PONJESLY COLLEGE OF ENGINEERING
NAGERCOIL.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SEMESTER III

CS8391- DATA STRUCTURES

UNIT IV

Non linear data structures - graphs

Definitions- Representation of Graph – types of Graph -Breadth First Traversal- Depth First Traversal - Topological sort - Biconnectivity- Cut Vertex - Euler Circuit-Applications of Graph

---

Q1) a) Define Graph(2)

Or

Q1) b) What do you mean by vertices and edges in a graph?(2)

ANSWER

DEFINITION OF GRAPH

A Graph G is a collection of two sets V and E where V is the collection of vertices V_0,V_1,V_2……V_n called nodes and E is the collection of edges e_1, e_2, e_3……e_n called arcs ,Where an edge is an arc which connects two nodes. Each edge is a pair (v, w) where v, w belongs to V. This can be represented as, G = (V,E).

Q2) a) What are the types of graph?(6)

Or

Q2) b) Define different types of Graph.(6)

ANSWER

TYPES OF GRAPHS

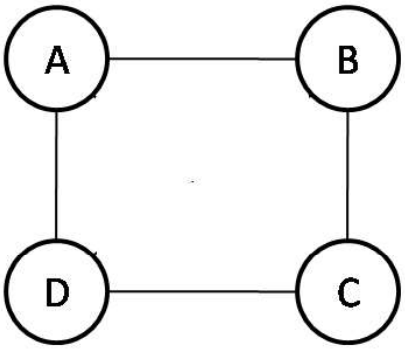
1. ***Undirected Graph***
2. ***Directed Graph***

Undirected Graph:

A graph, which has unordered or undirected pairs of vertices, is called undirected graph. The below shown graph has 4 nodes and 8 edges.

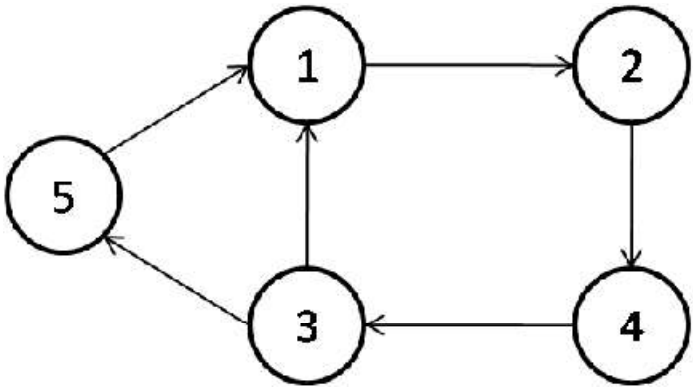
V (G) = {A,B,C,D}

E(G) = {(A,B),(A,D),(B,A),(B,C),(C,B),(C,D),(D,A)(D,C)}



Directed Graph:

A directed graph is a graph which has ordered or directed pair of vertices that is denoted as (V1,V2), where V1 is the tail and V2is the head of the edge. Directed graph is also known as digraph. The below shown graph has 5 nodes and 6 edges and they are,



V(G) = {1,2,3,4,5}
E(G) = {(1,2),(2,4),(4,3),(3,1),(3,5),(5,1)}

Weighted Graph:

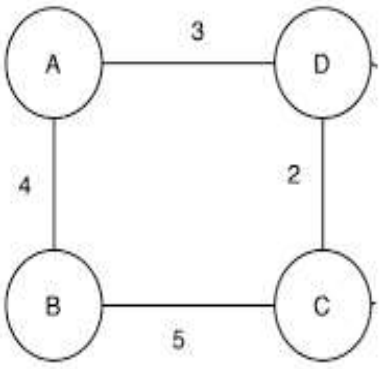
A graph is said to be weighted if its edges have been assigned some non negative values as weight. A weighted graph is also known as network. There are two types of weighted graph.

They are,

1) Weighted undirected graph
2) Weighted directed graph

1) Weighted undirected graph

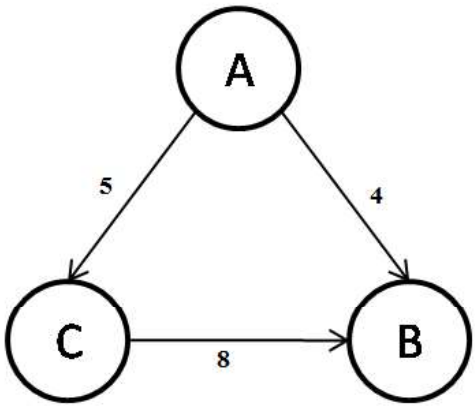
A graph, which has unordered or undirected pairs of vertices with weights is called weighted undirected graph.



V(G) = {A,B,C,D}
E(G) = {(A,B)=(B,A) = 4
(A,D),(D,A)= 3
(D,C)=(C,D)=2
(B,C) = (C,B)=5 }

2) Weighted directed graph

A weighted directed graph is a graph which has ordered or directed pair of vertices with weights



V(G) = {A,B,C}
E(G) = {(A,B) = 4
(A,C)=5
(C,B)=8 }

Q3) a) Write about basic terminologies of Graph(10)

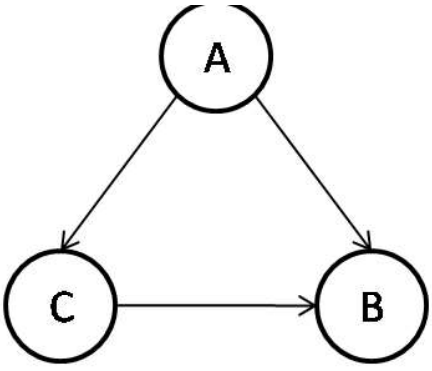
Or

Q3) b) What are the basic terms related to Graph.(13)

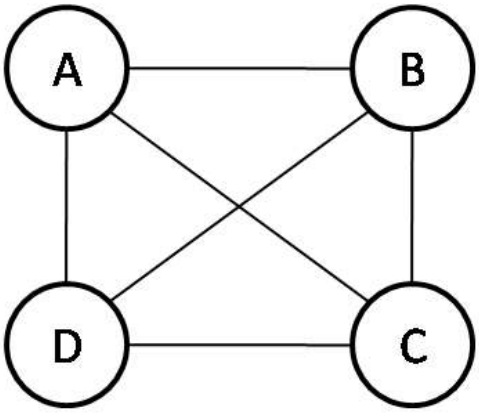
ANSWER

Adjacent Nodes:

Any two nodes which are connected by an edge are called adjacent nodes.



For A, adjacent nodes are B and C.



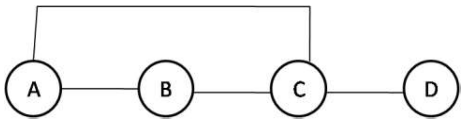
For A , adjacent nodes B,C,D

Incidence:

In an undirected graph, the edge (V0, V1) is incident on nodes.In a directed graph, the edge (V0,V1) is incident from node V0 and is incident to node V1.

Path:

Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.



- In the above graph path1 for ***(A,D)={A-B-C-D}***

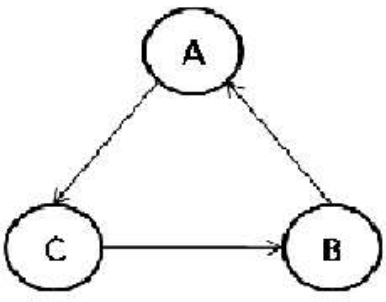
  Path2 for *(A,D)=* ***{A-C-D}***

Length of Path:

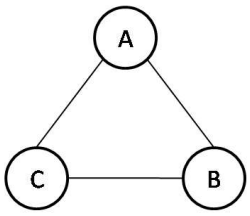
Length of a path is the total number of edges included in the path.

Length of the above path2 {A-C-D}= 2

Cycle: Cycle in a graph is a path in which first and last vertex are the same.

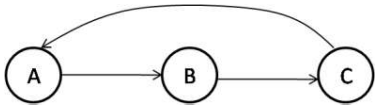


Cycle = A → C → B → A



Cycle = ABCA

Cyclic Graph: A graph is said to be cyclic, if the edges in the graph form a cycle.

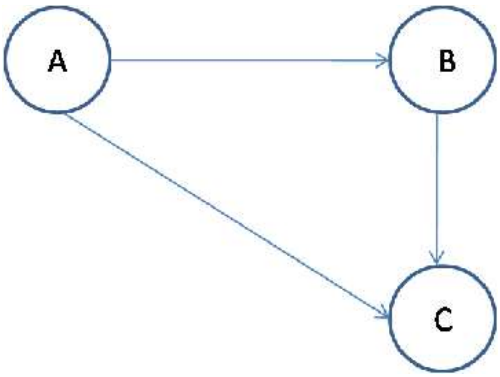


In the above graph, the cyclic path is { A – B – C – A}

Acyclic Graph: A graph is said to be acyclic, if the edges in the graph does not form a cycle and it shown below,

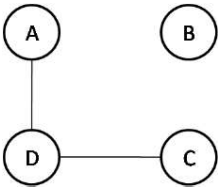
DAG: (DIRECTED ACYCLIC GRAPH)

If a directed graph is acyclic (i.e) if it has no cycles, such types of graphs is called DAG.



