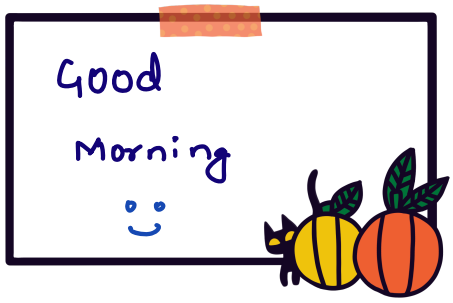


Content for Today

- → Introduction
- → Fibonacii series ✓
- → Climb stairs ✓
- → Max sum without adjacent elements ✓
- → Unique paths ✓
- → 0-1 Knapsack
- → Unbounded knapsack
- → Rod cutting problem
  Longest common subsequence ✓
- → Edit distance ✓

DP / Dynamic Programming

↳ optimize recursion

A[] = | 1 | 2 | 3 |

pf[] = | 1 | 3 | 6 |

(0 1 2)

pf[i] = sum of all elements from 0 to i

Fibonacii series

0 1 1 2 3 5 8 13 21 ...

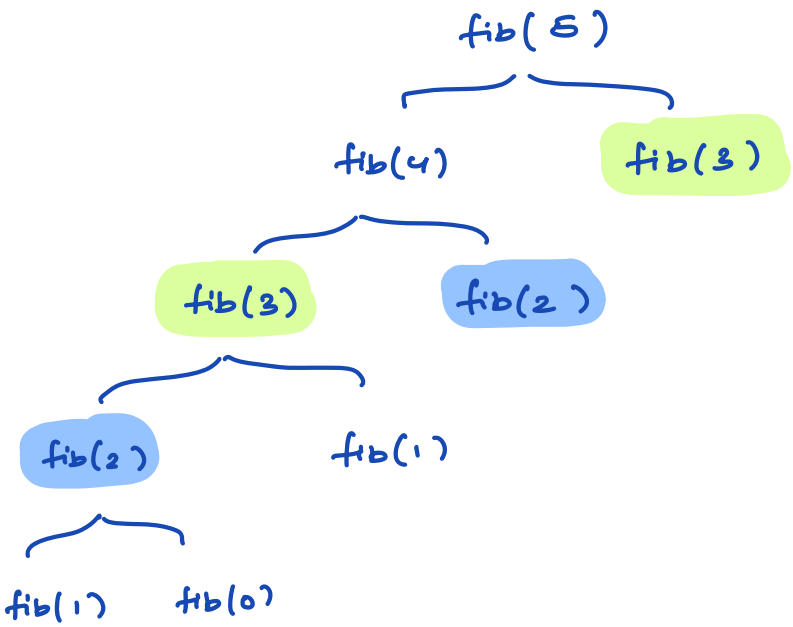
$$fib(n) = fib(n-1) + fib(n-2)$$

$$fib(0) = 0 \quad fib(1) = 1$$

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

TC : $O(2^n)$

SC : $O(n)$ → spare by recursive stack

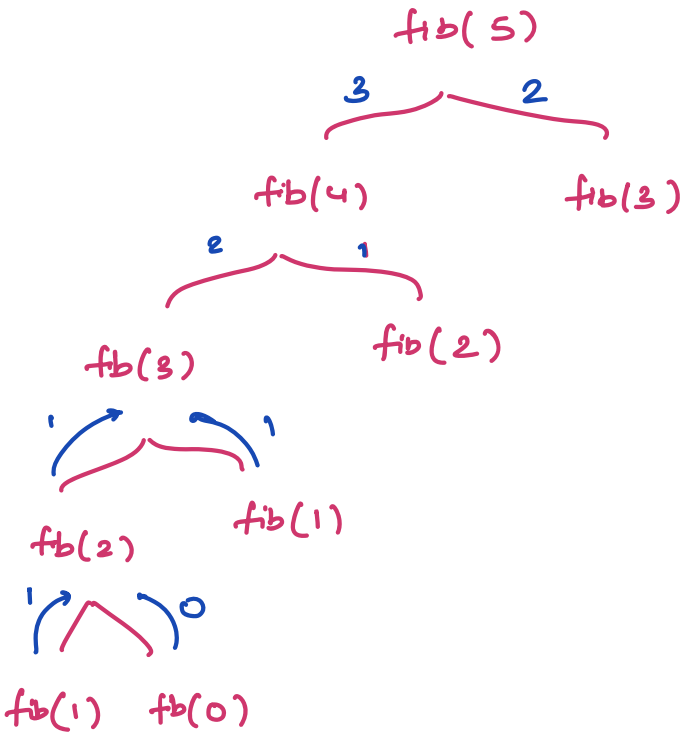


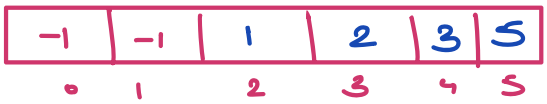
Overlapping sub problems → Repeating subproblems

Optimal substructure → Solving problem by reducing problem in smaller subproblems

Solution → store the answer for already solved problems.

