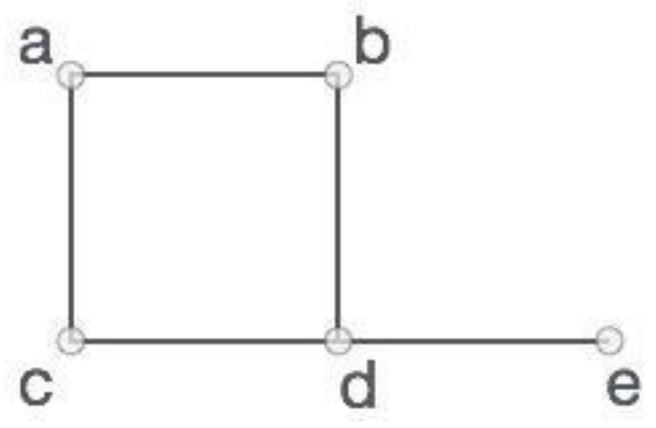
# Graphs (CO3)

- A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.
- Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices. Take a look at the following graph −



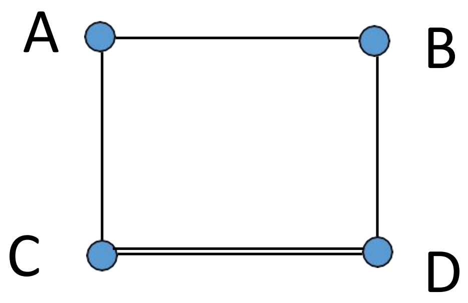

- In the given graph,
- V = {a, b, c, d, e}
- E = {ab, ac, bd, cd, de}

# Graph Terminology

- Two vertices joined by an edge are called the **end vertices** or **endpoints** of the edge.
- If an edge is directed its first endpoint is called the **origin** and the other is called the **destination**.
- Two vertices are said to be **adjacent** if they are endpoints of the same edge.
- • An edge is said to be **incident** on a vertex if the vertex is one of the edges endpoints.
- The **outgoing** edges of a vertex are the directed edges whose origin is that vertex.
- The **incoming** edges of a vertex are the directed edges whose destination is that vertex.

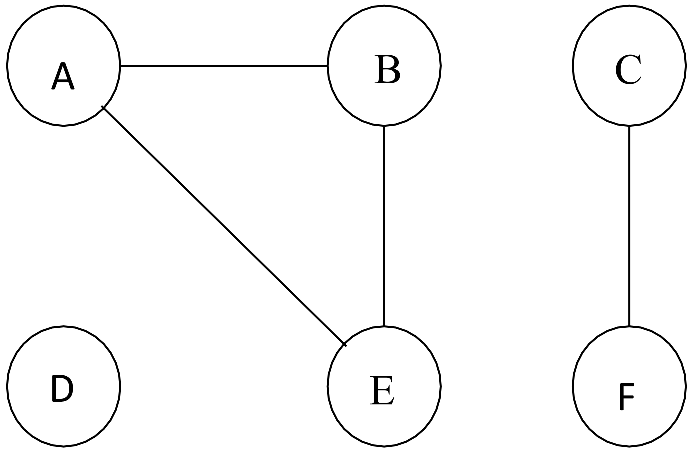
# Graph Terminology (Contd..)

- Adjacent, neighbors
    - Two vertices are *adjacent* and are *neighbors* if they are the
- endpoints of an edge
    - Example:
        - *A* and *B* are adjacent
        - *A* and *D* are not adjacent



# Graph Terminology (Contd..)

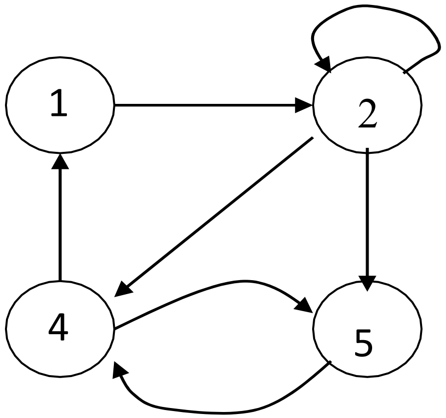
- Degree: Number of edges incident on a node




The degree of B is 2.

# Graph Terminology (Contd..)

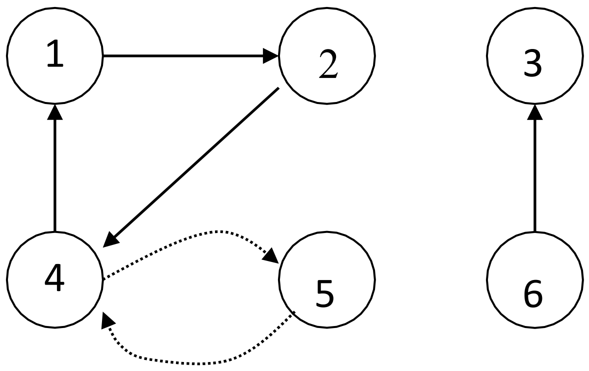
- Degree (Directed Graphs)
  - In degree: Number of edges entering a node
  - Out degree: Number of edges leaving a node
  - Degree = Indegree + Outdegree




The in degree of 2 is 2 and

the out degree of 2 is 3.

