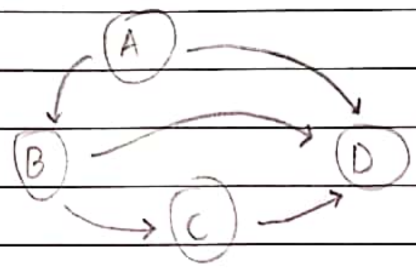
27/04/2020

Graphs - Introduction

① Graph is a collection of objects called as Vertices and together with a relationship between them called as Edges.

② Each edge in the graph join two vertices.

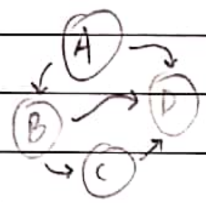
Graph (G) = { V, E } → Graph G is set of vertices 'V' & edges 'E'.



③ Vertices (V) = { A, B, C, D }

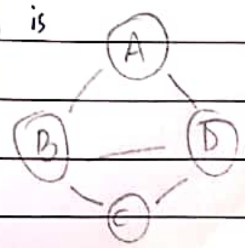
④ Edges (E) = { A → B, A → D, B → C, B → D, C → D }

① **Directed Edge** : An edge (u,v) is directed if pair (u,v) is ordered, with u preceding v.
Edge is oriented or Direction



Directed Graph or **DiGraph**

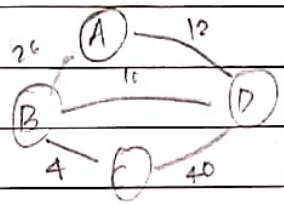
② **Undirected Edge**: An edge (u,v) is undirected if pair (u,v) is not ordered.
Edge has no orientation.

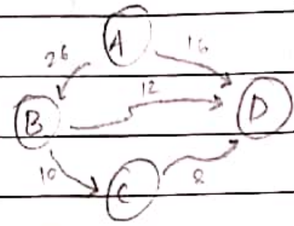


Undirected graph

27/05/2020

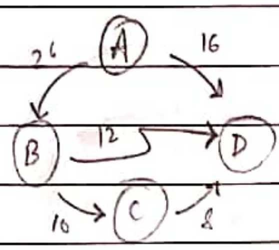
③ Weighted Edge : Cost or weight is assigned to each edge (u,v).





Weighted Undirected Graph Weighted directed Graph

—— x —— x ——



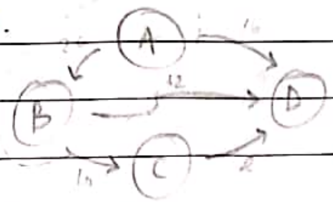
① End Vertices – Two vertices joined by an edge.
Eg: Vertices (A) & (B) joined by an edge are end vertices.

② Adjacent Vertices – Two vertices are adjecent if there is an edge b/w them.
Eg: There is an edge b/w (A) & (B) ∴ they are adjacent vertices.

③ Incident Edge – If vertex is one of the end points.
Eg: from vertex (A) to (B) is incident from vertex (A) to vertex (B)

27/08/2020

1. **Outgoing Edge** : origin is the vertex.

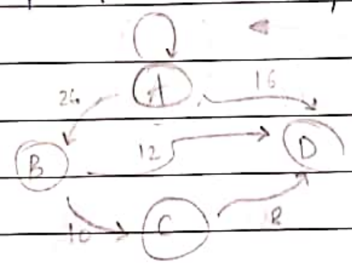


2. **Incoming Edge** : distination is the vertex.

Eg: let's take edge from vertex (A) to (B). So (A) is the outgoing vertex & (B) is the incoming vertex.

——— x ——— x ———

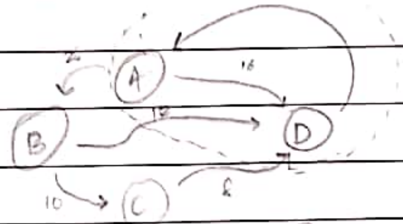
1. **Self loop** : if two end points are same.



Eg: (A) is a self loop.