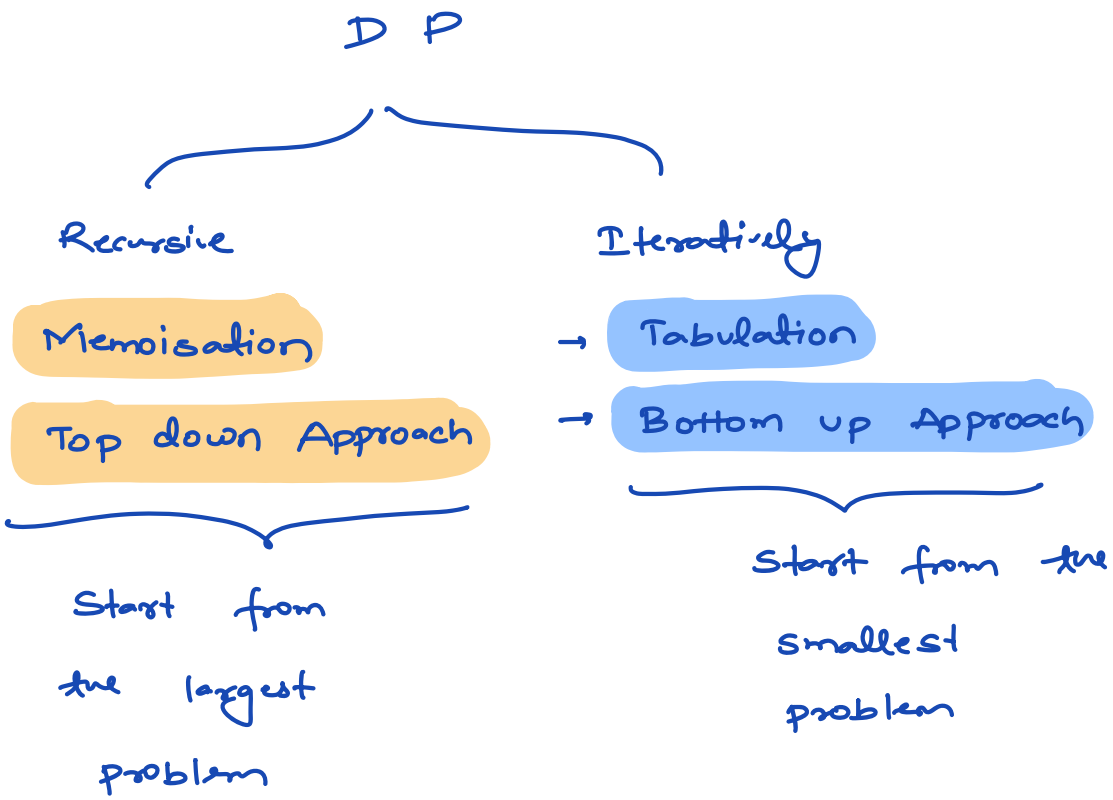
```
int [] dp = new int [n+1]   ↘ initialise the dp = -1

int fib(n)
{
    if (n ≤ 1) return n;
    if ( dp[n] != -1 ) return dp[n];
    return dp[n] = fib(n-1) + fib(n-2)
}
```


fib(5)
3
2
fib(4)
fib(3)
2
1
fib(3)
fib(2)
1
1
fib(2)
fib(1)
1
0
fib(1)
fib(0)


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

TC: O(n)

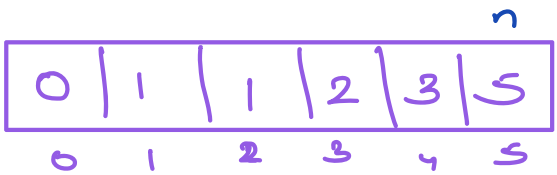
SC: O(n)


D P
Recursive
Memoisation
Top down Approach
Start from the largest problem
Iteratively
Tabulation
Bottom up Approach
Start from the smallest problem

dp size = n+1

fib(0) = 0

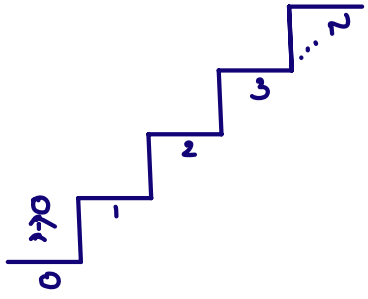
fib(1) = 1

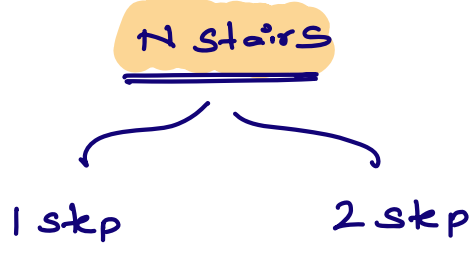



```
dp[0] = 0
dp[1] = 1
for (i=2; i<=n; i++) {
    dp[i] = dp[i-1] + dp[i-2]
}
return dp[N]
```

TC : O(n)
SC : O(n)
(No recursive stack space)

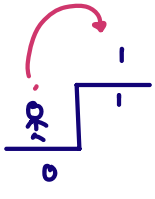
* Stairs → No. of ways to reach Nth stair

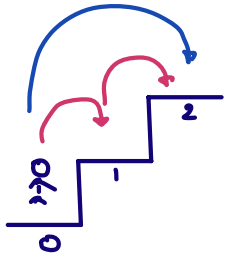




N=1

Ans=1





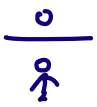
N=2

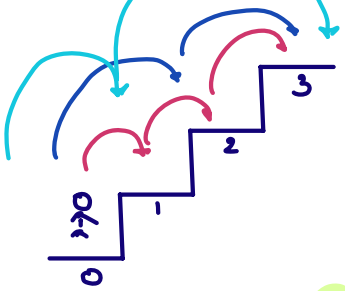
Ans=2

N = 0

Ans = 1

do nothing





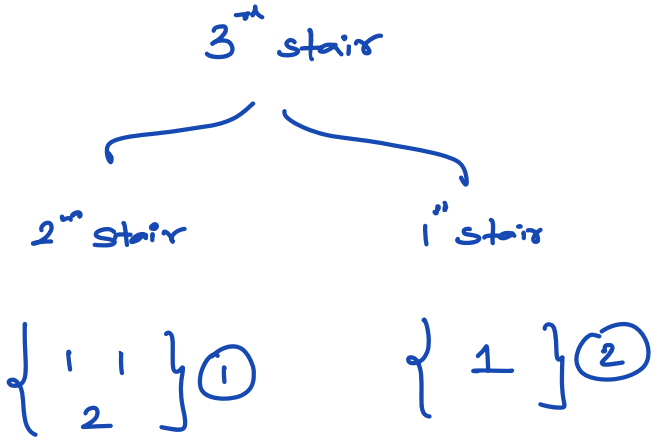
N = 3

Ans = 3

(1, 1, 1)

