

# Graph Theory

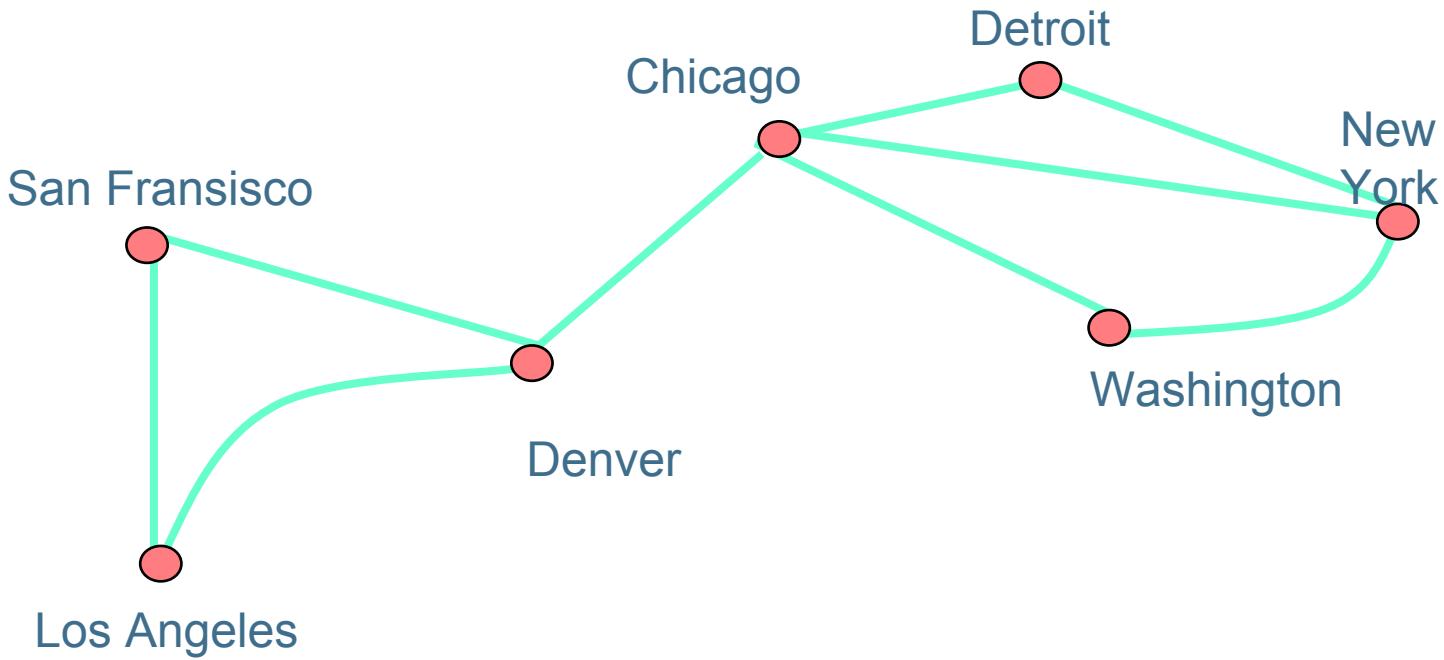
Dr. Shatrughan Modi  
Assistant Professor  
IIIT Una

# Table of Contents

1. Definition of a Graph
2. Different types of graph
3. Degree of a vertex
4. Complete Graph
5. Regular Graph
6. Cycle
7. Wheel
8. Hypercube
9. Complementary Graph
10. Converse of a Graph
11. Bipartite graph
12. Complete Bipartite graph
13. Representing Graphs
14. Graphs Isomorphism

## Definition

A **graph**  $G = (V,E)$  consists of  $V$ , a nonempty set of **vertices**, and  $E$ , a set of **edges**. Each edge has either one or two vertices associated with it, called its **endpoints**. An edge is said to connect its endpoints.

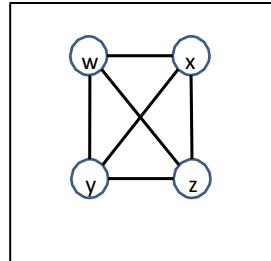


A Computer network

# 1. Simple Graph

A graph in which each edge connects two different vertices, and no two edges connect the same pair of vertices, then the graph is a **simple graph**.

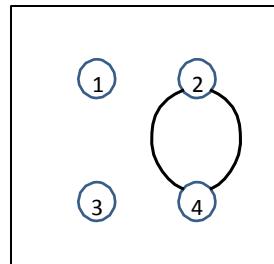
**Example:**

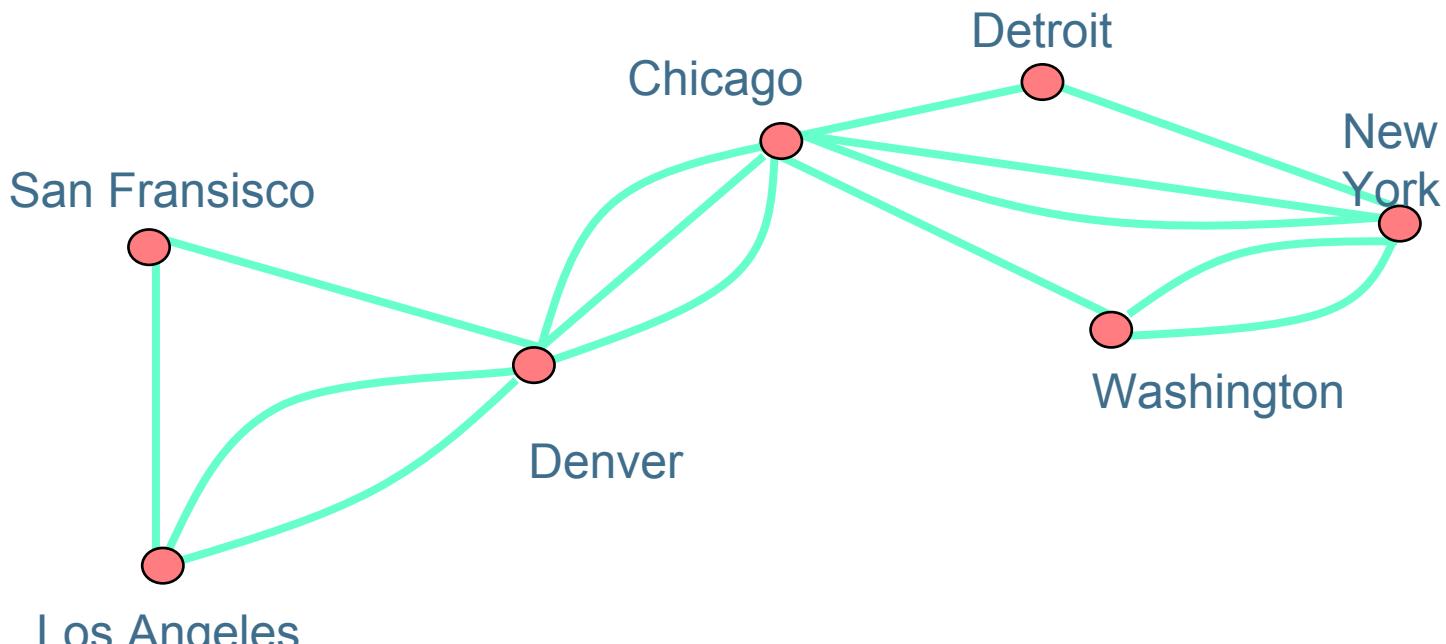


## 2. Multigraph

A graph in which more than one edge connecting the same pair of vertices. Graphs that may have multiple edges connecting the same vertices are called **Multigraphs**.

**Example:**

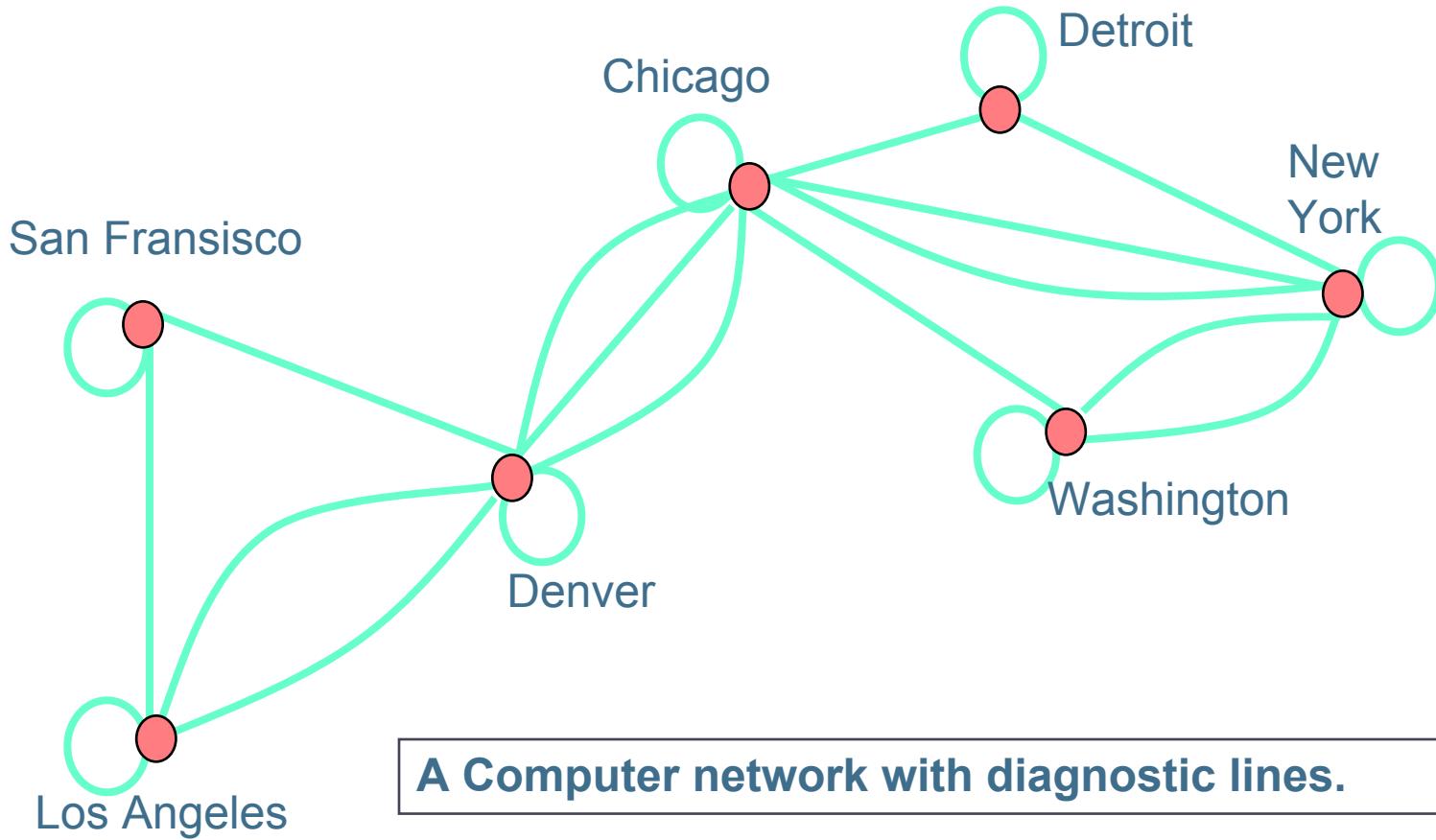




**A Computer network with multiple lines.**

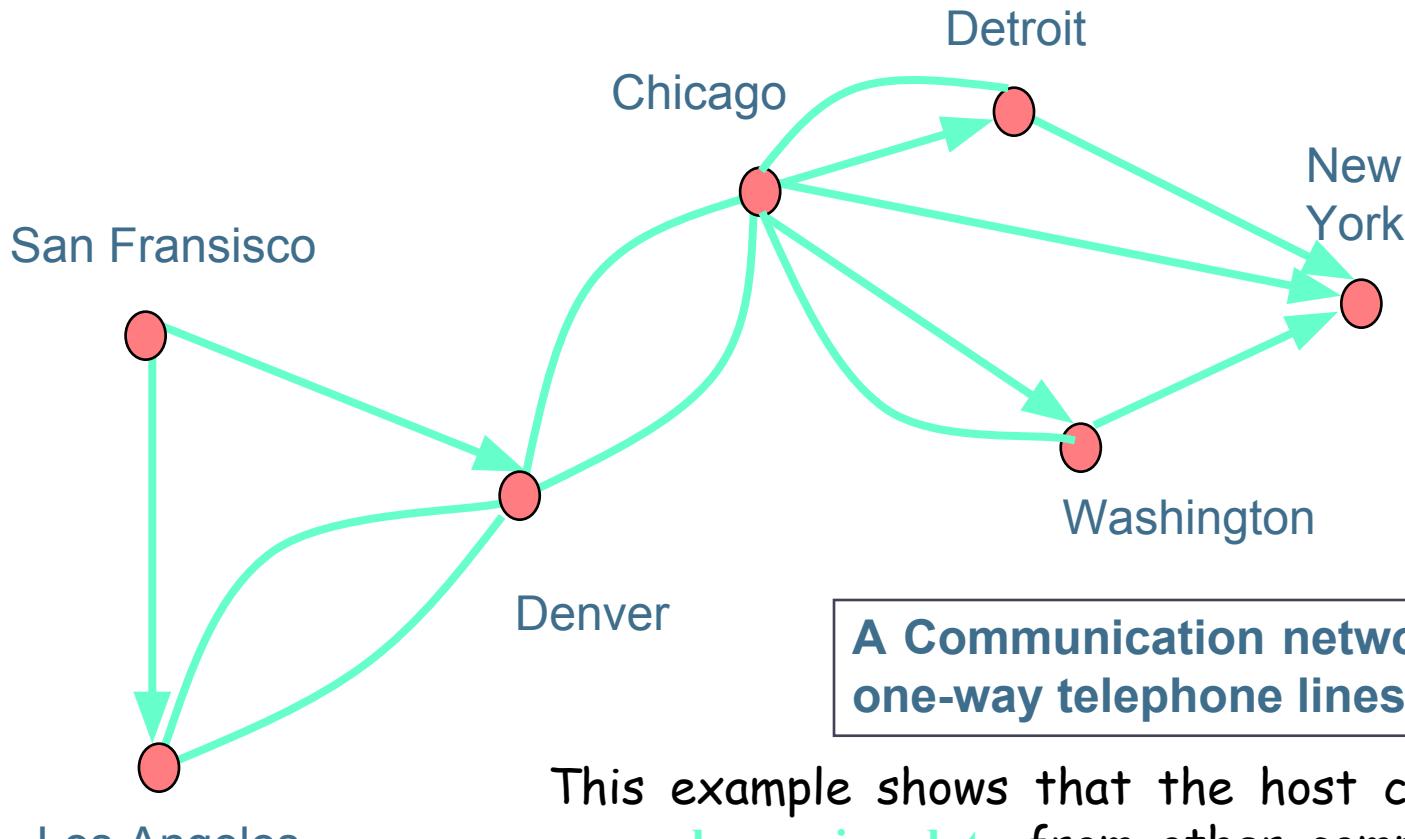
### 3. Pseudo Graph

A **pseudo graph**  $G = (V, E)$  consists of a set  $V$  of vertices, a set  $E$  of edges, and a function  $f$  from  $E$  to  $\{\{u, v\} \mid u, v \in V\}$ . An edge is a **loop** if  $f(e) = \{u, u\} = \{u\}$  for some  $u \in V$ .



## 4. Directed Graph

A **directed graph**  $G$  consists of a nonempty set  $V$  of vertices and a set  $E$  of directed edges, where each edge is associated with an ordered pair of vertices. We write  $G = (V, E)$  to denote the graph.



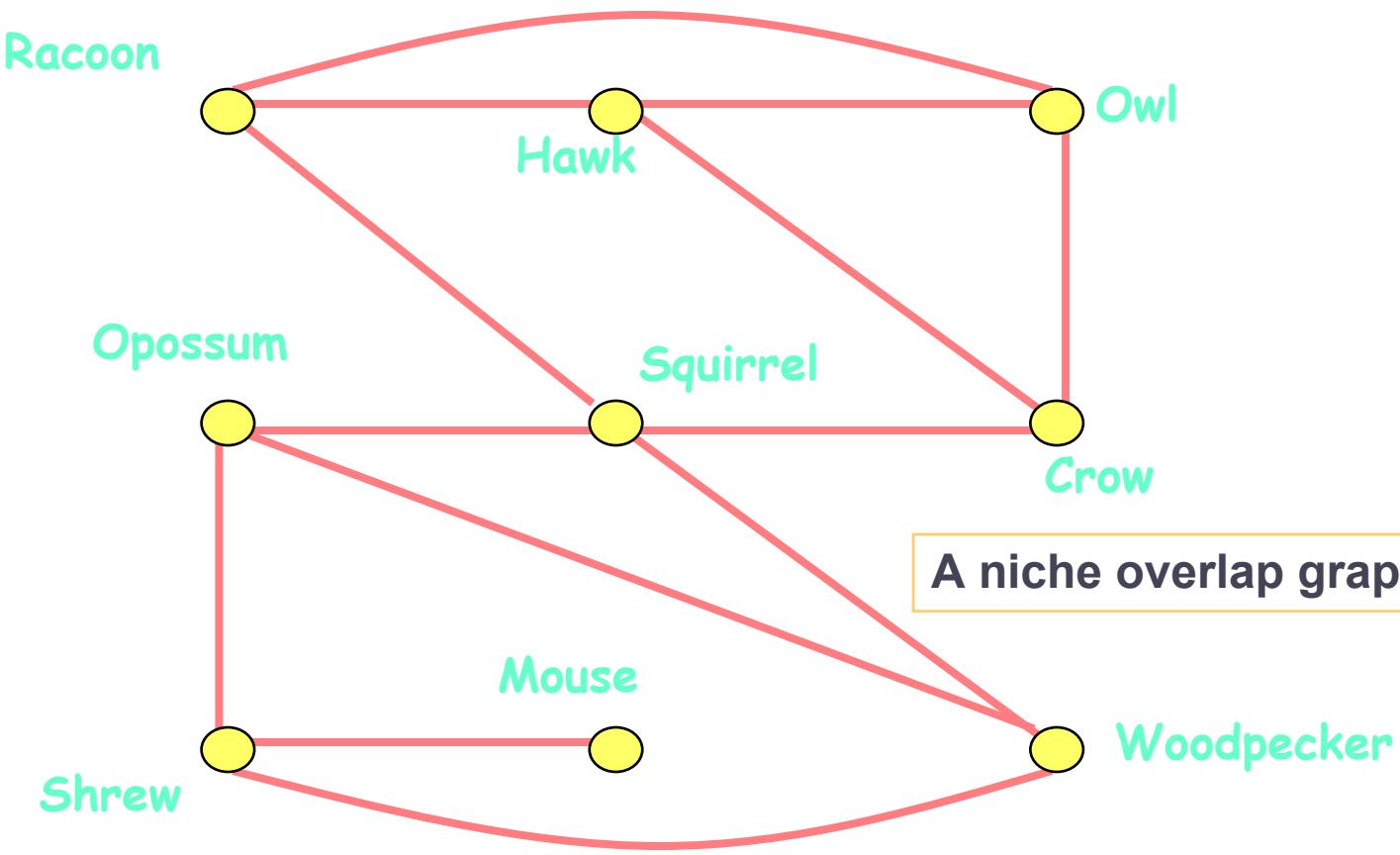
A Communication network with one-way telephone lines.

This example shows that the host computer can only receive data from other computer, it cannot emit.

## 5. Undirected Graph

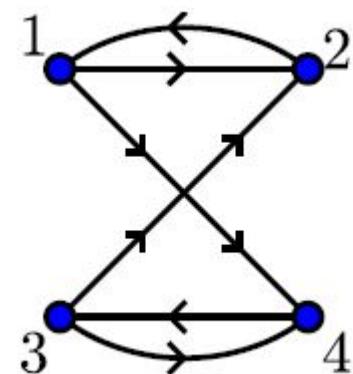
An **undirected edge** connect two vertices if the two species represented by these vertices compete for food.

**Example:** Competition between species in an ecological system can be modeled using a **niche overlap graph**.



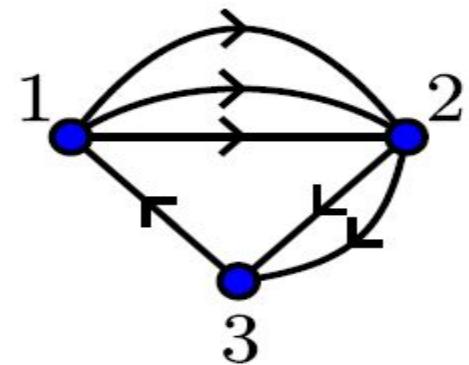
## 6. Simple Directed Graph

When a directed graph has no loops and has no multiple directed edges, it is called a simple directed graph.



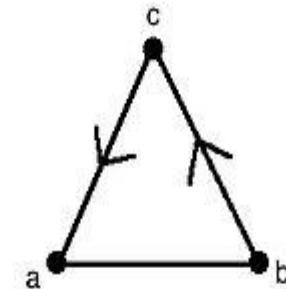
## 7. Directed Multigraphs

In some networks, multiple communication links between two data centers may be present. Directed graphs that may have multiple directed edges from a vertex to a second vertex are used to model such networks. We called such graphs are **directed multigraphs**.



## 8. Mixed Graph

A graph with both **directed** and **undirected** edges is called a **Mixed graph**.



# Graph Terminology

Type	Edges	Multiple Edges Allowed?	Loops Allowed?
Simple graph	Undirected	No	No
Multigraph	Undirected	Yes	No
Pseudograph	Undirected	Yes	Yes
Simple directed graph	Directed	No	No
Directed multigraph	Directed	Yes	Yes
Mixed Graph	Directed and Undirected	Yes	Yes

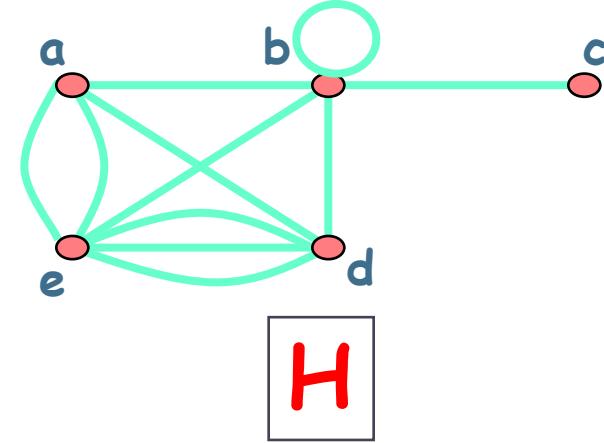
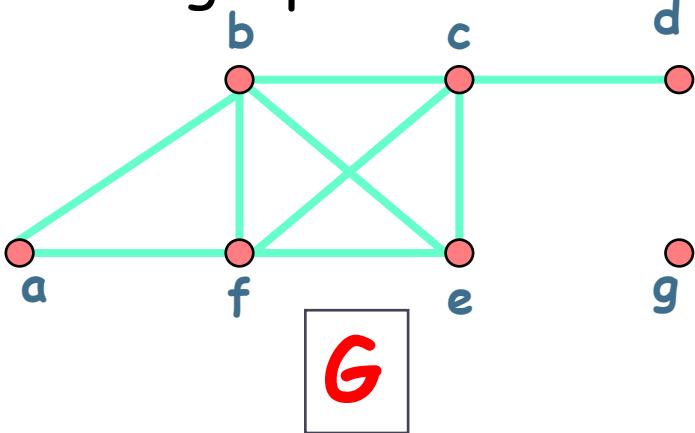
# Table of Contents

1. Definition of a Graph
2. Different types of graph
3. Degree of a vertex
4. Complete Graph
5. Regular Graph
6. Cycle
7. Wheel
8. Hypercube
9. Complementary Graph
10. Converse of a Graph
11. Bipartite graph
12. Complete Bipartite graph
13. Representing Graphs
14. Graphs Isomorphism

## Degree of a vertex

The **degree of a vertex in an undirected graph** is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex  $v$  is denoted by **deg(v)**.

**Example:** What are the degrees of the vertices in the graphs  $G$  and  $H$ ?



**Solution:**

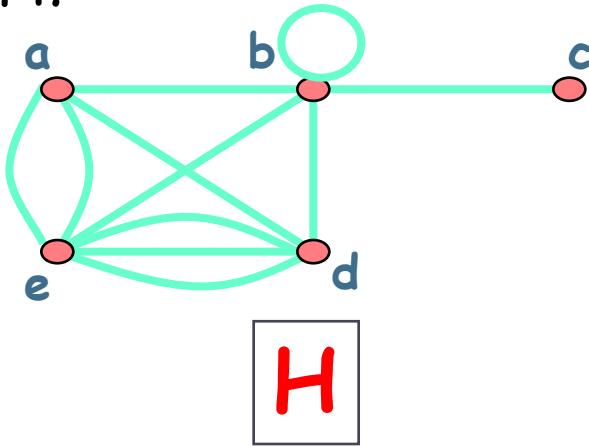
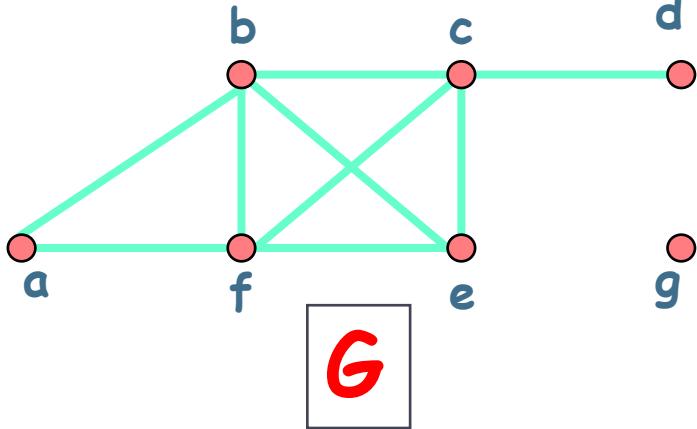
$$\text{In } G \begin{cases} \deg(a) = 2 \\ \deg(b) = \deg(c) = \deg(f) = 4 \\ \deg(d) = 1 \\ \deg(e) = 3 \\ \deg(g) = 0 \end{cases}$$

$$\text{In } H \begin{cases} \deg(a) = 4 \\ \deg(b) = \deg(e) = 6 \\ \deg(c) = 1 \\ \deg(d) = 5 \end{cases}$$

## Neighborhood of a vertex

The **neighborhood (neighbor set)** of a vertex  $v$  in a undirected graph, denoted  $N(v)$  is the set of vertices adjacent to  $v$ .