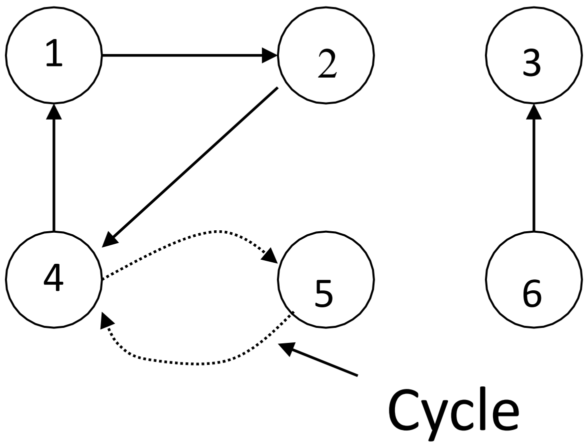
# Graph Terminology (Contd..)

- A *path* is a sequence of vertices such that there is an edge from each vertex to its successor.
- A path is *simple* if each vertex is distinct.
- A *circuit* is a path in which the terminal vertex coincides with the initial vertex
- 


- Simple path: [ 1, 2, 4, 5 ]
- Path: [ 1, 2, 4, 5, 4]
- Circuit: [ 1, 2, 4, 5, 4, 1]

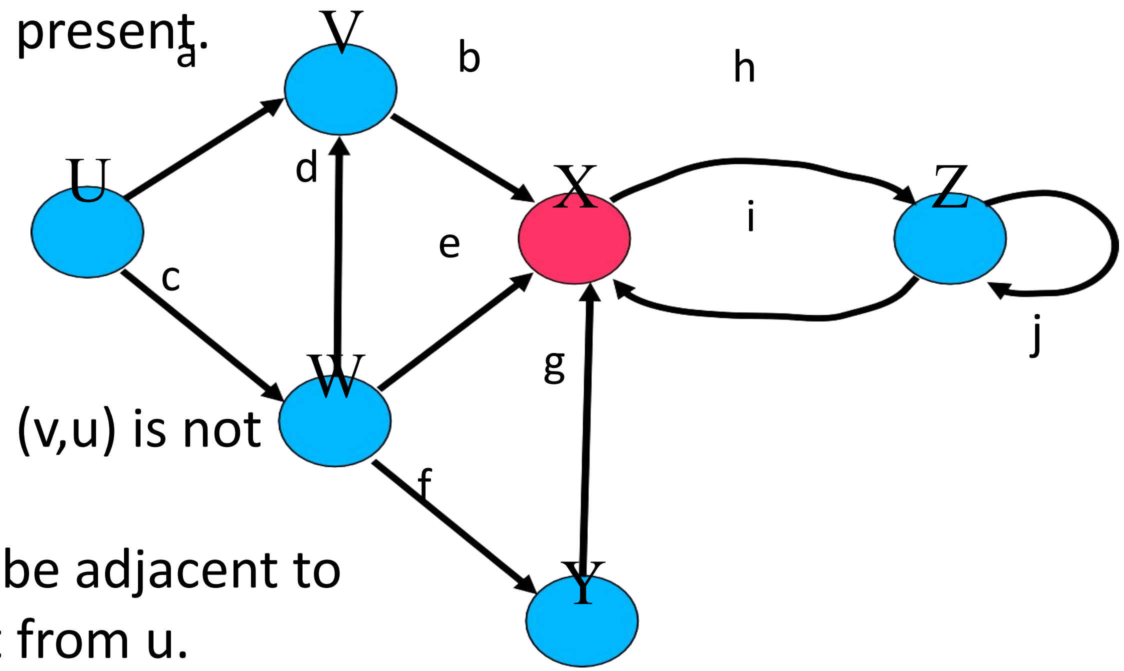
# Graph Terminology (Contd..)

- Cycle
  - A path from a vertex to itself is called a *cycle*.
  - A graph is called *cyclic* if it contains a cycle;
    - otherwise it is called *acyclic*



# Graph Terminology (Contd..)

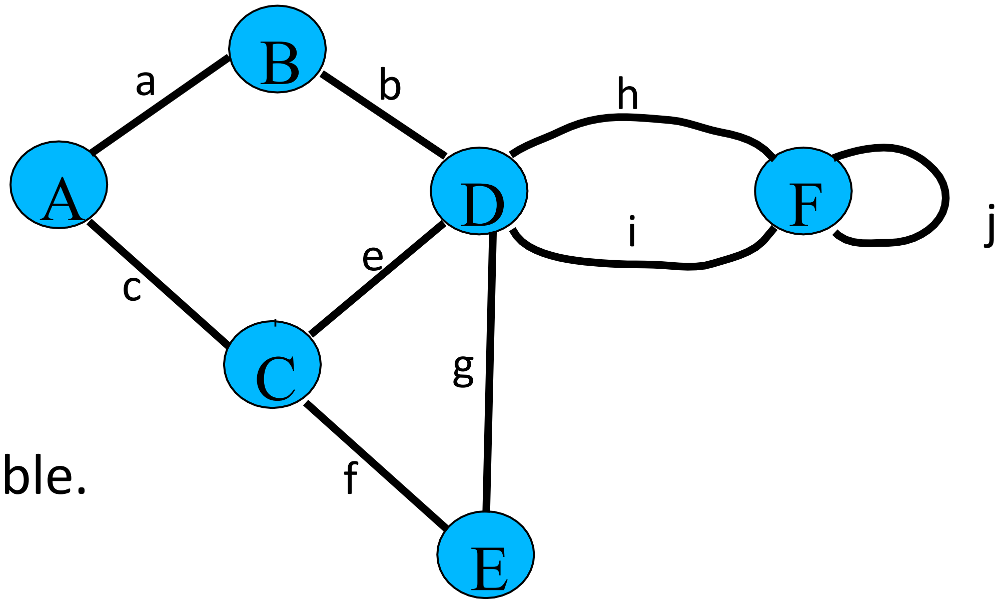
- **Directed Graph**
- A directed graph is one in which every edge (u, v) has a direction, so that (u, v) is different from (v, u)
- There are two possible situations that can arise in a directed graph between vertices u and v.
- i) only one of (u, v) and (v, u) is present.
- ii) both (u, v) and (v, u) are present.




