GREEDY TECHNIQUE

NOTE:- Most of the problems in greedy contain 'n' no. of inputs and our objective is finding a subset which will satisfy our conditions and optimizes our goal.

Basics of Greedy Technique :-

1.) Solution space:- set of all posible solutions over the given input is known as solution space.

2.) feasible solution:- set of all those solutions, which will satisfy our conditions is known as feasible solution.

3.) Optimal solution:- Those feasible solution, which will optimiss our goal is known as optimal solution.

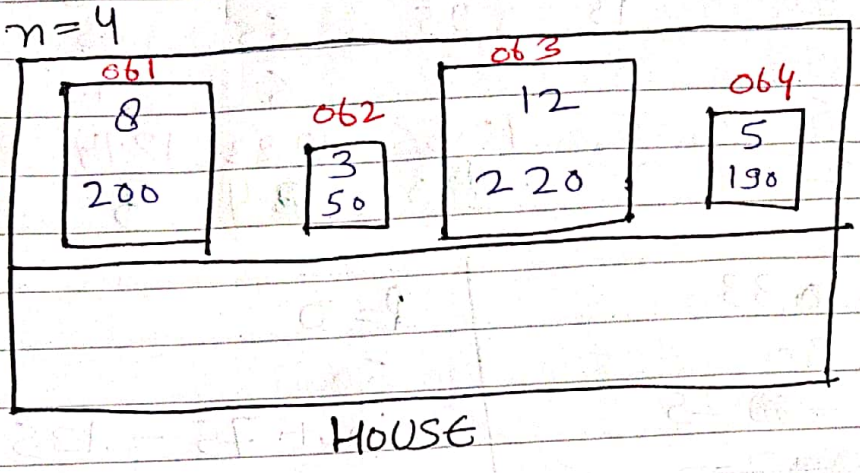
APPLICATIONS OF GREEDY TECHNIQUE:-

1. Knapsack
2. Job Sequencing with deadlines
3. Huffman Coding
4. Optimal Merge Pattern
5. Minimum Cost Spanning Tree
   (i) Prisms (ii) Kruskal
6. Single Source Shortest Path
   i.)
   ii.)
   iii.)

1. KNAPSACK PROBLEM

Ex 1

$n = 4$



$m = 24$ (Knapsack)

$\left(\frac{1}{x_1} \; \frac{0}{x_2} \; \frac{11/12}{x_3} \; \frac{1}{x_4}\right)$

Greedy about both (Profit & Weight)

o61 : $8 \rightarrow 200$

$1 \rightarrow \frac{200}{8} = 25$

o62 : $3 \rightarrow 50$

$1 \rightarrow \frac{50}{3} = 16.6$

o63 : $12 \rightarrow 220$

$1 \rightarrow \frac{220}{12} = 18.3$

o64 : $5 \rightarrow 190$

$1 \rightarrow \frac{190}{5} = 38$

$M = 24$

$24 - 1 \times 5 = 19$

$19 - 1 \times 8 = 11$

$11 - \frac{11}{12} \times 12 = 0$

$P = 0$

$0 + 1 \times 190 \Rightarrow 190$

$190 + 1 \times 200 \Rightarrow 390$

$390 + \frac{11}{12} \times 220 = 591.66$ ∴

x_1 x_2 x_3 x_4 x_5 x_6 x_7

(ex2:) $n = 7$, $m = 33$

Objects:	Ob_1	Ob_2	Ob_3	Ob_4	Ob_5	Ob_6	Ob_7
profits:	75	15	95	65	85	70	60
weight:	5	3	9	6	7	8	3
P.P.U: $O(n)$	15	5	10.56	10.83	12.14	8.75	20
$O(n\log n)$ {mergesort}	2	7	5	4	3	6	1

$O(n)$

$m = 33$

$\therefore m = 30$

$30 - 5 = 25$

~~$15 - 7 =$~~

$25 - 7 = 18$

$18 - 6 = 12$

$12 - 9 = 3$

$3 - \frac{3}{8} \times 8 = 0$

$P = 0$

60

$60 + 75 = 135$

$135 + 85 = 220$

$220 + 65 = 285$

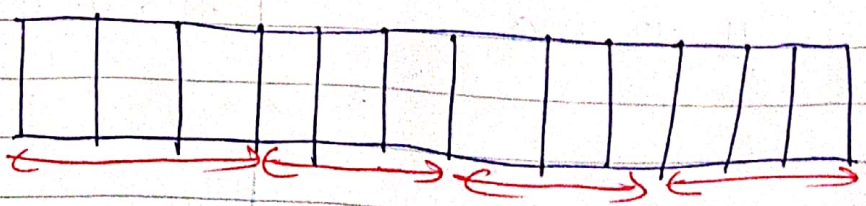
$285 + 95 = 380$

$380 + \frac{3}{8} \times 70 \Rightarrow 406.25$

1	0	1	1	1	$\frac{3}{8}$	1
x_1	x_2	x_3	x_4	x_5	x_6	x_7

1. $O(n)$
2. $O(n\log n)$
3. $O(n)$

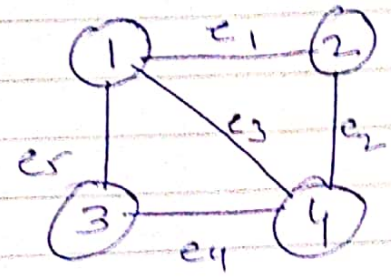
$O(n\log n)$ [EC]



SINGLE SOURCE SHORTEST PATH :-

$G(V, E)$ → set of edges
↳ vertices

$V = \{1, 2, 3, 4\}$
$E = \{e_1, e_2, \ldots e_5\}$



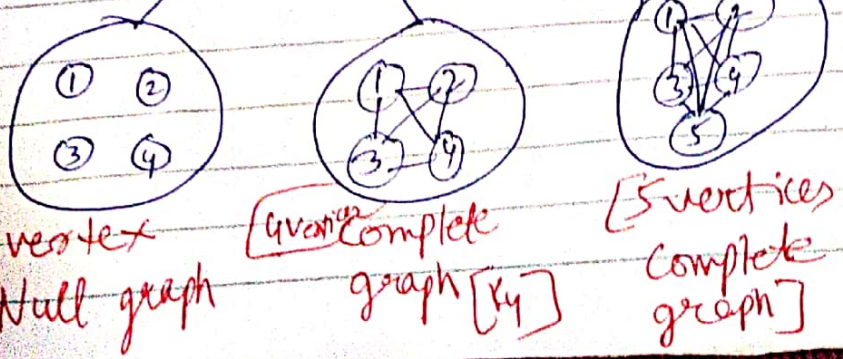
Types of graph

Simple	Multi graph
1.) No self loop	1.) Self loop
2.) No parallel edge	2.) Parallel Edge.

→ A simple graph with N vertices, Maximum degree of any vertices (n-1) & minimum degree '0'.
→ A Multigraph with N vertices, can have Maximum degree of any vertices ∞, minimum degree '0'.

TYPES OF SIMPLE GRAPH:

sum of degrees
4×3 = 6 edges
2



4 vertex Null graph

[4 vertex complete graph K_4]

[5 vertices complete graph]

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

Let $G(V, E)$ be a Simple Graph.

$$|E| \leq \frac{V(V-1)}{2}.$$

$$\leq C \cdot V^2$$

$$E = O(V^2)$$

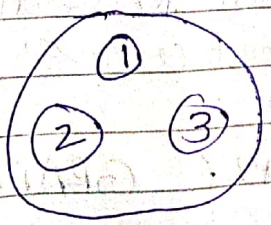
$$\log E = O(\log V)$$

$$V \log E$$
$$\Downarrow$$
$$O(V \log V)$$

No. of simple graphs possible with 'n' vertices.

$n = 3$

Max edge $= \frac{3 \times 2}{2} = 3$



e_1	e_2	e_3
0	0	0
0	0	1
0	1	0
1	0	0
1	0	1
1	1	0
0	1	1
1	1	1

Space complexity :- $O(V^2)$ – Adjancy Matrix
$O(V+E)$ – Adjancency List

$n=5$

Max edges $= \frac{5 \times 4}{2} = 10$ edges.