**Example:** What are the neighborhood of the vertices in the graphs G and H?



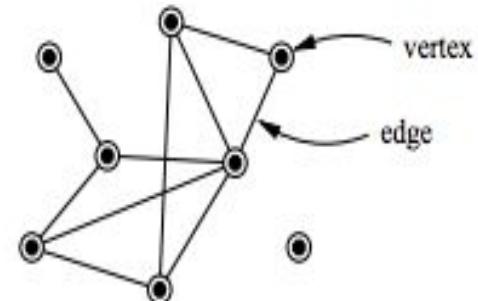
**Solution:**

$$\text{In } G \left\{ \begin{array}{l} N(a) = \{b, f\} \\ N(b) = \{a, c, f, e\} \\ N(c) = \{d, e, f, b\} \\ N(d) = \{c\} \\ N(e) = \{b, c, f\} \\ N(f) = \{a, b, c, e\} \\ N(g) = \{\} \end{array} \right.$$

$$\text{In } H \left\{ \begin{array}{l} N(a) = \{b, e, d\} \\ N(b) = \{c, e, d, b, a\} \\ N(c) = \{b\} \\ N(d) = \{e, a, b\} \\ N(e) = \{a, d, b\} \end{array} \right.$$

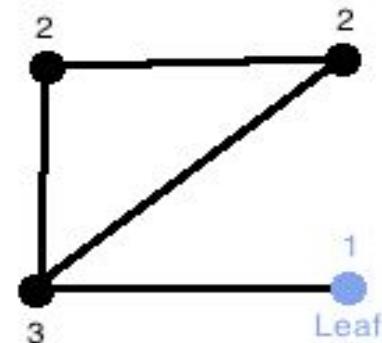
## Isolated vertex

An **isolated vertex** is a vertex with degree **zero**; that is, a vertex that is not an endpoint of any edge (the example image illustrates one isolated vertex).



## Pendant vertex

A **leaf vertex** (also **pendant vertex**) is a vertex with degree one.



## Handshaking Theorem

Let  $G = (V, E)$  be an undirected graph with  $e$  edges. Then

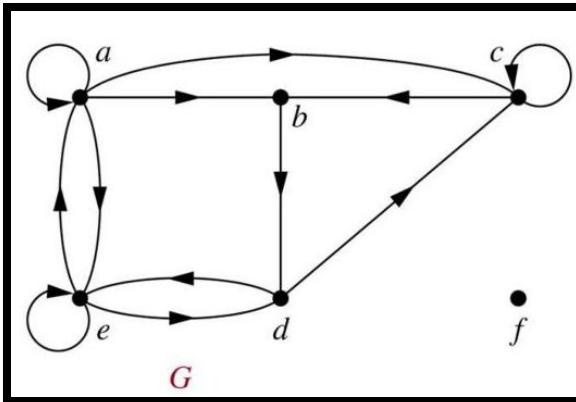
$$2e = \sum_{v \in V} \deg(v)$$

(Note that this applies even if multiple edges & loops are present.)

**Example:** How many edges are there in a graph with ten vertices each of degree 6 ?

**Solution:** Since the sum of the degrees of the vertices is  $6*10 = 60 \Rightarrow 2e = 60$ . Therefore,  $e = 30$

**Example:** Find the in-degree and the out-degree of each vertex in the graph  $G$ .



**Solution:** The in-degree of  $G$  are:  $\deg^-(a) = 2$ ,  $\deg^-(b) = 2$ ,  $\deg^-(c) = 3$   $\deg^-(d) = 2$ ,  $\deg^-(e) = 3$ , and  $\deg^-(f) = 0$ .

The out-degree of  $G$  are:  $\deg^+(a) = 4$ ,  $\deg^+(b) = 1$ ,  $\deg^+(c) = 2$ ,  $\deg^+(d) = 2$ ,  $\deg^+(e) = 3$ , and  $\deg^+(f) = 0$

Let  $G = (V, E)$  be a graph with directed edges. Then

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |E|.$$

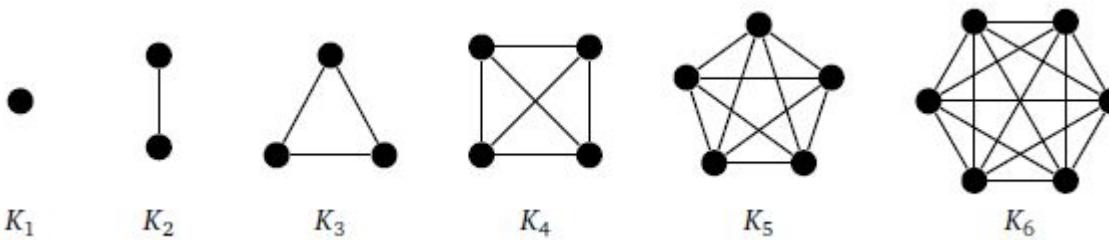
# Table of Contents

1. Definition of a Graph
2. Different types of graph
3. Degree of a vertex
4. Complete Graph
5. Regular Graph
6. Cycle
7. Wheel
8. Hypercube
9. Complementary Graph
10. Converse of a Graph
11. Bipartite graph
12. Complete Bipartite graph
13. Representing Graphs
14. Graphs Isomorphism

## Complete Graph

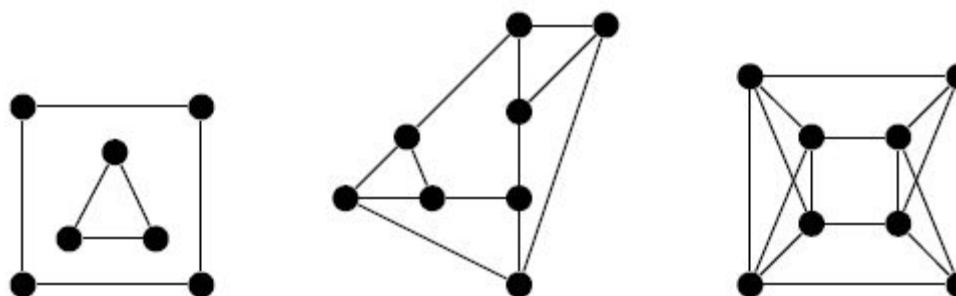
A **complete graph** on  $n$  vertices, denoted by  $K_n$ , is a simple graph that contains exactly one edge between each pair of distinct vertices. The graphs  $K_n$  for  $n=1, 2, 3, 4, 5, 6$ .

Examples :



## Regular Graph

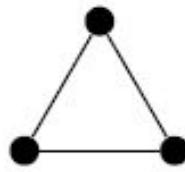
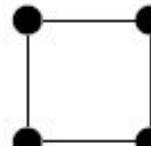
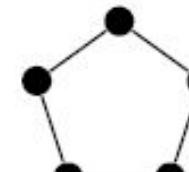
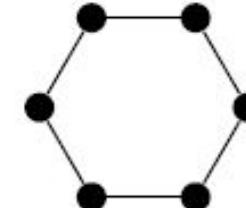
A **regular graph** is one in which every vertex has the same degree. The term  $k$ -regular is used to denote a graph in which every vertex has degree  $k$ . There are many types of regular graphs. Shown below are a 2-regular, a 3-regular, and a 4-regular graph.



## Cycle

A **cycle**  $C_n$ ,  $n \geq 3$ , is a graph that consists of  $n$  vertices  $v_1, v_2, \dots, v_n$  and  $n$  edges  $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$

Examples :

 $C_3$  $C_4$  $C_5$  $C_6$

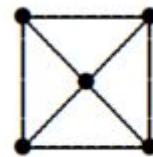
## Wheel

A **wheel**  $W_n$ ,  $n \geq 3$ , is a graph that consists of a cycle  $C_n$  with an extra vertex that connects to each vertex in  $C_n$ .

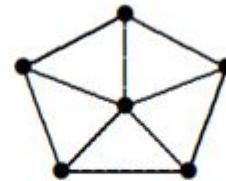
Examples :



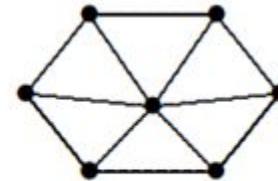
$W_3$



$W_4$



$W_5$

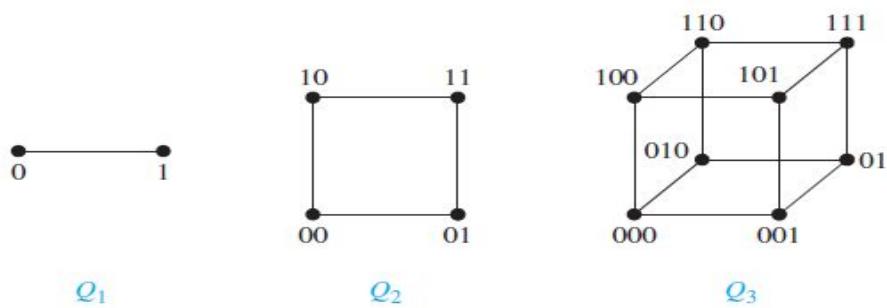


$W_6$

## N-dimensional Hypercube (n cube)

Hypercube graph  $Q_n$  is the graph formed from the vertices and edges of an n-dimensional hypercube. For instance, the cubical graph  $Q_3$  is the graph formed by the 8 vertices and 12 edges of a three-dimensional cube.  $Q_n$  has  $2^n$  vertices,  $2^{n-1}n$  edges, and is a regular graph with n edges touching each vertex.

Examples :



The  $n$ -cube  $Q_n$ ,  $n = 1, 2, 3$ .

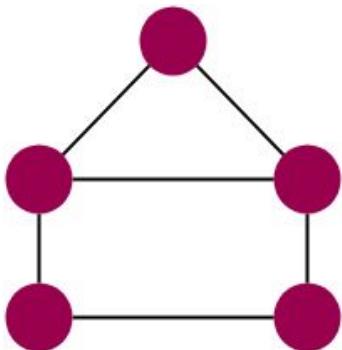
# Table of Contents

1. Definition of a Graph
2. Different types of graph
3. Degree of a vertex
4. Complete Graph
5. Regular Graph
6. Cycle
7. Wheel
8. Hypercube
9. Complementary Graph
10. Converse of a Graph
11. Bipartite graph
12. Complete Bipartite graph
13. Representing Graphs
14. Graphs Isomorphism

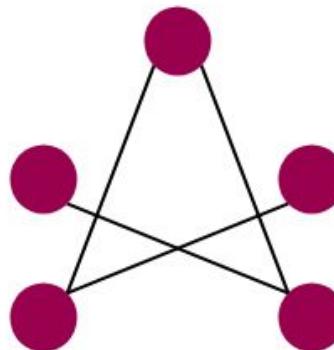
## Complementary Graph

The complement of a graph  $G$  is a graph  $\bar{G}$  with the same vertex set as  $G$ , but whose edges are exactly the opposite of the edges in  $G$ . That is,  $uv$  is an edge of  $\bar{G}$  if and only if  $uv$  is not an edge of  $G$ . Shown below are a graph and its complement.

Example :



Graph  $G$



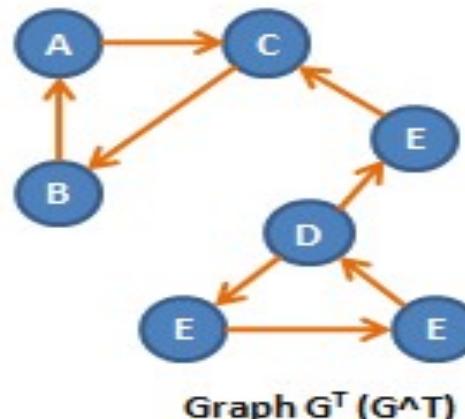
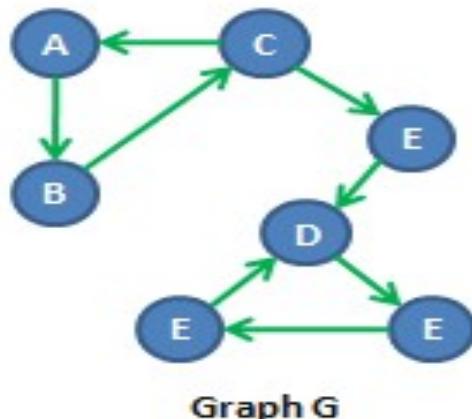
Complement Graph  $\bar{G}$

One way to think of the complement is that the original plus the complement equals a complete graph. So we can get the complement by taking a complete graph and removing all the edges associated with the original graph.

# Converse of a Graph

**Converse**, transpose or **reverse** of a directed **graph**  $G$  is another directed **graph** on the same set of vertices with all of the edges reversed compared to the orientation of the corresponding edges in  $G$ .

Example :

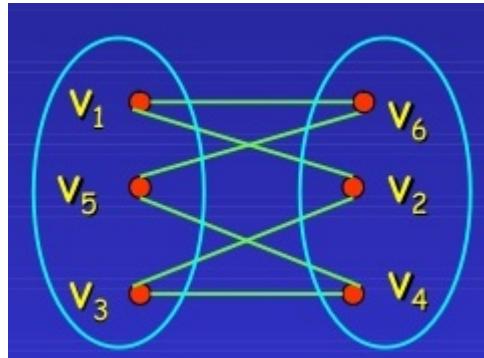


Graph  $G$  and its transpose  $G^T$

## Bipartite Graph

A simple graph  $G$  is called bipartite if its vertex set  $V$  can be partitioned into two disjoint sets  $V_1$  and  $V_2$  such that every edge in the graph connects a vertex in  $V_1$  and a vertex in  $V_2$ . In other words, there are no edge in  $G$  connects either two vertices in  $V_1$  or in  $V_2$ . When this condition holds, we call the pair  $(V_1, V_2)$  a bipartition of the vertex set  $V$  of  $G$ .

Example :

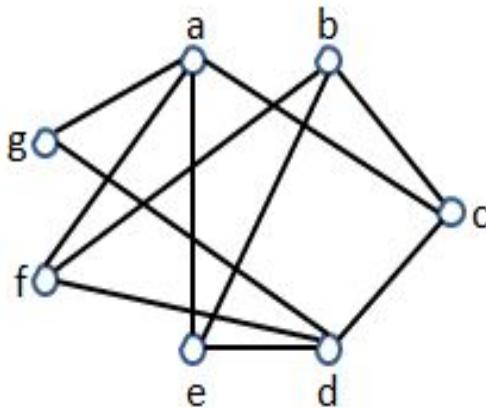


Its vertex set can be partitioned it into 2 sets:

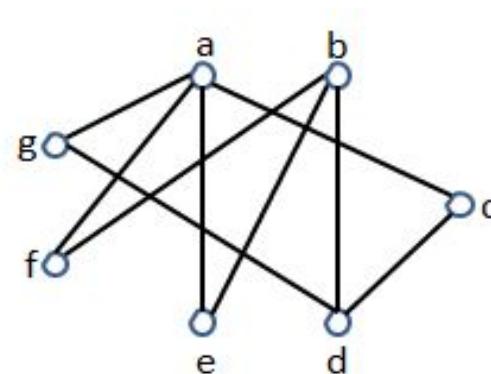
$$V_1 = \{v_1, v_3, v_5\} \quad V_2 = \{v_2, v_4, v_6\}$$

## Examples

Which of the following is a bipartite graph?



$G$



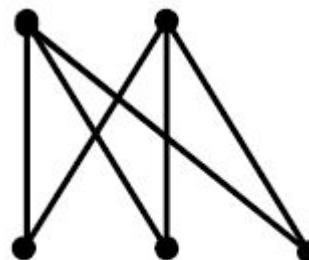
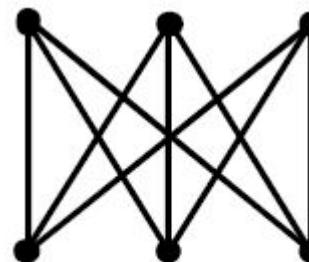
$H$

Theorem: A simple graph is bipartite if and only if it is possible to assign one of two different colors to each vertex, so that no two adjacent vertices are assigned the same color.

## Complete Bipartite Graph

A **complete bipartite graph  $K_{m,n}$**  is a graph that has its vertex set partitioned into two subsets of  $m$  and  $n$  vertices respectively with an edge between two vertices if and only if one vertex is in the first subset and other vertex is in the second subset.

Examples :

 $K_{2,3}$  $K_{3,3}$

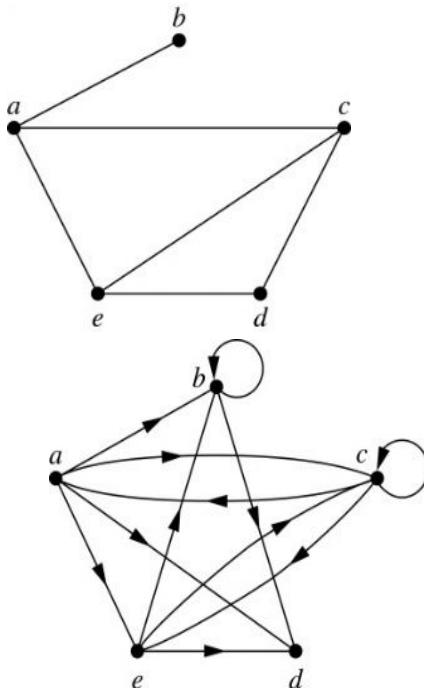
# Table of Contents

1. Definition of a Graph
2. Different types of graph
3. Degree of a vertex
4. Complete Graph
5. Regular Graph
6. Cycle
7. Wheel
8. Hypercube
9. Complementary Graph
10. Converse of a Graph
11. Bipartite graph
12. Complete Bipartite graph
13. Representing Graphs
14. Graphs Isomorphism

# Representing Graphs: Adjacency Lists

**Definition:** An *adjacency list* represents a graph (with no multiple edges) by specifying the vertices that are adjacent to each vertex.

Examples:



**TABLE 1** An Adjacency List for a Simple Graph.

Vertex	Adjacent Vertices
a	b, c, e
b	a
c	a, d, e
d	c, e
e	a, c, d

**TABLE 2** An Adjacency List for a Directed Graph.

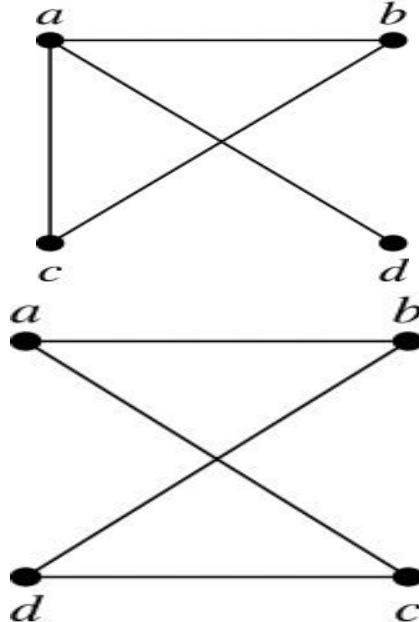
Initial Vertex	Terminal Vertices
a	b, c, d, e
b	b, d
c	a, c, e
d	
e	b, c, d

## Representing Graphs: Adjacency Matrix

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of } G, \\ 0 & \text{otherwise.} \end{cases}$$

**Examples:**

The vertex ordering is  $a, b, c, d$ .



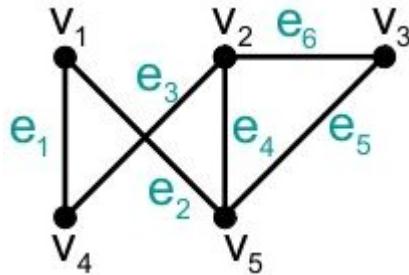
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

## Representing Graphs: Incidence Matrices

$$m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i, \\ 0 & \text{otherwise.} \end{cases}$$

**Example:**

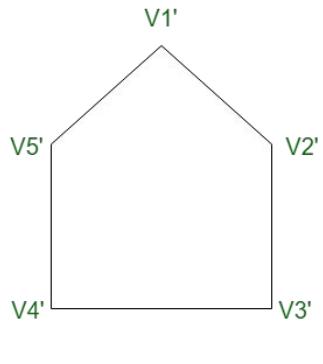
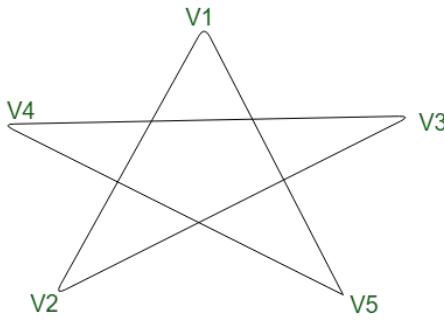


$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

## Isomorphism of Graphs

"The simple graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  are isomorphic if there exists a (one to one and onto) i.e. bijective function  $f$  from  $V_1$  to  $V_2$ .

Example:



Let  $f$  be a bijective function from  $V$  to  $V'$ .

Let the correspondence between the graphs be-

$$v_1' = f(v_1)$$

$$v_2' = f(v_5)$$

$$v_3' = f(v_4)$$

$$v_4' = f(v_3)$$

$$v_5' = f(v_2)$$

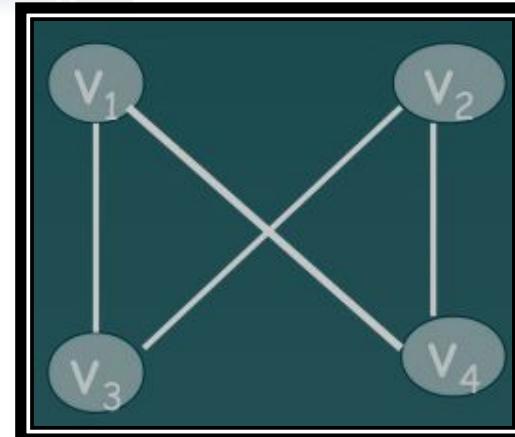
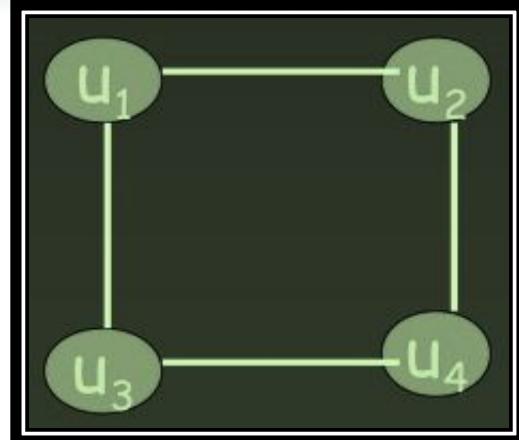
## Example:

Representation example:  $G1 = (V1, E1)$ ,

$G2 = (V2, E2)$

$f(u_1) = v_1, f(u_2) = v_4, f(u_3) = v_3, f(u_4) = v_2,$

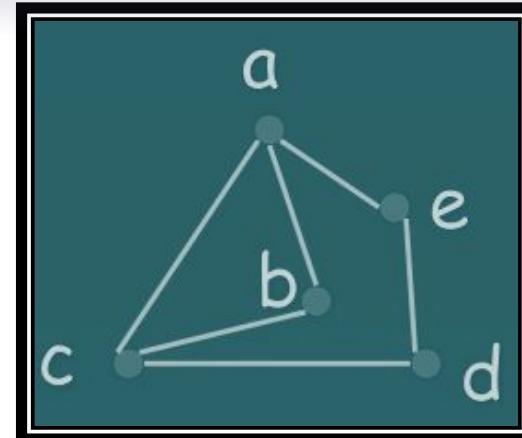
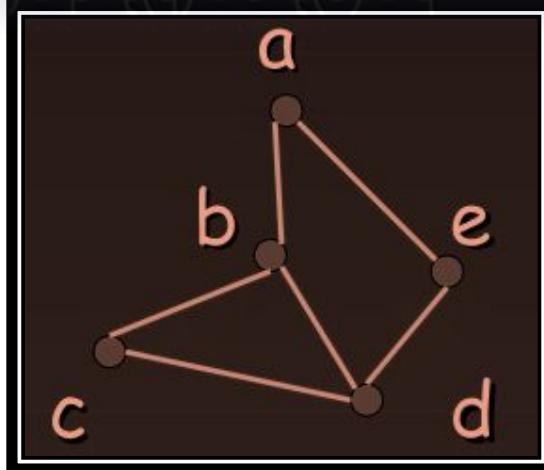
$$\perp(\Pi^T) = \wedge^{T^1} \perp (\Pi^S) = \wedge^{S^1} \perp (\Pi^B) = \wedge^{B^1} \perp (\Pi^A) = \wedge^{A^1}$$



## Example:

**Solution:** Yes, they are isomorphic, because they can be arranged to look identical. You can see this if in the right graph you move vertex b to the left of the edge {a, c}. Then the isomorphism  $f$  from the left to the right graph is:  $f(a) = e$ ,  $f(b) = a$ ,  $f(c) = b$ ,  $f(d) = c$ ,  $f(e) = d$ .