- Here (u,v) is possible where as (v,u) is not possible
- In a directed edge, u is said to be adjacent to v and v is said to be adjacent from u.

# Graph Terminology (Contd..)

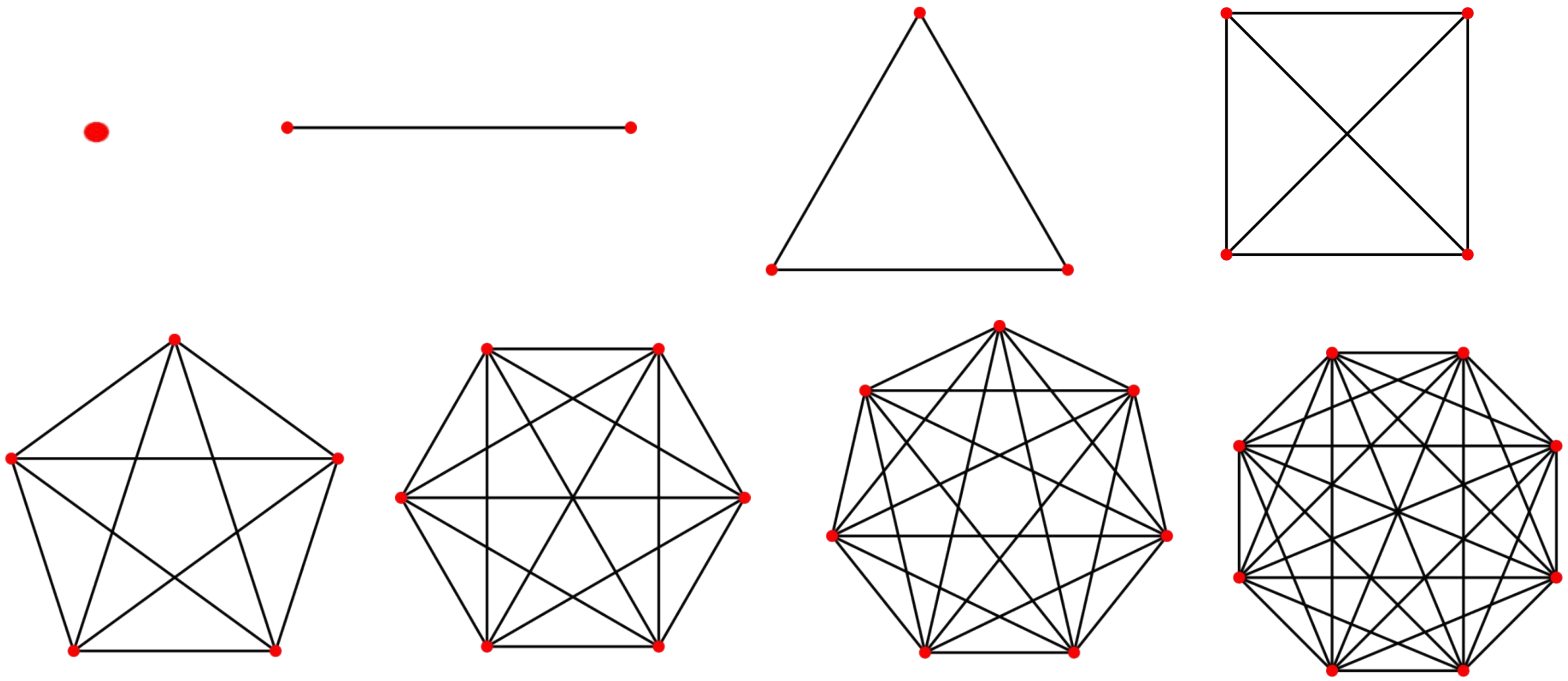
- **Undirected Graph**
- In an undirected graph, there is no distinction between (u, v) and (v, u).
- An edge (u, v) is said to be directed from u to v if the pair (u, v) is ordered with u preceding v.
- E.g. A Flight Route
- An edge (u, v) is said to be undirected if the pair (u, v) is not ordered
- E.g. Road Map




Here (u,v) and (v,u) both are possible.

