है तो अब बाकी Item पर चले जाएंगे।



अगर पसंद है तो फिर यही से process होगा।

अगर Item पसंद नहीं है तो ये process हो जाएगा।

Related problem

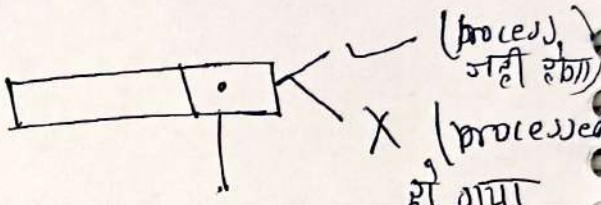
1) Rod cutting
2) coin Change I
3) Coin Change II
4) maximum Ribbon cut.

0 1 Knapsack

```
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]
```

Unbounded knapsack.

```
if (wt[i-1] <= j)
t[i][j] = max (val[i-1]
          + t[i][j-wt[i-1]],
          + t[i-1][j])
else
t[i][j] = t[i-1][j]
```



Rod cutting problem

IP: length[]: 1 2 3 4 5 6 7 8

Price[]: 1 5 8 9 10 17 17 20

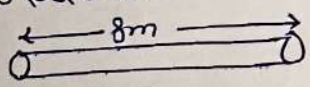
N: 8

O/P

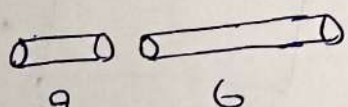
maximize profit.

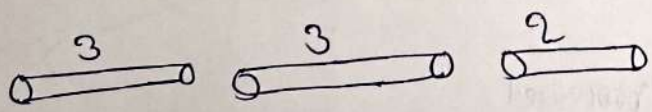
Icecream ~~beer~~ पसंद नहीं है तो एक बार ही offer किया जाएगा यदि पसंद है तो एक-अधिक बार offer होगा।

Problem statement



Target 8:





no restriction for cutting.

matching

length[]:

Price[]:

N:

knapsack

wt[]:

val :

w :

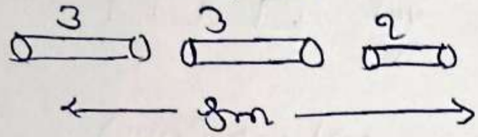
length[] — wt[]

Price[] — val[]

N — w

→ Identification of unbounded Knapsack.

same size piece cut more than 1.



Code variation.

length

if (wt[i-1] <= j) {   price

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

else

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

if (price[i-1] <= j) {

t[i][j] = max(price[i-1] +
t[i][j-length[i-1]],
t[i-1][j])

else

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

Coin Change problem : max^m no. of way.

IP

Coins :	1 | 2 | 3
sum :	5

O/P → max^m no. of way : 5

{1+1+1+1+1} : 5 — ①
{1+2+2} : 5 — ②
{1+1+3} : 5 — ③
{1+1+1+2} : 5 — ④
{2+3}

sol. [1 | 2 | 3]

✓ ✗ choice
ये decide
हो गया knapsack

Identification

1+1+1+2

select more than
so, this unbounded
Knapsack.

matching.

Coins [1 | 2 | 3]
sum = 5

{1+2+2} S1
{1+1+1+3} S2
{1+2+2}
{1+1+1+1+2}

यहाँ भी subset
बन रहे हैं
or count
करना
है

Count subset with given sum.

arr[] : [| |]
sum :

Count subset for given sum

coins

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

repeat हो सकता है

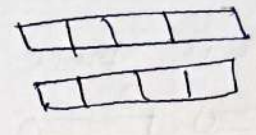
t[i][j] = t[i-1][j-arr[i-1]] + t[i-1][j]

}

else{

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

no. of ways



wt → coins

Val → ∞

w → sum.

if (coins[i-1] <= j)

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

+ t[i-1][j]

}

else

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

Coin change problem : Minimum number of coins.

IP

coins[ ]: 1 | 2 | 3

sum : 5

op : 2 {2, 3}

{1+1+1+1+1} — (5)

{1+1+1+2} — (4)

{1+1+3} — (3)

{1+2+2} — (3)

{2+3} — (2)

01 knapsack memoization.

constraint

$n <= 100$

$w <= 1000$

```
int static t[102][1002]
  memset(t, -1, sizeof(dp))
int knapsack(int wt[], int val[], ~~wt~~ int wl, 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] = t[n-1][w];
```

Largest common subsequence

I/P:

X: a b c d g h

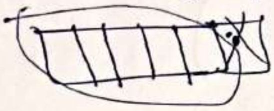
Y: a b e d f h i

O/P: a b d h

abdh. length = 4.

Recursive approach

How thinking — last में sochte h.



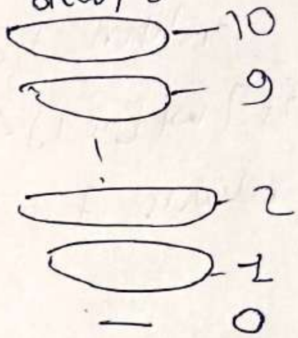
Base condition

X: ____ — n

Y: ____ — m

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

X: abc
Y: ——
LCS: 0



Choice diagram.

eg 1: X: abcdgh
Y: abedfhi

match nhi kr rha है।

तो X से पूरा call करते हैं
और Y से last leave
कर कर के call करते हैं

abcdgh abedfhi

ya

abcdgh abedfh.i

abcdgh abedfhi

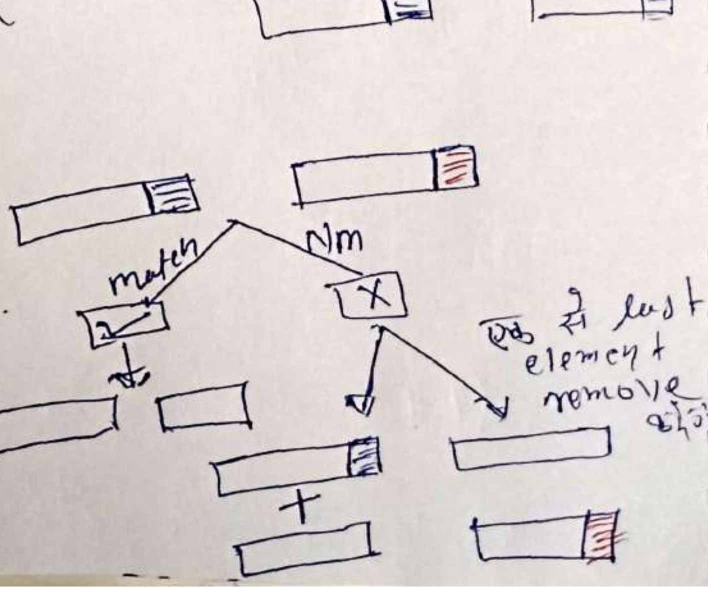
abcdgh + abedfh | abcdg + abedfhi

eg 2: X: abcdgh
Y: abedfh

last वाला match kr rha h.

abcdgh abedfh

dono के last element हटा देंगे।



~~Android development~~

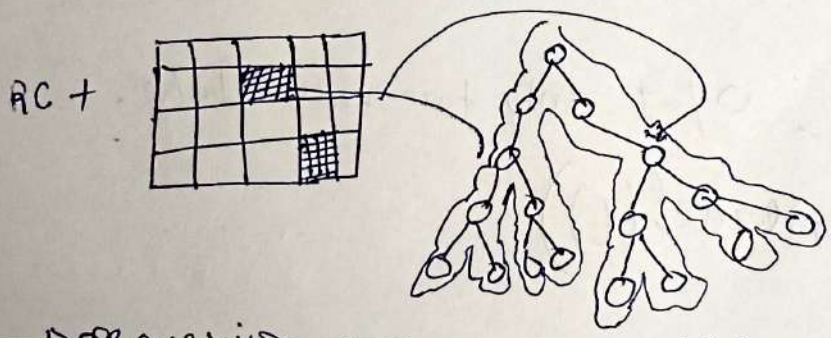
x : [ ] n

y : [ ] m, length.