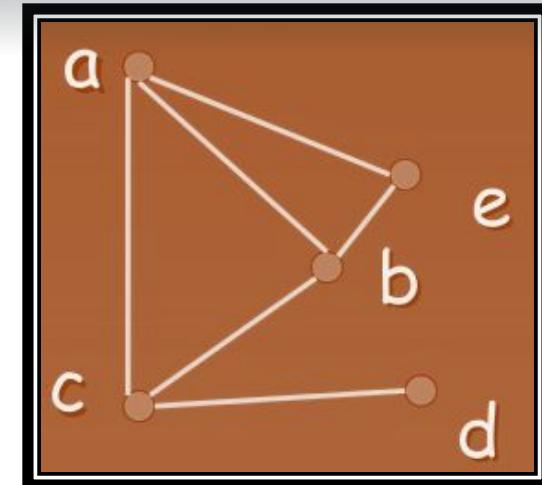
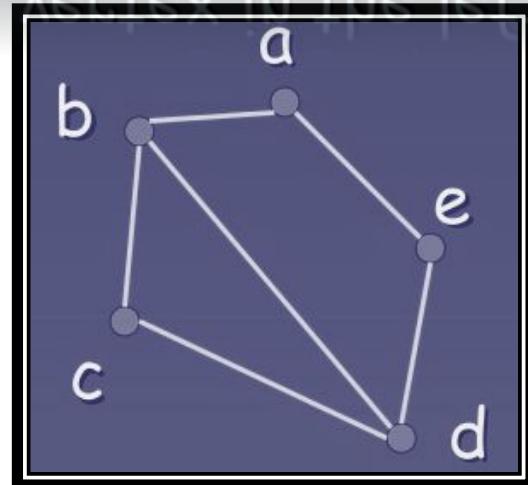
$f(c) = b$   $f(a) = e$   $f(s) = a$



## Example:

**Solution:** No, they are not isomorphic, because they differ in the degrees of their vertices.

Vertex d in right graph is of degree one, but there is no such vertex in the left graph.



# Table of Contents

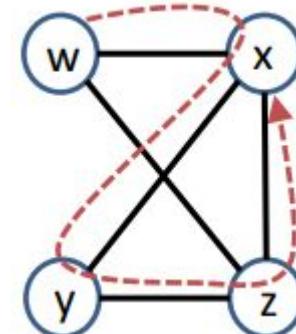
1. Path
2. Circuit
3. Connected Components
4. Cut vertices
5. Cut edge
6. Counting path between vertices
7. Euler path
8. Euler circuit
9. Hamiltonian path
10. Hamiltonian circuit
11. Graph coloring
12. Planar graph

## Path

A path is a sequence of edges that begins at a vertex, and travels from vertex to vertex along edges of the graph. The number of edges on the path is called the length of the path.

Example:

Consider the graph on the right.  $w \rightarrow x \rightarrow y \rightarrow z \rightarrow x$  corresponds to a path of length 4.



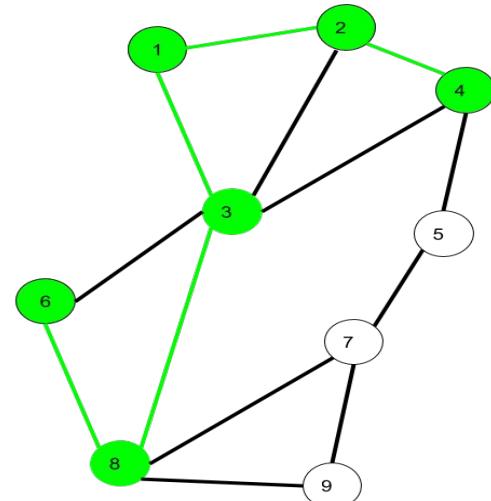
## Path

It is a trail in which neither vertices nor edges are repeated i.e. if we traverse a graph such that we do not repeat a vertex and nor we repeat an edge. As path is also a trail, thus it is also an open walk.

Vertex not repeated.

Edge not repeated.

Here  $6 \rightarrow 8 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 4$  is a Path.

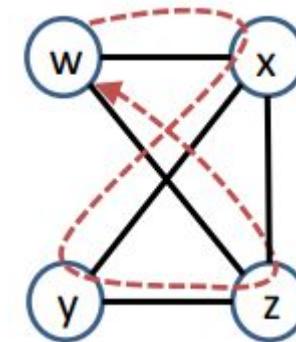


## Circuit

If a path begins and ends at the same vertex, the path is also called a circuit.

Example:

Consider the graph on the right.  $w \rightarrow x \rightarrow y \rightarrow z \rightarrow w$  corresponds to a circuit of length 4.

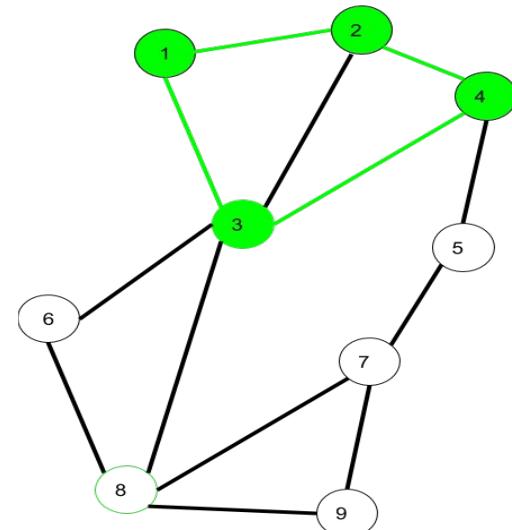


## Circuit/Cycle

Traversing a graph such that we do not repeat a vertex nor we repeat a edge but the starting and ending vertex must be same i.e. we can repeat starting and ending vertex only then we get a cycle.

Vertex not repeated.  
Edge not repeated.

Here  $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$  is a cycle.



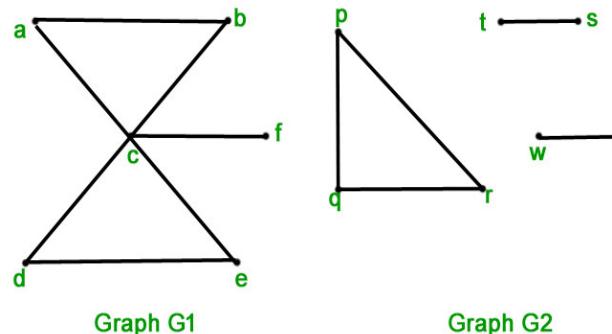
## Undirected Graph

A undirected graph is said to be connected if there is a path between every pair of distinct vertices of the graph. An undirected graph that is not connected is called disconnected.

## Connected Component

A connected component of a graph  $G$  is a connected subgraph of  $G$  that is not a proper subgraph of another connected subgraph of  $G$ .

For example, in the following diagram, graph  $G_1$  is connected and graph  $G_2$  is disconnected. Since  $G_1$  is connected there is only one connected component. But in case of  $G_2$  there are three connected components.



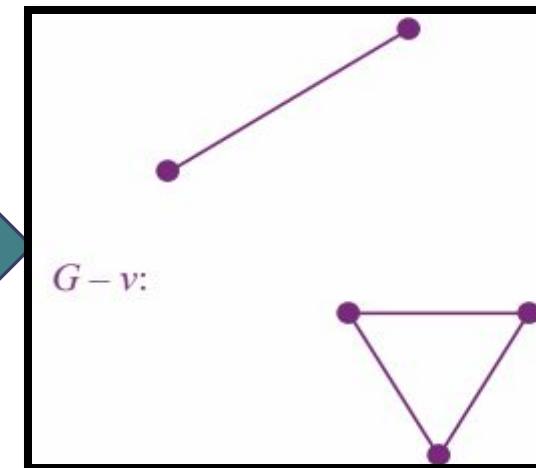
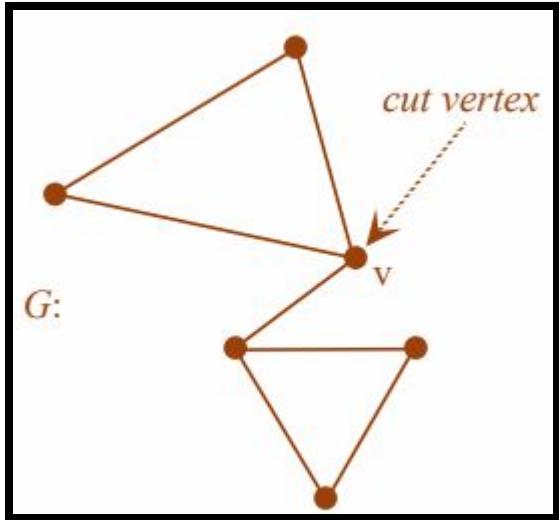
# Table of Contents

1. Path
2. Circuit
3. Connected Components
4. Cut vertices
5. Cut edge
6. Counting path between vertices
7. Euler path
8. Euler circuit
9. Hamiltonian path
10. Hamiltonian circuit
11. Graph coloring
12. Planar graph

## Cut Vertices

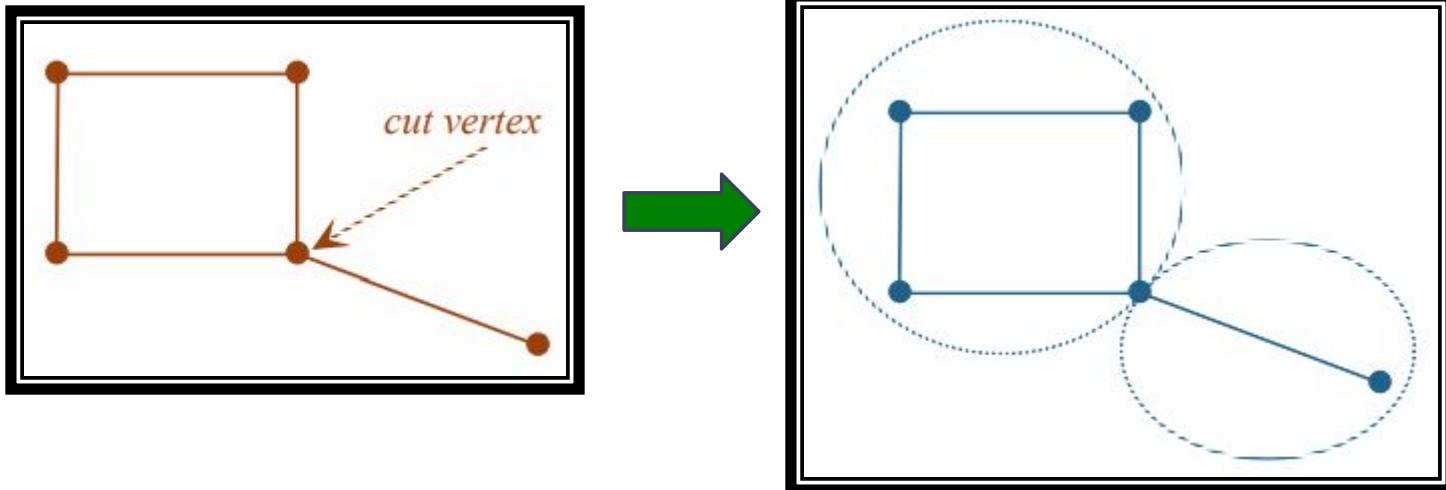
A vertex  $v$  of a graph  $G$  is a **cut vertex** or an **articulation vertex** of  $G$  if the graph  $G - v$  consists of a greater number of components than  $G$ .

Example.  $v$  is a cut vertex of the graph below:



A graph is **separable** if it is not connected or if there exists at least one cut vertex in the graph. Otherwise, the graph is **nonseparable**.

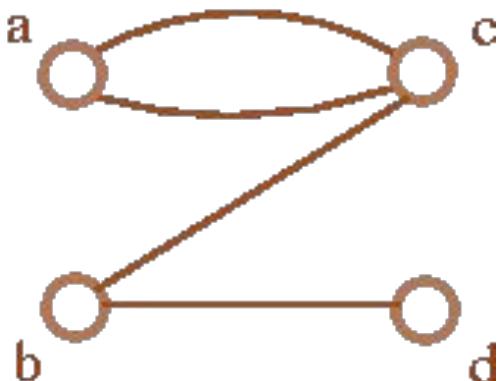
Example. The graph below is separable:



## Cut Edges/Bridge

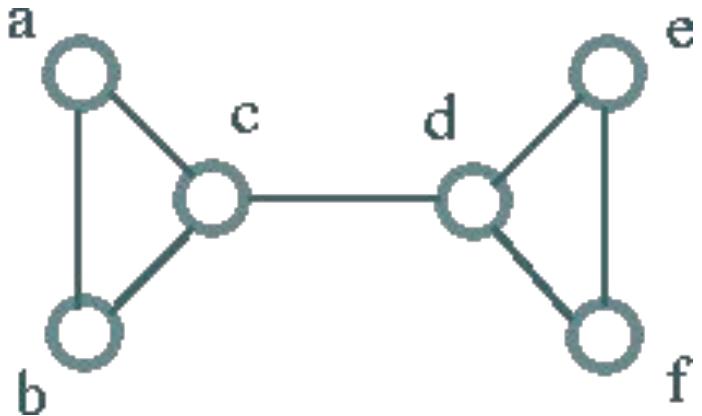
An **edge** whose removal produces a graph with more connected components than in the original graph is called **cut edge**.

Example



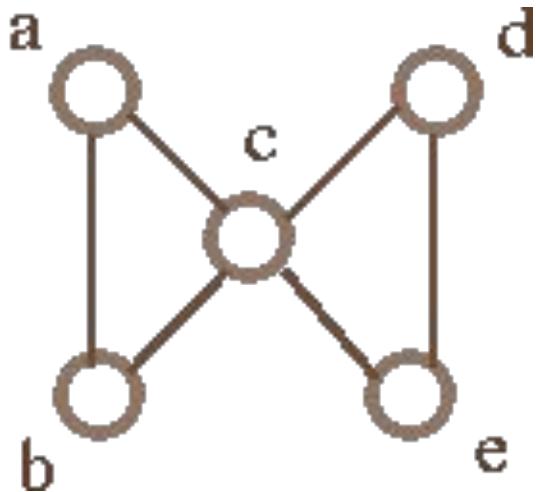
The graph  $G_1$  can be split up into two components by removing one of the edges  $\{b, c\}$  or  $\{b, d\}$ . Therefore, edge  $\{b, c\}$  or  $\{b, d\}$  is a bridge.

## Example



The graph  $G_2$  can be disconnected by removing a single edge,  $\{c, d\}$ . Therefore, edge  $\{c, d\}$  is a bridge.

## Example

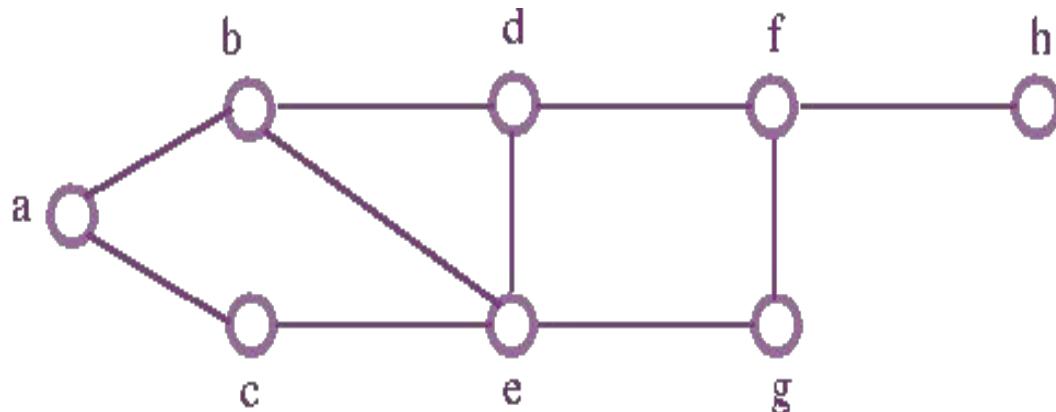


The graph  $G_3$  cannot be disconnected by removing a single edge, but the removal of two edges (such as  $\{a, c\}$  and  $\{b, c\}$ ) disconnects it.

## Vertex Cut

A cut-vertex is a single vertex whose removal disconnects a graph. It is important to note that the above definition breaks down if  $G$  is a complete graph, since we cannot then disconnect  $G$  by removing vertices.

Example:

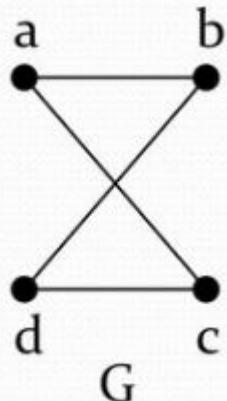


Vertex cutset  
of  $G$  is  $\{b, e\}$ .

## Counting path between vertices

How many path of length four are there from a to d in the simple graph  $G$ ?

Example:



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad A^4 = \begin{bmatrix} 8 & 0 & 0 & 8 \\ 0 & 8 & 8 & 0 \\ 0 & 8 & 8 & 0 \\ 8 & 0 & 0 & 8 \end{bmatrix}$$

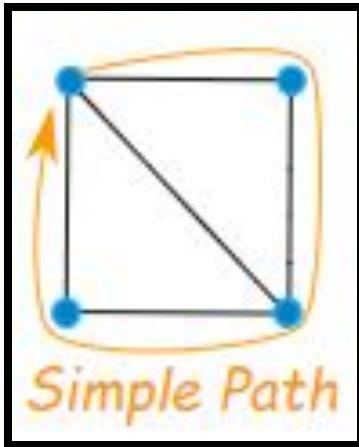
$a \rightarrow b \rightarrow a \rightarrow b \rightarrow d$   
 $a \rightarrow b \rightarrow a \rightarrow c \rightarrow d$   
 $a \rightarrow b \rightarrow d \rightarrow b \rightarrow d$   
 $a \rightarrow b \rightarrow d \rightarrow c \rightarrow d$   
 $a \rightarrow c \rightarrow a \rightarrow b \rightarrow d$   
 $a \rightarrow c \rightarrow a \rightarrow c \rightarrow d$   
 $a \rightarrow c \rightarrow d \rightarrow b \rightarrow d$   
 $a \rightarrow c \rightarrow d \rightarrow c \rightarrow d$

There are eight paths from a to d.

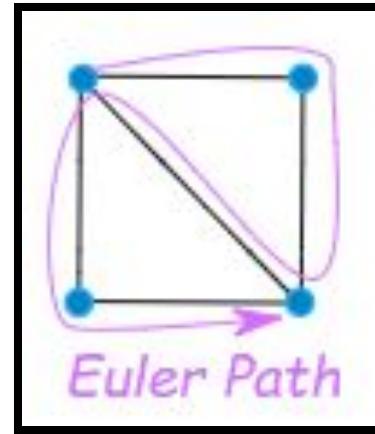
# Table of Contents

1. Path
2. Circuit
3. Connected Components
4. Cut vertices
5. Cut edge
6. Counting path between vertices
7. Euler path
8. Euler circuit
9. Hamiltonian path
10. Hamiltonian circuit
11. Graph coloring
12. Planar graph

## Difference between simple path and Euler path



A route around a graph that visits **every vertex** once is called a **simple path**.

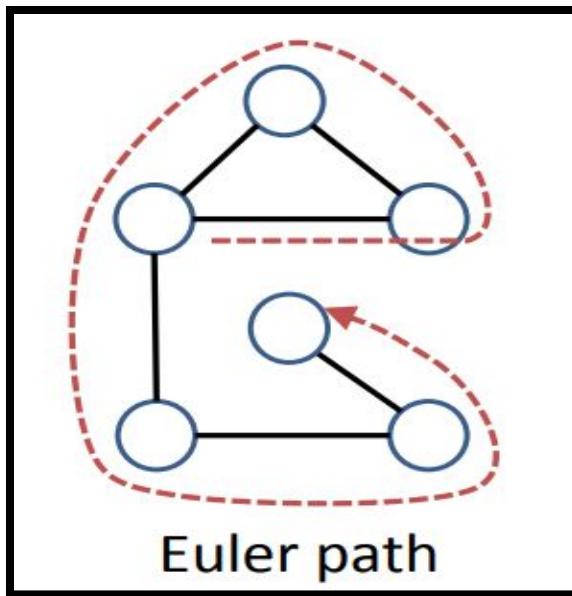


A route around a graph that visits **every edge** once is called an **Euler path**.

## Euler Path

An **Euler path** in a multigraph  $G$  is a simple path that contains every edge of  $G$ . (So, every edge occurs **exactly once** in the path.)

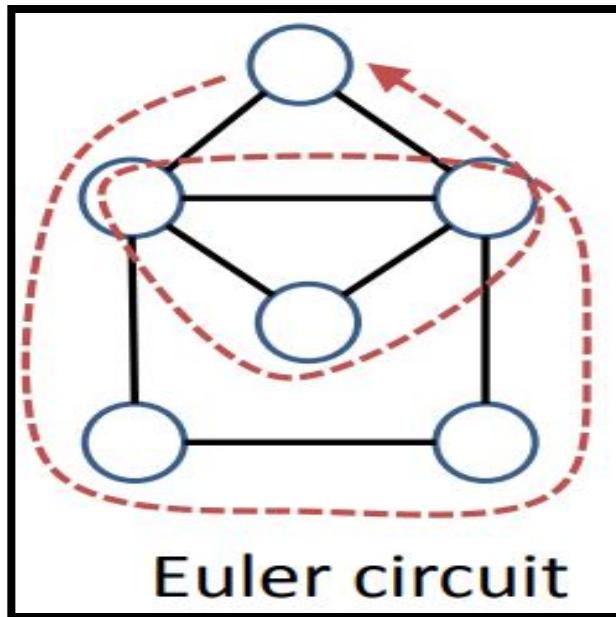
Example:



# Euler Circuit

An **Euler circuit** in an multigraph  $G$  is a simple circuit that contains **every edge** of  $G$ . (So, every edge occurs **exactly once** in the circuit.)

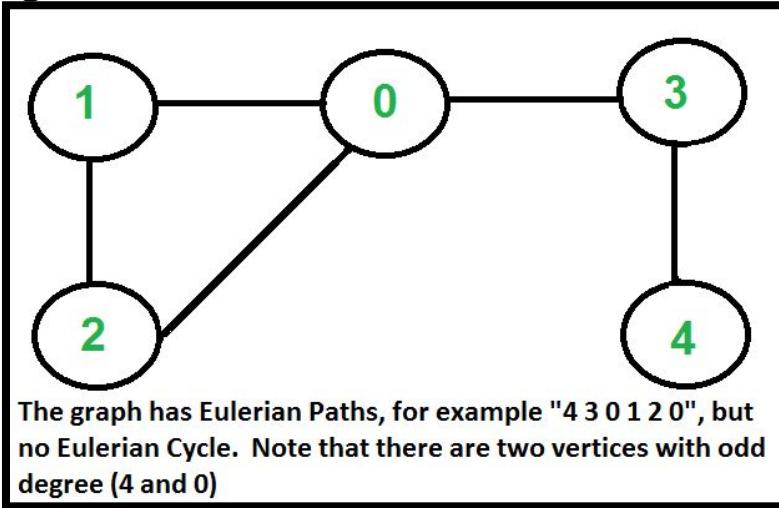
Example:



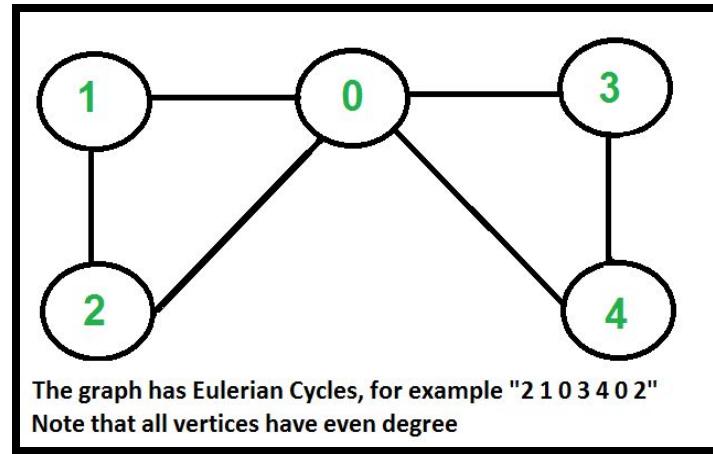
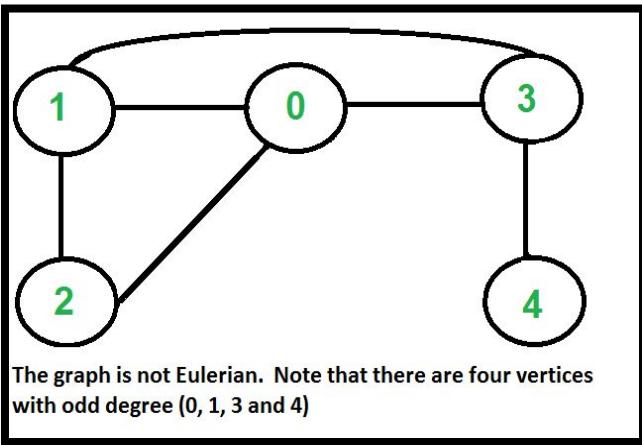
## Important points related to Euler Path and Circuit

- A graph has an Euler circuit if and only if the degree of every vertex is even.
- A graph has an Euler path if and only if there are at most two vertices with odd degree.

