Sub :

Sat Sun Mon Tue Wed Thu Fri

Time : Date : / /

Graph

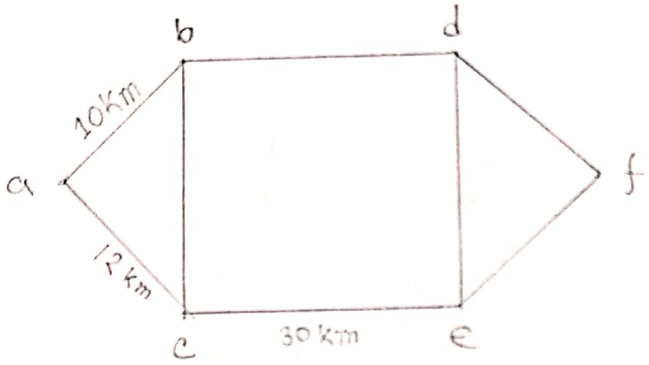
$G_1 = (V, E)$

objects — Relation

V = set of vertices

E = set of edges

1. Friends
2. Roads



$V = \{a, b, c, d, e, f\}$

$F = ((a,b), (b,d), (a,c), (e,f), (c,e), (d,e), (d,f), (b,c))$

1. Order of graph:

No. of vertices of graph $|V| = 6$

2. Size of graph:

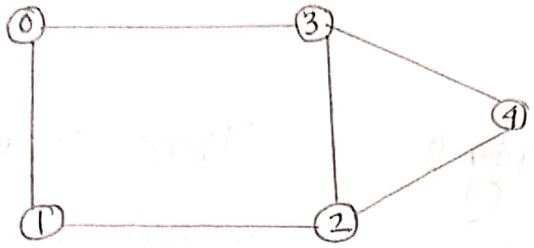
edges: $|E| = 8$

1. Order of graph
2. Size of graph

Sub : Lecture - 2 : Graph Representation

Sat ☐ Sun ☐ Mon ☐ Tue ☐ Wed ☐ Thu ☐ Fri ☐

Time : Date : 11 / 6 / 20



Matrix:

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

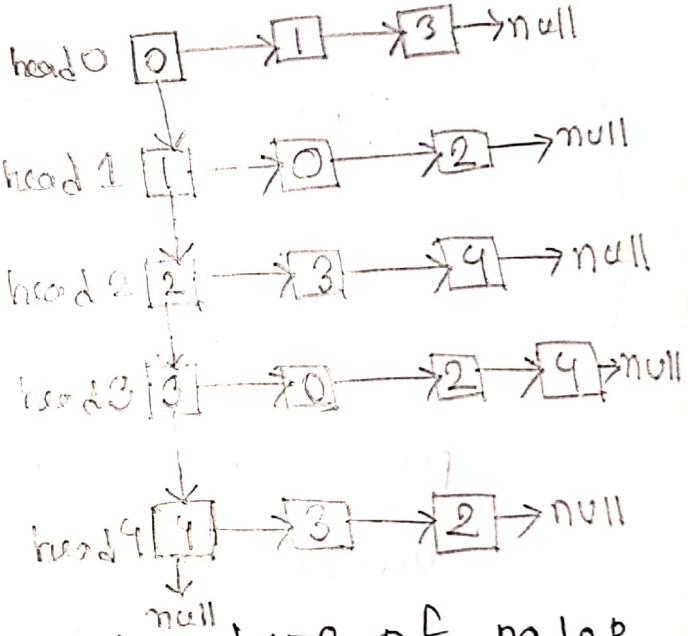
Sparce Matrix

Size = n x n

This Representation waste space.

So, Adjacency List comes.

Adjacency List



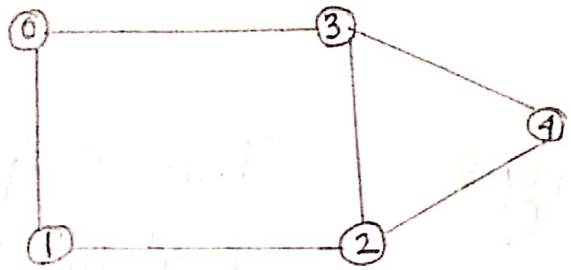
Structure of node:

{ int data,
vertex
edge
}

Sub: Lecture - 2 : Graph Representation

Sat Sun Mon Tue Wed Thu Fri

Time : Date : 11 / 6 / 20



Matrix:

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

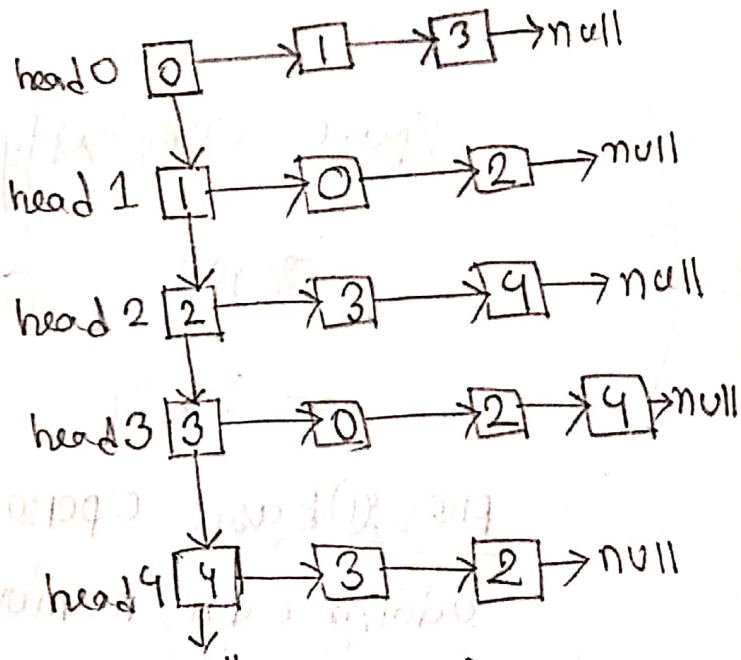
Space Matrix

Size = n x n

This Representation waste space.

