Unit-1

Page No.
Date: 2 5 24

Algorithm :-

Step by step procedure to find a Sol^{n} to given problem

* Features :-

i) Input [zero or more]
ii) Output [Atleast one output]
iii) Finiteness [Should terminate, finite steps]
iv) Definiteness [Every statement should unambigous]
v) Effectiveness [Statements or calculation can be solved with the help of pen & paper.

ex:- Sum of 3 Numbers
GCD [HCF] of 2 Numbers
Largest of 3 Numbers

Analysis of Algorithm :- [EoE]

To find the Complexity in term of time & Space that is how much time & Space is required to complete the execution of an Algorithm.

There are basically two ways of Analysing any Algorithm.

a) Priori Analysis [Theoritical Analysis]

* Machine independent
* Language independant
* Before implementation

b) Postpriori Analysis [Practical Analysis]

* Machine dependent
* Language dependent
* After implementation.

Ex:-

	O(1)	O(n)	$O(n^2)$	$O(\log_2 n)$	$O(n\log_2 n)$	$O(n^3)$
n=1	1	1	1	0	0	1
n=4	1	4	16	2	8	64
n=10	1	10	100	0.3010	3.010	1000
n=100	1	100	10000	2	200	1000000
	Constant			Faster way		

a) Priori Analysis:

This type of Analysis is always done in term of mathematical or equations. These Mathematical Func or eq^n are known as "Asymptotic Notations"

for doing a priori Analysis usually, following Notation are used -

a) Big oh (O) → (Worst Case) (Upper bound)
b) Omega (Ω) → (Best Case) (Lower Bound)
c) Theta (Θ) → (Avg Case) (Tight Bound)

=> Priori Analysis done in the term of worst case.

a) Big oh (O) :-

A fun^{c} f(n) is said to be big oh (O) of another fun^{c} g(n) iff ∃ (There exists) a positive constant c & n

s.t. $f(n) \leq c * g(n)$ ∀ $c, n > 0$ & $n \geq n_1$

=> $f(n) \in O(g(n))$

Ex:- $f(n) = 2n+2$

$g(n) = 2n^2$

n	f(n)	g(n)	Relation (T/F)
1	4	2	F
2	6	8	T
4	10	32	T
10	22	200	T

$2n+2 \leq 2(n^2)$ s.t. $n \geq 2 \rightarrow (n_1)$

$f(n) \in O(2n^2)$

$f(n) \in O(g(n))$ ∀ $c=1, n=2$

b) Omega (Ω):-

A fun^c $f(n)$ is said to be (Ω) of $g(n)$ iff $\exists$ 2 positive constants c & n_0 s.t.

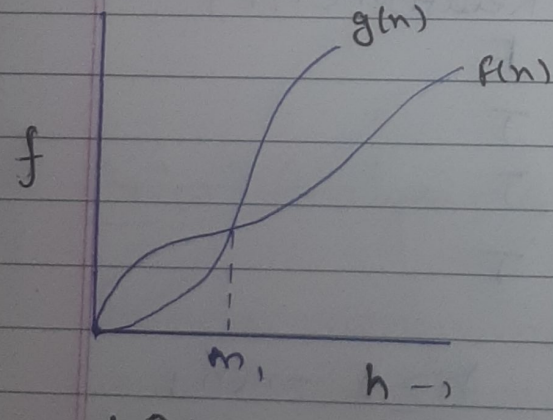
$f(n) \geq c * g(n)$ $\forall$ $c, n > 0$ & $n \geq n_1$

c) Theta (Θ):-

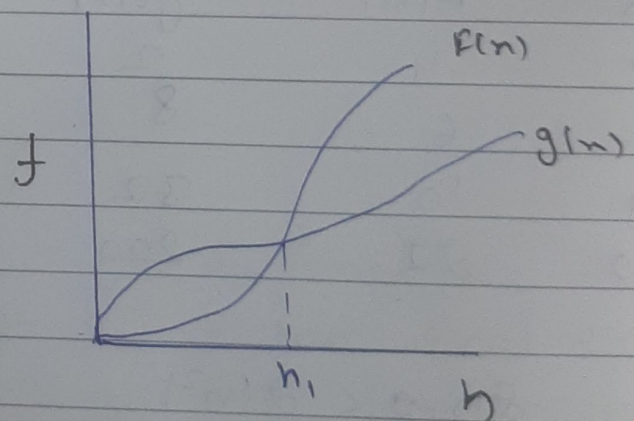
A fun^c $f(n)$ is said to be Θ of $g(n)$ iff $\exists$ 3 positive constants c_1, c_2 & n_0.

s.t. $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

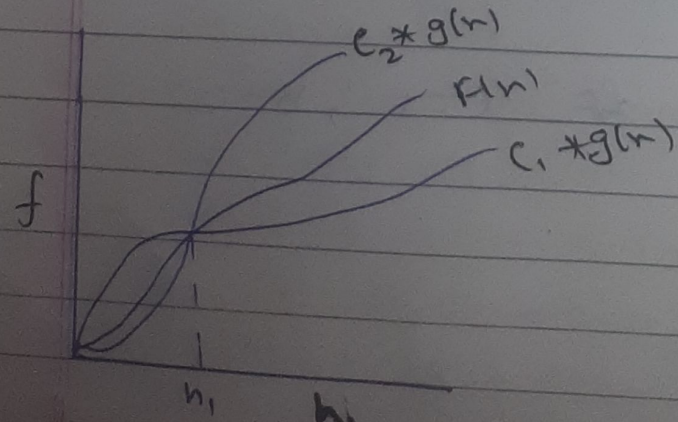
$\forall$ $c_1, c_2, n \geq 0$ & $n \geq n_1$



(O)



(Ω)



(Θ)

Merge Sort Algo :-

Step-1 : If it is only one element in the list, Consider it already sorted, so return.

Step-2 : Devide the list recursively into two halves untill it can no more be devided.

Step-3 :- Merge the Smaller lists into new list in sorted order.

Quick Sort Algo :-

Quick Sort is a sorting algorithm based on the devide and conquer Algorithm where -

1. An Array is devided into Subarrays by Selecting a pivot element (element Selected from the array

2. while deviding the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.

2. The left and right Subarrays are also devided using the same approach. This process continue

Page No.
Date :

until each subarray contains a single element

3. At this point, elements are already sorted. Finally, elements are combined to form a sorted array.

Binary Search Algo :-

- Divide the search space into two halves by finding the middle index "mid".
- Compare the middle element of the search space with the key.
- If the key is not found at middle element, choose which half will be used as the next search space
  - If the key is smaller than the middle element, then the left side is used for next search.
  - If the key is larger than the middle element, then the right side is used for next search.
- This process is continued until the key is found or the total search space is exhausted.

Recursive fun^c → fun^c calling itself in term of certain value (task & activity are same)

Recurrence Relation :-

A relation in which RHS term of relation or eq^n is represented in the form of LHS term.

Ex:- $T(n) = T(n-1) + n$

$T(n) = T(n/2) + 1$

$T(n) = T(n-1) + K$

$fact(n) = n \times fact(n-1)$

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

* Let find Complexity of $T(n) = T(n-1) + n$ -- i)

{ $T(n-1) = T(n-2) + (n-1)$ -- ii }

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

{ $T(n-2) = T(n-3) + (n-2)$ }

$T(n) = T(n-3) + (n-2) + (n-1) + n$ -- iv)

⋮

$T(n) = T(0) + 1 + 2 + 3 + \dots + (n-1) + n$

{ $T(0) = 0$ } let consider

$T(n) = 0 + 1 + 2 + 3 + \dots + n$

{ Sum of n term }

Page No.
Date :

$$T(n) = \frac{n(n+1)}{2}$$

$$T(n) = \frac{n^2}{2} + \frac{n}{2} \quad \left\{ \frac{n^2}{2} >>> n/2 \therefore n/2 = 0 \right\}$$

Neglect $T(n) = \frac{1}{2} \cdot n^2$

$$T(n) = c \cdot n^2 \quad \{ c = \text{const.} \}$$

$\{ f(n) \leq c.g(n), \; f(n) \in O(g(n)) \}$

$$\boxed{T(n) \leq c.n^2}$$

$$\boxed{T(n) \in O(n^2)}$$

Ex:- $T(n) = T(n/2) + k$ -- i)

$\{ T(n/2) = T(n/4) + k \}$

$T(n) = T(n/4) + k + k$ } -ii)