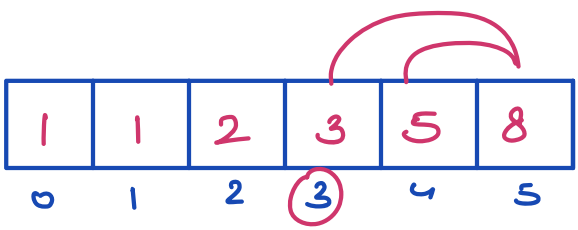
(1, 2)

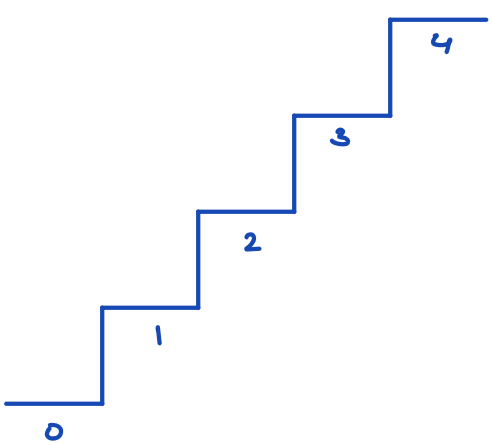
(2, 1)

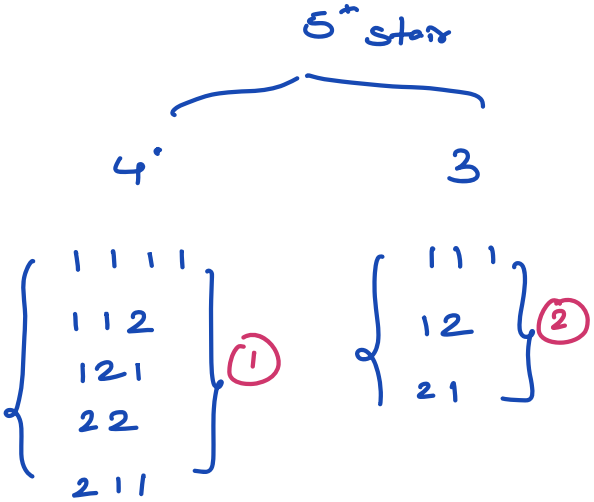


stairs(n) = stairs(n-1) + stairs(n-2)

dp[3] = no. of ways to reach 3rd step from 0th step







$dp[0] = 1$

$dp[i] = dp[i-1] + dp[i-2]$

Q3

Given $ar[N]$ calculate **max subsequence sum**

Note 1 → In a subseq 2 adj ele can't be picked

Note 2 → Empty subseq is also valid.

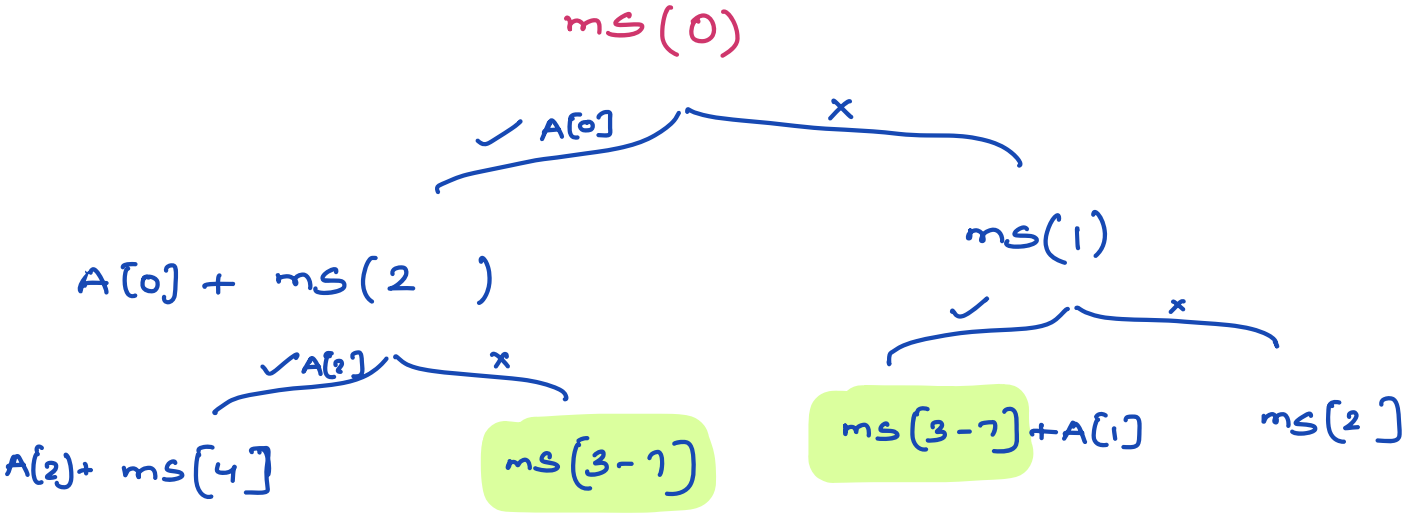
$ar[3] = \{9 \quad 14 \quad 3\} = \{14\} = 14$

$ar[4] = \{\underset{0}{9} \quad \underset{1}{4} \quad \underset{2}{13} \quad \underset{3}{24}\} = \{9, 24\} = 33$

$ar[3] = \{13 \quad 14 \quad 2\} = \{13, 2\} = 15$

$ar[4] = \{-4 \quad -3 \quad -2 \quad -3\} = \underline{0}$ ⇒ Empty subsequence

$A[] = \{\underset{0}{2} \quad \underset{1}{-1} \quad \underset{2}{-4} \quad \underset{3}{5} \quad \underset{4}{3} \quad \underset{5}{-1} \quad \underset{6}{4} \quad \underset{7}{2}\}$




```
int dp[n] = Int_Min
int maxsubsum (int[] A, int i)
{
    if (i >= n) return 0;
    if (dp[i] != -1) return dp[n]
    int pick = maxsubsum(A, i+2) + A[i]
    int npick = maxsubsum(A, i+1)

    return dp[i] = max(pick, notpick)
}

main(){
    SOP (max(maxsubsum(A,0), 0);
}
```