Example:



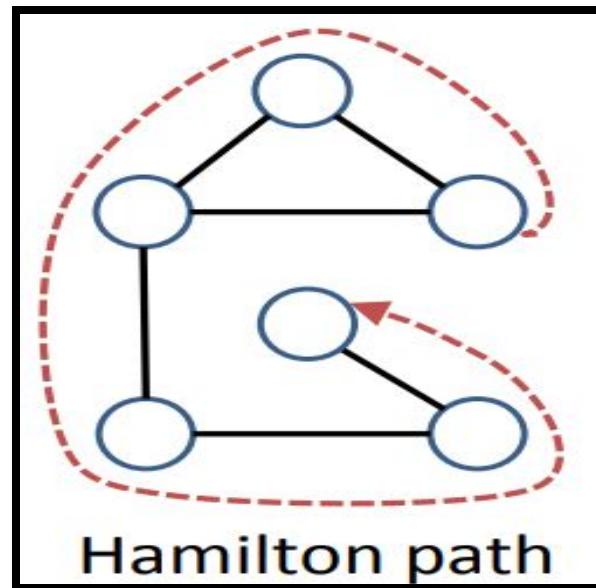
## Examples:



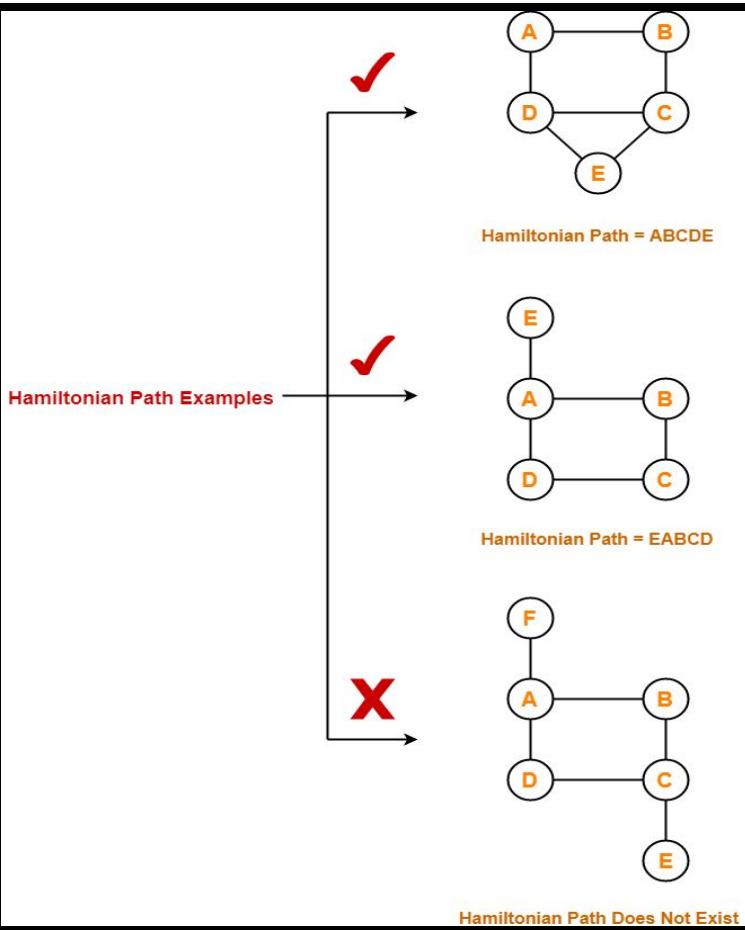
## Hamiltonian Path

A **Hamiltonian path** in a (undirected) graph  $G$  is a simple path that visits every **vertex** exactly **once**. (In other words, it is a tidy path that visits every vertex.)

Example:



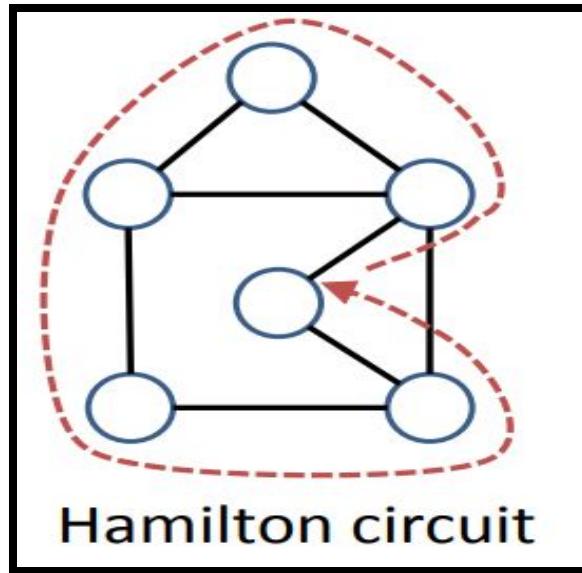
## Examples:



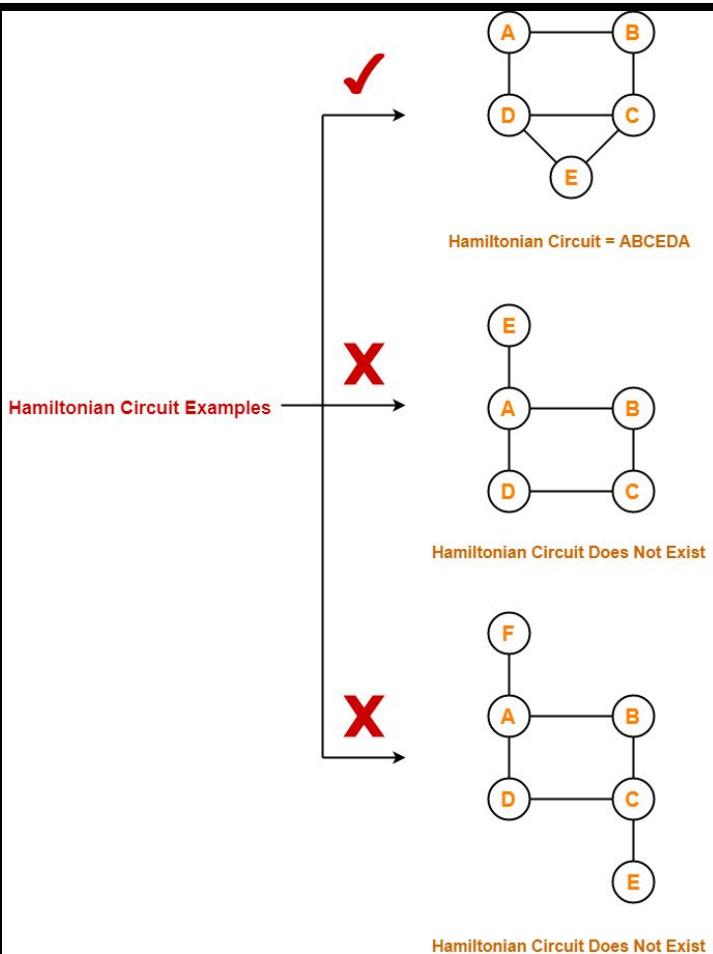
# Hamiltonian Circuit

A **Hamiltonian circuit** in a (undirected) graph  $G$  is a simple circuit that passes through every vertex exactly **once** (except for the common start and end vertex, which is seen exactly twice).

Example:



## Examples:



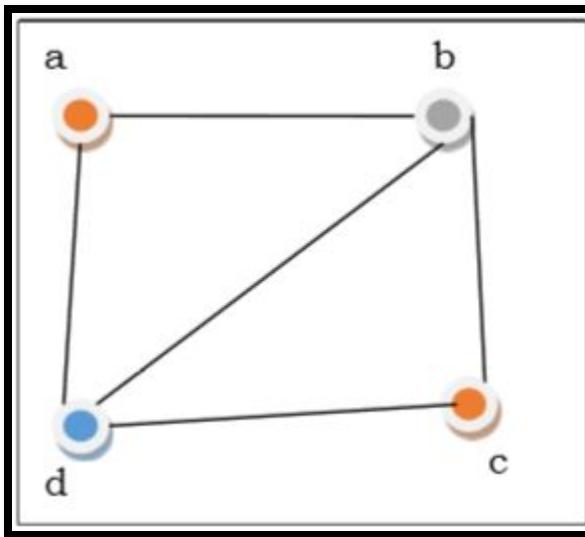
# Table of Contents

1. Path
2. Circuit
3. Connected Components
4. Cut vertices
5. Cut edge
6. Counting path between vertices
7. Euler path
8. Euler circuit
9. Hamiltonian path
10. Hamiltonian circuit
11. Graph coloring
12. Planar graph

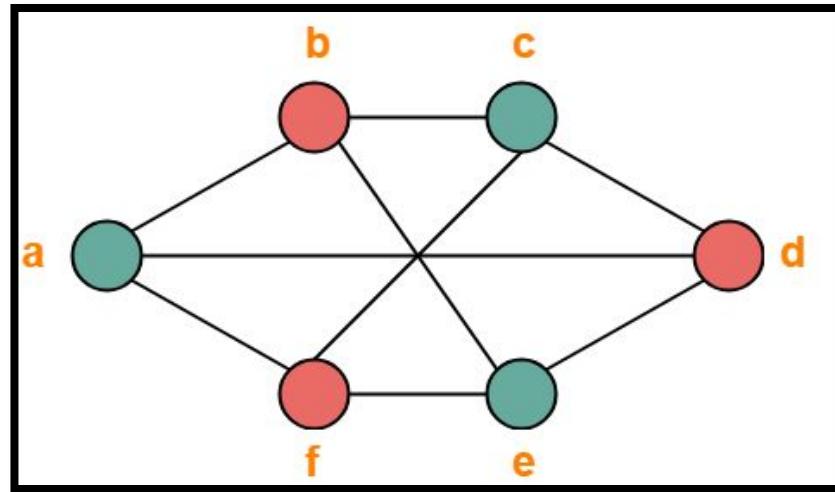
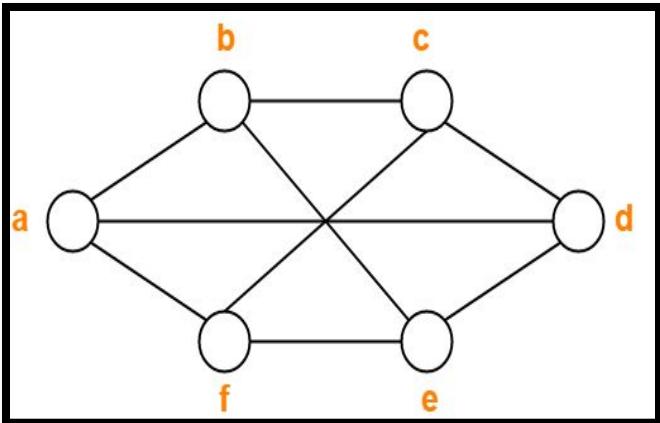
# Graph Coloring

**Graph coloring** is the procedure of assignment of colors to each **vertex** of a **graph**  $G$  such that no adjacent vertices get same **color**. The smallest number of colors required to **color** a **graph**  $G$  is called its **chromatic number** of that **graph**. It is denoted by  $\chi(G)$ .

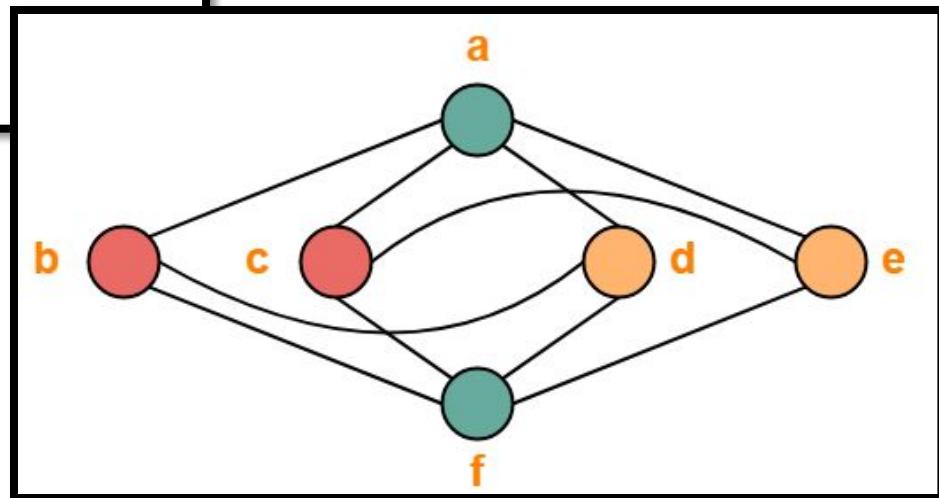
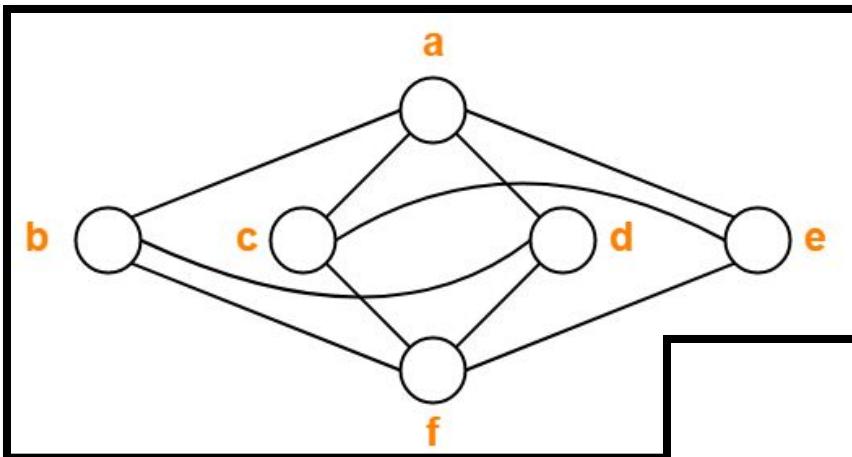
Example:



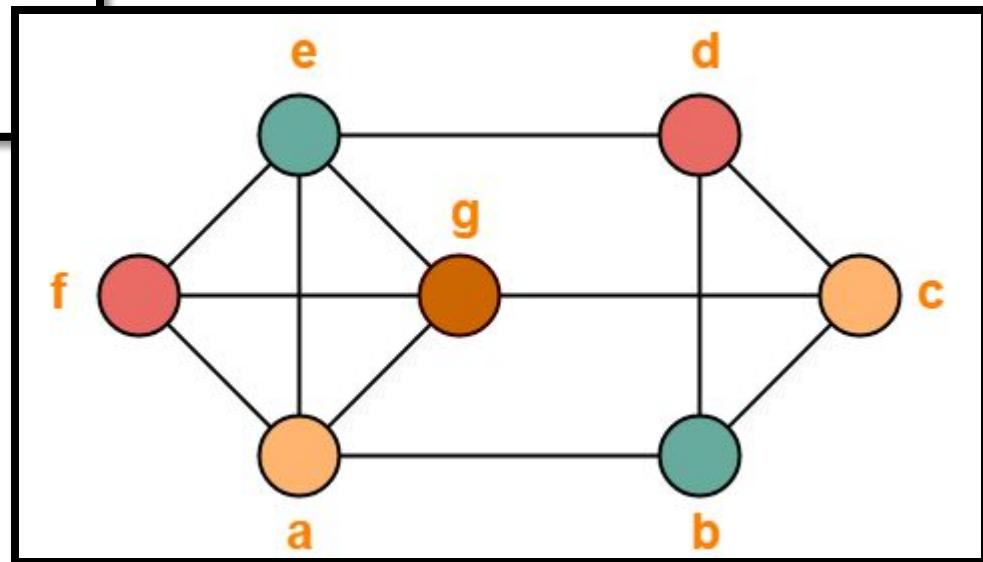
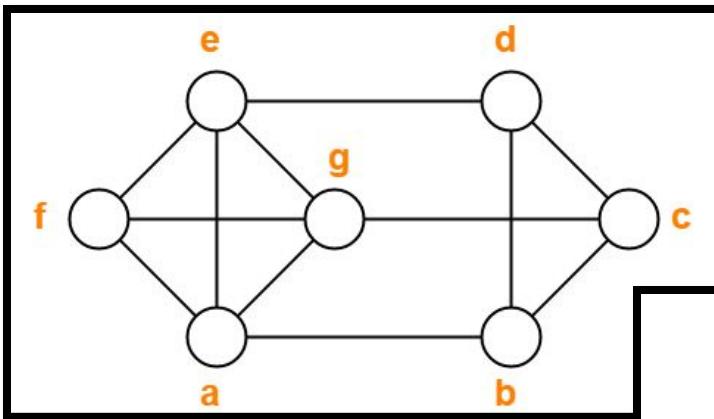
What is the chromatic number for the following graph?



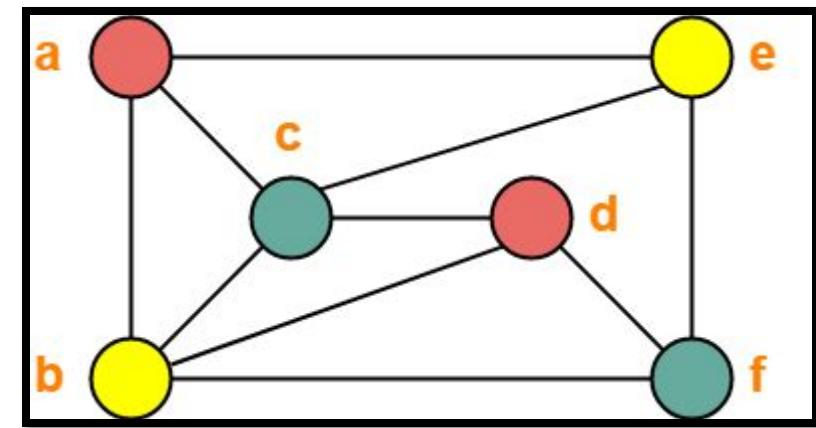
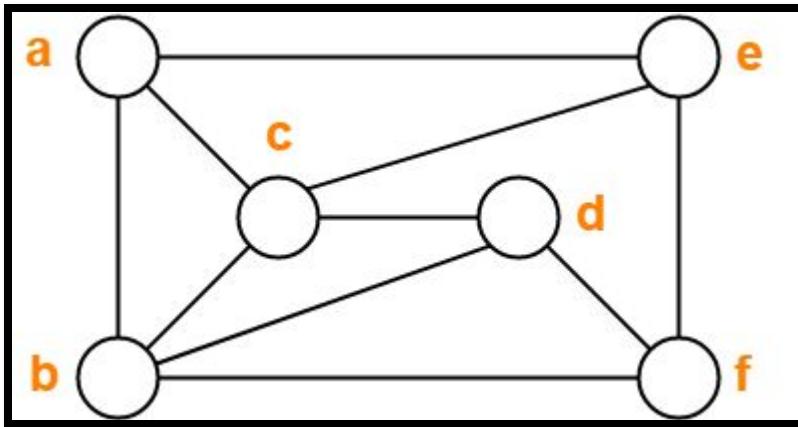
What is the chromatic number for the following graph?

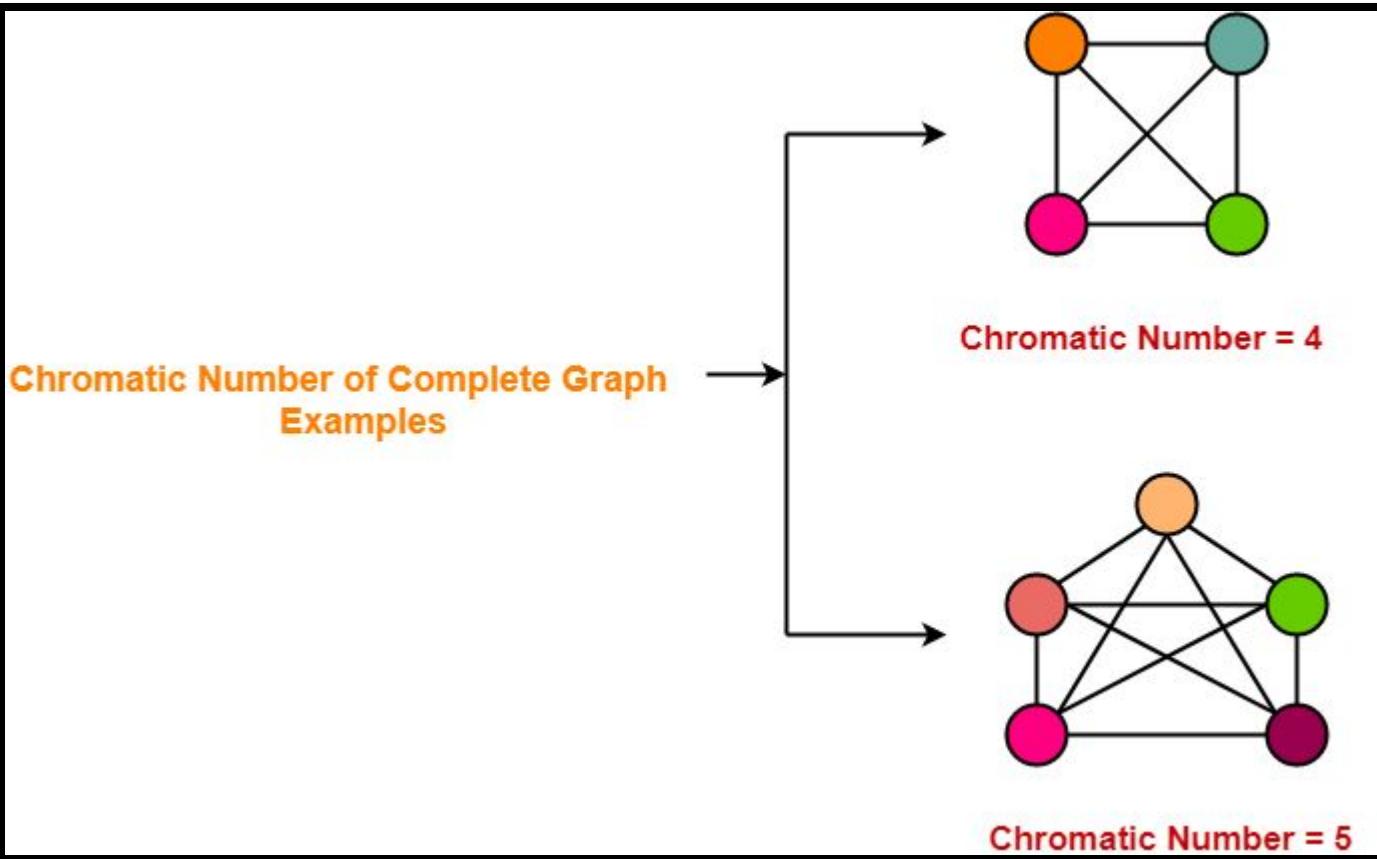


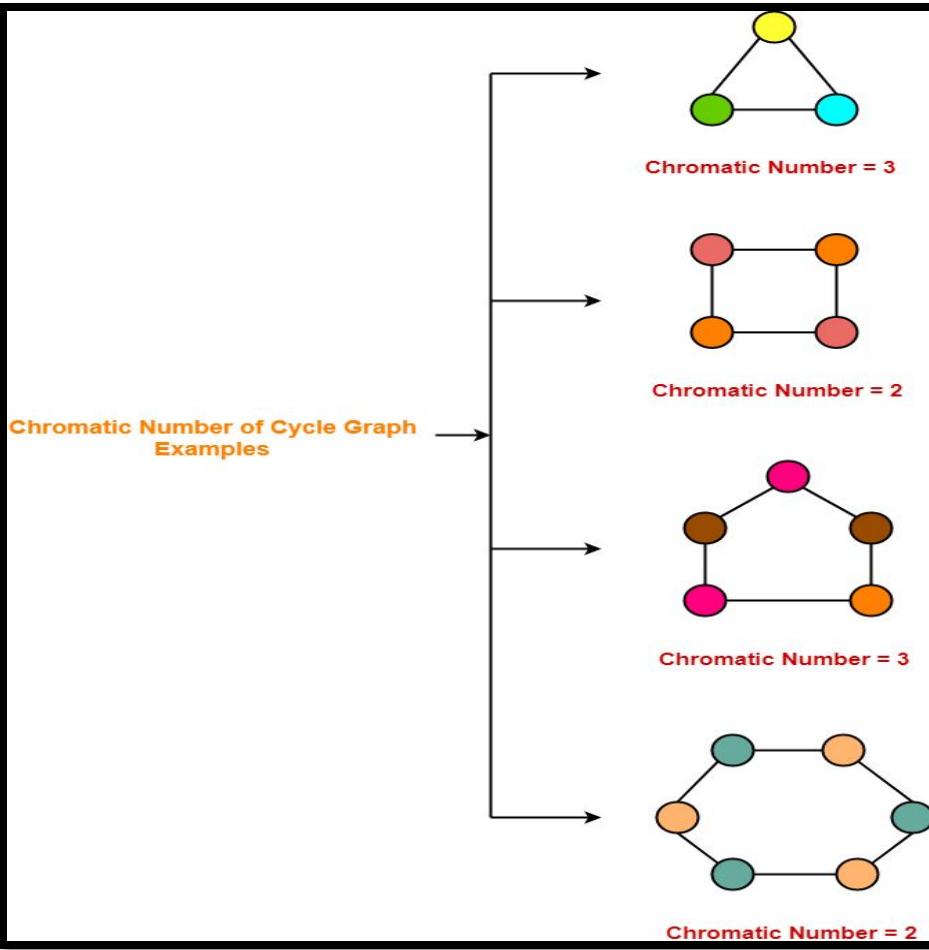
What is the chromatic number for the following graph?



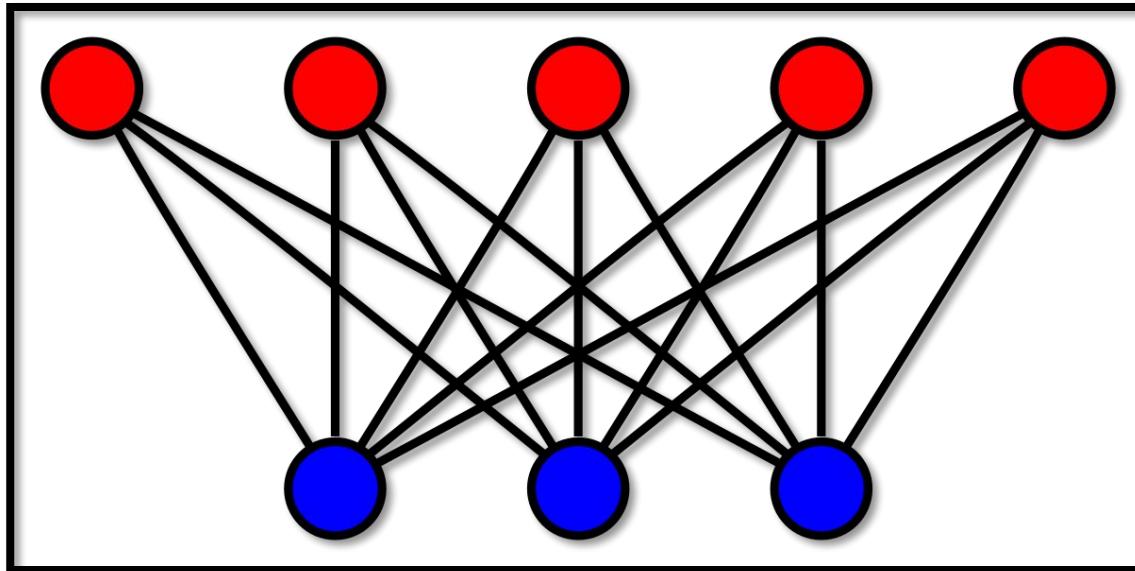
What is the chromatic number for the following graph?