$\{ T(n/4) = T(n/8) + k \}$

$T(n) = T(n/8) + k + k + k$

$T(n) = T(n/2^3) + 3k$

General form $\{ T(n) = T\left(\frac{n}{2^p}\right) + p.k$ -iii) $\}$ upto p no(s) of Term

let $\frac{n}{2^p} = 1$, $n = 2^p$ Let $\log_2$ both side

$\log_2 n = \log_2(2^p)$

$\boxed{p = \log_2 n}$

$T(n) = T(1) + p.k$

{ let assume $T(1) = 0$ }

$T(n) = p.k$

$T(n) = k.\log_2 n \quad \{ c = k = \text{const.} \}$

$T(n) = c * \log_2(n)$

$\{ f(n) \leq c.*g(n) \;,\; f(n) \in O(g(n)) \}$

$\boxed{T(n) \leq c.\log_2(n)}$

$\boxed{T(n) \in O(\log_2(n))}$

ex:- $T(n) = 2T(n/2) + n \quad \text{--- i)}$

$\{ T(n/2) = 2T(n/4) + (n/2) \}$

$T(n) = 2[2.T(n/4) + n/2] + n$

$T(n) = 2^2.T\left(\frac{n}{2^2}\right) + 2n \quad \text{--- ii)}$

$\{ T(n/4) = 2T(n/8) + (n/4) \}$

$T(n) = 2^2.[2T(n/8) + (n/4)] + 2n$

$T(n) = 2^3.T\left(\frac{n}{2^3}\right) + 3n \quad \text{--- iii)}$

Page No.
Date :

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

General form -

$$T(n) = 2^p T\left(\frac{n}{2^p}\right) + p.n \quad \text{--- iv}$$

let $\frac{n}{2^p} = 1$ $\quad p = \log_2 n$ {both side $\log_2$}

$$T(n) = 2^{\log_2 n} * 1 + p.n$$
$$= \cancel{1} + n.p$$
let 0

$$T(n) = n.p$$
$$= n\log_2(n) * 1$$

$$T(n) = c.\, n(\log_2 n)$$

$\{ f(n) \leq c.g(n),\ f(n) \in O(g(n)) \}$

$$\boxed{T(n) \leq c.\, n(\log_2 n)}$$

$$\boxed{T(n) \in O(n\log(n))}$$

Unit - 2

Various Design Methods of an Algorithm :-

1. Devide and Conquer → Binary Search, Merge Sort, Quick Sort, Multiplication of 2 n bit no, Matrix Multiplication (Strassen's)
2. Greedy Method
3. Dynamic programming
4. Back Tracking
5. Branch & Bound

a) Devide & Conquer Method :-

Devide & Conquer Algorithm is a problem Solving technic used to Solve problems by deviding the original problem into Sub problems, Solving them individually an then Merge / Combine those Solutions to find th Solutions of the original problem.

The basic idea of D & C approach is to recurrsively devide the problem into smaller Sub problems until they become simple enough to be solved directly.

Once the solutions to the Sub problems are found, they all are then Combined to pro the overall Solution.

=) This Algo. Can be devided into 3 steps-

Page No.
Date :

Step-1. Deviide :-

Break down the original problem into smaller sub problem. Where Each Sub problem Shoulel represent a part of original problem.
The Goal is to deviide the problem untill no further devision is possible.

Step-2. Conquer :-

Solve each of the smaller sub problems independently and if a sub problem is small enough often refered to as the base case, we solve ~~it~~ it directly.

The Goal is to find solutions of sub problems independently.

Step. 3:- Merge :-

Once the smaller sub problems are Solved, we recurresively combine their solution to get the solution of larger problem.

The Goal is to formulate a solution of the original problem by merging the results from the sub problems.

* Charachteristics [Form cite, read only]

Solving Recurrence [Binary Search]:-

$$T(n) = T(n/2) + 1 \rightarrow (T(1))$$
$$= T(n/4) + 1 + 1$$
$$= T(n/8) + 1 + 1 + 1$$
$$= T(n/2^3) + 3$$

+ -- upto p terms

$$T(n) = T(n/2^p) + p$$

$\{ n = 2^p$

$$T(n) = T(1) + \log_2 n \quad \{ p = \log_2 n$$

$$T(n) \leq c * \log_2 n$$
$$\boxed{T(n) \leq c * g(n)}$$

$$\boxed{T(n) \in O(\log_2 n)}$$

* Merge Sort:-

$$T(n) = T(n/2) + T(n/2) + n$$
$$T(n) = 2T(n/2) + n$$

$$= 2[2T(n/4) + n/2] + n$$
$$= 4T(n/4) + n + n$$
$$= 8T(n/8) + n + n + n$$
$$[T(n) = 2^3 T(n/2^3) + 3n]$$

upto p terms

$$T(n) = 2^p T(n/2^p) + p*n$$

$\{ n = 2^p, \quad p = \log_2 n \}$

$$T(n) = n.T(1) + n\log_2 n$$

$$\boxed{T(n) \leq c*(n\log_2 n)}$$

$$\boxed{T(n) \in O(n\log_2 n)}$$

✱ Quick Sort :-

$\Rightarrow$ $T(n) = \cancel{T(0)}^{0} + T(n-1)$

$$T(n) = T(n-1) + n, \quad n \geq 2$$

$\Rightarrow$ $T(n) = T(n-1) + n$

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

$= T(n-3) + (n-2) + (n-1) + n$

$\vdots$

upto $(n-1)$ numbers

$T(n) = T[n-(n-1)] + [n-(n-2)] \cdots + (n-1) + n$

$= T(1) + T(2) + \cdots + (n-1) + n$

$\vdots$

upto n times

$T(n) = T(0) + T(1) + \cdots n$

$= 1 + 2 + 3 + \cdots n$

$$T(n) = \frac{n(n+1)}{2}$$

$$\therefore T(n) = \frac{n^2}{2} + \frac{n}{2} = \frac{1}{2}[n^2+n]$$

$\{ n^2 >>> n \ , \ n \simeq 0 \}$

$$T(n) = \frac{1}{2} \cdot n^2$$

$$\left[\begin{array}{l} T(n) \leq c * n^2 \\ T(n) \in O(n^2) \end{array}\right]$$

* Average Case Time Complexity for quick Sort

$$T(n) = (n+1) + \frac{2[T(0)+T(1)+T(2)+\ldots+T(n-1)]}{n}$$

$$nT(n) = n(n+1) + 2[T(0)+T(1)+T(2)+\ldots+T(n-1)] \quad \text{--- i)}$$

put $(n-1)$ for n

$$(n-1)T(n-1) = n(n-1) + 2[T(0)+T(1)+\ldots+T(n-2)] \quad \text{--- ii)}$$

Substract i) - ii)

$$n(T(n)) - nT(n-1) + T(n-1) = 2n + 2[T(n-1)]$$

$$n[T(n) - T(n-1)] + T(n-1) = 2[n + T(n-1)]$$

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

$$\boxed{nT(n) = 2n + (n+1)T(n-1)} \quad \text{--- iii)}$$

Page No.
Date :

$$T(n) = 2 + \left(\frac{n+1}{n}\right) T(n-1)$$

$$\left\{ \frac{T(n)}{(n+1)} = \frac{2}{(n+1)} + \frac{T(n-1)}{n} \right\} \text{ - iv}$$

Using Substitution put '(n-1)' for 'n' in iv

$$\left\{ \frac{T(n-1)}{n} = \frac{2}{n} + \frac{T(n-2)}{(n-1)} \right\} \text{ - v}$$

Put (n-2) for n in iv

$$\left\{ \frac{T(n-2)}{(n-1)} = \frac{2}{(n-1)} + \frac{T(n-3)}{(n-2)} \right\} \text{ - vi}$$

put (n-3) for n in iv

$$\left\{ \frac{T(n-3)}{(n-2)} = \frac{2}{(n-2)} + \frac{T(n-4)}{(n-3)} \right\} \text{ - vii}$$

Further - see in pdf

* Multiplication of an bit numbers :-

Ex:- Multiply => X = 101 => odd
Y = 100
10100

make even using divide & Conquer

A , B
X = 0 1 1 0 1
0 1 , 0 0
C , D

A = $2^1 \times 0 + 1 \times 2^0$ = $2^{n/2}$ / $2^{n/2}$
B = $2^1 \times 0 + 1 \times 2^0$ =

$X = A * 2^{n/2} + B$
$Y = C * 2^{n/2} + D$

Multiply = $X * Y$

$\Rightarrow (A * 2^{n/2} + B) * (C * 2^{n/2} + D)$

$\Rightarrow AC * 2^n + AD * 2^{n/2} + BC * 2^{n/2} + BD$

$\Rightarrow [AC * 2^n + 2^{n/2}(AD + BC) + BD]$

$T(n) = \underline{4T(n/2)} + \underline{O(n)} \Rightarrow O(n^2)$
multiply Sum

To Reduce multiplication 4 time
We Can do So → Let V = AC , W = BD
Let $(A+B) * (C+D) = U$
$AC + AD + BC + BD = U$
$(AD + BC) + \underline{AC} + \underline{BD} = U$
V W