So, Adjacency List comes.

Adjacency List



Structure of node:

```
{ int data,
  vertex
  edge
}
```

Sub :

Sat Sun Mon Tue Wed Thu Fri

Time : Date : / /

Matrix

Time complexity:

$O(1)$

Space complexity:

$O(n^2)$

$= n \times n$

pros: (i) Basic operation: adding edge, removing and checking whether there is an edge from vertex i to vertex j are extremely efficient. constant time

(ii) If the graph is dense and num of edges is close the |V|²

Adjacency List

Time Complexity:

worst case: $O(|V|+|E|)$

Space complexity:

$O(|V|+|E|)$

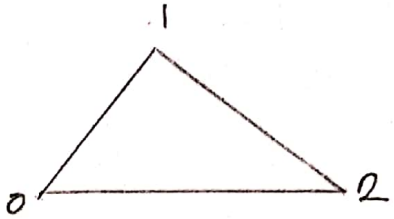
Sub :

Sat Sun Mon Tue Wed Thu Fri

Time : Date : / /

#3. Code for Matrix Representation of graph

1: Adjacency Matrix :



	0	1	2
0	0	1	1
1	1	0	1
2	1	1	0

```
public class Graph {
    private boolean adjMatrix[][];
    private int numVertices;
    public Graph(int numVertices) {
        this.numVertices = numVertices;
        adjMatrix = new Boolean[
                numVertices][numVertices];
    }
    public void addEdge(
        adjMatrix[i][j] = true
        adjMatrix[j][i] = true;
    }
```

adj

pub

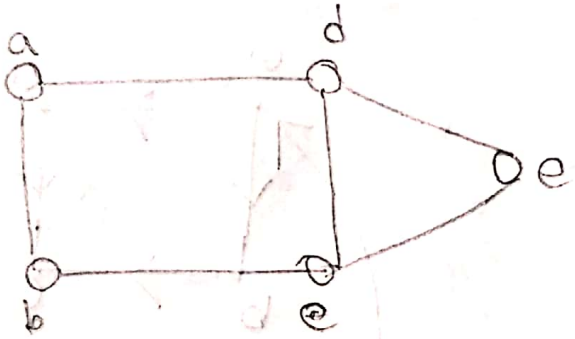
Sat Sun Mon Tue Wed Thu Fri

Sub : .....

Time : Date : / /

#5 Types of graphs

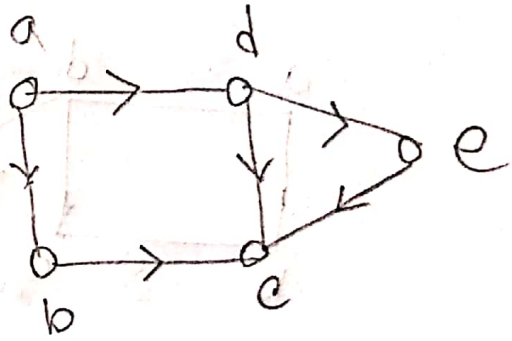
1. Undirected graph



if (x,y) then (y,x)

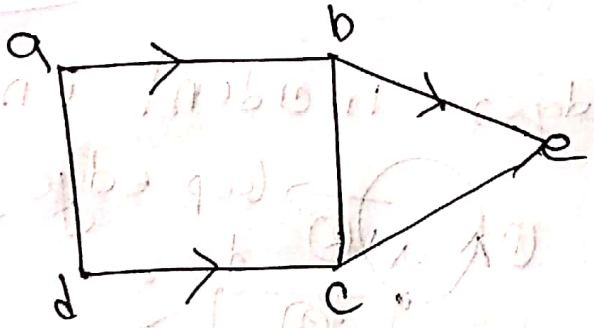
(Fb)

2. Directed graphs



if (x,y), then doesn't mean (y,x)

(Follower)



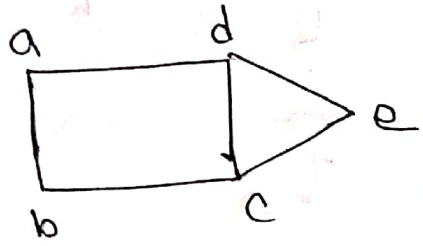
Sat Sun Mon Tue Wed Thu Fri

Sub .....