——— x ——— x ———

1. **Parallel Edges** : Edge from u to v (u,v) as well as an edge from v to u (v,u)

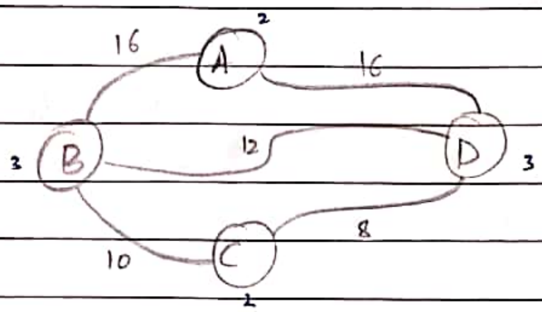


There is an edge between (A) → (D) and also (D) → (A)

∴ These are parallel Edges.

27/01/2020

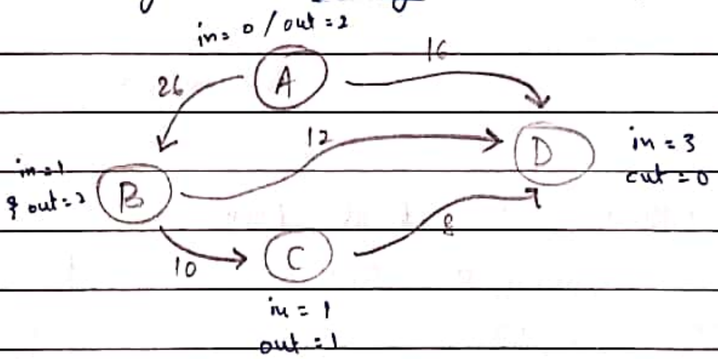
- Degree of Vertex = deg(v) : num of edges from a vertex.



→ for directed graphs we have:-

(i) In-degree — indeg(v) : no. of incoming edges

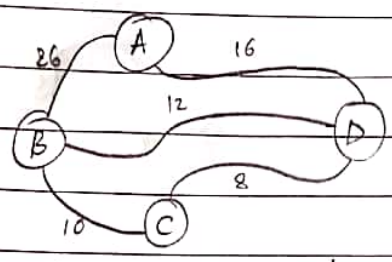
(ii) Out-degree — outdeg(v) : no. of outgoing edges



27/08/2020

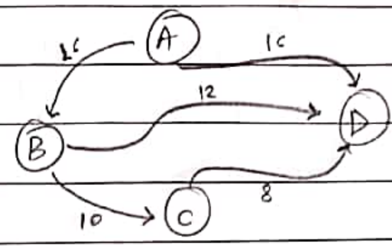
- **Path :**

Sequence of edges starting at one vertex and ending at another vertex.



Weighted Undirected Graph

paths ⇒ A-B-C, AB-C-D, A-B, A-D, A-D-C, A-D-C-B etc



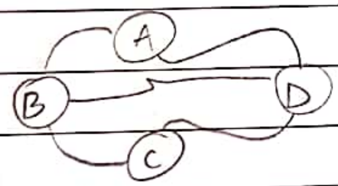
Weighted directed Graph

A - B, A-B-C, A-B-C-D, A-D etc.

(~~can only to direction wise~~) we have to follow orientation or the direction to create path.

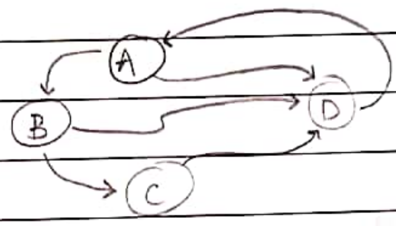
- **Cycle :**

path that starts and end at same vertex.



→ A-B-C-D-A,

→ A-B-D-A .. etc



→ A-B-C-D-A,

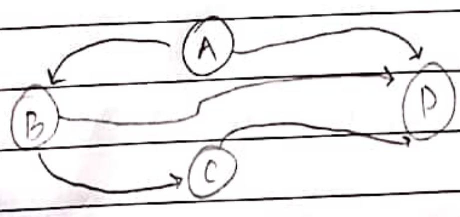
→ A-B-D-A,

→ B-D-A-B,

→ B-C-D-A-B

- **Directed Acyclic Graph:**

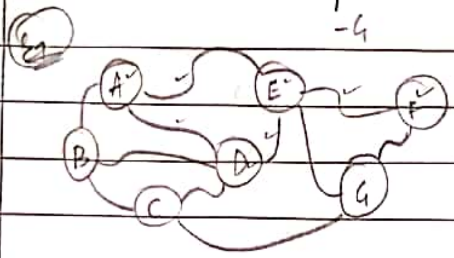
when there are no cycles in a directed graph.