Degree:

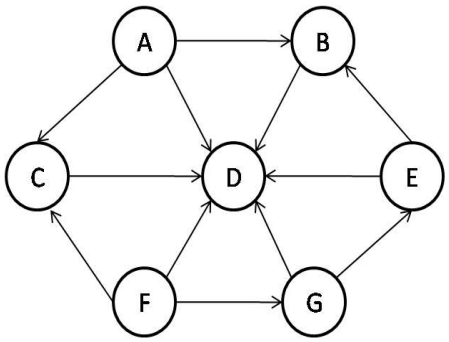
In an undirected graph, the number of edges connected to anode is called the degree of that node (or) degree of anode is the number of edges incident on it. In a directed graph (Digraph), there are two degrees for every node, known as indegree and Outdegree.



- In the above graph, Degree of node A is 1.
  Degree of node B is 0.

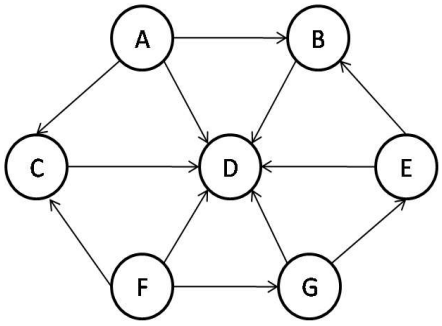
Indegree:

- The indegree of a node is the number of edges coming to that node or in other words edges incident on it.
- From the above graph, Indegree(A)= 0, Indegree(B) = 2, Indegree (D) = 6, Indegree(G) =1.



Outdegree:

- The outdegree of node is the number of edges going outside from that node or in other words the edges incident from it.
- From the above graph, Outdegree(B)= 1, Outdegree(D) = 0, Indegree (F) = 3.



Source:

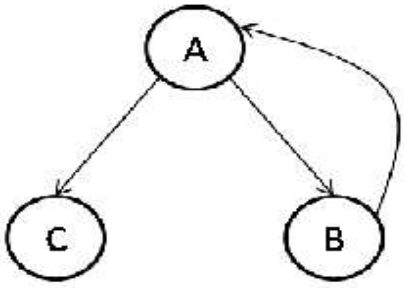
- A node which has no incoming edges, but has outgoing edges , is called a source. The indegree of source is zero. In the above graph, nodes A and F are sources.

Sink:

- A node which has no outgoing sink. The outdegree of a sink is zero. In the above graph, node D is a sink.

Pendant node:

- A node is said to be pendant node, if its indegree is equal to 1 and outdegree is equal to 0.



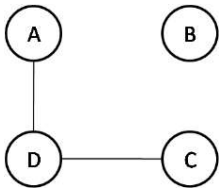
- C is pendant node

Reachable:

- If there is a path from a node to any other node then it will be called as reachable from that node.

Isolated node:

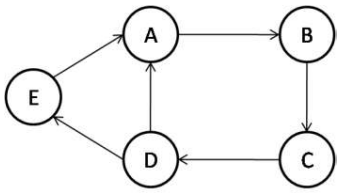
- If a node has no edges connected with any other node, then its degree will be zero and it will called as isolated node.



- In the above graph, B is the isolated node.

Successor and Predecessor:

- In a directed graph, if a node V0 is adjacent to node V1, then V0 is the predecessor of V1 and V1 is the successor of V0.

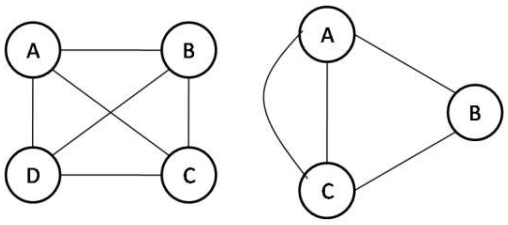


G5

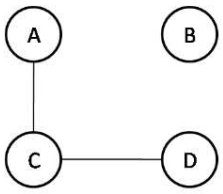
- From the above graph, node A is the predecessor of node B, node B is the predecessor of node C, node B is the predecessor of node A, node C is the predecessor of node B.

Connected Graph:

- An undirected graph is connected if there is a path from any node of graph to any other node or any node is reachable from any other node.



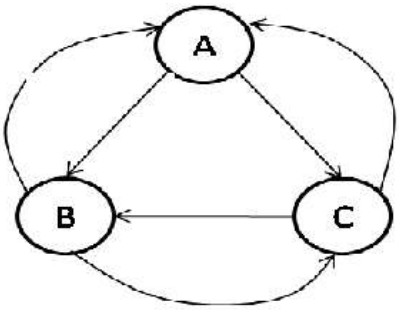
Connected Graph



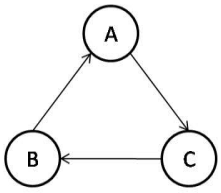
Not Connected

Strongly Connected:

- If there is a path from every vertex to every other vertex in a directed graph then it is said to be strongly connected graph. That is there should be path from any node to any other node.



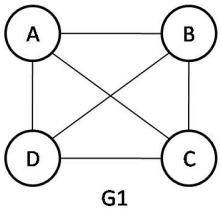
Strongly
Connected Graph



Weakly
Connected Graph

Complete Graph:

- A Complete graph is a graph in which there is an edge between every pair of vertices (or) every pair of nodes. A complete directed graph is a strongly connected graph and it is shown below.



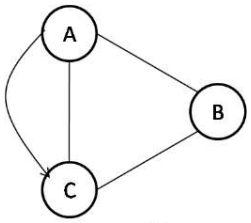
G1

Maximum number of edges in a graph:

- In an undirected graph, there can be n(n-1)/2 maximum number of edges (or) vertices.In digraph, there can be n(n-1) maximum edges , where n is the total number of nodes (or) vertices in a graph.

Multiple edge:

- If there are more than one edge in between a pair of nodes, then they are known as multiple edges.

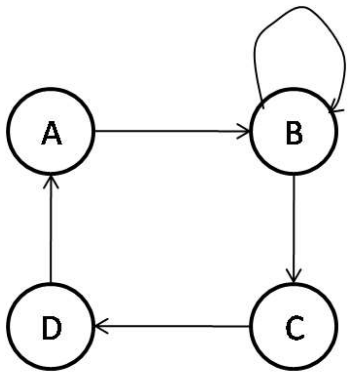


G3

- Here there are multiple edges between A and C.

Loop:

- An edge will be called loop (or) self edge if it starts and ends on the same node. (i.e) if a node has a vertex to itself, it is called loop.

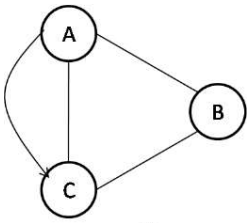


G4

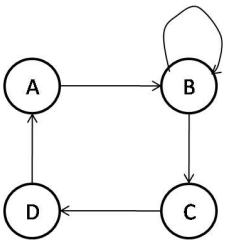
- Here, graph has a loop at node B.

Multigraph:

- A graph which has loop or multiple edges can be described as multigraph and it is shown below,



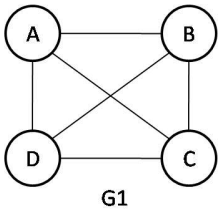
G3



G4

Regular Graph:

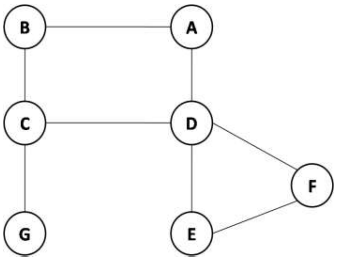
- A graph is regular if every node is adjacent to the same number of nodes. This graph is regular , since every node is adjacent to 3 nodes.

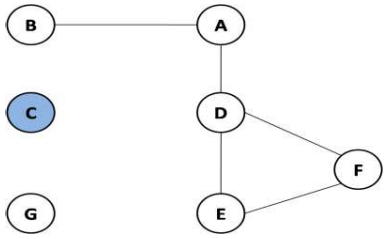


G1

Articulation Point:

- If on removing a node from the graph, the graph becomes disconnected , then that node is called the edge Articulation Point.





Bridge:

- If on removing an edge from the graph , the graph becomes disconnected, then that edge is called the bridge.

Biconnected Graph:

- A graph with no articulation points is called a biconnected graph.

Tree:

- An undirected connected graph will be called tree if there is no cycle in it. In tree structure, any node can have many children but it can have only one parent, while in graph structure, any node can have many children and many parents.

Q4) a) What are the different ways to represent graph?(13)

Or

Q4) b) Name the ways of representing the graph .Explain with examples.(13)

ANSWER

REPRESENTATION OF GRAPHS

Graphs are generally represented in the following two ways:

- ✓ ***Adjacency Matrix.***
- ✓ ***Adjacency List.***

ADJACENCY MATRIX

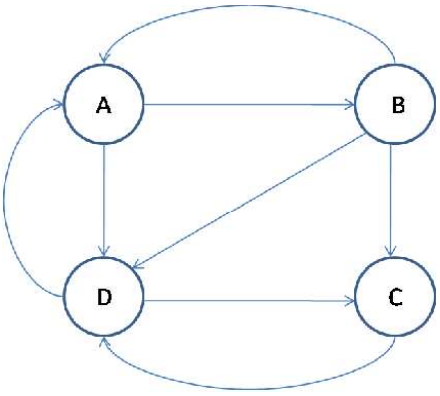
In this representation, the graph is represented using a matrix of size, by total number of vertices. That means a graph with 4 vertices is represented using 4 x 4 matrix.