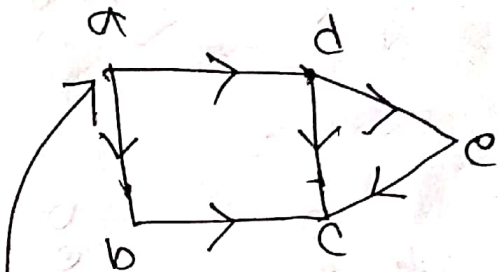
Time : Date : / /

Degree of a vertex :

Undirected



Directed



Indegree

Outdegree

Undirected

Number of edges incident on a vertex

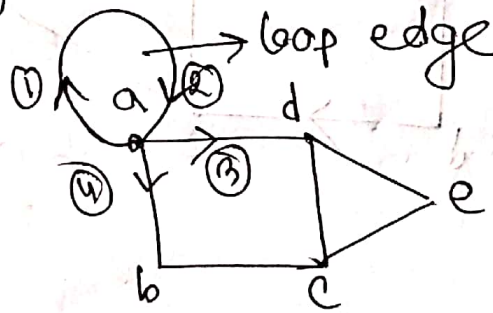
a → 2

b → 2

d → 3

c → 3

e → 2



degree is 1

total degree is = 4

Directed :

a indegree = 0

b outdegree = 1, indegree = 1

c = indegree = 3, outdeg = 0

d(I) = 1, d(O) = 2

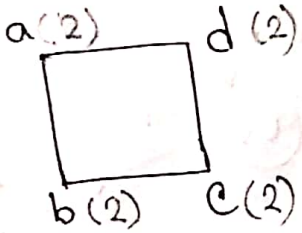
Sat Sun Mon Tue Wed Thu Fri

Sub :

Time : Date : / /

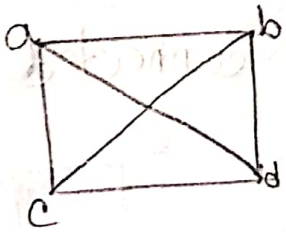
#classes of graph:

① Regular graph:

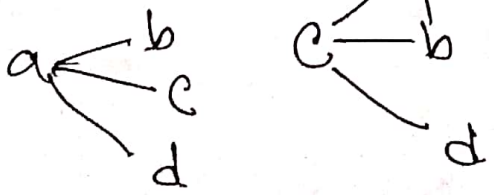


k-regular graph
(every vertex is same)
(2-regular graph)

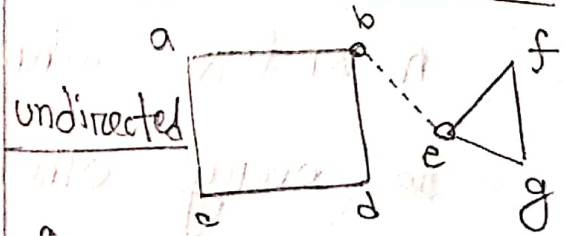
② Complete graph:

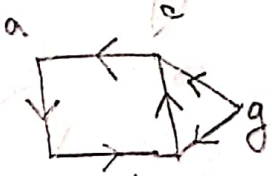


each vertex connected with each other or/ all other



③ Connected Graph:





Directed:

(#connected with a line to go each vertex)

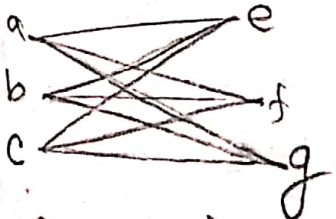
we can go all the each vertex, any vertex.

strongly connected:

4. planner graph:

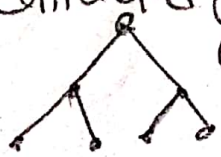
shouldnot be intersect at any edges.

⑤ Bipartite graph:



Devide two sets:

⑥ TREE: connected graph with NO cycle



(NO cycle)

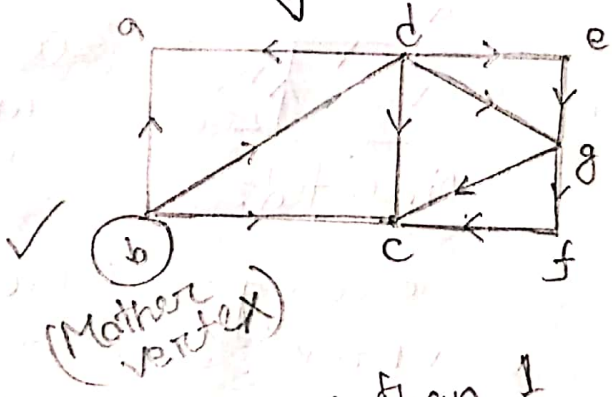
Sat Sun Mon Tue Wed Thu Fri

Sub : .....

Time : Date : / /

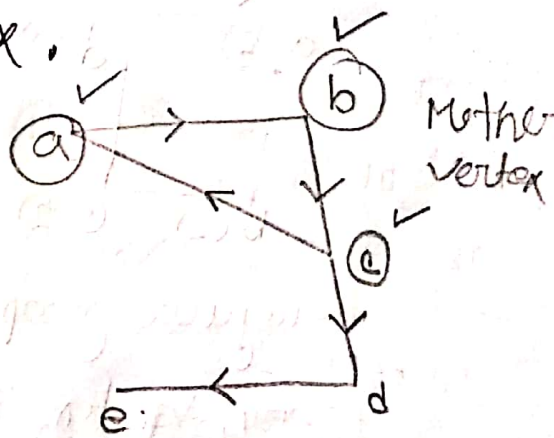
Mother vertex :

A vertex which has a path to reach to every other vertex.