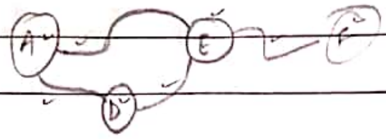
27/08/2020

- **Subgraph** : whose vertices and edges are subsets of vertices and edges of another graph

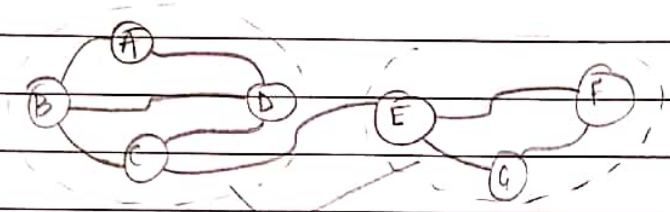
Eg. - G



subgraph of G - H



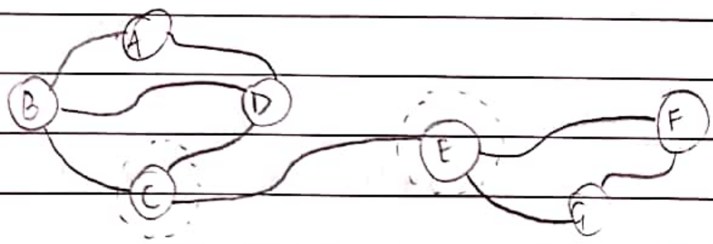
- **Connected Components** : connected subgraphs are known as connected components.



these two are subgraphs which are connected through edge between vertices C & E.

aka 'cut of a graph'

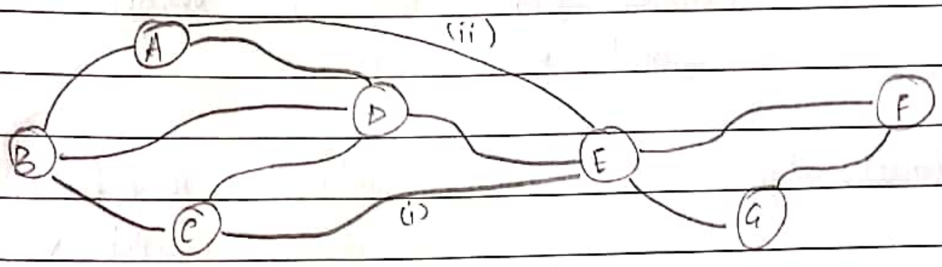
- **Articulation point**: Vertex whose removal results in connected components.



"~~poin~~ vertices C & E ~~separately~~ are 2 separate articulatn points.

27/08/2020

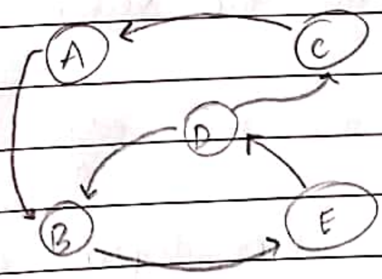
- **Bi-connected components** : components connected by two edges.



eg: Removal of either C - E edge or A - E edge won't result in graph splitting up in two components.

∴ Since we have two edges (i) & (ii) this graph is known to have Bi-connected components.

- **Strongly Connected Graph** : all the vertices are reachable from any vertex.



→ In this graph we can reach all the other vertices.

A graph is a representatn of relationships that exists b/w objects. A graph is a collection of vertices and edges.

27/08/2020

- ## Graph – Abstract Data Types (ADT)

1. create(n) : Creates graph with n vertices and no edges.
2. insert_edge (u, v, w=1) : creates edge from u to v, storing weight w (by default 1)
3. remove_edge (u, v) : delete edge from u to v.
4. exist_edge (u, v) : return true if edge exists between u and v, else false.
5. vertex_count () : returns number of vertices in the graph.
6. edge_count () : returns no. of edges in the graph
7. vertices () : returns all the vertices of the graph.
8. edges () : returns all the edges of the graph.
9. degree(u) : returns the degree of the vertex u.
10. indegree (u) : returns the indegree of the vertex u.
11. outdegree (u) : returns the outdegree of the vertex u.