This matrix is filled with 1 or 0 or weight of the edge. Here, 1 represents that there is a edge from one vertex to next. 0 represents that there is no edge. Weight represents that there is a weight in the edge.

- ➢ Now we are going to form a adjacency matrix for,
  1. ***Adjacency Matrix for Directed Graph.***
  2. ***Adjacency Matrix for Undirected Graph.***
  3. ***Adjacency Matrix for Weighted Graph.***

Adjacency Matrix For Directed Graph

Let us take the graph, the corresponding adjacency matrix for this graph will be,

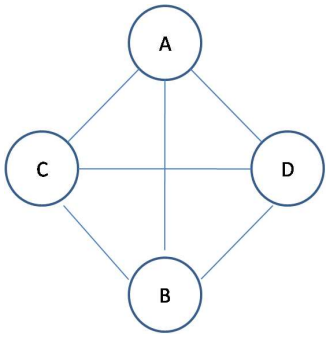


The adjacency matrix is maintained in the array arr[4][4]. Here the entry of matrix

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

Adjacency Matrix For UnDirected Graph

Let us take an undirected graph .The corresponding adjacency matrix for this graph will be,

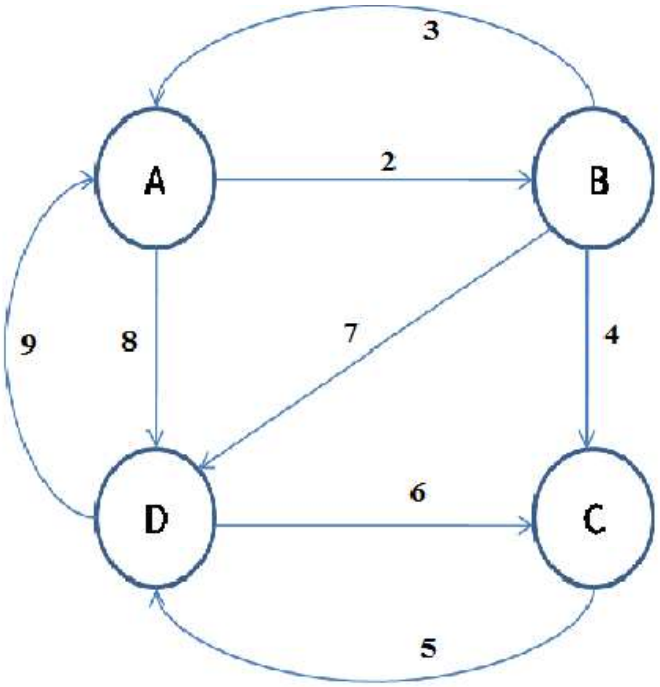


The adjacent matrix for an undirected graph will be a symmetric matrix. This implies that for every I and j. A[i][j] = A[j][i] in an undirected graph.

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

Adjacency Matrix For Weighted Graph

Let us take the Weighted graph .The corresponding adjacency matrix for this graph will be,

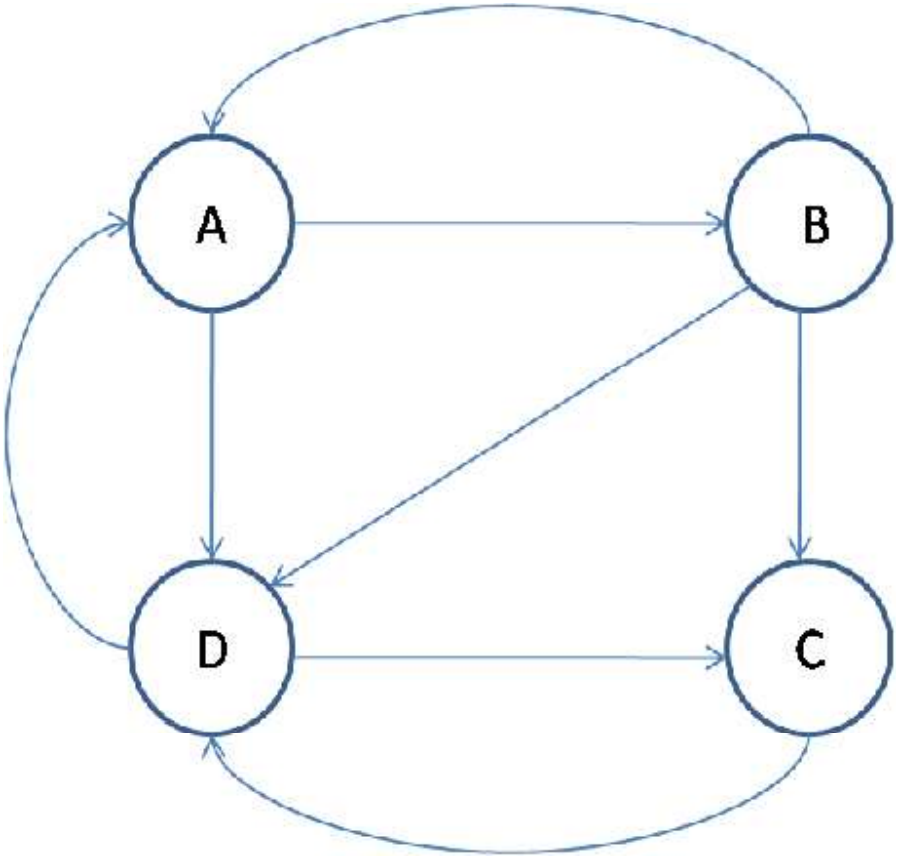


	A	B	C	D
A	0	2	0	8
B	3	0	4	7
C	0	0	0	5
D	9	0	6	0

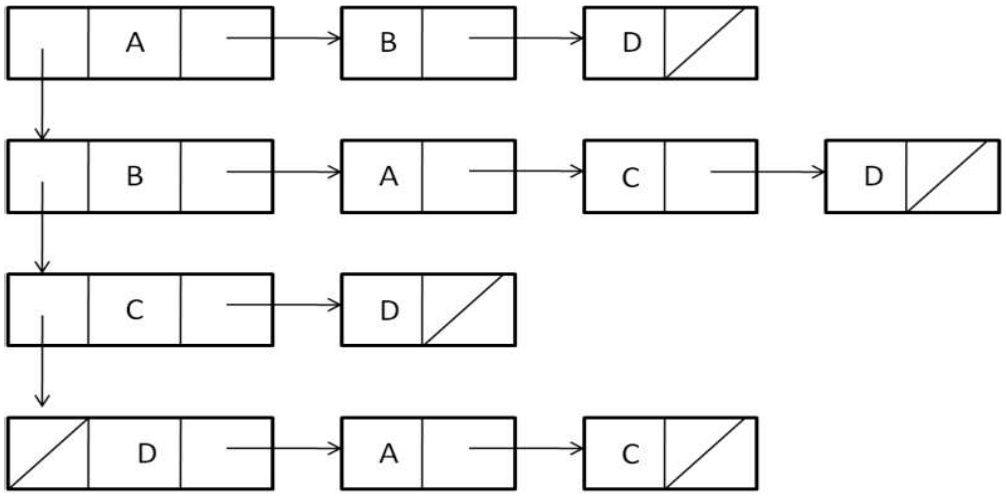
Adjacency List

A graph containing m vertices and n edges can be represented using a linked list, referred to as adjacency list. The number of vertices in a graph forms a singly linked list. Each vertex have a separate linked list, with nodes equal to the number of edges connected from the corresponding vertex.