$\{AD + BC = U - V - W\}$

$\Rightarrow [V * 2^n + 2^{n/2}(U - V - W) + W]$

Thus - $[T(n) = 3T(n/2) + O(n), \quad n \geq 2]$

multiply 3 time

$$T(n) = \begin{cases} 3T(n/2) + an, & n \geq 2 \\ b, & n = 1 \end{cases}$$

$\Rightarrow T(n) = 3T(n/2) + an$

Put $n = n/2$

$T(n/2) = 3T(n/4) + a * n/2$

$T(n) = 3\left[3T(n/4) + a * \frac{n}{2}\right] + an$

$= 3^2 T(n/2^2) + \frac{3}{2} * an + an$

$= 3^2\left[3T(n/8) + a * \frac{n}{4}\right] + \frac{3}{2} * an + an$

$\Rightarrow 3^3 T(n/2^3) + \left(\frac{3}{2}\right)^2 * an + \left(\frac{3}{2}\right)^1 an + \left(\frac{3}{2}\right)^0 an$

upto n times

$\Rightarrow 3^p T(n/2^p) + \left(\frac{3}{2}\right)^{p-1} an + \left(\frac{3}{2}\right)^{p-2} an + \left(\frac{3}{2}\right)^{p-3} an + \cdots$

$\left(\frac{3}{2}\right)^1 an + \left(\frac{3}{2}\right)^0 an$

$\Rightarrow 3^p T(n/2^p) + an\left[\left(\frac{3}{2}\right)^{p-1} + \left(\frac{3}{2}\right)^{p-2} + \cdots \left(\frac{3}{2}\right)^1 + \left(\frac{3}{2}\right)^0\right]$

use G.P. Sum.

$$\Rightarrow 3^p T\left(\frac{n}{2^p}\right) + an * \left[\frac{1\left(\left(\frac{3}{2}\right)^p - 1\right)}{3/2 - 1}\right]$$

$$\Rightarrow 3^p T\left(n/2^p\right) + 2an\left[\left(\frac{3}{2}\right)^p - 1\right]$$

$n = 2^p$, $p = \log_2 n$

$$\Rightarrow 3^{\log_2 n} T(1) + 2an\left[\frac{3^{\log_2 n}}{2^{\log_2 n}} - 1\right]$$

$$\Rightarrow n^{\log_2 3} * b + 2an\left[\frac{n^{\log_2 3}}{n} - 1\right]$$

$$\Rightarrow n^{\log_2 3} * b + 2a * n^{\log_2 3} - 2an$$

$$\Rightarrow n^{\log_2 3}(2a+b) - 2an$$

$$\boxed{T(n) \leq C * n^{\log_2 3}}$$

$$\{ T(n) \in O(n^{\log_2 3}) \ / \ T(n) \in O(n^{1.59}) \}$$

Ex:- $X = 101$ odd → make even
$Y = 100$

$$X = \overset{A}{0}\,1\,\overset{B}{|}\,0\,1 \qquad \Rightarrow X * Y = 10100$$
$$Y = \overset{C}{0}\,1\,\overset{D}{|}\,0\,0$$

Use formula $\Rightarrow X * Y = \underset{V}{AC} * 2^n + 2^{n/2}\underset{U-V-W}{(AD+BC)} + \underset{W}{BD}$

$$\Rightarrow V * 2^n + 2^{n/2}(U-V-W) + W$$

Page No.
Date :

* => $V = A * C = 01 * 01 = 001$

$W = B * D = 01 * 00 = 000$

$U = (A+B) * (C+D) = (01+01) * (01+00)$

$= (010) * (001)$

$\boxed{U = 0010}$

=> $U - V - W = (0010 - 001 - 000)$

$\boxed{U - V - W = 0001}$

=> $X * Y = 001 * 2^4 + 2^2 (0001) + 000$

$= 001 * (10000) + (100) * (0001) + 000$

$= 0010000 + 000100 + 000$

{ $X * Y = 0010100 / 10100$ } Ans

* Multiplication of two Matrices:-

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$A \times B = C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} \\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22} \end{bmatrix}$$

$$\Rightarrow T(n) = \underbrace{8T(n)}_{\text{(8 time multiplication)}} + \underbrace{O(n^2)}_{\text{(Sum)}} \quad \Rightarrow \text{time comp. } O(n^3)$$

(i x j) (i, j, k)

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

($n/2 \times n/2$ blocks)

Now By using Strassses' Concept:-

Recurrance by Reducing Multiplication by 1.

$$T(n) = \begin{cases} 7T(n/2) + an^2, & n \geq 2 \\ b, & n = 1 \end{cases}$$

$$T(n) = 7\left[7T(n/4) + a\times(\tfrac{n}{2})^2\right] + an^2$$

$$= 7^2 T(n/4) + \tfrac{7}{4}an^2 + an^2$$

$$\Rightarrow 7^2\left[7T(n/8) + a\times(\tfrac{n}{4})^2\right] + \tfrac{7}{4}an^2 + an^2$$

$$= 7^3 T\left(\frac{n}{2^3}\right) + \left(\tfrac{7}{4}\right)^2 * an^2 + \left(\tfrac{7}{4}\right)an^2 + \left(\tfrac{7}{4}\right)^0 an^2$$

$$\Rightarrow 7^3 T\left(\frac{n}{2^3}\right) + an^2\left[\left(\frac{7}{4}\right)^2 + \left(\frac{7}{4}\right)^1 + \left(\frac{7}{4}\right)^0\right]$$

$$\vdots$$

upto p terms

$$\Rightarrow 7^p T\left(\frac{n}{2^p}\right) + an^2\left[\underbrace{\left(\frac{7}{4}\right)^{p-1} + \left(\frac{7}{4}\right)^{p-2} + \cdots + \left(\frac{7}{4}\right)^1 + \left(\frac{7}{4}\right)^0}_{\text{G.P.}}\right]$$

$$\Rightarrow 7^p T\left(\frac{n}{2^p}\right) + an^2\left[\frac{\left(\frac{7}{4}\right)^p - 1}{\frac{7}{4} - 1}\right]$$

$$\Rightarrow 7^p T\left(\frac{n}{2^p}\right) + \frac{4}{3}an^2\left[\frac{7^p}{4^p} - 1\right]$$

$\{ p = \log_2 n \ , \ n = 2^p \}$

$$\Rightarrow 7^p T\left(\frac{2^p}{2^p}\right) + \frac{4}{3}an^2\left[\frac{7^{\log_2 n}}{4^{\log_2 n}} - 1\right]$$

$$\Rightarrow 7^p T(1) + \frac{4}{3}an^2\left[\frac{n^{\log_2 7}}{n^2} - 1\right]$$

$$\Rightarrow 7^p \times b + \frac{4}{3}a * n^{\log_2 7} - \frac{4}{3}an^2$$

$$\Rightarrow 7^{\log_2 n} * b + \frac{4}{3}a * n^{\log_2 7} - \frac{4}{3}an^2$$

$$\Rightarrow n^{\log_2 7} * b + \frac{4}{3}a * n^{\log_2 7} - \frac{4}{3}an^2$$

$$\Rightarrow n^{\log_2 7}\left(b + \frac{4}{3}a\right) - \frac{4}{3}an^2$$

$n^{\log_2 7} >>> n^2$, $n^2 \simeq 0$

$$\left\{\begin{array}{l} T(n) \leq c * n^{\log_2 7} \\ T(n) \in O(n^{\log_2 7}) \\ T(n) \in O(n^{2.81}) \end{array}\right\}$$

* Stransses' Matrix Algorithm :-

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

1) if $n = 1$ then
2) $C = A \cdot B$
3) else
4) $M_1 = (a_{11} + a_{22}) \cdot (b_{11} + b_{22})$
5) $M_2 = (a_{21} + a_{22}) \cdot b_{11}$
6) $M_3 = a_{11} \cdot (b_{12} - b_{22})$
7) $M_4 = a_{22} \cdot (b_{21} - b_{11})$
8) $M_5 = (a_{11} + a_{12}) \cdot b_{22}$
9) $M_6 = (a_{21} - a_{11}) \cdot (b_{11} + b_{12})$
10) $M_7 = (a_{12} - a_{22}) \cdot (b_{21} + b_{22})$
11) $C_{11} = M_1 + M_4 - M_5 + M_7$
12) $C_{12} = M_3 + M_5$
13) $C_{21} = M_2 + M_4$
14) $C_{22} = M_1 - M_2 + M_3 + M_6$

Output $A \cdot B = C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

Page No.