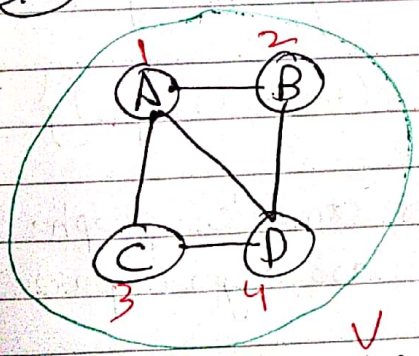
2^{10} - simple graphs.

No. of simple graphs using n vertices ⇒ $2^{\frac{n(n-1)}{2}}$.

GRAPH REPRESENTATIONS:-

1. Adjancency Matrix 2.) Adjancency List

Ex-1



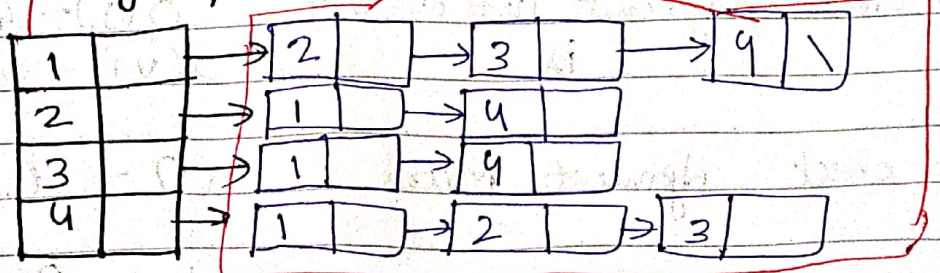
	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

→ $O(V^2)$ same {every case}

V

Adjancy List

2E

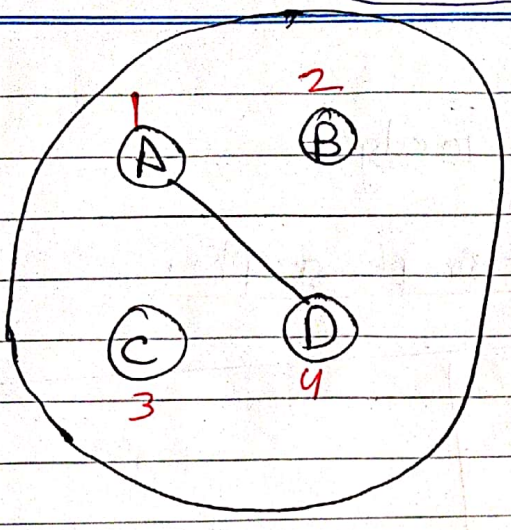


$= V+2E$
$= O(V+E)$

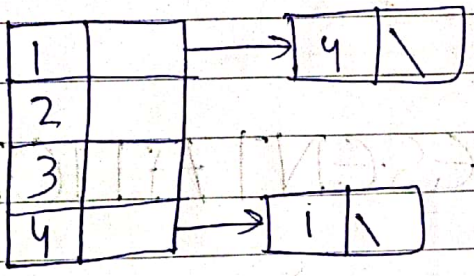
By default Adjancency List.

Less edges - sparse graph
More edge - (complete graph) — Dense graph.

Ex-2



	1	2	3	4
1	0	0	0	1
2	0	0	0	0
3	0	0	0	0
4	1	0	0	0



Adjancy Matrix Representation → $O(V^2)$ space [Every Case]

{ It is best for more edges — (Dense Graph) }

NOTE :— → Adjancency Matrix is better for complete graph.
→ Adjancency List is better for less edges. (Sparse graph).

{ → To check adjacent list or not, $O(1)$ - best
in , $O(V-1)$ - worst }

{ → To check adjacent Matrix, $O(1)$ — in every case }

{ Degree of adjancency Matrix is - $O(V)$ E.C.
List is —
BC — $O(1)$
WC - $O(V)$ }

Directed Graph

Vertex → Universal Sink.

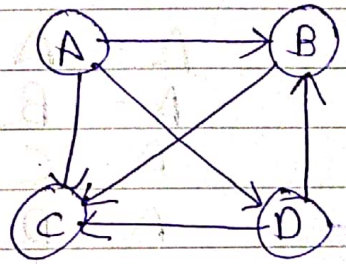
- outdegree = 0
- indegree = (N-1)

In the given directed graph, when outdegree of vertex = 0 & indegree of vertex = (N-1), the graph is said to be Universal Sink.

To find universal sink, time Complexity = ?

{ Graph Represented using Matrix }.

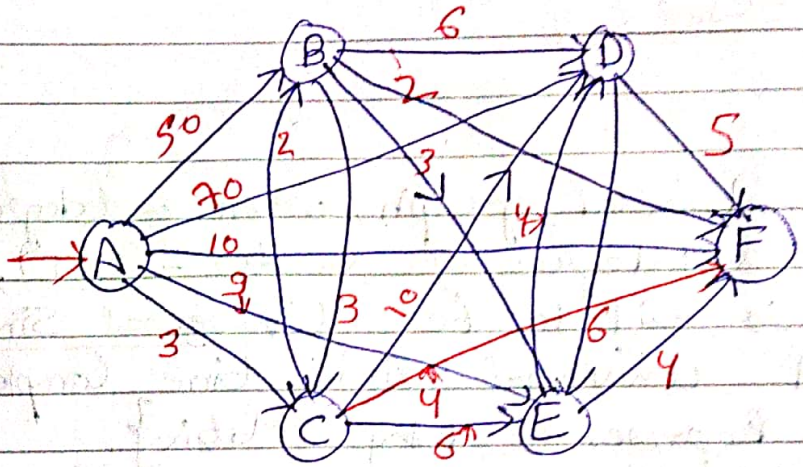
outdegree → \ indegree ↓	A	B	C	D
A	0	1	1	1
B	0	0	1	0
C	0	0	0	0
D	0	1	1	0



$= O(V)$.

DIJKSTRA'S ALGORITHM

Q.) Consider the following graph



A - A = 0
A - B = 5
A - C = 3
A - D = 11
A - E = 8
A - F = 7

Dijkstra's

Present in min heap

	A	B	C	D	E	F	
	0	∞	∞	∞	∞	∞	⇒ O(V)
	N	N	N	N	N	N	
A (log V)		50 A	2 A	70 A	9 A	10 A	⇒ 5 + 5 log V
C (log V)		5 C		13 C	9 A	9 C	⇒ 4 + 4 log V
B (log V)				11 B	8 B	7 B	⇒ 4 + 0 log V
F (log V)				11 B	8 B		⇒ 2 + 0 log V
E (log V)					10 E		⇒ 2 + 10 log V
D (log V)							⇒ 2 + 0 log V

V (log V) E

E + E log V

NOTE :— In min heap or maxheap increase key or decrease key opⁿ. will take
$O(\log n)$ — average & worst case
$O(1)$ — best case

