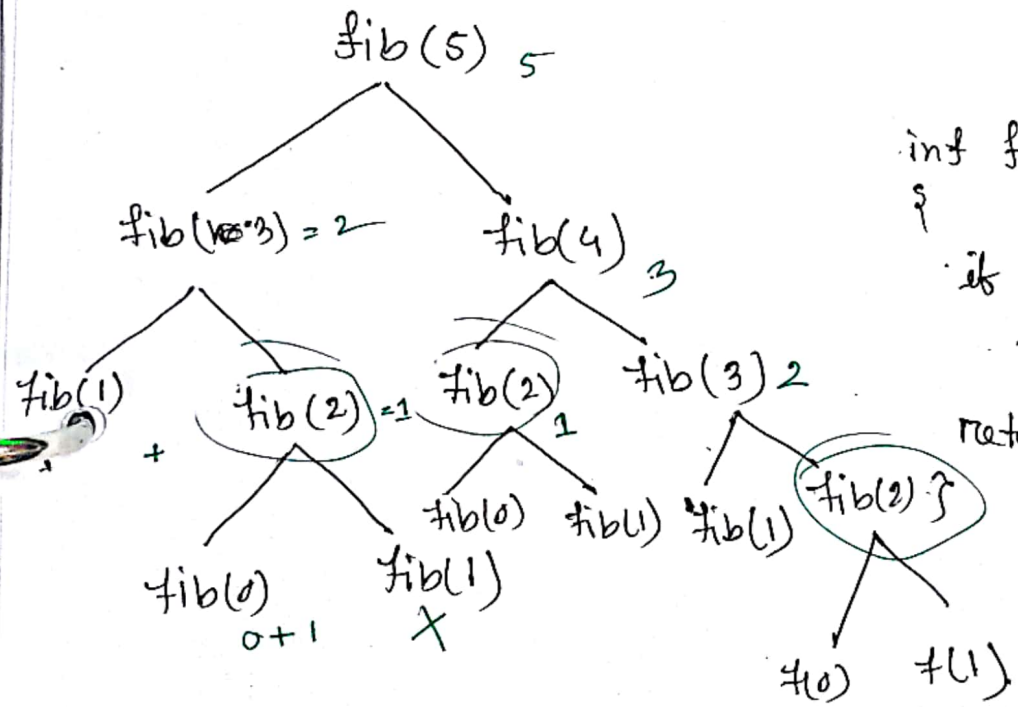
"Dynamic Programming"

1. Greedy Method
2. Dynamic Programming
   ↑
   Principle of Optimality.

Ex: Fibonacci

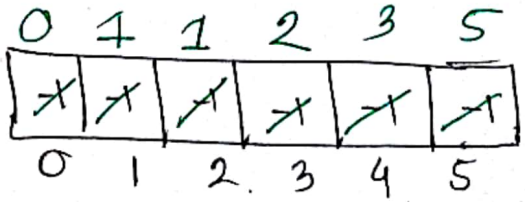
0, 1, 1, 2, 3, 5, 8, 13..
0 1 2 3 4 5 6 7..

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fib(n-2)+fib(n-1) & \text{if } n>1 \end{cases}$$




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

Global array F



fib (n) = n+1 calls
O(n)

"Memorisation" → Top down Approach

Tabulation Method: Bottom Up.

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fib(n-2)+fib(n-1) & \text{if } n>1 \end{cases}$$

F	0	1	1	2	3	5
	0	1	2	3	4	5
			i	i	i	i

fib (5)

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

"0/1 Knapsack Problem"

$m = 8$

$n = 4$

$x_i = 0/1$

$P = \{1, 2, 5, 6\}$

$w = \{2, 3, 4, 5\}$

$x = \{1, 0, 0\}$

1. Previous value copy..
2. Object profit + previous object value

$M=8 \quad P=\{1, 2, 5, 6\}$

$n=4 \quad w=\{2, 3, 4, 5\}$

P_i	w_i	V	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5	6	6	7	8

$V[i, w] = \max\{V[i-1, w], V[i-1, w-w[i]] + P[i]\}$

$V[4,1] = \max\{V[3,1], V[3, 1-5] + 6\}$

$3, -4 \rightarrow$ undifine.

