UNIT II

* Decrease and Conquer:-

step 1:- Reduce the problem instance is reduced to smaller instances of the same problem

Approach 1: Decrease by constant (usually by 1)
- eg. Insertion sort
- Topological sort
- Generating permutation

Approach 2: Decrease by a constant factor
- $n, n/2, n/4, n/8, \ldots$
- eg. Binary Search
- Bisection Method

Approach 3: Decrease by a variable size
- eg. Computing GCD using Euclids Algorithm, etc

* Insertion Sort:-

Array: 36, 29, 75, 10, 99, 7

36 > 29 and 36 < 75

29 | 36 75 10 99 7

(29: sorted; 36 75 10 99 7: unsorted)

75 > 36

29 36 | 75 10 99 7

10 < 75 & 10 < 36 10 < 29

10 29 36 75 | 99 7

10 29 36 75 99 | 7

7 10 29 36 75 99

Insertion sort → smaller set of data < 15

* Algorithm of Insertion Sort(A, n):-

```
// Sort the list in ascending order
// I/p → List A that has to be sorted
   n total no/. of elements in list.
// O/p → Sorted list

for i ← 1 to n-1 do
    Item ← A[i]  // Insert item from unsorted part
    j ← i-1      // Initial position of sorted part
    while (j >= 0 and item < A[j])
        A[j+1] ← A[j]   j ← j-1   // Finding the appropriate posn to insert item
    end while
    A[j+1] ← item   // Insert item at appropriate posn
end for
```

* Best Case Efficency:-

Best case occurs when the items in list are partially or nearly sorted. The expression item < A[j] is executed for n-1 time, then time complexity

$$T(n) = \sum_{i=1}^{n-1} 1 = (n-1) - 1 + 1 \quad (\text{upper limit} - \text{lower limit} + 1)$$
$$= n-1$$
$$= O(n)$$

* Worst case Efficiency:-

The Worst case occurs when the conditional expression item < A[j] is executed for maximum no/. of times. this situation occurs if the elements of list are sorted in reverse order.

for i ← 1 to n-1 do

:

while (j ≥ 0 and item < A[j])

$\sum_{j=0}^{i-1} 1$

$$T(n) = \sum_{i=1}^{n-1}$$

$$= \sum_{i=1}^{n-1} i-1-0+1$$

$$= \sum_{i=1}^{n-1} i$$

$$= 1+2+3+\ldots+(n-1)$$

$$= \frac{n(n-1)}{2}$$

$$= \frac{n^2}{2} - \frac{n}{2} < n^2$$

$$\therefore T(n) = O(n^2)$$

- Insertion sort satisfies Inplace and stable property
  - → Inplace :- Bcz it doesn't use any extra array i.e. space.
  - → Stable :- eg. 10 25 4 10 4
    (1) (2) (3)
    when you sort first 10 comes first and 2nd 10 comes next to it.
    i.e 4 4 10 10 25
    (1) (2) (1) (2)

* Greedy Algorithm :-

The greedy algorithm tries to find best soln for each sub problems with hope that it might yeild to an optimal solution for problem as a whole.

ments)

While solving the problems using greedy technique at each step the choice made must be,

1 Feasible :- The choice made must satisfies the constraint.

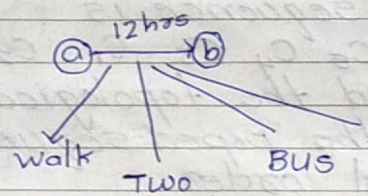
2. Local optimal - The current choice made has to be best local choice among all feasible solns.

3 Irrevocable:- Once the choice is made it should not be changed in subsequent steps of the algorithm.

Note:- Greedy Technique are applicable only for optimization problem

Greedy Technique
Dynamic
Branch-Bound
} - optimization

* eg.



According to time and cost constraints given Bus & Train are feasible soln.

* Greed Technique:-
- Minimum Spanning Tree (MST)
  - Prims
  - Kruskals
- Knapsack
- shortest path (Dijkotra's)
- Job sequencing

* Minimum Spanning Tree:-
For a given graph G = (V, E) spanning tree is a subgraph G' = (V', E') with the following properties.

1 The spanning tree should be connected and acyclic in nature

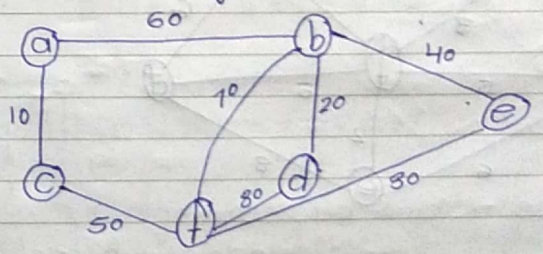
2 |V'| = |V|
No of vertices in spanning tree as of no of vertices in Graph.

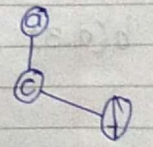
3 |E'| = |V| - 1
No of edges in spanning tree as of no of vertices in graph - 1.

Minimum spanning tree is spanning tree with minimum weight.

eg

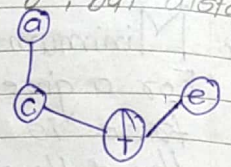
* Prims Algorithm to Find MST:-



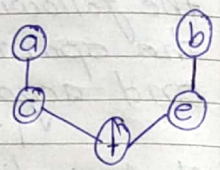
Tree Vertices	Remaining Vertices	Illustration
a(-,-)	b(a,60), c(a,10), d(a,∞) e(a,∞) f(a,∞)	
c(a,10)	b(a,60) d(a,∞) e(a,∞) f(c,50)	

Here we have taken b(a,60) as it is bcz even though f is adjacent to b, but distance is more i.e. 70

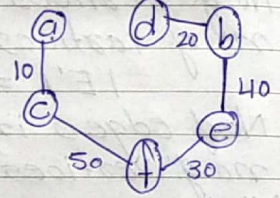
f(c,50) b(a,60) d(f,80) e(f,30)



e(f,30) b(e,40), d(f,80)