* Bottom up Approach (DP needs to be filled from last)

dp[i] = max subsequence sum [i n-1] such that there is no adj ele

```
int dp[n];
for (i=n-1; i>=0; i--){
    if (i+1)<n    dp[i] = dp[i+1] + 0
    if (i+2)<n    dp[i] = max(dp[i], dp[i+2]);
}
return dp[0];
```

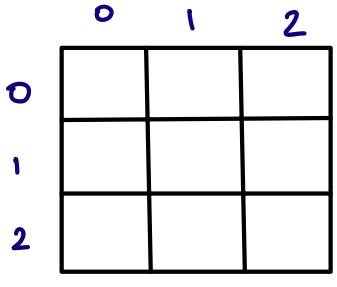
TC = O(n)

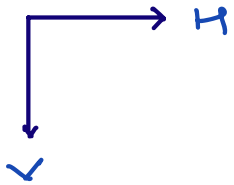
SC : O(n)

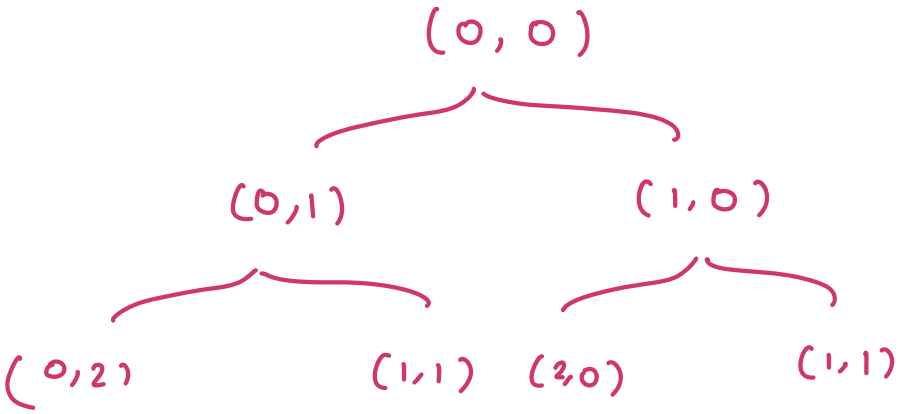
Q Number of ways to go from (0,0) → BR cell in mat[N][M]

Note :- From cell we can go to right or down

Eg :-



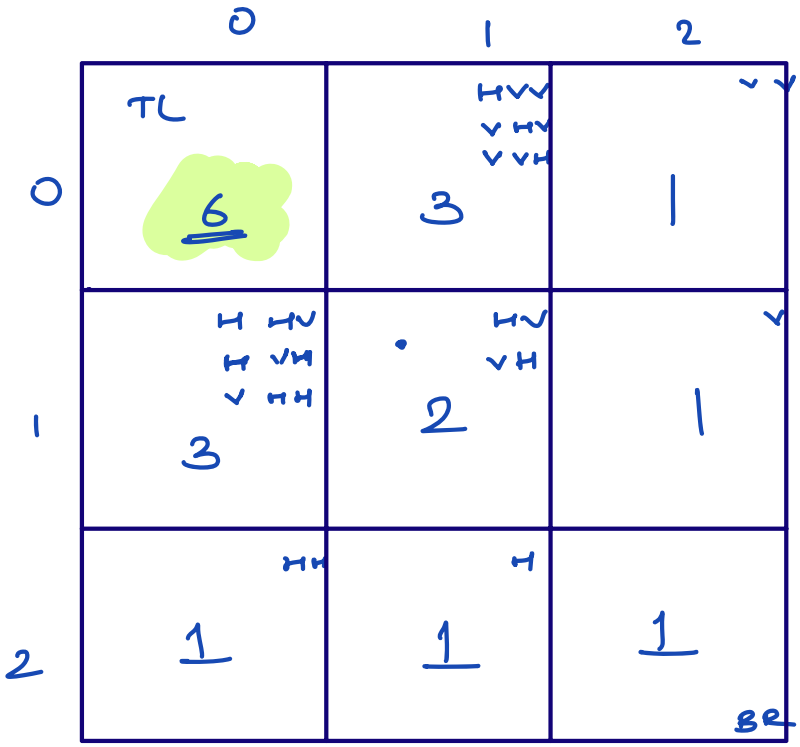





```
ways(int i, int j)
{
    if (i==n-1 && j==n-1) return 1;
    if (i≥n || j≥n) return 0;
    if (dp[i][j] != -1) return dp[i][j]
    return dp[i][j] = ways(i, j+1) + ways(i+1, j)
}
```

```
// iterative approach
for (i = n-1; i ≥ 0; i--)
{
    for (j=n-1; j≥0; j--){
        if (i==n-1 && j==n-1) dp[i][j] = 1
        else {
            dp[i][j] = dp[i+1][j] + dp[i][j+1]
        }
    }
}
return dp[0][0];
```

12:41 → 12:47 PM



0/1 KnapSack

Given N items each with a weight & value, find max value which can be obtained by picking items such that total weight of all items $\leq K$

Note 1 :- Every item can be picked at max 1 time

Note 2 :- We can not take a part of item

Eg:-

N = 4 items K = 50

Items:	0	1	2	3
Value[] :	100	60	120	150
Weight[]:	20	10	30	40
V/W =	5	6	4	3.75

Greedy → Pick item with max V/W ratio ✗

$\Sigma V = 60 + 100 = 160$

$\Sigma wt = 10 + 20$

Greedy - Pick item with max value

$\Sigma V = 150 + 60 = 210$

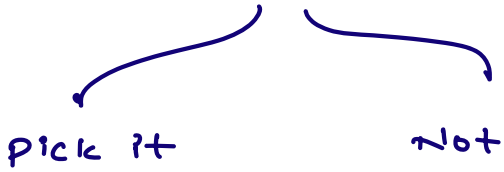
$\Sigma wt = 40 + 10 = 50$

* Correct ans $\quad \Sigma v = 100 + 120 = 220$

$\Sigma wt = 20 + 30 = 50$

Idea 2 → Generate all combinations.