What is the chromatic number of the complete bipartite graph  $K_{m,n}$  where m and n are positive integers?



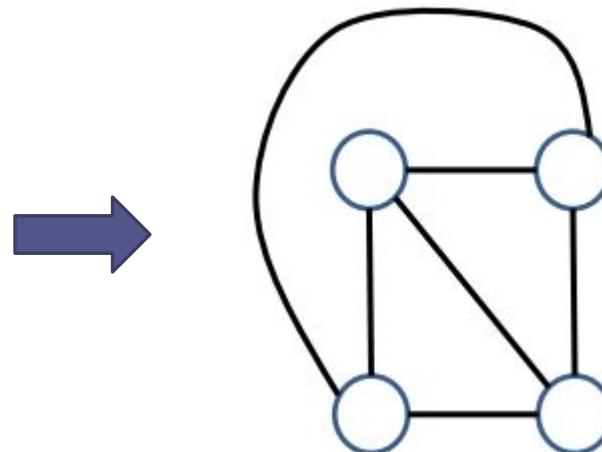
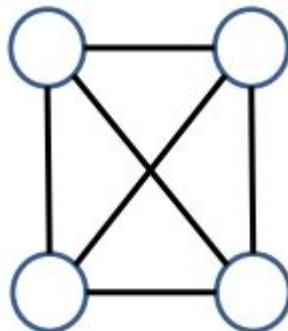
# Table of Contents

1. Path
2. Circuit
3. Connected Components
4. Cut vertices
5. Cut edge
6. Counting path between vertices
7. Euler path
8. Euler circuit
9. Hamiltonian path
10. Hamiltonian circuit
11. Graph coloring
12. Planar graph

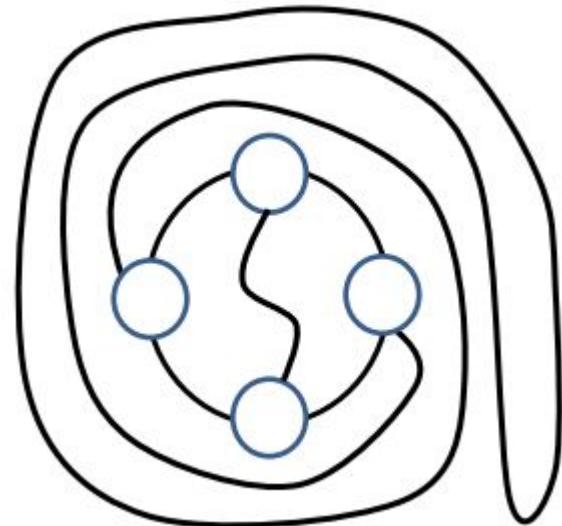
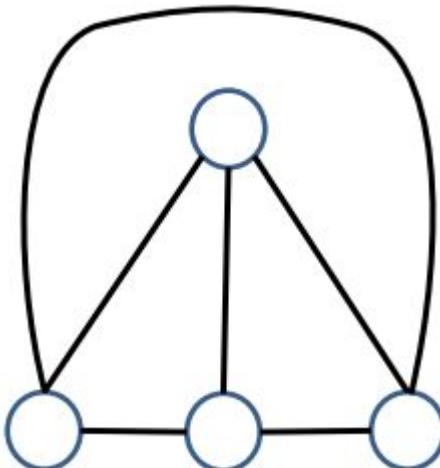
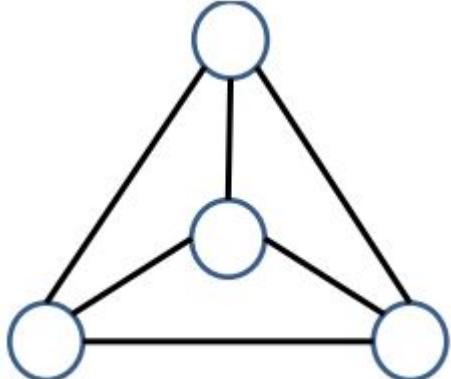
## Planar Graph

A graph is called planar if it can be drawn in a plane without any edges crossing. Such a drawing is called a planar representation of the graph.

Example:  $K_4$  is a planar graph

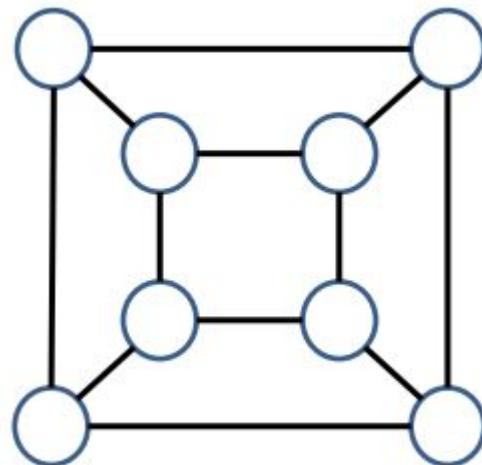
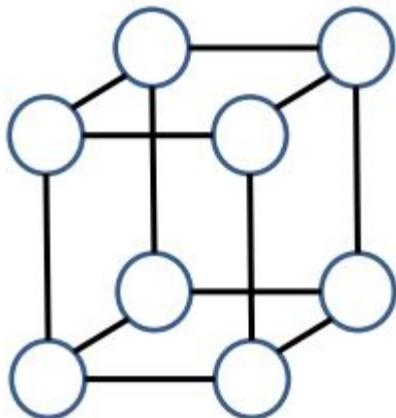


## Other planar representations of $K_4$



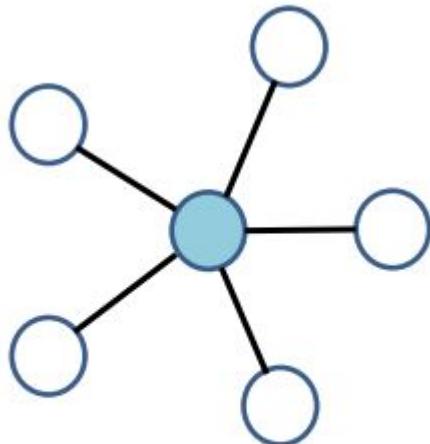
## Examples of Planar Graph

$Q_3$  is a planar graph.

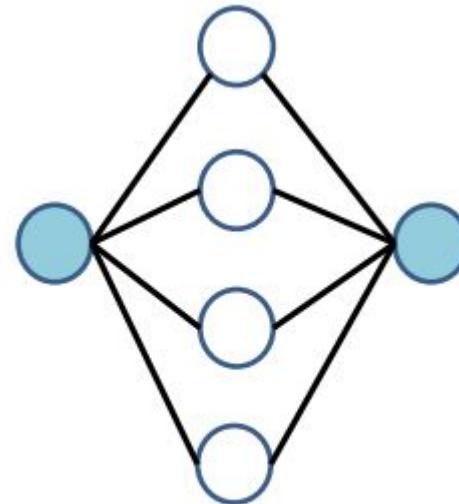


## Examples of Planar Graph

$K_{1,n}$  and  $K_{2,n}$  are planar graphs for all n.



$K_{1,5}$



$K_{2,4}$

## Euler's Formula

Let  $G$  be a connected planar simple graph with  $e$  edges and  $v$  vertices. Let  $r$  be the number of regions in a planar representation of  $G$ . Then

$$r = e - v + 2$$

**Example:** Suppose that a connected planar simple graph has 20 vertices, each of degree 3. Into how many regions does a representation of this planar graph split the plane?

**Solution:** There is a formula for calculating the number of regions

$$\text{number of regions} = e - v + 2$$

$$\text{Sum of degrees of edges} = 20 * 3 = 60 = 2 * \text{number of edges}$$

$$2 * e = 60$$

$$e = 30$$

$$\text{Regions} = 30 - 20 + 2 = 12$$

# Table of Contents

1. Shortest path algorithm (Dijkstra's Algorithm)
2. Minimum Spanning Tree (Kruskal's Algorithm)
3. Minimum Spanning Tree (Prim's Algorithm)
4. All Pair Shortest Path (Floyd Warshall's Algorithm)

## Shortest path algorithm (Dijkstra's Algorithm)

- An algorithm for finding the shortest path between two vertices in a graph.
- It Works on both directed and undirected graphs. However, all edges must have nonnegative weights.
- A weighted graph associates a label (weight) with every edge in the graph. Weights are usually real numbers.

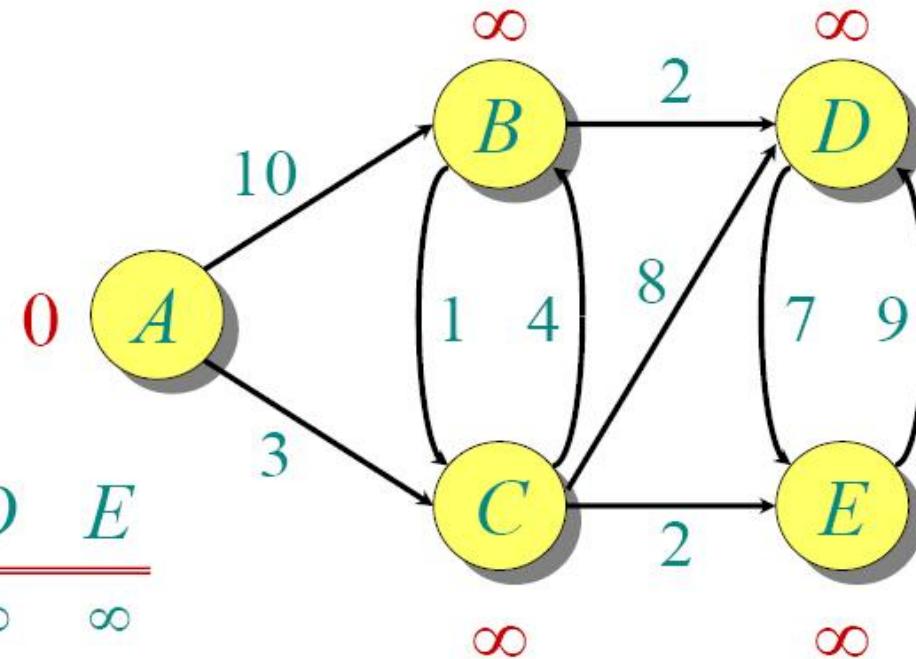
## Shortest path algorithm (Dijkstra's Algorithm)

- Google Maps are the examples of real life networks.
- In any of the map each town is a vertex (node) and each road is an edge (arc).
- In real life we often want to know what is the shortest path between two places.
- The Computer cannot decide which route is the best, so it uses an algorithm to do so.
- One such algorithm is Dijkstra's.

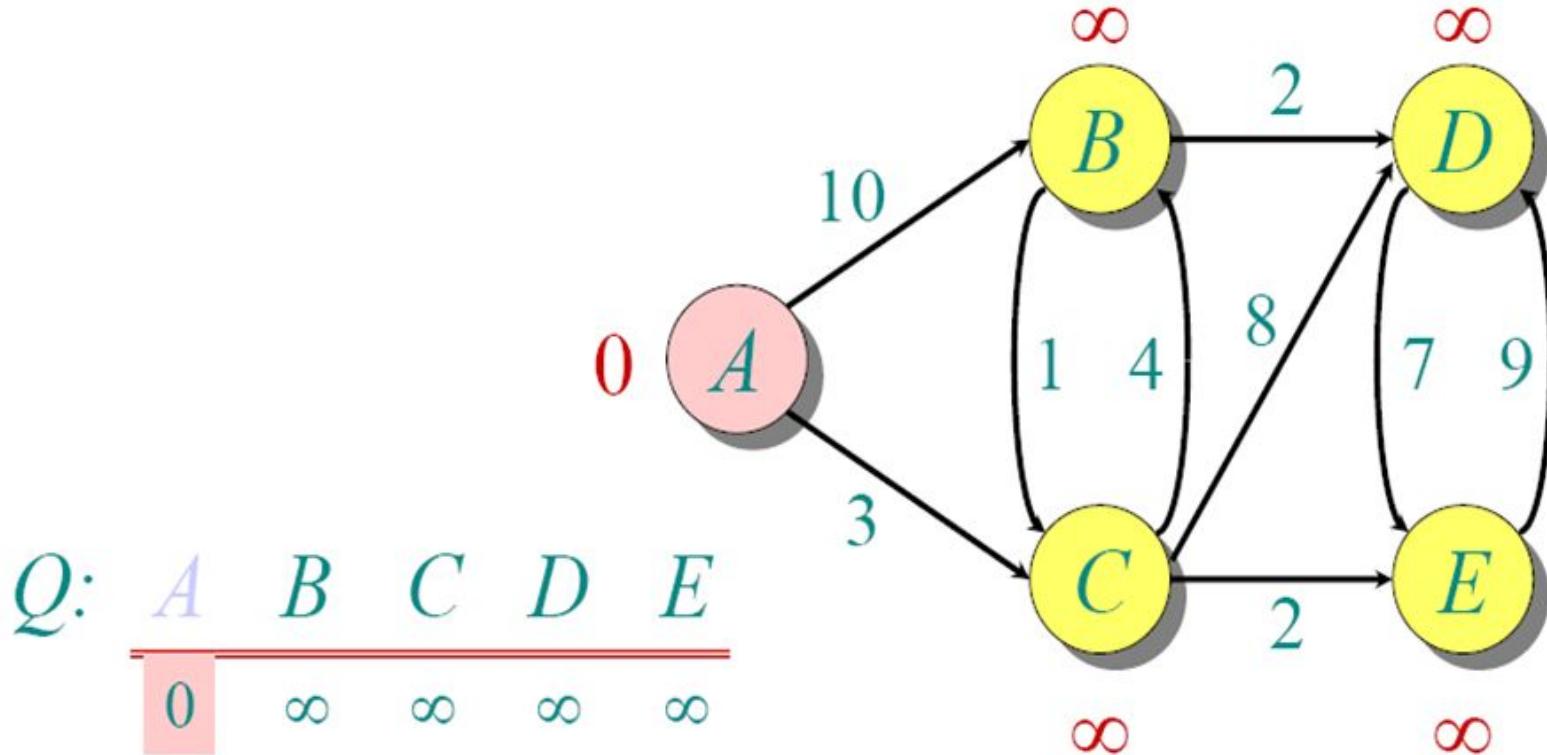
# Dijkstra's Algorithm

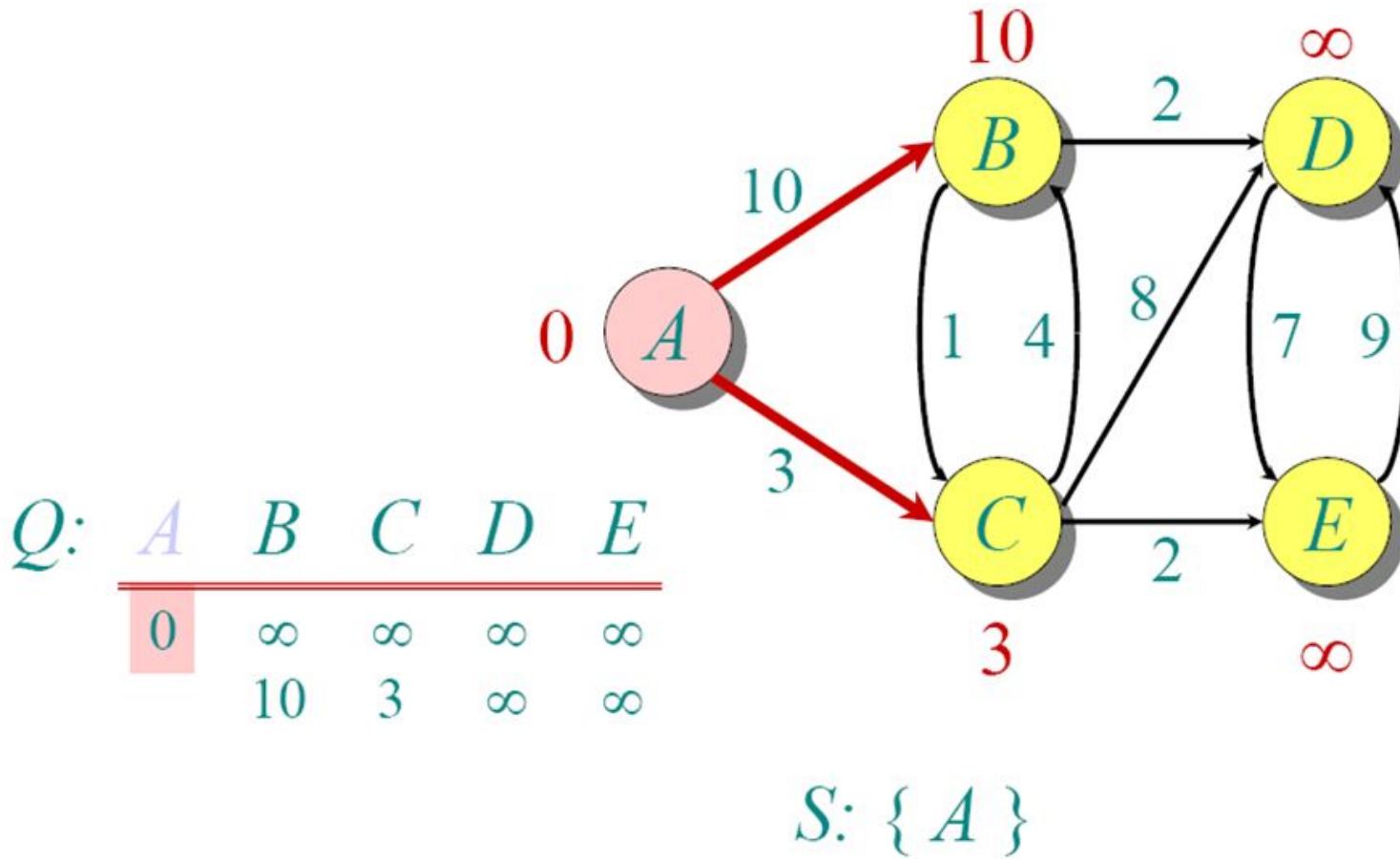
**Initialize:**

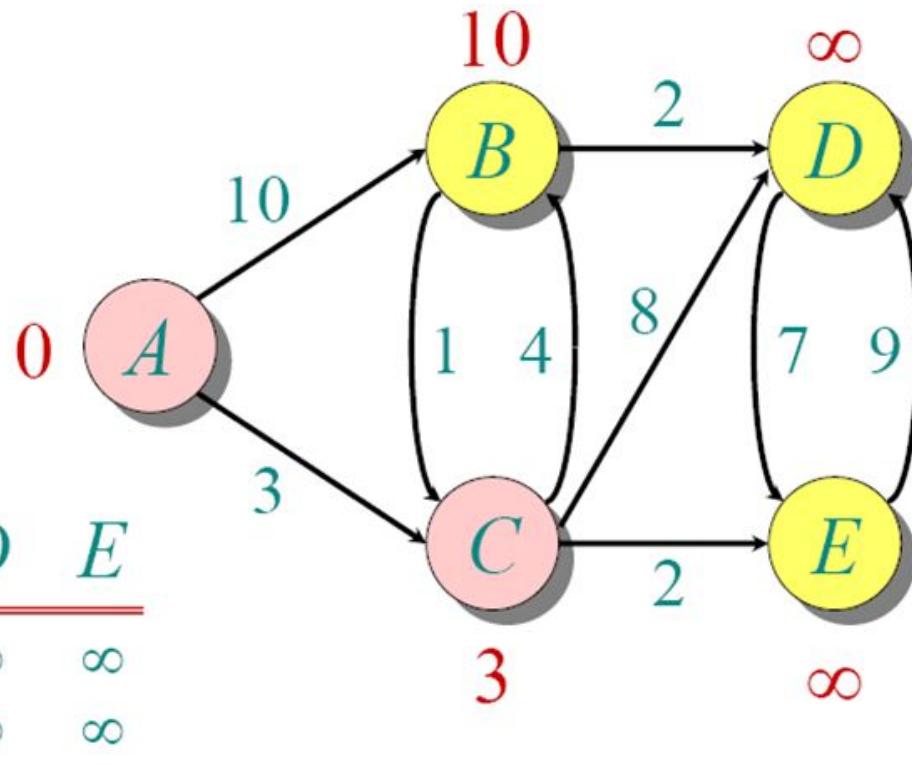
$$Q: \frac{A \quad B \quad C \quad D \quad E}{0 \quad \infty \quad \infty \quad \infty \quad \infty}$$



$$S: \{\}$$

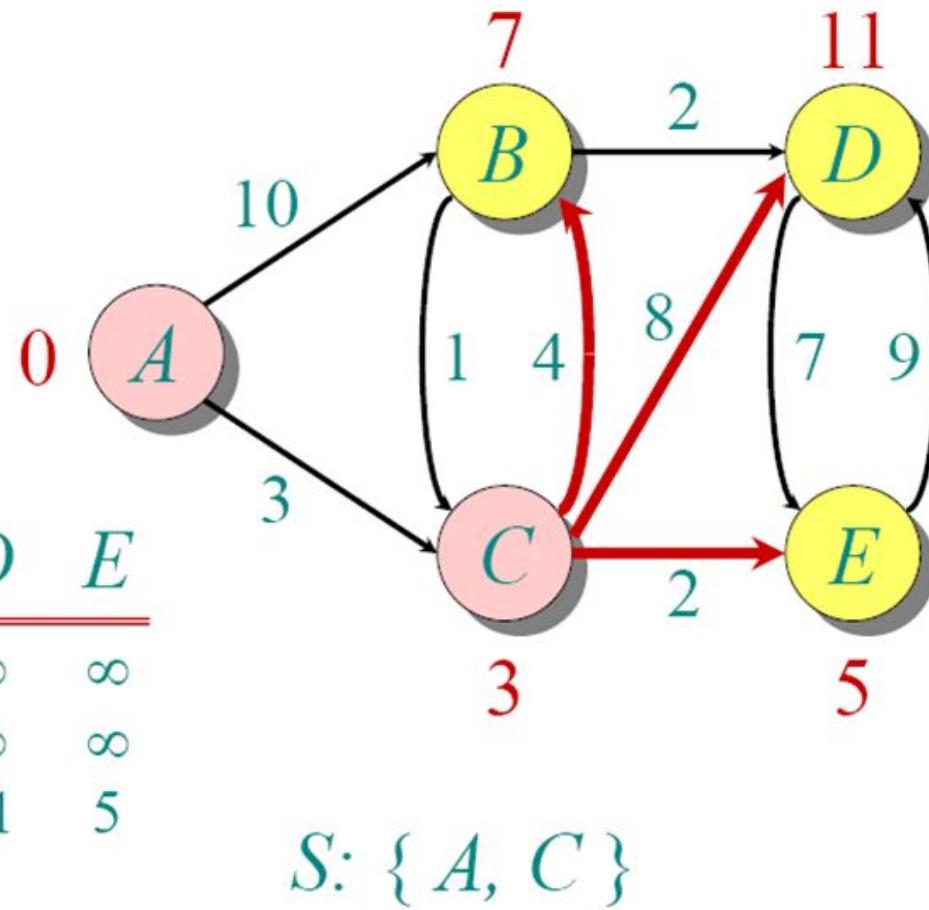


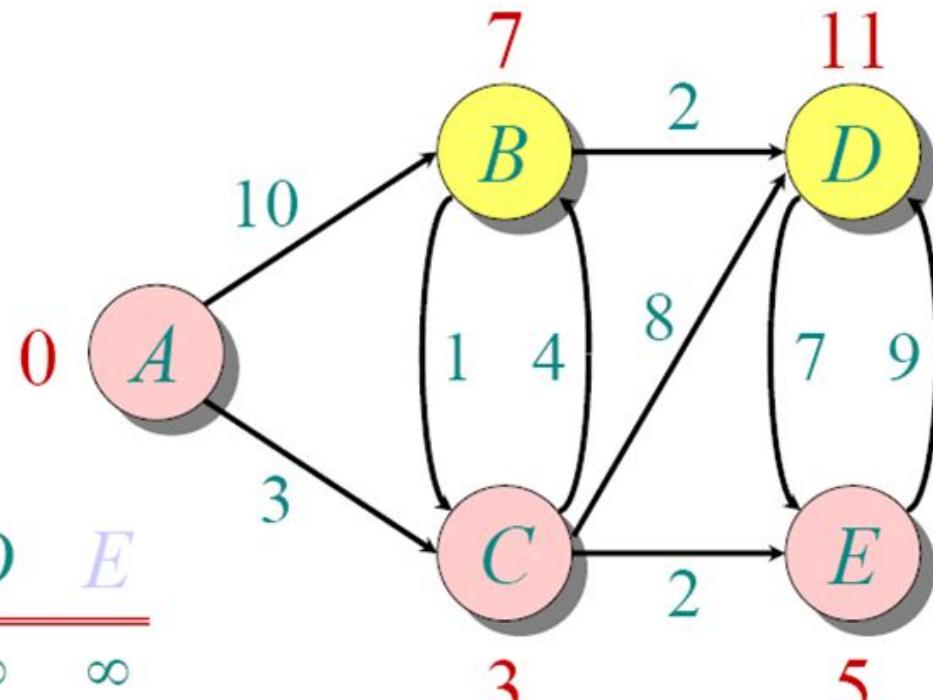




$Q:$	$A$	$B$	$C$	$D$	$E$
	0	$\infty$	$\infty$	$\infty$	$\infty$
	10	3	$\infty$	$\infty$	

