Unit-5

Graph Algorithm

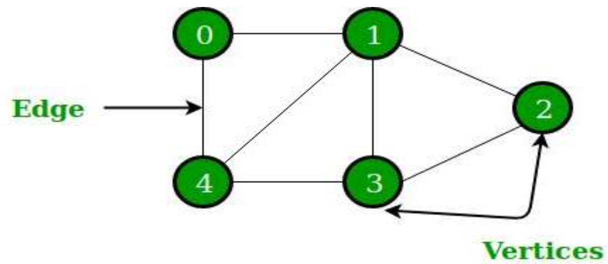
Prof. Ritesh Upadhyay

Faculty of Engineering and Technology

Graph Theory

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph can be defined as,

A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes.



In the above Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

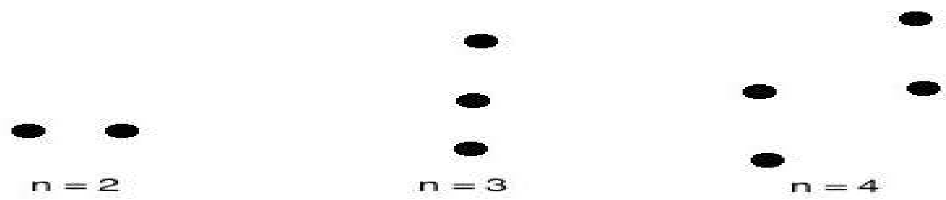
Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or

telephone network or circuit network. Graphs are also used in social networks like LinkedIn, Facebook. For example, in Facebook,

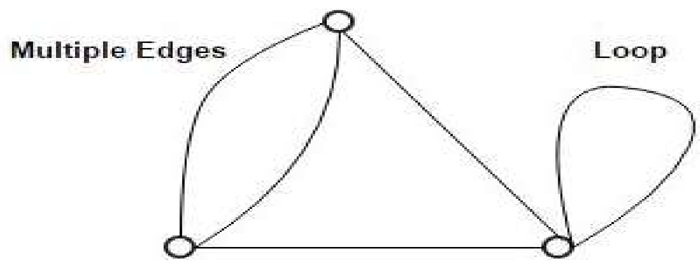
Each person is represented with a vertex(or node). Each node is a structure and

Types of Graphs

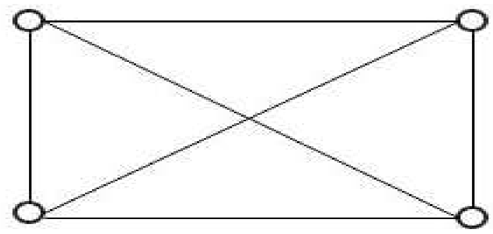
1. Null Graph: A null graph is a graph in which there are no edges between i vertices. A null graph is also called empty graph.



2.Simple Graph: A simple graph is the undirected graph with no parallel edges and no loops



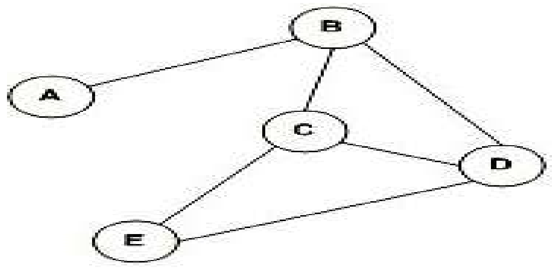
Not a Simple Graph



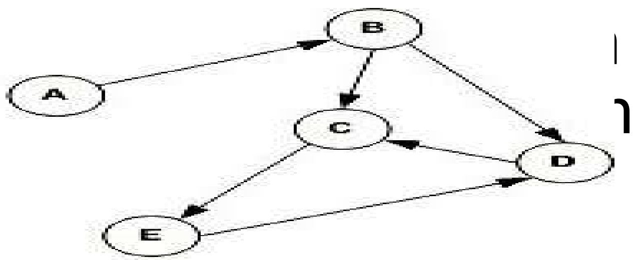
Simple Graph

Types of Graphs

3.Undirected Graph: An undirected graph is a graph whose edges are not directed.



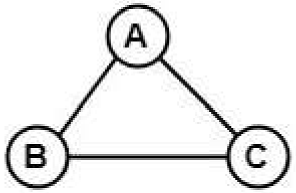
In the above graph since there is no directed edges, therefore it is an undir graph.