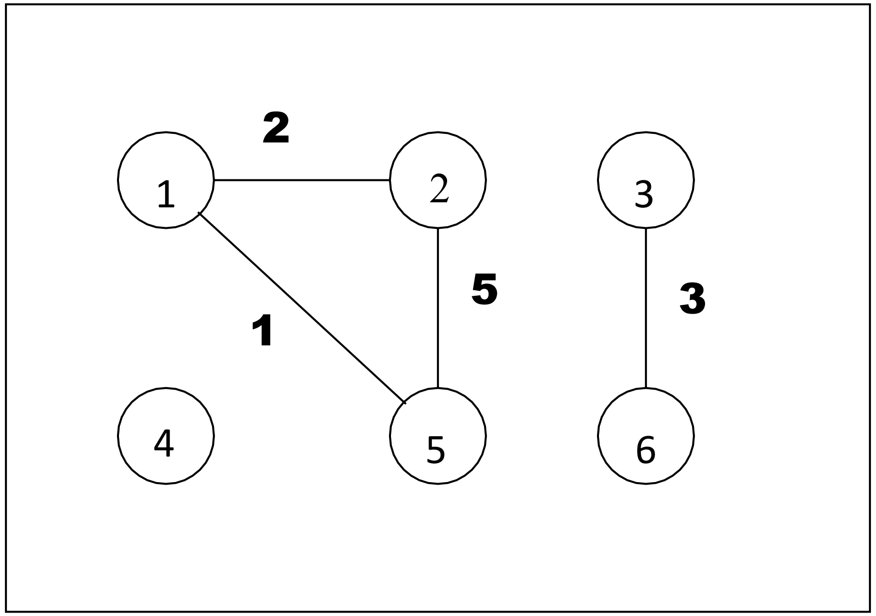
# Graph Terminology (Contd..)

- Complete Graph
- Complete Graph: A simple graph in which every pair of vertices are adjacent
- If no of vertices = n, then there are n(n-1)/2 edges



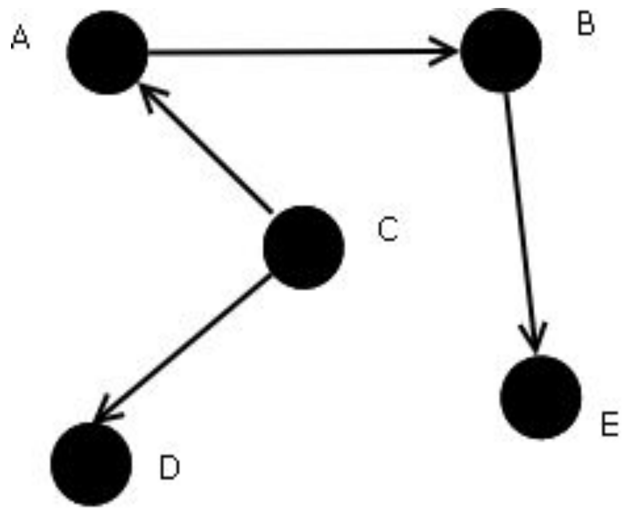
# Graph Terminology (Contd..)

- **Weighted Graph**
  Weighted graph is a graph for which each edge has an associated ***weight***, usually given by a ***weight function*** *w: E* ® **R**

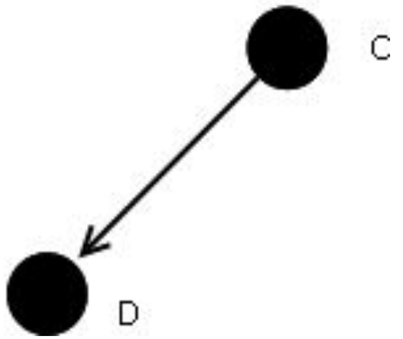


# Graph Terminology (Contd..)

- Subgraph
  - A graph whose vertices and edges are subsets of another graph.
  - A subgraph G'=(V',E') of a graph G = (V,E) such that V' ⊆ V and E' ⊆ E , Then G is a supergraph for G'.



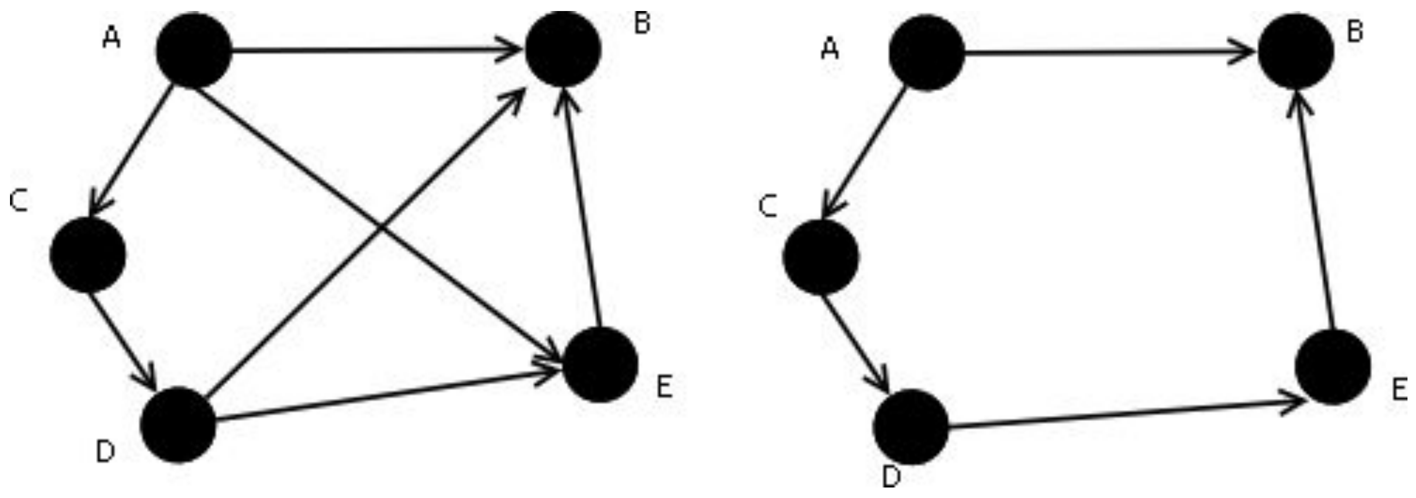

(G)