Every ele has two choices



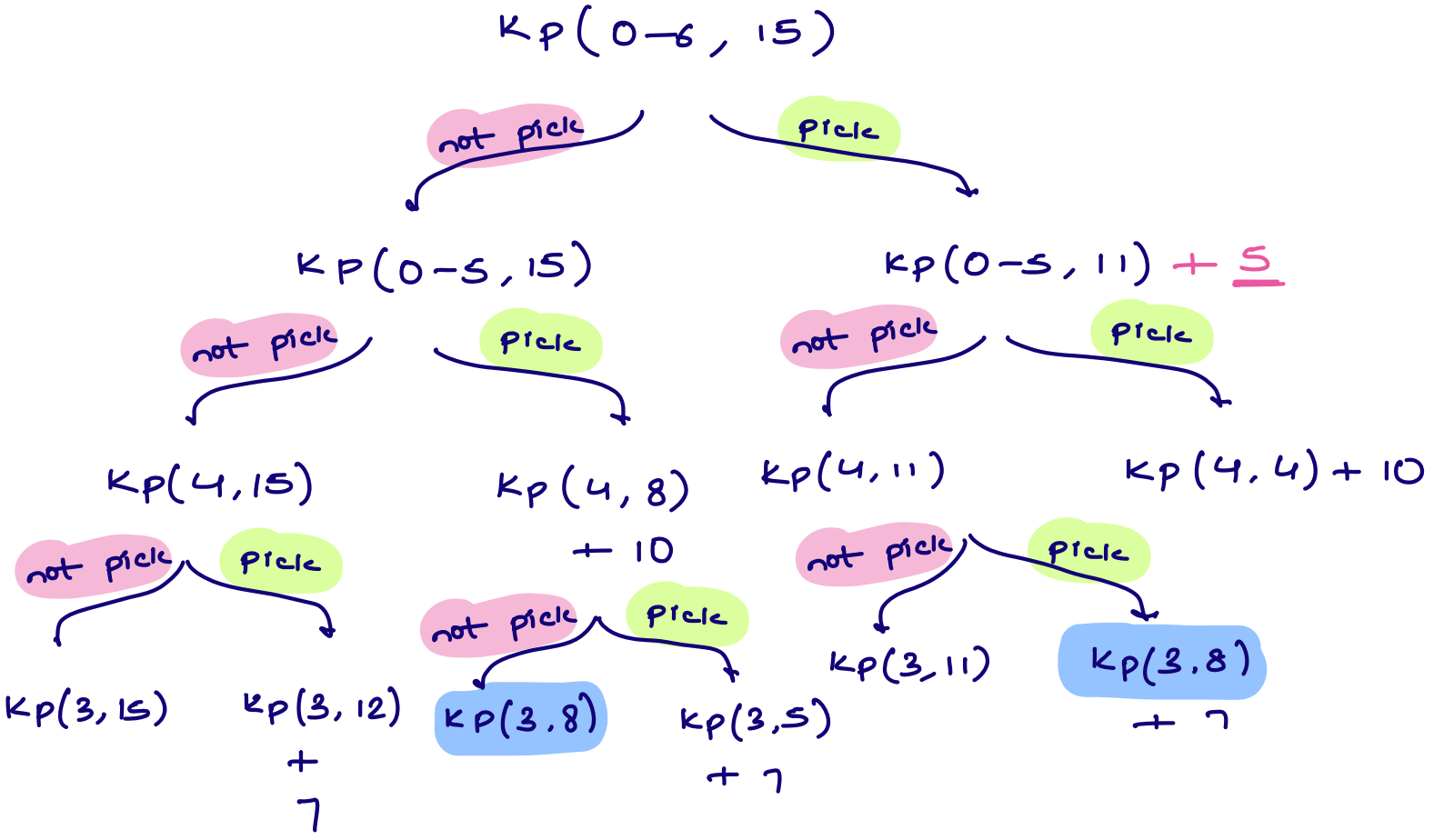
& return the ans which is providing the max value.

Cap = 15

N = 7

wt[] = { 4 1 5 4 3 7 4 }
0 1 2 3 4 5 6

val[] = { 3 2 8 3 7 10 5 }



Code →

```
int knapSack ( int i, int j , int [] w, int [] val )
{
  if ( i<0 || j ≤ 0 ) return −∞;
  int notpick = knapsack ( i−1, j , wt , val ) ;
  int pick = 0
if ( j ≥ wt[i] ){
        pick = knapsack ( i−1, j − wt[i] , w, val ) + val[i]
}
  return max ( not pick, pick )
}
```

n-1

K

Top down

```
int [][] dp = new int[n][k+1];
```

```
int knapsack ( i , j , wt , val )
{
    if (i<0 || j<=0) return 0;

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

    int val = knapsack ( i-1, j , wt, val)

    if ( j >= wt)
    {
        val = max( val , knapsack( i-1, j-wt[i] , wt, val) +val[i])
    }

    return dp[i][j] = val ;
}
```

```
int [] wt = {3  6  5  2  4}        cap=7
             0  1  2  3  4
int [] val = {12  20  15  6  10}
```

```
int dp[n+1] [k+1]
```

* Bottom Up Approach

dp[i][j] = max sum we can generate using i elements & bag capacity = j

val	wt	ele ↓ / cap →	0	1	2	3	4	5	6	7
		0	0	0	0	0	0	0	0	0
12	3	1	0	0	0	12	12	12	12	12
20	6	2	0	0	0	12	12	12	20	20
15	5	3	0	0	0	12	12	15	20	20
6	2	4	0	0	6	12	12	18	20	21
10	4	5	0	0	6	12	12	18	20	22

not pick 4th ele → we have to ask our first 3 ele to go & fill the bag of cap = 3 & generate max value

⇒ 12

pick 4th ele = 6 + ram bag cap = 1

ask 3 ele to go & fill the bag of cap=1 & gen max value

dp[i][j] = max value that we can generate using i elements & j as bag capacity

$$dp[i][j] = \max\left(dp[i-1][j],\ val[i-1] + dp[i-1][j-wt[i-1]]\right)$$

$dp[i-1][j]$ ↑ not pick

$val[i-1] + dp[i-1][j-wt[i-1]]$ if $(j - wt[i-1] \geq 0)$

TC : O(n * k)