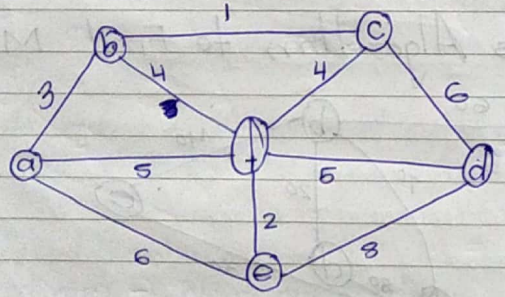


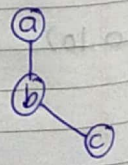
b(e,40) d(b,20)



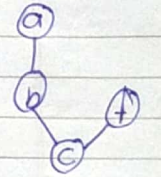
COST MST = 10+50+30+40+20 = 150

eg 2

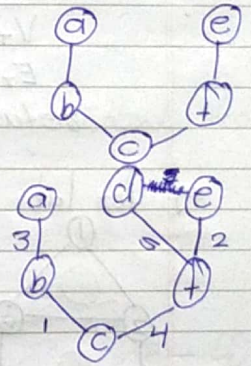


Tree Vertices	Remaining Vertices	Illustration
a(-,-)	b(a,3), c(a,∞) d(a,∞), e(a,6) f(a,5)	
b(a,3)	c(b,1) d(a,∞) e(a,6) f(b,4)	

c(b,1) d(c,6) e(a,6) f(c,4)



f(c,4) d(f,5) e(f,2)



e(f,2) d(f,5)

COST MST = 3+1+4+2+5
= 15

* Algorithm Prince (G(V,E)):-

// Prince algorithm for constructing MST
// I/P → A weighted connect graph G=(V,E)
// O/P → The set of edges E_T composing the MST of graph G.

$V_T \leftarrow \{V_0\}$ // The set of Tree vertices can be initialize with any vertex.

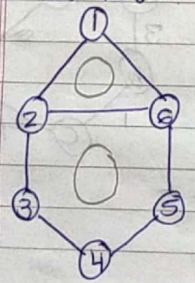
$E_T \leftarrow \phi$

for i ← 1 to |V| - 1 do

find minimum weight edge
$e^* = (u^*, v^*)$ among all the edges $e = (u, v)$
such that
The vertex v is in set of vertices V_T and u is in $V - V_T$

$V_T \leftarrow V_T \cup \{u^*\}$
$E_T \leftarrow E_T \cup \{e^*\}$
return E_T

eg for lab



No of spanning Tree $= {}^{|E|}C_{|V|-1} -$ No of cycles with vertices

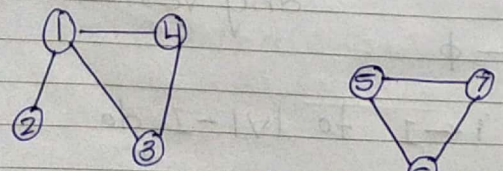
$= {}^{7}C_5 - 2$

$= 19$

Note → No of cycles
↳ cycles should not include all the vertices

* Time Complexity
In case of adjacency matrix → $\theta(|V|^2)$
In case of adjacency list → $\theta(|E| \log |V|)$

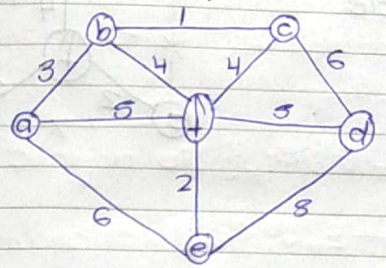
Note:-

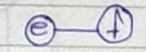
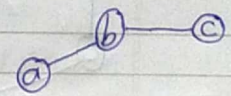
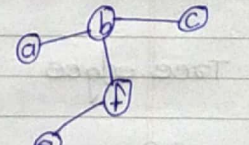
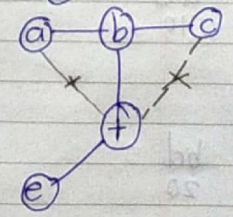
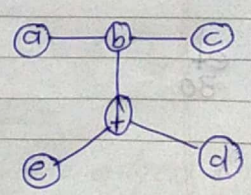


We can not find MST for this bcz it is not connected

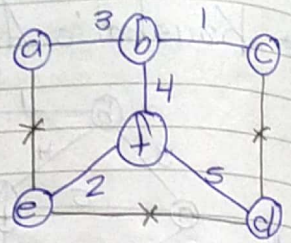
* Kruskal's Algorithm to find MST:-

eg.



Tree Edges	sorted list of edges	Illustration
bc 1	bc, ef, ab, bf, cf, af, df, ae, cd, de 1 2 3 4 4 5 5 6 6 8 X X	
ef 2		
ab 3		
bf 4		
cf 4	Not considered as it forms acycle	
af 5	Not considered as it forms cycle	
df 5		

ae
6 Not considered as it forms cycle

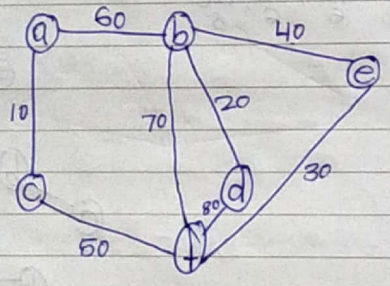


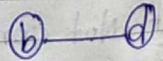
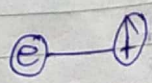
cd
6 — 11 —

de
8 — 11 —

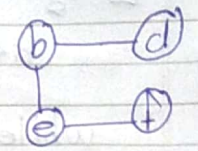
COST of MST = 1 + 3 + 2 + 4 + 5
= 15

eg 2

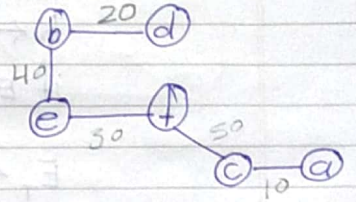


Tree edges	sorted list of edges	Illustration
ac 10	ac, bd, ef, be, cf, ab, bf, fd 10 20 30 40 50 60 70 80	
bd 20		
ef 30		

be
40



cf
50



ab Not considered
60

bf Not considered
70

fd No Considered
80

COST of MST = 20 + 40 + 30 + 50 + 10
= 150

* Algorithm of kruskal CG(V,E))