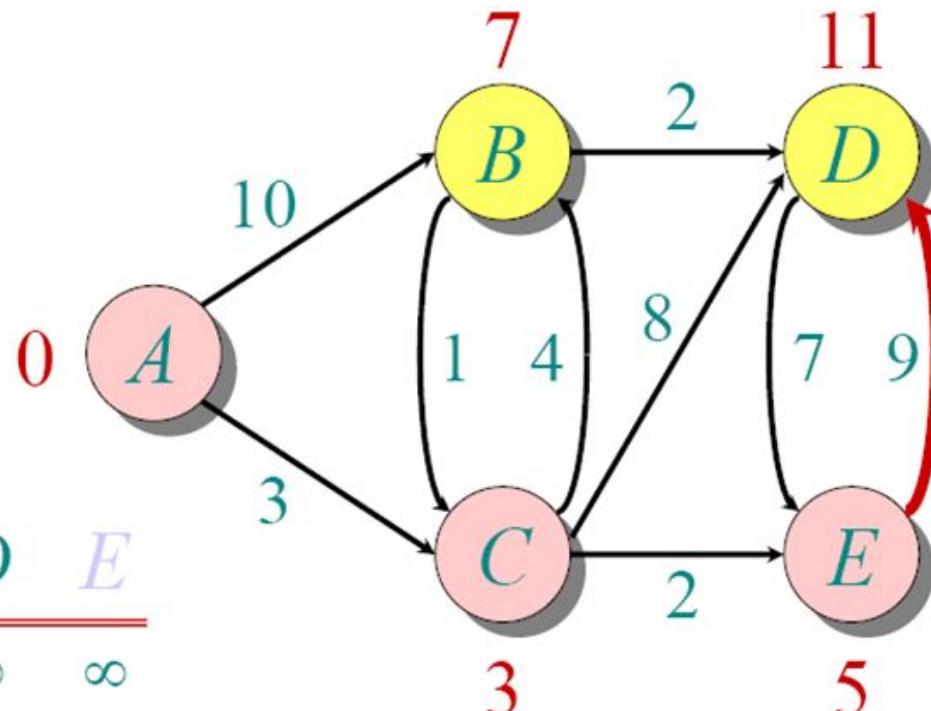
$S: \{ A, C \}$





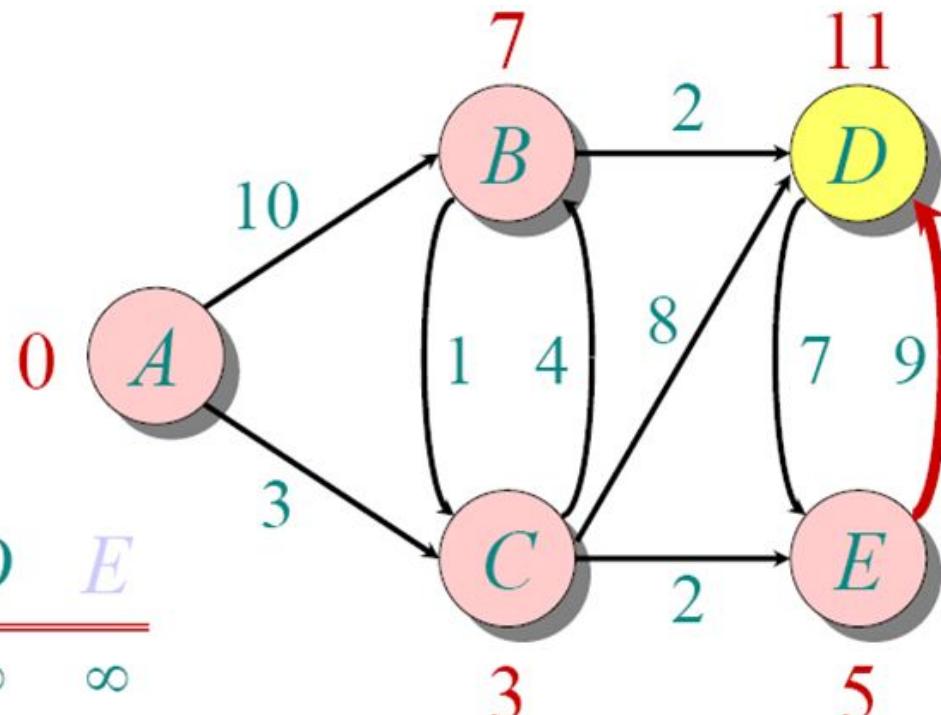
$Q:$	$A$	$B$	$C$	$D$	$E$
0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
10		3	$\infty$	$\infty$	
7		11	5		

$S: \{ A, C, E \}$



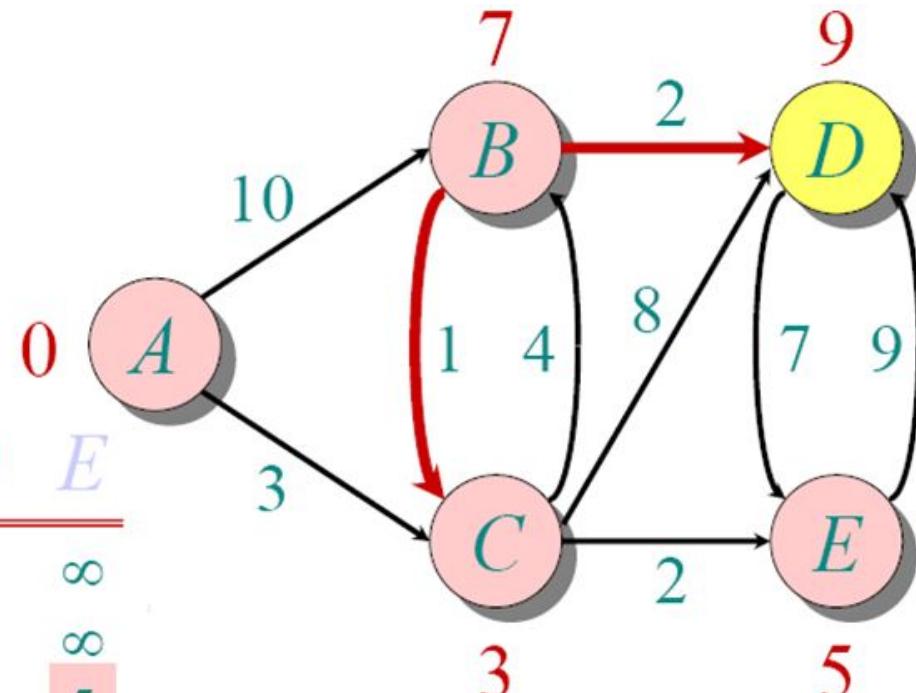
$Q:$	$A$	$B$	$C$	$D$	$E$
0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
10		3	$\infty$	$\infty$	
7		11	5		
7		11			

$S: \{ A, C, E \}$



$Q:$	$A$	$B$	$C$	$D$	$E$
0	0	$\infty$	$\infty$	$\infty$	$\infty$
10		3	$\infty$	$\infty$	
7	7		11	5	

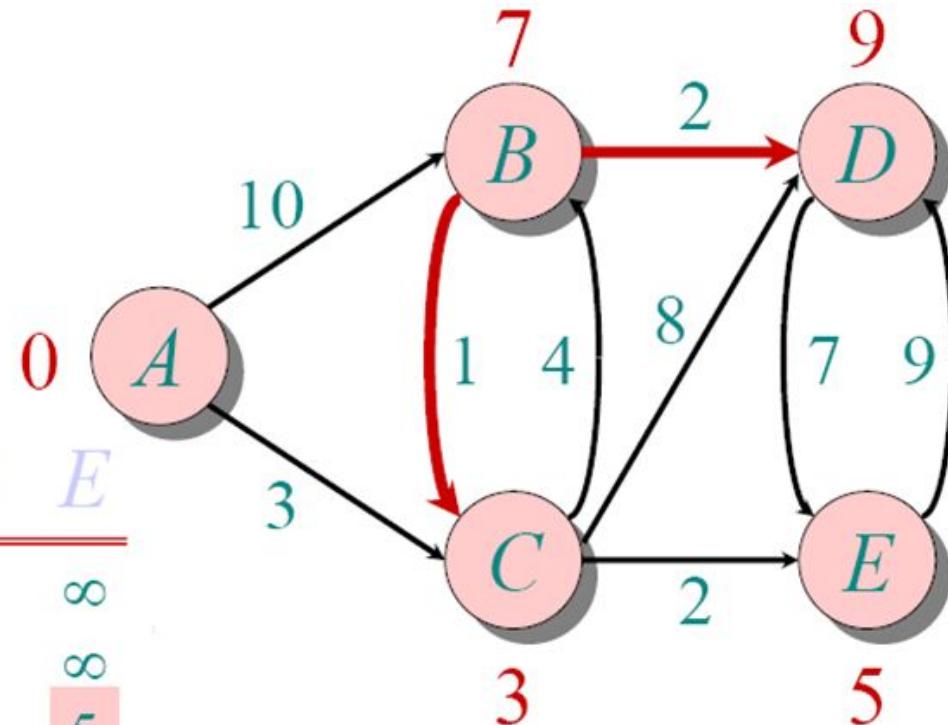
$S: \{ A, C, E, B \}$



$A$	$B$	$C$	$D$	$E$
0	$\infty$	$\infty$	$\infty$	$\infty$
10	3	$\infty$	$\infty$	
7	7	11	5	

$S: \{ A, C, E, B \}$

$Q:$	$A$	$B$	$C$	$D$	$E$
0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
10		3	$\infty$	$\infty$	
7	7		11	5	



$S: \{ A, C, E, B, D \}$

# Table of Contents

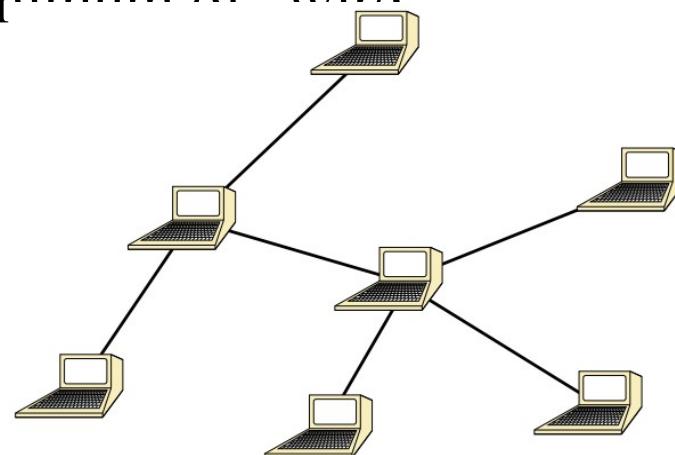
1. Shortest path algorithm (Dijkstra's Algorithm)
2. Minimum Spanning Tree (Kruskal's Algorithm)
3. Minimum Spanning Tree (Prim's Algorithm)
4. All Pair Shortest Path (Floyd Warshall's Algorithm)

## Minimum Spanning Tree

- A **tree** is an acyclic, undirected, connected graph.
- A **spanning tree** of a graph is a tree containing all vertices from the graph.
- A **minimum spanning tree** is a spanning tree, where the sum of the weights on the tree's edges are minimal.

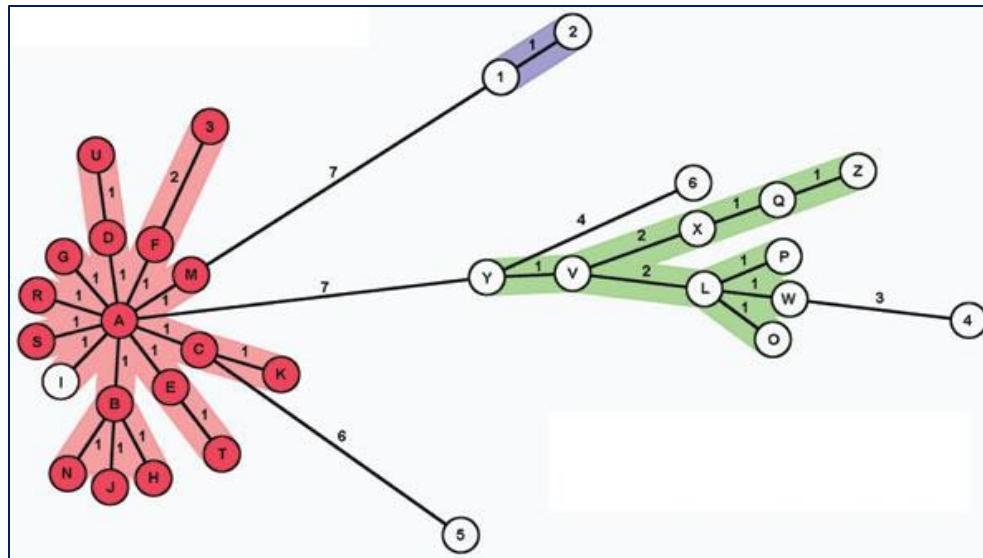
## Minimum Spanning Tree

- Find a minimum-cost set of edges that connect all vertices of a graph
- Applications
  - Connect “nodes” with a minimum of “wires”
    - Networking
    - Circuit design



# Minimum Spanning Tree

- Applications
    - Collect nearby nodes
      - Clustering, taxonomy construction



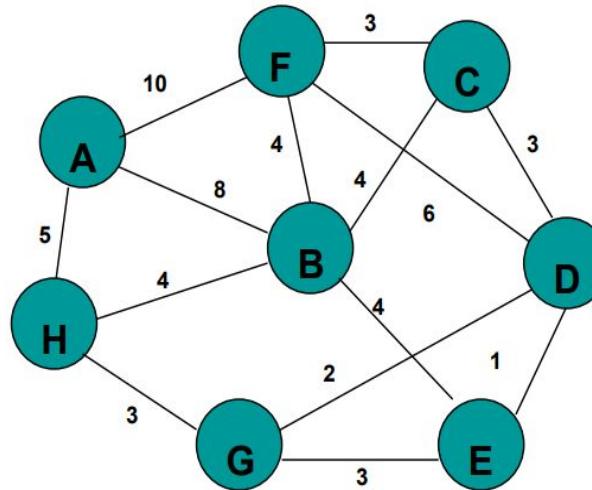
## Minimum Spanning Tree

- Both Kruskal's and Prim's Algorithms work with undirected graphs
- Both work with weighted and unweighted graphs but are more interesting when edges are weighted.
- Both are greedy algorithms that produce optimal solutions.

# Minimum Spanning Tree

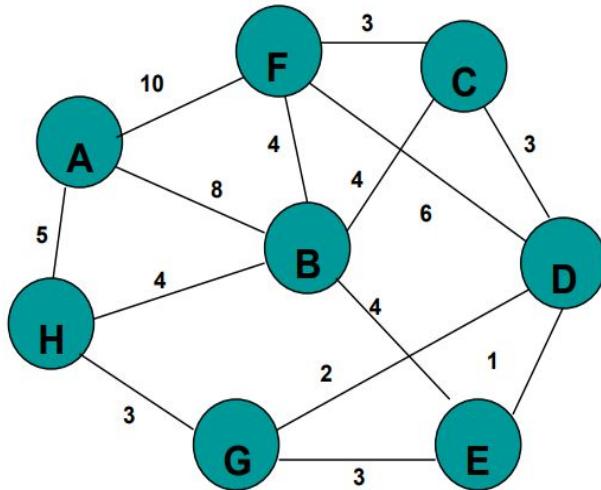
Kruskal's algorithm:

- Work with edges
- Two steps:
  - Sort edges by increasing edge weight
  - Select the first  $|V| - 1$  edges that do not generate a cycle
- Walk through:



# Kruskal's algorithm: Example

Sort the edges by increasing edge weight.

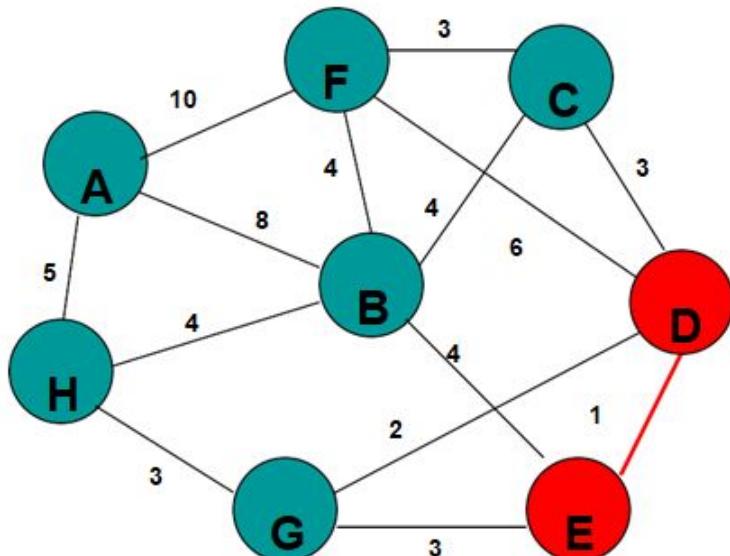


edge	$d_v$	
(D,E)	1	
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	$d_v$	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

# Kruskal's algorithm: Example

Select first  $|V|-1$  edges which do not generate a cycle.

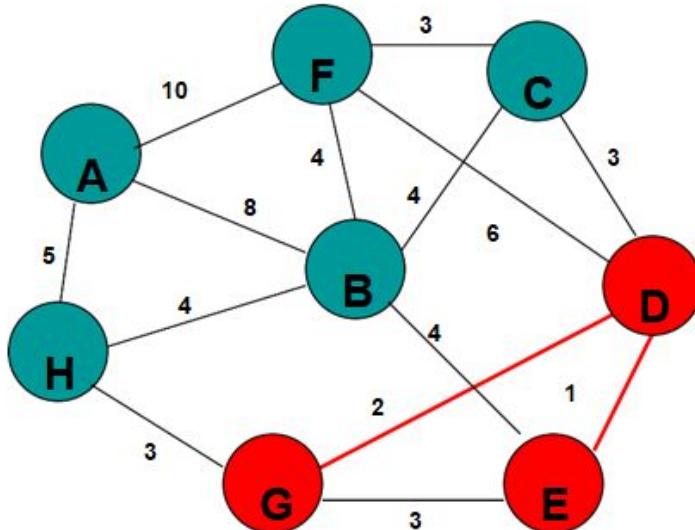


<i>edge</i>	$d_v$	
(D,E)	1	✓
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

<i>edge</i>	$d_v$	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle

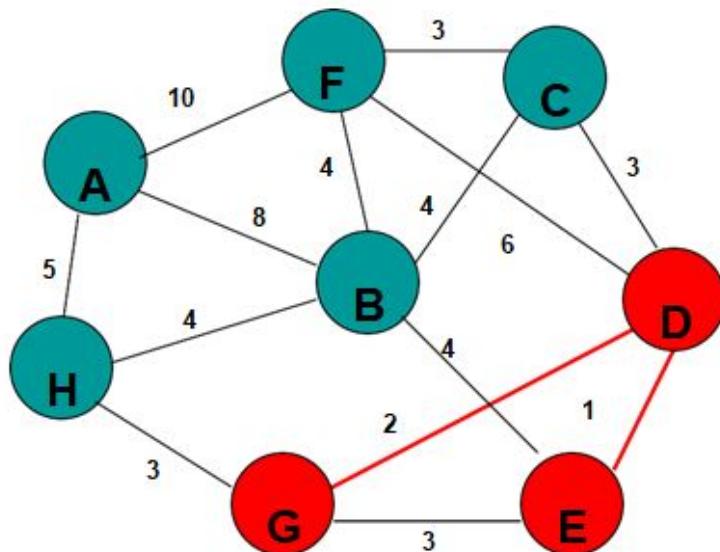


<i>edge</i>	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

<i>edge</i>	$d_v$	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle



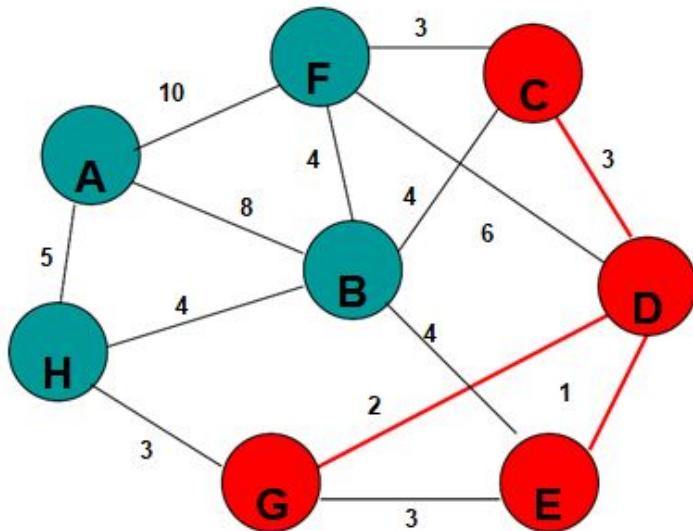
<i>edge</i>	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

<i>edge</i>	$d_v$	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Accepting edge (E,G) would create a cycle.

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle

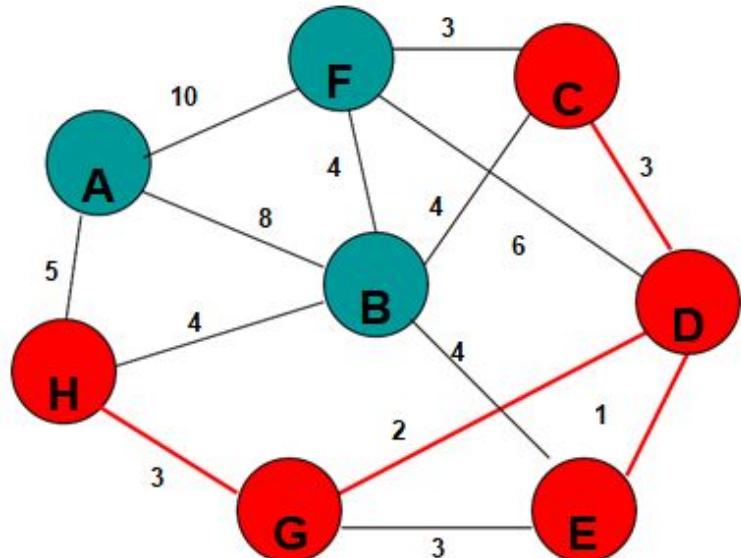


edge	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	$d_v$	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle

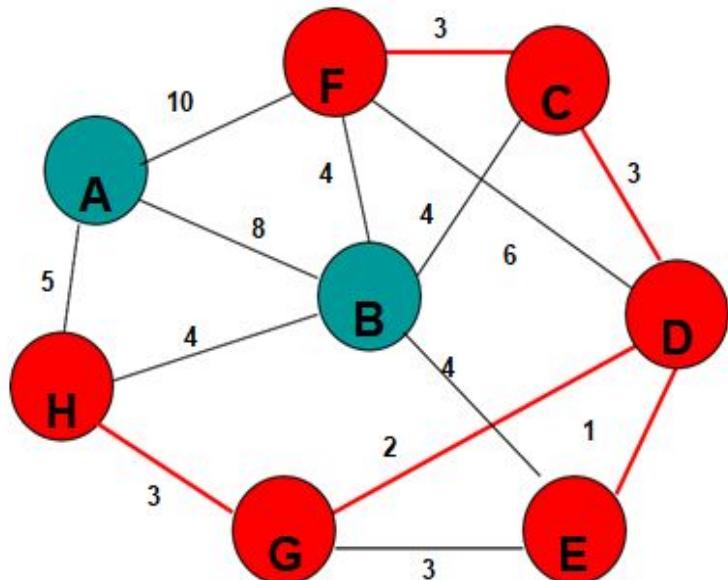


edge	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	
(B,C)	4	

edge	$d_v$	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle

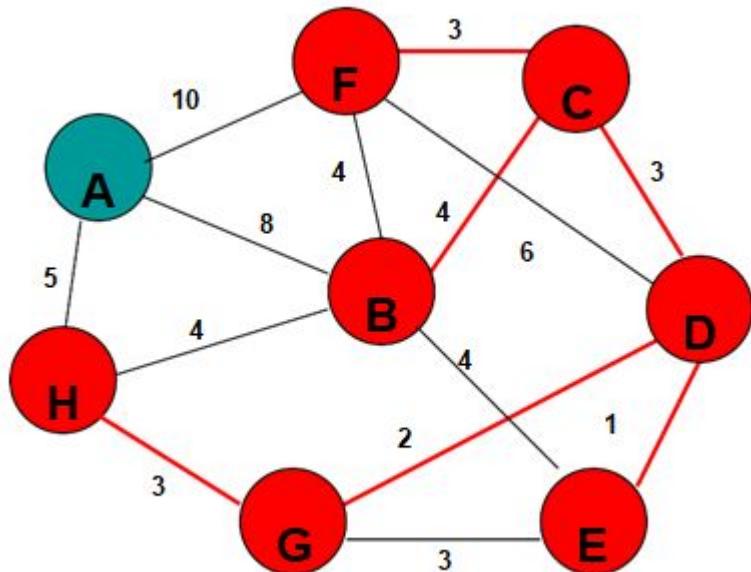


<i>edge</i>	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	

<i>edge</i>	$d_v$	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle

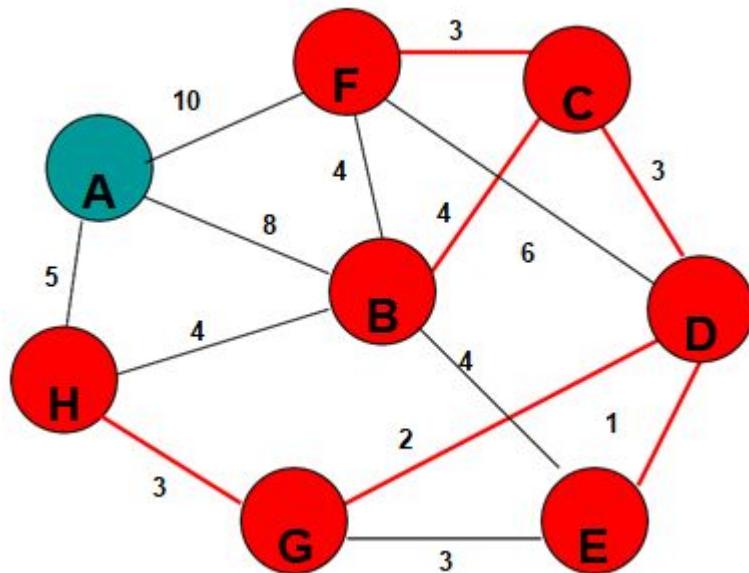


<i>edge</i>	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

<i>edge</i>	$d_v$	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle



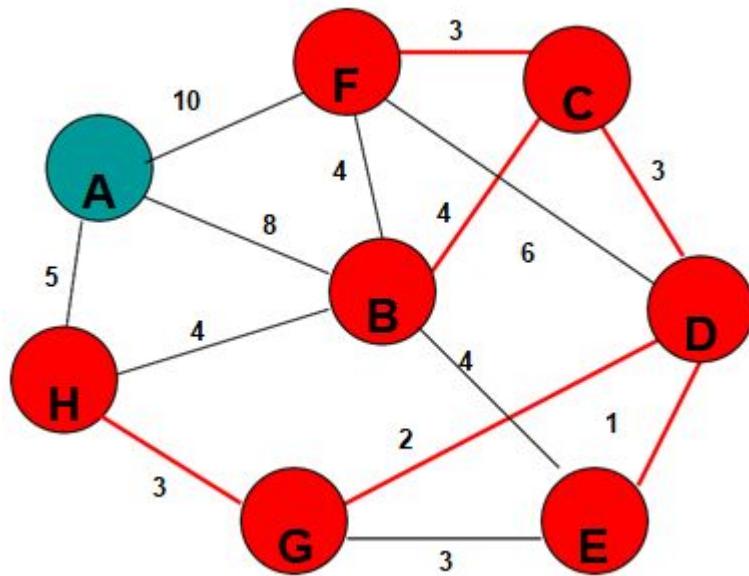
edge	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

edge	$d_v$	
(B,E)	4	✗
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Accepting edge (B,E) would create a cycle.

