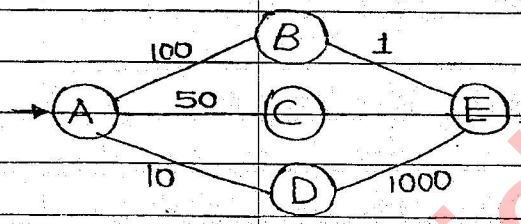
Chapter 4 : Dynamic Programming

DP	GT
considers all posibilities	less possibilities
always correct answer	sometimes correct
More time	less time



Greedy ADE
DP A→B→E

Greedy technique bothers about local optimization.
while dp bothers for global optimization.

Applications of Dynamic Programming -

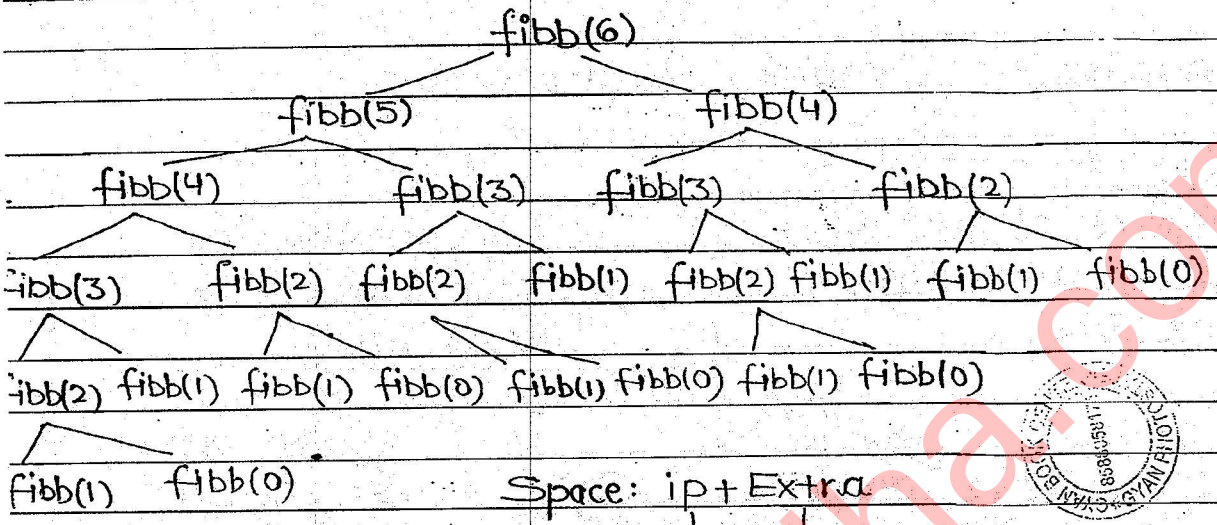
1. Fibonacci Series →

n	0	1	2	3	4	5	6	7	8	9	10
fibb	0	1	1	2	3	5	8	13	21	34	55

Recurrance rel^n →

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

```
fibb(n) {
    if (n==0) return 0; if (n==1) return 1;
    else
    a = fibb(n-1) ; b = fibb(n-1)
    c = a+b;  return c; }
```



Space: ip + Extra
2B nB $\Rightarrow O(n)$

In the above recursive tree, many functions are repeating (overlapping subproblems). Why to compute again & again? In dynamic programming, we will compute only distinct function because as soon as we compute any function, we will store its value in some data-structure, so that we can reuse it.

Q. How many distinct function calls are there in fibbonacci(n)?

Sol. n+1 functions

$\Downarrow$

$n \times O(1) \Leftarrow n \times 1$ addition

$\hookrightarrow O(n)$ [using dynamic programming]

Space

table

0	1	1	2	3	5		

fibb(0) fibb(1) fibb(2) fibb(3) fibb(4) fibb(5) fibb(6) . Array

DP also requires stack (same size). Normal programming will use the stack more frequently. DP uses the table more frequently.

fibbonacci series using dynamic programming

```
DP- fibb(n)
{
if(n==0) return 0;
if(n==1) return 1;
else
{
if (table[n-1] == NULL)
  table[n-1] = DP-fibb(n-1)
if (table[n-2]== NULL)
  table[n-2] = DP-fibb(n-2)
table[n] = table[n-1]+ table[n-2]
    return (table[n]);
  }
```

Space :- i/p + Extra

i/p : 2B

Extra : Stack NB + Table NB = O(n)

8588058175

2. Longest Common Subsequence =

Subsequence - of a given sequence is just the given sequence only in which 0 or more symbols are left out

Ex- S = (A, B, B, A, B, B)
1 2 3 4 5 6

SS_1 = (B, B, B, B)
2 3 5 6

SS_2 = (A, B, B, A, B, B)
1 2 3 4 5 6

SS_3 = ( ) ✓

SS_4 = (B, A, B, A) ✗
2 4 5

SS_5 = (B, B, A, B, B) ✓
2 3 4 5 6

Page No. 07
Date : 13/03/16

Common subsequence- Z is a common ss of two sequences x and y if z is subsequence to both x and y.