Ex:

$$A = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ \hline 1 & 1 & 1 & 1 \\ 0 & 2 & 1 & 2 \end{array}\right]_{4\times4} \qquad B = \left[\begin{array}{cc|cc} 0 & 0 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ \hline 2 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{array}\right]_{4\times4}$$

A_{11} A_{12} A_{21} A_{22} — B_{11} B_{12} B_{21} B_{22}

$$\Rightarrow M_1 = (a_{11} + a_{22}) \cdot (b_{11} + b_{22})$$

$$\Rightarrow \left\{\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}\right\} * \left\{\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\right\}$$

$$\Rightarrow \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 0 & 0 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix}$$

$$\left\{ M_1 = \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} \right\}$$

$$\Rightarrow M_2 = (a_{21} + a_{22}) \cdot b_{11}$$

$$\Rightarrow \left\{\begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}\right\} * \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 2 & 2 \\ 1 & 4 \end{bmatrix} * \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 4 \end{bmatrix}$$

$$\left\{ M_2 = \begin{bmatrix} 2 & 2 \\ 4 & 4 \end{bmatrix} \right\}$$

$$\Rightarrow M_3 = a_{11} \cdot (b_{12} - b_{22})$$

$$\Rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} * \left\{ \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \right\}$$

$$\Rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$$

$$\left\{ M_3 = \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} \right\}$$

$$\Rightarrow M_4 = a_{22} \cdot (b_{21} - b_{11})$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} * \left\{ \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \right\}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 2 & 0 \end{bmatrix}$$

$$\left\{ M_4 = \begin{bmatrix} 2 & 0 \\ 2 & 0 \end{bmatrix} \right\}$$

$$\Rightarrow M_5 = (a_{11} + a_{12}) \cdot b_{22}$$

$$\left\{ \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right\} * \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} = \left\{ \begin{bmatrix} 1 \cdot 0 \\ 0 & 0 \end{bmatrix} = M_5 \right\}$$

=) $C_{21} = M_2 + M_4$

$$\begin{bmatrix} 2 & 3 \\ 4 & 4 \end{bmatrix} + \begin{bmatrix} 2 & 0 \\ 2 & 0 \end{bmatrix} = \left\{ \begin{bmatrix} 4 & 2 \\ 6 & 4 \end{bmatrix} = C_{21} \right\}$$

=) $C_{22} = M_1 - M_2 + M_3 + M_6$

$$\begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} - \begin{bmatrix} 2 & 3 \\ 4 & 4 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 4 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$$

$$\left\{ C_{22} = \begin{bmatrix} 3 & 3 \\ 4 & 2 \end{bmatrix} \right\}$$

$$\left\{ C = \begin{bmatrix} 3 & 1 & 2 & 2 \\ 2 & 0 & 0 & 0 \\ 4 & 2 & 3 & 3 \\ 6 & 4 & 4 & 2 \end{bmatrix} \right\}$$

=) $C_{21} = M_2 + M_4$

$$\begin{bmatrix} 2 & 3 \\ 4 & 4 \end{bmatrix} + \begin{bmatrix} 2 & 0 \\ 2 & 0 \end{bmatrix} = \left\{ \begin{bmatrix} 4 & 2 \\ 6 & 4 \end{bmatrix} = C_{21} \right\}$$

=) $C_{22} = M_1 - M_2 + M_3 + M_6$

$$\begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} - \begin{bmatrix} 2 & 3 \\ 4 & 4 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 4 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$$

$$\left\{ C_{22} = \begin{bmatrix} 3 & 3 \\ 4 & 2 \end{bmatrix} \right\}$$

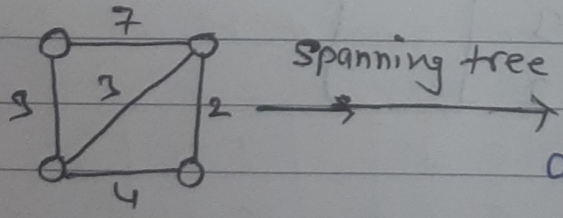
$$\left\{ C = \begin{bmatrix} 3 & 1 & 2 & 2 \\ 2 & 0 & 0 & 0 \\ 4 & 2 & 3 & 3 \\ 6 & 4 & 4 & 2 \end{bmatrix} \right\}$$

Greedy Method :-

It is one of the Strategies like Divide & Conquer used to ~~Store~~ Solve the problems. This method is used for Solving optimization problems. An optimization problem is a problem that demands either Max^m or min^m result. It is one of the simple & straight forward approach. The main func^c of this approach is taken on the basis of currently available information. Whatever the current information is available, the decision is made without worrying about the effect the current decision in future.

This technic is basically used to determine fisible Sol^n that may or may not be optimal.

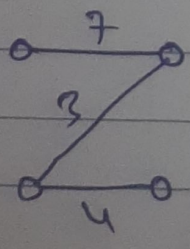
Ex^l :-



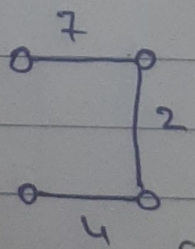
Spanning tree →

Vertices = 4 = n

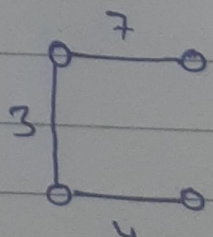
atmost edges = (n-1) = 4-1 = 3

① 

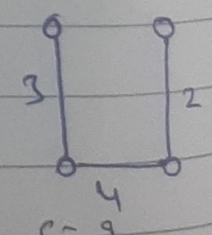
Cost = 14

② 

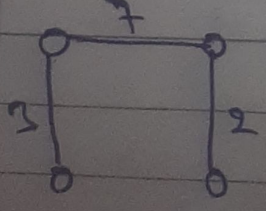
C = 13

③ 

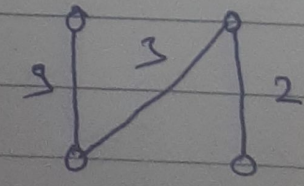
C = 14

④ 

C = 9

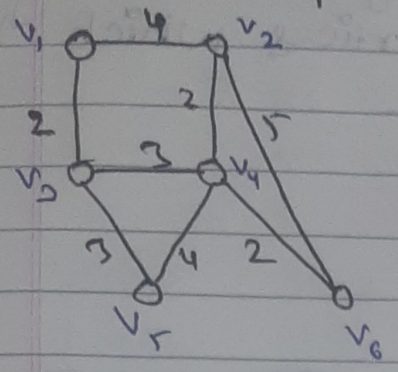
⑤ 

C = 12

⑥ 

C = 8 ⇒ min^m Cost ⇒ optimal.

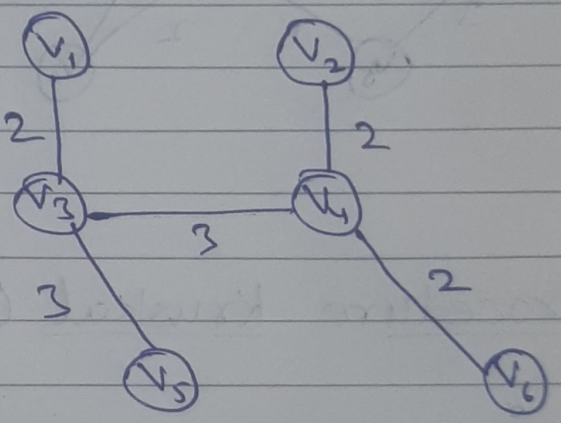
Ex:- Create Spanning tree using kruskal algorithm.



Vertices = 6
edges = 6-1 = 5 atmost

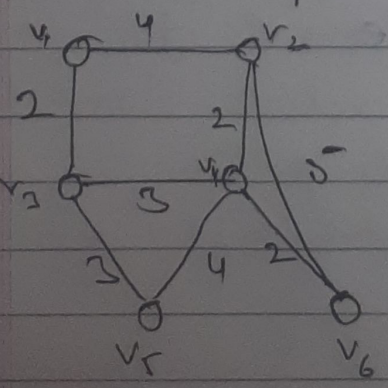
=> Arrange Cost in ASC order and make Spanning tree skiping circle.

$(v_1, v_3) = 2$
$(v_2, v_4) = 2$
$(v_4, v_6) = 2$
$(v_3, v_4) = 3$
$(v_3, v_5) = 3$
$(v_1, v_2) = 4$
$(v_4, v_5) = 4$
$(v_2, v_6) = 5$



Cost = 12 => optimal.

Ex:- Create Spanning tree using Prim's Algorithm.