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle



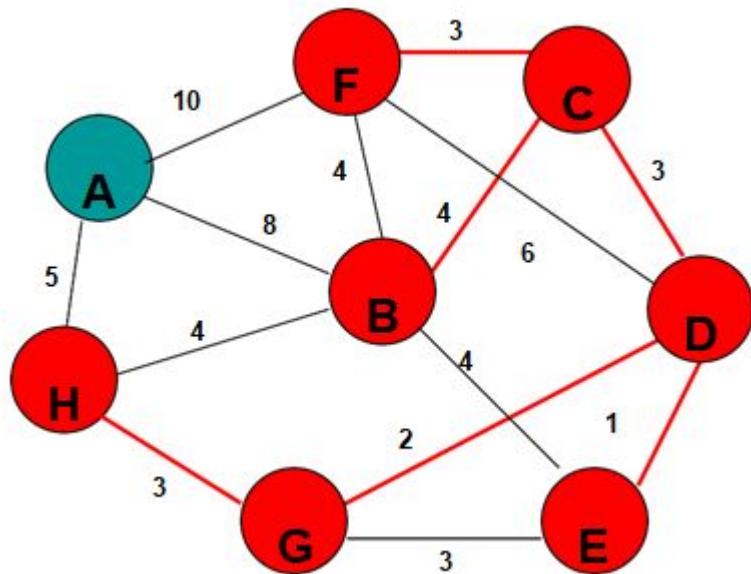
edge	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

edge	$d_v$	
(B,E)	4	✗
(B,F)	4	✗
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Accepting edge (B,F) would create a cycle.

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle



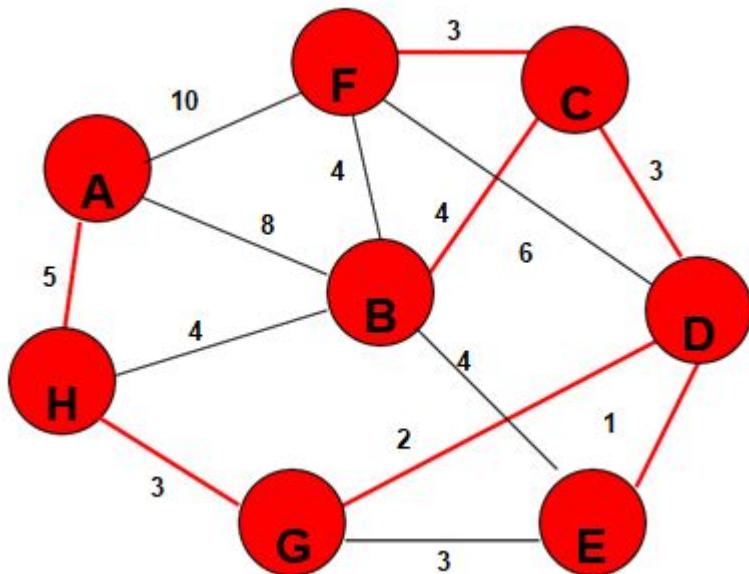
edge	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

edge	$d_v$	
(B,E)	4	✗
(B,F)	4	✗
(B,H)	4	✗
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Accepting edge (B,H) would create a cycle.

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle

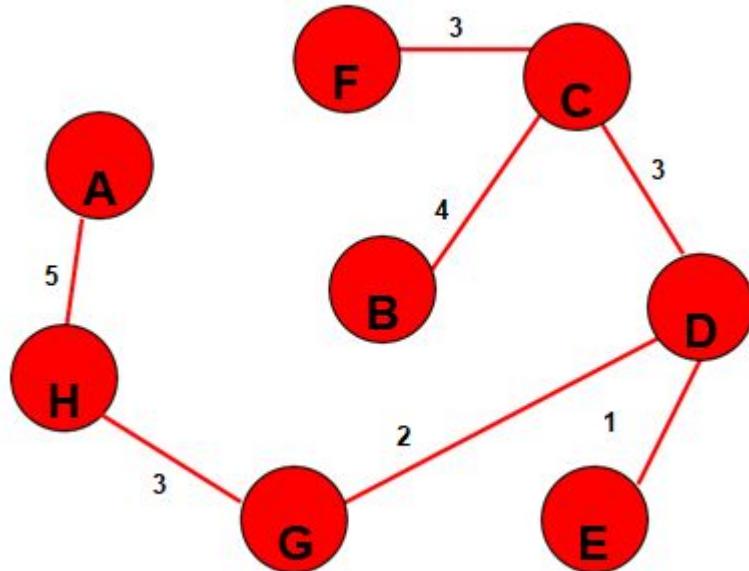


edge	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

edge	$d_v$	
(B,E)	4	✗
(B,F)	4	✗
(B,H)	4	✗
(A,H)	5	✓
(D,F)	6	
(A,B)	8	
(A,F)	10	

# Kruskal's algorithm: Example

Select next  $|V|-1$  edges which do not generate a cycle



<i>edge</i>	$d_v$	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

<i>edge</i>	$d_v$	
(B,E)	4	✗
(B,F)	4	✗
(B,H)	4	✗
(A,H)	5	✓
(D,F)	6	
(A,B)	8	
(A,F)	10	

} not considered

Done

Total Cost = 21

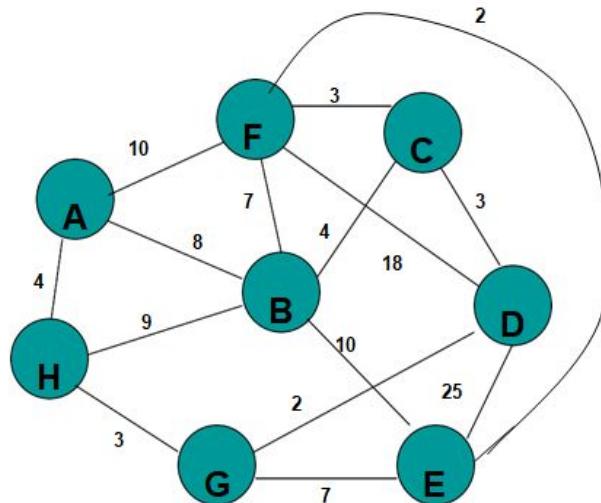
# Table of Contents

1. Shortest path algorithm (Dijkstra's Algorithm)
2. Minimum Spanning Tree (Kruskal's Algorithm)
3. Minimum Spanning Tree (Prim's Algorithm)
4. All Pair Shortest Path (Floyd Warshall's Algorithm)

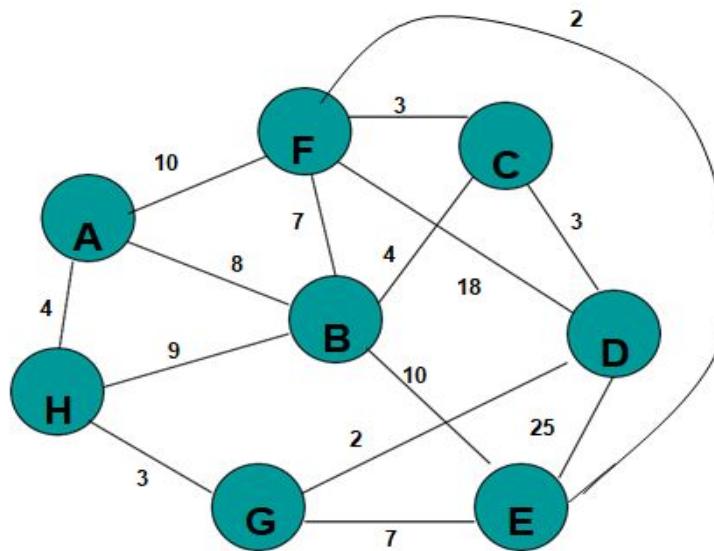
# Minimum Spanning Tree

Prim's algorithm:

- Work with nodes (instead of edges)
- Two steps
  - Select node with minimum distance
  - Update distances of adjacent, unselected nodes
- Walk through:



# Prim's algorithm: Example



$K$ : whether in the tree

$d_v$ : distance to the tree

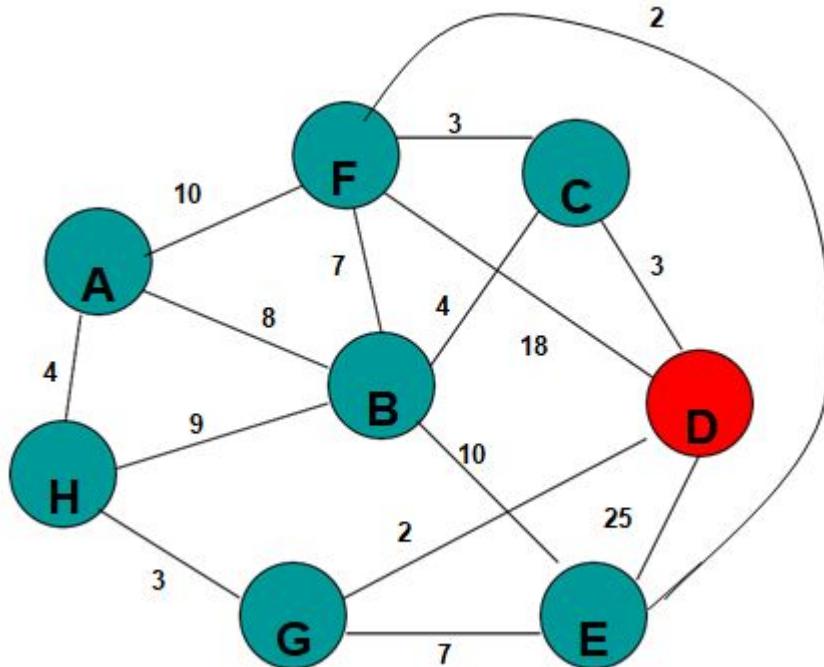
$p_v$ : closest node that is in the tree

Initialize array

	$K$	$d_v$	$p_v$
A	F	$\infty$	-
B	F	$\infty$	-
C	F	$\infty$	-
D	F	$\infty$	-
E	F	$\infty$	-
F	F	$\infty$	-
G	F	$\infty$	-
H	F	$\infty$	-