Ex1 X = (A, B, B, A, B, B)
Y = (B, A, A, B, A, A)

CS_1 = (A, A) CS_2 = (') CS_3 = (A)
LCS_4 = (A, B, A) LCS_5 = (B, A, B)

Ex2. X = (A, A, B, A, A, B, B)
Y = (B, A, B, A, B, B)

CS_1 = (A) CS_2 = (A, B) CS_3 = (A, B, A)
CS_4 = (A, B, A, B) LCS_5 = (A, B, A, B, B)

Q- How to find LCS of two given sequences-?

Sol Consider

X = (A B A B A B A)
Y = (B A A A B A)

LCS(7,6) = 1 + LCS(6,5) (matching)

⇩

1 + LCS(5,4) ⇨ 1 + LCS(4,3)

⇩

Ans: 5 ② ⇦ Max [LCS(3,3) / LCS(4,2)] (Not-matching)

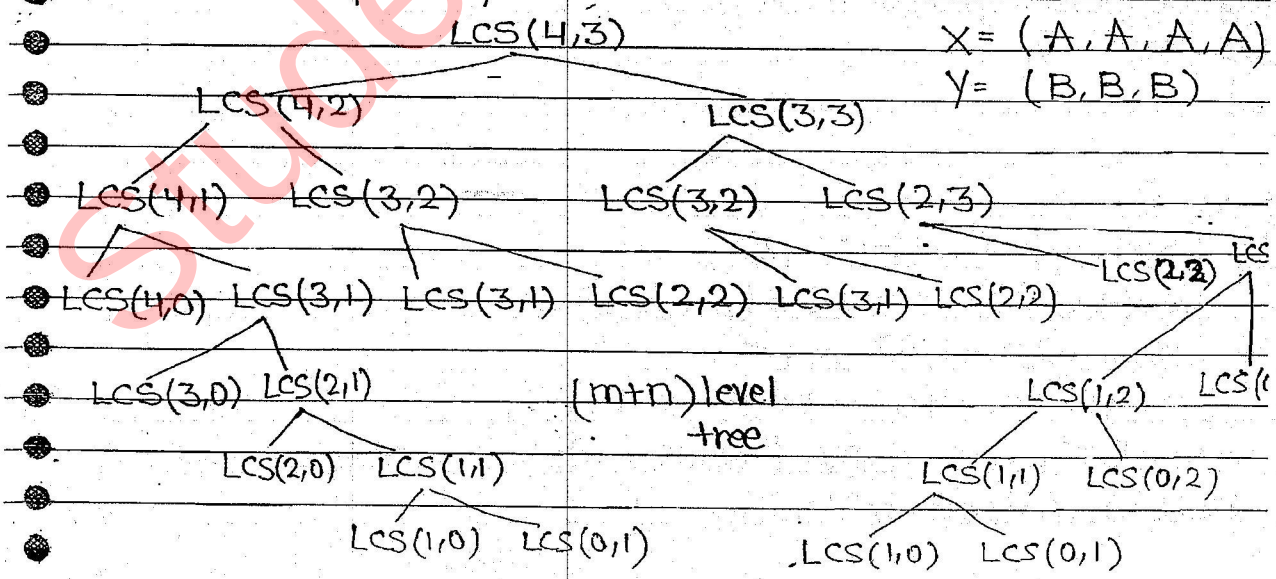
Recurrance Relation- Let LCS(m, n) = length of longest common subsequence possible where m is no. of symbols in X and n be no. of symbols in Y sequence.

$$LCS(m,n) = \begin{cases} 0 & \text{if } m=0 \text{ or } n=0 \\ 1 + LCS(m-1,n-1) & \text{if } x_m = y_n \\ \max\begin{cases} LCS(m,n-1) \\ LCS(m-1,n) \end{cases} & \text{if } x_m \neq y_n \end{cases}$$