In this algorithm make a set of vertices

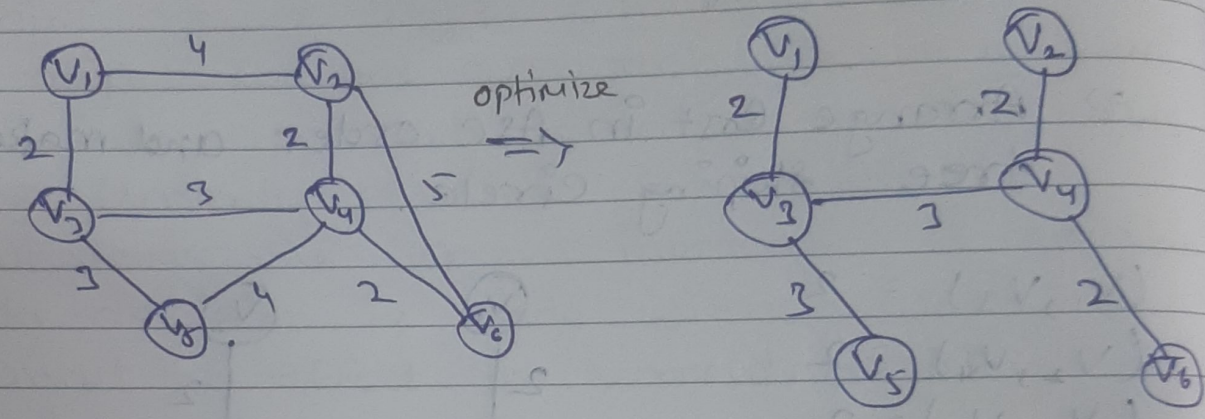
$A = \{ v_1, v_2, v_3, v_4, v_5, v_6 \}$

(arbitrary)

first Vertices will be assigned randomly then assigned its connected vertices which have lower cost and so on...

$A = \{ V_3, V_1, V_4, V_2, V_6, V_5 \}$

arbitrary



#* Procedure Kruskal (V, E) :-

Begin

- $T \leftarrow \phi$
- $n \leftarrow |V|$ => mode represent total No.
- Repeat
- Select $e \in E$ of the lower cost
- If (TUe) makes a cycle
  Reject e
- $E \leftarrow E - e$
- else
  $T \leftarrow TUe$
- end if
- Until $|T| \neq (n-1)$
  Return (T)
- End

Procedure Prim's (V, E)

- Begin
- $T \leftarrow v \in V$ (arbitrary vertices)
- $V \leftarrow V - v$
- $B = B \cup V$
- While $V \neq$ empty do
- begin
- Select edge (u, v) s.t. $u \in B$ & $v \in V$ of lower cost.
- If $T \cup (u, v)$ does not makes cycle.
- $T \leftarrow T \cup (u, v)$
- $B \leftarrow B \cup v$

end if

$V \leftarrow V - v$

end

return (T)

End

Knapsack Problem: (GG site)

We are given n no. of item with their associate weight vector $W = (w_1, w_2, w_3, \ldots, w_n)$ and Profit vector $P = (p_1, p_2, p_3, \ldots, p_n)$

The capacity of the bag is C.

our objective is to maximize profit.

$$\sum_{i=1}^{n} p_i x_i$$

Subject to

$$\sum_{i=1}^{n} w_i x_i \leq C$$

Where $x_i \in \{0, 1\}$ or $0 \leq x_i \leq 1$
in case of fractional problems.

Ex:-

Input $N = 3$, $W = 4$ Profit[] = {1, 2, 3}
Weight[] = {4, 5, 1}

=> Soln 1 :-
Only 1st item is consider [4]
{profit = 1}

Soln 2: Only 3rd item is consider [1]
{profit = 3} => optimal Ans

Ex:- $N = 3$ $(w_1, w_2, w_3) = (10, 7, 5)$
$(P_1, P_2, P_3) = (13, 8, 4)$
$C = 10$

=> Soln 1: Increasing order of weight
$(w_3, w_2, w_1) \rightarrow (5, 7, 10)$
item 1 → 2 → 3

Soln 2: Decreasing order of Profit
$(w_1, w_2, w_3) \rightarrow (13, 8, 4)$
(P_1, P_2, P_3) item 1 → 2 → 3

(P/w)

Soln 3: Arrange by (Profit per unit Weight) wise

$P_1/W_1 = 1.3$

$P_2/W_2 = 1.14$

$P_3/W_3 = 0.8$

$W_1 \to W_2 \to W_3 \Rightarrow 1 \to 2 \to 3$

* By using fractional [Soln 1]:-

item No.	W_i	P_i	x_i	$W_i x_i \leq C$	$P_i x_i$
3	5	4	1	$5 \leq 10$	4
2	7	8	$5/7$	$5 + \frac{5}{7} \times 7 = 10$	$4 + 8 \times \frac{5}{7} = 4 + 5.71 = 9.71$
1	10	13	0	10	9.71

Soln vector $\Rightarrow (x_1, x_2, x_3) \to (0, 5/7, 1)$

profit = 9.71

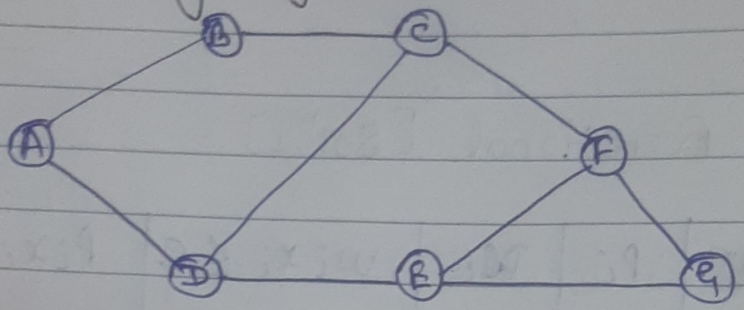
Soln 2:-

item No.	W_i	P_i	x_i	$W_i x_i \leq C$	$P_i x_i$
1.	10	13	1	$10 \leq 10$	13
2	7	8	0	10	13
3	5	4	0	10	13

Page No.
Date :

Soln Vector $(x_1, x_2, x_3) \Rightarrow (1, 0, 0)$ with Profit = 13

Searching Algorithms [qq]



⇒ DFS (use Stack) , BFS (use Queue)

Firstly all unvisited

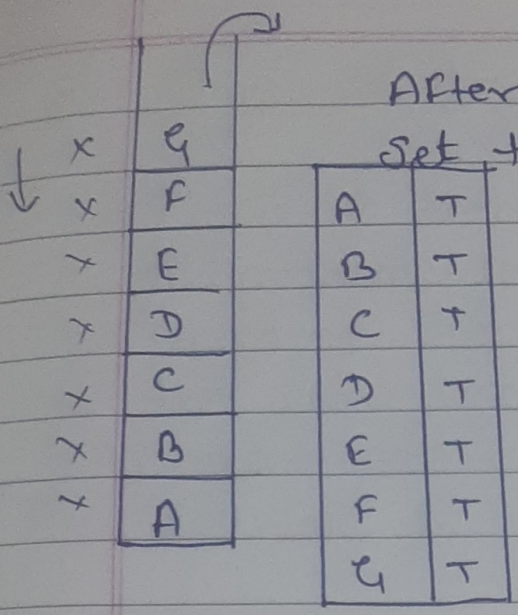
A	f
B	f
C	f
D	f
E	f
F	f
G	f

Ado Adjacent Nodes of elements

A	B → D
B	C → A
C	B → D → F
D	A → C → E
E	D → F → G
F	C → E → G
G	E → F

Now by Calling DFS

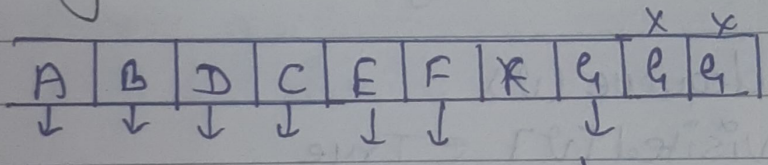
Start from A



After exit of all adjacent node then we set the main node 'TRUE'.

GFECDAB

By Calling BFS



T = A B D C E F G

} false can be repeated in Queue and true will not be entered again.

* Algorithm of 'DFS' :-

- DFS (v)
  Begin
- push (v)
- For all adjacent nodes u of v
- If ! (visited[u])
  push (u)
  DFS (u)
- end for

- print (v).
- visited [v] ← True
  end.