// kruskal Algorithm for MST
// A weight connected graph G(V,E)
// o/p → E_T the set of edges composing MST of graph G
sort E in increasing order of their weights
$w(e_{i_1}) \leq w(e_{i_2}) \leq \dots \leq w(e_{i_{|E|}})$

$E_T = \phi$, encounter ← 0
// Initialize the set of tree edges & its size

ae 6

$k \leftarrow 0$ // Initialize no of processing edge

while encounter $< (|V|-1)$

$k \leftarrow k+1$

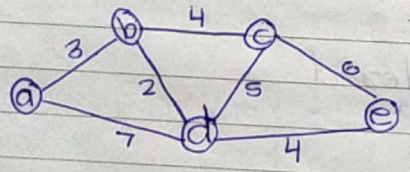
if $E_T \cup \{e_{i_k}\}$ is acyclic

$E_T \leftarrow E_T \cup \{e_{i_k}\}$, encounter $\leftarrow$ encounter+1

return E_T

* Dijkstra's Algorithm (single-source shortest path)

→ Extended version of Prim's Algorithm



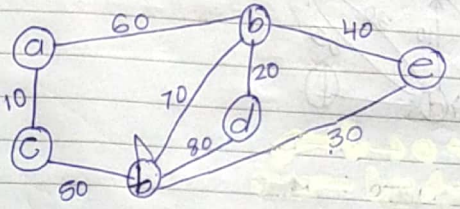
Tree Vertices	Remaining Vertices	Illustration
a(-,-)	b(a,3), c(a,∞), d(a,7), e(a,∞)	
b(a,3)	c(b, 4/3), d(b, 3+2/5), e(a,∞)	
d(b,5)	min{c(d,7+5), c(d,5+5)} e(d,5+4)	
c(b,7)	min{e(c,7+6), e(d,9)}	

For a→c : a→b→c

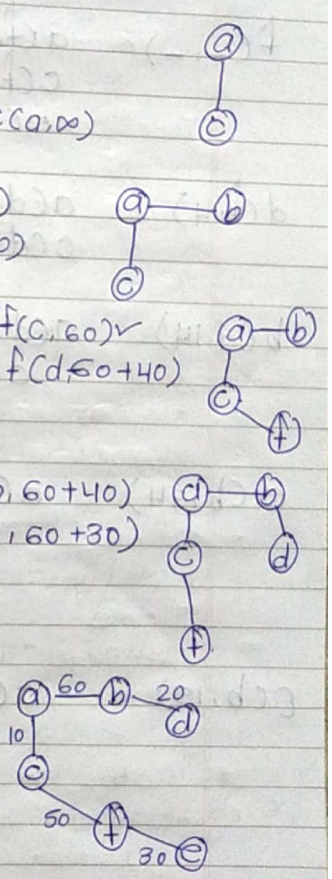
a→d : a→b→d

a→e : a→b→d→e

eg 2



a(-,-)	a(b,60) c(a,10) d(a,∞) f(a,∞) e(a,∞)	
c(a,10)	b(a,60) d(a,∞) e(a,∞) f(e,50)	
b(a,60)	{d(b,80) {d(a,∞) ∨ {e(b,100) {e(a,∞) ∨ {f(c,60) {f(d,60+40)	
f(c,60)	{d(b,80) {d(b,80+60) {e(b,60+40) {e(f,60+30)	
d(b,80)	{e(d,80+40) {e(f,60+30)	



a→b = a→b cost 60

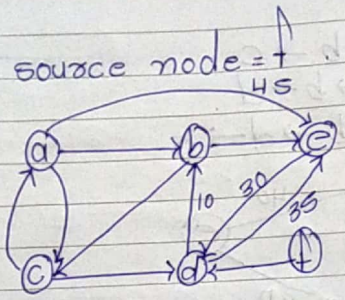
a→c = a→c cost 10

a→d = a→b→d cost 80

a→e = a→c→f→e cost 90

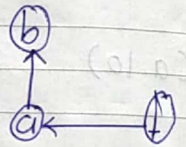
a→f = a→c→f cost 90

eg source node = f

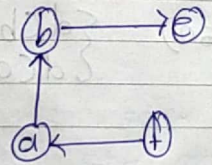


f(–,–) a(f,∞) b(f,∞)
c(f,∞) d(f,4)
e(f,∞)

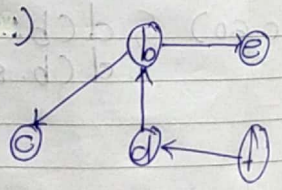
d(f,4) a(d,∞) b(d,14)
c(d,∞) e(d,39)



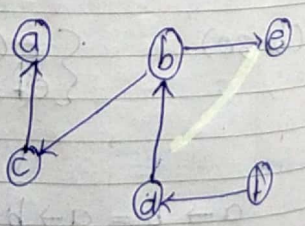
b(d,14) a(d,∞), e(b,34)
c(d,∞)



e(b,34) a(d,∞), c(b,15)
c(b,29)



c(b,15) a(c,29+20)



f → d = 4
f → d → b → e = 34
f → d → b = 14
f → d → b → c = 29
f → d → b → c → a = 49

* Knapsack Problem using Greedy Technique :-
bag, container (Fractional knapsack)

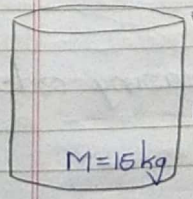
objects:	1	2	3	4	5	6	7
profits:	10	5	16	7	6	18	3
weights:	2	3	5	7	1	4	1