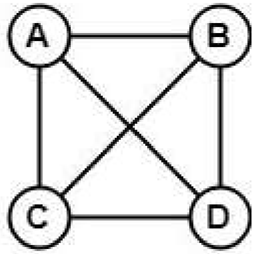
4. Directed Graph: A di 

in which the edges are directed by arrows. D own as digraphs

Types of Graphs

5.Complete Graph: A graph in which every pair of vertices is joined by exa
one edge is called complete graph. It contains all possible edges.

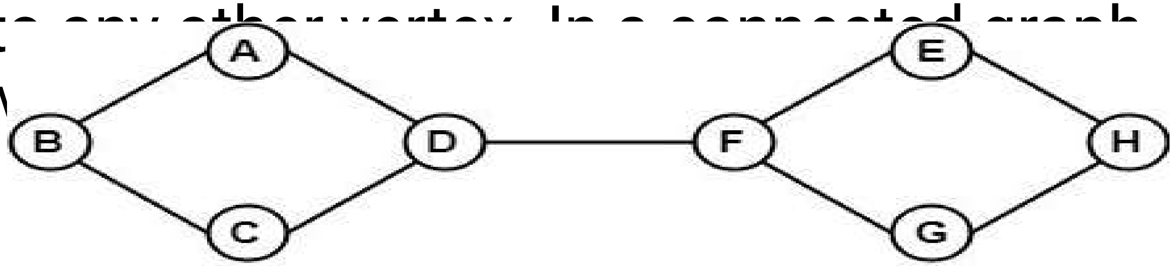




In the above example, since each vertex in the graph is connected with all t
remaining vertices through exactly one edge

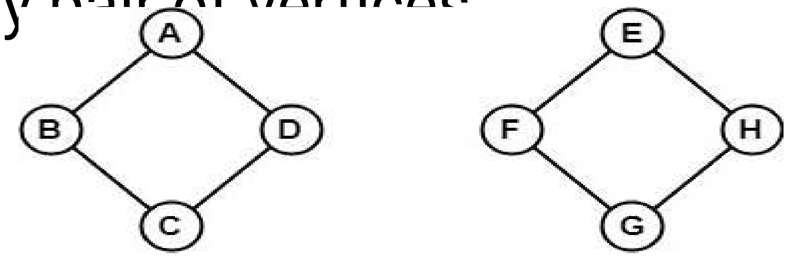
therefore, both graphs are complete graph.

6. Connected Graph: A connected graph is a graph in which we can visit fro
any one vertex t at least one ed
path exists betw



Types of Graphs

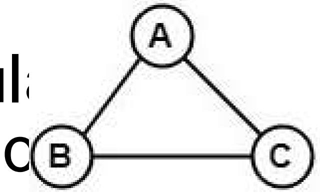
7. Disconnected Graph: A disconnected graph is a graph in which any path
not exist between every pair of vertices

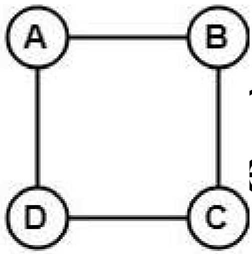


The above graph consists of two independent components which are disconnected. Since it is not possible to visit from the

vertices of one component to the vertices of other components therefore, i
disconnected graph.

8. Regular Graph: A Regul
hich degree of all the verti
is same. If the degree c
en it is called k-regular gra





Graph Representation

As in other ADTs, to manipulate graphs we need to represent them in some useful
Basically, there are three ways of doing this:

1. Adjacency Matrix
2. Adjacency List
3. Adjacency Set

Adjacency Matrix: In this method, we use a matrix with size V*V. The value
matrix are Boolean. Let us assume the

matrix is Adj. The value Adj [u, v] is set to 1 if there is an edge from vertex u
vertex v and 0 otherwise.