(G1)

# Graph Terminology (Contd..)

- Spanning Subgraph
  - A *spanning subgraph* is a subgraph that contains all the vertices of the original graph.

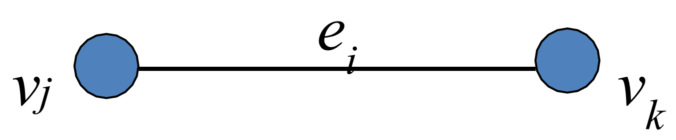


# Graph Representation (CO3)

- Adjacency Matrix
- Incidence Matrix
- Adjacency List

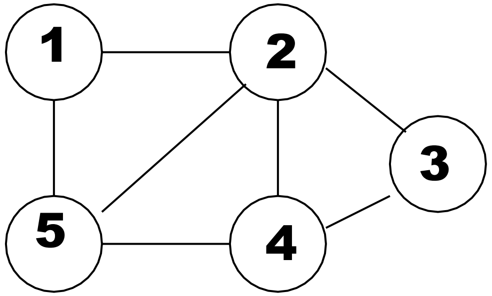
# Graph Representation

- Adjacency, Incidence, and Degree
  - Assume $e_i$ is an edge whose endpoints are $(v_j, v_k)$
  - The vertices $v_j$ and $vk$ are said to be ***adjacent***
  - The edge $ei$ is said to be ***incident upon*** $v_j$
  - ***Degree*** of a vertex $v_k$ is the number of edges incident upon $v_k$ . It is denoted as $d(v_k)$



# Adjacency Matrix

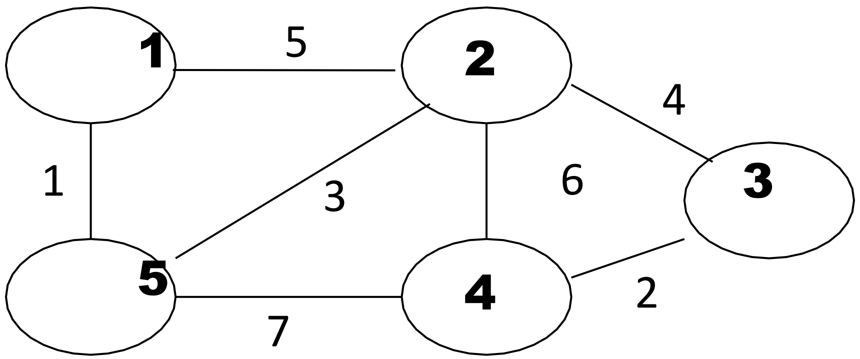
- Let $G = (V, E)$, $|V| = n$ and $|E|=m$
- The ***adjacency matrix*** of $G$ written $A(G)$, is the $|V|$ x $|E|$ matrix in which entry $a_{i,j}$ is 1 if an edge exists otherwise it is 0




| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |

# Adjacency Matrix (Weighted Graph)

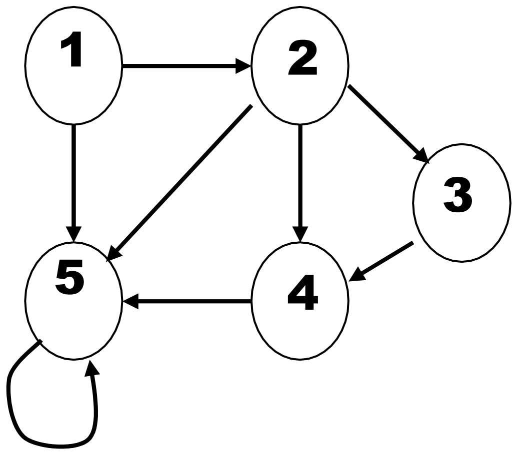
- Let $G = (V, E)$, $|V| = n$ and $|E|=m$
- The ***adjacency matrix*** of $G$ written $A(G)$, is the $|V|$ x $|E|$ matrix in which entry $a_{i,j}$ is weight of the edge if it exists otherwise it is 0