N = 7 → No of objects
M = 15 → Maximum capacity of knapsack

P_i/W_i :	5	1.66	3	1	(6)	4.5	3

object factor

$X = \{1, 2/3, 1, 0, 1, 1, 1\}$
$x_1, x_2, x_3, x_4, x_5, x_6, x_7$



max capacity
15 - 1 = 14
14 - 2 = 12 weight of maximum P_i/w_i
12 - 4 = 8
8 - 5 = 3
3 - 1 = 2
2 - 2 = 0

Total weight of objects inserted

$= (1*2)+(2/3*3)+(1*5)+(0*7)+(1*1)+(1*4)+(1*1)$

$= 2+2+5+0+1+4+1$

$= 15$

$\sum x_i w_i \leq M$

Constraint is

$\sum x_i w_i \leq$ Max capacity of knapsack

objective fun = $\boxed{\text{Max} \sum x_i P_i}$

Total profit $= (1*10)+(2/3*5)+(1*15)+(0*7)+(1*6)+(1*18)+(1*3)$

$= 10+\frac{10}{3}+15+6+15+3$

$= 55.35$

Q Find an optimal soln to knapsack instance with no of objects = 4 and max capacity of knapsack is 5

→ N = 4 M = 5

Items	1	2	3	4
weights	2	1	3	2
Profits	12	10	20	15

→ Step 1:-

Arrange the objects in decreasing order of profit-weight ratio.

$\frac{P_1}{w_1} = \frac{12}{2} = 6$ $\frac{P_2}{w_2} = \frac{10}{1} = 10$ $\frac{P_3}{w_3} = \frac{20}{3} = 6.66$ $\frac{P_4}{w_4} = \frac{15}{2} = 7.5$

$\frac{P_2}{w_2} > \frac{P_4}{w_4} > \frac{P_3}{w_3} > \frac{P_1}{w_1}$

Step 2:-

object (i)	w_i	P_i	$x = 1$ or $\frac{rc}{w_i}$	Profit $x \cdot P_i$	remaining capacity
–	–	–	–	–	$rc = m = 5$
x_2	1	10	$x = 1$	$1*10 = 10$	$rc = 5-1 = 4$
x_4	2	15	$x = 1$	$1*15 = 15$	$rc = 4-2 = 2$
x_3	3	20	$x = \frac{2}{3}$	$\frac{2}{3}*20 = 13.32$	$rc = 2-2 = 0$
x_1	–	–	$x = 0$	–	$2-(3*\frac{2}{3})$

Step 3:-

Total profit $= 10+15+18.32$

$= 38.32$

$X = \{0, 1, \frac{2}{3}, 1\}$

Q

objects	1	2	3	4	5	6	7
profits	10	5	15	7	6	18	3
Weights	2	3	5	7	1	4	1

$N = 7$ & $M = 15$

step 1:-

$\frac{P_1}{w_1} = \frac{10}{2} = 5$ $\quad \frac{P_2}{w_2} = \frac{5}{3} = 1.56$

$\frac{P_3}{w_3} = \frac{15}{5} = 3$ $\quad \frac{P_4}{w_4} = \frac{7}{7} = 1$

$\frac{P_5}{w_5} = \frac{6}{1} = 6$ $\quad \frac{P_6}{w_6} = \frac{18}{4} = 4.5$

$\frac{P_7}{w_7} = \frac{3}{1} = 3$

$$\frac{P_5}{w_5} > \frac{P_1}{w_1} > \frac{P_6}{w_6} > \frac{P_3}{w_3} > \frac{P_7}{w_7} > \frac{P_2}{w_2} > \frac{P_4}{w_4}$$

step 2:-

objects (i)	w_i	P_i	$x = 1$ or $\frac{rc}{w_i}$	profit $= x \cdot P_i$	$rc = rc - (w_i \cdot x_i)$
–	–	–	–	–	$rc = M = 15$
x_5	1	6	1	$1 \cdot 6 = 6$	$rc = 15 - 1 = 14$
x_1	2	10	1	$1 \cdot 10 = 10$	$rc = 14 - 2 = 12$
x_6	4	18	1	$1 \cdot 18 = 18$	$rc = 12 - 4 = 8$
x_3	5	15	1	$1 \cdot 15 = 15$	$rc = 8 - 5 = 3$
x_7	1	3	1	$1 \cdot 3 = 3$	$rc = 3 - 1 = 2$
x_2	3	5	$\frac{2}{3}$	$\frac{1}{3} \cdot 5 = 3.33$	$rc = 2 - (3 \cdot \frac{2}{3}) = 0$
x_4	–	–	0	–	–

step 3 :-

Total profit = 6 + 10 + 18 + 15 + 3 + 3.33

= 55.33

$= (2^n)^3 \cdot f(n)$

Function value is increased by $(2^n)^3$ times

* Algorithm for generating Combinatorial Objects
- Three types of Combinatorial objects are → Permutation
  Combination
  Subset of given set
- Generating Permulation:-
  1) Bottom-UP minimal change algorithm
  2) Johnsen Trotter
  3) Lexicographic Ordering

* Bottom-Up minimal change algorithm:- (Decr con)

step 1 Assume that we have solution to a given smaller problem that is we have solution for n-1 elements

step 2 The solution to the larger problem with n elements is obtained by inserting n^{th} element in various possible position in (n-1) element

step 3 All the permutations are distinct and total number obtained $n*(n-1)! = n!$

Q Generate the permutation for given set of numbers {1,2,3,4}

→ Initial : 1

Insert 1: On the left side
21
On the right side
12

Insert 3: Insert 3 for 21 from Left to right
321, 231, 213
Insert 3 for 21 from right to left
123, 132, 312

Insert 4: Insert 4 for 321 from left to right
4321, 3421, 3241, 3214
Insert 4 for 231 from L-R
4231, 2431, 2341, 2314
Insert 4 for 213 from L-R
4312, 3412, 3142, 3124

- Disadvantage:-
To get permutation of n elements, we need to know permutation of (n-1) elemen

* Johnsen Trotter:-
- This algo. is used to overcome disadvantage of minimal change algo.
- In this algorithm, permutation of n elements can be generated without knowing permutation of n-1 element.
- This can be done by using associated direction with each elements in a permutation and the direction can be left to right or from right to left
eg. $\overrightarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}$
- If element k points to smaller no. adjacent to it then element k is said to mobile.

