Graph

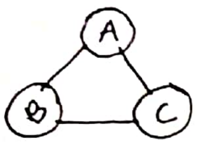
What is graph?

- A Graph in the data structure can be termed as a data structure consisting of data that is stored among many groups of edges (paths) and vertices (nodes), which are interconnected.

- Graph data structure (V, E) is structured with a collection of Nodes and Edges.

Nodes $\Downarrow$ V   Edges $\Downarrow$ E

Example:



Graph, G

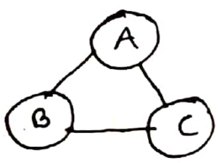
Vertex

$V = \text{Vertices} = \{A, B, C\}$

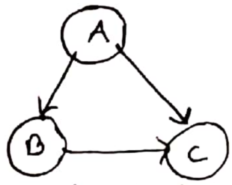
$E = \text{Edges} = \{AB, BC, AC\} = \{BA, CB, CA\}$

$\text{Graph}, G \equiv (V, E)$

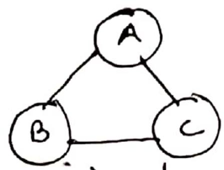
Different types of Graph



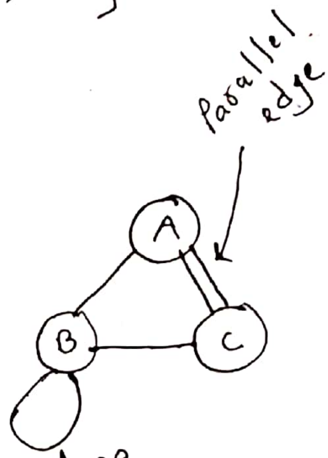
Undirected



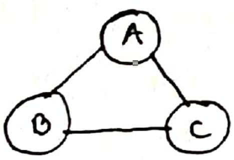
Directed



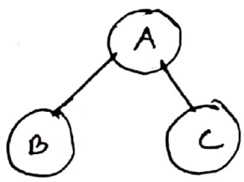
simple



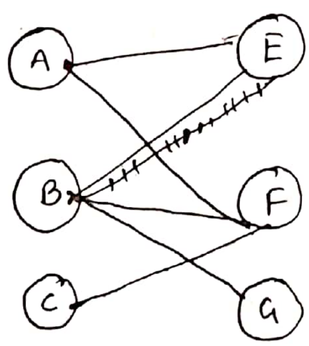
Multigraph



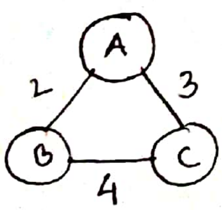
cyclic



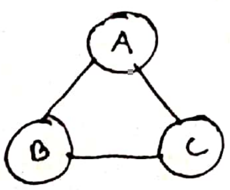
Acyclic



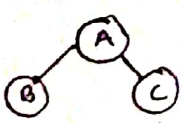
Bipartite graph



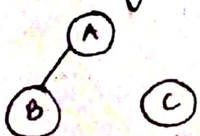
Weighted



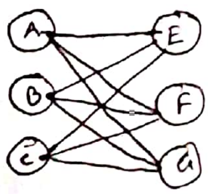
Unweighted



connected graph

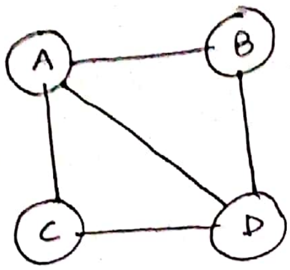


Disconnected graph



complete Bipartite graph

Degree of a vertex :



Degree of vertex A ≡ 3

Degree of vertex B ≡ 2

Degree of vertex C ≡ 2

Degree of vertex D ≡ 3

Total degree = 10

= 2 * 5

= 2 * Total Number of edges

$\Downarrow$

$\underbrace{\{AB, BD, CD, AC, AD\}}_{5}$

A vertex with degree 1 is called an "end vertex"

A vertex with degree 0 is called an isolated vertex

Degree-sum formula

If we add up the degree of all the vertices in a (finite) graph, the result is twice the number of the edges in the graph

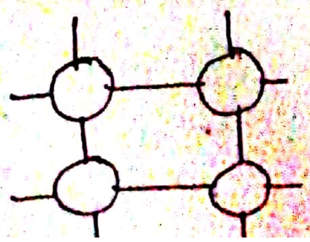
Finite graph :

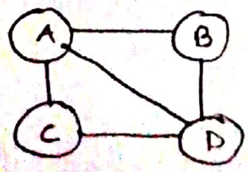
Vertex and edge are finite sets.