Adjacency List for Directed Graph

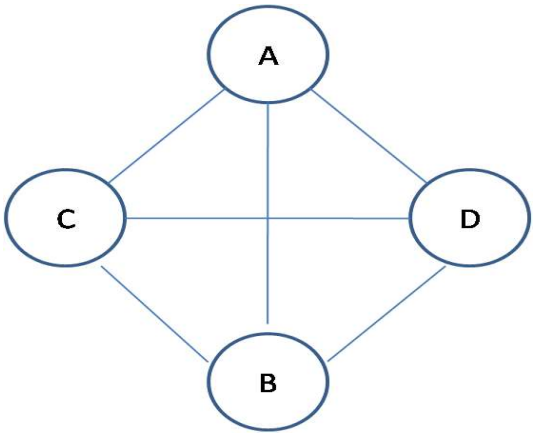


- The adjacency list for the above graph is as follows,

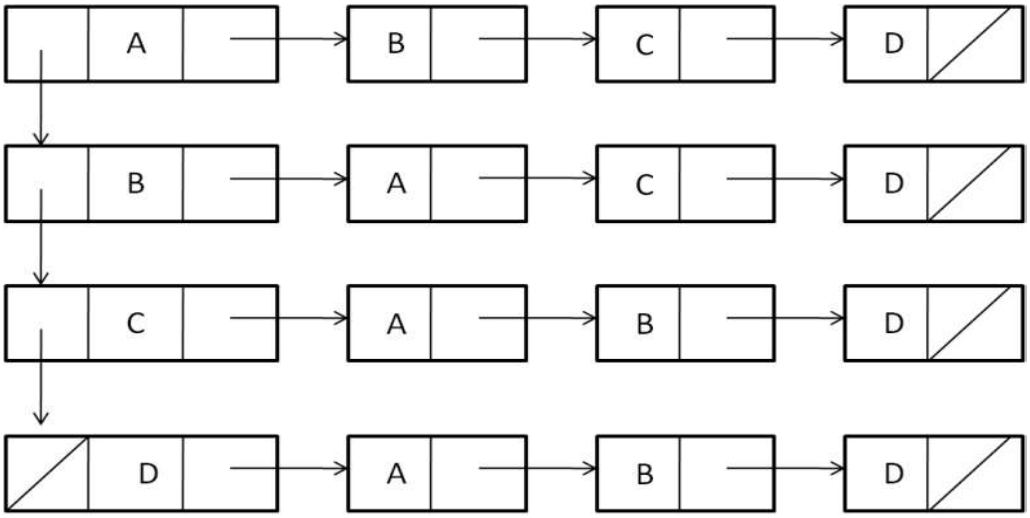


Adjacency List

Adjacency List for Undirected Graph

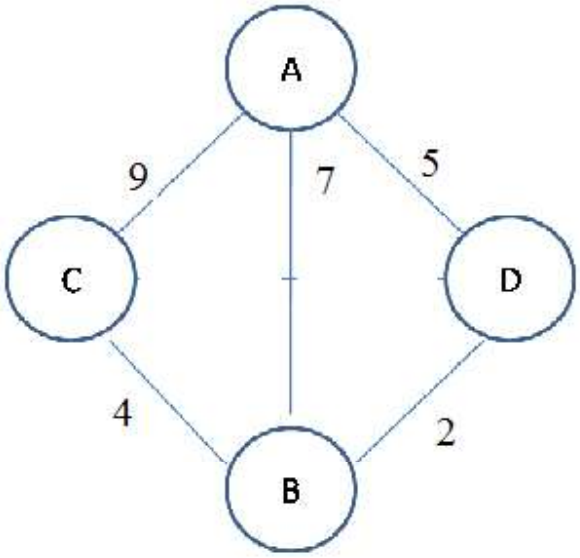


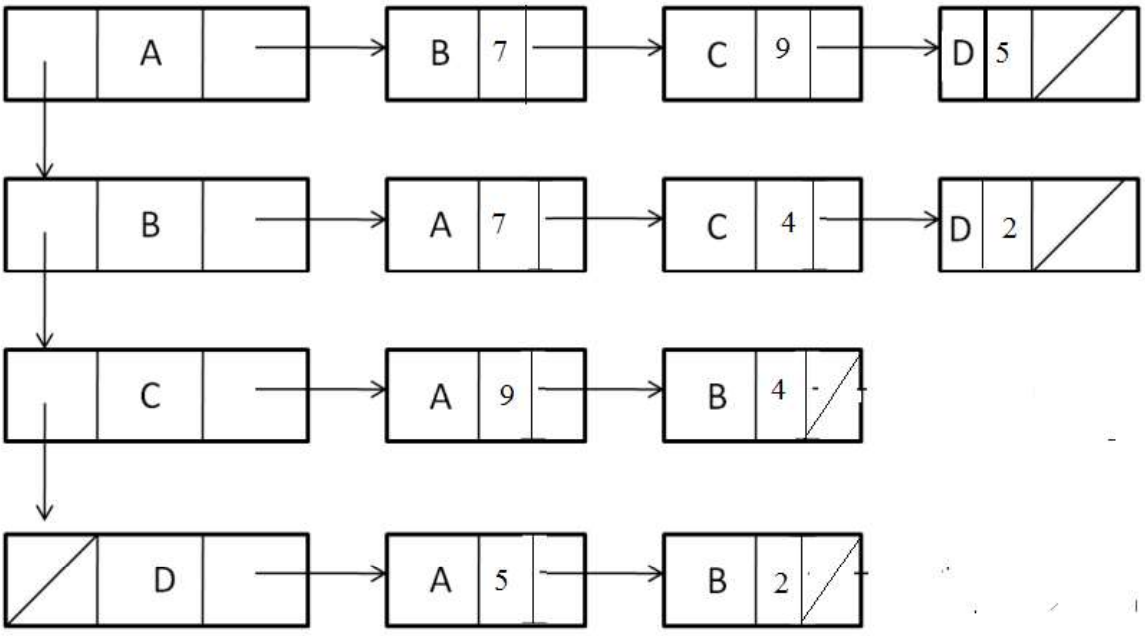
- The adjacency list for the above graph is as follows,



Adjacency List

Adjacency list for weighted graph





Advantages

- Eventhough adjacency list is a difficult way of representing graphs, a graph with large number of nodes will use small amount of memory.
- In adjacency list representation of graph, we will maintain two lists.
- First list will keep track of all the nodes in the graph and second list will maintain a list of adjacent nodes for each node.

Q5 a) Explain Topological sort with an example.(8)

Or

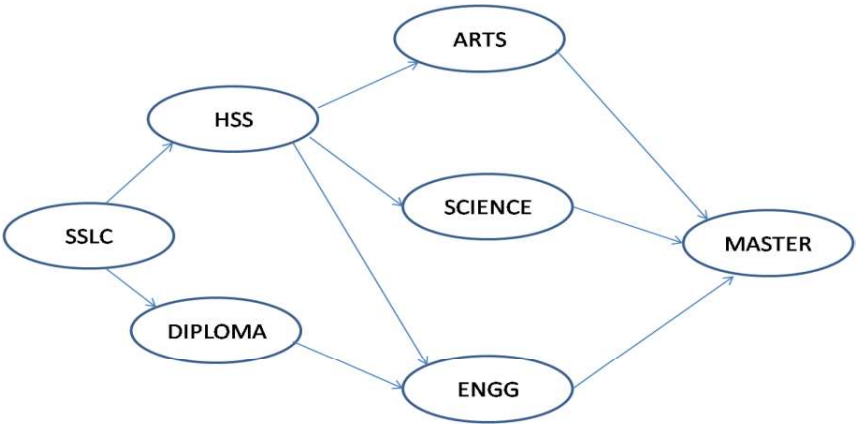
Q5) b) Explain the steps to perform topological sort with queue representation(13)

TOPOLOGICAL SORTING

- Topological sorting is a natural sub problem used on directed acyclic graph. Topological sorting, orders the vertices and edges of a directed acyclic graph in a simple way and can be used to schedule tasks under precedence constraints.
- Let us assume , that we have to do a set of jobs . But certaion jobs have to be performed after completion of other jobs.
- These precedence constraints form a directed acyclic graph and any topological sort defines an order to do these jobs such that each job is performed only after all of its constraints are satisfied.

Example:

- A course prerequistic structure is shown as a graph in this example.A directed edge between (SSLC, HSS) indicates that course SSLC must be completed before attempting to course HSS.



- The legal Toplogical Orderings are:

  - ✓ ***SSLC, HSS, Arts, Master***
  - ✓ ***SSLC, HSS, Science, Master***
  - ✓ ***SSLC,HSS, Engineering, Master***
  - ✓ ***SSLC, Diploma, Engineering, Master.***

Definition:

- A Topological sort is an ordering of vertices in a Directed Acyclic Graph (DAG) such that if there is a path from Vi to Vj , then Vj appears after Vi in the ordering

Pseudocode to perform topological sort with queue representation

```
Void TopSort(Graph G)
{
        Queue A;
        Int Counter = 0;
        Vertex V,W;
        Q = CreateQueue(Num Vertex);
        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 a cycle");
        DisposeQueue(Q);
}
```

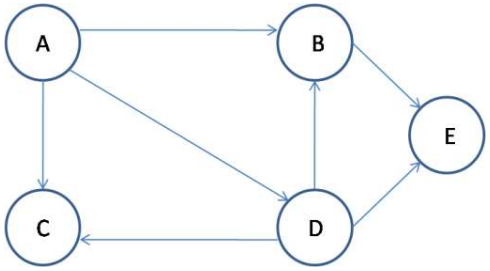
- The sequence of steps to perform the topological sort of the DAG with queue representation is given below:
    1. Assume that the graph is read from an adjacency list and the indegree are computed and stored in an array.
    2. Queue is created for the number of vertices [(i.e) NumVertex] of the given graph.
    3. Empty the queue.
    4. The vertices with indegree 0 are inserted into the queue. [Enqueue (V,Q)].
    5. While the queue is not empty, then:

a) Delete the vertex from the queue. Dequeue (Q).
b) Assign a topological number.
c) Decrement the adjacent vertices of vertex V and check its indegree also. If the indegree is zero, then insert that vertex w into the queue.

6. Repeat step 5 until the queue becomes empty.
7. Print the topological order from TopNum array.

Example to perform topological sort with queue representation

- Perform simple topological sort for the following graph,



Solution:

Execution Table Explanation

Step 1:

- Initially, indegrees are;

Indegree(A) = 0
Indegree(B) = 1
Indegree(C) = 1
Indegree(D) = 3
Indegree(E) = 2

- Adjacent nodes are,

Adjacent nodes for A=B,C,D
Adjacent nodes for B=D,E
Adjacent nodes for C=D
Adjacent nodes for D=E
Adjacent nodes for E=Nil

Step 2:

- These indegrees are entered in column 1. The indegree of A is 0 in column 1 marked inside box is enqueued and then dequeued. The adjacent nodes of A are B,C and D.
- Their indegrees are decremented by 1 and entetred in column 2. Then in the 2nd column, the indegree of B and C is zero. So it is enqueued (both B & C). But we can dequeue only 1(i.e) B.
- In the third column no one is zero but queue has only node C. So dequeue that. Then in the 2nd column, the nodes adjacent to B are A, D & E. So decrement their indegree values. In the 3rd column, the nodes adjacent to C are A and D.