27/08/2020

- Graph – Representation

A graph can be represented using diff. data structures.

1. Edge List: Maintains list of all edges.

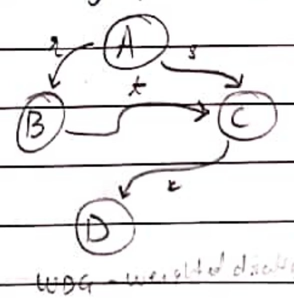
2. Adjacency List: For each vertex, separate list of edges is maintained.

3. Adjacency Matrix: Maintains a matrix of vertices, where each cell stores the reference to the edges.

1. Edge List

- most simple representation of graph
- but there is no eficient way to find a particular edge or set of edges incident on vertex.

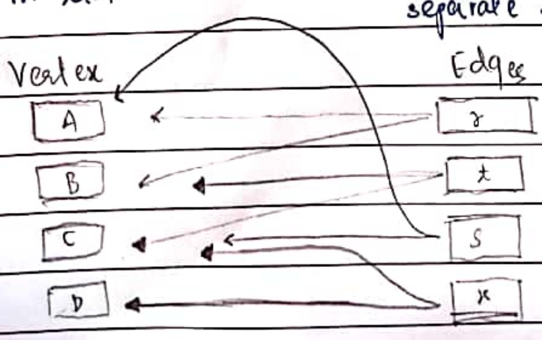
It maintains list of all the edges in the graph.



* a linked list or doubly linked list can be used to represent vertices & edges.

all the vertices are stored in list

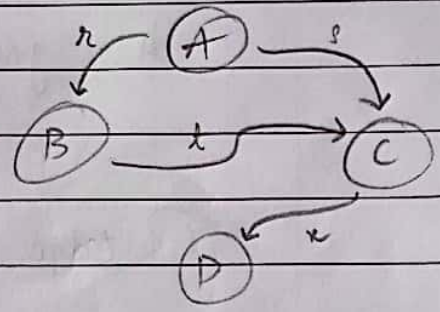
all the edges are also stored in a separate list



27/08/2020

- **Edge list Performance**

Vertices - n (list)   Edges - m (list)



Operations	Time Complexity
insert_edge(u,v,w)	O(1)
remove_edge(u,v)	O(1)
exist_edge(u,v)	O(m)
vertex_edge()	O(1)
edge_count()	O(1)
vertices()	O(n)
edges()	O(m)
degree(u)	O(m)

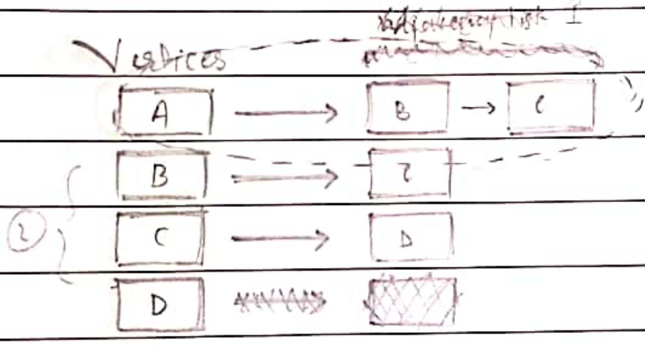
Space Complexity → O(n+m)

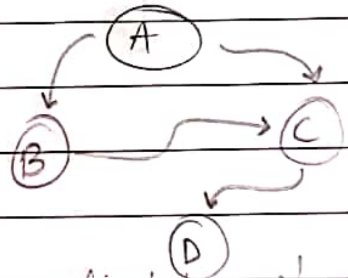
Scanned with CamScanner

27/08/2020

② Adjacency List

For each vertex, separate list of edges is maintained. it basically create separate list that are incident on or to a vertex. This representatⁿ is more efficient ~~bec~~ since all the edges can be easily accesed & we can efficiently find all the edges incident to a vertex.





directed graph.