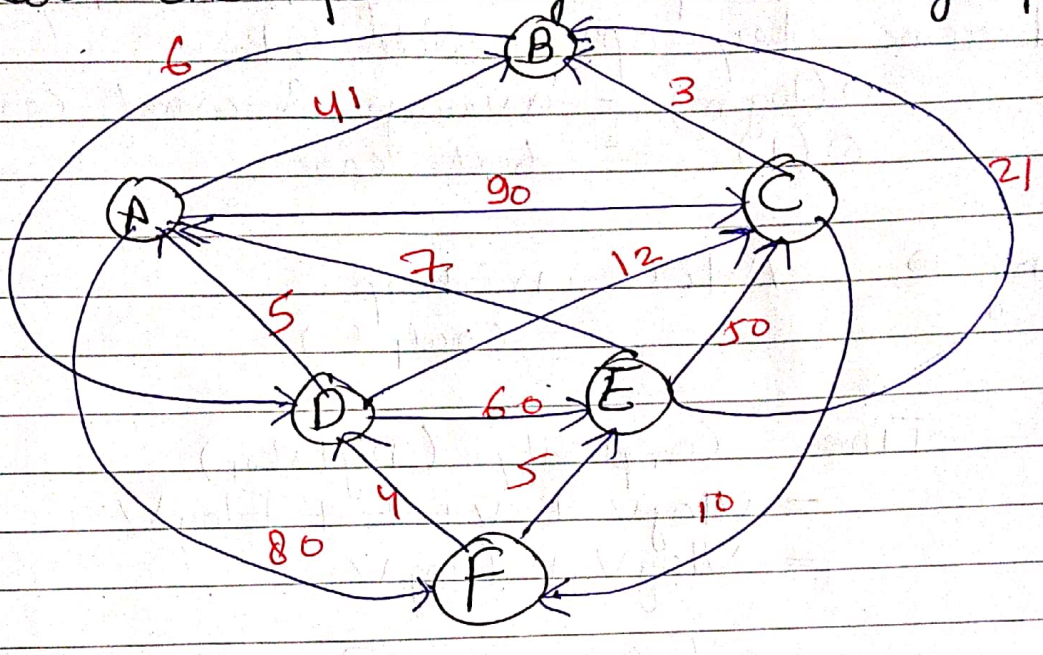
① Using ⇒ A.list, minheap (output)

Time Complexity (Dijkstra)
$= V\log V + V + E + E\log V$
$= V\log V + E\log V$
$= O[(V+E)\log V]$

② Using A.Matrix, minheap

$TC(\text{Dijkstra}) = V\log V + V + V^2 + E\log V$
$= V^2 + E\log V$ [$\because E = V^2$, $E = O$]
$= O[V^2 + E\log V]$

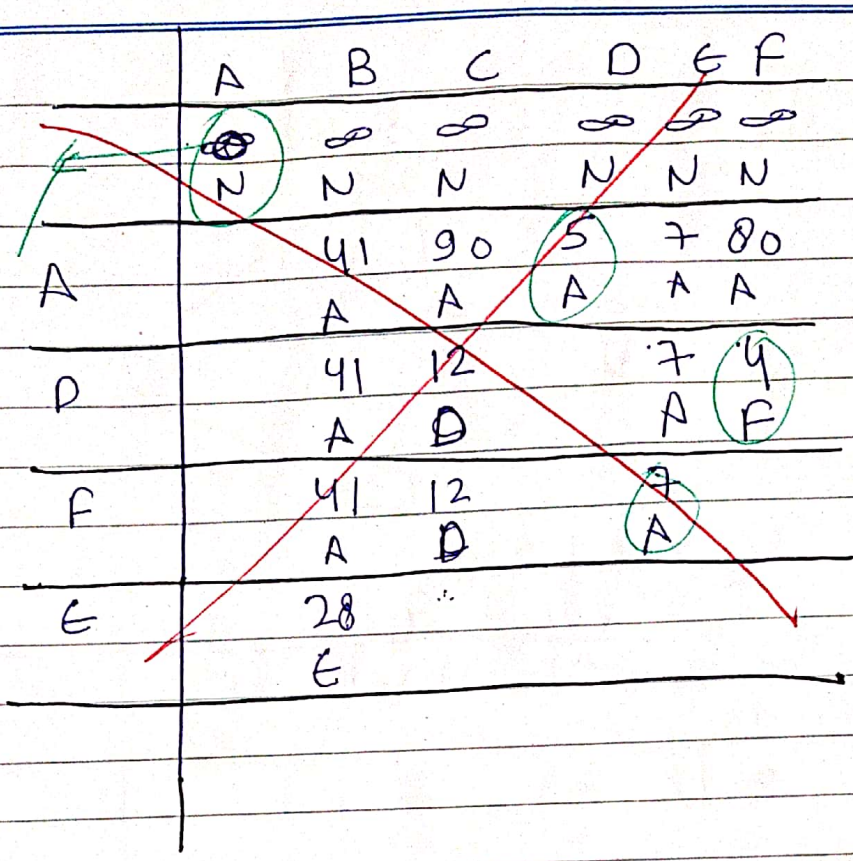
Consider the following directed graph



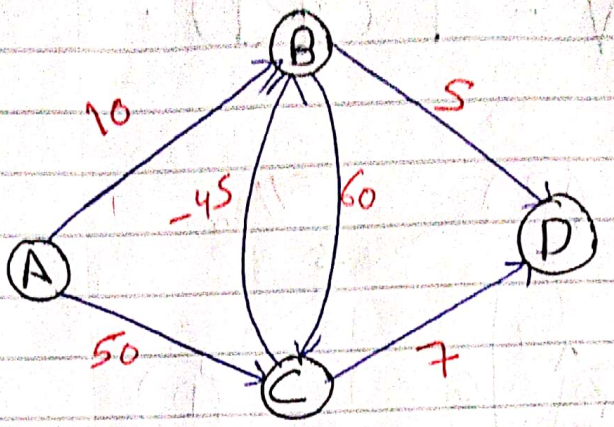
(i.) Print the sequence of vertices identified by Dijkstra's algorithm when algo starts from vertex A.

(ii.) what will be the cost of shortest path from A to E.

(iii.) what is the shortest path from A to E.



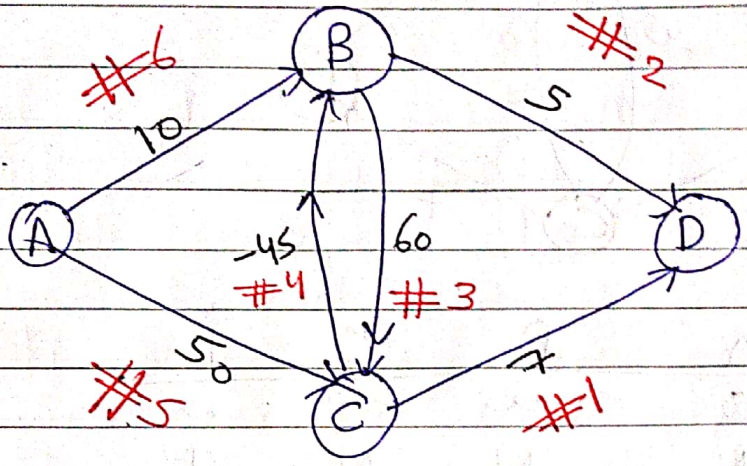
Ex-3.)



	A	B	C	D
	0 N	∞ N	∞ N	∞ N
A		10 A	50 A	∞ N
B			50 A	15 B
D			50 A	
C				

(B & D values are wrong using Dijkstra's Algorithm).
In dijkstra's every vertex will be relaxed once.

BELLMAN - FORD :-



Result in array

	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	10	50	∞
	N	A	A	N
i=2	0	5	50	15
	N	C	A	B
i=3	0	5	50	10
	N	C	A	B
i=4	0	5	50	10
	N	C	A	B

$\Rightarrow E \times O(1) \Rightarrow O(E)$

$\Rightarrow E \times O(1) \Rightarrow O(E)$

$\Rightarrow E \times O(1) \Rightarrow O(E)$

$\Rightarrow E \times O(1) \Rightarrow O(E)$

$\Rightarrow E \times O(1) \Rightarrow O(E)$

V-1

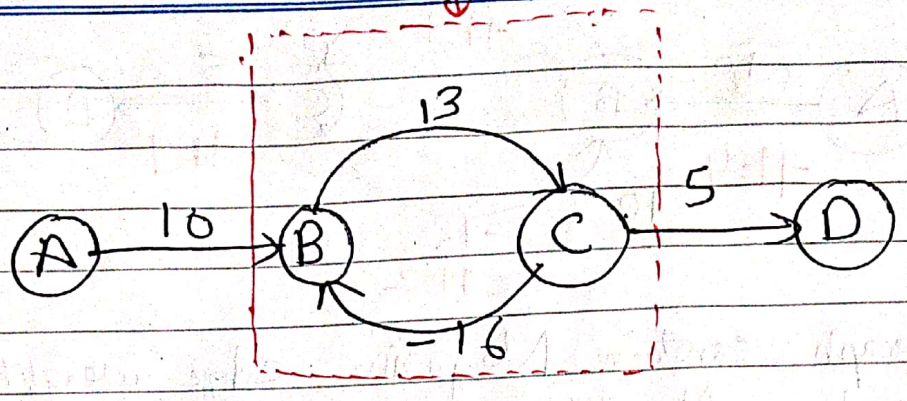
for verification

$O(VE)$

→ Graph contains all positive edge weight then Dijkstras Algorithm always do correct answer, otherwise Dijkstras Algorithm may fail.

negative edge weight cycle.