* Algorithm for Johnson Trotter :-

1 Obtained the first permutation
$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}\ \overleftarrow{4} \ldots \overleftarrow{n}$

2 Generate subsequent permutation
while mobile integer k exists
find the largest mobile element k
swap the mobile element k with
immediate adjacent integer pointing to
by an overview mark.

reverse the direction of arrows of all
integer larger than k
end while

3 return

eg. 1 → $\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}\ \overleftarrow{4}$

4 is mobile so exchange 4 and 3

2 → $\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}\ \overleftarrow{3}$

4 is mobile so exchange 4 and 2

3 → $\overleftarrow{1}\ \overleftarrow{4}\ \overleftarrow{2}\ \overleftarrow{3}$

4 is mobile so exchange 4 and 1

4 → $\overleftarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}$

3 is mobile integer, so exchange 3 & 2
Reverse the direction of 4
(BCZ 4 & 3 is mobile elements
& 4 is greater than 3)

5 → $\overrightarrow{4}\ \overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}$

4 is mobile, so exchange 4 & 1

6 → $\overleftarrow{1}\ \overrightarrow{4}\ \overleftarrow{3}\ \overleftarrow{2}$

7 → $\overleftarrow{1}\ \overleftarrow{3}\ \overrightarrow{4}\ \overleftarrow{2}$

8 → $\overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}$

9 → $\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}\ \overrightarrow{4}$ 2<3

10 → $\overleftarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{4}$

11 → $\overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{1}\ \overrightarrow{4}$

12 → $\overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{3}\ \overrightarrow{4}$

9 → $\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}$

10 → $\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{4}\ \overleftarrow{2}$

11 → $\overleftarrow{3}\ \overleftarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}$

12 → $\overleftarrow{4}\ \overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}$

13 → $\overrightarrow{4}\ \overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}$

14 → $\overrightarrow{3}\ \overrightarrow{4}\ \overleftarrow{2}\ \overleftarrow{1}$

15 → $\overrightarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}$

16 → $\overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{4}$

17 → $\overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{1}\ \overrightarrow{4}$

18 → $\overleftarrow{2}\ \overrightarrow{3}\ \overrightarrow{4}\ \overleftarrow{1}$

19 → $\overleftarrow{2}\ \overrightarrow{4}\ \overrightarrow{3}\ \overleftarrow{1}$

20 → $\overrightarrow{4}\ \overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{1}$

21 → $\overrightarrow{4}\ \overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{3}$

22 → $\overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}\ \overrightarrow{3}$

23 → $\overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{4}\ \overrightarrow{3}$

24 → $\overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{3}\ \overrightarrow{4}$
→ No mobile element Stop i

* Dynamic Programming:-
Dynamic programming is a general algorithm for solving problems defined by recurrences with overlapping sub-problems. The main idea behind dynamic programming is
1. Set up a recurrence relating a solution of larger instance to the solutions of some smaller instances.
2. Solve smaller instances once and record the solutions in a table.
3. Extract the solution to the initial instance from that table.

* Difference between Divide & Conquer and Dynamic programming:-

Divide & Conquer	Dynamic Programming
- It's applicable when the sub problems are independent in nature	- It's applicable when sub problems are dependent in nature.
- sub problems are solved separately and combined to get the solution for the original problem.	- The original problem is solved using the results of previous sub problems
- Every instance of the sub problem is recomputed and it is not stored	- Only one instance of the sub problem is computed and stored

- If is not so efficient because of recomputations involved in it
- It was Top-Down approach
- eg Quicksort, Merge sort, Binary search

- More efficient bcz recomputations are not done.
- It uses Bottom-up approach.
- eg Warshall's algorithm, All way shortest path (Floyd's), Knapsack, TSP

* Algorithm Warshall (n, A, p) :-

// To compute the Transitive closure (path matrix)

// I/p :- Adjacency matrix A[n,n] when n is no. of vertices

// O/p :- The path matrix of size n by n

// Make a copy of adjacency matrix

```
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
        p[i,j] = A[i,j]
    end for
end for
```

// Find a transitive closure

```
for k ← 0 to n-1 do // for intermediate vert
    for i ← 0 to n-1 do // for starting vertex
        for j ← 0 to n-1 do // for target vertices
            if (p[i,j] == 0 and (p[i,k] == 1 and
                p[k,j] == 1)) then
                p[i,j] = 1
            end if
        end for
    end for
end for
return
```

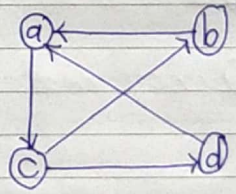
* Time Complexity :-

$$\sum_{k=0}^{n-1} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1$$

$$\sum_{k=0}^{n-1} \sum_{i=0}^{n-1} n$$

$$\sum_{k=0}^{n-1} n^2 = n^3$$

Q.



Adjacency matrix / Initial path matrix

	a	b	c	d
a	0	0	1	0
b	1	0	0	0
c	0	1	0	1
d	1	0	0	0

Intermediate vertex choosen is 'a'

	a	b	c	d
a	0	0	1	0
b	1	0	1	0
c	0	1	0	1
d	0	0	1	0

Intermediate vertex is b

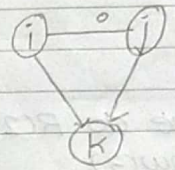
	a	b	c	d
a	0	0	1	0
b	1	0	1	0
c	1	1	1	0
d	1	0	1	0

Intermediate vertex is c

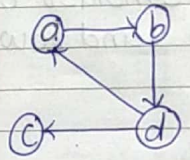
	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	1	1	1	1
d	1	1	1	1

* Warshal's Algorithm (Transitive Closure)