* Algorithm of BFS

- BFS (v)
  Begin
- enque (v)
- While Queue ≠ empty
- begin
  dequeue (v)
  visited [v] ← True
  For all adjacent vertices u of v
  enqueue (u)
  end for
- end while

(Unit - 2 Complete)

Page No.
Date :

Dynamic Programming :-

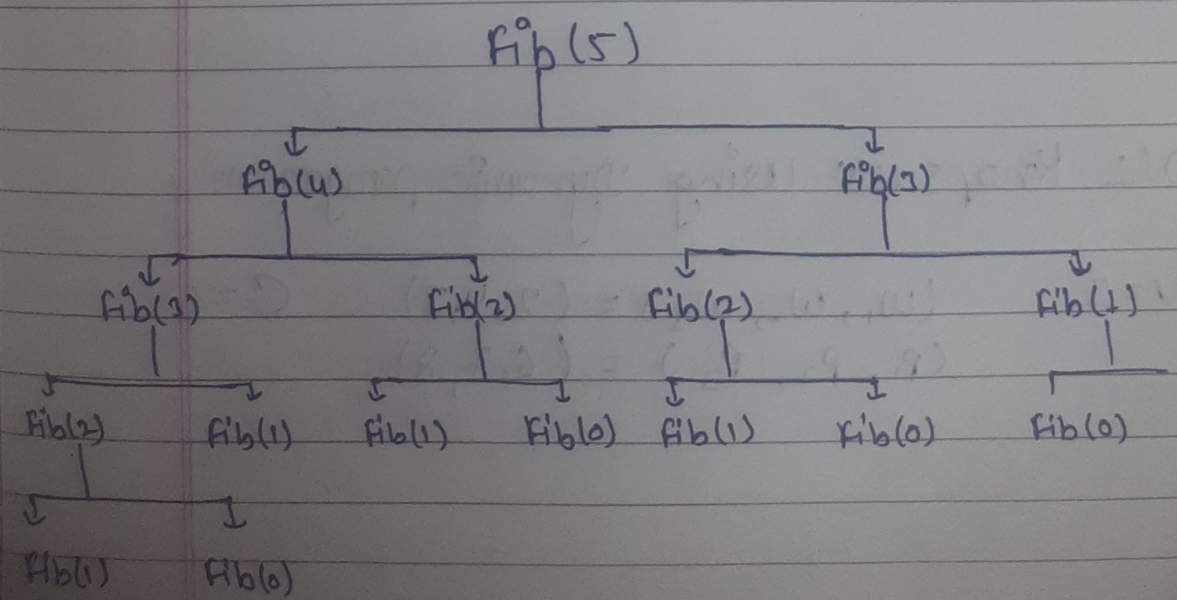
It is a technic that breaks the problem into subproblems and saves the result for future purposes, so that we do not need to compute the same result again & again. The sub-problem are optimized to optimize the overall solution, is known as principle of optimility.

The main use of dynamic programming is to solve optimization problem in which we use either maximization or minimization as one of the criteria.

The dynamic programming guaranties to find optimal solution of a problem if it exists.

ex:- The Fibannaci series

$$F_n = F_{n-1} + F_{n-2}$$



As we can see in the above figure, we store the results of subproblem [memorization] so that the same subproblem is not required to be calculated more than once, because it reuse the results from the table. Hence we can say that the complex problem can be solved in a faster way by avoiding re-calculation of overlap sub problems.

→ In this case the space complexity would be increase because we are storing the intemediate results but the time complexity would be decreased.

* We can solve these two problems by using Dynamic programming :-

i) 0/1 knapsack

ii) Travelling ~~knapsack~~ Salesman problem Backtracking.

* 0/1 knapsack using Dynamic programming :-

Ex:- $N = 3$ $(w_1, w_2, w_3) = (2, 4, 3)$ $C = 6$

$(P_1, P_2, P_3) = (6, 7, 8)$

Page No.
Date :

* Recursive formula :-

$$f_i(y) = \max\{ f_{i-1}(y),\ f_{i-1}(y-w_i) + p_i \}$$

$\because y = C$

Assumption :-

- $f_0(y) = f_1(y) = f_2(y) = f_3(y) = -\infty \quad , \forall\ y < 0$
- $F_1(y) = F_2(0) = F_3(0) = 0$
- $F_0(y) = 0 \quad \forall\ y \geq 0$

Q. $W = (2, 4, 8)$
$P = (6, 7, 8)$
$C = 6$

=> For $i = 1$

$$F_1(1) = \max\{ F_0(1),\ F_0(1-2) + 6 \}$$
$$= \max\{ F_0(1),\ F_0(-1) + 6 \}$$
$$= \max\{ 0, -\infty + 6 \}$$
$$= \max\{ 0, -\infty \}$$
$$= 0$$

$\{ F_1(1) = 0 \}$

→ $F_1(2) = \max\{F_0(2); F_0(2-2)+6\}$

$= \max\{F_0(2); F_0(0)+6\}$

$= \max\{0, 0+6\}$

$= \max\{0, 6\}$

$\boxed{F_1(2) = 6}$

→ $F_1(3) = \max\{F_0(3), F_0(3-2)+6\}$

$= \max\{F_0(3), F_0(1)+6\}$

$= \max\{0, 0+6\}$

$= \max\{0, 6\}$

$\boxed{F_1(3) = 6}$

Same with $F_1(4)$, $F_1(5)$ & $F_1(6) = 6$.

⇒ for $i = 2$

→ $F_2(1) = \max\{F_1(1), F_1(1-4)+7\}$

$= \max\{F_1(1), F_1(-3)+7\}$

$= \max\{0, -\infty+7\}$

$= \max\{0, -\infty\}$

$\boxed{F_2(1) = 0}$

→ $F_2(2) = \max\{F_1(2), F_1(2-4)+7\}$

$= \max\{F_1(2), F_1(-2)+7\}$

$= \max\{F_1(2), -\infty+7\}$

$= \max\{F_1(2), -\infty\}$

$= \max\{0, -\infty\}$

$\boxed{F_2(2) = 0}$

→ $F_2(3) = \max\{F_1(3), F_1(3-4)+7\}$
$= \max\{F_1(3), F_1(-1)+7\}$
$= \max\{F_1(3), -\infty+7\}$
$= \max\{0, -\infty\}$
$\boxed{F_2(3) = 0}$

→ $F_2(4) = \max\{F_1(4), F_1(4-4)+7\}$
$= \max\{F_1(4), F_1(0)+7\}$
$= \max\{0, 0+7\}$
$= \max\{0, 7\}$
$\boxed{F_2(4) = 7}$

→ $F_2(5) = \max\{F_1(5), F_1(5-4)+7\}$
$= \max\{6, F_1(1)+7\}$
$= \max\{6, 0+7\}$
$= \max\{6, 7\}$
$\boxed{F_2(5) = 7}$

→ $F_2(6) = \max\{F_1(6), F_1(6-4)+7\}$
$= \max\{6, F_1(2)+7\}$
$= \max\{6, 6+7\}$
$= \max\{6, 13\}$
$\boxed{F_2(6) = 13}$

⇒ For $i = 3$

$F_3(1) = \max\{F_2(1), F_2(1-8)+8\}$
$= \max\{F_2(1), F_2(-7)+8\}$
$= \max\{0, -\infty+8\}$

$= \max\{0, -\infty\}$

$\boxed{F_3(1) = 0}$

→ $F_3(2) = \max\{F_2(2), F_2(2-8)+8\}$

$= \max\{F_2(2), F_2(-6)+8\}$

$= \max\{6, -\infty+8\}$

$= \max\{6, -\infty\}$

$\boxed{F_3(2) = 6}$

→ $F_3(3) = \max\{F_2(3), F_2(3-8)+8\}$

$= \max\{F_2(3), F_2(-5)+8\}$

$= \max\{6, -\infty+8\}$

$= \max\{6, -\infty\}$

$\boxed{F_3(3) = 6}$

→ $F_3(4) = \max\{F_2(4), F_2(4-8)+8\}$

$= \max\{F_2(4), F_2(-4)+8\}$

$= \max\{7, -\infty+8\}$

$= \max\{7, -\infty\}$

$\boxed{F_3(4) = 7}$

→ $F_3(5) = \max\{F_2(5), F_2(5-8)+8\}$

$= \max\{F_2(5), F_2(-3)+8\}$

$= \max\{7, -\infty+8\}$

$= \max\{7, -\infty\}$

$\boxed{F_3(5) = 7}$

→ $F_3(6) = \max \{ F_2(6), F_2(6-8)+8 \}$

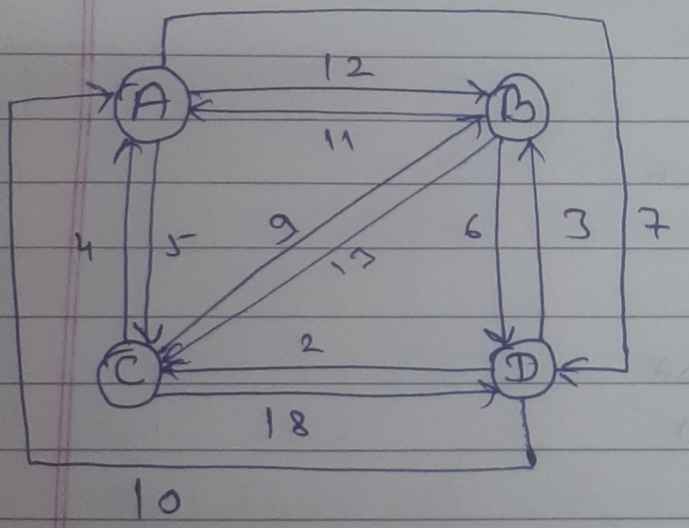
$= \max \{ F_2(6), F_2(-2)+8 \}$

$= \max \{ 13, -\infty \}$

$\boxed{F_3(6) = 13}$

$\{ (x_1, x_2, x_3) = (1, 1, 0) \}$

Travelling Salesman Problem:-



	A	B	C	D
A	∞	12	5	7
B	11	∞	13	6
C	4	9	∞	18
D	10	3	2	∞

Let the vertices of the graph be numbered 1 through n. W/o loses of generality, we start a tour from the vertex 1 & also ends at vertex 1.