(ex-4.)



Dijstras:-

	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
A	~~0~~	10	∞	∞
	~~N~~	A	N	N
B			23	∞
			B	N
C				28
				C

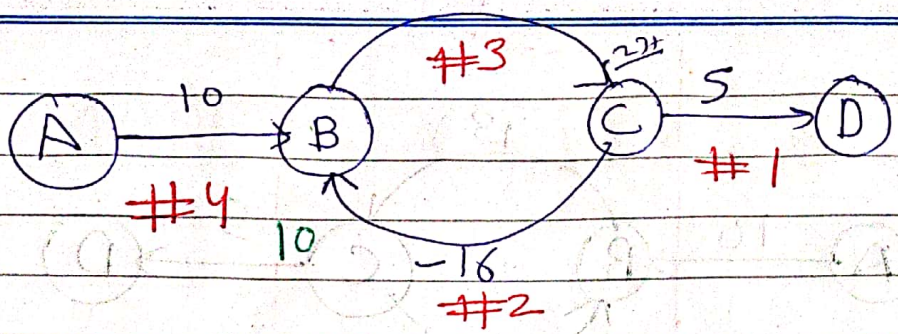
B is wrong, thats why C & D are also wrong (Depend on B).

BellMan ford :-

		A	B	C	D	
		0	∞	∞	∞	
		N	N	N	N	
V-1	i=1	0	10	∞	∞	⇒O(E)
		N	A	N	N	
	i=2	0	10	23	∞	⇒O(E)
		N	A	B	N	
	i=3	0	7	20	28	⇒O(E)
		N	C	B	C	
verification	i=4	0	4	17	25	⇒O(E)
		N	C	B	C	O(VE)

Note ⇒ If in verification case, if any value is changing, there is a negative edge weight cycle. ⇒O(VE)

-∞ -∞ -∞



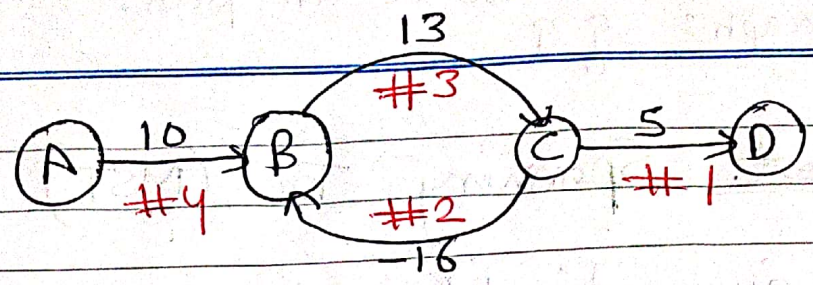
NOTE:- If graph contain Negative edge weight cycle, DijKstra's Algorithm will always fail.

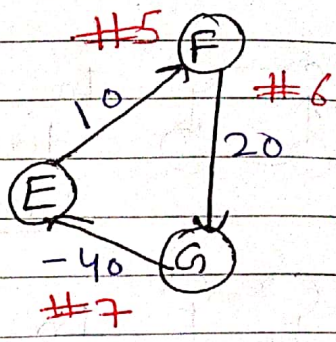
→ Graph contain only negative edge weight, but not -ve edge weight cycle, DijKstra's Algorithm may fail.

→ Graph contain +ve edge weighed, -ve edge weight or both, but not -ve weight cycle, Bellman ford will always give correct answer.

→ If graph contain -ve edge weight cycle, Bellman ford Algo will give correct answers for those vertices which dont have ~~negative~~ any relation with -ve edge wt cycle, for other vertices answers is Undefined.

NOTE :- Bellman ford Algori will find out all the -ve edge weighted cycle present in the given graph if they are reachable from source.





	A	B	C	D	E	F	G
	0 N	∞ N	∞ N	∞ N	∞ N	∞ N	∞ N
i=1	0 N	10 A	∞ N	∞ N	∞ N	∞ N	∞ N
i=2	0 N	10 A	23 B	∞ N	∞ N	∞ N	∞ N
i=3	0 N	7 C	20 B	28 C	∞ N	∞ N	∞ N
i=4	0 N	7 C	17 B	25 C	∞ N	∞ N	∞ N
i=1							
i=6	0 N	-∞	-∞	-∞	∞ N	∞ N	∞ N

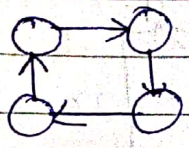
Bellman ford algo can't Detect the negative weight cycle, because it is not reachable from source.

⇒ Bellmand ford Algo will findout all -ve edge weight cycle present in the given if they are reachable from source.

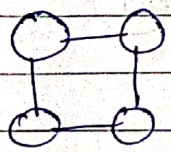
→ Connected graph: In the give undirected graph b/w two vertices there must be an path.

→ Disconnected graph: 9 8

#Minimum Cost Spanning Tree (MST)



In the given directed graph without removing direction if graph is connected it is strongly connected graph.

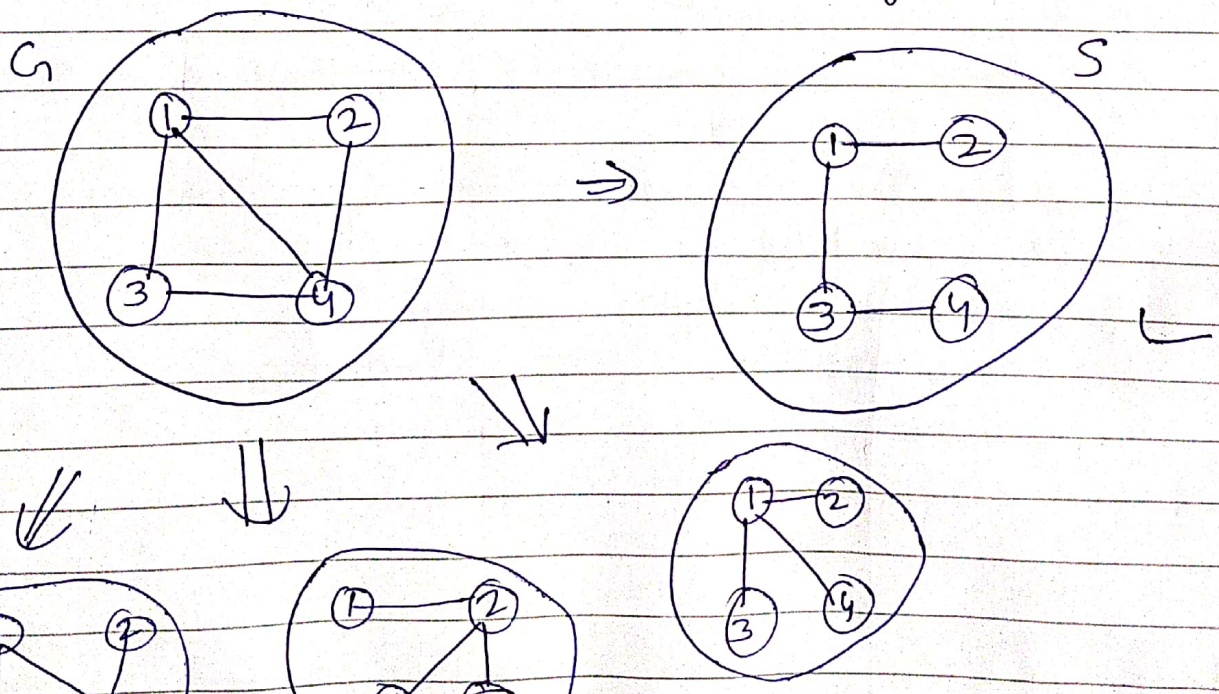


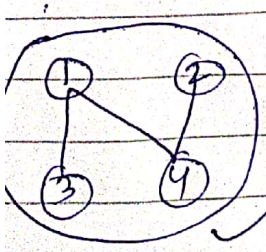
In the given directed graph after removing direction if graph is connected then it is weekly connected graph.