* Recursive Program -

```
LCS(m,n) {
if (m==0 || n==0) return 0
else
if (x[m] == y[n])
return (1+LCS(m-1,n-1))
else
{
a = LCS(m,n-1)
b = LCS(m-1,n)
c = max(a,b)
return(c);
}
}
```

* Time complexity -



LCS(m,n) will generate (m+n) CBT (assume)

⇩

(m+n) function calls = $2^{m+n}-1$

⇩

2^{m+n} × Comparison ⇒ $O(2^{m+n})$

Time-complexity $O(2^m \cdot 2^n)$

Space-complexity = i/p + Extra

= (m+n)B + (m+n)(stack)

= O(m+n)

In above recursive tree, many function calls are repeating. So move to Dynamic Programming.

Q- How many distinct f^n calls are there in LCS(m,n)

Ans. (m+1)(n+1) calls

= mn × 1-comp. ⇒ mn × O(1) ⇒ O(mn)

Space = i/p + Extra

m+n (Stack m+n, Table mn) ⇒ O(mn)

Taking more space

Using dynamic programming-

```
LCS(m,n) {
   |
else {
 if (table[m][n-1]== NULL)
   table[m][n-1] = LCS(m, n-1);
 if (table[m-1][n] == NULL
     table[m-1][n] = LCS(m-1, n);
 table[m][n] = max (table[m][n-1], table[m-1][n];
  return (table[m][n]); } }
```

3. Matrix chain Multiplications →

Ex 1

A 2X3 — for 1 - element → 3 multiplications

B 3X4

C = AXB (2X4) — 2X4X3 = 24 multiplications done

Ex2 A(3X4) B(4X5) C(5X3)

D = ABC No. of mul?

(i)X AB (3X5) 3X5X4 = 60

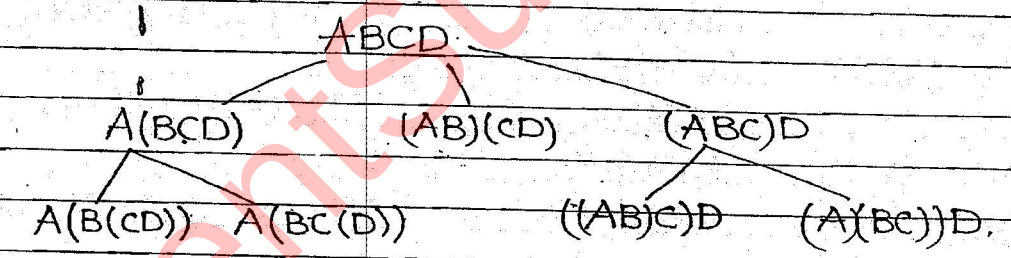
(AB)C (3X3) 3X3X5 = 45 Total = 105

(ii)✓ BC (4X3) 4X3X5 = 60

A(BC) (3X3) 3X3X4 = 36 Total = 96

(iii)X CA (5X4) 5X4X3 = 60

B(CA) (4X4) 4X4X5 = 80 Total = 120



Ex3 A (1X3) B (3X5) C (5X4) D (4X2)

E = ABCD NO. of mul?-

(i) (CD) (5X2) 5X2X4 = 40

B(CD) (3X2) 3X2X5 = 30

A(B(CD)) (1X2) 1X2X3 = 6 Total = 76.

(ii) (BC) (3X4) 3X4X5 = 60

(BC)D (3X2) 3X2X4 = 24

A(BC)D) (1X2) 1X2X3 = 6 Total = 90

(iii) AB (1X5) 1X5X3 = 15

(AB)(CD) 1X2 1X2X5 = 10 Total = 65

Page No.
Date : / /

iv) (AB)C (1X4) 1X4X5 = 20 (Optimal 2)
((AB)C)D (1X2) 1X2X4 = 8 Total = 43
v) A(BC) (1X4) 1X4X3 = 12
(A(BC))D (1X2) 1X2X4 = 8 Total = 80

Ex-4 A B C D E = ABCD
1X3 3X1 1X2 2X1 No. of mul?

