19-06-2023.

UNIT - 6 - GRAPHS

* Graph :-

→ 'Graph' is a Non-linear Data structure.

→ The difference b/w 'Tree' & 'Graph' is tree doesnot contain cycles but 'graph' may contains 'cycles'.

→ Graph contains 'vertices' & 'Edges'; G(V, E).

→ Every 'Tree' is a 'Graph' but every 'graph' need not be a 'tree'.

→ Graph is classified as
① 'Directed graph'
② 'Undirected graph'.

* Types of Graphs :-

① Regular Graph :-

→ The degree of every vertex is same in the graph is called 'Regular graph'.

② cyclic Graph :-

→ The graph which contains cycle is called 'cyclic graph'.

③ complete Graph :-

→ If every vertex of graph connected to other vertex then the graph is called 'complete graph'

④ Bipartite Graph :-

→ set-1 vertices connected to the vertices of set-2 is called 'Bipartite Graph.'

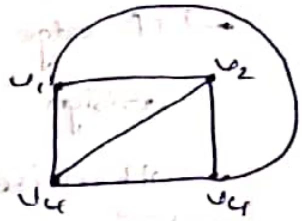
→ If all the vertices are connected in both sets then it is called 'complete Bipartite Graph'.

* **Note:-**

→ If there are 'n' vertices then the complete graph contains '$\frac{n \times (n-1)}{2}$' edges.

Ex:- 4 vertices

$$\text{Edges} = \frac{4 \times 3}{2} = 6.$$



* **Degree:-**

→ No. of edges connected to a vertex is called 'Degree of vertex'.

* **simple graph:-**

→ If they are no parallel edges in a graph then it is called 'simple graph.'

* **Multigraph:-**

→ Having parallel edges in a graph.

* **In-Degree & out-Degree:-**

→ The Incoming edges towards a vertex in a Directed graph is called 'In-Degree'.

→ The No. of outgoing edges from a vertex in an Undi Directed graph is called out-degree.

* **connected graph:-**

→ If a graph contains path which is passing through all vertices then it is a connected graph

* **Isolated Node /vertex:-**

→ A vertex having 'degree-0' then it is called Isolated vertex.

* Graph Representations :-

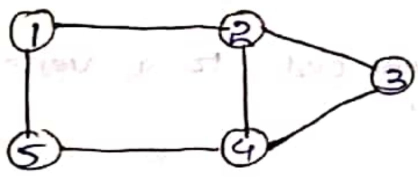
① Adjacency Matrix :-

→ If edge is present b/w two vertices then we will assign '1' to respective position in a matrix

→ otherwise we will assign '0'.

→ Time complexity is '$O(V^2)$'.

Ex :-



Adjacency Matrix :-

$$\begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 0 & 0 & 1 & 0 \end{array}$$

DrawBack :-

→ If the no. of vertices are more ~~then~~ and no. of edges will be less. ~~Then~~ Adjacency matrix is not suitable.

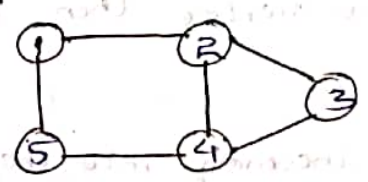
Ex :- 100 × 100 matrix, needs 10,000 memory locations to store the values.

→ If given matrix is sparce [More no. of '0's] then don't consider Adjacency matrix; it is difficult.

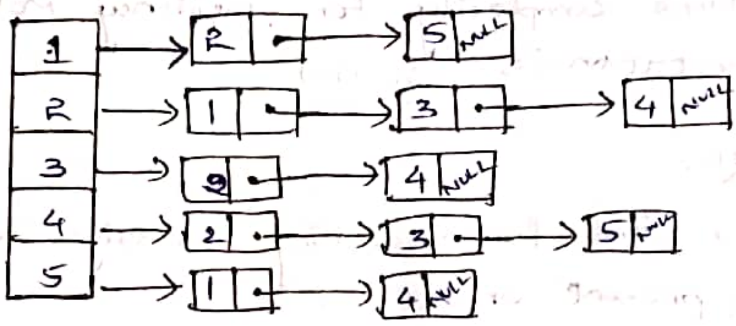
→ If given matrix is Dense [having more edges → '1'] then we can choose Adjacency matrix representation

(2) Adjacency List Representation :-

Ex:-



Adjacency List :-



→ The Time complexity is 'O(V+E)'.