⇒ **Spanning Tree**

A subgraph 's' of the given graph 'G' is said to be spanning tree if and only if.

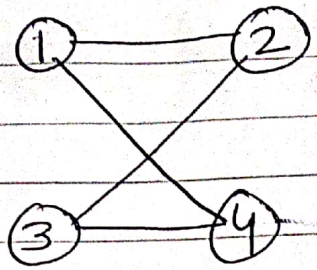
1. Subgraph 's' should contain all vertices of 'G'
2. Subgraph 's' should contain (v-1) edges without cycle, where 'v' is no. of vertices.



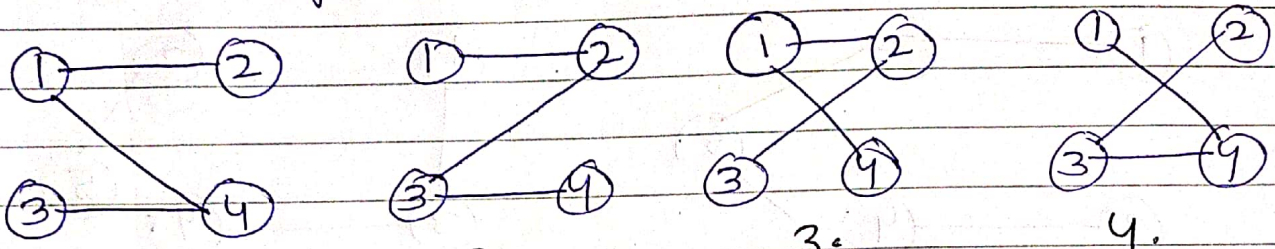


✗ Because there is no edge b/w 3, 2 NOT subgraph in graph 'G', so you can't take in subgraph 'S'.

Ex-2

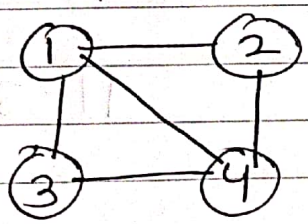


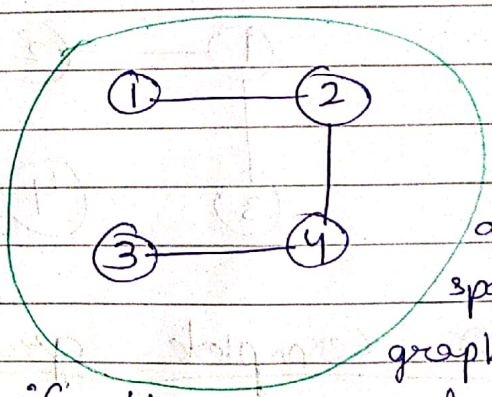
Spanning graph possible are:-



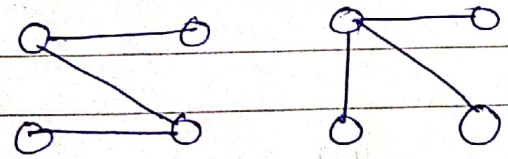
1. 2. 3. 4.

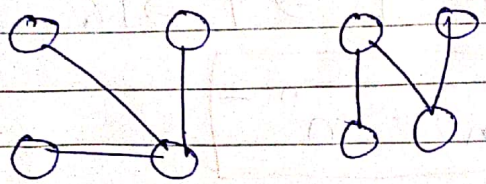
Ex-3





rotate 90° and get spanning graph.

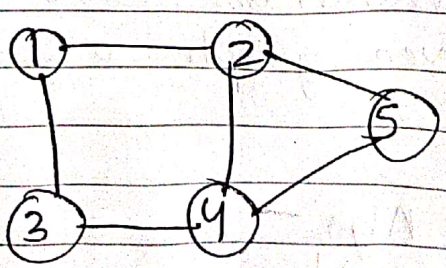


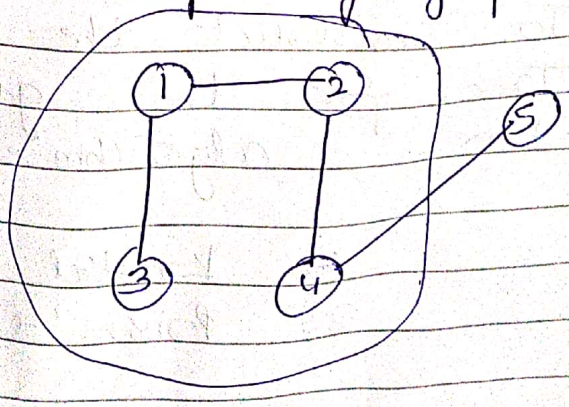


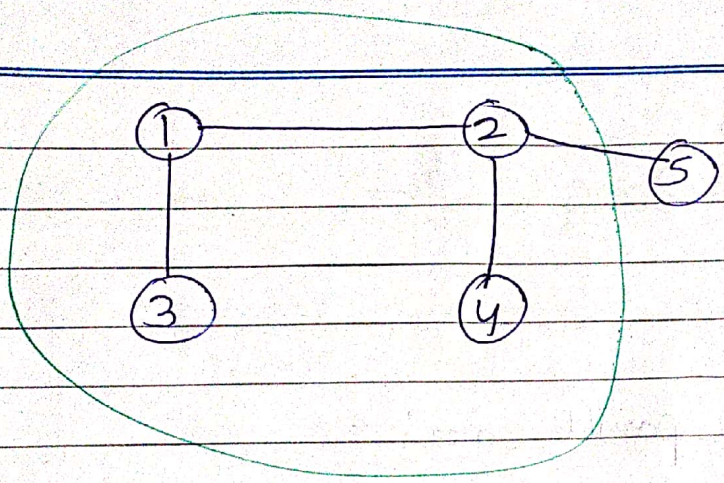
if these are cycle in graph take that cycle. and remove one edge and rotate 90° every time you get one spanning graph.

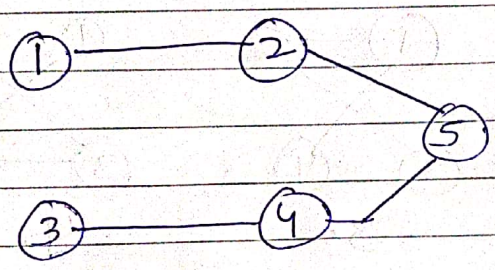
4+4=8 Spanning graph.

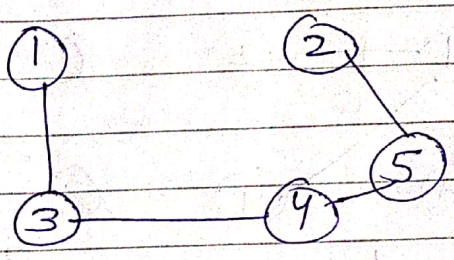
Ex-4

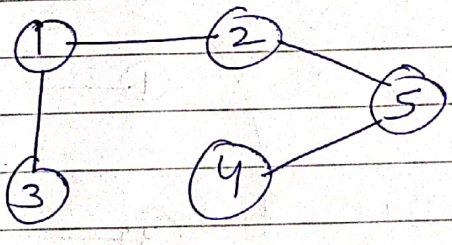






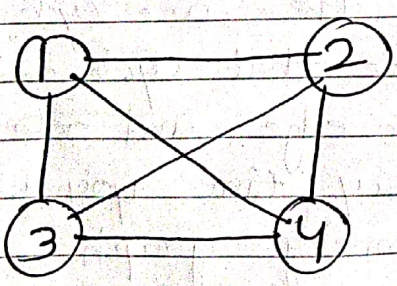






11-ST

For complete graph:-



$ST(K_4) = 4^{4-2} \Rightarrow 4^2 \Rightarrow 16$

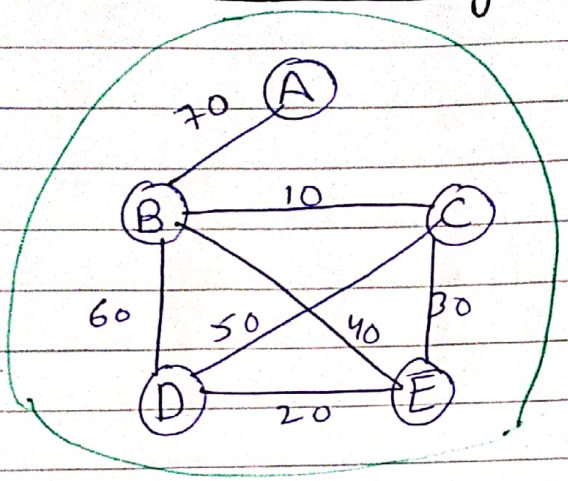
$$ST(K_n) = n^{n-2}$$

n - vertices.