① Adjacency list of vertex A will contain the reference of vertex B and C as there is an edge from A to them.

② Similarly from B to C & C to D, as there is no outgoing edge from D, there'll be no reference from it. ie adjacency list of vertex D will be empty.

Scanned with CamScanner

27/08/2020

- for weighted directed graph.

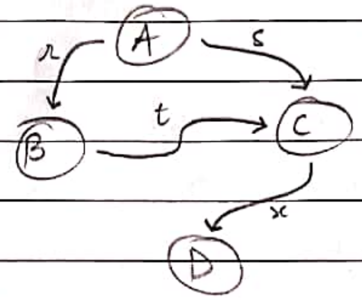
Vertices

A → B|r → C|s

B → C|t

C → D|x

D



(Weighted directed graph)

here in the adjacency list along with the reference of adjacent vertex will also keep the weight of the edge!

- for undirected Graph.

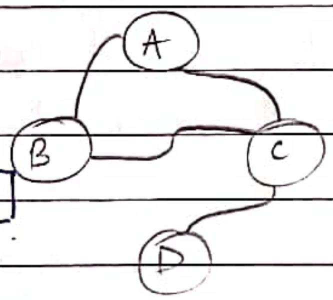
Vertices

A → B → C

B → A → C

C → A → B → D

D → C



(undirected graph)

★ If weight is also present with the undirected graph then along with the adjacent vertices we also have to store the weight of the edge.

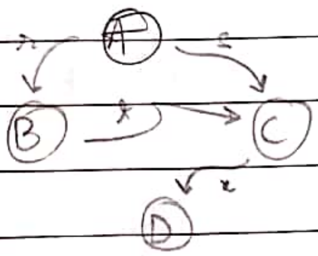
27/05/2020

• Adjacency List Performance

Vertices - n Edges - m

Operations	Adjacency list
insert_edge(u,v,w)	$O(1)$
remove_edge(u,v)	$O(1)$
exist_edge(u,v)	$O(\min(d_u, d_v))$
vertex_count()	$O(1)$
edge_count()	$O(1)$
vertices()	$O(n)$
edges()	$O(m)$
degree(u)	$O(1)$

→ d : degree



Space Complexity — $O(n+m)$

27/08/2020

③ Adjacency Matrix

Maintains a matrix of vertices, where each cell stores the reference to the edge.

Basically it's the extension of edge list structure where a sq. matrix is maintained which has the size of row and the cols as the no. of vertices and each cell of the matrix stores the reference to the edge but if no edge exist then the cell may contain the null value or some other value to represent that there is no edge.

$A[i, j] : 1 \rightarrow$ stores the reference of the edge from vertex u to vertex v.

$u \rightarrow v$

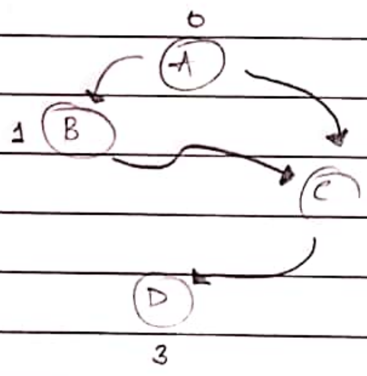
u → vertex with index i

v → vertex with index j

Vertices $V = \{0, 1, 2, 3 \ldots n-1\}$

& If $A[i,j]$ holds null or no val then $A[i,j] = 0$

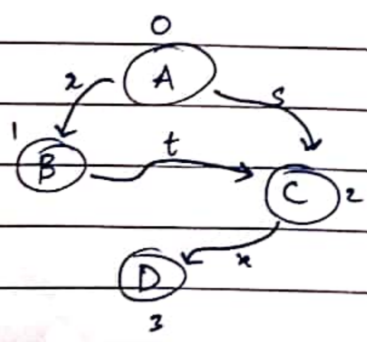
$$A = \begin{array}{c|c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \; j \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ i \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



Directed Graph

- Weighted directed Graph

$$A = \begin{array}{c|c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & r & s & 0 \\ 0 & 0 & t & 0 \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