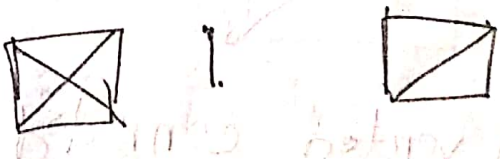


(Mother vertex)

(can more than 1 mother vertex)



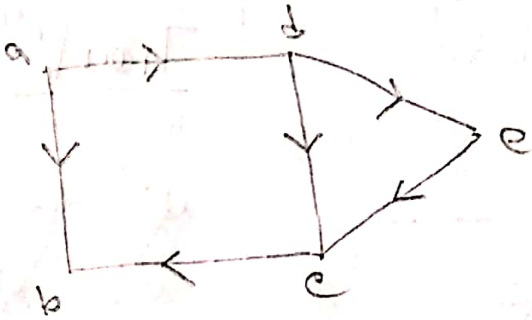
Case :
1. Directed graph
2. undirected connected graph
3. undirected disconnected ,,



Sat ☐ Sun ☐ Mon ☐ Tue ☐ Wed ☐ Thu ☐ Fri ☐

Sub : .... Time : Date : / /

Transative clousure of a graph:



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

Reachability Matrix

DFS algo

Floyd-warshall algo

{ (a,a), (a,b), (a,c), (a,d), (a,e)
(b,b) (b,d), (c,c),
(d,b) (d,c), (d,d), (d,e), (e,b), (e,c), (e,e) }

path from (i,j) কতগুলো path আছে।
প্রত্যেক গ্রাফ কতখানি connected আছে।

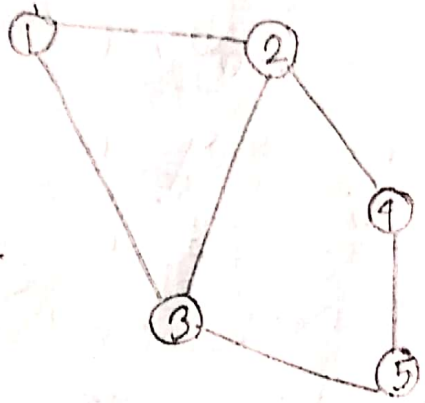
Sub: #9. Lecture

Sat Sun Mon Tue Wed Thu Fri

Time: Date: / /

walk, Trail, Circuit, path, circle

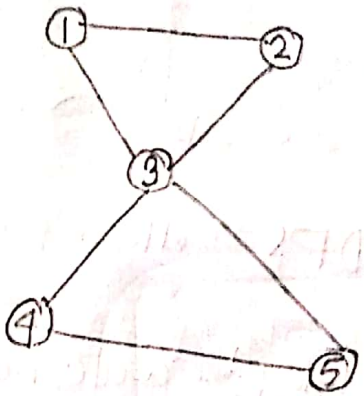
walk:



Rep- vertex
eated edge

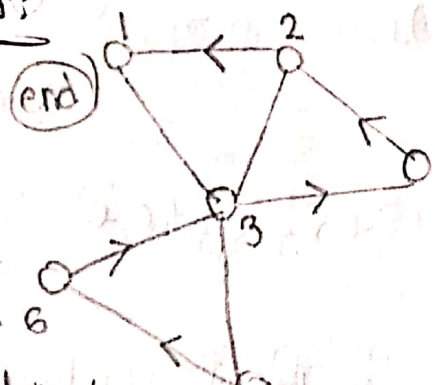
1 → 2 → 4 → 5 → 3 → 2 → 1

Trail:



Repeated vertex ✓
edge ↓
NOT Repeated

path:



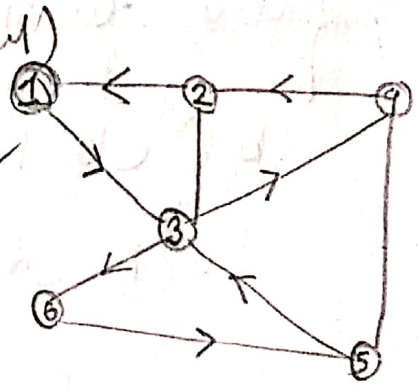
vertex: X
edges: X

No Repeated:

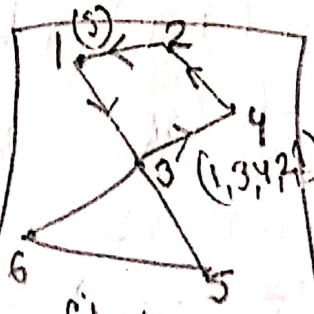
5 (Start) — 6 — 3 — 4 — 2 — 1

✓ (Many path) Containing diff edge and vertices

Circuit: Trail



Vertex ✓
edge



Cycle:

Same as path
But closed (S = end)
✓ vertex Repeated

Starting and ending vertex should be same

1 → 3 → 6 → 5 → 3 → 4 → 2 → 1

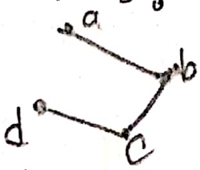
edge → Not

Sub : .... Sat ☐ Sun ☐ Mon ☐ Tue ☐ Wed ☐ Thu ☐ Fri ☐
Time : Date : / /

Tree and Its property:

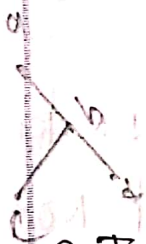
A is an undirected graph. that satisfies:

1. Tree is connected and Acyclic



2. T is acyelic and a simple cycle is formed if any edge is added to T.

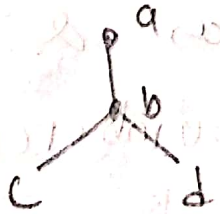
যদি আমরা কোনএকটি edge connected করি তাহলে cycle হয়ে যাবে



(abc) (abd)

3. Two vertices in T can be connected by simple unique path..

(4) T connected and has n-1 edge. (Most important property)