- So decrement their indegree values(X). In the 4th column, the indegree of D is zero. So dequeue it after enqueue.
- The nodes adjacent to D are B & E. So decrement their indegree in 5th column. In the 1st iteration, the initial indegree of all the vertices are shown.
- Vertex A has indegree 0, therefore that is inserted and deleted from the queue. In the 2nd iteration, both B and C have indegree 0, but the queue is not empty, therefore C is deleted from the queue.
- Similarly 4th & 5th iteration are carried out and all the vertices with indegree 0, indicates the end of the routine. The vertices are printed in the dequeued order, derives a topological order. (i.e.) **A B C D E**.

Execution Table

Vertex	Indegree before Dequeue				
	1	**2**	**3**	**4**	**5**
A	0	0	0	0	0
B	1	0	0	0	0
C	1	0	0	0	0
D	3	2	1	0	0
E	2	2	1	1	0
Enqueue	A	B	C	D	E
Dequeue	A	B	C	D	E

Analysis of simple topological sort

In this algorithm, FindNewVertexOfIndegreeZero is a simple sequential scan of indegrgee array, each call to it takes O(V) time. Since there are |V| such calls, the running time of the algorithm is $O(|V|^2)$.

Drawback of this simple topological sort pseudocode:

- If the graph is sparse, only a few vertices have their indegree to be updated during iteration. But this algorithm carries a sequential search of all the vertices.
- To overcome this, a queue to be maintained for vertices with indegree 0 and not yet processes.

Q6) a) Explain the different ways of graph traversal.(13)

Or

Q6) b) Explain about

i) Breadth first Traversal

ii) Depth first Traversal

ANSWER

GRAPH TRAVERSAL

Traversing a graph is visiting each of its vertices in a systematic manner.

Condition is,when traversing a graph, we must be careful to avoid going round in circles We do this by marking all vertices which have already been visited

- 2 traversals are:

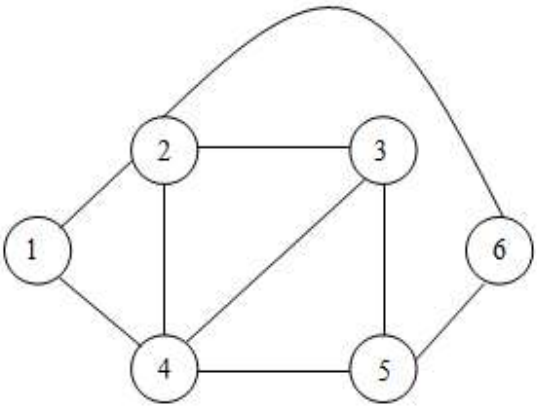
iii) Breadth first Traversal

iv) Depth first Traversal

1) BREADTH FIRST TRAVERSAL

- A queue data structure is used to trace the operations of BFS
- The queue is initialized with the starting vertex which is marked as visited
- On each iteration, all unvisited vertices that are adjacent to the starting vertex are added to the queue, after that the front vertex is removed from the queue and marked as visited
- The algorithm terminates when the queue becomes empty
- Steps are
  - ***First, we examine the starting vertex V***
  - ***Then we examine all the neighbors of V***
  - ***then we examine all the neighbors of neighbors of V and so on***

Example1

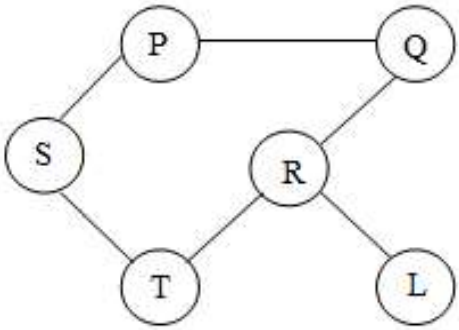


- Let us start with vertex 1

Adjacent nodes of {1} = {2,4}
Adjacent nodes of {2,4} = {3,6,5}
(Only select the nodes that are not visited)
Adjacent nodes of {3,6,5} = Nil

- ***Thus, the BFS of the graph is 1 – 2 – 4 – 3 – 6 – 5***

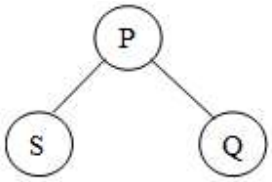
Example2



Step 1: Choose any node as the starting vertex consider "P" and mark it as visited.

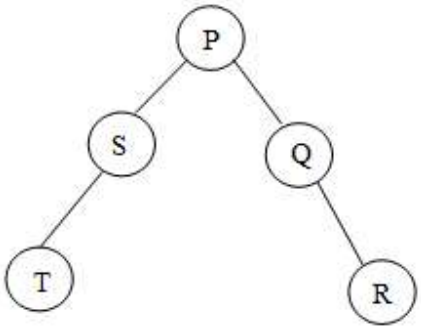
P

Step 2: Go to adjacent nodes of P which are not visited already and visit (ie) S and Q are visited from P.



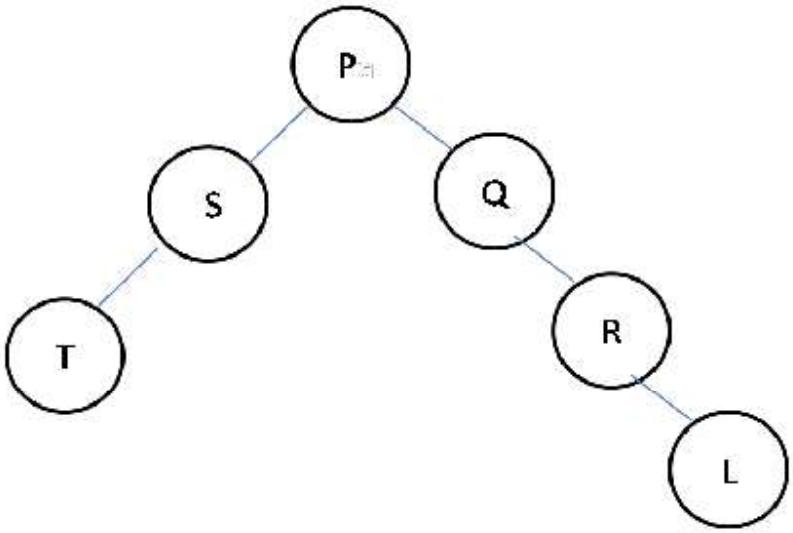
P-S-Q

Step 3: Go to adjacent nodes of S and Q which are not visited already and visit (ie) T is visited from S and R is visited from Q



P-S-Q-T-R

Step 4: Go to adjacent nodes of T and R which are not visited already and visit (ie) L is visited from R.



P-S-Q-T-R-L

Applications of BFT:

- To check whether the graph is connected or not
- To determine whether a graph is cycle
- The unweighted graph, BFS find the shortest path

Algorithm to perform BFS

```
BFS(G,V)
        Queue Q = create(n);
        enqueue (Q,V);
        while Q is not empty
                v = front(Q);
                dequeue(Q);
                For each vertex W adjacent to V
                        enqueue(Q,W);
                        parent[W] = v;
```

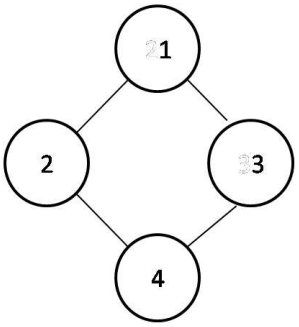
2) DEPTH FIRST TRAVERSAL

- Depth First Search of a graph starts at any vertex (i.e.) adjacent to the current one and by marking it as visited
- This process continues until a dead end (a vertex with no adjacent unvisited vertices is called dead end) is encountered
- At a dead end the algorithm backtrack to the previous vertex it came from and tries to continue visiting unvisited vertex from there
  The algorithm stops after backing up to the starting vertex

Data structure used:

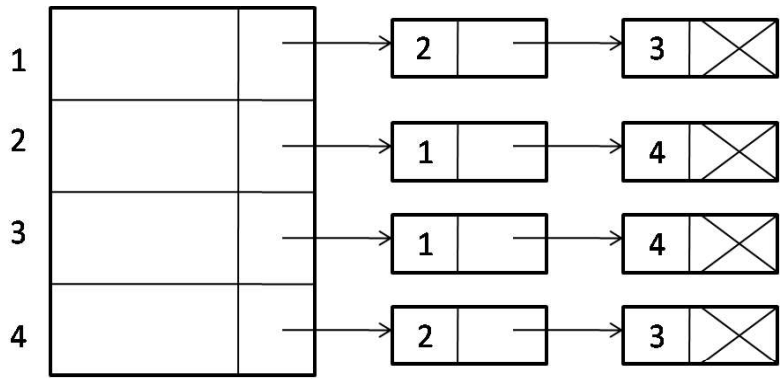
- A stack data structure is used to trace the operation of DFS
- When we visit the unvisited vertex for the first time, then push a vertex onto a stack
- Pop a vertex from a stack when it becomes a dead end

Example for DFS

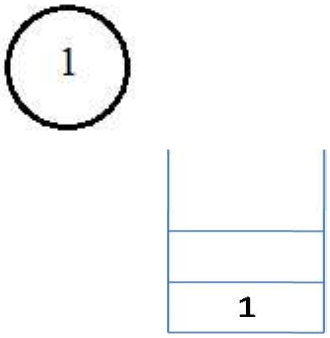


Solution:

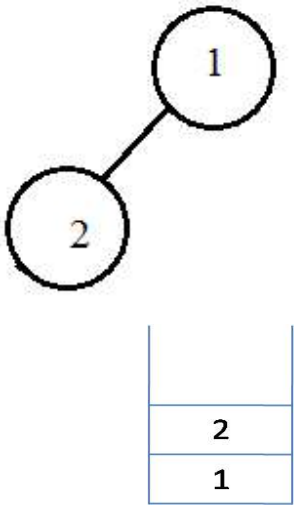
1. Using the adjacency list of the graph find a node adjacent to the search node that has not been visited yet.