code

```
int LCS (string x, string y, int n, int m)
{
    Base condition    if(n==0 || m==0)
                        return 0;

    Choice diagram    if(X[n-1] == Y[n-1])
                        return 1 + LCS(X,Y, n-1, m-1)
                      else
                        return max( LCS(X,Y,n-1,m),
                                    LCS(X,Y,n,m-1) );
}
```

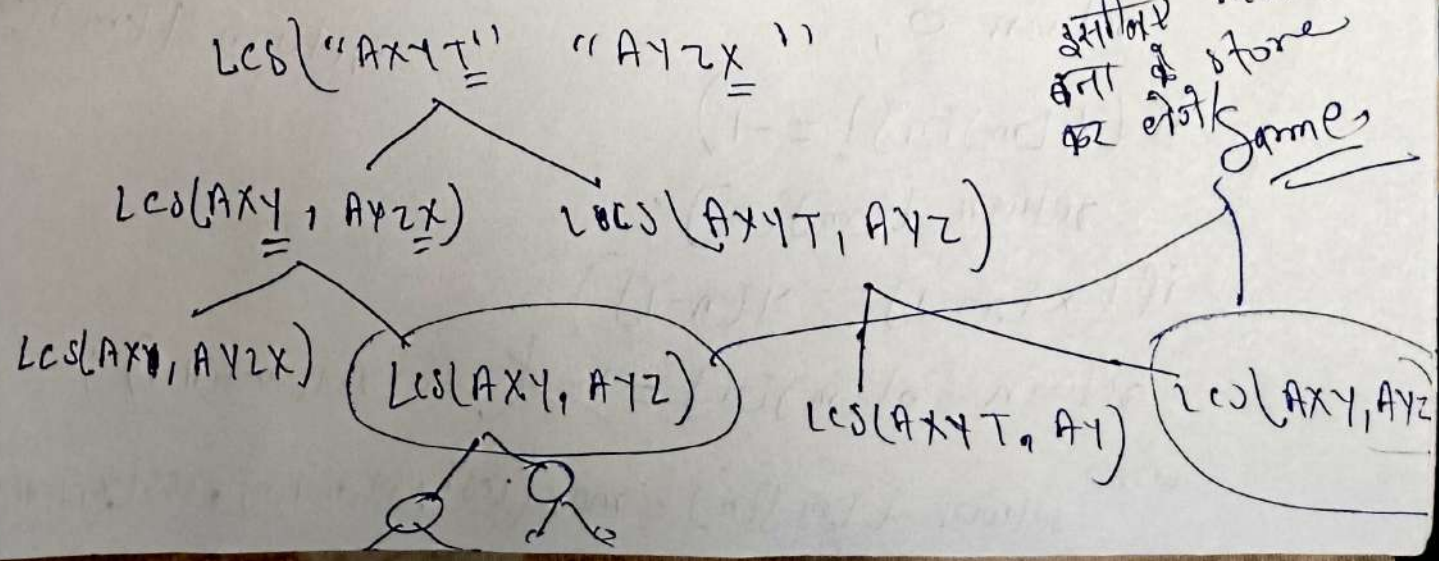
LCS(memoized) Bottom up DP



→ recursion call none se phle check kia jata h. matrix uska value present h. ya nhi.

need of matrix

LCS("AXYT" "AYZX")

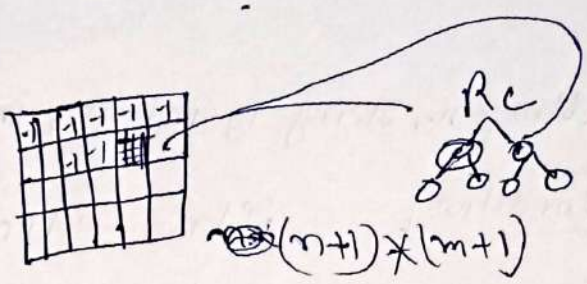


subproblem repeat on letter. ~~sub~~

→ t[n+1][m+1]

→ t[][] → -1

→ ↑ check before recursive call.



code variation.

Global variable ↑ matrix define

```
int static t[1001][1002]
LSC( ){

}
int main(){
    memset(t, -1, sizeof(t))
```

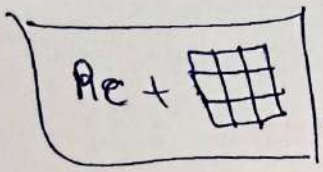
1 ≤ x ≤ 1000
1 ≤ y ≤ 1000

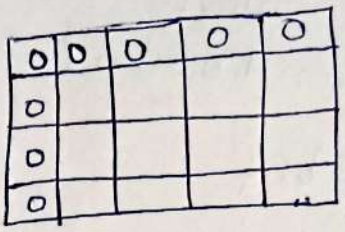
0/-1 only two value take.

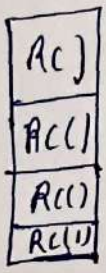
code

```
int static t[1001][1001];
int LCS(int x, int y, int m, int n){
    if(m==0 || n==0)
        return 0;
    if(t[m][n]!=-1)
        return t[m][n];
    if(x[m-1]==y[n-1])
        return t[m][n] = LCS(x,y,m-1,n-1);
    else return t[m][n] = max(LCS(x,y,m-1,n), LCS(x,y,m,n-1));
```

Top down DP longest common subsequence



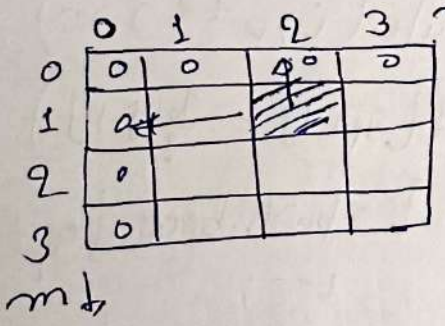




Stack

```
if(x[m-1] == Y[n-1])
return 1+LCS(x,Y,m-1,n-1)
else
return max(LCS(x,Y,m,n-1),
           LCS(x,Y,m-1,n))
```

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



t[1][2]

Printing longest common subsequence.

s_1 : a b c f — m (4)

s_2 : a b c d a f — n (6)

o/p: abcf

$t[m+1][n+1] \rightarrow t[5][7]$

n (sizeof s_2)

		a	b	c	d	a	f
	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
a 1	0	1	1	1	1	1	1
b 2	0	1	2	2	2	2	2
c 3	0	1	2	3	3	3	3
f 4	0	1	2	3	3	3	4
5	0						

"fcba"

reverse("fcba") → abcf

```
if( t[i-1][j] > t[i][j])
{   i--;
}
else { j--; }
```

```
int i=m; int j=m;
while(i>0 && j>0)
{ if(s1[i-1] == s2[j-1])
  { s.push_back(s1[i]);
    i--;
    j--;
  }
else {
if(s1[i-1][j] > t[i][j-1])
```

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

Shortest Common Supperseqence.

I/p: a: AGGTAB — m = 6
b: GXTXAYB — m = 7

o/p: 9

P.S.: AGGTAB
GXTXAYB
→ AGGXTXAYB
Shortest Common Supperseqence

Approach

a: AGGTAB — n = 6
b: GXTXAYB — m = 7

(a+b)
AGGTABGXTXAYB — LCS(a,b) GTAB
(m+n) − LCS(a,b)
= 13 − 4 = 9.

∴ return (m+n) − LCS(a,b,m,n);

Q minimum no. of insertion and deletion to convert string a to string b.

I/p: a: heap
b: pea

O/p:

Identification for DP
→ choice ✗
→ Optimal — (min)

0/1 Knapsack

LCS

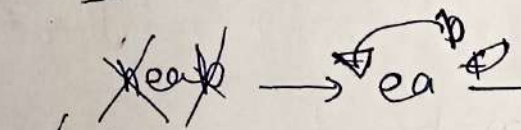
two string given
find optim.

