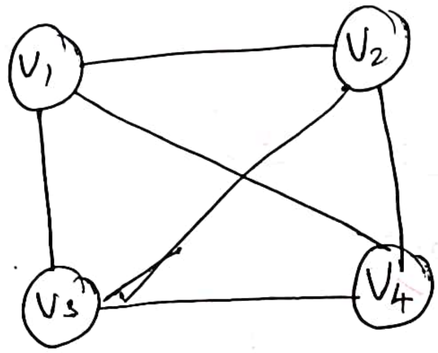
UNIT - III
Graphs

Defn.

A graph $G = (V, E)$ consists of a set of vertices V and set of edges 'E'



→ vertices referred as nodes.

→ Arc between the nodes are referred as edges.

→ V_1, V_2, V_3, V_4 are vertices

→ (V_1, V_2) (V_1, V_3) (V_1, V_4) (V_2, V_3). (V_4, V_1) (V_3, V_4) edges.

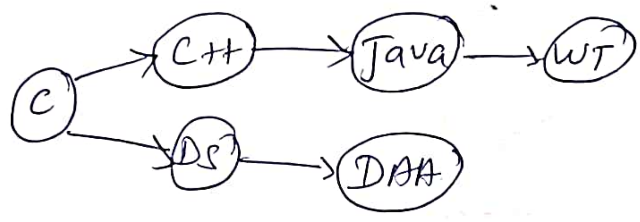
Basic Terminologies (Types of graphs)

1. Directed graph
2. Undirected "
3. weighted "
4. Complete " $n(n-1)/2$
5. Degree (Indegree, outdegree)
6. cyclic graph
7. Acyclic "

Topological sorting:

→ It's a linear ordering of vertices in a directed acyclic graph. such that if there is a path from v_i to v_j, v_j appears after v_i in linear ordering.

Eg.



→ A directed edges (v, w) indicates that Course v must be Completed before the Course w may be attempted.

→ Topological ordering is not possible if the graph has cycle.

* To implement the topological sort perform the following steps:

① find indegree for every vertex.

② Place the vertices whose indegree is '0' on the empty queue.

③ Dequeue the vertex 'v' and decrement the indegree of all its adjacent vertices

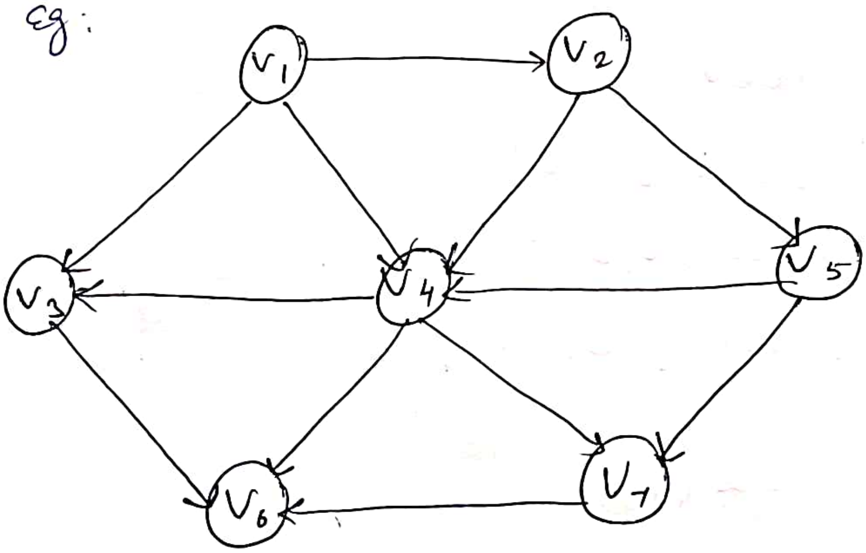
④ Enqueue the vertex on the queue if its indegree falls to '0'.

⑤ Repeat from step 3 until the queue becomes empty.

⑥ Topological ordering is the order in which the vertices dequeue.

→ Use stack or queue for implementation.

eg:



Ind

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	1	1	1	0	0	0
V_2	0	0	0	1	1	0	0
V_3	0	0	0	0	0	1	0
V_4	0	0	1	0	0	1	1
V_5	0	0	0	1	0	0	1
V_6	0	0	0	0	0	0	0
V_7	0	0	0	0	0	1	0

Indegree	v_1	v_2	v_3	v_4	v_5	v_6	v_7
	0	1	2	3	1	3	2

Routine:

```
void topsort (Graph G)
{
  Queue Q;
  int Counter =0;
  Vertex V, W;
  Q = Createqueue (NumVertex);
  Makeempty (Q);
  for each vertex V
      if (Indegree [V] = =0)
          Enqueue (V,Q);
   while (!IsEmpty (Q))
    {
      V= Dequeue (Q);
      Topnum [V] =++ Counter;
      for each W adjacent to V
       if (-- Indegree [W]==0)
            Enqueue (W,Q);
    }
   if (Counter! = NumVertex)
       Error ("Graph has cycle");
   Dispose queue (Q);
}
```

Indegree before Enqueue

Vertex	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	→ 0	0	0	0	0	0	0
V_2	1	→ 0	0	0	0	0	0
V_3	2	1	1	1	→ 0	0	0
V_4	3	2	1	→ 0	0	0	0
V_5	1	[illegible]	→ 0	0	0	0	0
V_6	3	3	3	3	2	1	→ 0
V_7	2	2	2	1	0	→ 0	0
Enqueue	V_1	V_2	V_5	V_4	V_3, V_7	–	V_6
Dequeue	~~V_1~~	V_2	V_5	V_4	V_3	V_7	V_6

adjacent: V_3, V_4 | V_4, V_7 | V_3, V_6, V_7 | V_6 | V_6