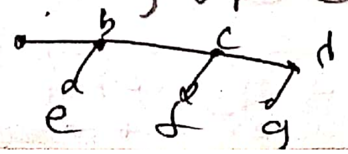


(a,b), (b,c), (b,d)

(4 টা vertices 3 টা edges)

যদি (n-1) এর বেশি একটি edge হয় তাহলে cycle হবে diameter

(5) Internal vertex & external ,, . ↓ (leaf) →



Scanned with CamScanner

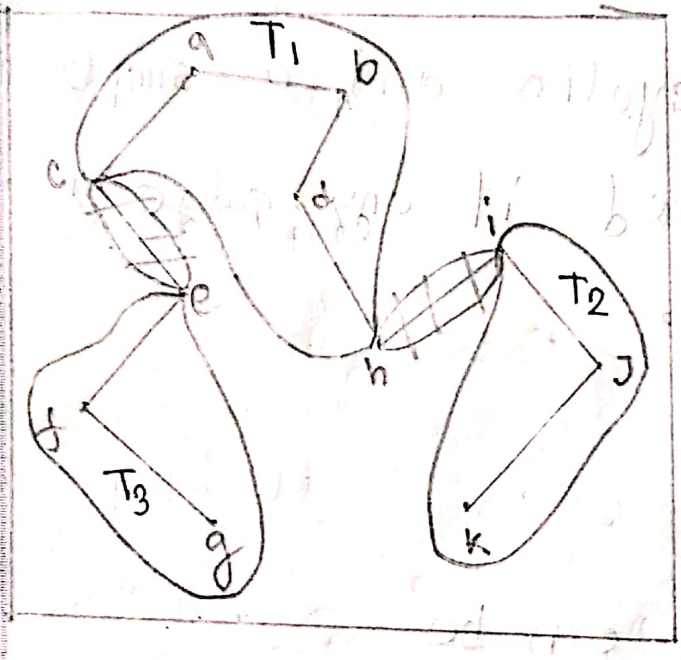
Sub :

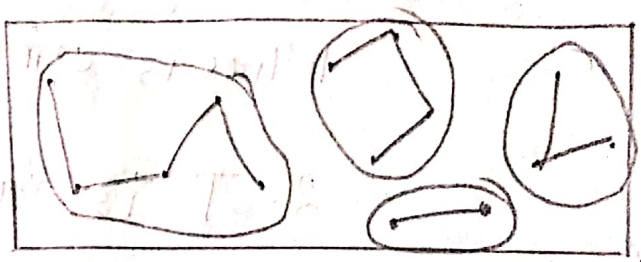
Sat Sun Mon Tue Wed Thu Fri

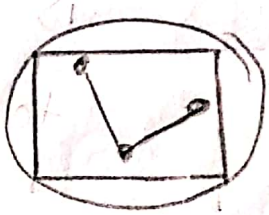
Time : Date : / /

Forest in graph theory :

A forest is the disjoint union of trees.







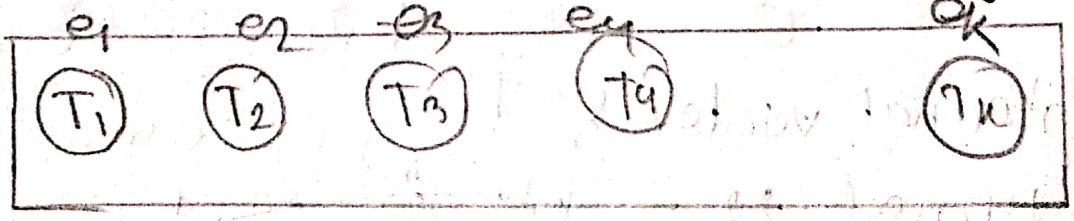
Every tree is forest

Every forest Not Tree

$\{a,b,d,h,c\}$ $\{i,j,k\}$, $\{e,f,g\}$

T_1 T_2 T_3

If there are 'n' vertices & 'k' component in a forest, then number of edges —.



$n_1 + n_2 + n_3 + n_4 + n_k = n$

$e_1 + e_2 + e_3 + e_4 + e_k = e$ or E.

Sat Sun Mon Tue Wed Thu Fri

Sub : .....

Time : Date : / /

$(n_1-1)+(n_2-1)+(n_3-1)+(n_4-1)+\ldots(n_k-1) = E_F$

$(n_1+n_2+n_3+n_4+\ldots n_k-(1+1+1+1\ldots+k\text{ time}))$

$= E_F$

$(n-k) = E_F$

Total Number of edges in forest:

$(n-k) = E_F$

Total No. of Trees:

$k = n - E_F$

$= T_V - T_E$

$T_E = 6, \; T_V = 9,$

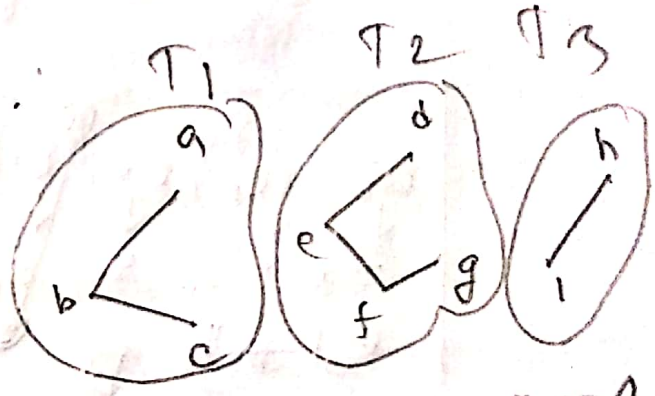
How Many tree

$k = 9 - 6$

$= 3$

n = no of vertices

k = edge vertex



$n_1 = 3 \quad n_2 = 4 \quad n_3 = 2$

$k = 3$

$n = n_1 + n_2 + n_3$

$= 3+4+2$

$= 9$

$E_F = (n-k)$

$= 9-3 = 6$

Sat Sun Mon Tue Wed Thu Fri

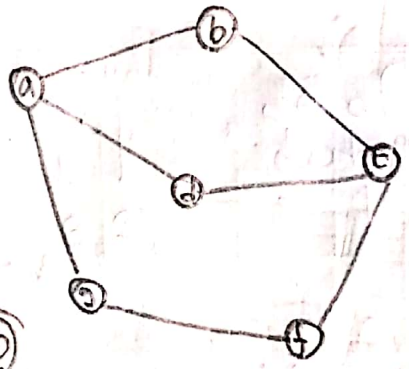
Sub :

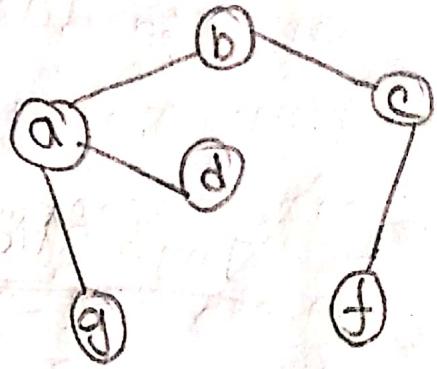
Time : Date : / /

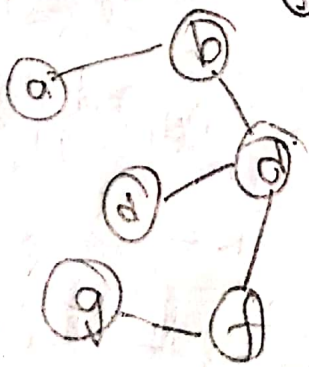
Spanning tree:

A graph which contains all vertices with minimum number of edges

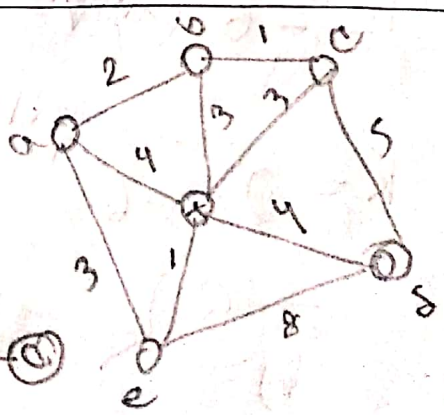
1. A spanning tree subgraph
2. A Tree

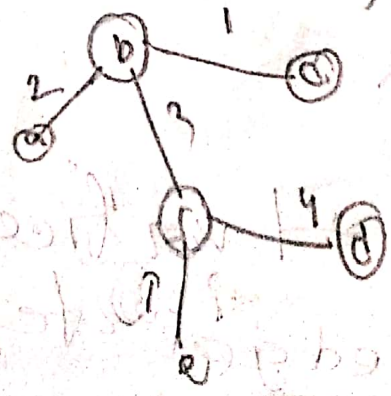






Minimum Spanning tree:





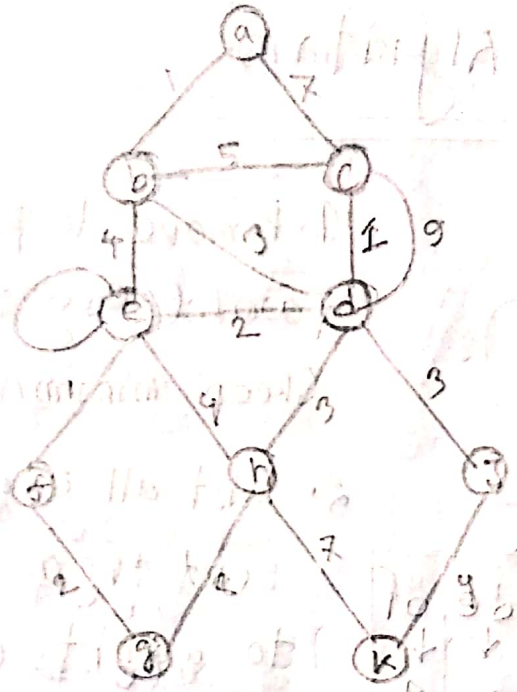
1. Prims Algo
2. Kruskal's algo

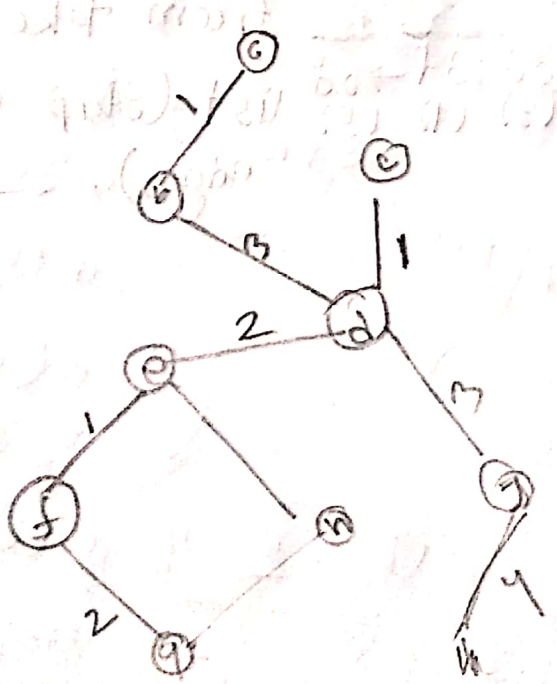
Sat Sun Mon Tue Wed Thu Fri

Sub :