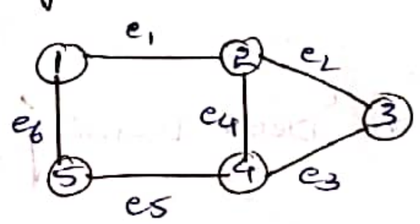
Drawback :-

→ The time complexity in worst case for searching an edge is present or not is 'Θ(n)'.

Advantage :-

(3) Incidency Matrix Representation :-

Ex:-



Incidency matrix :-

$$
\begin{array}{c}
 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5
\end{array}
\begin{array}{c}
\begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1
\end{bmatrix}
\end{array}
$$

→ If the graph is Directed graph, if an edge is outgoing from a vertex then the value is '1'.

→ If the ~~vertex~~ edge is Incoming into the vertex then the value is '−1'.

→ The Time complexity for Incidency Matrix representation is 'O(VE)'.

Draw Back:-

→ Takes time for searching an edge whether it is present or not.

* Graph Traversals:-

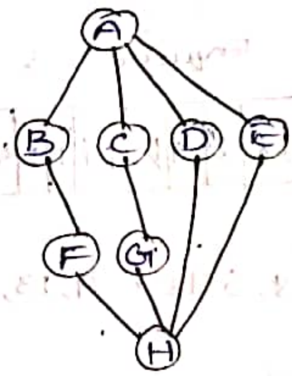
→ There are two methods to traverse the Graph.

① "Breadth First Search [BFS]" → Queue concept.

② "Depth First search [DFS]" → By stacks concept.

① Breadth First Search [B.F.S] :- [Level-order Traversal]

