Topological Sorting

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering.

Input File:

So, the input file that we have used in our program is ***'graphs.txt'***. File contains 50 graphs, where first line contains the 5 vertices and last line contains the 50 vertices. Each graph G :(V, E) is started with the ** and ends with the dashed line. V is the index set of vertices. Let |V|=6, then V= {0, 1, 2, 3, 4, 5} and E is the number of directed edges.

Following is the sample input of graph;

```
**G1:V={01234}
(u, v) E = {( 0, 1)( 0,  3)( 0,  4)( 3, 1) }
----------------
**G2:V={01234}
(u, v) E = {( 1, 0)( 1,  2)( 1,  3) ( 1, 4) ( 2, 4) ( 4, 0) ( 4, 3) }
----------------
**G3:V={01234}
(u, v) E = {( 1, 0)( 1,  2)( 1,  4)( 2, 3)( 2, 4)( 3,  0) }
----------------
**G4:V={01234}
(u, v) E = { ( 0, 1)( 0, 2)( 0, 4)( 1,  3)( 1, 4)( 2, 3) }
----------------
**G5:V={01234}
(u, v) E = { ( 1, 0)( 1, 3)( 1, 4)( 3, 2)( 4, 0)( 4, 2)( 4, 3) }
----------------
```

Output:

Below is the output of the Program when graphs.txt is given as input

```
Topological Orders:
G1:   2 0 4 3 1
G2:   1 2 4 3 0
G3:   1 2 4 3 0
G4:   0 2 1 4 3
G5:   1 4 3 2 0
G6:   4 3 1 2 9 0 8 5 6 7
G7:   7-> No more in-degree 0 vertex; not an acyclic graph
G8:   4 6 2 5 0 9 8 7 1 3
G9:   5 4 7 1 6 2 3 8 9 0
G10:   4 0 7 8 3 6 5 1 9 2
G11:   3 7 13 10 11 9 14 6 1 2 12 0 8 4 5
G12:   9 10 6 11 1 7 2 0 5 14 13 8 4 3 12
G13:   13 7 2 11 9 12 5 6 3 4 14 8 10 1 0
G14:   9 8 1 4 10 13 11 7 5 3 6 0 14 2 12
G15:   14 12 9 7 2 8 11 10 6 1 13 5 3 0 4
G16:   18 10 11 6 15 4 0 17 8 2 12 3 14 19 16 7 5 9 13 1
G17:   9 18 12 14 8 7 10 6 3 1 15 0 4 16 17 13 2 11 19 5
G18:   15 3 14 19 18 1 4 16 11 5 17 6 0 10 12 8 2 9 7 13
```