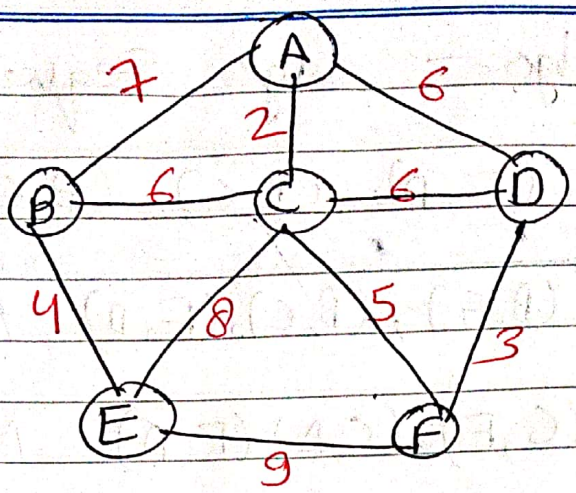
→ To construct the minimum cost spanning Tree, for the given graph we have two algorithm.

1.) Kruskal Algo
2.) Prism's Algo

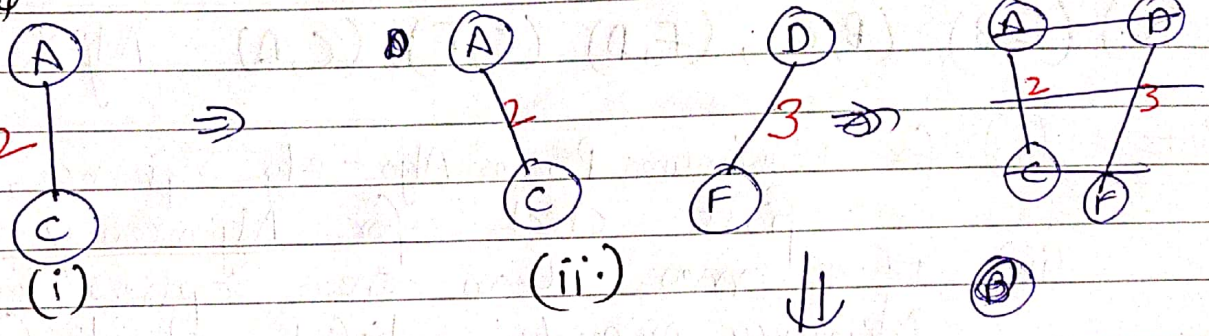
Kruskal Algorithm:-

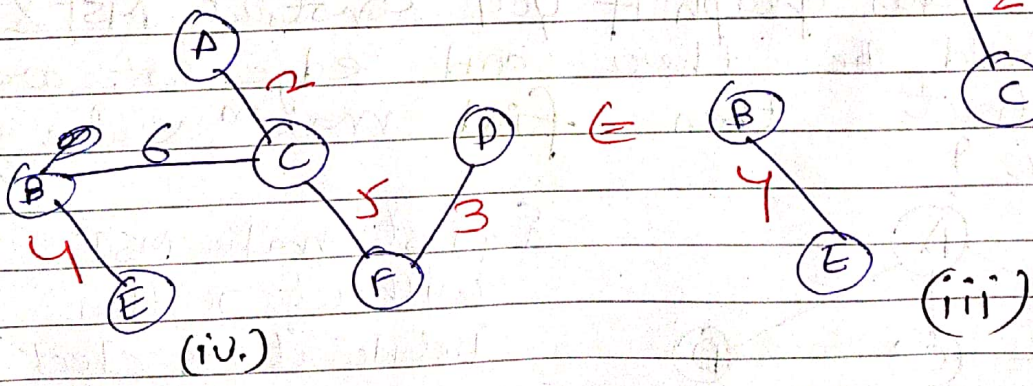


Ex-2

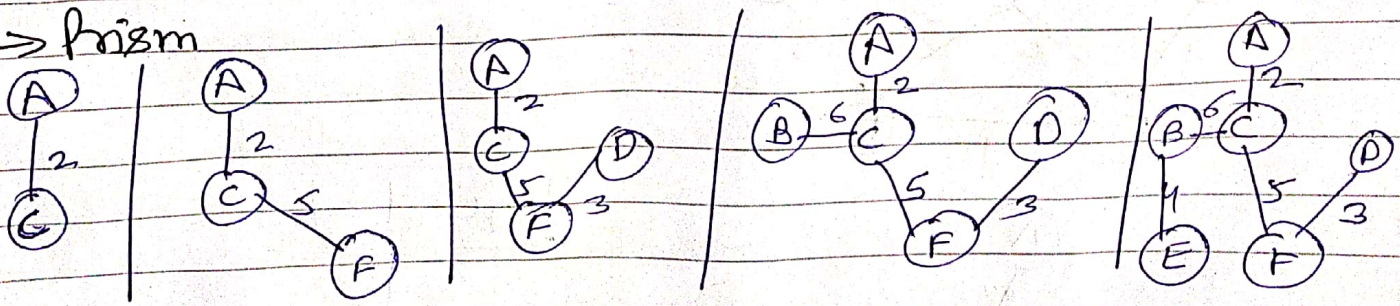


→ Kruskal





→ Prism



use this method

ex3) Prisms - Algo - edge Sequence?

i.) (A,C), (C,F), (F,D), (B,C), (B,E) Adj ✓ min ✓

ii.) (A,C), (C,F), (B,E), (B,C), (F,D) Adj ✗

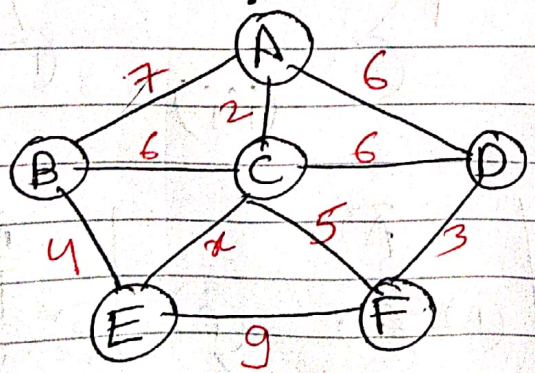
iii.) (E,B), (B,C) (C,F) (C,A) (F,D) Adj ✓

iv.) (E,B), (B,C), (F,D) (C,F), (C,A) Adj ✗

NOTE:- (i.) for minimum Prisms-Algo - edge Sequence, first check for Adjamency.

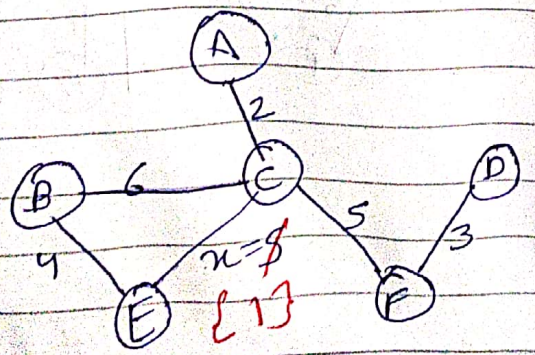
(ii.) If more than one options have Adjamency property satisfied, check for min. (minimum distance).