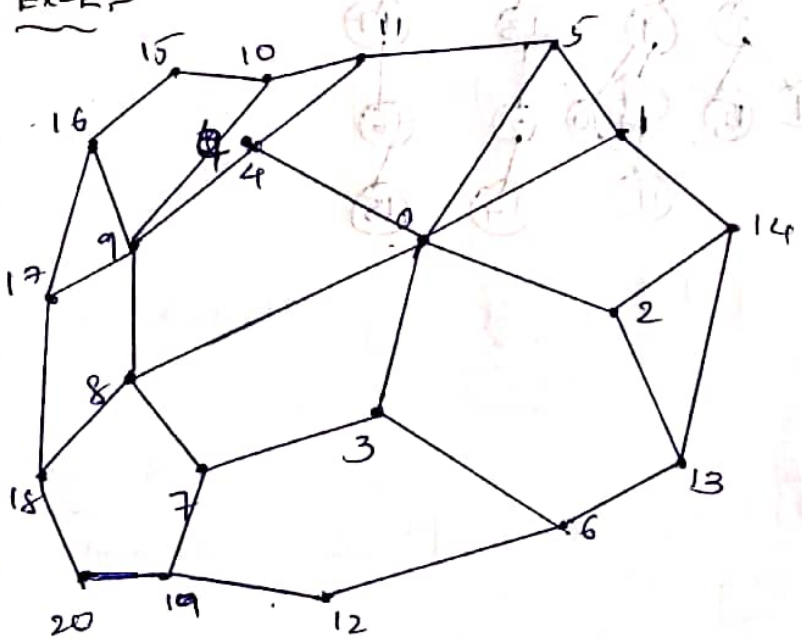
Ex-1



~~A~~	~~B~~	~~C~~	~~D~~	~~E~~	~~F~~	~~G~~	~~H~~

O/P :- A, B, C, D, E, F, G, H

Ex-2:-



Let,

starting vertex is 19. — for ~~BST~~ B.F.S

starting vertex is 0. — for D.F.S

Step-1 :- Enqueue the starting vertex

19	

Front
Rear

o/p :- 19

Step-2 :- Dequeue 7 and Enqueue its adjacent vertices and so on

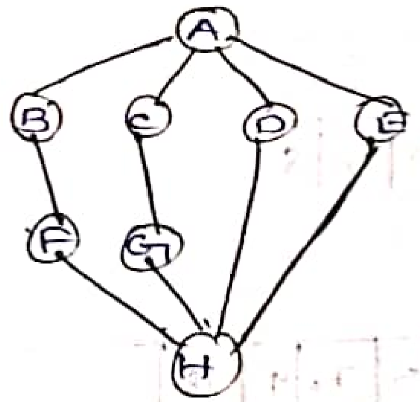
~~19~~	~~7~~	~~12~~	~~20~~	~~3~~	~~8~~	~~6~~	~~18~~	~~0~~	~~9~~	~~13~~	~~17~~	1	2	4	5	10	16	14	15

o/p :- 19, 7, 12, 20, 3, 8, 6, 18, 0, 9, 13, 17, 1, 2, 4, 5, 10, 16, 14, 15.