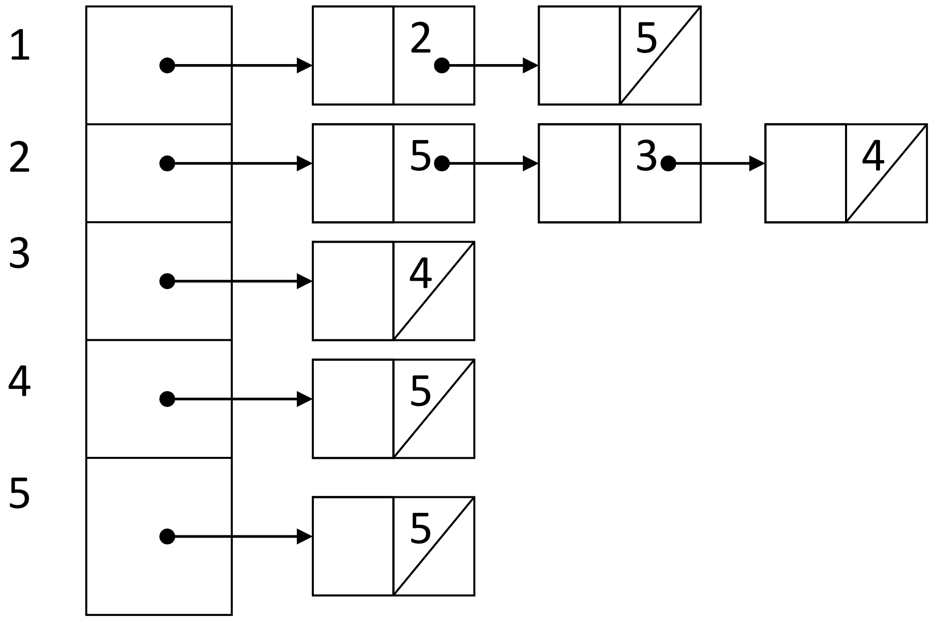



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 5 | 0 | 0 | 1 |
| 2 | 5 | 0 | 4 | 6 | 3 |
| 3 | 0 | 4 | 0 | 2 | 0 |
| 4 | 0 | 6 | 2 | 0 | 7 |
| 5 | 1 | 3 | 0 | 7 | 0 |

# Adjacency List

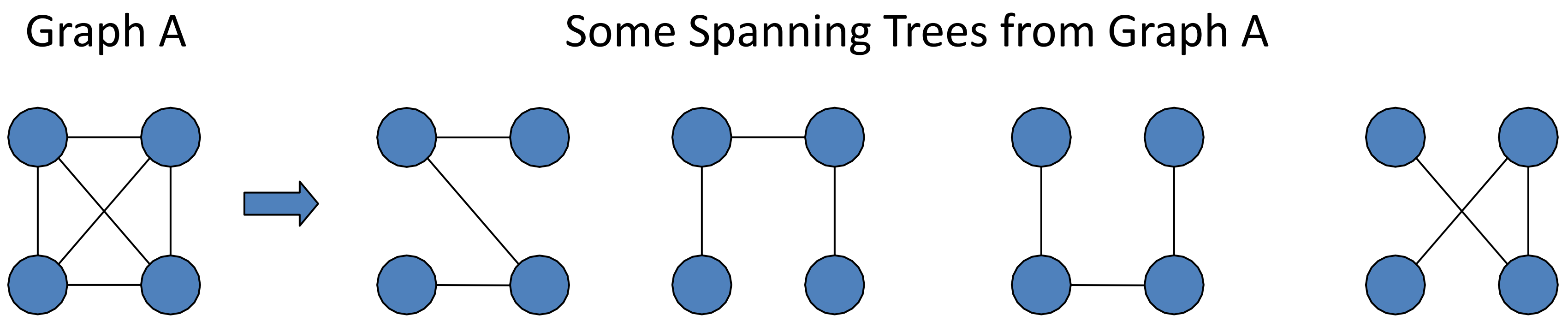
- Adjacency-list representation
  - an array of |*V* | elements, one for each vertex in *V*
  - For each *u in V* , *ADJ* [ *u* ] points to all its adjacent vertices.



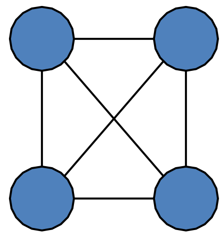
# Spanning Trees

- A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.
- A graph may have many spanning trees.