SC : O(n * k)

```
* for (i=0; i<n; i++)
  {
    for (j=0; j<m; j++){
      if (i==0 || j==0) dp[i][j]=0
      else{
        rej = dp[i-1][j]
        sel = 0
        if ( j ≥ wt[i-1])
        {
          select = val[i-1] + dp[i-1][j- wt[i-1]];
        }
        dp[i][j]= max( rej, select);
      }
    }
  }
  return dp[N][K];
```

Unbounded Knapsack

Given N items each with a weight & value, find max value which can be obtained by picking items such that total weight of all items $\leq K$

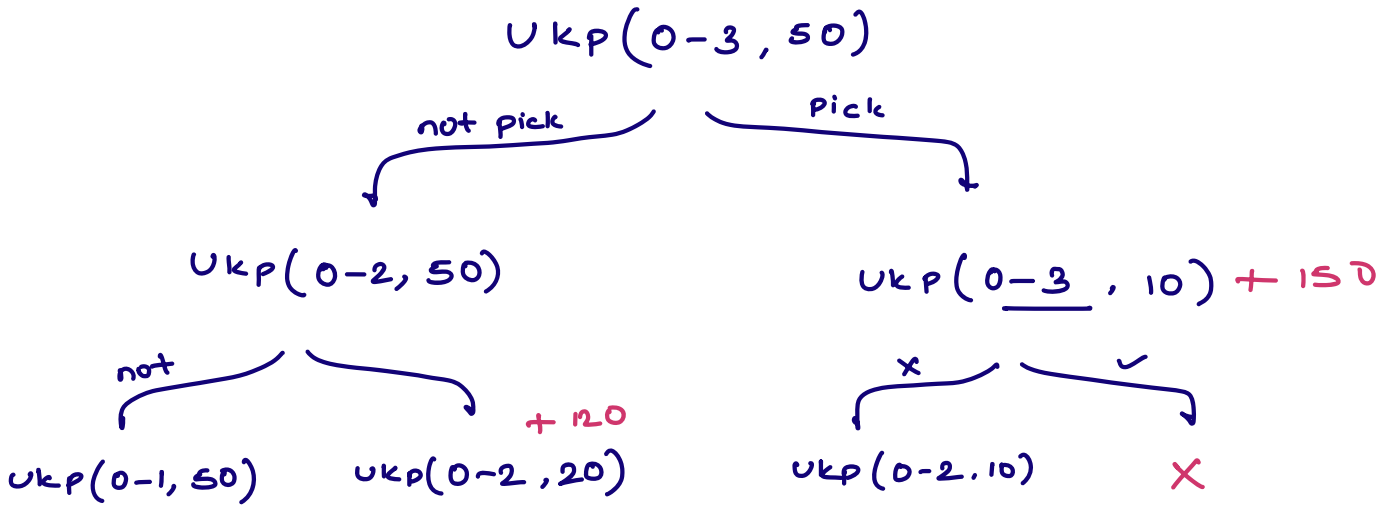
Note 1 :- Every item can be picked infinite no. of times

Note 2 :- We cannot take a part of item

Eg:- N = 4 items K = 50

Items:	0	1	2	3
Weight[]:	20	10	30	40
Value[]:	100	60	120	150

Ans = 300



```
for (i=0; i<n; i++)
    for (j=0; j<m; j++){
        if (i==0 || j==0) dp[i][j]=0
        else{
            rej = dp[i-1][j]
            sel = 0
            if ( j ≥ wt[i-1])
                select = val[i-1] + dp[i][j- wt[i-1]];
            dp[i][j]= max( rej, select);
        }
    }
}
return dp[N][K];
```

Q Given 2 strings, find the length of longest common subsequence.

S_1 = abc → {a, b, c, ab, ac, abc, bc}

S_2 = ace → {a, c, e, ac, ae, ace, ce}

Ans = 2

S_1 = abbcdgh

S_2 = bachegf

{ach, acg, bcg} Ans = 3

S_1 = abbcdgf

S_2 = achegf

{acgf} Ans = 4

S_1 = abbcdgf (indices 0 1 2 3 4 5 6)

S_2 = achegf (indices 0 1 2 3 4 5)

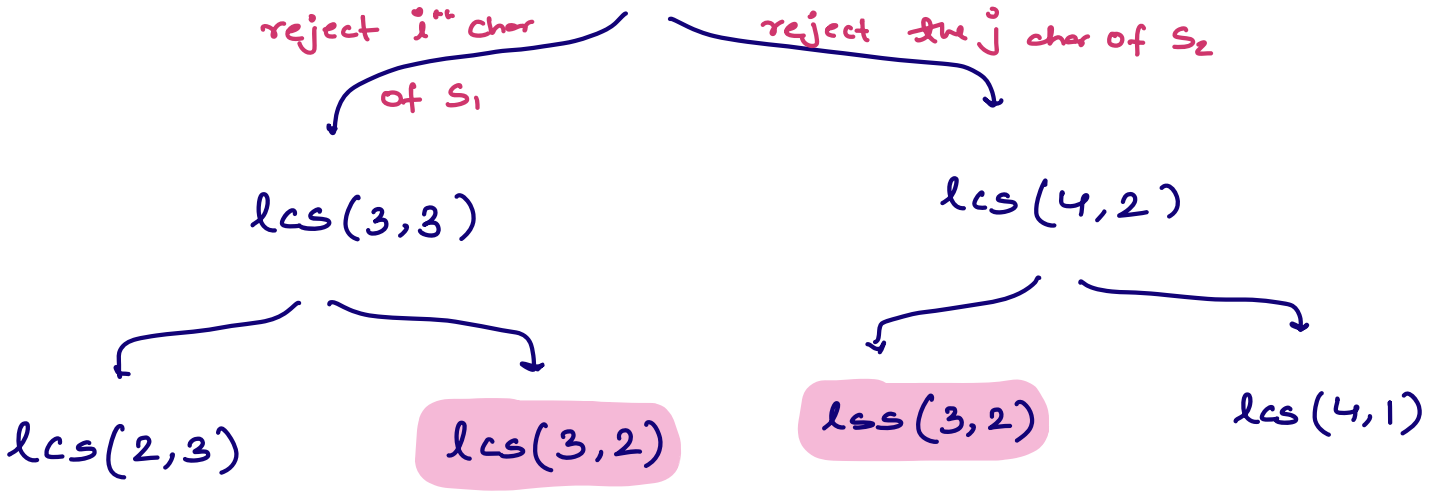
lcs(6, 5)