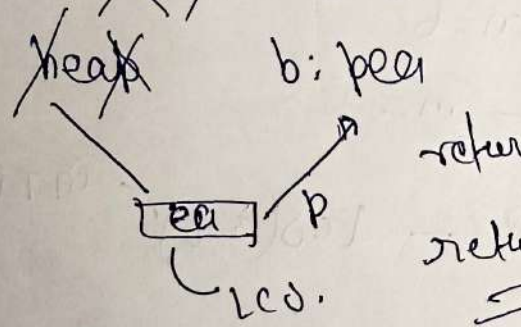
matching ✓ ✓

	I/P	φ	O/P
LCS:	a: b:	LCS	int
φ :	a: b:	min	int.

here two is matching
तो LCS use होगा।

a: heap b: pea





return no. deletion: length(heap) - lcs(a,b)
return no. of insertion: length(pea) - lcs(a,b)

Longest palindromic subsequence

I/P: S: agbcba

O/P: 5

agbcba
abcba.

PS: agbcba

subsequence

agb
bcb
abcba

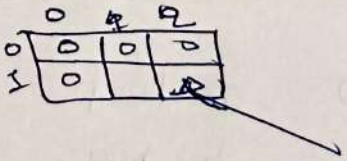
— LPS (abcba.)

matching.

	I/P	φ	O/P
LCS:	a: b:	LCS	int
LPS:	a:	LPS	int

b: hidden.

S_1: agbcba
S_2: reverse(S_1.begin(), S_1.end())
abcbga.



4/3 match.

S1: abcba S1 → S1

S2: abcba S2 → reverse(S1.begin(), S2.begin())

o/p: abcba → length = 5.

Sequence pattern

a = "AXY"

b = "ADXCPY" lcs(a, b) = 3 = min(a.length(), b.length()).

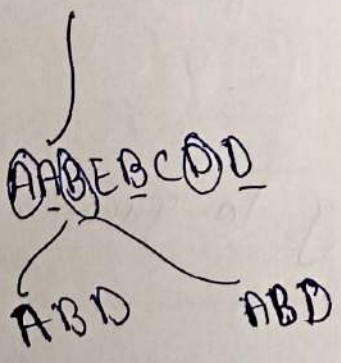
o/p: True/false.

o/p: True.

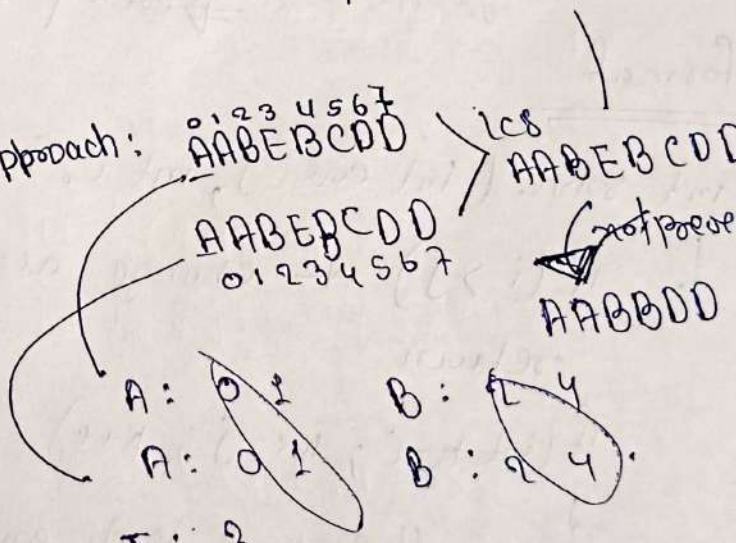
Longest Repeating subsequence.

Str = "AABEBCDD"

o/p: ABD



lcs = 3.



code variation

```
if(S1[i-1] == S2[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])
```

Minimum number of insertion in string to make it a palindrome.

I/P : "aebcbda" — m

a d e b c b d a

a d b c b e a

LCS = a b c b a

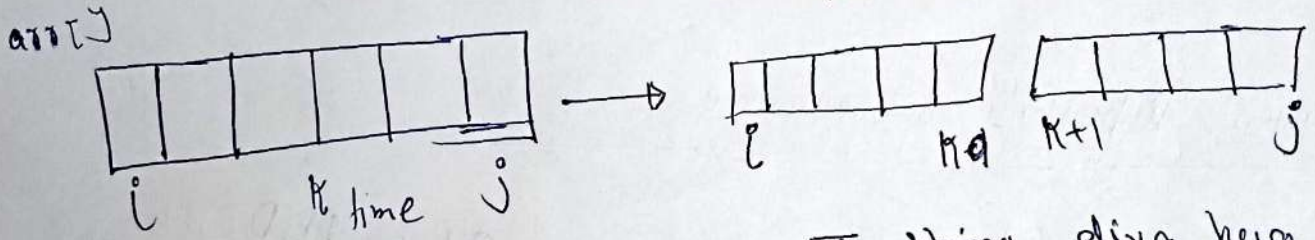
7 − 5 = 2

o/p: 2

s_2 = reverse(s_1)

return m − LCS(s_1, s_2);

Matrix chain multiplication



⇒ array ka string diya hua hoga.

formal

```
int solve (int arr[], int i, int j)
{  if (i > j)  ← change according to question.
     return
   if (int k=i; k<=j; k++)
   {     // Evaluat temp. ans.
         temp. ans = solve(arr, i, k)
                   + solve (arr, k+1, j)
         ans ← fun (temp. ans)
   }
   return ans.
};
```

matrix chain multiplication

recursive approach.

	0	1	2	3	4
arr[] :	40	20	30	10	30

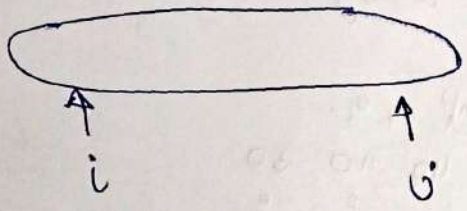
A_1 : 40 x 20

A_2 : 20 x 30

A_3 : 30 x 10

A_4 : 10 x 30

$A_i \rightarrow A[i-1] \times A[i]$



$A_j \rightarrow A[j-1] \times A[j]$

$A_4 \rightarrow A[3] \times A[4]$

10×30

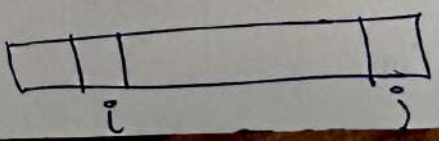
$\therefore j = 4 = n-1$

$\therefore i = 1 , j = n-1$

```
int solve(int arr[], int i, int j)
{ if (i>=j)
    return 0;
```

1. find i & j value.
2. find right BC
3. move k → i to j.

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



Check i and j condition

0	1	2	3	4
40	20	30	10	30

i j

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

$A_1 \rightarrow A[0] * A[1]$

$40 * 20$

$A_0 \rightarrow A[-1] * A[0]$ false.

$\therefore i = 1.$

$\therefore$

40	20	30	10	30

i = 1, j = n-1

if (i > j) → i == j

i,j

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

i,j

40

$\therefore i == j$ is not allowed.

(40 20) (30 10 30)   loop कैसे चलाना है।

i, k ↑ ... k ↑ ... k = j−1 ... j   if k == j.

k+1

i to k
matrix

empty

k+1 to j

40x20
20x30
30x10
~~10x~~

10 * 30
one matrix.