Scanned with CamScanner

(2) Depth First Traversal :-

Ex :-



step-1 :-

A

step-2 :-

o/p: A

E
D
C
B

step-3 :-

o/p :- A, E

D
C
B

step-4 :-

o/p :- A, E,

H
D
C
B

step-5 :-

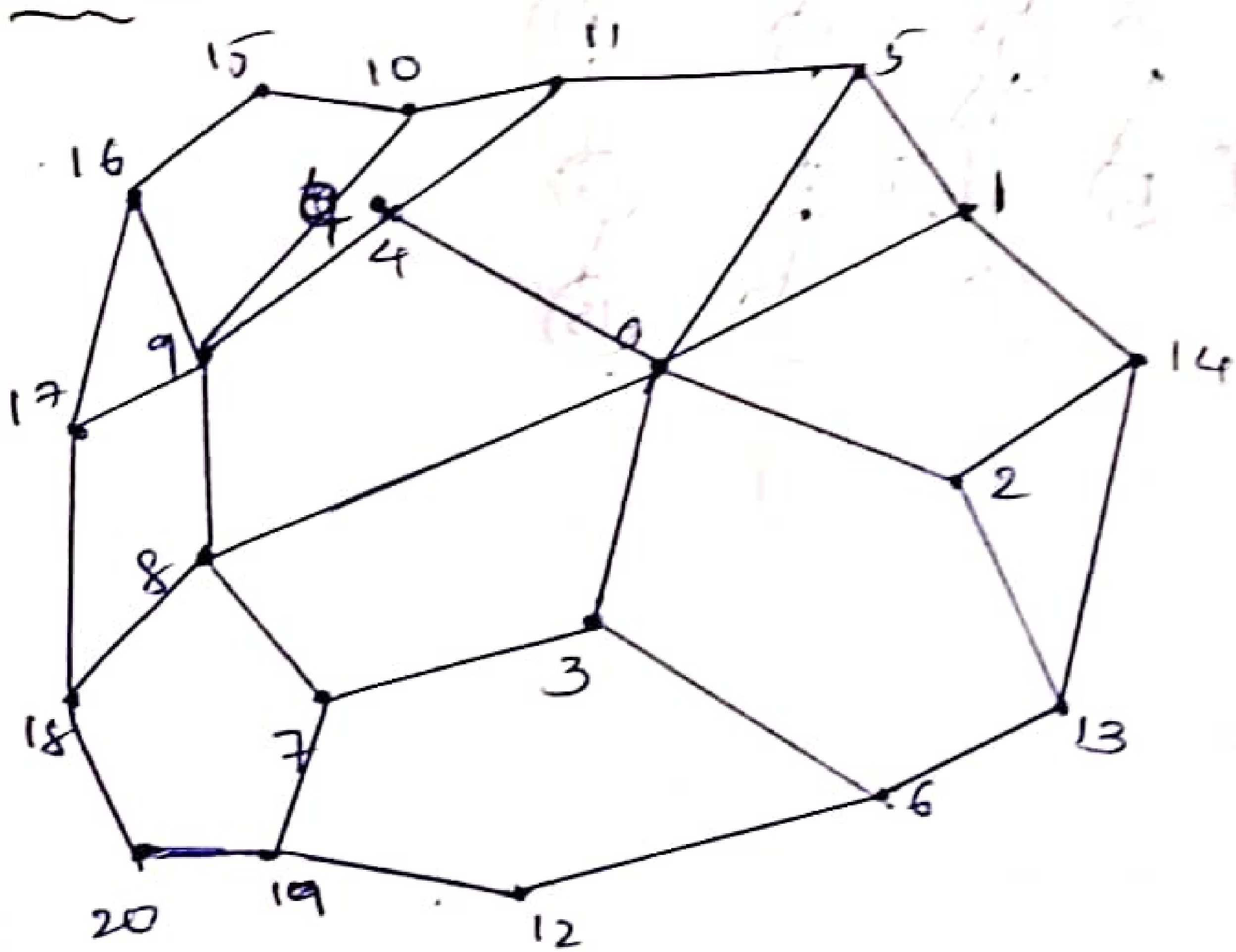
o/p :- A, E, H

G
F
D
C
B

step-6 :-

o/p :- A, E, H, G, F, D, C, B

Ex-2:-



Let,

starting vertex is 19. — for ~~BST~~ B.F.S

starting vertex is 0. — for D.F.S

Ex-2 :-

step-1 :-

0	

step-2 :-

1	2	3	5	8	

o/p :- 0

step-3 :-

1	2	3	5	7	9	18	

o/p :- 0, 8

step-4 :-

1	2	3	5	7	9	17	20	

o/p :- 0, 8, 18

step-5 :-

1	2	3	5	7	9	17	19	

o/p :- 0, 8, 18, 20

step-6 :-
o/p

1	2	3	5	7	9	17	12	

o/p :- 0, 8, 18, 20, 19

step-7 :-

1	2	3	5	7	9	17	16	

o/p :- 0, 8, 18, 20, 19, 12

step-8 :-

1	2	3	5	7	9	17	13	

o/p :- 0, 8, 18, 20, 19, 12, 6

step-9 :-

1	2	3	5	7	9	17	14	

o/p :- 0, 8, 18, 20, 19, 12, 6, 13.

step-10 :-

1	2	3	5	7	9	17	

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14

step-11 :-

1	2	3	5	7	9	16

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17

step-12 :-

1	2	3	5	7	9	15

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17, 16

step-13 :-

1	2	3	5	7	9	10

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17, 16, 15

step-14 :-

1	2	3	5	7	9	11

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17, 16, 15, 10

step-15 :-

1	2	3	5	7	9

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17, 16, 15, 10, 11

step-16 :-

1	2	3	5	7

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17, 16, 15, 10, 11, 9

step-17 :-

1	2	3	5

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17, 16, 15, 10, 11, 9, 7

step-18 :-

1	2	3

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17, 16, 15, 10, 11, 9, 7, 5

step-19 :-

1	2

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17, 16, 15, 10, 11, 9, 7, 5, 3

step-20 :-

o/p :- 0, 8, 18, 20, 19, 12, 6, 13, 14, 17, 16, 15, 10, 11, 9, 7, 5, 3, 2, 1

* Minimum Spanning Trees :- [M.S.T].

→ Let us consider a "weighted graph", we have to create a Tree from that graph such that it should cover all the 'n' no of vertices with minimum coust.

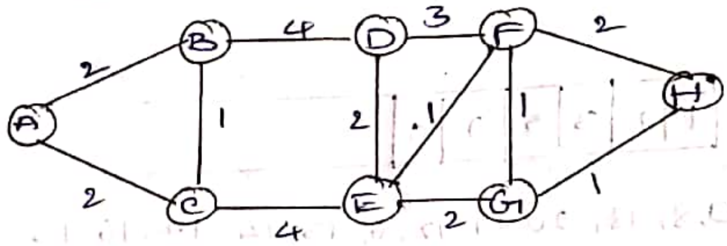
→ The tree obtained have "(n-1) edges".