If the tour is optimal, each vertex from the vertex list should be visited exactly once. Hence, the principle of optimality holds here.

Let $S_p(i, s)$ be the length of the

Shortest path starting at vertex i, going through all the vertices in S and terminating at the vertex 1, then it can be represented as follows :

$$Sp(i,S) = \min_{j \in S} \{ W_{ij} + Sp(j, S-\{j\}) \}$$

Now,

① $S = \phi$

$$\therefore Sp(B,\phi) = W_{BA}$$
$$= 11$$

$$Sp(C,\phi) = W_{CA}$$
$$= 4$$

$$Sp(D,\phi) = W_{DA}$$
$$= 10$$

Now for $S = \{B\}$

$$Sp : (C, \{B\}), Sp(D, \{B\})$$

$$\Rightarrow Sp(C, \{B\}) = \min \{ W_{CB} + Sp(B,\phi) \}$$
$$= \min \{ 9 + 11 \}$$
$$= \min \{ 20 \}$$

$$\Rightarrow Sp(D, \{B\}) = \min \{ W_{DB} + Sp(B,\phi) \}$$
$$= \min \{ 3 + 11 \}$$
$$= \min \{ 14 \}$$

Page No.
Date :

→ For $S = \{C\}$

$$Sp(B, \{C\}) = \min\{W_{BC} + Sp(C, \phi)\}$$
$$= \min\{13 + 4\}$$
$$= 17$$

$$Sp(D, \{D\}) = \min\{W_{DC} + Sp(C, \phi)\}$$
$$= \min\{2 + 4\}$$
$$= 6$$

⇒ For $S = \{D\}$

$$Sp(B, \{D\}) = \min\{W_{BD} + Sp(D, \phi)\}$$
$$= \min\{6 + 10\}$$
$$= 16$$

$$Sp(C, \{D\}) = \min\{W_{CD} + Sp(D, \phi)\}$$
$$= \min\{18 + 10\}$$
$$= 28$$

⇒ Now for $S = \{B, D\}$

$$Sp(C, \{B, D\}) = \min\{W_{CB} + Sp(B, D), W_{CD} + Sp(D, B)\}$$
$$= \min\{9 + 16, 18 + 14\}$$
$$= \min\{25, 32\}$$
$$= 25$$

⇒ For $S = \{B, C\}$

$$Sp(D, \{B, C\}) = \min\{W_{DB} + Sp(B, C), W_{DC} + Sp(C, B)\}$$
$$= \min\{3 + 17, 2 + 20\}$$
$$= \min\{20, 22\}$$
$$= 20$$

→ For $S = \{C, D\}$

$Sp(B, \{C, D\}) = \min\{W_{BC} + Sp(C, D), W_{BD} + Sp(D, C)\}$

$= \min\{13+16, 6+17\}$

$= \min\{29, 23\}$

$= 23$

⇒ Now for $S = \{B, C, D\}$

$Sp(A, \{B, C, D\}) = \min\{W_{AB} + Sp(B, \{C, D\}), W_{AC} + Sp(C, \{B, D\}), W_{AD} + Sp(D, \{B, C\})\}$

$= \min\{12+12, 5+25, 7+20\}$

$= \min\{24, 30, 27\}$

$= 24$

$A \xrightarrow[12]{} B \xrightarrow[6]{} D \xrightarrow[2]{} C \xrightarrow[4]{} A = 24.$

Page No.
Date: 22 | 7 | 24

Backtracking [Read from Google]

* Problems of Backtracking:

1. N-Queens' Problem: [Read from Google]

N-Queens' should be placed on NXN Board S.t. no 2 queens can be on the same diagonal or same col^{m} or same row.

Ex:- N=4 , (Q_1, Q_2, Q_3, Q_4)

Q_1			
x	x	Q_2	
x	x	x	x

↑ Backtrack as Q_3 has no place

→

Q_1			
x	x	x	Q_2
x	Q_3		
x	x	x	x

↑ Backtrack change Q_1's position.

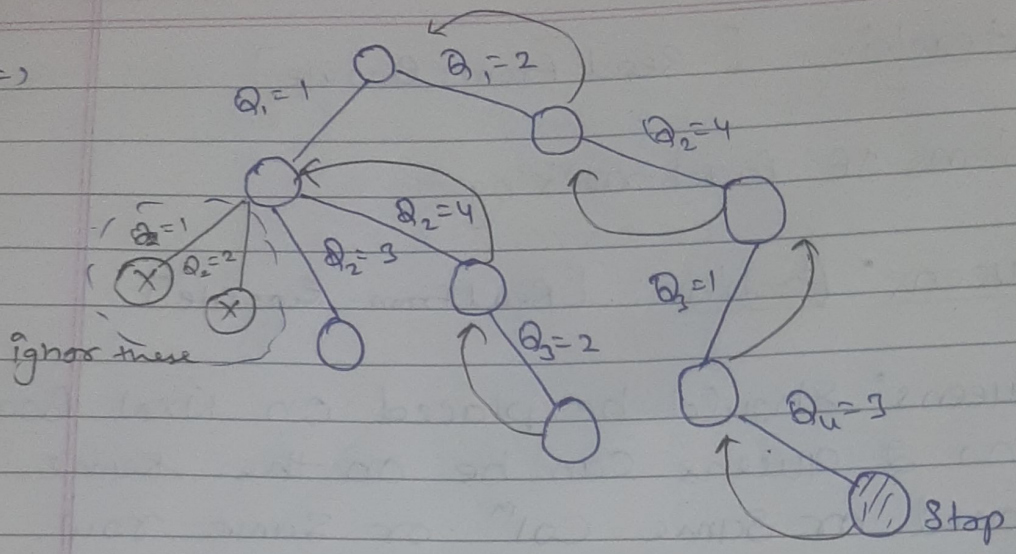
→

x	Q_1		
x	x	x	Q_2
Q_3			
x	x	Q_4	

=) $(Q_1, Q_2, Q_3, Q_4) = (2, 4, 1, 3)$ ⤵ mirror image will be another

$(Q_1, Q_2, Q_3, Q_4) = (3, 1, 4, 2)$ soln.

=>



* Backtracking :-

The B.T. (Back Tracking) Algo. is used to solve n mean problems in which the board is recurrsively filled with Queens, one at a time and backtracked, is no valid Soln is found.

At each step the algo. checks to see if the newly placed Queen conflicts with any previously placed Queen and so, then it back tracked. The proccess is repeated untill all N Queen are placed on the board without conflict or untill all possible Soln are exchausted.

Backtracking is a problem Solving techanic that involves recursively trying out different Solutions and B.T. or undoing previous choices when they don't leave to a valid Soln.

It is an effective technic as it allows us to incrementally build a Sol^n and avoids the need to consider all possible Sol^n From Scratch.