$a \to b$, $b \to c$ then $a \to c$



Q Apply warshal's algorithm to compute transitive closure for following graph



Adjacency Matrix is

$R(0)=$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

Consider col 1 and row 1

$(d,a)=1$ $(a,b)=1 \Rightarrow (d,b)=1$

$R(1)=$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	1	1	0

Consider col 2 and row 2

First go for column: $(a,b)=1$, $(d,b)=1$. $(b,d)=1$, $(b,d)=1$ — then check for row $\Rightarrow (a,d)=1$, $\Rightarrow (d,d)=1$

$R(2) =$

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

The matrix remains same as R(2) after considering col n(3) and row(3)

$R(3) = R(2)$

$R(3) =$

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

consider col 4 and row 4

$R(4) =$

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

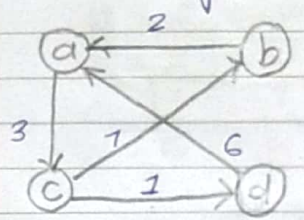
$(a,d) = 1 \;,\; (d,a) = 1 \Rightarrow (a,a) = 1$

$(b,d) = 1 \quad (d,b) = 1 \Rightarrow (b,b) = 1$

$(d,d) = 1 \quad (d,c) = 1 \Rightarrow (d,c) = 1$

* Floyd's Algorithm :-

- All pairs √ shortest path
- $D[i,j] = \min[D(i,j), D(i,k), D(k,j)]$



	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

@→

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

$b \to a = 2,\; a \to c = 3$

$\Rightarrow b \to c$

$d(b,c) = \min[d(b,c), d(b,a)+d(a,c)]$

$= \min[\infty, 2+3]$

$= \min[\infty, 5]$

$= 5$

$d \to a = 6,\; a \to c = 3$

$\therefore d \to c$

$d(d,c) = \min[d(d,c), d(d,a)+d(a,c)]$

$= \min[\infty, 6+3]$

$= 9$

	a	b	c	d
a	0	6	3	∞
b	2	0	5	∞
c	∞	7	0	1
d	6	∞	9	∞

Through B →

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	∞	7	0	1
d	6	∞	9	⊗

c → b = 7, b → a 2

→ c → a

d(c,a) = min [d(c,a), d(c,b) + d(b,a)]

= min [∞, 7+2]

= 9

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	⊗

c → b = 7, b → c = 5 } → Ruled out

c → c = 0

c → b = 7, b → d = ∞

c → d

d(c,d) = min [d(c,d), d(c,b) + d(b,d)]

= min [1, 7+∞]

= 1

Through c

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

a → c = 3, c → b = 7

a → b

d(a,b) = min [d(a,b), d(a,c) + d(c,b)]

= min [∞, 3+7]

= 10

a → c = 3, c → d = 1

a → d

d(a,d) = min [d(a,d), d(a,c) + d(c,d)]

= min [∞, 3+1]

= 4

b → c = 5, c → a = 9

b → a

d(b,a) = min [d(b,a), d(b,c) + d(c,a)]

= min [2, 5+9] → No change

= min 2

b → c = 5 c → d = 1

b → d

d(b,d) = min [d(b,d), d(b,c) + d(c,d)]

= min [∞, 5+1]

= 6

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	9	7	0	1
d	6	∞	9	0

a → Through d.

4 + 6 = 10 (a,d) (d,a) → (a,a) 0
4 + ∞ = ∞ (a,d) (d,b) → (a,b) 10
4 + 9 = 13 (a,d) (d,c) → (a,c) 3
4 + 0 = 4 (a,d) (d,d) → (a,d) 4

b → Through d.

6 + 6 = 12 (b,d) (d,a) → (b,a) 2 ×
6 + ∞ = ∞ (b,d) (d,b) → (b,b) 0
6 + 9 = 15 (b,d) (d,c) → (b,c) 5
6 + 0 = 6 (b,d) (d,d) → (b,d) 6

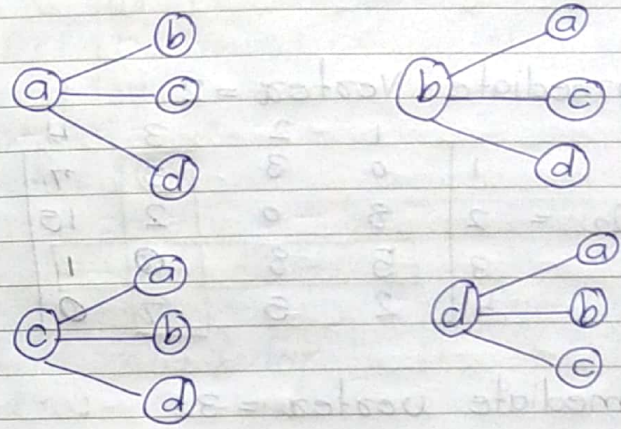
C → Through d.

1 + 6 = 7 (c,d) (d,a) → (c,a) 9 ✓
1 + ∞ = ∞ (c,d) (d,b) → (c,b) 7 ×
1 + 9 = 10 (c,d) (d,c) → (c,c) 0 ×
1 + 0 = 1 (c,d) (d,d) → (c,d) 1 ×

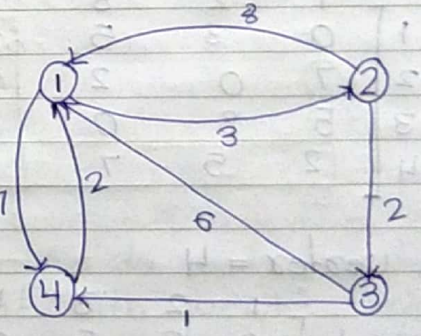
* Floyd's Algorithm
→ All pair shortest path problem.