(40 20 30 10) (30)
i   k = j−1   j
k+1

Scheme 1.

k = i   k = j − 1   (i to k)   (k+1 to j)

now,
i
(40 20) (30 10 30)
k = i+1   j

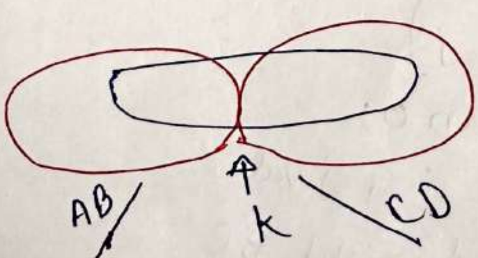
i to k−1   k to j   20x30
40 * 20   30x10
10 x 30

if ex 2.
40 20 30
i j
k
fun(i to k)   fun(k+1 to j)
40x20   20x30
40x20x30
arr[i−1] x arr[k] x arr[j]

k = i   k = j − 1   fun(i to k) fun(k+1 to j)

k = i+1   k = j   fun(i to k−1) fun(k to j)

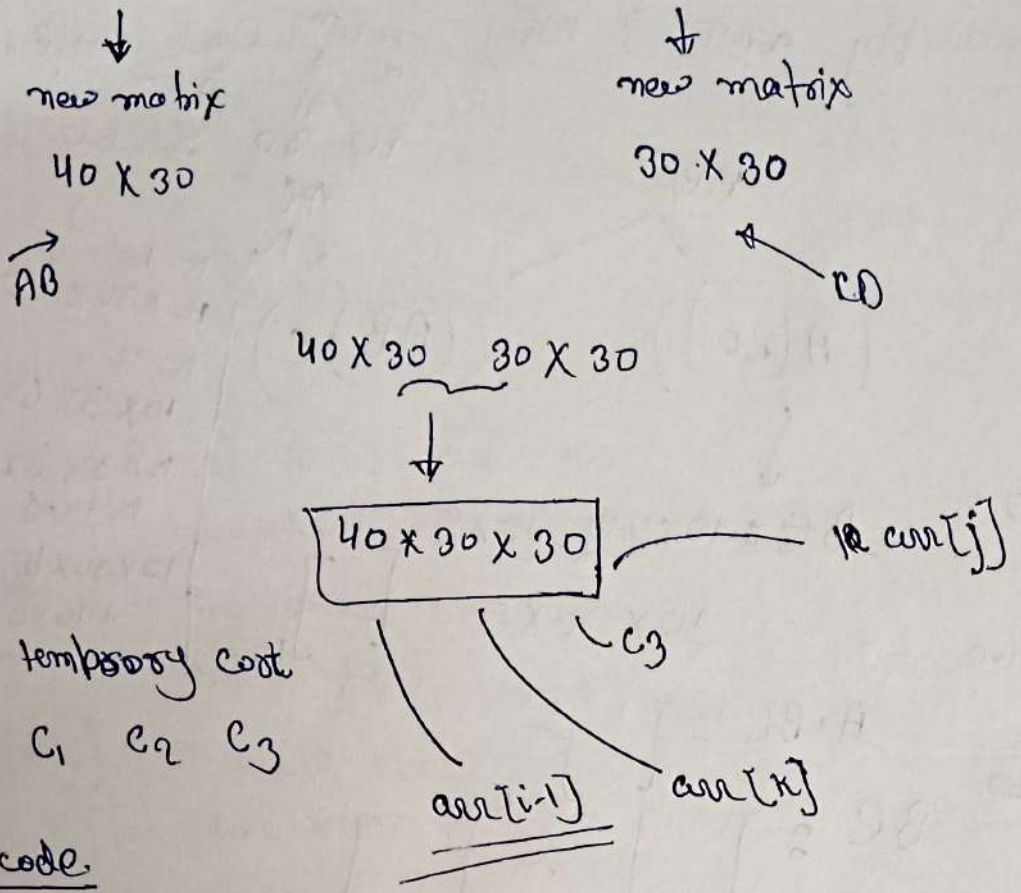
```
for (int k = i; k <= j-1; k++)
{
    solve(arr, i, k);
    solve(arr, k+1, j)
}
```



solve (i to k)   +   solve (k+1 to j)
↓
cost.

eg.:
0 1 2 3 4
40 20 30 10 30
i k j

fun(i to k)   fun(k+1 to j)
40*20 20x30   30x10 10x30
40 * 20 * 30 / c1   30 x 10 x 30 → c2



code

```
int solve (int arr[] , int i, int j)
{
  if(i>=j)
    return 0;
    int temp = INT_MIN;
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] ;
    if (temp > min)
      { min = temp;
  }
}
  return min;
}
```

find
1. i j
2. find BC
3. find k loop scheme.
4. Calculate ans for temp ans.

k = i
k = i+1
k = i+2
:
k = j-1

matric ABC को multiply करना है जिसमें min cost ज्ञात करें.

ABC

Size of matrix

$A = [\ ]_{10 \times 30}$

$B = [\ ]_{30 \times 5}$

$C = [\ ]_{5 \times 60}$

$A \cdot BC : 10 \times 30 \quad 30 \times 60$

$: 10 \times 30 \times 60$

Cost

$= (30 \times 5 \times 60) + (10 \times 30 \times 60)$

$= 27000$

ABC

(A(BC)) ((AB)C)

~~BC : 10×30 30×5~~

~~10×30×5~~

~~A·BC = [ ]~~

$BC : [\ ]_{30 \times 5} \times [\ ]_{5 \times 60}$

$: [\ ]_{30 \times 60}$

$BC : 30 \times 5 \quad 5 \times 60$

$= 30 \times 5 \times 60$

Now ((AB)C)

$AB : 10 \times 30 \quad 30 \times 5$

$= 10 \times 30 \times 5$

$= 1500$

$(AB)C : 10 \times 5 \quad 5 \times 60$

$: 10 \times 5 \times 60$

$: 3000$

$ABC = 10 \times 30 \times 5 + 10 \times 5 \times 60 = 4500$

$\min(A(BC), (AB)C) = 4500$

min
3 2 3
A B C
10 30 5 60
i k j

k=1
10×30×60
k=2
10×5×60
k=3, j-1=2
k != j-1
10×30×60
+10×5×60

ABCD

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

A	B	C	D
40	20	30	10	30

((AB)C)D

AB → 40×20 20×30 → 40×30

40×20×30 — C_1

(AB).C → 40×30 30×10 — C_2

40×30×10 [ ]$_{40\times10}$

((AB)C)D → 40×10 10×30 — C_3

40×10×30 [ ]$_{40\times30}$

Cost = 40×20×30 + 40×30×10 + 40×10×30

= 48000

0	1	2	3
10	30	50	60

i j

↑ K = 1

10 × 30 × 60

arr[0] × arr[i] × arr[3]

arr[i-1] × arr[k] × arr[j]

solve(arr, i, k) — solve(arr, 1, 1) → 0

solve(arr, 2, 3)

A(BC)