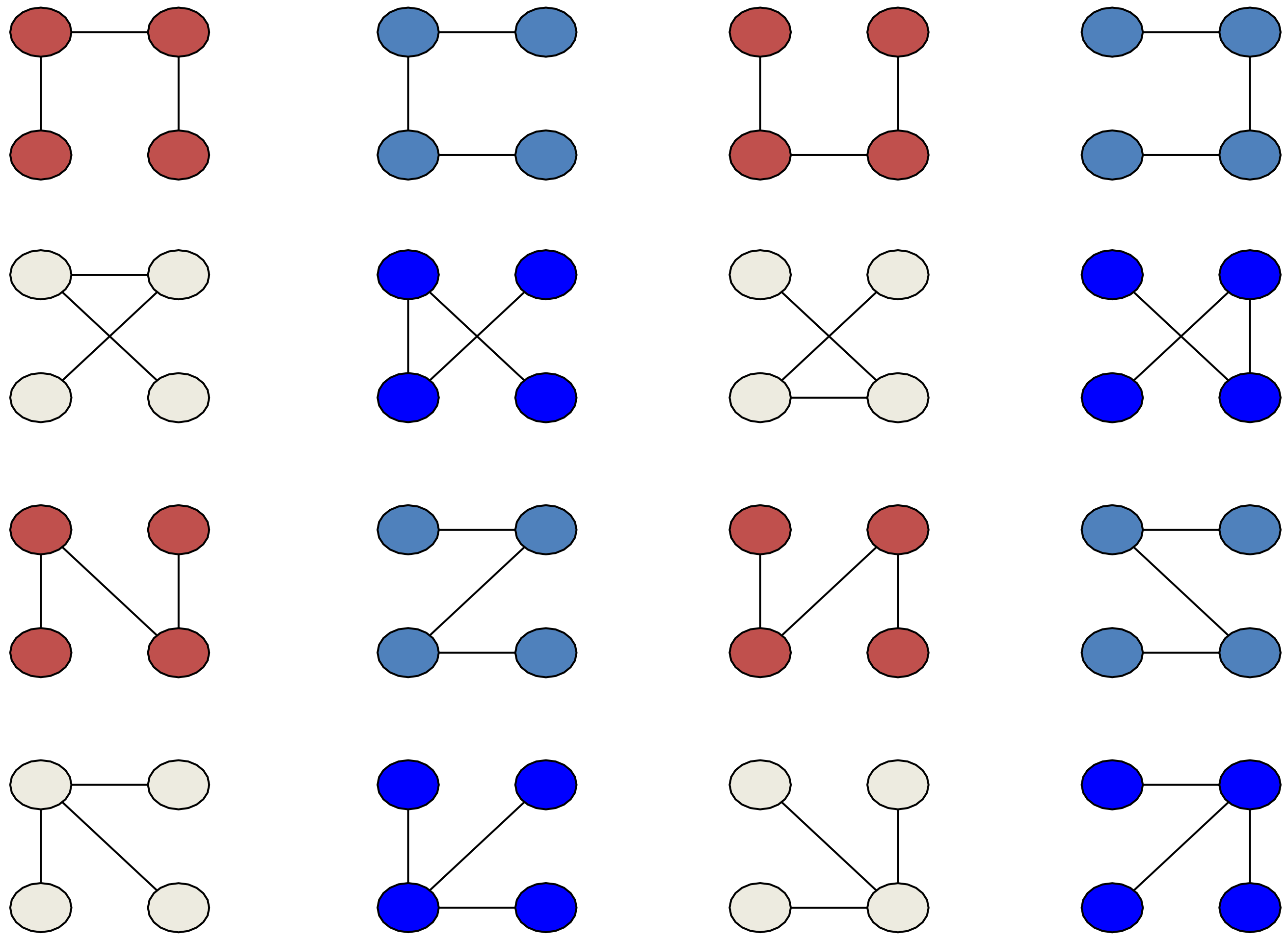


# Spanning Trees

Complete Graph



All 16 of its Spanning Trees



# MST Algorithms

- Kruskal Algorithm
- Prim’s Algorithm

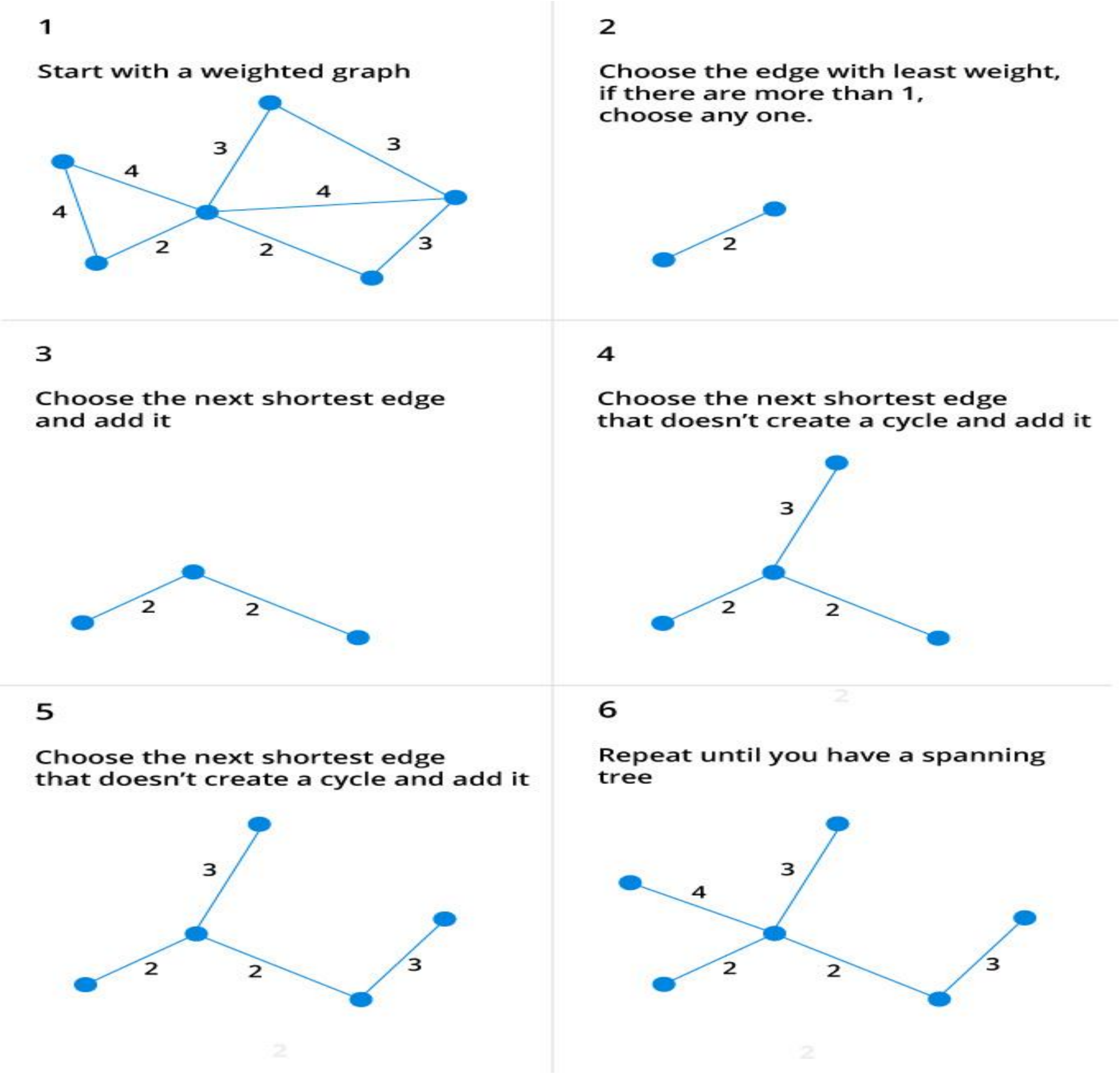
# Kruskal’s Algorithm

- Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which
  - form a tree that includes every vertex
  - has the minimum sum of weights among all the trees that can be formed from the graph

# Kruskal’s Algorithm Working

- We start from the edges with the lowest weight and keep adding edges until we we reach our goal.
- The steps for implementing Kruskal's algorithm are as follows:
  - Sort all the edges from low weight to high
  - Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