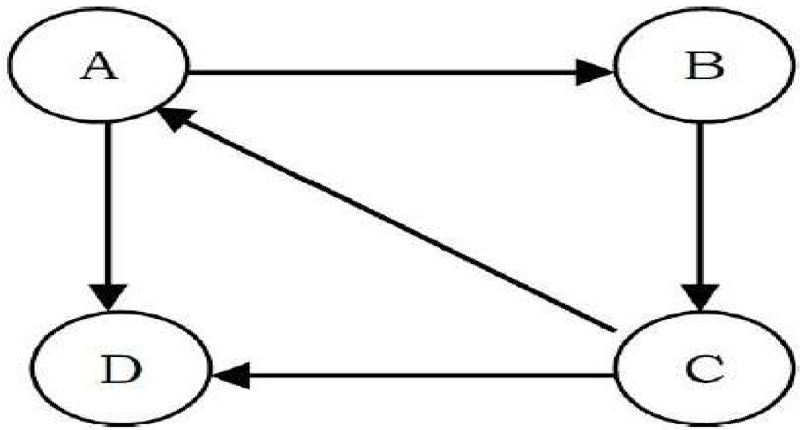
In the matrix, each edge is represented by two bits for undirected graphs. Th
means, an edge from u to v is represented by 1 value in

both Adj[u,v] and Adj[u,v]. To save time, we can process only half of this sym
matrix. Also, we can assume that there is an

“edge” from each vertex to itself. So, Adj[u, u] is set to 1 for all vertices. If th

Adjacency Matrix

An example, consider the directed graph below.



The adjacency matrix for this graph can be given as:

	A	B	C	D
A	0	1	0	1
B	0	0	1	0
C	1	0	0	1
D	0	0	0	0

Adjacency List

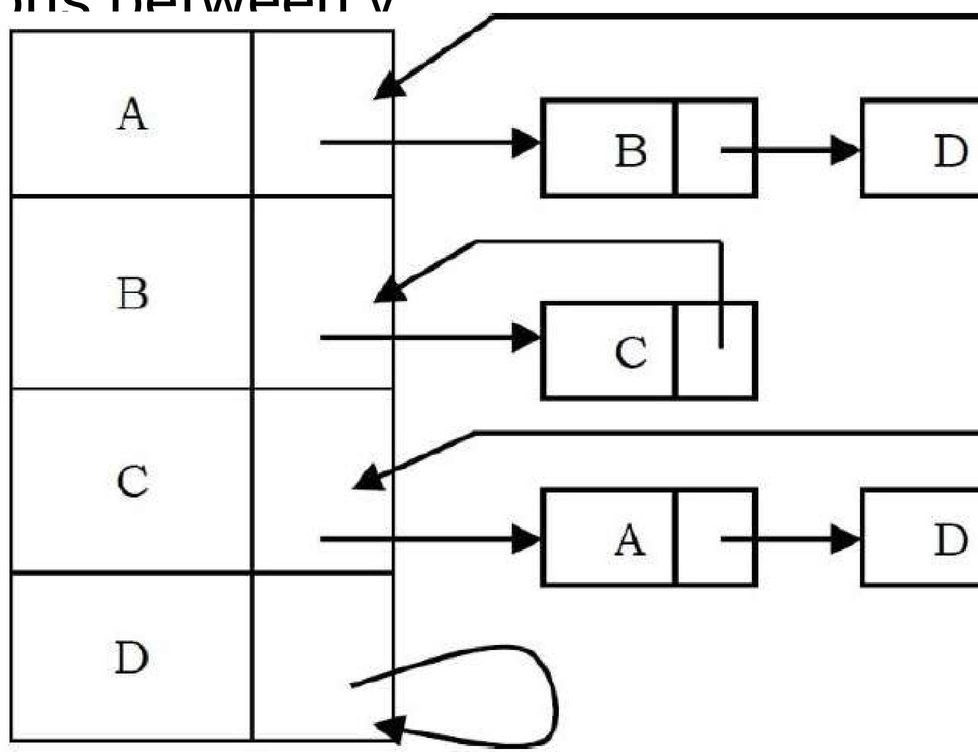
In this representation all the vertices connected to a vertex v are listed on ar adjacency list for that vertex v. This can be easily

implemented with linked lists. That means, for each vertex v we use a linked and list nodes represents the connections between v

and other vertices to which v has an ec equal to the number of vertices in tl

Considering the same example as that

the adjacency list representation can l



Graph Traversals

To solve problems on graphs, we need a mechanism for traversing the graphs. Graph traversal algorithms are also called

graph search algorithms. Like trees traversal algorithms (In-order, Preorder, Post-order and Level-Order traversals), graph

search algorithms can be thought of as starting at some source vertex in a graph and "searching" the graph by going

through the edges and marking the vertices. Now, we will discuss two suc algorithms for traversing the graphs.