Time : Date : / /

prims Algorithms





steps:

1. **Remove loops:** and parallel edge (keep min weight)
2. while adding new edge, select edge with minimum weight out of the edges from visited vertex (no cycle allowed)
3. stop at (n-1) edges

visited vertices:

a, b, d, c, e, f, g, J

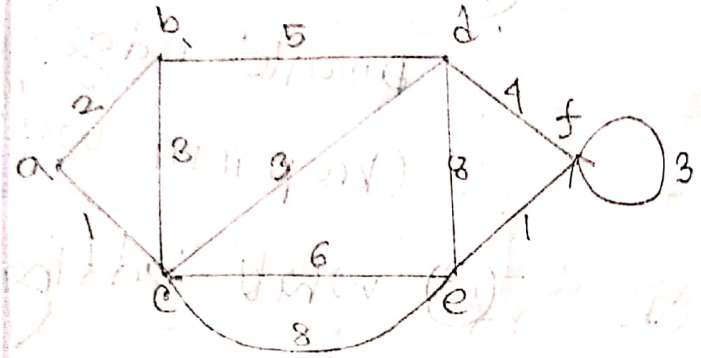
visited edges: ab, ac, bc, db, de, bf, dh, hj, fk, gh, fg

Sat Sun Mon Tue Wed Thu Fri

Sub : Time : Date : / /

(a)

Kruskal's Algorithm

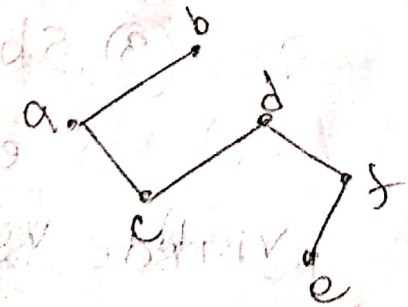


1. Remove loops and parallel edges. (keep minimum weight)
2. List all edges and sort them according to weights (ascending)
3. Take 'n-1' edges from take sorted list (skip cycle making edges).

ab,	ac,	bd,	bc	cd	ce,	de	df
↓	↓	↓	↓	↓	↓	↓	↓
2	1	5	3	3	6	8	4
ef → 1							

ac → ef → ab → bc → cd → df → bd
(1) (1) (2) (3) (3) (4) (5)

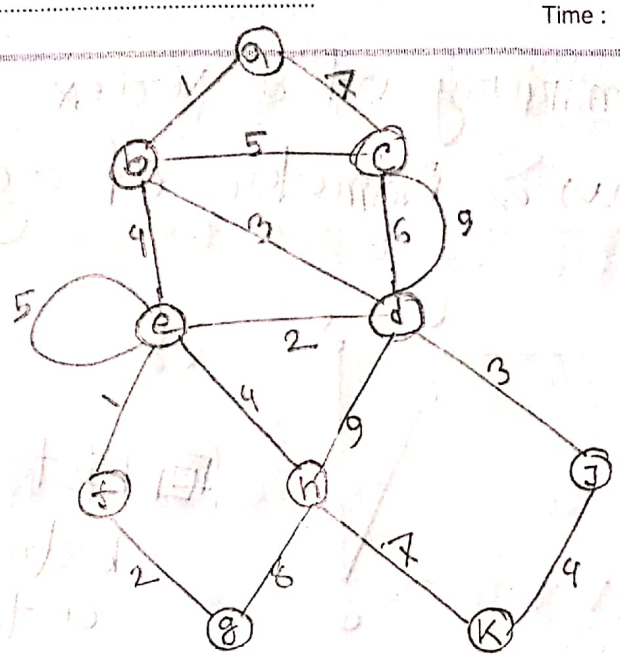
ce → de
(6) (8)



Sat Sun Mon Tue Wed Thu Fri

Sub :.....

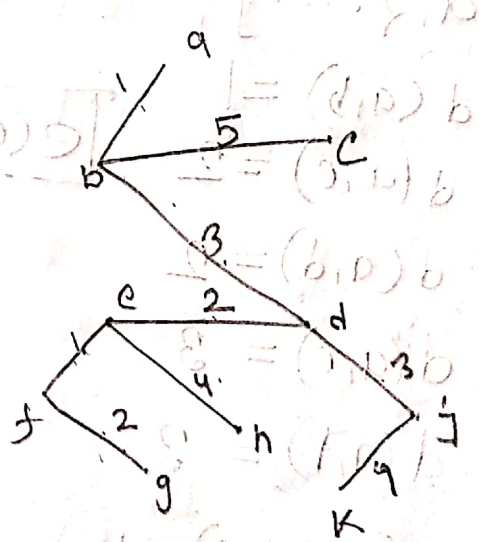
Time : Date : / /



ab – 1	ab – 1
ac – 7	ef – 1
bc – 5	de – 2
bd – 3	fg – 2
be – 4	bd – 3
cd – 6	dj – 3
de – 2	~~be – 4~~
dh – 9	eh – 4
dj – 3	kj – 9
ef – 1	bc – 5
eh – 4	cd – 6
~~fg – 2~~	~~ac – 7~~
gh – 8	~~hk – 7~~
hk – 7	~~cd~~
kj – 4	gh – 8
	dh – 9