  - Keep adding edges until we reach all vertices.

# Kruskal’s Algorithm Example

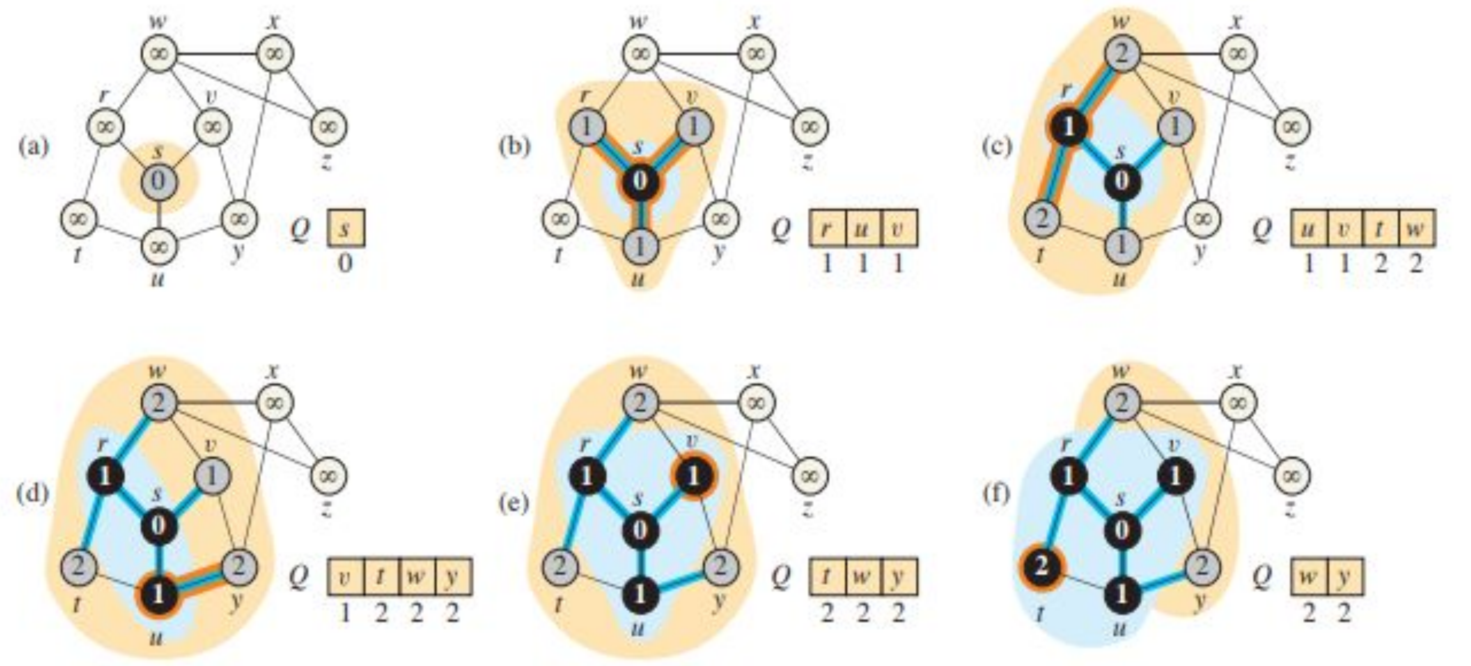


# Graph Traversal (CO3)

- Graph traversal is a technique used for a searching vertex in a graph.
- The graph traversal is also used to decide the order of vertices is visited in the search process.
- A graph traversal finds the edges to be used in the search process without creating loops.
- That means using graph traversal we visit all the vertices of the graph without getting into looping path.
- There are two graph traversal techniques and they are as follows...
  - **DFS (Depth First Search)**
  - **BFS (Breadth First Search)**

# BFS (Breadth First Search)

- **Example:**



# BFS (Breadth First Search)

- **Example:**