- Depth First Search [DFS]
- Breadth First Search [BFS]

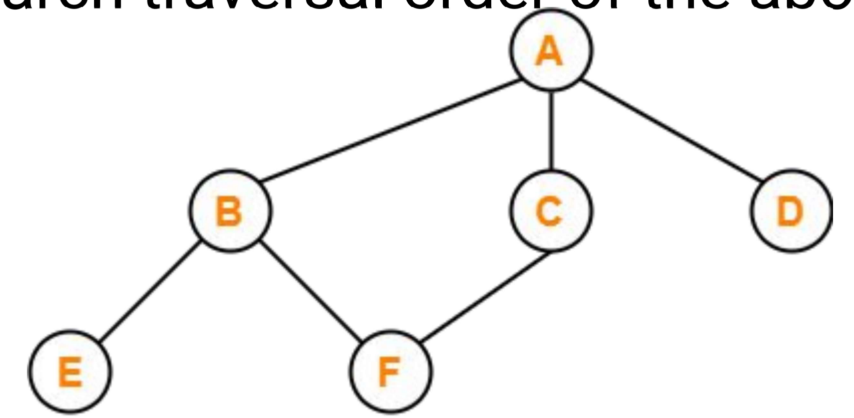
Breadth First Search [BFS]

- Breadth First Search or BFS is a graph traversal algorithm.
- It is used for traversing or searching a graph in a systematic fashion.
- BFS uses a strategy that searches in the graph in breadth first manner whenever possible.
- Queue data structure is used in the implementation of breadth first searc

BFS Example- the breadth first search traversal order of the above graph is-

A, B, C, D, E, F

Time Complexity : O(V+E)



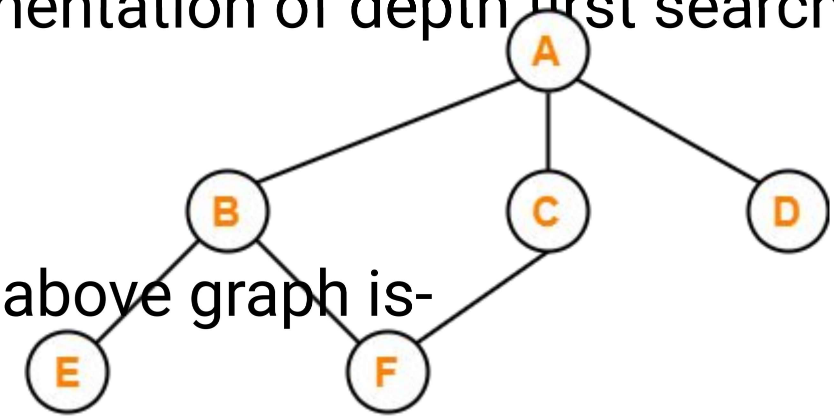
Breadth First Search Example

Algorithm of Breadth First Search [BFS]

```
BFT(V)
{
Visited(V)=1;
Add(V,Q);
While(Q ≠ Ø)
{
v=delete(Q);
For all the adjacent w for v
{ if(! Visited w)
{
Add (w,Q);
Visited(V)=1;}
}}}
```

Depth First Search [BFS]

- Depth First Search or DFS is a graph traversal algorithm.
- It is used for traversing or searching a graph in a systematic fashion.
- DFS uses a strategy that searches “deeper” in the graph whenever possible.
- Stack data structure is used in the implementation of depth first search.



Depth First Search Example

The depth first search traversal order of the above graph is-

A, B, E, F, C, D

Algorithm of Depth First Search [DFS]

```
DFT(V)
{
Visited(V)=1;
For each vertex W adjacent of V
{
If(!visited W)
DFT(W)
}
}
```

Time Complexity : O(V+E)

Topological order & 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. Topological Sorting for a graph is not possible if the graph is not a DAG.

OR

- Topological Sort is a linear ordering of the vertices in such a way that if there an edge in the DAG going from vertex ‘u’ to vertex ‘v’, then ‘u’ comes before ‘v’ the ordering.