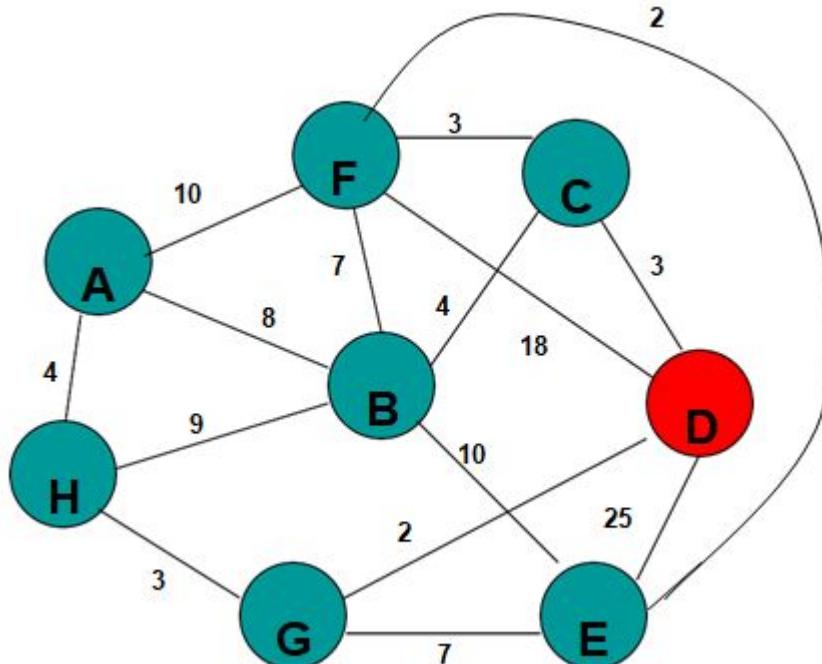
# Prim's algorithm: Example

Start with any node, say D



	$K$	$d_v$	$p_v$
A			
B			
C			
D	T	0	-
E			
F			
G			
H			

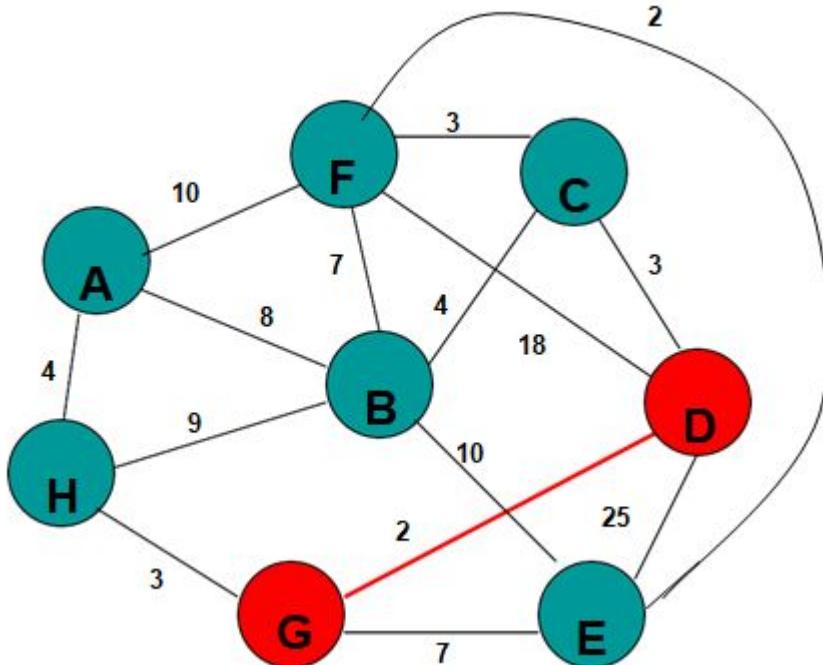
## Prim's algorithm: Example



Update distances of adjacent, unselected nodes

	$K$	$d_v$	$p_v$
A			
B			
C		3	D
D	T	0	-
E		25	D
F		18	D
G		2	D
H			

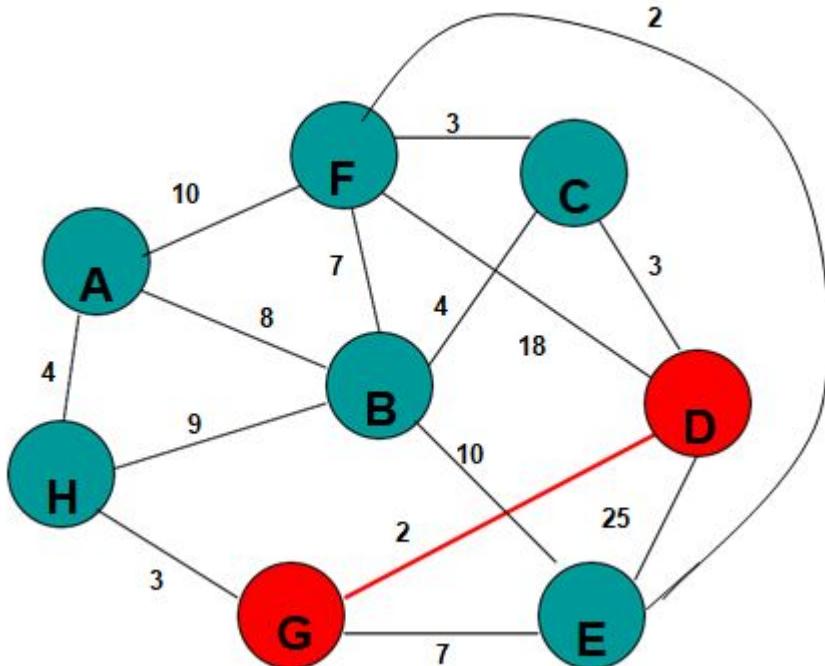
## Prim's algorithm: Example



Select node with  
minimum distance

	$K$	$d_v$	$p_v$
A			
B			
C		3	D
D	T	0	-
E		25	D
F		18	D
G	T	2	D
H			

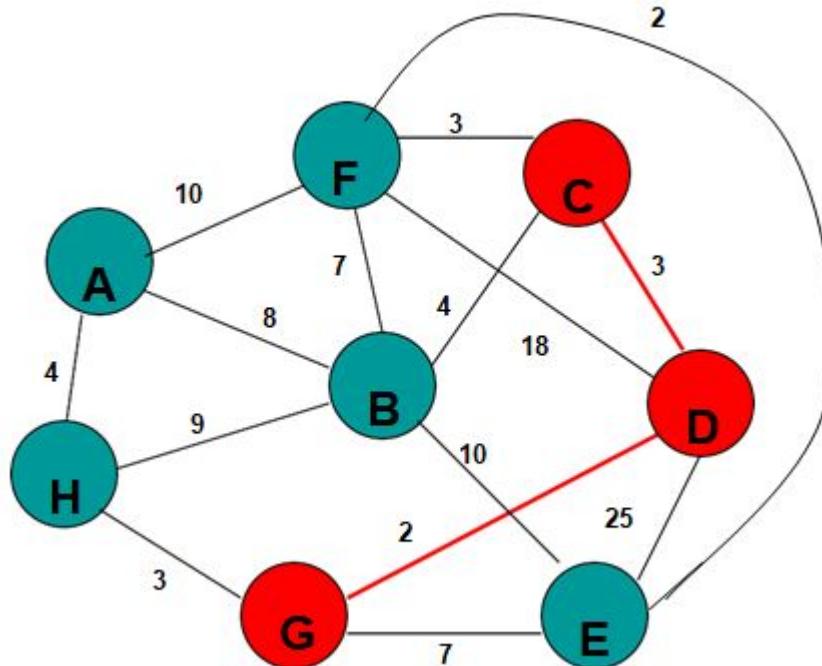
# Prim's algorithm: Example



Update distances of adjacent, unselected nodes

	$K$	$d_v$	$p_v$
A			
B			
C		3	D
D	T	0	-
E		7	G
F		18	D
G	T	2	D
H		3	G

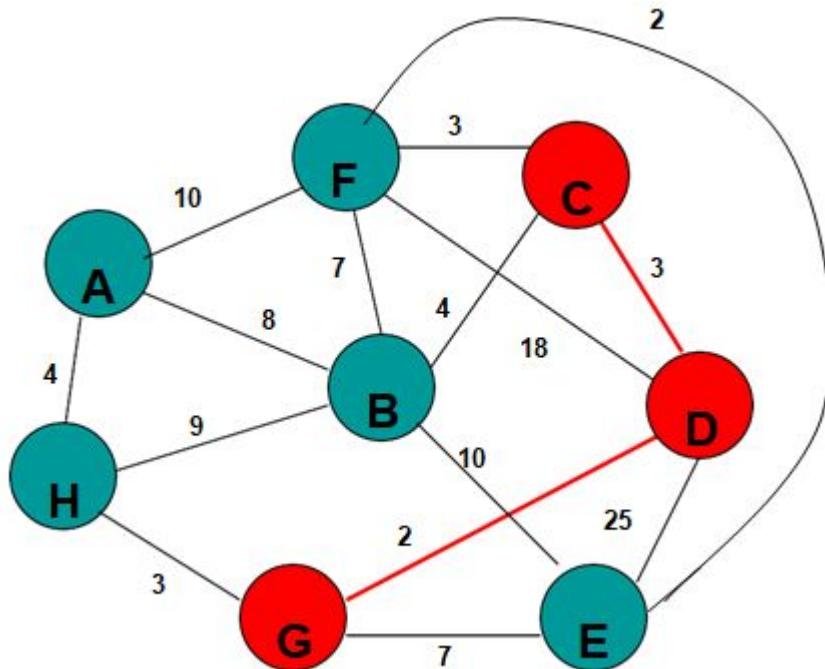
## Prim's algorithm: Example



Select node with minimum distance

	$K$	$d_v$	$p_v$
A			
B			
C	T	3	D
D	T	0	-
E		7	G
F		18	D
G	T	2	D
H		3	G

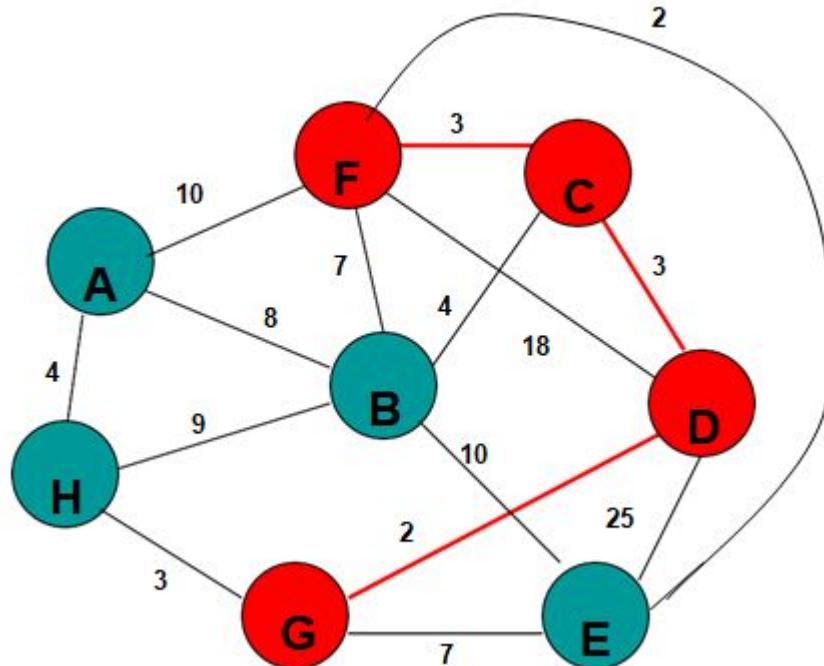
## Prim's algorithm: Example



Update distances of adjacent, unselected nodes

	$K$	$d_v$	$p_v$
A			
B		4	C
C	T	3	D
D	T	0	-
E		7	G
F		3	C
G	T	2	D
H		3	G

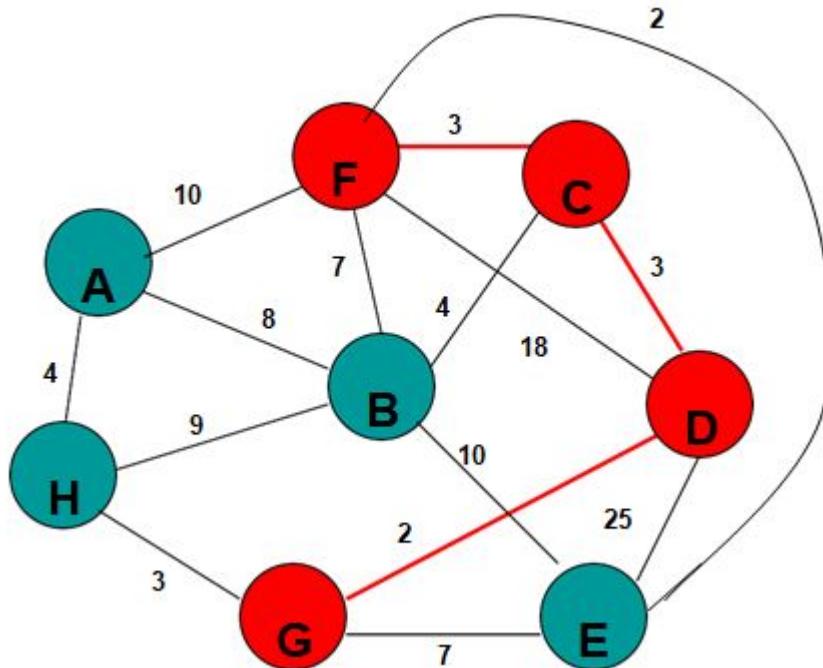
# Prim's algorithm: Example



Select node with  
minimum distance

	$K$	$d_v$	$p_v$
A			
B		4	C
C	T	3	D
D	T	0	-
E		7	G
F	T	3	C
G	T	2	D
H		3	G

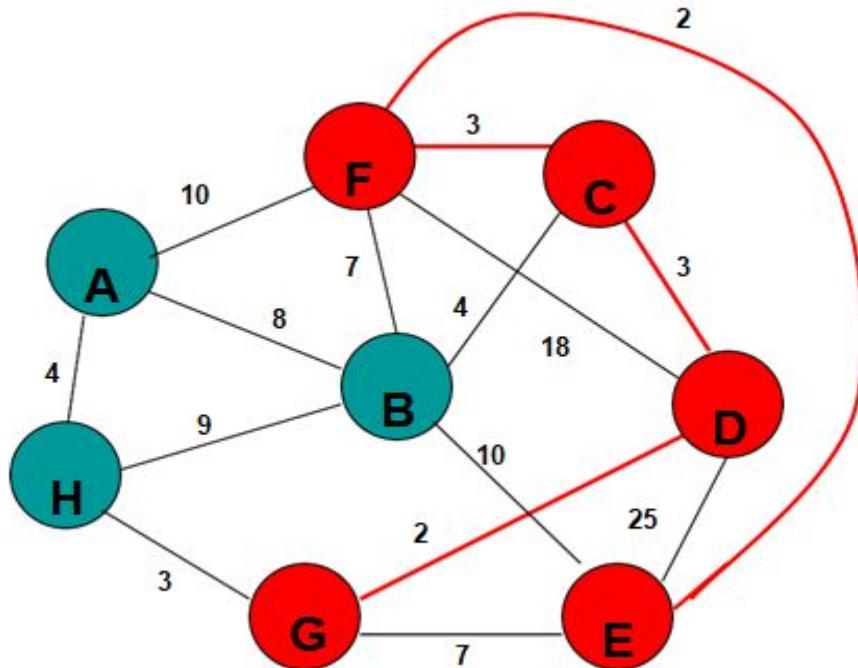
# Prim's algorithm: Example



Update distances of  
adjacent, unselected nodes

	$K$	$d_v$	$p_v$
A		10	F
B		4	C
C	T	3	D
D	T	0	-
E		2	F
F	T	3	C
G	T	2	D
H		3	G

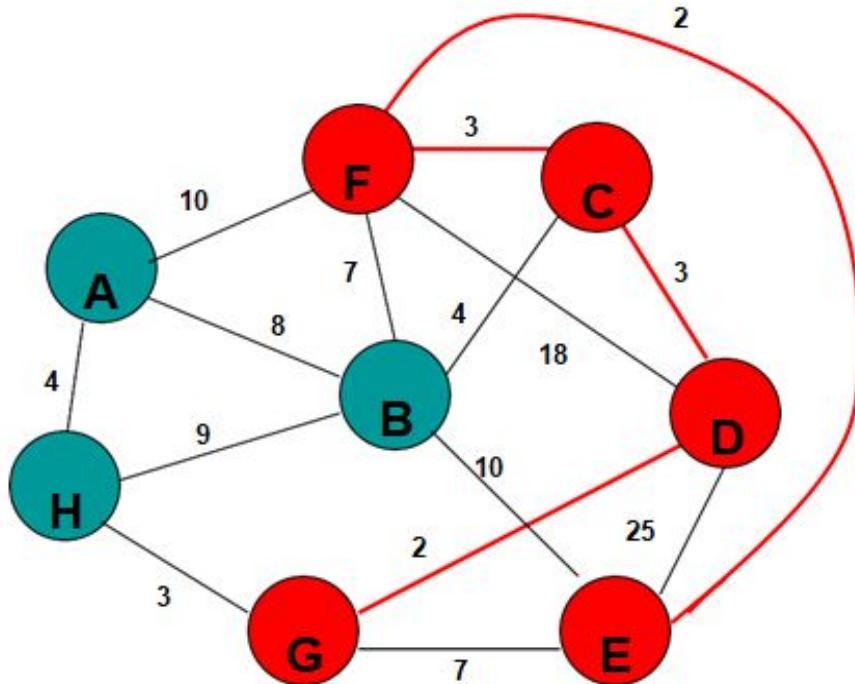
## Prim's algorithm: Example



Select node with  
minimum distance

	$K$	$d_v$	$p_v$
A		10	F
B		4	C
C	T	3	D
D	T	0	-
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

# Prim's algorithm: Example

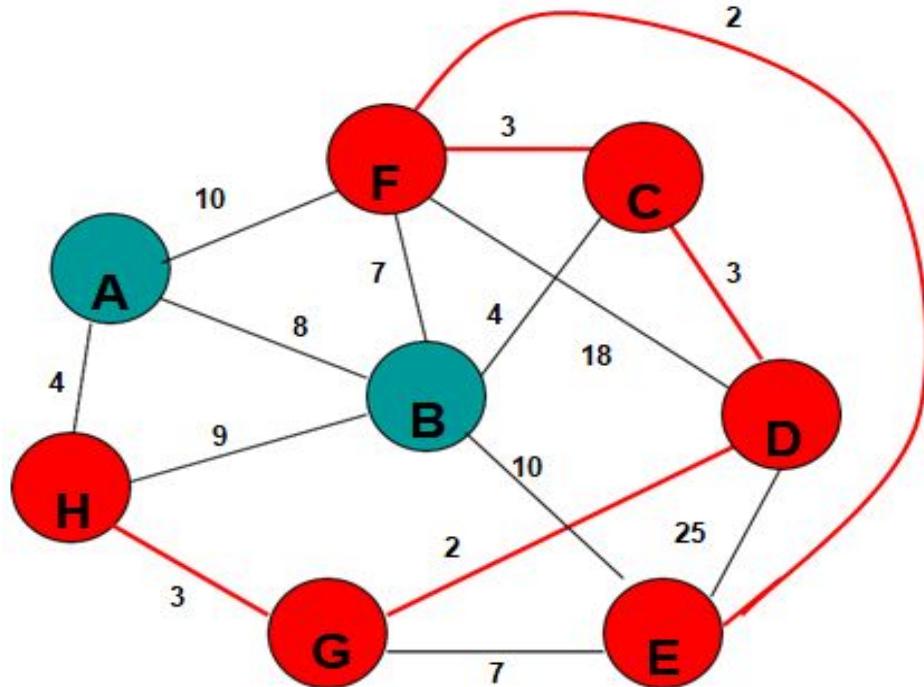


Update distances of adjacent, unselected nodes

	$K$	$d_v$	$p_v$
A		10	F
B		4	C
C	T	3	D
D	T	0	-
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

Table entries  
unchanged

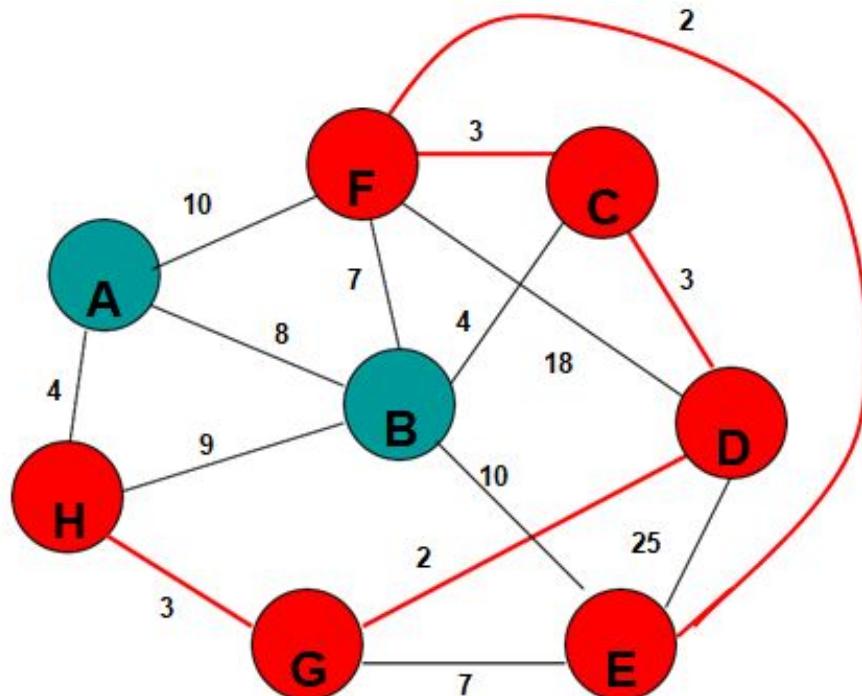
## Prim's algorithm: Example



Select node with  
minimum distance

	$K$	$d_v$	$p_v$
A		10	F
B		4	C
C	T	3	D
D	T	0	-
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

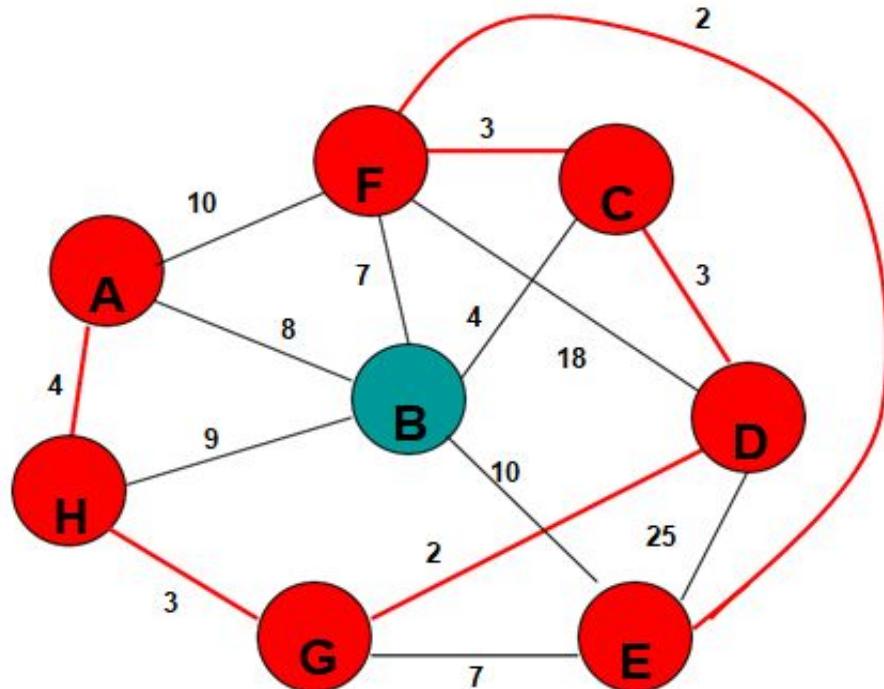
## Prim's algorithm: Example



Update distances of adjacent, unselected nodes

	$K$	$d_v$	$p_v$
A		4	H
B		4	C
C	T	3	D
D	T	0	-
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

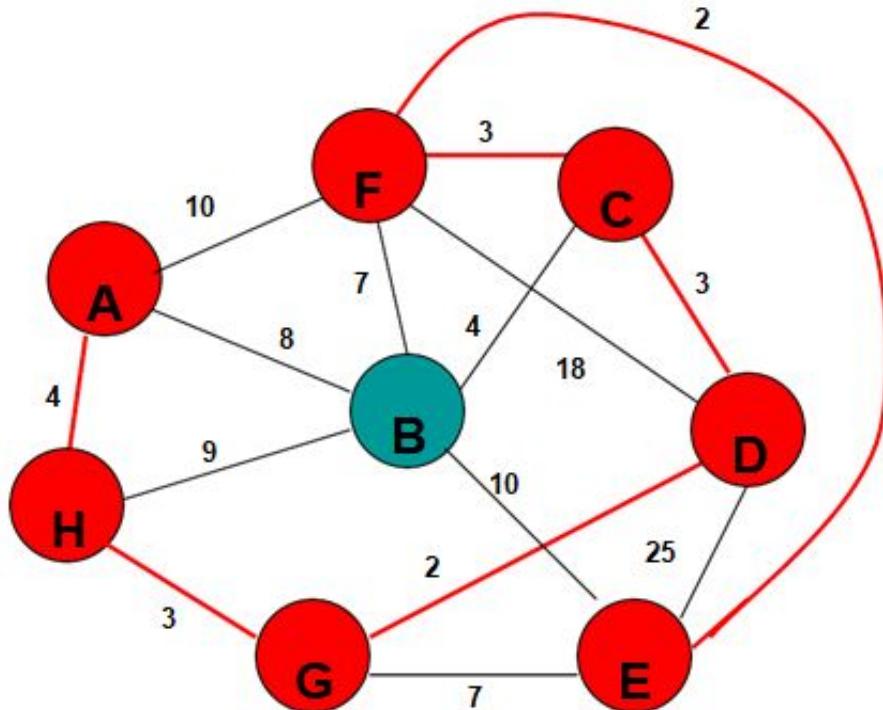
## Prim's algorithm: Example



Select node with  
minimum distance

	$K$	$d_v$	$p_v$
A	T	4	H
B		4	C
C	T	3	D
D	T	0	-
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

# Prim's algorithm: Example

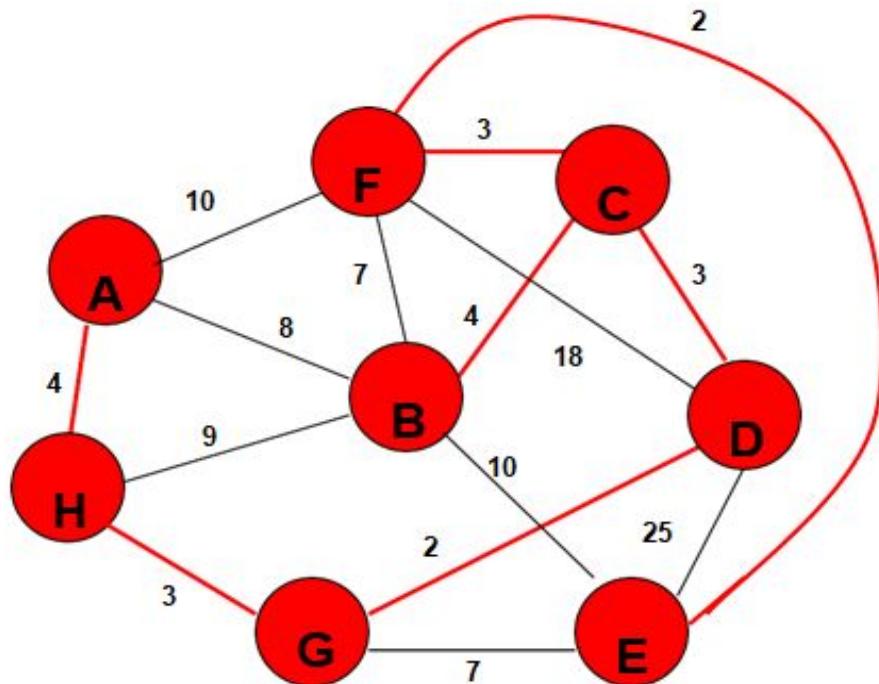


Update distances of adjacent, unselected nodes

	$K$	$d_v$	$p_v$
A	T	4	H
B		4	C
C	T	3	D
D	T	0	-
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Table entries  
unchanged

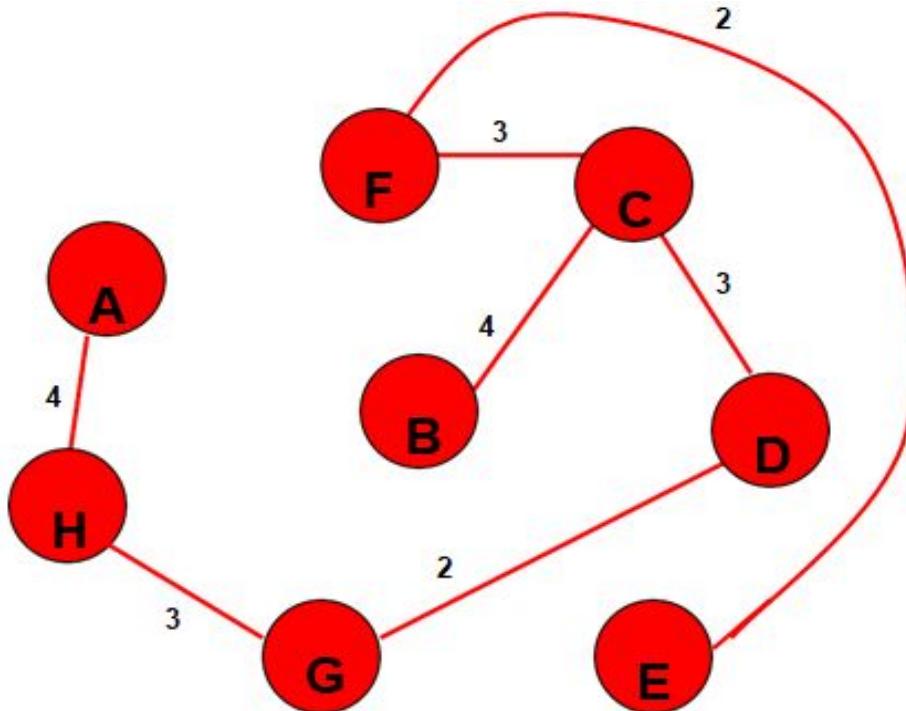
## Prim's algorithm: Example



Select node with minimum distance

	$K$	$d_v$	$p_v$
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	-
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

## Prim's algorithm: Example



Cost of Minimum  
Spanning Tree = **21**

	$K$	$d_v$	$p_v$
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	-
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

**Done**

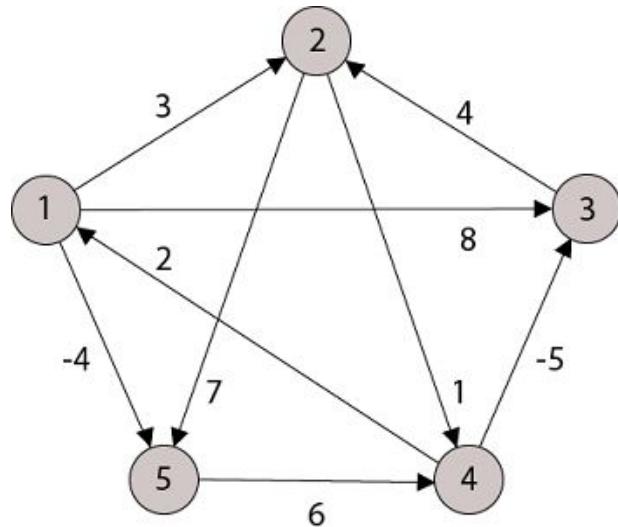
# Table of Contents

1. Shortest path algorithm (Dijkstra's Algorithm)
2. Minimum Spanning Tree (Kruskal's Algorithm)
3. Minimum Spanning Tree (Prim's Algorithm)
4. All Pair Shortest Path (Floyd Warshall's Algorithm)

## Floyd Warshall's Algorithm

- Floyd Warshall Algorithm is a famous algorithm.
- It is used to solve All Pairs Shortest Path Problem.
- It computes the **shortest path between every pair of vertices** of the given graph.
- Floyd Warshall Algorithm is an example of dynamic programming approach.
- It can also handle the negative weight.

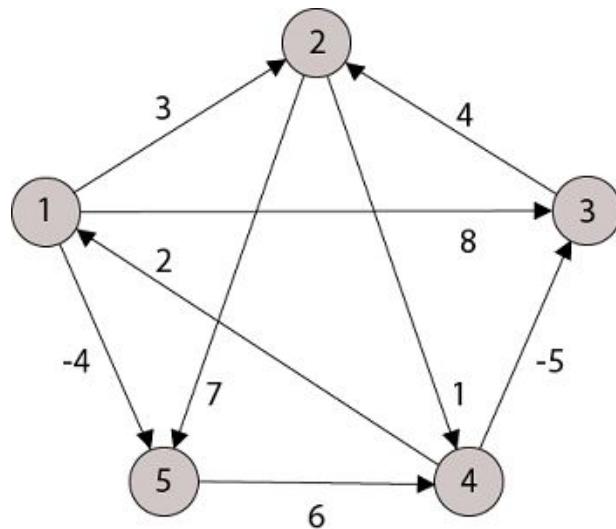
## Floyd Warshall's Algorithm: Example



$$D(0) =$$

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>1</b>	0	3	8	$\infty$	-4
<b>2</b>	$\infty$	0	$\infty$	1	7
<b>3</b>	$\infty$	4	0	$\infty$	$\infty$
<b>4</b>	2	$\infty$	-5	0	$\infty$
<b>5</b>	$\infty$	$\infty$	$\infty$	6	0

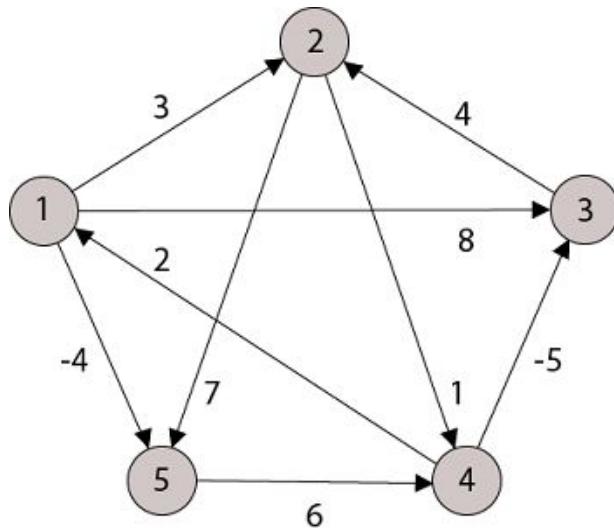
# Floyd Warshall's Algorithm: Example



$D(1) =$

	1	2	3	4	5
1	0	3	8	$\infty$	-4
2	$\infty$	0	$\infty$	1	7
3	$\infty$	4	0	$\infty$	$\infty$
4	2	5	-5	0	-2
5	$\infty$	$\infty$	$\infty$	6	0

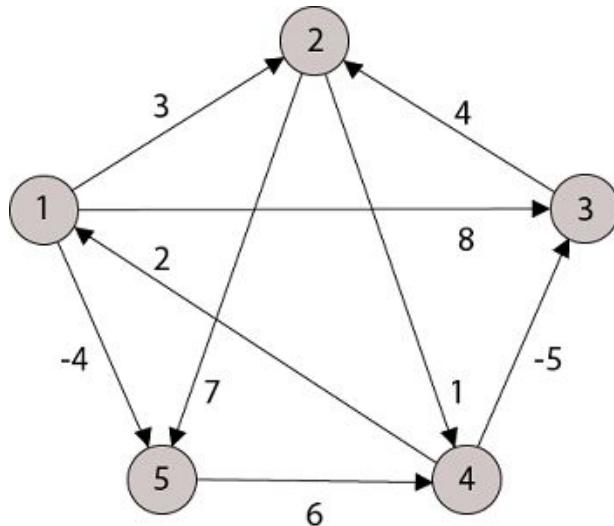
## Floyd Warshall's Algorithm: Example



$D(2) =$

	1	2	3	4	5
1	0	3	8	4	-4
2	$\infty$	0	$\infty$	1	7
3	$\infty$	4	0	5	11
4	2	5	-5	0	-2
5	$\infty$	$\infty$	$\infty$	6	0

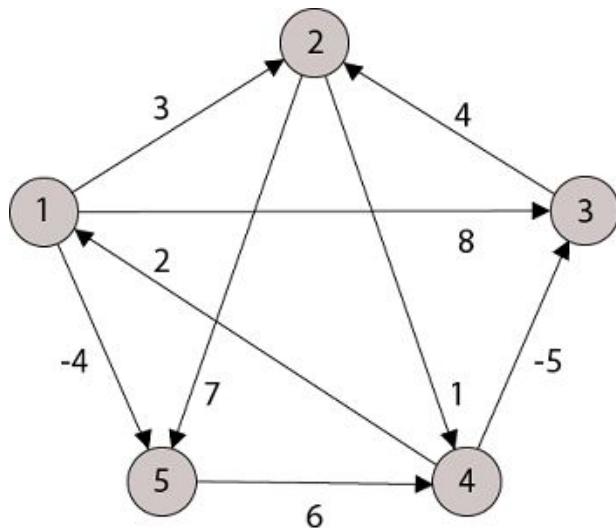
# Floyd Warshall's Algorithm: Example



$D(3)=$

	1	2	3	4	5
1	0	3	8	4	-4
2	$\infty$	0	$\infty$	1	7
3	$\infty$	4	0	5	11
4	2	-1	-5	0	-2
5	$\infty$	$\infty$	$\infty$	6	0

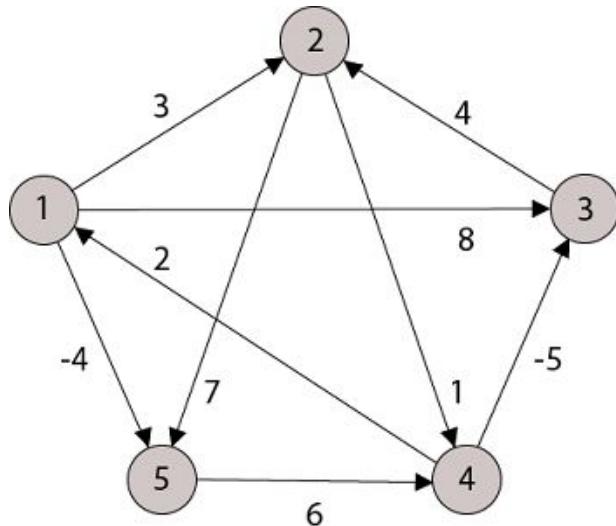
## Floyd Warshall's Algorithm: Example



$D(4) =$

	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

# Floyd Warshall's Algorithm: Example



$D(5) =$

	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