AB (1X1) 1X1X3 = 3
(AB)C (1X2) 1X2X1 = 2,
(AB)C)D (1X1) 1X1X2 = 2, Total = 3+2+2 = 7
BC (3X2) 3X2X1 = 6
A(BC) (1X2) 1X2X3 = 6
A(BC))D (1X1) 1X1X2 = 2 Total = 6+6+2 = 14
CD (1X1) 1X1X2 = 2,
(AB)(CD)) 1X1X1 = 1 Total = 2+3+1 = 6 (Optimal)
i) B(CD) (3X1) 3X1X1 = 3
A(B(CD)) 1X1 1X1X3 = 3 Total = 3+3+2 = 8
) (BC)D (3X1) 3X1X2 = 6
A((BC)D) 1X1 1X1X3 = 3 Total = 6+6+3 = 15

ABCDE

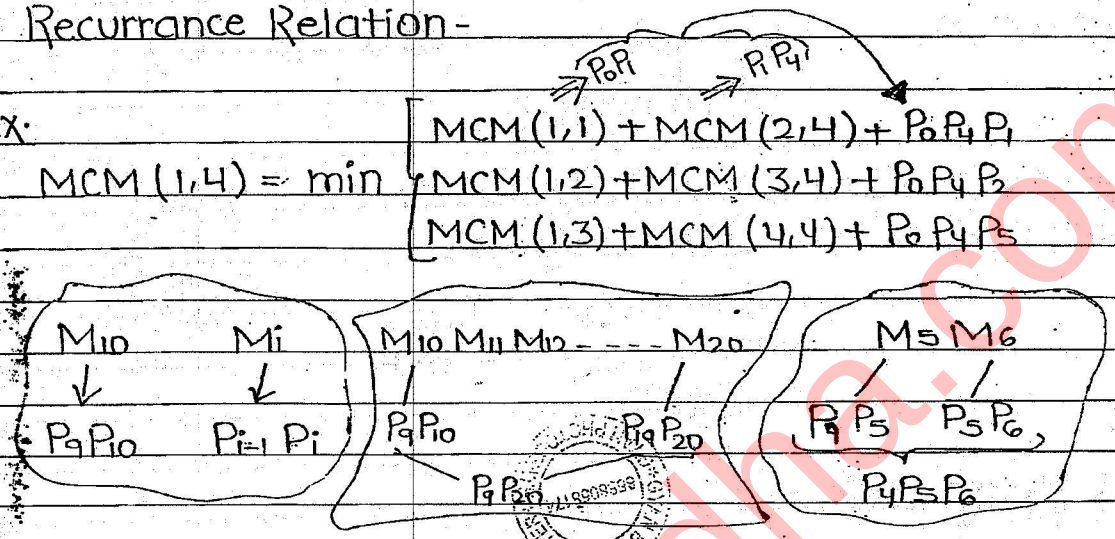
A(BCDE)	AB(CDE)	(ABC)DE	(ABCD)E	
5-ways	2 ways	2 ways	5 ways	= 14 ways
1X5	1X2,	2X1	5X1	

1 ⇒ 1
2 ⇒ 1
3 ⇒ 2
4 ⇒ 5
5 ⇒ 14

≡ Recurrance Relation -

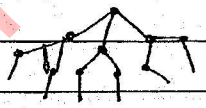
Ex.

$$MCM(1,4) = \min \begin{cases} MCM(1,1) + MCM(2,4) + P_0P_4P_1 \\ MCM(1,2) + MCM(3,4) + P_0P_4P_2 \\ MCM(1,3) + MCM(4,4) + P_0P_4P_5 \end{cases}$$



$$MCM(i,j) = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k < j} [MCM(i,k) + MCM(k+1,j) + P_{i-1}P_jP_k] & \text{if } i \neq j \end{cases}$$

MCM (1, n)


(4 Matrices)

⇓

m-level-n array tree

Total function calls = $(n-1)(n-2)(n-3)\cdots 2\times 1 = (n-1)!$

Time complexity = $(n-1)! \times O(n)$

= $n!$ ↳ finding minimum out o

Space complexity = i/p + extra

= $4nB + nB \Rightarrow O(n)$

(2+2)n Stack

In the above recursive tree, many function calls are repeating, so dynamic programming.

Q- How many distinct function calls are there in MCM (i, j) ?

Sol- n^2 ways M(i, j). (actually $n^2/2$)

$n \times n$ (change)

Time complexity = $n^2 \times O(n) = O(n^3)$

Space complexity = i/p + Extra

4nB | Stack nB | Table n^2B

$= O(n^2)$

1. 0/1 knapsack :-

n=3 M=10

Object	obj 1	obj 2	obj 3
Profit	80	45	50
Weight	7	4	6

With Greedy technique, 1,0,0, profit = 80, fails, so move to dynamic programming, which will give always correct answer, as it covers all possibilities.

Using dynamic programming -

obj	ob1	ob2	ob3	ob4	ob5
profit	20	15	70	60	30
weight	3	7	3	5	3

M = 15

0/1KS(m,n) = maximum profit got when capacity of knapsack is m and no. of objects are n.