→ There are two types of methods to create a Minimum spanning Tree from weighted graph

1. "Prim's Algorithm."
2. "Kruskal's Algorithm."

→ We will use M.S.T for "Multicasts" & "Broad casts" etc.

Ex :-

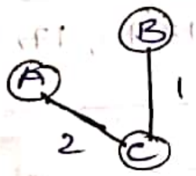


1. Prim's Algorithm :-

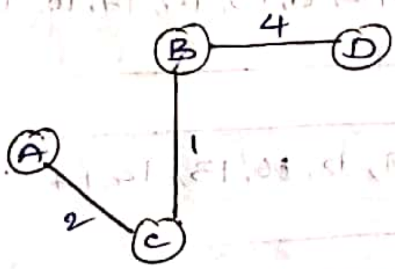
step-1 :- Take any arbitary vertex.

step-2 :-

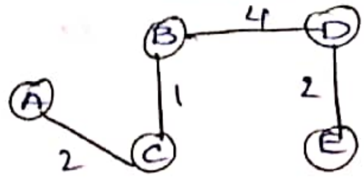
step-3 :-



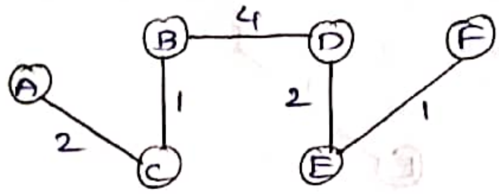
step-4 :-



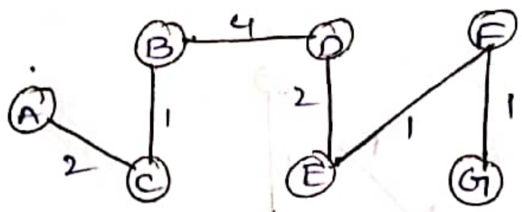
Step-5 :-



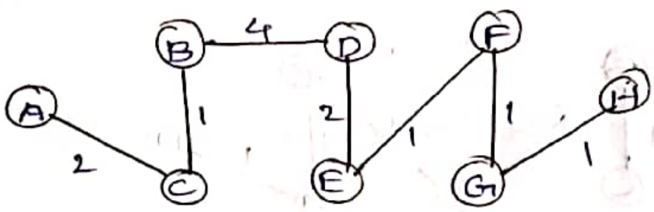
Step-6 :-



Step-7 :-



Step-8 :-



We covered all the vertices without cycles and with minimum costs.

$\therefore$ The cost of Minimum spanning Tree $= 2+1+4+2+1+1+1$

$= 12$

Edges $= n-1 = 7$.