Ex-4) for this graph if you construct MST & x should be there and edge wt are distinct then find max^m value of x ?



{first make MST without x, then include it & check}

x = 1



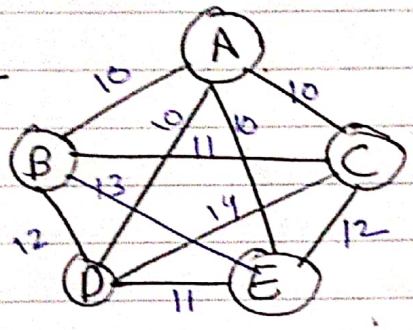
internal node degree = 2 | in directed {outdegree = 0 → leaf node}
in undirected {degree = 1 → leaf node}

(∴ x should be there, inorder to remove cycle delete the maximum, i.e. 6 and assign x (& del)

{ Not Distinct question — x = 6 or 5 (max) (max)
Distinct given — x = 5 / 4 / 3 / 2 / 1 i.e. → They are already in graph. }

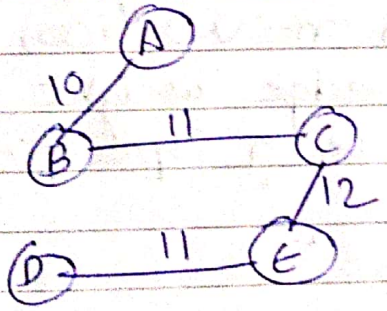
(if min ask − 1, 2, 3, 4, 5, <6 } ⇒ Not distinct
{ " " " − 1 } ⇒ distinct

Ex-5.)

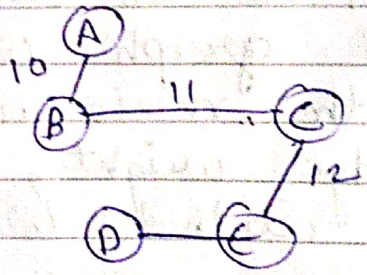


What is the cost of the MST for this graph in such way that MST 'A' will be the leaf node.

Prim's



Kruskal



Ex-6.) Let G be an undirected weighted connected graph with distinct edge weights, e_{max} be the max^m - edge weight && e_{min} be its minimum edge weight.

T/F:-

a.) G − contain − UMST ✓
b.) MST of G may (must) contain e_{min} ✗
c.) " " " " " e_{max} ✓
d.) if e_{max} is in MST then its removal from G must disconnect G.

ex 7.) Let G be an undirected weighted connected graph with n-vertices, W be the minimum edge weight among all edge weight, e be a specific edge with w.

T/F.

✗ a.) G contain unique MST

✗ b.) Every MST of G must contain e

✓ c.) Every MST of G must contain atleast one edge with weight w.

✓ d.) If e is not in MST then in that cycle all edges contain same weight w.

(which is not there)

(a. Ans.)

(b. Ans.)

(c. Ans.)

(d. Ans.)

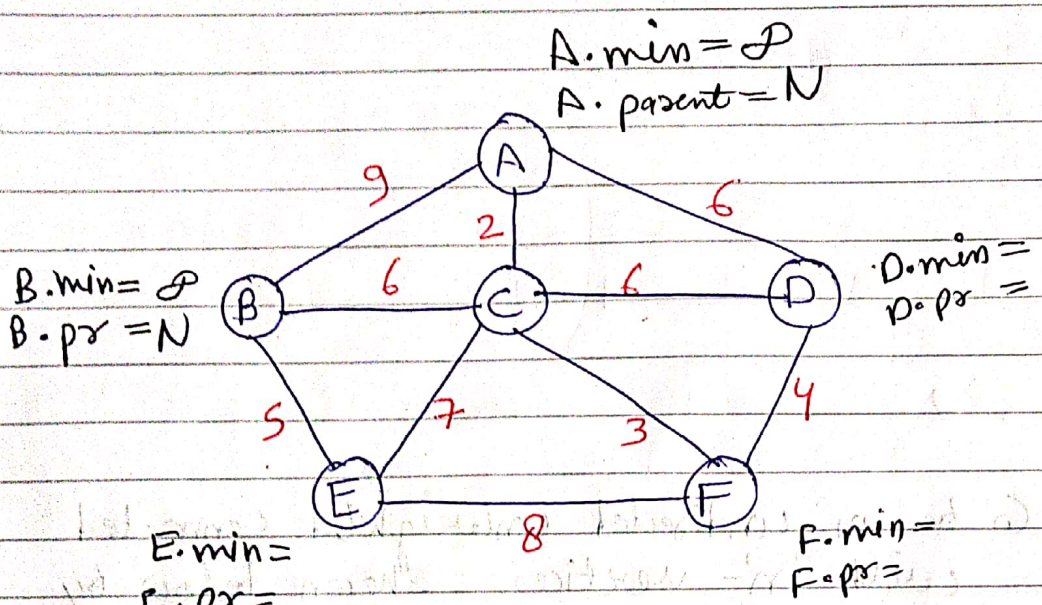
(ex-8.) Let G be an undirected unweighted connected graph with 'n'- vertices shown below by an n×n adjancency matrix in which

(i.) All diagonal element 0's
(ii.) All Non- diagonal element 1's

T/F

a.) G contain Unique MST
b.) G dont have MST
c.) G contain multiple MST with diffrent cost.
d.) G contain multipk MST with same cost.

TIME COMPLEXITY OF PRISM'S ALGORITHM:



Present in Min heap

	A	B	C	D	E	F	
	∞ N	∞ N	∞ N	∞ N	∞ N	∞ N	⇒ O(V)
logV A		9 A	2 A	6 A	∞ N	∞ N	⇒ 3 + 3logV
logV C		6 C		6 A	7 C	3 C	⇒ 5 + 4logV
logV F		6 C		4 F	7 C		⇒ 3 + 2logV
logV D		6 C			7 C		⇒ 3 + 0logV
logV B					5 B		⇒ 3 + 1logV
VlogV E							⇒ 3 + 0logV

2E + ElogV

↓

Undirected

Time Complexity $= V\log V + 2E + E\log V + V$
$= (V+E)\log V$
$= O[(V+E)\log V]$

using Adjancency Matrix, minheap.

$TC = V\log V + V^2 + E\log V + V$
$= O(V^2 + E\log V)$

using Adjancency list, sorted array

$T.C = V + 2E + EV + V$
$= O(EV)$.

Using adj matrix, sorted Double Linked list

$T.C = V + V^2 + EV + V$
$= O(V^2 + EV)$

Using adj list and normal unsorted array

$T.C = V^2 + 2E + E + V$
$= O(V^2)$.

HUFFMAN CODING :-

{Data compression tech}
{Non uniform coding technique}

ex-1

m = 100

a = 6	b = 30
c = 45	d = 3
e = 1	f = 15

ASCII

⇓

a - 6×8
b - 30×8
c - 45×8
f - 15×8

100 char = 800 bit

char = 8 bit/char.