G19: No more in-degree 0 vertex; not an acyclic graph
G20: 14 6 13 4 5 10 18 12 8 7 0 16 3 17 9 19 1 2 15 11
G21: 13 12 21 1 22 7 4 18 17 2 15 19 14 23 24 16 5 3 0 10 9 11 20 8 6
G22: No more in-degree 0 vertex; not an acyclic graph
G23: 13 8 7 3 10 6 18 2 4 16 14 5 1 24 22 21 19 20 9 23 11 12 0 17 15
G24: No more in-degree 0 vertex; not an acyclic graph
G25: 15 5 14 0 7 12 4 17 11 20 13 16 3 8 6 23 22 10 2 19 21 9 1 24 18
G26: 25 14 7 10 6 22 19 15 1 17 20 0 27 18 11 26 21 9 8 12 3 28 5 29 13 16 24 23 2 4
G27: 12 7 21 20 25 23 11 28 27 1 16 4 18 14 19 8 24 5 26 22 15 29 6 2 0 17 13 9 3 10
G28: 13 0 3 29 15 6 1 8 16 19 11 17 4 26 28 18 20 27 23 22 21 24 12 14 5 10 7 9 25 2
G29: 25 23 17 3 2 5 9 4 29 18 11 13 15 20 16 1 10 8 21 6 14 24 19 28 7 12 0 26 22 27
G30: No more in-degree 0 vertex; not an acyclic graph
G31: 6 33 34 15 4 2 1 16 8 12 20 30 7 25 9 19 13 26 11 32 3 10 24 17 31 14 23 29 5 22 27 21 18 0 28
G32: No more in-degree 0 vertex; not an acyclic graph
G33: 0 18 7 29 33 25 6 3 34 19 10 20 28 11 32 17 21 8 22 5 1 27 4 23 15 26 30 9 24 12 31 14 13 2 16
G34: 15 13 7 5 4 19 27 14 0 28 23 25 21 6 11 31 12 34 3 20 9 1 29 26 16 8 10 33 32 22 18 30 2 17 24
G35: 26 25 19 16 11 8 22 14 13 6 31 32 21 5 0 10 30 24 28 18 3 34 15 33 29 27 9 23 7 1 12 4 2 17 20
G36: 19 17 4 18 10 0 30 12 24 26 25 8 31 27 7 36 21 22 23 3 15 9 20 28 11 37 14 1 38 16 5 33 29 34 2 39 13 35 6 32
G37: 10 33 3 16 34 12 15 13 11 21 1 30 29 5 4 37 20 32 17 39 2 35 14 6 27 25 38 18 24 28 36 8 22 26 9 31 7 19 0 23
G38: 39 17 11 5 9 35 1 23 26 8 34 12 18 15 36 31 28 24 27 10 7 2 21 13 30 0 38 6 29 16 37 32 22 33 20 14 19 25 4 3
G39: 20 13 0 16 12 7 37 8 6 36 18 32 31 29 26 17 34 39 11 30 21 3 1 23 25 24 5 33 28 4 19 38 10 2 35 27 14 22 9 15
G40: 4 39 20 14 24 11 9 17 34 8 29 33 31 18 16 36 22 6 23 27 21 5 3 13 25 30 38 32 2 12 35 1 10 15 28 37 19 0 26 7
G41: 21 39 8 0 37 23 11 13 33 12 28 1 15 34 18 38 19 25 2 41 5 44 16 20 35 3 22 14 31 36 4 26 32 43 24 9 27 42 30 7 10 6 17 29 40
G42: 39 2 21 16 14 0 37 7 41 15 19 20 12 22 40 31 11 25 27 13 30 23 42 8 36 5 26 29 1 3 32 34 4 44 28 18 38 35 17 43 9 33 10 24 6
G43: No more in-degree 0 vertex; not an acyclic graph
G44: 30 22 44 40 4 24 13 43 9 0 23 14 5 42 32 18 36 17 31 27 10 39 19 37 16 34 33 21 11 2 29 28 35 6 15 8 25 41 7 38 26 20 1 3 12
G45: 28 19 23 36 34 41 30 12 5 16 3 43 11 17 26 13 7 35 18 22 1 14 42 6 40 9 32 24 0 44 38 10 20 25 8 37 21 31 4 29 2 39 33 27 15
G46: 38 14 41 37 22 11 47 7 31 5 48 23 17 39 29 26 3 28 46 44 45 19 43 1 8 42 33 27 36 12 0 21 34 15 49 32 2 9 40 20 25 10 4 6 30 18 16 24 35 13
G47: No more in-degree 0 vertex; not an acyclic graph
G48: 38 31 24 35 16 1 27 12 10 18 32 2 40 26 6 37 48 20 19 9 0 22 33 36 13 4 47 15 49 45 39 46 29 7 3 23 21 11 5 17 43 14 44 28 30 42 41 25 8 34
G49: 33 5 4 30 21 37 24 23 8 22 15 6 3 32 14 18 11 9 1 34 29 7 28 2 0 38 19 47 42 27 10 39 36 12 49 31 20 16 48 35 46 43 41 17 25 26 45 44 40 13
G50: No more in-degree 0 vertex; not an acyclic graph

Method:

In case of ***Topological Sort***, we use

- Temporary Stack
- Recursively call the Topological sort for sorting all the adjacent vertices of graph
- Store the vertices in the Stack
- Print the nodes present in Stack

Algorithm:

Algorithm that is used for ***Topological Sort*** is

- Initializing the stack to store nodes
- Initializing the List of length(l) to store visited nodes
- Start loop from 0 to l-1

- Check if node is not marked true in visited array
  - Call the topological sort for
    - Marking node true in visited array
    - Run loop on all nodes which has direct edge with the current node
    - Check if node is not marked true in visited array
      - Call the topological sort
  - Push nodes in stack
- Print the nodes in stack

Data Structure:

We have used ***Stack, List Array*** in our program.

Stack:

Stack is used to store the nodes.

List Array:

List Array is used to mark the nodes that are visited or not.

Efficiency of Program:

Time complexity:

O (V+E) , where V is the number of vertices and E is the number of edges

Space complexity:

O (V) , where V is the number of vertices