solve(arr, 2, 2) → 0

solve(arr, 3, 3) → 0

= 10×30×60 + 30×5×60

(AB)C

10×30×5 + 10×5×60

0	1	2	3
10	30	5	60

i K

K = 2

10×5×60

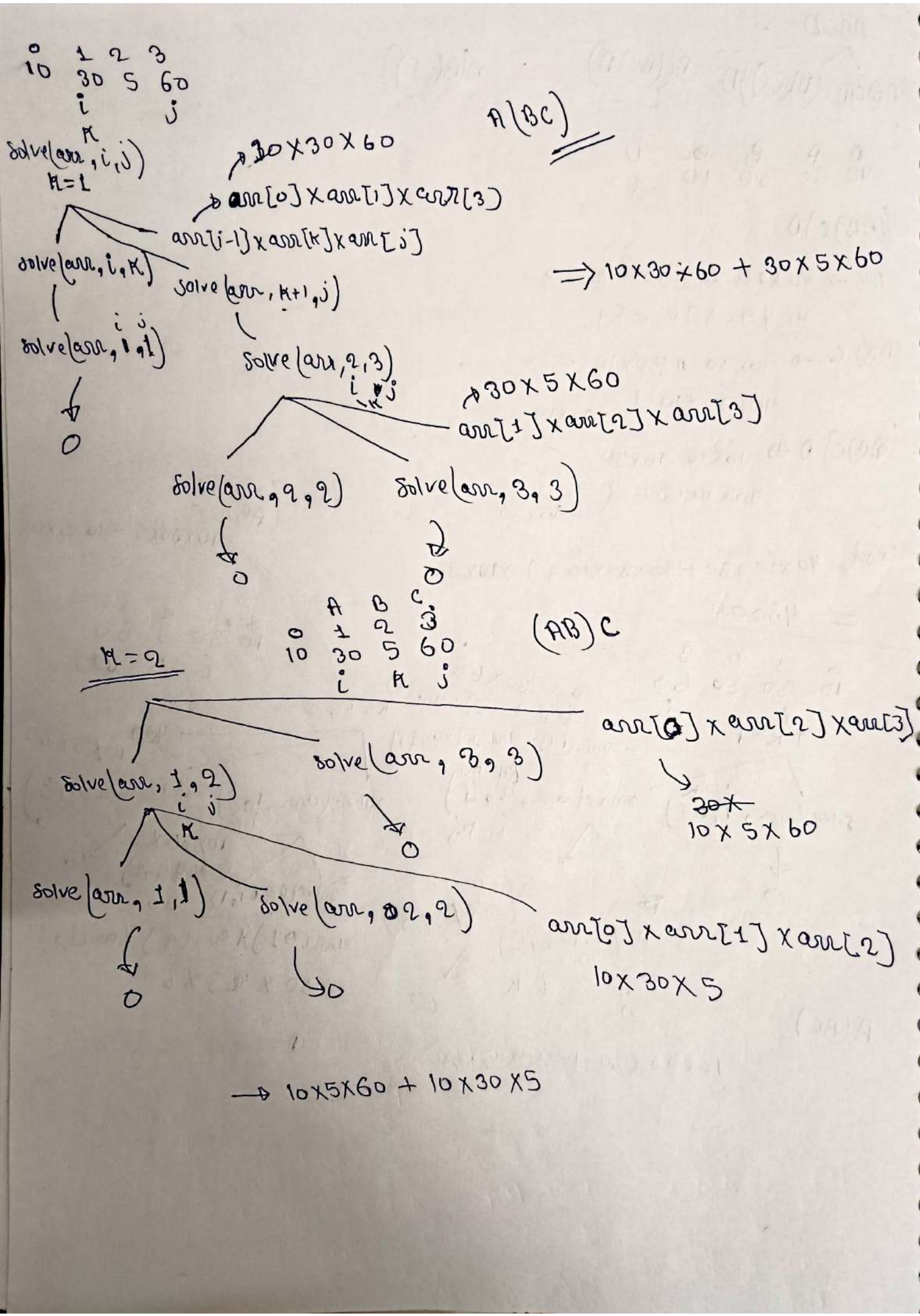
solve(arr, 1, 2) — solve(arr, 3, 3) → 0

10×30×5

solve(arr, 1, 1) → 0

arr[0] × arr[2] × arr[3]

30 × 5 × 60

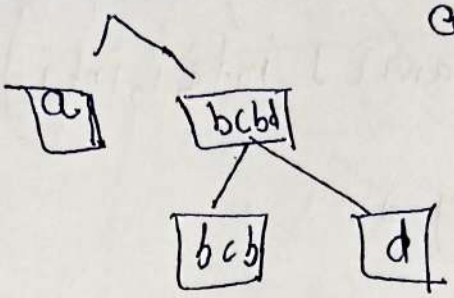

0 1 2 3
10 30 5 60
i k j
solve(arr, i, j)
k=1
30 x 30 x 60
arr[0] x arr[1] x arr[3]
A(BC)
arr[i-1] x arr[k] x arr[j]
solve(arr, i, k)
solve(arr, k+1, j)
⇒ 10x30x60 + 30x5x60
solve(arr, i, i)
0
solve(arr, 2, 3)
30 x 5 x 60
arr[1] x arr[2] x arr[3]
solve(arr, 2, 2)
solve(arr, 3, 3)
0
0
A B C
0 1 2 3
10 30 5 60
i k j
(AB)C
k=2
arr[0] x arr[2] x arr[3]
solve(arr, 1, 2)
solve(arr, 3, 3)
30x
10 x 5 x 60
0
solve(arr, 1, 1)
solve(arr, 2, 2)
arr[0] x arr[1] x arr[2]
10 x 30 x 5
0
0
→ 10x5x60 + 10x30x5

Palindrome partitioning Recursive.

String s: nitik , abcbd

op : 2

Problem statement

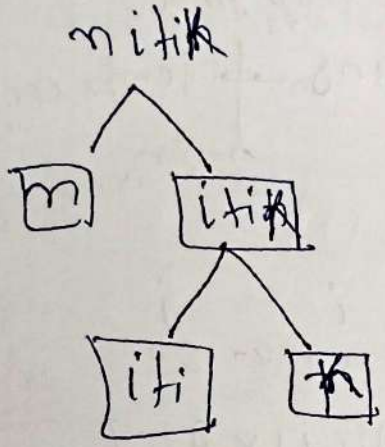


a _ bcb _ d

a | bcb | d

min of partition = 2.

op = 2.



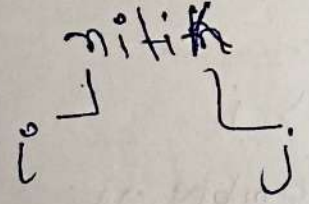
worst case

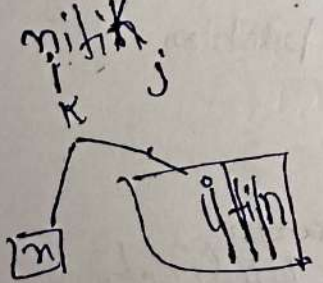
nitin

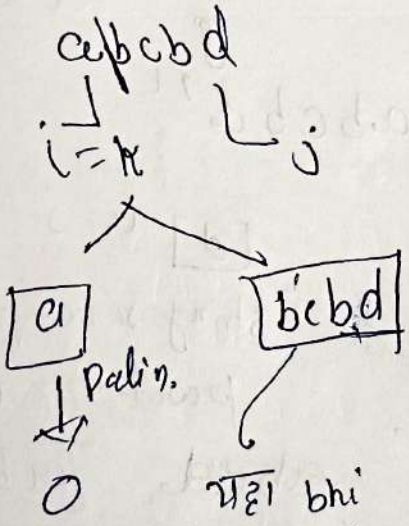
n | i | t | i | n.

∴ no. of partition = n-1 (s.size().

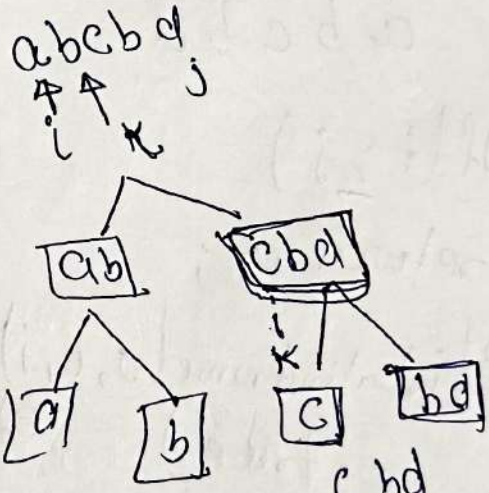
mcm formal



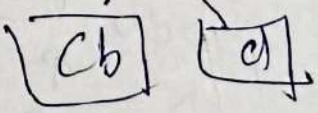




यहाँ भी
har place
पर partition करके
dekhna hoga kaha
palindrome mil रही है।

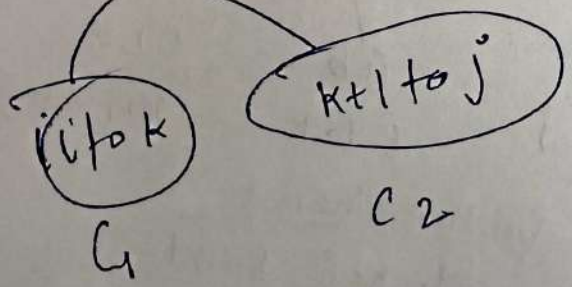


c b | d



~~abcd~~

a|bcbd



formal specification.

problem specification

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

0 , n-1

string s

if (i > j)
return 0;

Steps

1. find i/j
2. BC
3. K loop find
4. ans ~~=~~ temp ans
min.