Infinite graph

Infinite number of vertices and edges

Infinite





Vertex = A, B, C, D

= 4

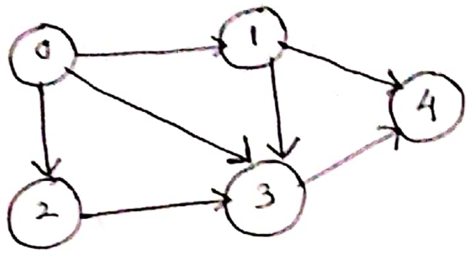
Edges = AB, AC, CD,

Graph Representation

A graph representation is a technique to store graph into the memory of computer. 7

Two ways

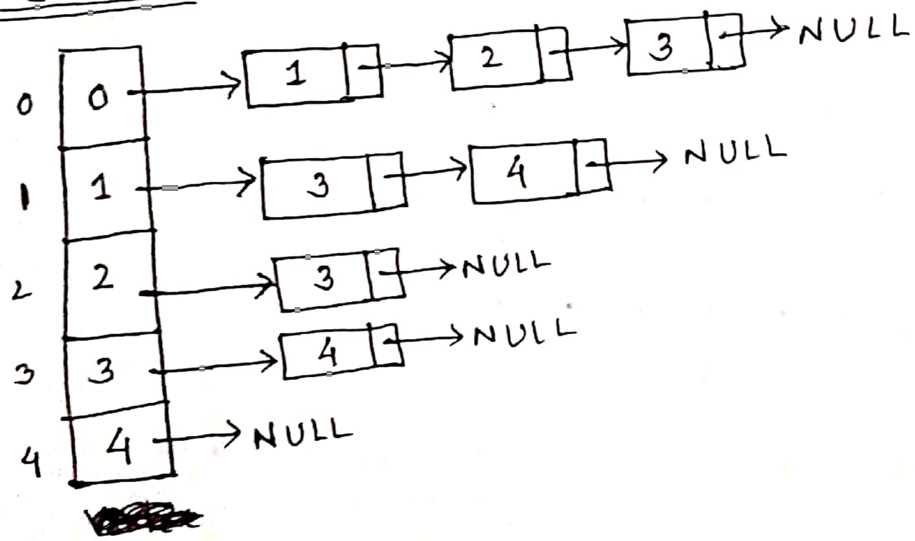
- Adjacency matrix
- Adjacency list



Adjacency Matrix:

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

Adjacency list:



Adjacency structure or neighborhood relationship or graph representation

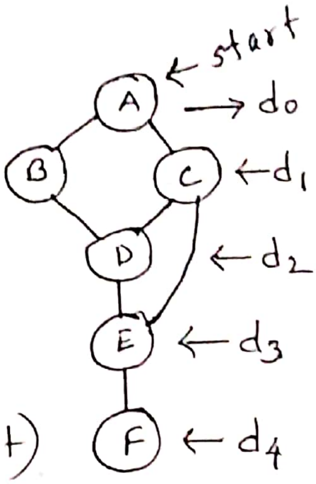
Graph Traversal Algorithms

- Breadth First Search (BFS)
- Depth First Search (DFS)

BFS algorithm

```
Procedure BFS(G, start)
    Queue ⇐ start, Visited ⇐ Empty
    while Queue ≠ Empty
        v ⇐ Queue
        if v is not visited
            Display v
            Visited ⇐ v
            for all non-visited adjacent u of v
                Queue ⇐ u
            end for
        end if
    end while
    return Visited
End Procedure
```

(Undeepest or shallowest node first)



Example:

Queue: | A̸ | | | | | |

Visited: | | | | | | |

v ⇐ Queue	Display v	Visited	Non-visited Adjacents	Queue ⇐ u
v = 'A'	A	A	u = {B, C}	~~B~~ C
v = 'B'	B	A B	u = {~~A~~, D}	~~C~~ D
v = 'C'	C	A B C	u = {~~A~~, D, E}	~~D~~ D E
v = 'D'	D	A B C D	u = {~~B~~, ~~C~~, E}	~~D~~ ~~E~~ E
v = 'D'	X	X	X	X
v = 'E'	E	A B C D E	u = {~~C~~, ~~D~~, F}	~~E~~ F
v = 'E'	X	X	X	X
v = 'F'	F	A B C D E F	u = {~~E~~}	X

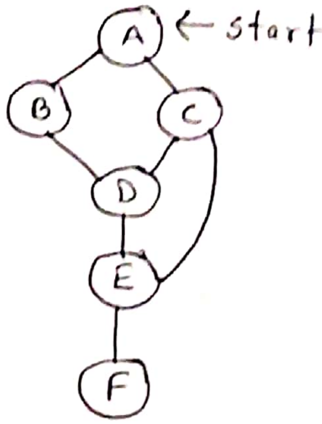
Output = ABCDEF

DFS algorithm

```
Procedure DFS(G, start)
    stack ⇐ start
    Visited ← Empty
    while stack ≠ Empty
        v ⇐ stack
        if v is not visited
            Display v
            Visited ← v
            for all non-visited adjacents u of v
                stack ⇐ u
            end for
        end if
    end while
    return Visited
End Procedure
```