(Giving some name to the problem)

Page No. 1
Date : 19/03/16

≐ In dynamic programming, the no. of levels in the tree are more comparing to Divide & conquer as it covers all the possibilites.

OIKS (15,5) = maximum profit get when capacity of knapsack is 15 & objects are 4.

→ Termination condⁿ:

$$OIKS(m,0) = 0 \qquad OIKS(0,n) = 0$$
$$OIKS(0,0) = 0$$

→ If $w_n > m$

$$OIKS(m,n) = OIKS(m,n-1) \text{ if } w_n > M$$

→ If $w_n \le m$

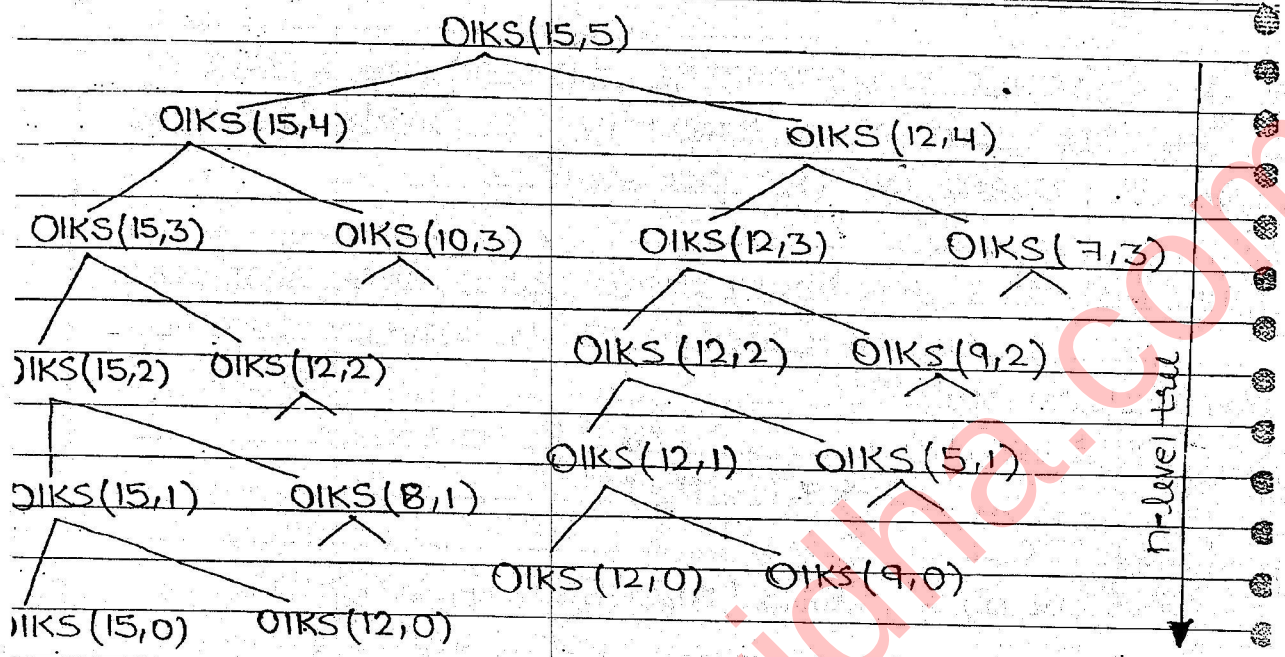
$$OIKS(m,n) = \max \begin{pmatrix} OIKS(m-w_n, n-1) + P_n \\ OIKS(m, n-1) \end{pmatrix} \text{ if } w_n \le$$

≐ Recurrance Relation -

$$OIKS(m,n) = \begin{cases} 0 & \text{if } m=0 \text{ or } n=0 \\ OIKS(m,n-1) & \text{if } w_N > M \\ \max \begin{pmatrix} OIKS(m-w_n, n-1) + P_n \\ OIKS(m, n-1) \end{pmatrix} & \text{if } w_n \le \end{cases}$$

≐ Recursive Program -

```
OIKS (m,n) {
  if (m==0 || n==0)  return 0;        ] Stopping purpo
  else if (w[n] > m)                  ] Best case O(
     return (OIKS (m, n-1));
  else {                              ] Worst case
  a = OIKS (m-w[n], n-1) + P[n]
  b = OIKS (m, n-1)
  c = max (a, b)
  return c; } }
```



Time complexity :-

OIKS(m,n) ⇒ n-level CBT

$2^n - 1$ nodes / function calls.