n-vertices = 10

No of ~~edges~~ = 9 $|n-1|=9$



$(1+5+3+3+4+2+1+4+2)$

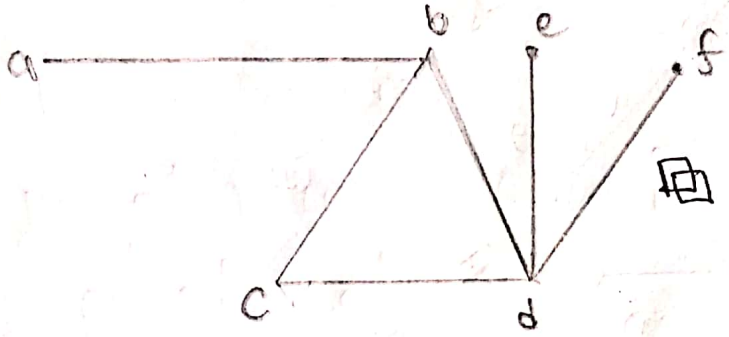
$= 25$

Sat Sun Mon Tue Wed Thu Fri

Time : Date : / /

min value of Radius
e(a)(f) = 2

Eccentricity of a vertex
Radius & Diameter of a graph



- Distance between vertex a to f.

Eccentricity — e(a) = maximum distance between a, {. . . . }

- abcdf - 3
- abdf - 4
- d(a,b) shortest path

d(a,b) = 1
d(a,c) = 2
d(a,d) = 2
d(a,e) = 3
d(a,f) = 3

e(a) = 3

e(a) = 3
e(b) = 2
e(c) = 2
e(d) = 2
e(e) = 3
e(f) = 3

R = 2

e(b) { d(b,a) = 1, d(b,c) = 1, d(b,d) = 1, d(b,f) = 2, d(b,e) = 2 } e(b) = 2

e(c) { d(c,a) = 2, d(c,b) = 1, d(c,d) = 1, d(c,f) = 2 } 2

e(d) { d(d,e) = 2, d(d,b) = 1, d(d,c) = 1, d(d,a) = 2, d(d,f) = 1 } = 1

Sub: #16. Eular graph

Sat Sun Mon Tue Wed Thu Fri

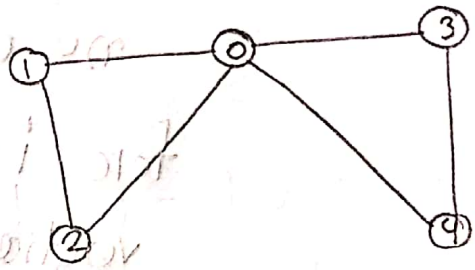
Time: Date: 18/06/20

Eular circuit:

A trail which starts and ends at same vertex

Trail – A walk in which no edge is repeated

walk – Any random traversal in graph.



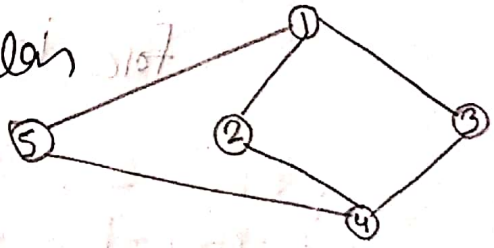
✓

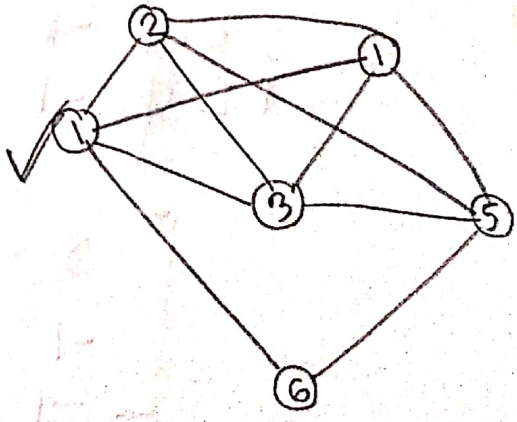
Every vertex in the graph have even degree

even degree – Eular

odd degree – Semi eular

×





Kirchhoff's Matrix-Tree theorem

Find Number of spanning trees.

1. Make a matrix.

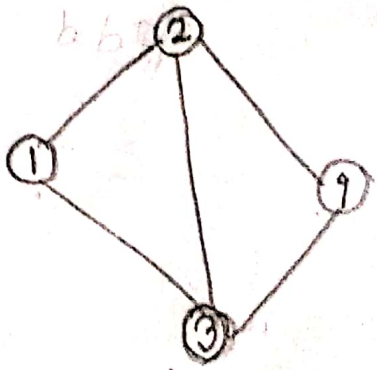
No of rows = no of column = no of vertices

for $i \neq j$, if i & j are adjacent vertices, then $M[i][j] = 1$

if, not $M[i][j] = 0$

for $i = j$, $M[i][j]$ = degree of vertex

i \ j	1	2	3	4
1	2	-1	-1	0
2	-1	3	-1	-1
3	-1	-1	3	-1
4	0	-1	-1	3



2. Delete any one column & find determinant

$$\begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 2 \end{bmatrix} = 3\left[(3\times 2) - (-1)(-1)\right] - (-1)\left[(-1)2 - (-1)(-1)\right] + (-1)\left[(-1)(-1) - (-1)(3)\right]$$

$= 8$