It is important to note that-

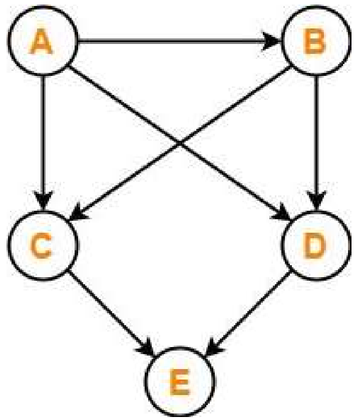
- Topological Sorting is possible if and only if the graph is a Directed Acyclic Graph.
- There may exist multiple different topological orderings for a given directed acyclic graph.

<u>APPLICATION OF TOPOLOGICAL SORT</u> : Few important applications of topologic sort are-

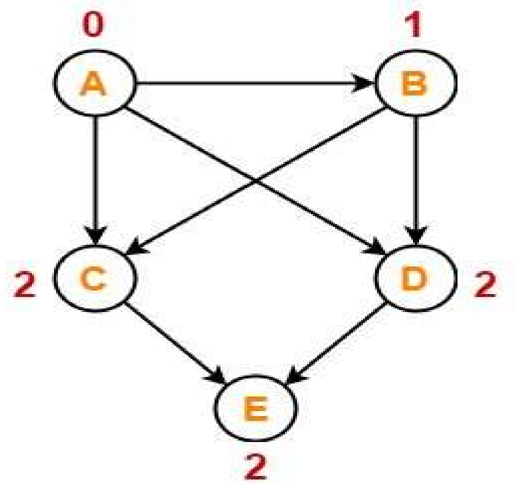
1) Scheduling jobs 2) Instruction Scheduling 3) Data Serialization

Example of Topological Sort

Find the number of different topological orderings possible for the given graph-



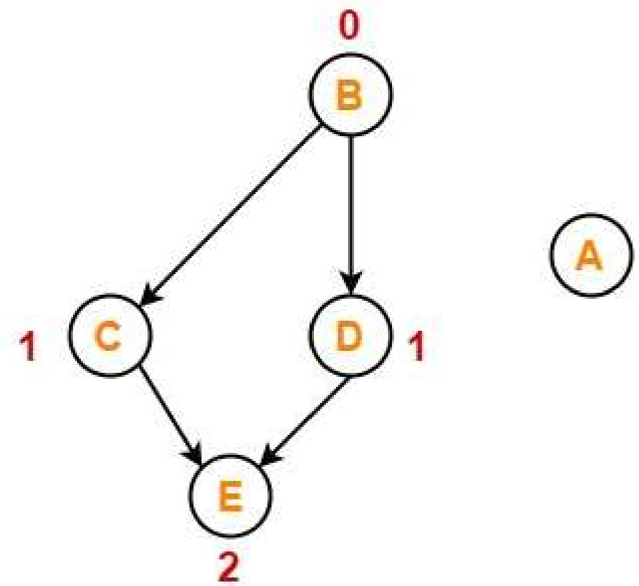
Solution: Step-01: W ch vertex-



Example of Topological Sort

Step-02

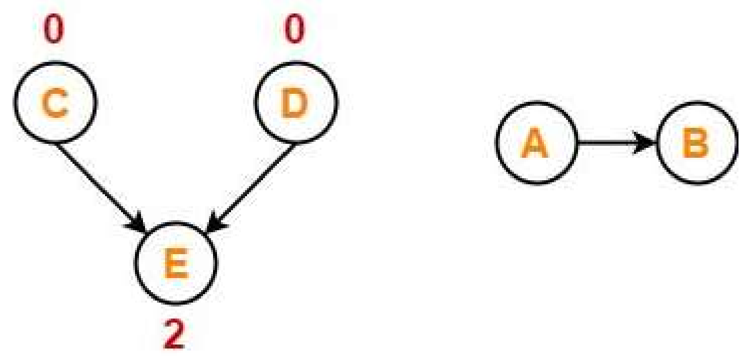
- Vertex-A has the least in-degree.
- So, remove vertex-A and its associated edges.
- Now, update the in-degree of other vei



Example of Topological Sort

Step-03:

- Vertex-B has the least in-degree.
- So, remove vertex-B and its associated edges.
- Now, update the in-degree of other vertices.