$2^n \times 1$-comparison ⇒ $O(2^n)$

Space complexity :-

i/P + extra.

6nB (2+2+2) + O(n) Stack (n-level tree) ⇒ O(n)

In the above recursive tree, some functions calls are repeating, so dynamic programming. (very less repetitions)

Q- How many distinct function calls are there?

OIKS(15,5)

14 13 12 11 ... 1 0 — 15+1

4 3 2 1 0 — 5+1

OIKS(m,n)

(m+1)(n+1) calls

= O(mn) (Time complexity)

Using dynamic programming

Space = i/p + extra
GnB + Stack + table
GnB + nB + mnB ⇒ O(mn)

∴ As less no. of functions are repeating, O(mn) is close to $O(2^n)$. It is a NP-complete proble

5. Sum of Subset -

i/p: An array of n-elements and integer m
o/p: Find subset of n-elements whose sum is m.

Ex

1	2	3	4	5	6	7	8

A = (50, 20, 40, 60, 80, 70, 10, 30) M = 100

S_1 = (50, 40, 10) ✓

S_2 = (20, 80) ✓

SOS(M,N) - finding a sum of subset from given array of N elements so that sum is M.

→ Termination cond^n -

SOS(0,n) = return S as $S = \phi$
SOS(0,0) = return S
SOS(m,0) = Error

→ If $w_N > m$
SOS(M,N) = SOS(M, N-1)

→ Else
SOS(M,N) = { SOS(M-w_N, N-1) ; S = S U w_N
SOS(M, N-1) }

∴ Recurrance Relation -

$$SOS(M,N) = \begin{cases} S & \text{if } M=0 \\ \text{error} & \text{if } N=0 \\ SOS(M,N-1) & \text{if } W_N > M \\ SOS(M-W_N, N-1), S=S\cup W_N & \text{if } W_N \le M \\ SOS(M,N-1) & \end{cases}$$

$\rightarrow$ n-level tree.

Time complexity = $2^n \times O(1) = O(2^n)$

Space complexity = $nB + nB = O(n)$

With dynamic programming -

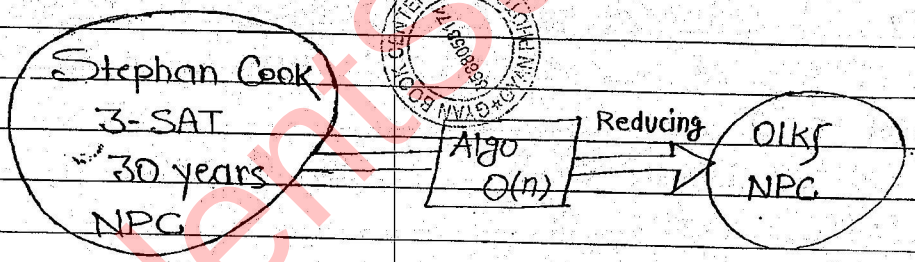
SOS(M,N) $\Rightarrow$ mn distinct function calls

$mn \times O(1)$

$\Rightarrow O(mn)$ (Time complexity)

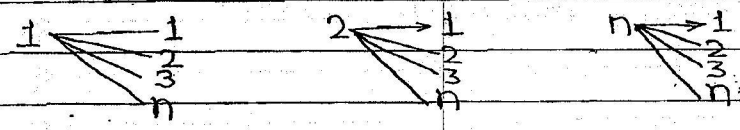
Space complexity = $n + n + mn = O(mn)$.

∴ It is also one of the NP-complete problem because of the recursive repetitions.



0/1 Knapsack problem can be polynomially reduced to sum of subset problem. So, both are closely related. Actually sum of subset problem is reduced to 0/1 Knapsack. So, NPC.

6. All Pair Shortest Path -



Floyd-Warshall Algorithm $O(n^3)$

Works for (+ve edge weights ✓
-ve edge weights ✓
-ve edge cycle ✗)

7. Optimal Cost Binary Search Tree -

i	0	1	2	3	4	5
p_i		p_1	p_2	p_3	p_4	p_5
q_i	q_0	q_1	q_2	q_3	q_4	q_5

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l \qquad \text{[sum of probabilities]}$$

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j \end{cases}$$

Running time = $\Theta(n^3)$

8 Travelling Salesperson problem -

Problem of determining shortest closed tour that connects a given set of n points in the plane. The general problem is NP-complete and its solution is therefore beleived to require more than polynomial time.