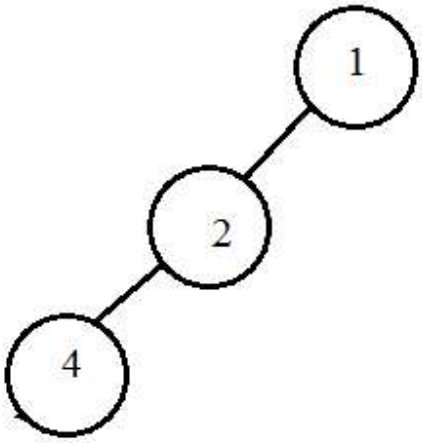
2. Start from vertex 1 and put it into the stack



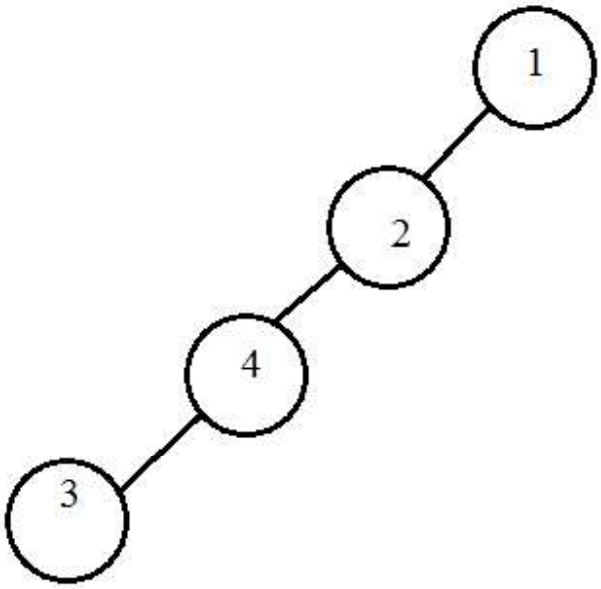
3. One of the neighbors of vertex “1” is “2”. Put it in stack

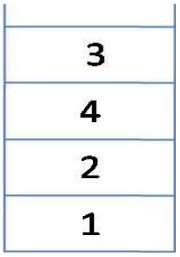


4. Neighbors of “2” are “1” and “4”. “1” is already visited. so visit “4”



5. Neighbors of “4” are “2” & “3” . but “2” is already visited. So visit “3”



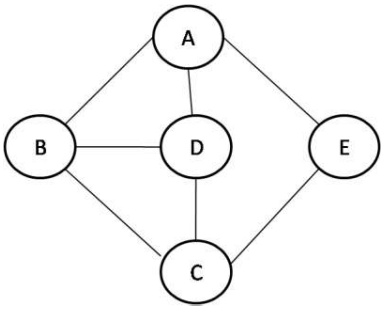


6. So DFS search is 1 – 2 – 4 – 3

Example2 for DFT

Undirected Graph

A undirected graph is "connected" if and only if a depth first search starting from any node visits every node

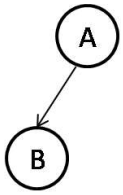


	A	B	C	D	E
A	0	1	0	1	1
B	1	0	1	1	0
C	0	1	0	1	1
D	1	1	1	0	0
E	1	0	1	0	0

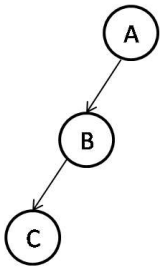
Adjacency Matrix

1. DFS starts at vertex "A" and by marking it as visited

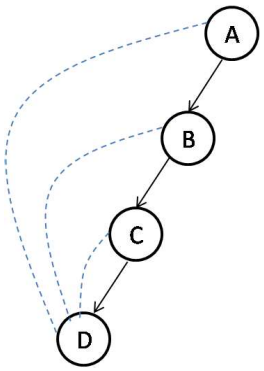
2. To vertex "A", "B", "D", "C" are adjacent vertices. But "B" is encountered first on a left to right



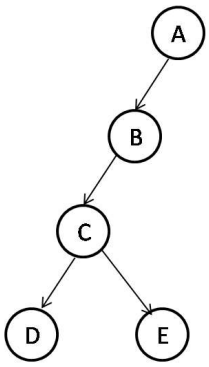
3. Now to B, vertices "C" & "D" are adjacent. First visit "C"



4. To "C" vertices "D" & "E" are unvisited adjacent nodes. First visit "D"



5. From "D" there are no any unvisited adjacent vertices. So backtrack to "C" and visit vertex "E"



- Since all the vertices starting from "A" are visited, the above graph is said to be connected
- If the graph is not connected, then processing all nodes requires several calls to DFS and each generates a tree. This entire collection is a depth first spanning tree

Algorithm for DFS

```
void DFS(Vertex V)
{
        visited[V] = true;
        For each W adjacent to V
                if(!visited[W])
                        DFS(W);
}
```

Applications of Depth First Search

1. To check whether the undirected graph is connected or not
2. To check whether connected undirected graph is Biconnected or not
3. To check the Acyclicity of the directed graph

Q7) a) Define Biconnectivity and Biconnected graph.(4)

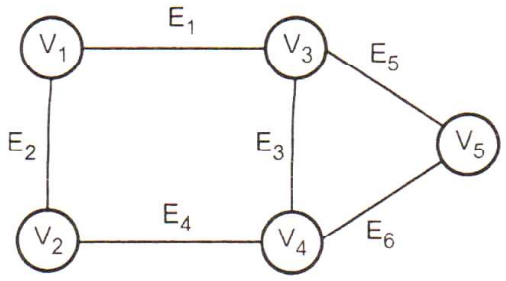
Or

Q7) b) Explain the biconnectivity property with an example.(8)

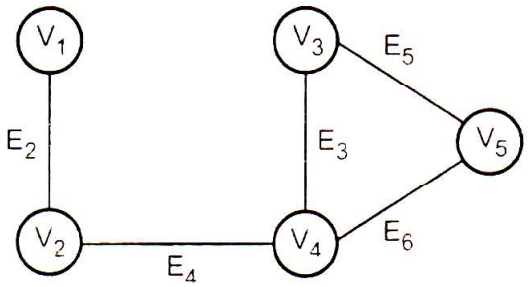
ANSWER

BICONNECTIVITY

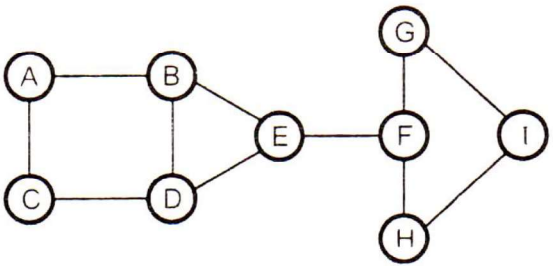
- Biconnected graphs are the graphs which can not be broken into two disconnected pieces (graphs) by connecting single edge.
- **For example :**



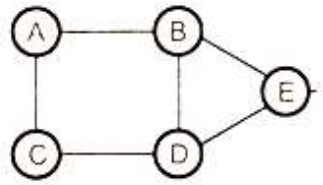
- From the above graph even if we remove any single edge the graph does not disconnected.
- For example if we remove an edge E_1 the graph does not become disconnected.

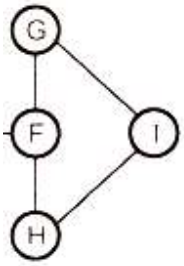


- We do not get two disconnected components of graph. Same is the case with any other edge in the given graph.
- But the following graph does not possess the property of Biconnectivity.



- In above graph if we remove an edge **E-F then** we will get two distinct graphs.





Properties of Biconnected Graph

1. There are **two disjoint paths** between any two vertices.
2. There exists **simple** cycle between two vertices.
3. There should not be any **cut vertex** (Cut vertex is a vertex which if we remove then the graph becomes disconnected.)