Output = ACEFDB

Example:



stack: | ~~A~~ | | | | | |

Visited: | | | | | | |

v ⇐ stack	Display v	Visited	Non-visited Adjacents	stack ⇐ u
v = 'A'	A	A	u = {B, C}	stack: B, ~~C~~
v = 'C'	C	A, C	u = {~~A~~, D, E}	stack: B, D, ~~E~~
v = 'E'	E	A, C, E	u = {~~C~~, D, F}	stack: B, D, ~~D~~, ~~F~~
v = 'F'	F	A, C, E, F	u = {~~E~~}	×
v = 'D'	D	A, C, E, F, D	u = {B, ~~C~~, ~~E~~}	stack: ~~B~~, ~~D~~, ~~B~~
v = 'B'	B	A, C, E, F, D, B	u = {~~A~~, ~~D~~}	×
v = 'D'	×	×	×	×
v = 'B'	×	×	×	×

Topological Ordering:

L ← Empty list that will contain the sorted elements

Set ← Set of all nodes ~~that~~ with no incoming edge

while Set is not empty do

    remove a node m from Set

    add n to L

    for each node n with an edge e from m to n do

        remove edge e from the graph

        if n has no other incoming edges then

            insert n into Set

        endif

    end for

end while

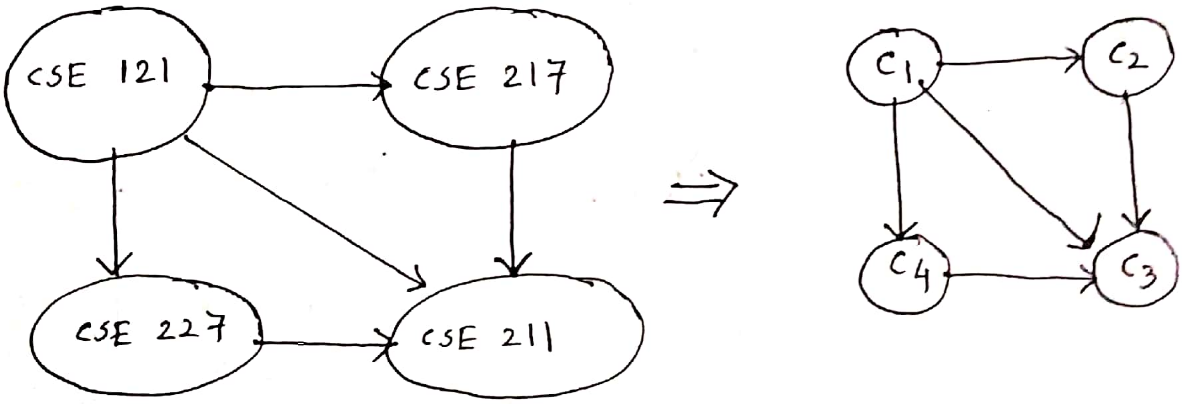
if graph has edges then

    return error (graph has at least one cycle)

else

    return L (a topologically sorted error)

Example:

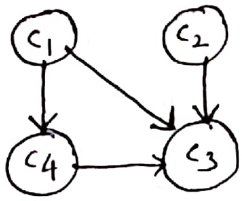
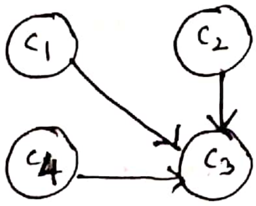
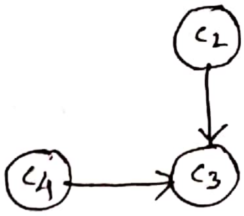


Find topological order

Ans:

L = { }

Set				
	~~C_1~~			

$m \Leftarrow$ Set	$L \Leftarrow m$	mn	Discard mn	set $\Leftarrow n$
$m = c_1$	$L \Leftarrow c_1$	$\underset{m}{c_1}\ \underset{n}{c_2}$		set $\cancel{c_2}$
		$\underset{m}{c_1}\ \underset{n}{c_4}$		set $\cancel{c_2}$ $\cancel{c_4}$
		$\underset{m}{c_1}\ \underset{n}{c_3}$		$\times$
$m = c_2$	$L \Leftarrow c_1 c_2$	$c_2 c_3$		$\times$
$m = c_4$	$L \Leftarrow c_1 c_2 c_4$	$c_4 c_3$	Empty	set $\cancel{c_3}$
$m = c_3$	$L \Leftarrow c_1 c_2 c_4 c_3$	$\times$	Empty	$\times$

return L

$\Rightarrow c_1 c_2 c_4 c_3$

Output $= c_1\ c_2\ c_4\ c_3$

CSE 121   CSE 217   CSE 227   CSE 211

Topological Order $\equiv$ CSE 121   CSE 217   CSE 227   CSE 211