1. find i/j

i=0 , j=n-1

i j
a b c b d

i j
20 30 40 50

$A_i \rightarrow A[i-1] \times A[i]$

$A_j \rightarrow A[j-1] \times A[j]$

i=j 40 × 50

2. BC

i j
a b c b d

if (i ≥ j)
return 0;

if (isPalindrome(s, i, j) == true)
return 0;

s, i
a b c b d

d
i == j
string size 1
partion 0.

i > j
empty string
partion 0

abcd ~~abcd~~ aba palindrome.
return 0 partion

3. K loop find

i j
a b c b d

i to K
K+1 empty
abcbd

i j
a b c b d
K
a
i to K
bcbd
K+1 to j

i j
a b c b d
K
ab
i to K
cbd
K+1 to j

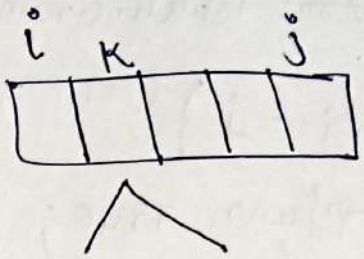
∴ i j
a b c b d
K
abcb d

partion scheme

1. K=i K=j-1
i to K
K+1 to j

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

partition



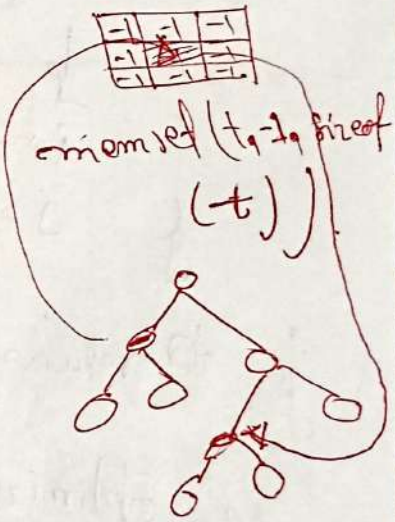
i to k-1 k to j

Q. k = i+1 k = j { i to k-1 } { k to j }

Code

```
int static t[1001][1001]
int solve(string s, int i, int j){
  if(i>=j)
    return 0;
  if(isPalindrome(s, i, j)==true)
    return 0;
  if(t[i][j]!=-1) return t[i][j];
  int ans = INT_MAX;
  for(int k=i; k<=j-1; k++){
    int temp = solve(s, i, k) + solve(s, k+1, j) + 1;
    ans = min(ans, temp);
  }
  t[i][j] = ans;
  return ans;
}
```

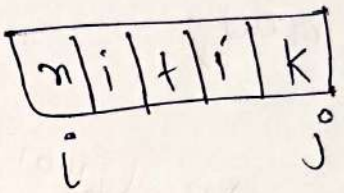
for memoization



Check planidrome.

```
bool ~~isPalin~~ isPalindrome (string S, int i, int j)
{  if (i == j)
     return True;
   if (i > j)
     return true;
   while (i < j){
     if (S[i] != S[j])
       { ~~re~~ return false;
       }
     i++;
   }  j--;
}  ~~re~~ return true;
```

S → Yes palindrome



code optimization:

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

left / right

int temp: left + right + 1

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

```
if (t[k+1][j] != -1){
   right = t[k+1][j]
}else{
   right = solve(S, k+1, j);
   ~~t[k+1][j]~~
   t[k+1][j] = right;
}
```

Evaluate Expression to True Boolean Parenthesization Recursive.

String : T or F and T

IP — T | F & T

Problem statement

इसमें कितने ways bracket introduce करे कि True आ जाए।

String : T, F, &, |, ^ — char + 3 string

Identification for mcm e.g.

T | F & T

((T|F)&T) (T|(F&T))

S : "T | F & T ^ F"

no. of ways.

(—)()—T

(——)()—T

()()()

T | F & T ^ F

i k k+2

(T)|(F&T^F)

for mcm

1.) find i & j

i=0, j=n-1

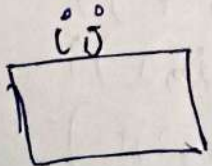
0 1 2 3 4 5 6

T | F & T ^ F

i j

Q. BC

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



$(^{i}_{T} | F \& T) \wedge (^{j}_{F})$

K

$(i \text{ to } K-1)$ $(K+1 \text{ to } j)$

Exp ∧ Exp

( )

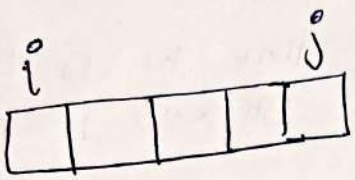
i to $K-1$ $K+1$ to j

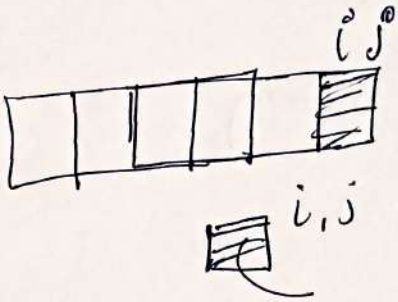
int solve (string S, int i, int j, boolean True)

BC

$^{i}_{T} | F \& T \wedge ^{j}_{F}$

K






```
if (i > j)
    return false;
if (i == j) {
    if (istrue == True) {
        return S[i] == 'T'
    }
    else { return S[i] == 'F' }
}
```

~~if (isto~~

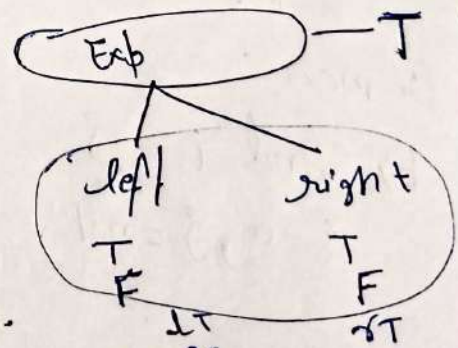
3.) find K loop

$(i \text{ to } K-1)$ $(K+1 \text{ to } j)$

$K = j-1$

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

i $K = i+1$, $K = K+2$ j



scheme 1.

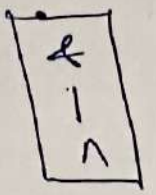
$K = i+1$ $K = j-1$

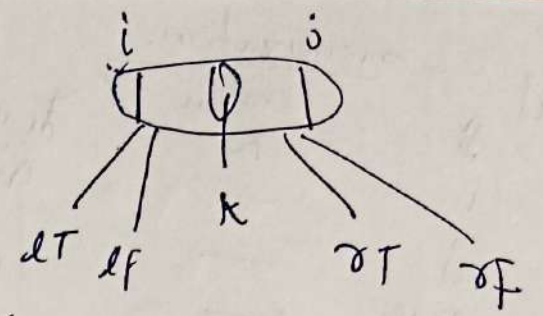
$(i \text{ to } K-1)$ $(K+1 \text{ to } j)$

```
int ans = 0
for (int K = i+1; K <= j-1; K+2)
{ 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);
```

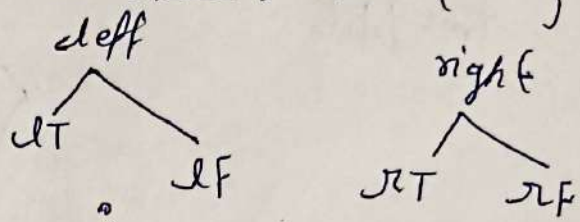
ans += lT * rF + lF * rT

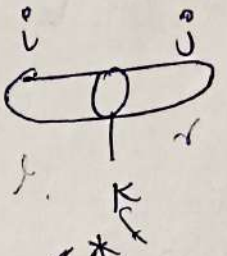
total Operators

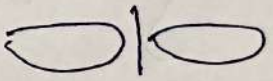


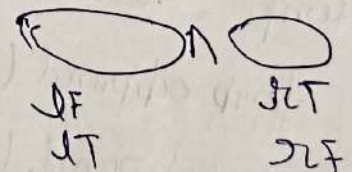



```
if (S[K] == '&')
{
  if (isTrue == True)
    { ans+ = lT & rT;
    }
  else{
      ans+ = ans+ lF * rT +
             lT * rF + lF + rF;
    }
}
else if (S[K] == '|')
{
  if (istrue == True)
  ans = ans+ lT * rT + lF * rT + lT * rF ;
  else
      ans = ans+ lF * rF ;
}
elseif (S[K] == '^')
{
    if (istrue == True)
    ans = ans+ lF * rT + lT * rF ;
    else
      ans = ans + lT * rT + lF * rF ;
}
}
return ans ;
```









memoization.