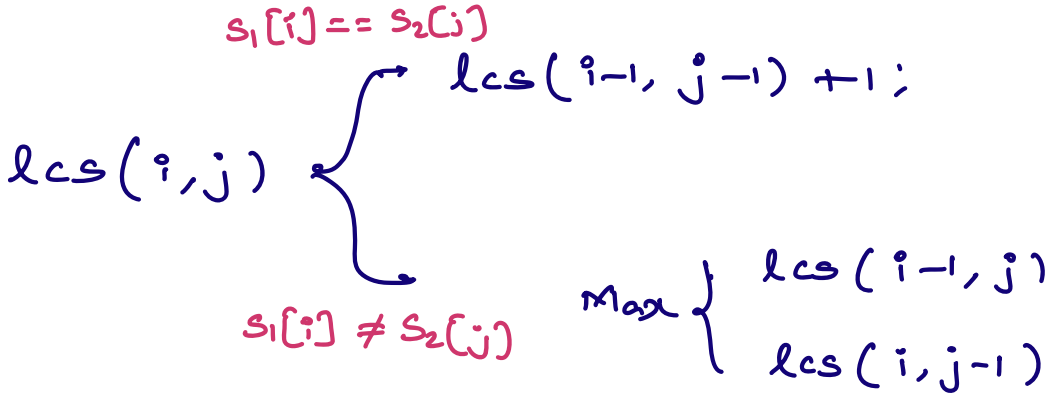
↓ if ($S_1[6] == S_2[5]$)

lcs(5,4) + 1

↓ $S_1[5] == S_2[4]$

lcs(4,3) + 1






```
int [][] dp = new int[n][m]
```

(→)

```
int lcs (string s1, string s2, i, j)
```

n-1 m-1

```
    if (i<0 || j<0) return 0;
    if (dp[i][j] != -1) return dp[i][j];
      ans = 0
    if ( s1[i] == s2[j]){
        ans = lcs ( s1, s2, i-1, j-1) + 1
    }
    else {
        int a = lcs (s1, s2, i-1, j);
```

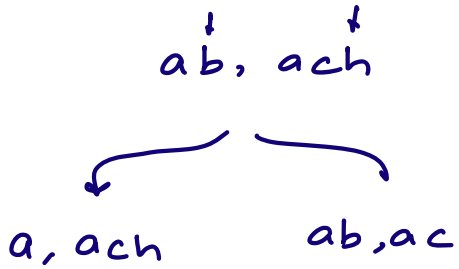
TC: O(n*m)

SC: O(n*m)

```
int b= lcs(s1, s2, i, j-1);

ans= Max(a,b)

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

dp[i][j] → max length of common subsequence in (0-i) of s_1 & (0-j) of s_2

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