$V[4,5] = \max\{V[3,5], V[3, 5-5] + 6]$

$5, \quad 0+6$

$V[4,6] = \max\{V[3,6], V[3, 6-5] + 6]$

$6 \quad 0+6$

$V[4,7] = \max\{V[3,7], V[3, 7-5] + 6\}$

$7 \quad 1+6$

$V[4,8] = \max\{V[3,8], V[3, 8-5] + 6\}$

$7, \quad 2+6$

x_1	x_2	x_3	x_4
0	1*	0	1

$8 - 6 = 2$

$2 - 2 = 0$.

Longest Common Sub sequence (LCS)

String 1 : a b c d e f g h i j

String 2 : c d g i

c d g i → longest common sub sequence.

d g i

g i

String 1 : a b c d e f g h i j

→ no (intersect হচ্ছে না তাই নেওয়া যাবে না)

: e c d g i

: a b c d e f g h i j

: e c d g i

e g i

c d g i → LCS

String 1 : a b d a c e

String 2 : b a b c e

b a c e] mutiple subsequence with same length.

a b c e

m A

b	d
1	2

n. B

a	b	c	d
1	2	3	4

		a	b	c	d
	0	1	2	3	4
0	0	0	0	0	0
b 1	0	0	1	1	1
d 2	0	0	1	1	2
			b		d

O(mxn)

if (A[i] = B[j]
LCS[i,j] = 1+LCS [i-1,j-1]
Diagonal down

else
Lcs [i,j] = max(Lcs [i-1,j], Lcs [i,j-1])

Stn 1: stone

Stn 2: longest

		l	o	n	g	e	s	t
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
s 1	0	0	0	0	0	0	1	1
t 2	0	0	0	0	0	0	1	2
o 3	0	0	1	1	1	1	1	2
n 4	0	0	1	2	2	2	2	2
e 5	0	0	1	2	2	3	3	3

o n e

if (A[i] = B[j])

LCS[i,j] = 1 + LCS[i-1, j-1]

else

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

"Coin Change Problem"
Dynamic Programming.

Find total number of ways:

⇒ Coins = {1, 3, 5}

Amount = 8

1; 1, 1, 1, 1, 1, 1, 1

1 1 1 1 1 3

3 3 1 1

5 3

5 1 1 1

} 5 ways.

Coins \ Amount	0	1	2	3	4	5	6	7	8
0	1	0	0	0	0	0	0	0	0
1	1	0+1 =1	0+1 =1	0+1 =1	1	1	1	1	1
3	1	1	1	1+1 =2	1+1 =2	1+1 =2	1+2 =3	1+2 =3	1+2 =3
5	1	1	1	2	2	2+1 =3	3+1 =4	3+1 =4	3+2 =5

If now > col, then copy the value from above

(i) Exclude new coin.

(ii) Include new coin

(iii) Add (i) + (ii).

```
for (i=0; i<=c.length; i++)
 {
  for (j=0, j<= amount; j++)
  {
   If (i==0 && j==0)
       Solution [i] [j] = 1;
   else
       If (c[i] > j)
          Solution [i] [j] = solution [i-1] [j];
       else
          Solution [i] [j] = Solution [i-1] [j] + Solution
                                  [i] [j - c[i]];
  }
}
```

" Minimum number of coins"

Coins = {1, 5, 7, 9}

Amount = 12

If now > col then copy the value from above

min (exclude new coin, 1 + include new coin)

	0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	2	3	4	5	6	7	8	9	10	11	12
5	0	1	2	3	4	min(5, 1+0=1) = 1	min(6, 1+1=2) = 2	min(7, 1+2=3) 3	min(8, 1+3=4) 4	min(9, 1+4=5) 5	min(10, 1+1=2) 2	min(11, 1+2=3) 3	min(12, 1+3=4) 4
7	0	1	2	3	4	1	2	min(3, 1+0=1) 1	min(4, 1+1=2) 2	min(5, 1+2=3) 3	min(2, 1+3=4) 2	min(3, 1+4=5) 3	min(4, 1+1=2) 2
9	0	1	2	3	4	1	2	1	2	min(3, 1+0=1) 1	min(2, 1+1=2) 2	min(3, 1+2=3) 3	min(4, 1+3=4) 2

```
for (i=0, i<=c.length, i++)
  {
  for(i=0, j<= amount, j++)
  {
   If(c[i] > j)
   {
   solution [i][j] = Solution [i-1] [j];
     }
else
   solution [i][j] = min (Solution[i-1][j], 1+ Solution[i][j-
                                                    c[i]] .
```