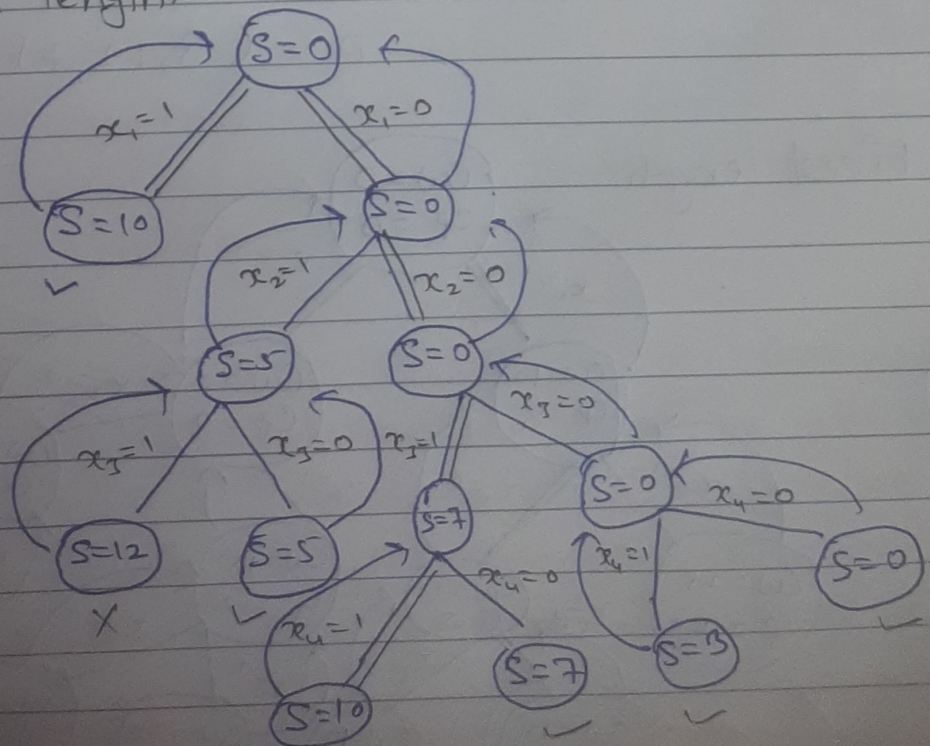
Sum of Subsets problem :-

Ex:- $S = (10, 5, 7, 3)$
$c = 10 \Rightarrow S_1 = (10)$
$S_2 = (7, 3)$

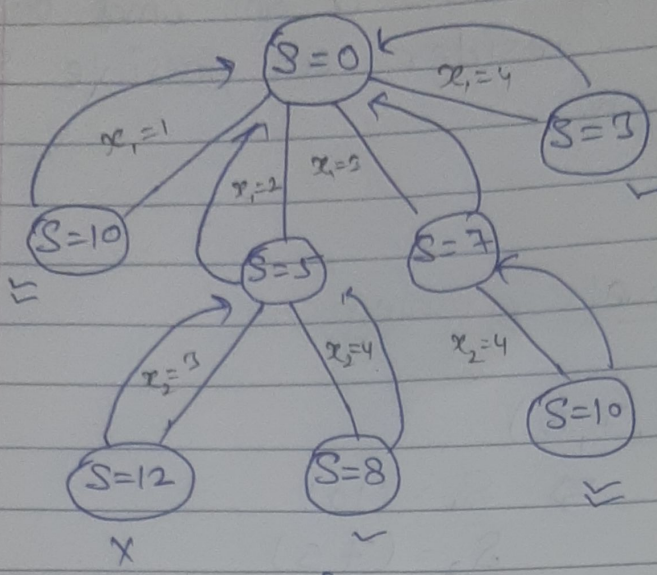
i) $(x_1, x_2, x_3, x_4) = (1, 0, 0, 0)$ => fixed length
$= (0, 0, 1, 1)$

ii) (1)] variable length, tuple representation
(3,4)

=> fixed length.



=> Variable length:



Solution=> $x_1 = (1)$

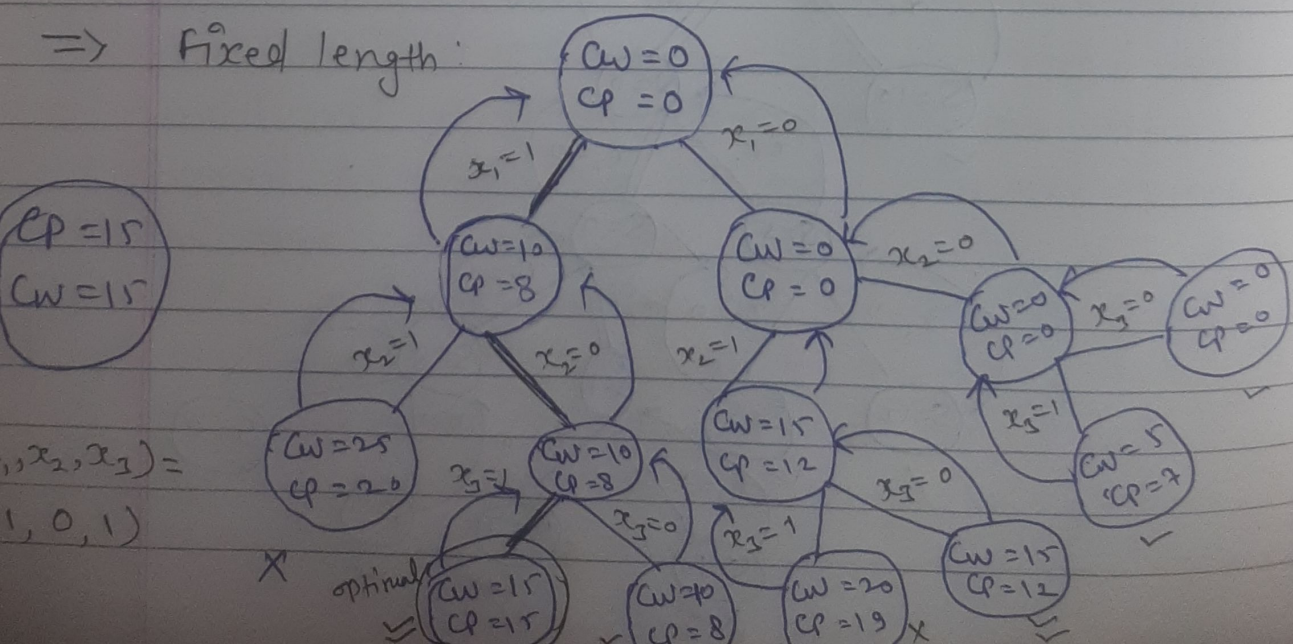
$(x_1, x_2) = (3, 4)$

0/1 knapsack problem using backtracking:-

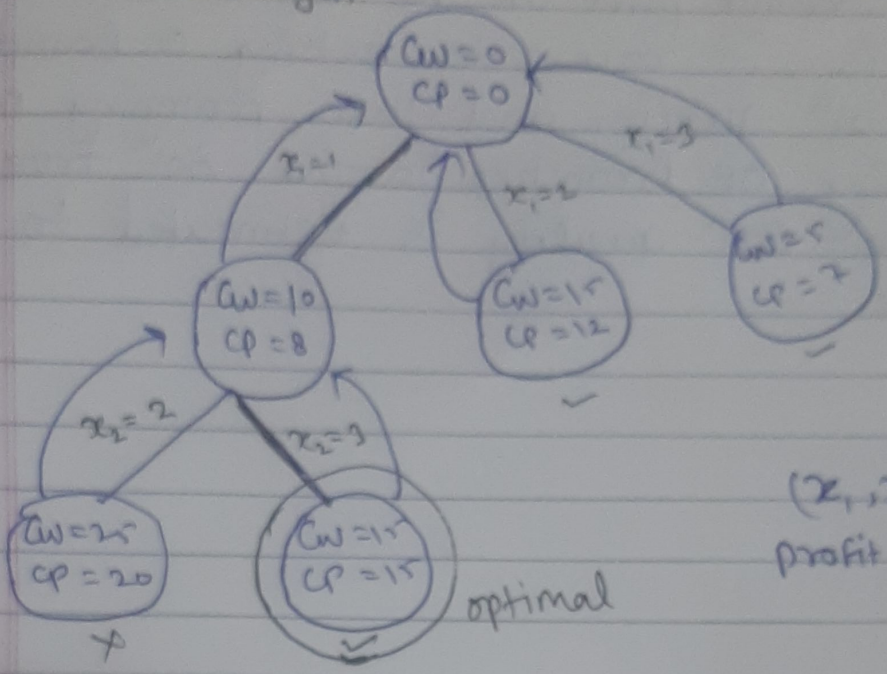
Ex:- $N=3$, $C=15$, $w = (10, 15, 5)$

$P = (8, 12, 7)$

=> Fixed length:



=) Variable length:



$(x_1, x_2) = (1, 3)$
profit value = 15

Branch & Bound :-

It is a General technic for improving the searching process by systematically enumerating all the candidate soln and discarding of imposible soln. This type of technic usually applies to those problems that ~~are~~ have finite solutions in which the soln can be representive as a sequence of option.

→ The first part of B&B is "Branching" that requires several choices to be mee so that the choices branch-out into the soln space. In this method the soln space is organised as a tree like structure.

→ The Second part OF the B & B is "Bounding" that is, it refers to setting a Bound on the Soln quality and then pruning means trimming OF Branches in the Soln tree, whose Soln quality is estimated to be poor.