```
for (i=0; i<n ; i++)
    for (j=0; j<m; j++){
        if ( s1[0] == s2[0]){
            dp[0][0] = 1;
        }
        else if ( j==0 ){
            if (s1[i] == s2[j] ) dp[i][j] = 1
            else  dp[i][j] = dp[i-1][j];
        else if (i==0)
            if (s1[i] == s2[j] ) dp[i][j] = 1
            else  dp[i][j] = dp[i][j-1];
        }
        else {
            if (s1[i] == s2[j])
                dp[i][j] == dp[i-1][j-1] + 1;
            else{
                dp[i][j] = max ( dp[i-1][j], dp[i][j-1]);
            }
        }
    }
}
return dp[n-1][m-1];
```

Q Given a string, find **longest palindromic subsequence**

A = "scalar" Ans = 3

A = "abcdefb" Ans = 3

A = "abdcgba" Ans = 5

palindrome of A = palindrome of reverse of A

S_1 = scalar
S_2 = ralacs
} find LCS → 3

A = abcdefb
B = bfedcba
} Ans = 3

A = abdcgba
B = abgcdba
} Ans = 5

10:02 → 10:12 pm

Q Given 2 strings S_1 & S_2, find min operations to be performed in S_1 so that it becomes equal to S_2.

Operations allowed

01. Insert → we can insert any character
02. Replace → we can replace any character
03. Delete → we can delete any character

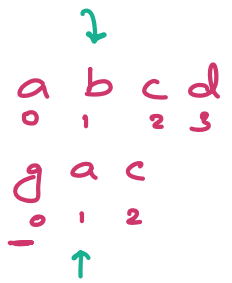
Eg:- S_1 = d f a c l (g)
S_2 = f g l

→ delete d
→ delete a
→ replac c

} Ans = 3

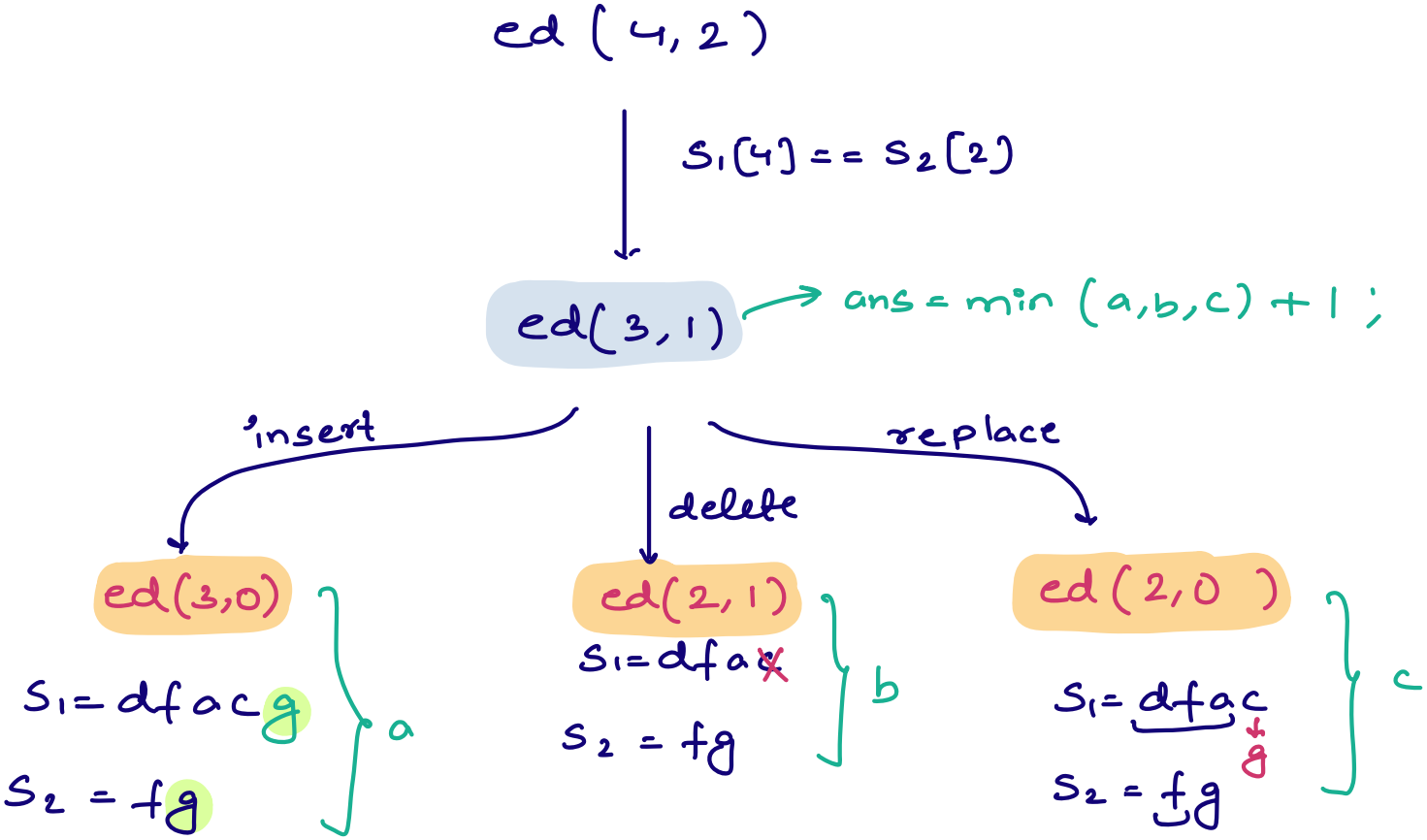
Eg:- S_1 = d f a x z
S_2 = f x z

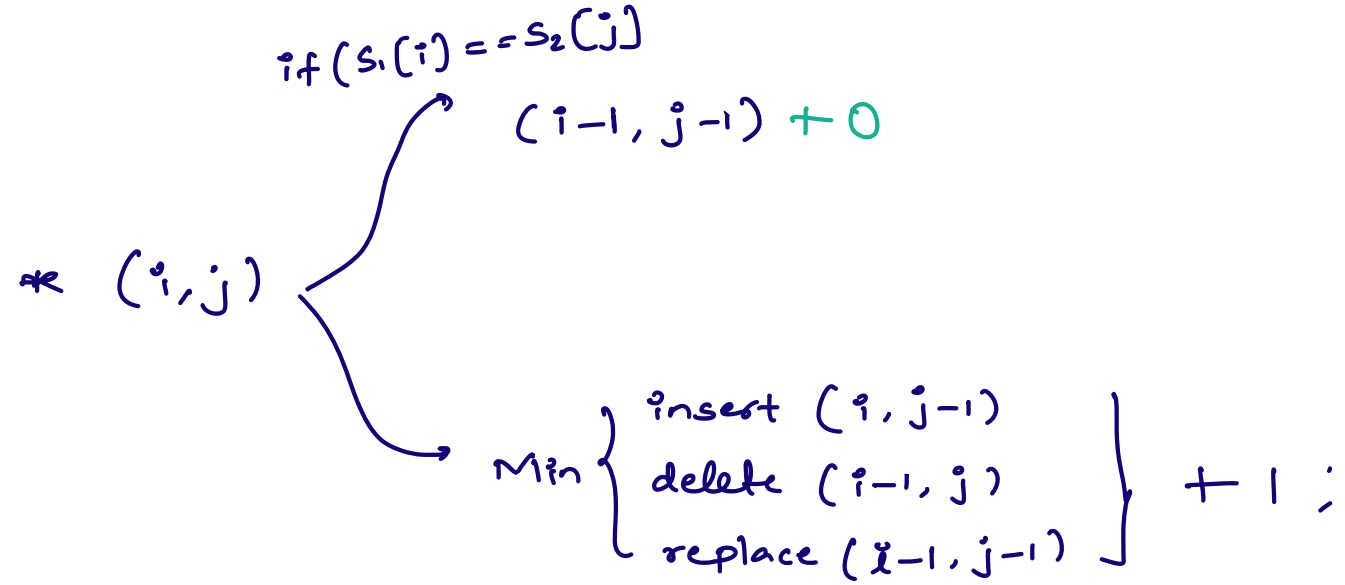
} Ans = 4



S_1 = d f a c l
0 1 2 3 4

S_2 = f g l
0 1 2





int [][] dp = new int [n] [m] ⟶ -1

```
int ed(string s1, s2, i, j)
{
    if (i<0 && j<0) return 0;
    if (i<0) return j+1      // j+1 no. of char to insert
    if (j<0) return i+1      // i+1 char to be deleted
    if (dp[i][j] != -1) return dp[i][j];
    int ans=0
    if (s1[i]==s2[j])
    {
        ans = ed(s1, s2, i-1, j-1);
    }
    else {
        int a = ed(s1, s2, i, j-1);   // insert
        int b = ed(s1, s2, i-1, j);   // delete
        int c= ed(s1, s2, i-1, j-1);  // replace
        ans = min(a,b,c)+1;
    }
    dp[i][j] = ans;
    return ans;
}
```

TC : O(n*m)

SC : O(n*m)