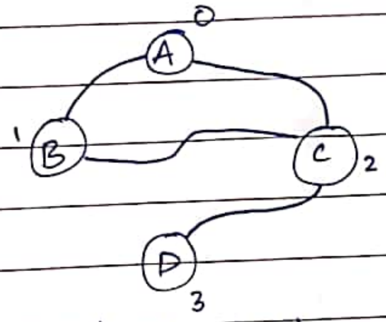


Weighted Directed Graph

27/08/2020

- Undirect Graph

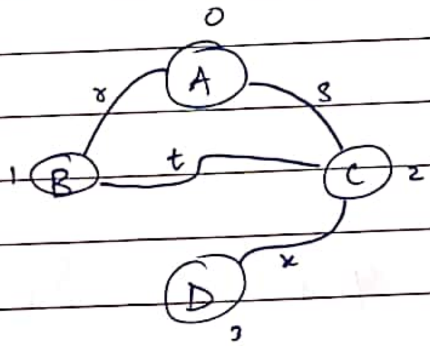
$$A = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 0 & 0 & 1 & 0 \end{array}$$



Undirected Graph

- Weighted Undirected Graph

$$A = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & r & s & \\ 1 & r & 0 & t & 0 \\ 2 & s & t & 0 & x \\ 3 & 0 & 0 & x & 0 \end{array}$$



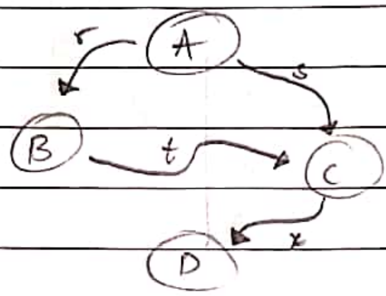
Weighted Undirected Graph

27 / 08 / 2020

• Adjacency Matrix Performance

Vertices = n   Edges = m   (since its a sq. matrix n = m)

Operations	Adjacency Matrix
insert_edge(u,v,wal)	O(1)
remove_edge(u,v)	O(1)
exist_edge(u,v)	O(1)
vertex_count()	O(1)
edge_count()	O(1)
vertices()	O(n)
edges()	O(m)
degree(w)	O(n)



Space Complexity $= O(n \times m)$
$= O(n \times n)$
$= O(n^2)$

★ Graphs — Summary of Performance

Vertices = n , Edges = m

	Edge list	Adjacency list	Adjacency Matrix
Space Complexity	O(n+m)	O(n+m)	O(n²)

27/06/2020

Operations	Edge list	Adjacency list	Adjacency Matrix
insert_edge (u,v,w=1)	O(1)	O(1)	O(1)
remove_edge (u,v)	O(1)	O(1)	O(1)
exists_edge (u,v)	O(m)	O(min(du, dv))	O(1)
vertex_count ()	O(1)	O(1)	O(1)
edge_count ()	O(1)	O(1)	O(1)
vertices ()	O(n)	O(n)	O(n)
edges ()	O(m)	O(m)	O(m)
degree ()	O(m)	O(1)	O(n)

28/08/2020

- Graph Traversals

(i) Traversal is a systematic procedure of exploring a graph.
just like a ~~bi~~ traversin a binary tree ie examining or exploring all of its vertices & edges.

(ii) Exploring : Examining all the vertices and edges of the graph

(iii) Efficient time : Visits to all vertices and edges is in efficient time.

"Graph traversal algorithms are used to determine how to travel from vertex to another following paths in the graph."

- Can Answer qs of reachability - Undirected Graphs

1. Computing a path from one vertex to another vertex.
2. Compute path to reach all other vertices given start vertex
3. Find whether a graph is conectd.
4. Computing connected components of the graph.
5. " cycle in a graph.
6. " spanning tree of the graph.

28 / 08 /2020

- Can ~~Ae~~ Ans qs of reachability — Directed Graph

① Computing direct path from one vertex to another vrtx.
② Finding all the vertices that can be reachable from given vertex.
③ Determine whether a graph is strongly conncted.
④ " " " " " acyclic.

- GRAPH TRAVERSAL ALGORITHSM

Breadth-First Search ← GRAPH TRAVERSAL ALGORITHSM → Depth-first search

In simple terms graph traversal is a technique or a method of startn from one ~~point~~ vertex and visiting all the vertices that can be reached from the start vertex

Scanned with CamScanner