Q8) a) Define Cut vertex

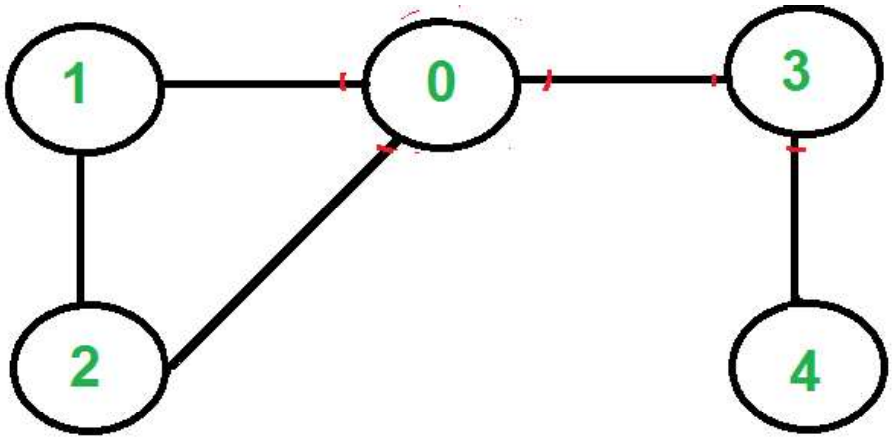
Or

Q8) b) Define articulation point with an example.(4)

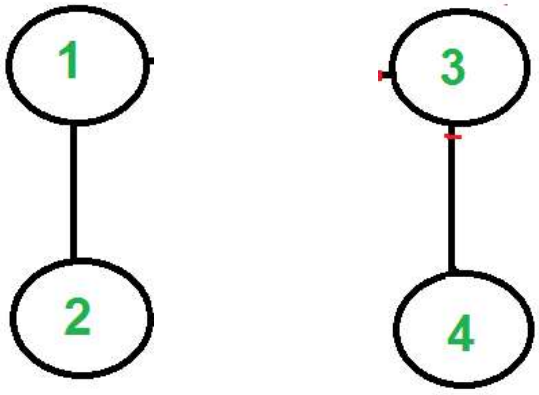
ANSWER

CUT VERTEX OR ARTICULATION POINT

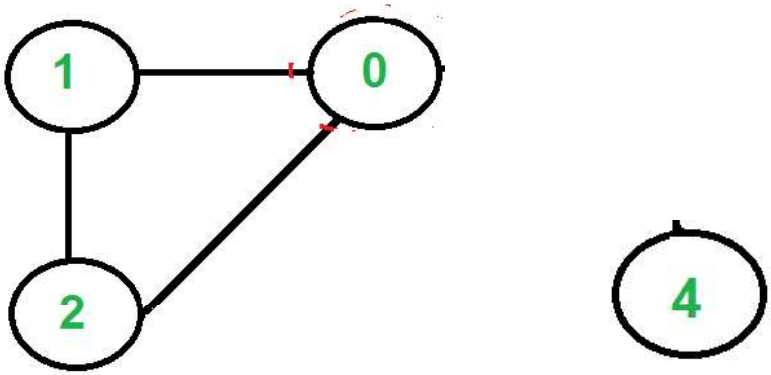
- A vertex in an undirected connected graph is an articulation point (or cut vertex) if removing it disconnects the graph.
- Articulation points represent vulnerabilities in a connected network – single points whose failure would split the network into 2 or more disconnected components.
- They are useful for designing reliable networks.For a disconnected undirected graph, an articulation point is a vertex removing which increases number of connected components.
- Example



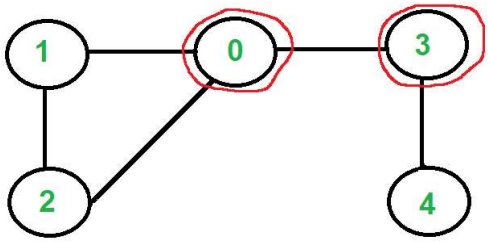
Solution 1: From this graph if we remove the node 0 then graph will be disconnected as follows



Solution 2: From this graph if we remove the node 3 then graph will be disconnected as follows



Final graph with articulation points encircled with red color.



Articulation points are 0 and 3

Q9) a) Give an account of Euler circuits in the applications of graph.(13)

Or

Q9) b) Give a short notes on Euler circuits.(6)

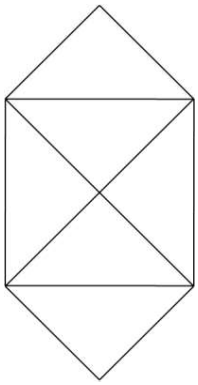
EULER CIRCUITS

Definition

An Euler circuit is a circuit that uses every edge of a graph exactly once. An Euler circuit starts and ends at the same vertex.

This problem was solved in 1736 by Euler. This is called as Euler Path(Euler tour) or Euler circuit

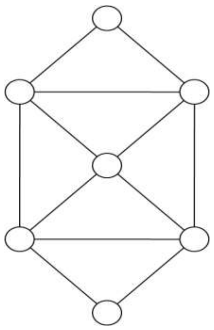
- Consider the following diagram,



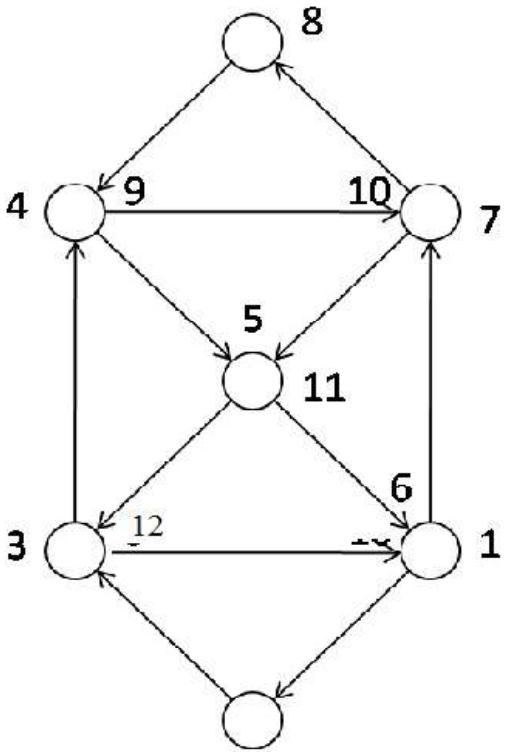
- Reconstruct this diagram using pen. 3 conditions are there
    1. Drawing each line exactly once
    2. The pen may not be lifted from the paper while the drawing is being performed
    3. Make the pen finish at the same point at which it started

Solution

- Convert this diagram into graph theory problem by assigning a vertex to each intersection. This is shown below



Ans:

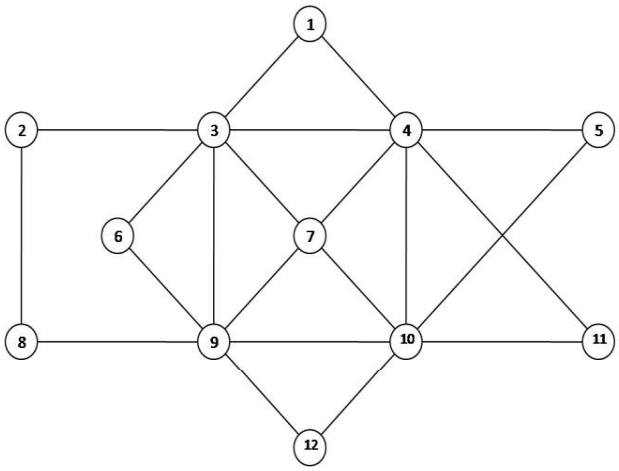


Euler path is 1-2-3-4-5-6-7-8-9-10-11-12-1

Steps to find Euler Circuit

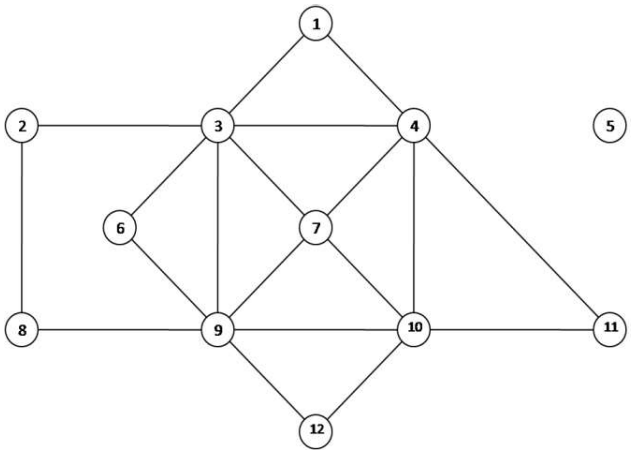
1. We must visit a portion of the graph and return to the starting point DFS. This produce one circuit
2. If all edges coming out of the start vertex have been used up, then the part of the graph is untraversed
3. From the untraversed vertices fix the starting point and traverse using another DFS. this produce another circuit
4. Repeat step 2 and 3 until all the vertices are visited

- Consider the following graph



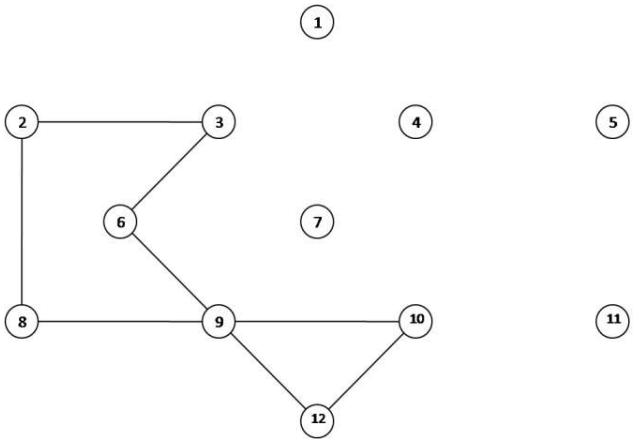
Step1:

Consider, start the traversal at vertex 5, and traverse 5,4,10,5 so we get,



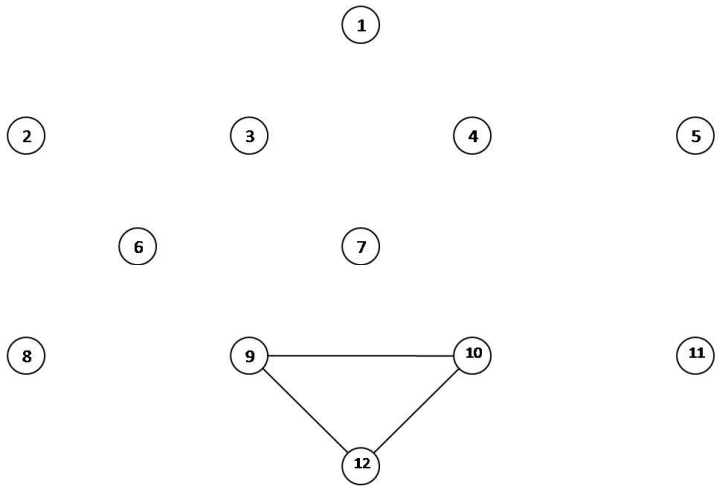
Step 2:

- Then continue from vertex 4. A DFS might come up with the path 4,1,3,7,4,10,7,9,3,4. If we splice this path into the previous path of 5,4,10,5 then we get a new path of 5,4,1,3,7,4,11,10,7,9,3,4,10,5



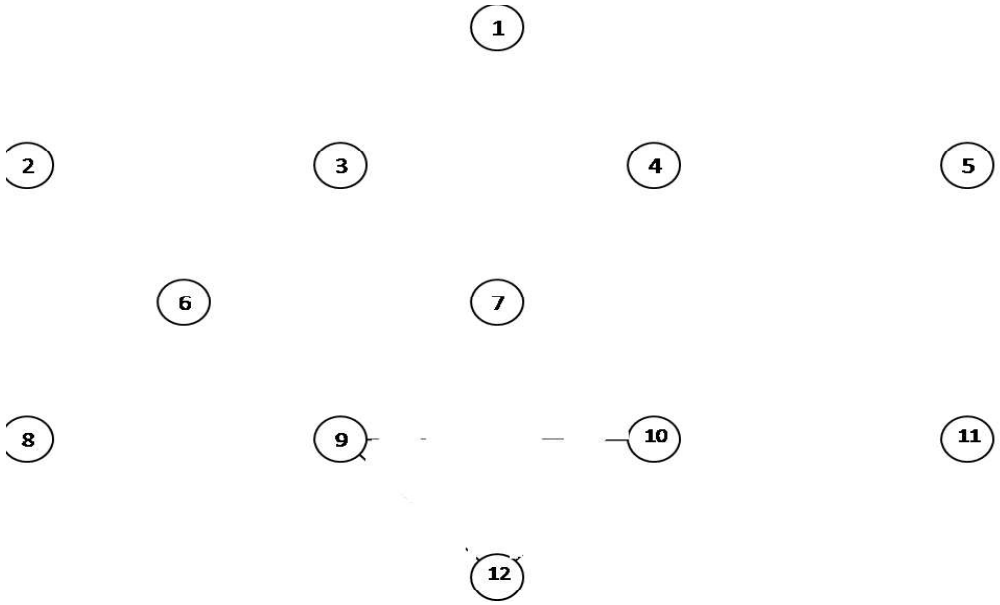
Step 3:

- Then next vertex 3, A possible circuit would be 3,2,8,9,6,3
- We got the following graph using 5,4,1,3,2,8,9,6,3,7,4,11,10,7,9,3,4,10,5



Step 4:

- Then next vertex 9, A possible circuit would be 9,12,10,9
- Finally we got 5,4,1,3,2,8,9,12,10,9,6,3,7,4,11,10,7,9,3,4,10,5



Step 5: As all the edges are traversed , the algorithm terminates with an euler circuit DFS is 5-4-1-3-2-8-9-12-10-9-6-3-7-4-11-10-7-9-3-4-10-5.

Q10) a) Write notes on applications of graph.(8)

Or

Q10) b) Name some areas where we can use graph(6)

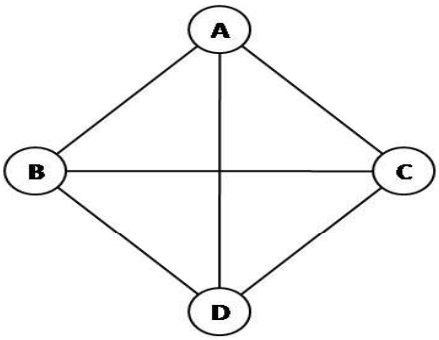
ANSWER

APPLICATIONS OF GRAPH

The problems using graph representation are useful in many fields for different operations. Some examples are,

1) **AIRLINE ROUTE MAP- Undirected Graph**

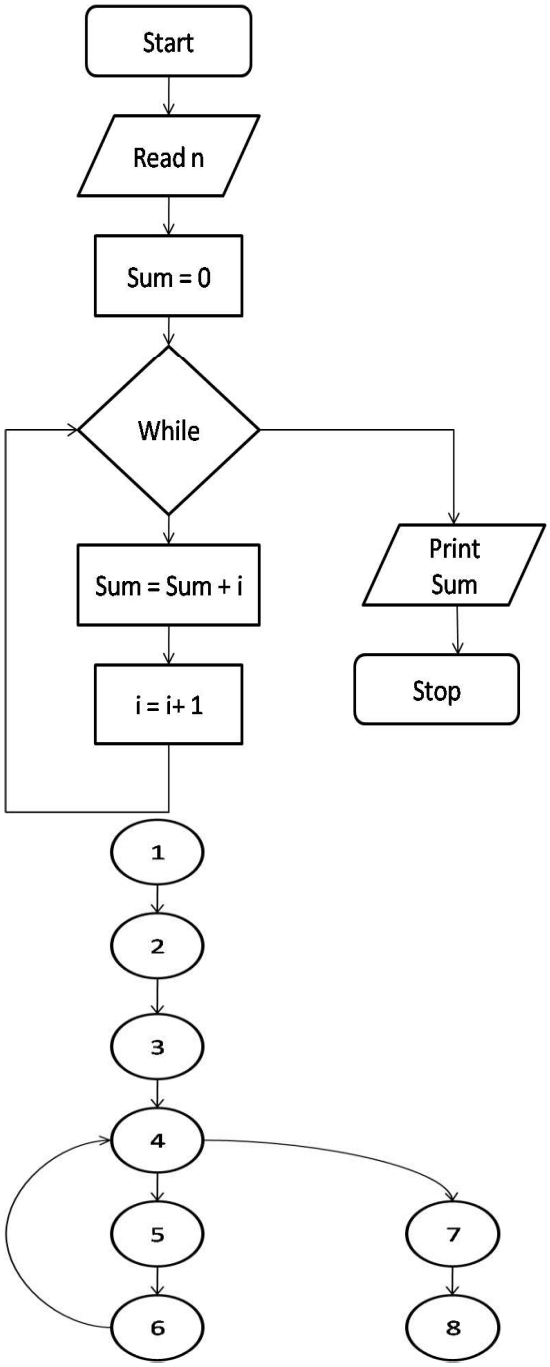
The points are the cities, a line connects two cities if and only if there is a non stop flight (and also to and fro) between the cities in both directions. For example,



- A graph between for cities [A, B, C, D }

2) FLOW CHARTS – Directed Graph

The points are the flow chart boxes, the connecting arrows(edges) are the flowchart arrows.

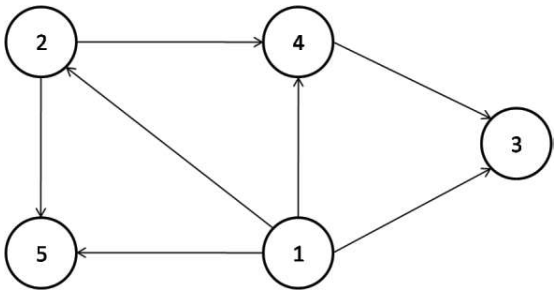


In software engineering and all they will be using these flowcharts which is in software projects and draw the corresponding directed graphs to check whether there is a mapping from one component to another component.

3) A BINARY RELATION- Directed Graph

To define a binary relation R on the set S = { 1,2,3,4,5} consisting of ordered points (x, y) (i.e) x R y for (x, y) € R.

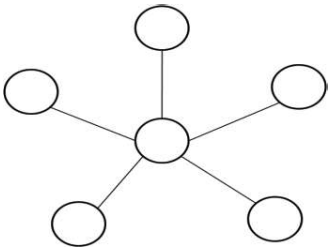
- In the given digraph, the points are the elements of S and there is an arrow from x to y if and only if x R y.
- Notice that R is transitive: If x R y and y R z both hold, then x R z also holds.

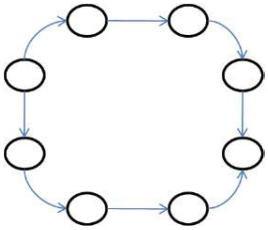


- The transitive relation R for the above graph is,
  1 R 2, and 2 R 5 → 1 R 5, 1 R 2 and 2 R 4 → 1 R 4, 1 R 4 and 4 R 3 → 1 R 3.

4) COMPUTER NTEWORKS

The points are computers. The lines (undirected graph) or arrows (directed graph) are the communication links.





5) AN ELECTRICAL CIRCUIT

The points could be diodes, transistors, capacitors, switches and so on. Two points are connected by a line9 edge) if there is a wire connecting them.