Example of Topological Sort

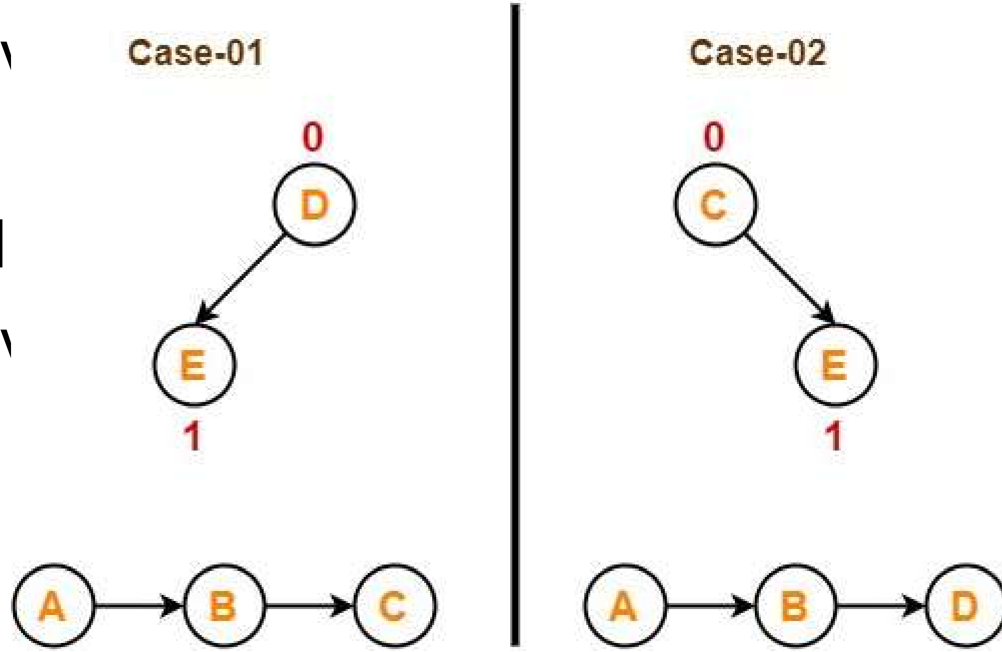
Step-04: There are two vertices with the least in-degree. So, following 2 cases are possible-

In case-01

1. Remove vertex-C and its associated edges.
2. Then, update the in-degree of other v

In case-02

1. Remove vertex-D and its associated
2. Then, update the in-degree of other v



Example of Topological Sort

Step-05: Now, the above two cases are continued separately in the similar manner.

In case-01

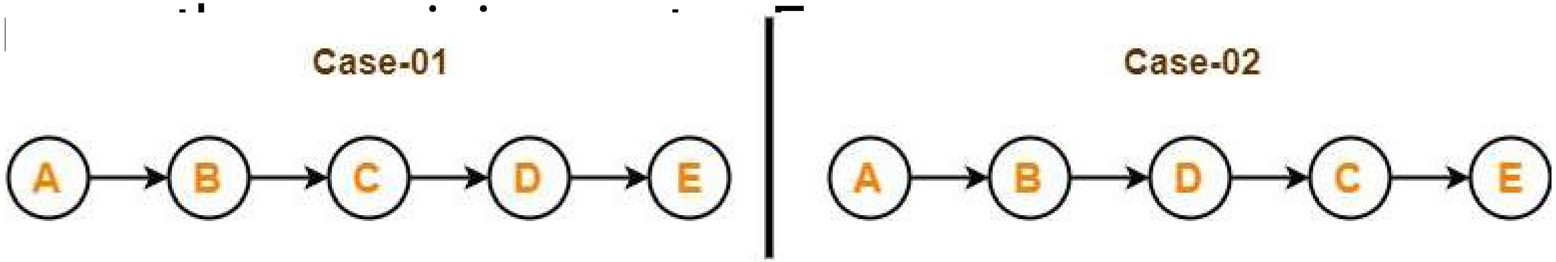
Remove vertex-D since it has the least in-degree.

Then, remove the remaining vertex-E.

In case-02

Remove vertex-C since it has the least in-degree.

Then, re



Title fonts (Roboto Black 34 Pt)

- Body fonts (Roboto 24 Pt)
- Body fonts (Roboto 24 Pt)
- Body fonts (Roboto 24 Pt)
- Body fonts (Roboto 24 Pt)