② Kruskal's Algorithm :-

Costs :-

$(B,C) = 1$

$(E,F) = 1$

$(F,G) = 1$

$(G,H) = 1$

$(A,B) = 2$

$(A,C) = 2$

$(D,E) = 2$

$(E,G) = 2$

$(F,H) = 2$

$(D,F) = 3$

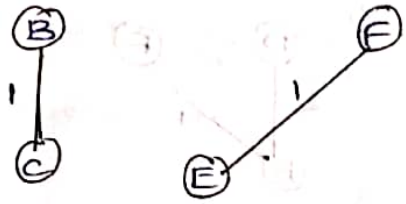
$(B,D) = 4$

$(C,E) = 4$

Scanned with CamScanner

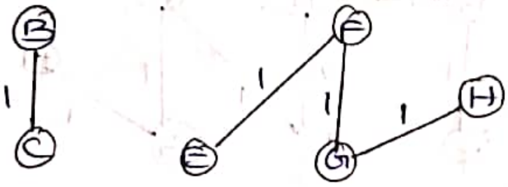
step-1 :-

step-2 :-

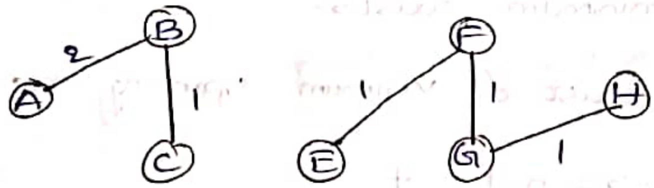


step-3 :-

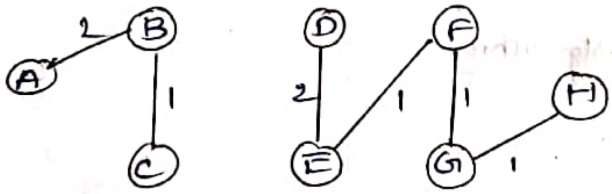
step-4 :-



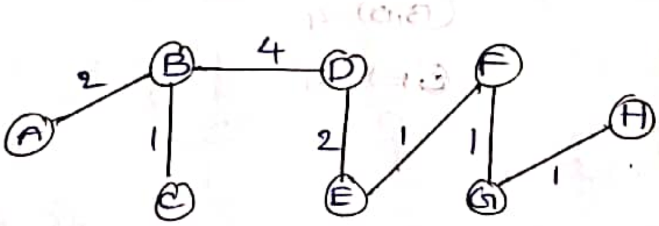
step-5 :-



step-6 :-



step-7 :-



Minimum coust = 12.

→ We created a M.S.T and discarded all the edges which makes a cycle.

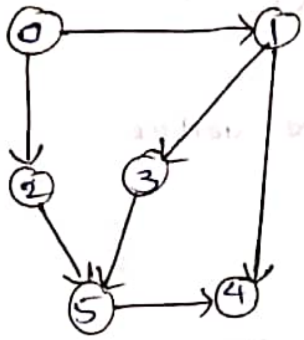
* Topological sort:-

→ It is used to sort the elements by using graphs

→ It is applied only on Directed Asyclic Graphs [DAG]

→ 'DAG':- A graph contains directions but not contains cycle is called 'Directed Asyclic Graphs'.

Ex:-

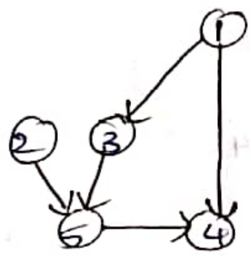


step-1:-

→ consider a vertex which contains In-degree as '0'.

o/p:- 0

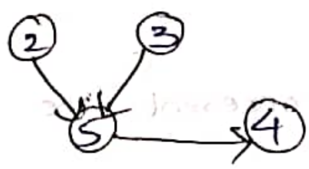
step-2:- Remove the edges connected with vertex-'0'.



step-3:- Again select the '0'- indegree vertex.

o/p:- 0, 1

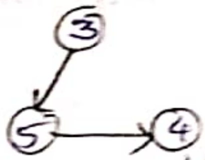
step-4:- Remove '1' and its corresponding edges.



step-5 :- Select '0-Indegree' vertex

o/p :- 0, 1, 2

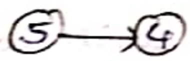
step-6 :- Remove '2' and its corresponding edges.



step-7 :- Select '0-degree' vertex

o/p :- 0, 1, 2, 3

step-8 :- Remove '3' and its corresponding edges.



step-9 :- Select '0-degree' vertex

o/p :- 0, 1, 2, 3, 5

step-10 :- 

step-11 :- 0, 1, 2, 3, 5, 4.

* uses :-

→ It is used for project completion.

* Trie- Data structure :-

→ It is basically 'Non linear Data structure'.

→ It is also known as 'Digital Tree' (or) 'Prefix Tree'

→ It is used to represent the strings in the form of Tree.

→ It contains tree fields 'Data field', 'pointer field' and 'Flag field'.

Ex:- a b c

a b c d e

a b c g h

a b c g l m

l m n

l m n o

l m n x y