4 Vertices

ⓐ ⓑ ⓒ ⓓ



eg 1



Adjacency Distance Matrix

	1	2	3	4
1	0	3	∞	7
2	8	0	2	∞
3	5	∞	0	1
4	2	∞	∞	0

Change columnwise

Intermediate Vertex = 1

$D_1 =$

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	1
4	2	5	∞	0

Intermediate Vertex = 2

$D_2 =$

	1	2	3	4
1	0	3	5	7
2	8	0	2	15
3	5	8	0	1
4	2	5	7	0

Intermediate vertex = 3

$D_3 =$

	1	2	3	4
1	0	3	5	6
2	7	0	2	3
3	5	8	0	1
4	2	5	7	0

Intermediate vertex = 4

$D_4 =$

	1	2	3	4
1	0	3	5	6
2	5	0	2	3
3	3	6	0	1
4	2	5	7	0

* Algorithm Floyd (A[1...n][1...n])

```
D ← A
for k ← 1 to n do // Intermediate Vertex
    for i ← 1 to n do // Source Vertex
        for j ← 1 to n do // destination Vertex
D[i,j] = min{ D[i,j], D[i,k] + D[k,j] }
    end for
end for
end for
```

* Recurrance Relation :-

$$D_{ij}^{k} = \min \{ D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)} \}$$

$$\forall\ k > 1,\ D_{ij}^{(0)} = A_{ij}$$

* 0/1 Knapsack problem using dynamic programming :-

Let **i** represent the i^{th} object to be selected or not, to get maximum profit with weight w_i and profit P_i with remaining capacity j. V[i, j] is the profit matrix obtained by considering all i^{th} object with the capacity j

Case 1 :- If there are no objects (i.e. i = 0) or the capacity of knapsack is 0 (i.e. j = 0) then profit will be 0

The total profit obtained is

$V[i,j] = 0$ if $(i = 0, j = 0)$ ⟶ (1)

Case 2:- Suppose we have situation that weight of i^{th} object w_i is greater than the remaining capacity j then, i^{th} object can not be selected. so the profit will be

$$V[i,j] = V[i-1, j] \text{ if } w_i > j \longrightarrow (2)$$

Case 3:- Suppose the weight of i^{th} object w_i is less than the remaining capacity of knapsack j. Then we have two alternatives

1. we reject i^{th} object
   $V[i,j] = V[i-1, j]$
2. we accept i^{th} object
   $V[i,j] = V[i-1, j-w_i] + P_i$

out of these situations, whichever yeilds the maximum profit that has to be ~~continued~~ considered

$$V[i,j] = \max\{V[i-1, j], V[i-1, j-w_i]+P_i\}$$
$$\text{if } w_i \leq j \longrightarrow (3)$$

* The Recurance Relation is,

$$V[i,j] = \begin{cases} 0 \\ V[i-1, j] \\ \max\{V[i-1, j], V[i-1, j-w_i]+P_i\} \end{cases}$$

eg: Apply dynamic programming technique to find the solution for following instances of knapsack.

M = Max Capacity of knapsack = 8
n = no of objects = 4
P = {1, 2, 5, 6}
W = {2, 3, 4, 5}

objects \ capacity of knapsack	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

V[n, M] = Max profit

P_i	W_i
1	2
2	3 = 5
5	4
6	5

- $P_i = 1$ is inserted. bcz first object weight is less than knapsack capacity so it is inserted at knapsack capacity 2
- First object and 2nd object is $2+3=5$. so it will fit till 4 as separately

Hw → M = 5 n = 4
W = {2, 1, 3, 2}
P = {12, 10, 20, 15}

P_i	W_i
12	2
10	1
20	3
15	2

P_i	W_i
10	1
12	2
15	2
20	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	10	10	10	10	10
2	0	10	12	22	22	22
3	0	10	15	25	27	37
4	0	10	15	20	30	37

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

V[4,5]
V[n,m]

$V[i,j] = max\{V[i-1,j],\ V[i-1,j-w_i]+P_i\}$

$V[4,2] = max\{V(3,2),\ V(3,0)+15\}$

$V[4,3] = max\{V(3,3),\ V(3,1)+P_i\}$
22 30 +15

$V[i,j] = V[4,5]$

$j=5$ when 4th object is inserted whose weight $W_4=2$ remaining capacity

$j = j - w_4 = 5-2 = 3$

$i = i-1$

=

Way of writing { Recurrence Relation, Matrix, steps

Given $i=4$ & $j=5$

$w_1=2$ $w_2=1$ $w_3=3$ $w_4=2$

To start with let $\{x_1, x_2, x_3, x_4\} = \{0,0,0,0\}$ indicates no object is selected.

step1:-

V[4,5] V[3,5]

37 32 both are diffrent

∴ object 4 is selected

∴ $x_4 = 1$

since object 4 is selected. remaining capacity is $j = 5 - w_4 = 5-2 = 3$

$i = i-1 = 4-1 = 3$

consider $V[i,j] = (3,3)$

V[3,3] V[2,3]

22 22 both are same

so 3rd object is not selected ∴ $x_3 = 0$ $1 \neq 1-1 \neq 3-1 \neq 2$

Since object 3 is not selected. remaining capacity is $j=3$ itself. $i = i-1 = 3-1 = 2$

so consider $V[i,j] = (2,3)$

V[2,3] V[1,3]

22 12 both are different

so 2nd object is selected

∴ $x_2 = 1$

since object 2 is selected. remaining capacity is $j = 3 - w_2 = 3-1 = 2$

$i = i-1 = 2-1 = 1$

consider $V[i,j] = (1,2)$

V[1,2] V[0,2]

12 0 both are different

so 1st object is selected

∴ $x_1 = 1$

since object 1 is selected. remaining capacity is $j = 2 - w_1 = 2-2$, $i = i-1 = 1-1 = 0$