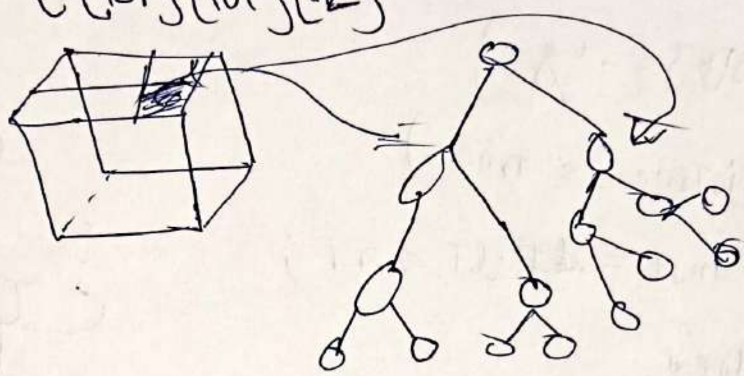
variable	change
s	X
i	✓
j	✓
True	✓

True / True false

है यहाँ 3D matrix बनेगा।

0 < s < 100

t[101][101][2]



mapping

Global declare

```
unordered_map<string, int> mp;
int main()
{
    mp.clear();
    solve()
}
```

"50_90_f" → 9

sc है ठीक

```
string temp = to_string(i);
    temp.append(" ");
temp.append(to_string(j));
    temp.append(" ")
    temp.append(to_string(istrue));
if (mp.find(temp) != mp.end())
    return mp[temp];

return mp[temp] = ans.
```

temp	value
1_0	

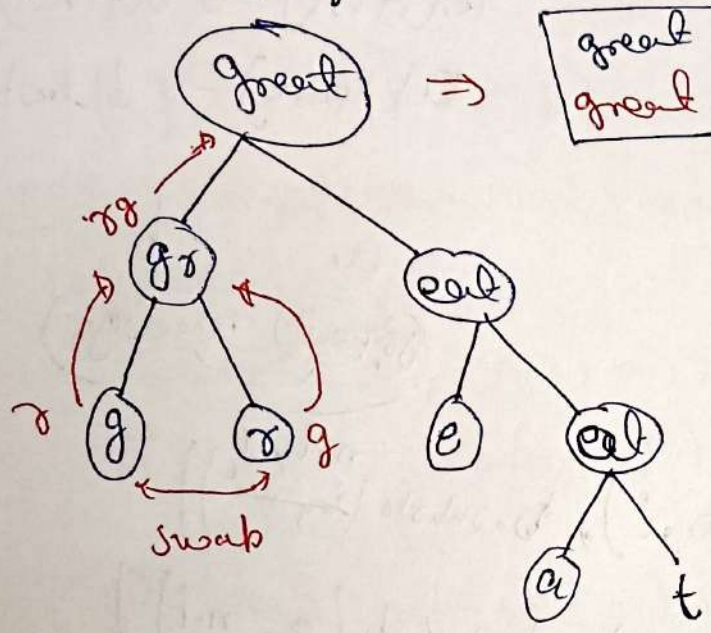
mp[temp]

Scrambled String

I/P : a: "great"
b: "rgeat"

O/P : True

Scramble string



great
great

⇒ this is Scramble string.

इस तरह - तरह break कर सकते

great

g | reat

gr | eat

gre | at

grea | t

great |

great | "" not allow.

i=1 to i=n-1

this mcm problem.

Approach.

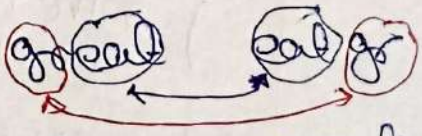
Scramble

Swapping occur

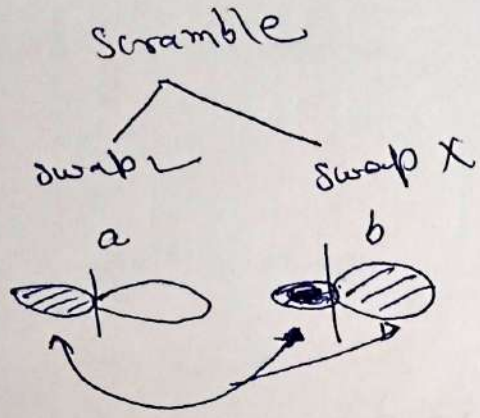
swapping not occur.

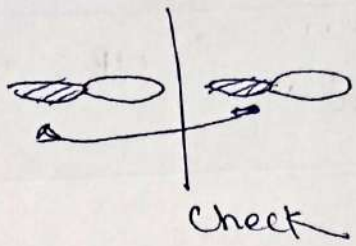
i=2

swap ~



swap X great great





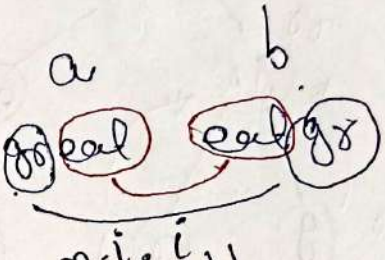
Check
a → b
a(~~last~~ first) → b(last)

a(first) → b(first)
a(last) → b(last)

Case-1
Swap

i
0 1 2 3 4
gr eat

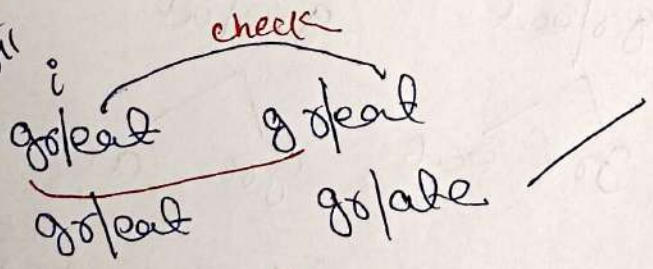
n-i
0 1 2 3 4
eat gr

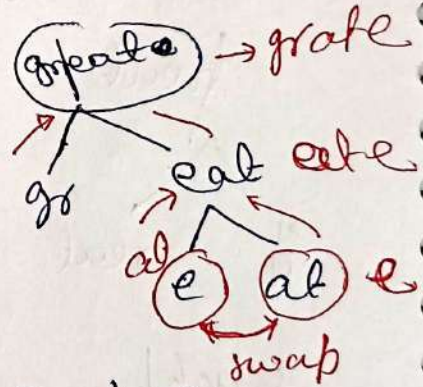


Condition-I

```
if ( solve(a.substr(0,i), b.substr(~~i, n-i~~))
     &&
     solve(a.substr(i, n-i), b.substr(0, n-i)) )
```

Case-II
swap ~~did~~ not





Condition-II

```
if ( solve(a.substr(0,i), b.substr(0,i)) == true
     &&
     solve(a.substr(i, n-i), b.substr(i, n-i)) == true )
```

if (condition I ~~=~~ || condition II == true)
return true.

Base condition

great a ngreat b

if (length is diff)
return false.

if (a & b are empty) return true.

if (a.compare(b) == b) → return true.

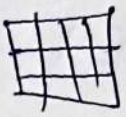
if (a.length <= 1) → return false.

Code unordered_map<string, int/bool> mp

```
int main() {
    string a, b; cin>>a>>b
    if (a.length() != b.length())
        return false;
    if (a.length() == b.length() == 0)
        return true;
    mp.clear();
    solve (a, b);
}

bool solve ( string a, string b) {
    if (a.compare(b) == 0 )
        return true;
    If ( a.length() <= 1)
        return false;
    int n = a.length();
    bool flag = false
    string key = "a" + " " + "b";
    if (mp.find(key) != mp.end() )
        return mp[key];
```

```
for(int i=1; i<n ; (++)
{ if(condI || condII)
  {
    flag == true;
      break;
  }
}
return flag;
```

mp[key] = false;

AC + 

mapping.