By default 2way

Merge those who are minimum

OPTIMAL MERGE PATTERN :-

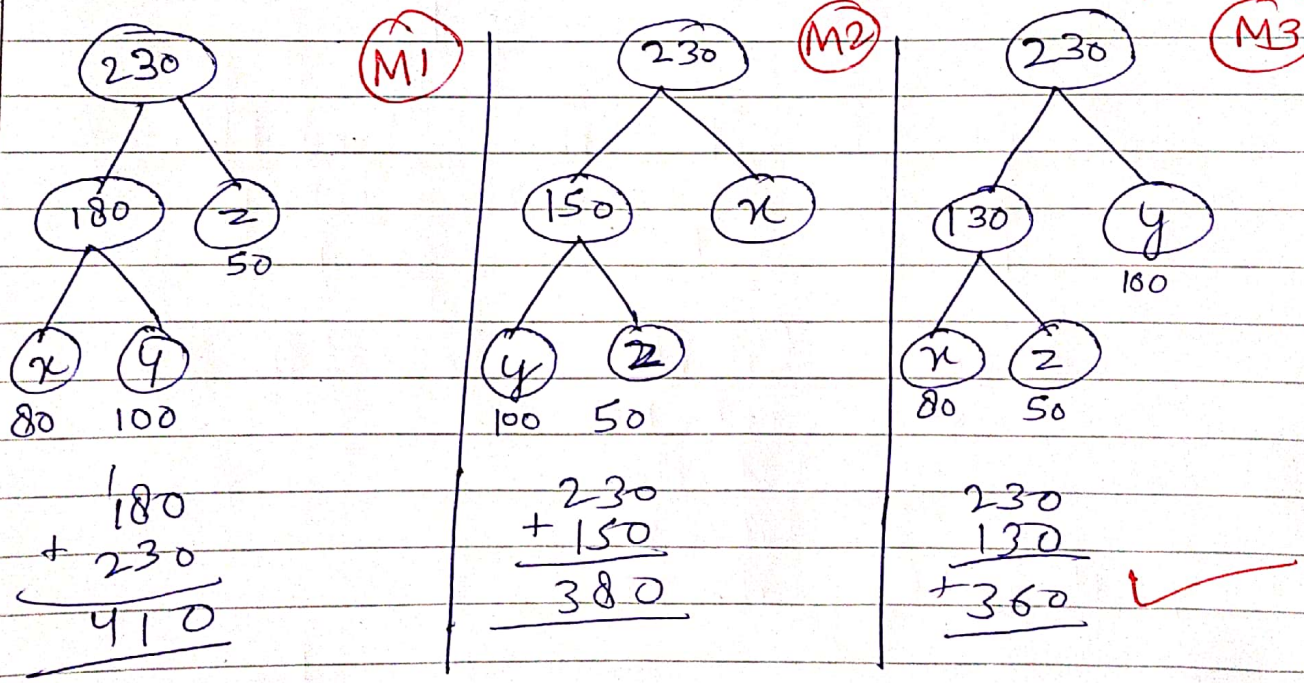
Ex-1.) n = 3 (3 files)

⇓

x — 80 (80-records)
y — 100
z — 50

Minimum moves = ?

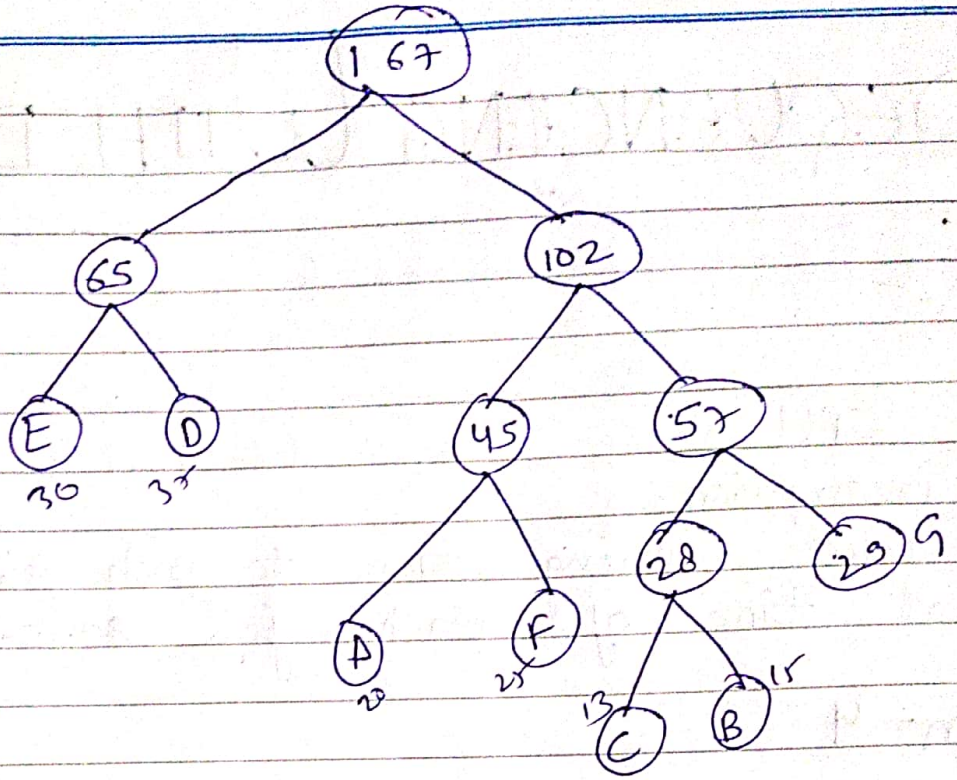
↑ Merge 2 minimum for min moves.



ex-2 n = 7

⇓

A = 20 B = 15
C = 13 D = 35
E = 30 F = 25
G = 29



Minimum record moves = Sum of internal nodes.

$= 28 + 57 + 102 + 45 + 65 + 167$

$= \underline{464}$.

JOB SEQUENCING WITH DEAD LINES:-

1. Single CPU.
2. No Preemption.
3. One unit running time to each job.
4. Arrival time of each job same.

ex1.) $n=4$

Jobs	J_1	J_2	J_3	J_4
profit	175	125	150	200
Deadline	2	1	2	1

$(J_2, J_1) \Rightarrow 300$

~~(J_1, J_2)~~

$(J_4\ J_3) = 350$

$(J_2\ J_3) = 275$

$(J_1\ J_3) = 325$

$(J_3\ J_1) = 325$

optimal soln → $(J_4, J_1) = 375$ ← Max profit

feasible		
J_1		
J_2	(1)	
J_3		
J_4		

$6+4+1 = 11$ feasible

Shortcut:-

① Take the Max Deadline at 2 place and then less Deadline at 1 place

Max profit →

J_4	J_1
1	2

* Order of optimal soln J_4, J_1
* optim soln profit - 375
* which Job you left J_2, J_3
* what profit you left = 275

(ex-2.) n=7

Penalty

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7
profit	300	250	150	200	175	275	270
Deadline	5	3	5	3	2	1	5

O(n) O(n) O(n) O(n) O(n)

J_6	J_4	J_2	J_7	J_1
1	2	3	4	5

[Those whose didn't done called penalty]

optimal soln = 1295

seqn = J_6, J_4, J_2, J_7, J_1

penalty - 325

ex-3

n=9

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
profit	75	50	25	70	55	30	60	45	68
	5	3	7	3	5	4	2	5	3

J_7	J_1	J_4	J_5	J_1	////	J_3
1	2	3	4	5	6	7

Shortcut

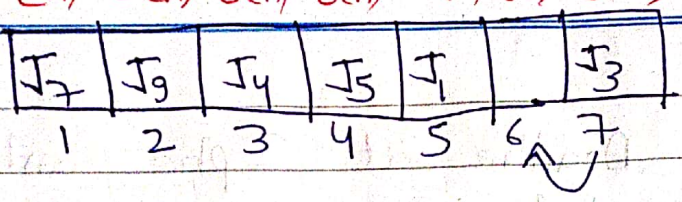
① Sort in Decreasing order of profit ⇒ (n log n) Merge sort

Jobs : J_1 J_4 J_9 J_7 J_5 J_2 J_8 J_6 J_3

Profit: 75 70 68 60 55 50 45 30 25

Deadline: 5 3 3 2 5 3 5 4 7

O(n) O(n) O(n) O(n) O(n) O(1) O(n)

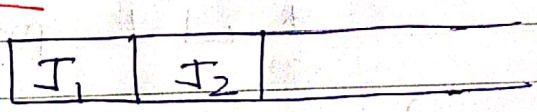


$\Rightarrow O(n^2)$

$n^2 + n\log n \Rightarrow O(n^2)$ [Worst Case]

$O(n\log n)$ [Best Case]

* if adjacent job have <u>same Job dead line</u> then you can <u>swap</u> them.



Dead line → 3 3

swap possible