* Time Compleaity Analysis

```
for i←0 to n do
{
    for j←0 to m do
    {
        V[i,j] = max{V[i,j], V[i-1,j-W_i] + P_i}
    }
}
```

$$\sum_{i=0}^{n} \sum_{j=0}^{m} 1$$

$$\sum_{i=0}^{n} (m-0+1) = m+1 \sum_{i=0}^{n} 1$$

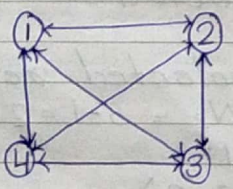
$$= (m+1)(n-0+1)$$

$$= (m+1)(n+1) \quad \text{(bigger one)}$$

$$= mn+m+n+1$$

$$= \underline{\theta(mn)}$$

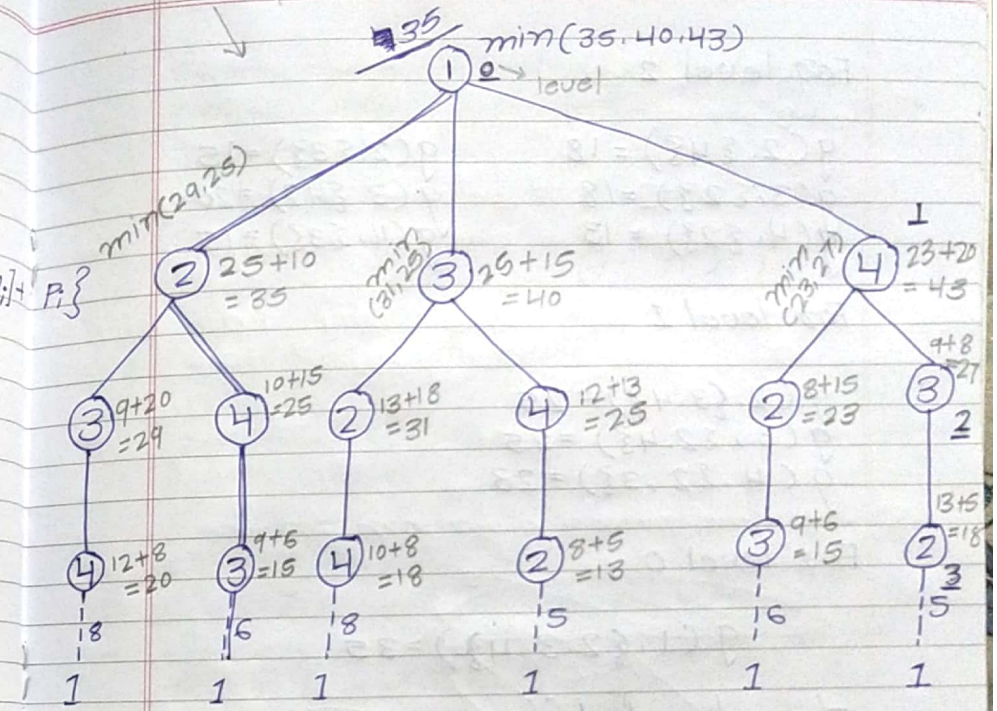
* Travelling Salesman Problem using dynamic programming:-



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

Find Hamiltonean cycle with shortest distance.

TSP using brute force



TSP using dynamic programming

* Recurrence Relation for TSP using DP:-

$$g(i, S) = \min_{k \in S} \{C_{ik} + g(k, S - \{k\})\}$$

$$g\left(i, \underset{i \notin \{2,3,4\}}{S}\right) = \min_{k \in S} \begin{cases} C_{12} + g(2, \{3,4\}) \\ C_{13} + g(3, \{2,4\}) \\ C_{14} + g(4, \{2,3\}) \end{cases}$$

$$(2,3,4)$$

$g(2, \phi) = 5$
$g(3, \phi) = 6$
$g(4, \phi) = 8$

— For the last stage in tree (level-3)

For level 2

$g(2, \{4\}) = 18$ $\quad$ $g(2, \{3\}) = 15$

$g(3, \{2\}) = 18$ $\quad$ $g(3, \{4\}) = 20$

$g(4, \{2\}) = 13$ $\quad$ $g(4, \{3\}) = 15$

For level 1

$g(2, \{3, 4\}) = 25$

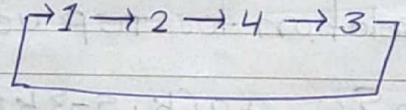
$g(3, \{2, 4\}) = 25$

$g(4, \{2, 3\}) = 23$

For level 0

$$g(1, \{2, 3, 4\}) = 35$$

The optimal tour is



* Time Complexity :-

As we know that Dynamic programming approach contains subproblems

In TSP, after reaching the i^{th} node finding remaining minimum distance to that i^{th} node is a subproblem.

If we solve recursive iteration we will get total $(n-1) \cdot 2^{(n-2)}$

$(n-1) \cdot 2^{(n-2)} \rightarrow$ subproblems we are getting, which is nearly equal to $O(n \cdot 2^n)$

Each subproblem will take $O(n)$ times (Finding the path to remaining $n-1$ nodes).

Total Complexity $= O(n \cdot 2^n) * O(n)$

$$= O(n^2 2^n)$$

* Generating Subsets :-

- Decrease by one idea is applicable for generating subsets, all the subsets of $A = [a_1, a_2, \ldots, a_n]$ can be divided into 2 groups,

1) That contain a_n

2) That do not contain a_n - $\{a_1, a_2, \ldots a_{n-1}\}$

- We can get first group by adding the n^{th} element i.e. a_n to all the subsets of 2^{nd} group.

- eg. $A = \{a_1, a_2, a_3\}$

n	Subsets
0	ϕ
1	$\phi, \{a_1\}$
2	$\phi, \{a_1\}, \{a_2\}, \{a_1, a_2\}$
3	$\phi, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_2, a_3\}, \{a_1, a_3\}, \{a_1, a_2, a_3\}$

- A convinent way of solving the problem directly is based 1-1 correspondence blw 2^n subsets of n^{th} element set $A = \{a_1, a_2, a_3\}$ and 2^n bit strings $b_1 b_2 b_3$ of length n.

$A = \{a_1, a_2, a_3\}$

bit strings

$b_i = 1$ if it belongs to the subset

$b_i = 0$ if it doesn't belongs to subset

bit strings	000	001	010	011	100	101	110	111
	ϕ	$\{a_3\}$	$\{a_2\}$	$\{a_2, a_3\}$	$\{a_1\}$	$\{a_1, a_3\}$	$\{a_1, a_2\}$	$\{a_1, a_2, a_3\}$