Key	Value

S = aAaaBACacbE

no. of pair like, aA, Bb, ~~bE~~, Cc (small and captial)

Sol. (a, A

```
map<char, int> m
for(int i=0; i<n; i++)
  { m[s[i]]++;
  }

for(int c='a'; c<='z'; c++)
  { int h1 = m[c];
    int h2 = m[(char)(c-32)];

    int ct += min(h1, h2);
  }

cout << ct << endl;
```

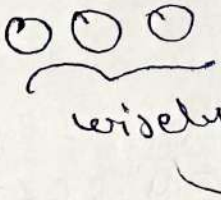
output : 4

{aA, aA, bB, cC}

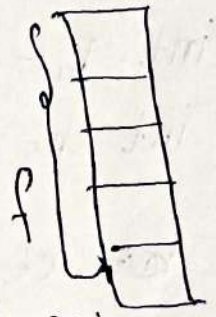
Egg Dropping Problem.

I/P: egg(e) : 3
floor(f) : 5

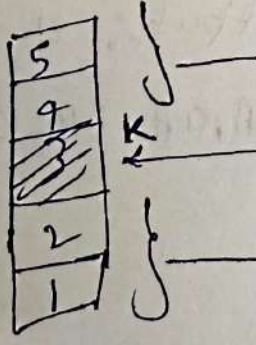
हमें find करना है critical floor जहाँ egg break ना करें और minimize करना है no. of attempt to break the egg.



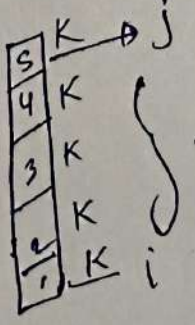
If given e=1 and f is floor then worst case f attempt होंगे।



हम start bottom से करेंगे तभी find use करें of egg not break.



— Here also egg break
— of from here egg break
— chance to not break



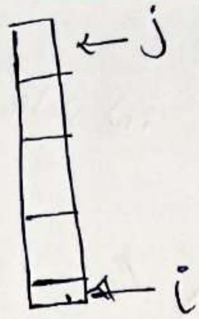
question
from where start dropping egg (means find K)

So, every place to check by loop.

```
for (k=1; k<=f; k++){
}
```

← mcm format.

1. find i & j
2. BC
3. K loop
4. ans ← from temp ans.



1. i = 1, j = 1

3. K = i ; K = j

BC

I/P

e → 0/1

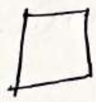
0 eggs → no find ans (invalid input)

1 eggs → if (e == 1) → return f (worst case में पहले से last floor से dropp.)

f → 0/1

0 floor → return 0;

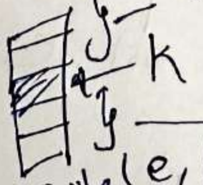
If (f == 1) → return 1;



If (e == 1)
return f

If (f == 0 || f == 1)
return f;

Temp ans.



chance h break hone का। इसमें floor से drop करने पर

f solve(e, f) — वैसे check करेंगे नहीं

000 Break → solve(e-1, f-1)

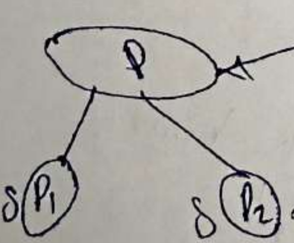
000 Break X → solve(e, f-K)

```
int solve(int e, int f)
{  if(e == 1)
      return f;
   if(f==0 || f==1)
      return f;
   if(t[e][f] != -1) return t[e][f]
   int ans = INT_max;
   for(int k=1; k<=f; k++){
      int temp = 1 + max(solve(e-1, f-1), solve(e, f-k));
      ans = min(ans, temp);
   }
   t[e][f] = ans.
   return ans;
```

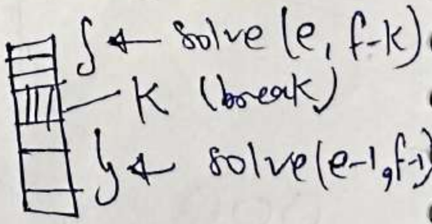
1<e<1000 1<f<100

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

main() {
   memset(t, -1, sizeof(t))
   return solve(e, f)
}
```

हर बार attempt कर रहे worst case think कर रहे है



से check होगा पर इसके लिए sp1 & sp2 को solve करना पड़ेगा पर हो सकता है sp1 solve और sp2 नहीं। तो हम क्यू करेंगे इसे भी check करें solve करने पहले।

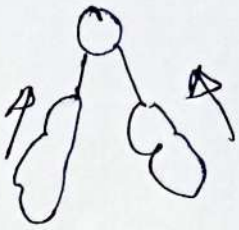



```
int low =  if(t[e-1][j-1] != -1)
               int low = t[e-1][j-1]
           else
               int low = solve(e-1, f-1);
if(t[e][f-k] != -1)
   int high = t[e][f-e];
else
   int high = solve(e, f-e).

int temp = 1+ max(low, high)
```

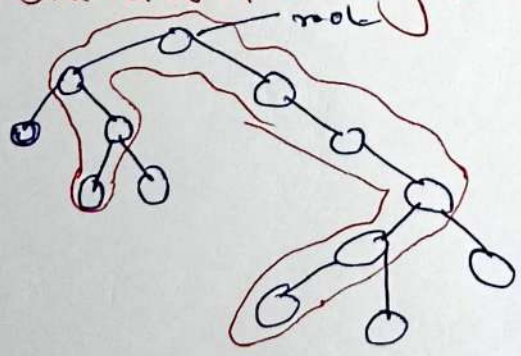
Dynamic Programming on Tree

General Syntax.

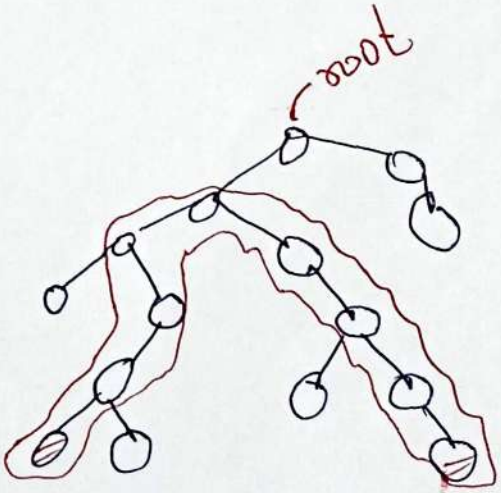
```
int solve(Node* root, int &res)
{
    if(root==nullptr)
        return 0          BC

    int l = solve(root->left, res);
    int r = solve(root->right, res);
    int temp= Calculate temp answer relation (l+i+r)
    int ans= max(temp, ans)
    res = max(res, ans)
    return temp
```



Diameter of Binary tree.



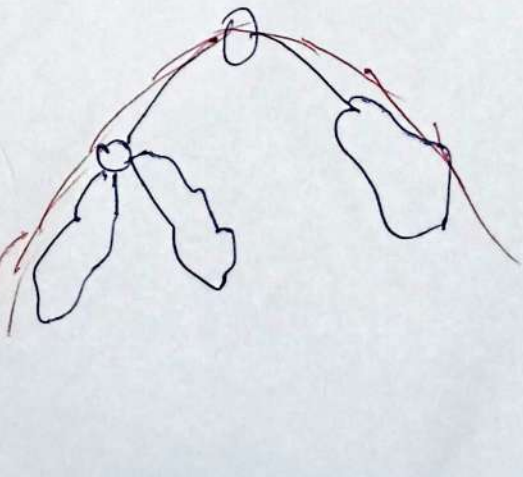
root node include.



root node not include

Code

```
int solve(Node *root, int &res)
{
    if(root==nullptr)
        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;
}
```